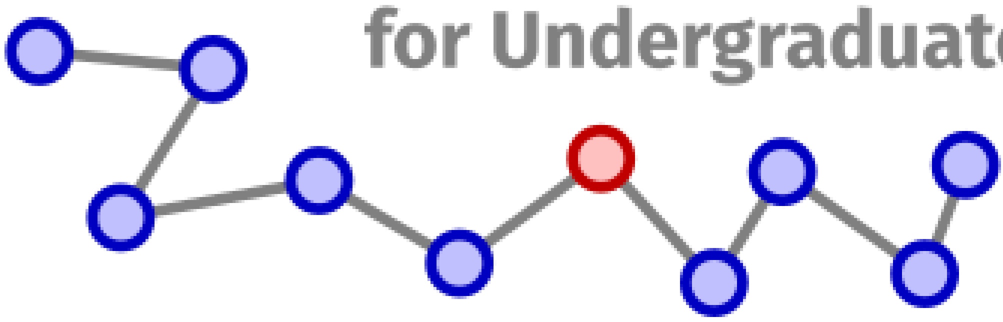
Data Science and Artificial Intelligence for Undergraduates

Jens Flemming

May 30, 2024

CONTENTS

Data Science and Artificial Intelligence for Undergraduates



This book covers a wide range of topics in data science and artificial intelligence. It's an attempt to provide self-contained learning material for first-year students in data science related courses. Most, not all, of the material is tought in the undergradute course on data science[1] at Zwickau University of Applied Sciences[2].

Starting teaching data science in 2019 the author[3] faced the problem that there seems to be no text book covering math, computer science, statistical data science, artificial intelligence and related topics in a well structured, accessible, thorough way. Basic Python[4] programming should be covered as well as state of the art deep reinforcement learning for controlling autonomous robots. All this with hands-on experience for students, interesting real-world data sets, and sufficiently rich theoretical background.

Classical paper books or PDF ebooks do not suit the needs for this project. Working with data requires lots of source code, interactive visualizations, data listings, and easy to follow pointers to online resources. Jupyter Book[5] is an awesome software tool for publishing book-like interactive content. For the author writing this book is also a journey of discovery to the possible future of publishing. Having authored two paper books the author knows the tight limits of paper books and publishing companies. The greater his enthusiasm is for the freedom in writing and publishing provided by Jupyter Book and its community, The Executable Books Project[6].

The author expresses its gratitude towards all the more or less anonymous people developing the wonderful open source tools used in this book and for writing the book. There are too many tools to list them here. The author also thanks his students and colleagues at Zwickau University, especially Hendrik Weiß, who constantly find typos and make suggestions for improving the book.

Jens Flemming[7], Zwickau, December 2022

[1] https://datascience.fh-zwickau.de
[2] https://www.fh-zwickau.de
[3] https://www.fh-zwickau.de/~jef19jdw
[4] https://www.python.org
[5] https://jupyterbook.org
[6] https://executablebooks.org
[7] https://www.fh-zwickau.de/~jef19jdw

Part I

How to Use This Book

CHAPTER

ONE

AN EXECUTABLE BOOK

This book provides several online features for executing presented Python code and for contributing content. This chapter contains all information you need for using these features.

- *Read Online or Offline* (page 5)
- *Manipulate and Execute Code* (page 6)
- *Contribute* (page 9)

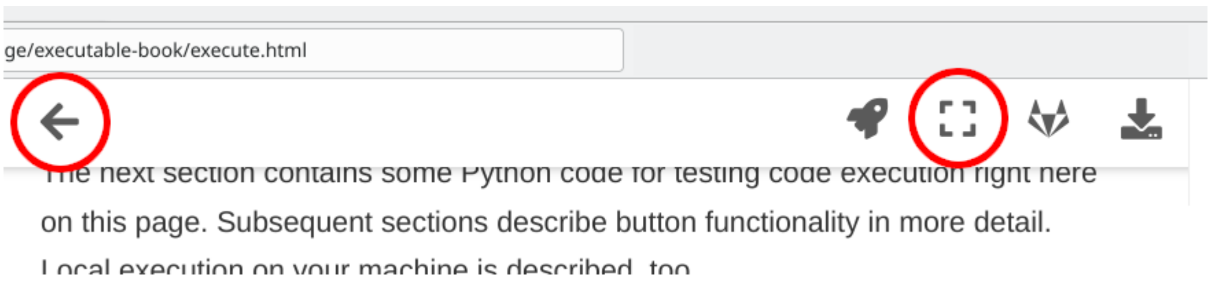
1.1 Read Online or Offline

This book has been created with Jupyter Book[8]. It comes in different formats and variants. The reader may choose according to her or his preferences. The reader may even switch between different variants at will.

1.1.1 Online in a Webbrowser

The intended medium for reading this book is a website in a webbrowser, that is, the HTML rendering of the book[9]. There you have full functionality including interactive features and live code editing and execution, see *Manipulate and Execute Code* (page 6) for details.

The HTML rendering comes with an optional fullscreen mode (button in the upper right corner). You may also hide the table of contents sidebar on small screens (button in the upper left corner).


ge/executable-book/execute.html

The next section contains some Python code for testing code execution right here on this page. Subsequent sections describe button functionality in more detail. Local execution on your machine is described, too.

Fig. 1.1: Fullscreen button and sidebar button allow to adjust the book's layout to your screen.

[8] https://jupyterbook.org
[9] https://www.fh-zwickau.de/~jef19jdw/data-science-ai

1.1.2 Offline in a Webbrowser

You may download the whole HTML rendering as ZIP archive. After extracting the archive open the file `index.html` in a webbrowser. All features will work like in the online version, but some content, like externally hosted videos, won't be available without internet connection.

The HTML rendering is a static website, that is, no webserver is needed for reading. Everything works on your local machine.

Download HTML rendering in ZIP file

[10]

1.1.3 Offline PDF ebook

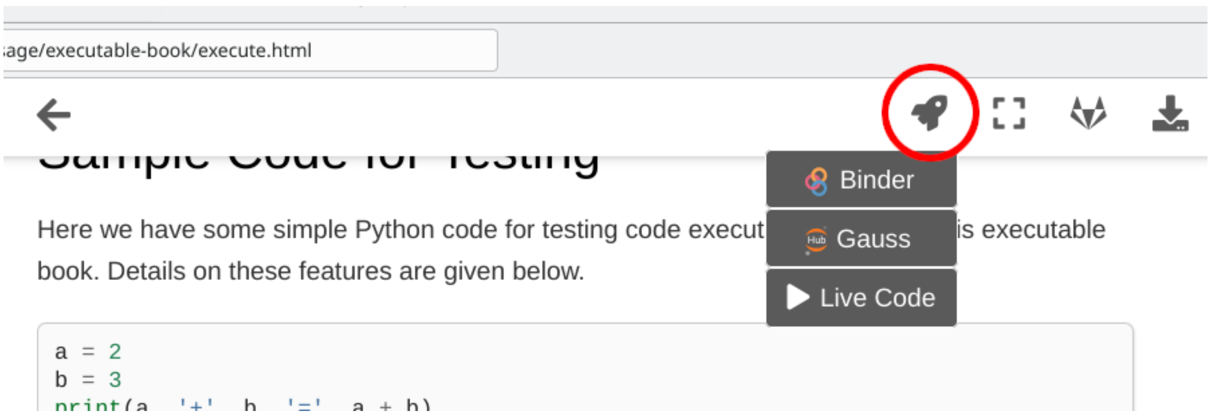
For printing and for friends of higher quality typesetting there is a PDF version of the book. Of course, the PDF version lacks interactive features like in-place code editing and execution.

Download PDF ebook

[11]

1.2 Manipulate and Execute Code

Python code in this book can be executed in different ways without copying the code manually. The HTML rendering's upper right corner shows a rocket symbol. The rocket button provides several options for executing a page's code.



age/executable-book/execute.html

Here we have some simple Python code for testing code execut is executable
book. Details on these features are given below.

```
a = 2
b = 3
```

Fig. 1.2: Hovering over the rocket symbol provides several options for code execution.

The next section contains some Python code for testing code execution right here on this page. Subsequent sections describe button functionality in more detail. Local code execution on your machine is described, too.

Attention: All code execution features but Live Code use the book's Jupyter[12] rendering. For technical reasons the Jupyter rendering lacks some figures and text formatting may be incorrect. For reading without a need for code execution stay with the HTML or PDF renderings.

[10] https://www.fh-zwickau.de/~jef19jdw/data-science-ai/data-science-ai.zip
[11] https://www.fh-zwickau.de/~jef19jdw/data-science-ai/data-science-ai.pdf
[12] https://jupyter.org

1.2.1 Sample Code for Testing

Here we have some simple Python code for testing code execution features of this executable book. Details on these features are given below.

```
a = 2
b = 6
print(a, '+', b, '=', a + b)
```

```
2 + 6 = 8
```

1.2.2 Launch on Binder

The Binder launch button opens a JupyterLab[13] session on mybinder.org[14]. There you find the book's Jupyter rendering. The Jupyter rendering is a collection of Jupyter Notebooks (files with `ipynb` extension). The Binder launch button opens your current page's Jupyter rendering, but all other pages are available, too, in one and the same Binder session.

Starting repository: https://gitlab.hrz.tu-chemnitz.de/jef19jdw--fh-zwickau.de/data-science-ai/main

New to Binder? Check out the Binder Documentation for more information

Fig. 1.3: Binder startup requires cloning the book's Git repository if something has changed since last Binder usage.

Starting the Binder session may take some seconds. Keep in mind that mybinder.org[15] is a free service provided by volunteers and supported by donators. Don't overuse it to keep it free and available to everybody. Don't run complex computations like neural network training on Binder.

The JupyterLab session on Binder allows for code editing and repeated execution. You may also save your files there, but they will be lost as soon as you end the session. Don't forget to download modified files to your local machine before you leave.

[13] https://jupyter.org
[14] https://mybinder.org
[15] https://mybinder.org

1.2.3 Launch on Gauss

Gauss[16] is a GPU server at Zwickau University of Applied Sciences **only available in the university's intranet**. Students with access to Gauss should use Gauss instead of Binder. The Gauss launch button runs the book's Jupyter rendering in JupyterLab on Gauss very similar to Binder.

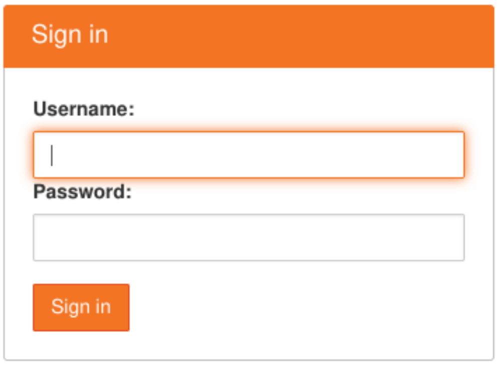
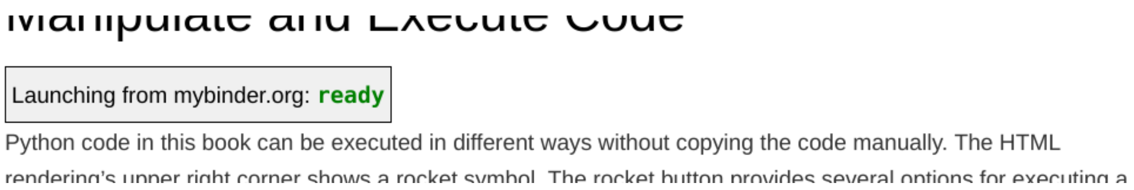


Fig. 1.4: Gauss asks for username and password before launching a book's page in JupyterLab.

A click on the Gauss launch button copies the whole GitLab repository[17] of the book to the user's personal directory on Gauss. Thus, modifications to code and other files are saved to the user's directory, too, and are persistent. Repeated clicks on the Gauss launch button do not overwrite a user's modifications, but may update files untouched by the user but modified in the GitLab repository. Thus, the user's version will always be up-to-date while preserving the user's modifications as far as possible. For details on the merge process run when clicking the Gauss launch button see Automatic Merging Behavior[18] in nbgitpuller's documentation.

1.2.4 Live Code

The Live Code button makes code cells editable and executable on-the-spot using Thebe[19]. Clicking the Live Code button starts a Python kernel on mybinder.org[20] and connects the book's HTML rendering to that kernel. Progress and success of the startup process are shown below the page's heading.


Launching from mybinder.org: ready

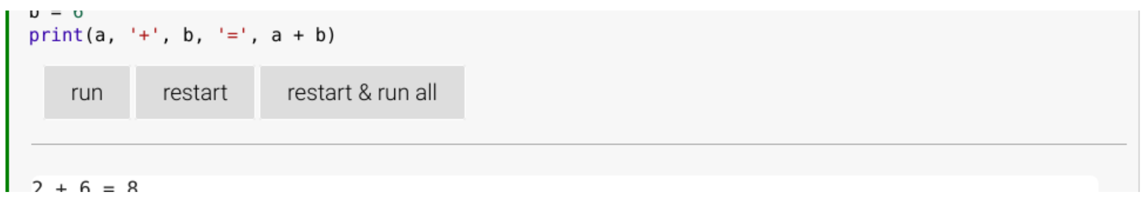
Python code in this book can be executed in different ways without copying the code manually. The HTML

Fig. 1.5: A box with progress information appears after clicking the Live Code button.

Code cells on the page change their appearance. Outputs now belong to the cell, some buttons appear, and the code becomes editable.

Cells are not run immediately after clicking the Live Code button. Clicking the 'run' button executes the cell. Alternatively, one may run all cells on a page by clicking the 'restart & run all' button.

[16] https://gauss.fh-zwickau.de
[17] https://gitlab.hrz.tu-chemnitz.de/jef19jdw--fh-zwickau.de/data-science-ai
[18] https://jupyterhub.github.io/nbgitpuller/topic/automatic-merging.html
[19] https://github.com/executablebooks/thebe
[20] https://mybinder.org



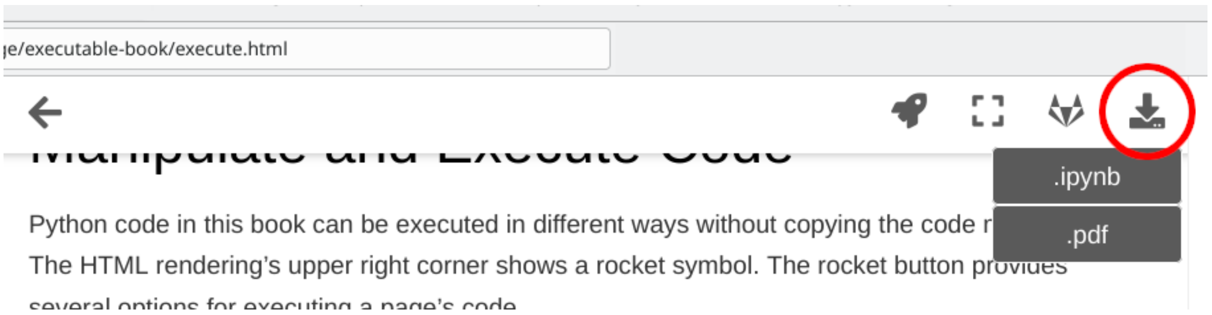
```
print(a, '+', b, '=', a + b)
```

```
2 + 6 = 8
```

Fig. 1.6: After lauching Live Code each code cell shows buttons for starting and controlling code execution.

1.2.5 Local Code Execution

To execute the book's Python code on your local machine download a page's Jupyter rendering by clicking the download button in the upper right corner of the HTML rendering or clone the book's Git repository[21] to your machine.



Python code in this book can be executed in different ways without copying the code r
The HTML rendering's upper right corner shows a rocket symbol. The rocket button provides

Fig. 1.7: Hovering the download symbol shows a list of available formats.

On your machine you need JupyterLab[22] or a similar tool from the Jupyter ecosystem to view and modify the `ipynb` files. For install instructions have a look at the *Install Jupyter Locally* (page 905) project.

1.3 Contribute

This book is not static. It will grow over time and existing material will be rearranged, updated, reduced or extended as required by future developments in data science, AI and teaching. In this sence, it's a living or dynamic book. And you, the reader, are invited to contribute to the book.

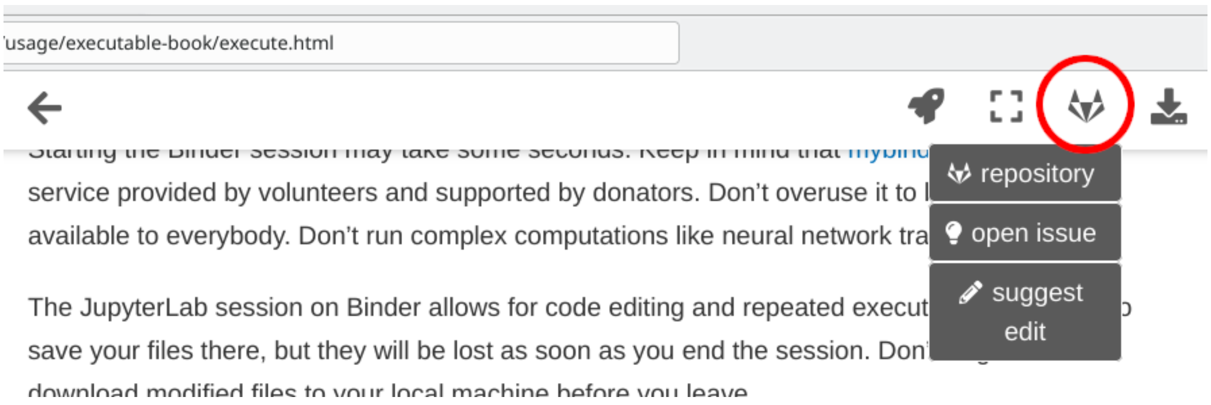
The book's HTML rendering shows a contribution button in the upper right corner offering several options. Those options will be discussed in detail below.

1.3.1 Repository Button

The repository button is a simple link to the book's GitLab repository. There you find all source files needed to render the book. If you are familiar with Git and GitLab the repository button is a good starting point for contributing to the project. If you are not familiar with Git and GitLab, don't worry and use one of the other contribution options.

Important: The book's public Git repository is hosted on a GitLab instance provided by Chemnitz University of Technology[23] for all Saxon universities. Actions requiring a user account are restricted to members of Saxon universities.

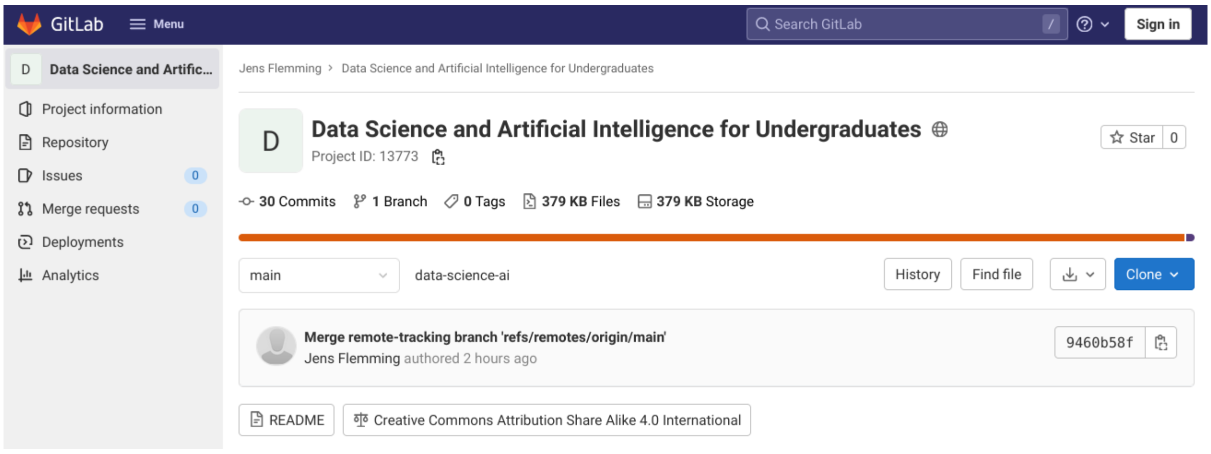
[21] https://gitlab.hrz.tu-chemnitz.de/jef19jdw--fh-zwickau.de/data-science-ai
[22] https://jupyter.org
[23] https://www.tu-chemnitz.de



Starting the Binder session may take some seconds. Keep in mind that mybind
service provided by volunteers and supported by donators. Don't overuse it to l
available to everybody. Don't run complex computations like neural network tra

The JupyterLab session on Binder allows for code editing and repeated execut
save your files there, but they will be lost as soon as you end the session. Don'
download modified files to your local machine before you leave.

Fig. 1.8: Hovering over the contribution button shows all options for contributing to the book.


GitLab
Menu
Search GitLab
Sign in
Data Science and Artific...
Project information
Repository
Issues 0
Merge requests 0
Deployments
Analytics
Jens Flemming › Data Science and Artificial Intelligence for Undergraduates
Star 0
main
data-science-ai
History
Find file
Clone
README
Creative Commons Attribution Share Alike 4.0 International

Data Science and Artificial Intelligence for Undergraduates

Project ID: 13773

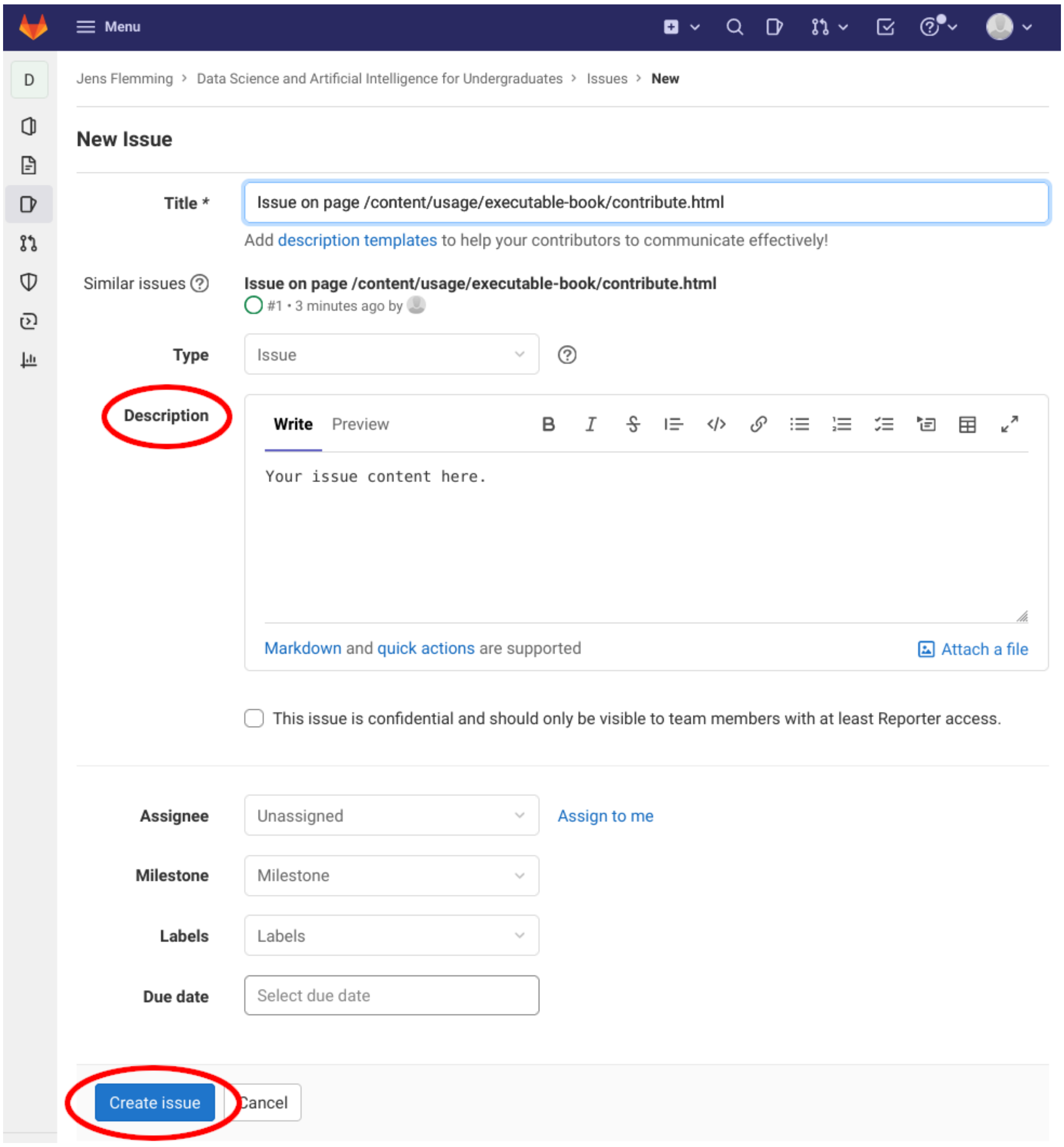
30 Commits 1 Branch 0 Tags 379 KB Files 379 KB Storage

Merge remote-tracking branch 'refs/remotes/origin/main'
Jens Flemming authored 2 hours ago
9460b58f

Fig. 1.9: The repository button leads to a page in GitLab with information about the project, including information about the most recent update.

1.3.2 Open Issue Button

The open issue button allows to ask questions or report bugs, typos and so on. Clicking this button opens a new issue in the book's GitLab repository. Put your question, bug report, what ever in the description field and click the 'Create issue' button.


Menu
D
Jens Flemming > Data Science and Artificial Intelligence for Undergraduates > Issues > New
New Issue

Title * Issue on page /content/usage/executable-book/contribute.html

Add description templates to help your contributors to communicate effectively!

Similar issues Issue on page /content/usage/executable-book/contribute.html
#1 • 3 minutes ago by

Type Issue

Description

Your issue content here.

Markdown and quick actions are supported

This issue is confidential and should only be visible to team members with at least Reporter access.

Assignee Unassigned

Milestone Milestone

Labels Labels

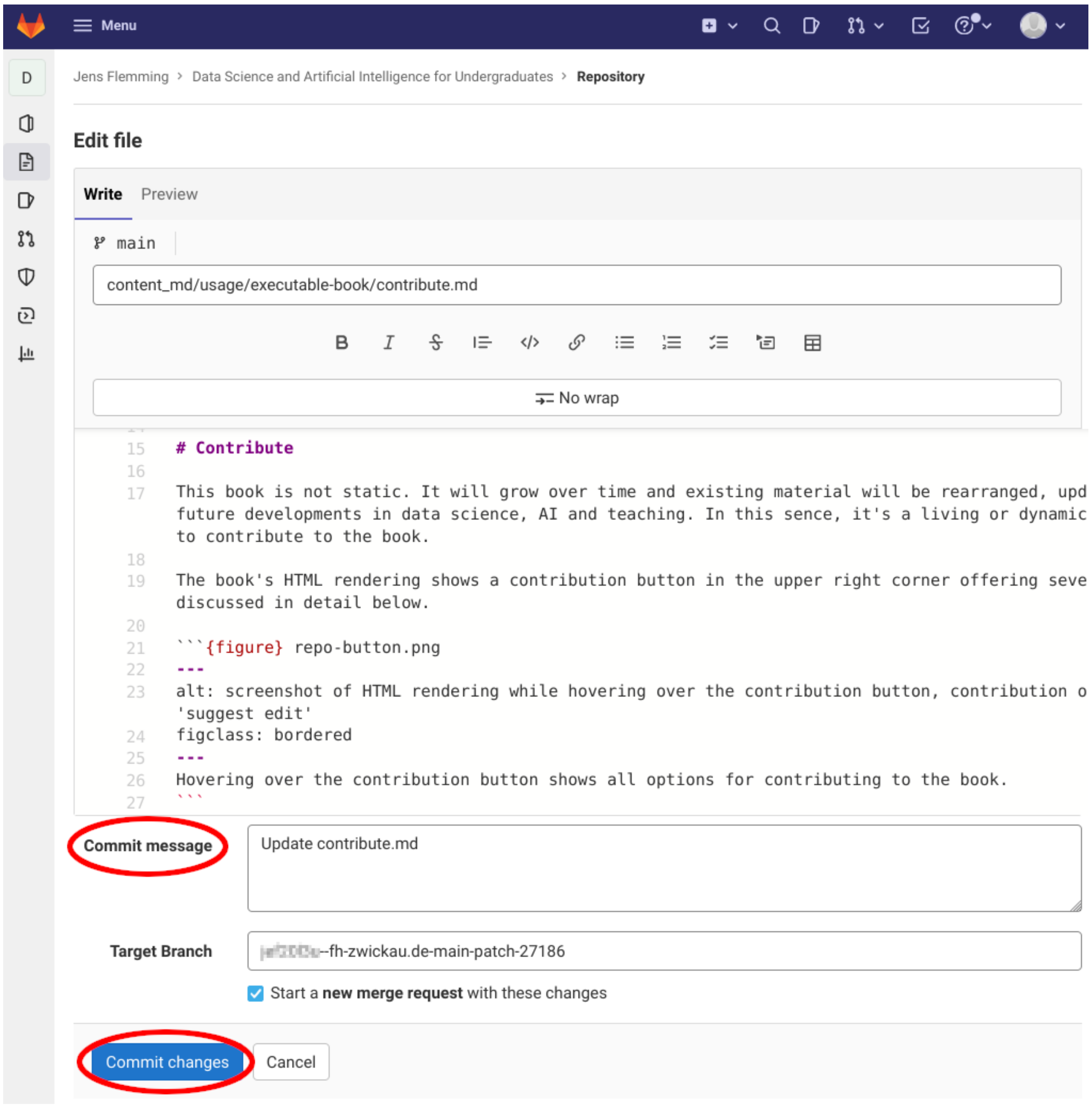
Due date Select due date

Fig. 1.10: To open an issue simply type a description and click on 'Create issue'. The title field is prefilled and should remain untouched.

The author will have a look at the issue as soon as possible and post an answer. Depending on your GitLab account's configuration you will receive email notifications if someone posts a comment on your issue. Each reader may comment on each other reader's issues. So readers may help other readers to solve problems where appropriate.

1.3.3 Suggest Edit Button

If you spot a typo or if you would like to add some explanatory note or an additional code example you should hit the 'suggest edit' button. The button opens the current page for editing in GitLab. The syntax of the text file is MyST markdown[24] and should be self-explanatory. If your edit contains more than a typo correction, leave a commit message summarizing your edit. Then click 'Commit changes'.


Menu
D
Jens Flemming › Data Science and Artificial Intelligence for Undergraduates › Repository
Edit file
Write
Preview
main
No wrap

content_md/usage/executable-book/contribute.md

~~~
# Contribute

This book is not static. It will grow over time and existing material will be rearranged, upd
future developments in data science, AI and teaching. In this sence, it's a living or dynamic
to contribute to the book.

The book's HTML rendering shows a contribution button in the upper right corner offering seve
discussed in detail below.

```{figure} repo-button.png
---
alt: screenshot of HTML rendering while hovering over the contribution button, contribution o
'suggest edit'
figclass: bordered
---
Hovering over the contribution button shows all options for contributing to the book.
```
~~~

Commit message: Update contribute.md

Target Branch: [redacted]-fh-zwickau.de-main-patch-27186

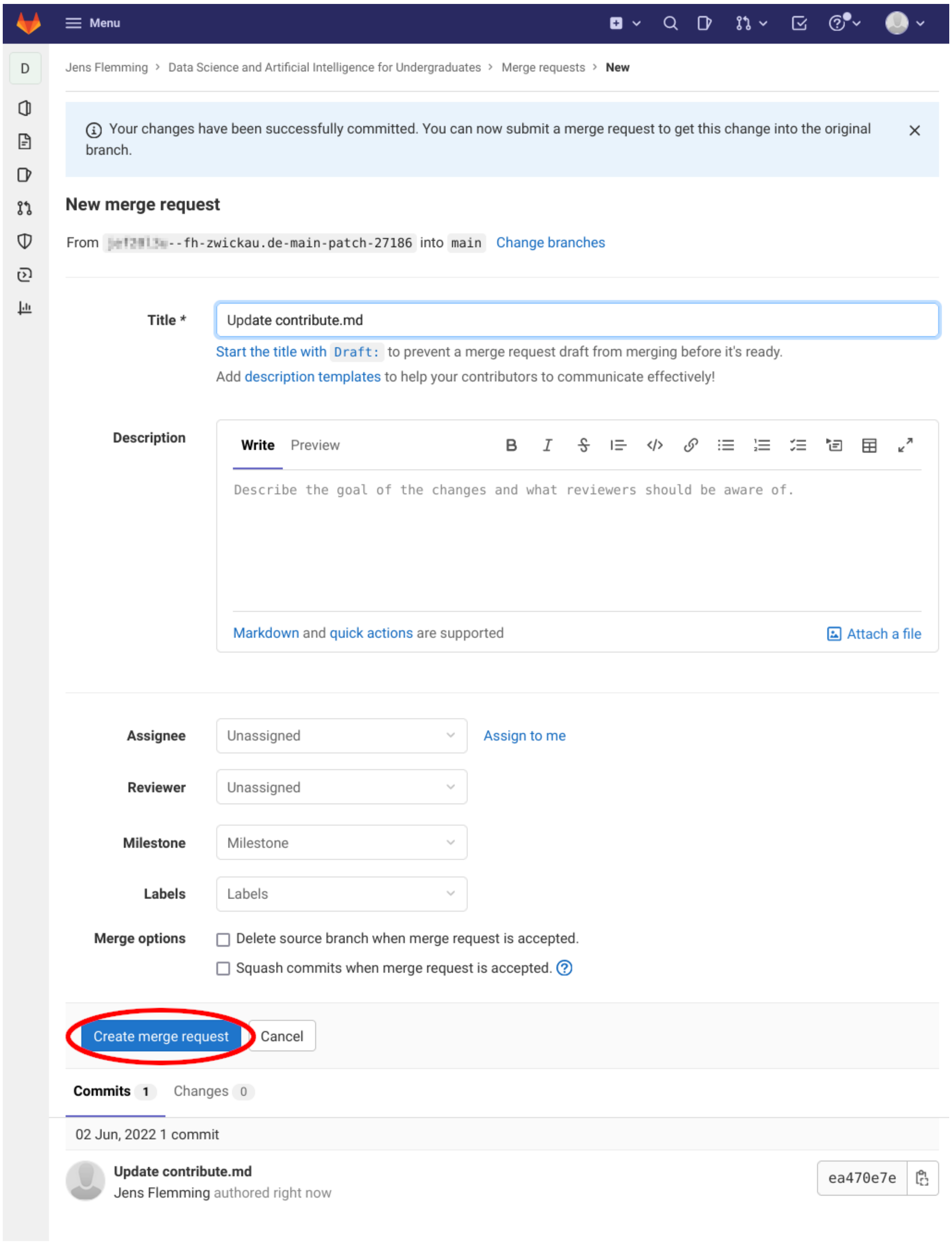
✓ Start a **new merge request** with these changes

Fig. 1.11: Edit the pages markdown source and leave a commit message to describe more elaborate edits.

On the next page click 'Create merge request'. This will send a notification to the author that somebody suggested an edit.

The author will have a look at your suggestion as soon as possible. GitLab's merge request feature allows for discussing and modifying the edit if necessary before the author merges the edit into the book's source and, thus, into the published book.

[24] https://myst-parser.readthedocs.io



Jens Flemming > Data Science and Artificial Intelligence for Undergraduates > Merge requests > **New**

Your changes have been successfully committed. You can now submit a merge request to get this change into the original branch.

New merge request

From `[redacted]--fh-zwickau.de-main-patch-27186` into `main` Change branches

Title * Update contribute.md

Start the title with `Draft:` to prevent a merge request draft from merging before it's ready.
Add description templates to help your contributors to communicate effectively!

Description

Describe the goal of the changes and what reviewers should be aware of.

Markdown and quick actions are supported

Assignee	Unassigned — Assign to me
Reviewer	Unassigned
Milestone	Milestone
Labels	Labels
Merge options	☐ Delete source branch when merge request is accepted.
	☐ Squash commits when merge request is accepted.

02 Jun, 2022 1 commit

Update contribute.md
Jens Flemming authored right now

ea470e7e

Fig. 1.12: Simply click ‘Create merge request’. Fill the description field if you feel a need for additional explanation next to your commit message.

1.3.4 Other Forms of Contribution

If you don't have an account at the book's GitLab instance feel free to send issues and suggestions for edits to the author[25] by email.

[25] https://www.fh-zwickau.de/~jef19jdw

CHAPTER

TWO

GUIDED READING

The teaching material in this book, including exercises and projects, may be arranged in different ways to meet the needs of a comprehensive lecture series or to accompany a one-day workshop on machine learning, for instance. In each section the chapter presents a selection of material together with hints on working through it. Students of Zwickau University's data science course will find the material for each semester here.

- *Data Science I (course at WHZ)* (page 15)
- *Data Science II (course at WHZ)* (page 20)
- *Data Science III (course at WHZ)* (page 24)
- *Data Science IV (course at WHZ)* (page 28)

2.1 Data Science I (course at WHZ)

The first part of the data science lecture series introduces the Python programming language and some Python libraries required for data processing. Next to Python the focus is on working with big data, obtaining, understanding and restructuring data, as well as extracting basic statistical information from data.

2.1.1 Warm-Up

Week 1

Lectures

- *Data Science, AI, Machine Learning* (page 33)
- *Python and Jupyter* (page 37)
- *Computers and Programming* (page 43)

Self-study

- *Computer Basics* (page 847) (exercises)
- *Install and Use Python* (page 901)
  - *Working with JupyterLab* (page 901) (project)

Practice session

- *Install and Use Python* (page 901)
  - *Install Jupyter Locally* (page 905) (project)

2.1.2 Python for Data Science

Week 2 (Crash Course I)

Lectures

- *Crash Course* (page 53)
  - *A First Python Program* (page 53)
  - *Building Blocks* (page 56)

Self-study

- *Finding Errors* (page 851) (exercises)
- *Basics* (page 857) (exercises)

Practice session

- *Install and Use Python* (page 901)
  - *Python Without Jupyter* (page 909) (project)
- *Python Programming* (page 915)
  - *Simple List Algorithms* (page 915) (project)

Week 3 (Crash Course II)

Lectures

- *Crash Course* (page 53), continued
  - *Screen IO* (page 66)
  - *Library Code* (page 67)
  - *Everything is an Object* (page 69)

Self-study

- *More Basics* (page 859) (exercises, last task is bonus)

Practice session

- *Python Programming* (page 915)
  - *Geometric Objects* (page 918) (project, last section is bonus)

Week 4 (Variables and Operators)

Lectures

- *Variables and Operators* (page 85)
  - *Names and Objects* (page 85)
  - *Types* (page 90)
  - *Operators* (page 95) (section *Operators as Member Functions* (page 98) is bonus)
  - *Efficiency* (page 99) (all but section *Garbage Collection* (page 101) is bonus)

Self-study

- *Variables and Operators* (page 861) (exercises)
- *Memory Management* (page 864) (exercises, all but the last two tasks are considered bonus)

Practice session

- *Weather* (page 923)
  - *DWD Open Data Portal* (page 923) (project)
- *Python Programming* (page 915)
  - bonus: *Vector Multiplication* (page 919) (project)

Week 5 (Lists and Friends, Strings)

Lectures

- *Lists and Friends* (page 103)
  - *Tuples* (page 103)
  - *Lists* (page 105)
  - *Dictionaries* (page 108)
  - *Iterable Objects* (page 108)
- *Strings* (page 115)
  - *Basics* (page 115)
  - *Special Characters* (page 117)
  - *String Formatting* (page 119)

Self-study

- *Lists and Friends* (page 867) (exercises)
- *Strings* (page 870) (exercises, last one is bonus)

Practice session

- *MNIST Character Recognition* (page 931)
  - *The xMNIST Family of Data Sets* (page 931) (project)

Week 6 (Accessing Data)

Lectures

- *Accessing Data* (page 121)
  - *File IO* (page 121)
  - *Text Files* (page 124)
  - *ZIP Files* (page 128)
  - *CSV Files* (page 128)
  - *HTML Files* (page 129)
  - *XML Files* (page 131)
  - *Web Access* (page 132)

Self-study

- *File Access* (page 872) (exercises)

Practice session

- *Cafeteria* (page 947), download part (project)

Week 7 (Functions, Modules, Packages)

Lectures

- *Functions* (page 137)
  - *Basics* (page 137)
  - *Passing Arguments* (page 138)
  - *Anonymous Functions (Lambdas)* (page 142)
  - *Function and Method Objects* (page 142)
  - *Recursion* (page 143)
- *Modules and Packages* (page 145)

Self-study

- *Functions* (page 873) (exercises)

Practice session

- *Cafeteria* (page 947), parsing part (project)

Week 8 (Errors, Debugging, Inheritance)

Lectures

- *Error Handling and Debugging Overview* (page 149)
- *Inheritance* (page 153) (last section *Exceptions Inherit from Exception* (page 157) is bonus)
- *Unified Modeling Language (UML)* (page 1027) (bonus)

Self-study

- *Object-Oriented Programming* (page 875) (exercises)
- *Further Python Features* (page 159) (bonus reading)

Practice session

- *Weather* (page 923)
  - *Getting Forecasts* (page 925), download part (project)

2.1.3 Managing Data with Python

Week 9 (NumPy Basics)

Lectures

- *Efficient Computations with NumPy* (page 165)
  - *NumPy Arrays* (page 165)
  - *Array Operations* (page 171)
  - *Advanced Indexing* (page 175)
  - *Vectorization* (page 176)

Self-study

- *NumPy Basics* (page 877) (exercises)

Practice session

- *Weather* (page 923)

  - *Getting Forecasts* (page 925), parsing part (project, automatic download is bonus)

Week 10 (Advanced NumPy)

Lectures

- *Efficient Computations with NumPy* (page 165), continued
  - *Array Manipulation Functions* (page 178)
  - *Copies and Views* (page 181)
  - *Efficiency Considerations* (page 183)
  - *Special Floats* (page 185)
  - *Linear Algebra Functions* (page 187)
  - *Random Numbers* (page 189)
- *Saving and Loading Non-Standard Data* (page 191)

Self-study

- *Image Processing with NumPy* (page 879) (exercises, last one is bonus)

Practice session

- *MNIST Character Recognition* (page 931)
  - *Load QMNIST* (page 933) (project)

Week 11 (Pandas Basics)

Lectures

- *High-Level Data Management with Pandas* (page 199)
  - *Series* (page 200)
  - *Data Frames* (page 210)

Self-study

- *Pandas Basics* (page 881) (exercises)

Practice session

- *Public Transport* (page 949)
  - *Get Data and Set Up the Environment* (page 949) (project)

Week 12 (Advanced Indexing, Dates and Times)

Lectures

- *High-Level Data Management with Pandas* (page 199), continued
  - *Advanced Indexing* (page 219)
  - *Dates and Times* (page 229)

Self-study

- *Pandas Indexing* (page 884) (exercises)

Practice session

- *Corona Deaths* (page 959) (project)

Week 13 (Categories, Restructuring)

Lectures

- *High-Level Data Management with Pandas* (page 199), continued
  - *Categorical Data* (page 235)
  - *Restructuring Data* (page 239)
  - *Performance Issues* (page 245)

Self-study

- *Advanced Pandas* (page 886) (exercises)
- *Pandas Vectorization* (page 889) (exercises)

Practice session

- *Weather* (page 923)
  - *Climate Change* (page 927) (project)

2.2 Data Science II (course at WHZ)

The second semester of the data science lecture series starts with visualization techniques. Then supervised machine learning for generating predictions from data is introduced. Linear regression and artificial neural networks are discussed in depth.

2.2.1 Data Visualization

Week 1 (Matplotlib Basics)

Lectures

- *Matplotlib* (page 251)
  - *Matplotlib Basics* (page 251)

Self-study

- *Matplotlib Basics* (page 891) (exercises; *Circular Colorbar* is bonus)

Practice session

- *Chemnitz Trees* (page 961): download, cleaning, short names (project)

Week 2 (Advanced Matplotlib)

Lectures

- *Matplotlib* (page 251), continued
  - *3D Plots* (page 282)
  - *Animations* (page 287)
  - *Seaborn* (page 292)
  - *Maps* (page 296)

Self-study

- *Advanced Matplotlib* (page 893) (exercises)

Practice session

- *Chemnitz Trees* (page 961): information extraction, presentation (project)

Week 3 (Ploty and Folium)

Lectures

- *Plotly* (page 303)
- *Folium* (page 305)

Self-study

- revisit *Get Data and Set Up the Environment* (page 949) (project)

Practice session

- *Find Connections* (page 954) (project)

2.2.2 Supervised Learning

Week 4 (Introduction)

Lectures

- *Machine Learning Overview* (page 311)
- *General Considerations* (page 313)
  - *Problem and Workflow* (page 313)
  - *Introductory Example (k-NN)* (page 317)

Self-study

Practice session

- *Interactive Map* (page 956) (project, *Color-Coded Distances* are bonus)

Week 5 (Quality Measures)

Lectures

- *General Considerations* (page 313), continued
  - *Quality Measures* (page 328) (proof of AUC interpretation is bonus)

Self-study

- revisit *Climate Change* (page 927) (project)

Practice session

- *Weather Animation* (page 928) (project)

Week 6 (Feature Reduction)

Lectures

- *General Considerations* (page 313), continued
  - *Feature Reduction* (page 343)

Self-study

Practice session

- *Forged Banknotes* (page 963)
  - *Detecting Forgery with k-NN* (page 963) (project)
  - *Quality Measures* (page 965) (project, AUC section is bonus)

Week 7 (Hyperparameter Optimization)

Lectures

- *General Considerations* (page 313), continued
  - *Hyperparameter Tuning* (page 350)

Self-study

- *MNIST Character Recognition* (page 931)
  - revisit/complete *Load QMNIST* (page 933) (project)
- revisit/complete *Image Processing with NumPy* (page 879) (exercises)

Practice session

- *MNIST Character Recognition* (page 931)
  - *QMNIST Feature Reduction* (page 935) (project)

Week 8 (Linear Regression)

Lectures

- *Linear Regression* (page 359)
  - *Approach* (page 359)
  - *Regularization* (page 375)
  - *Worked Example: House Prices I* (page 388)

Self-study

- *Linear Regression* (page 359), continued
  - *Worked Example: House Prices II* (page 421)

Practice session

- *Forged Banknotes* (page 963)
  - *Hyperparameter Optimization* (page 967) (project)
- *House Prices* (page 971)
  - *House Prices GUI* (page 971) (bonus project)

Week 9 (More on Regression)

Lectures

- *Linear Regression* (page 359), continued
  - *Outliers* (page 430)
- *Logistic Regression* (page 441)

Self-study

- finish previous projects

Practice session

- finish previous projects

Week 10 (ANN Basics)

Lectures

- *Artificial Neural Networks* (page 453)
  - *ANN Basics* (page 453)
  - *Training ANNs* (page 462) (mathematical derivation of ANN gradients and ANN implementation from scratch are bonus)

Self-study

Practice session

- *House Prices* (page 971)
  - *House Prices ANN* (page 972)

Week 11 (ANNs with Keras)

Lectures

- *Cloud Computing* (page 1029)
- *Artificial Neural Networks* (page 453), continued
  - *ANNs with Keras* (page 479)

Self-study

Practice session

- *PCA and ANN for QMNIST* (page 936)
- *Long-Running Tasks* (page 912)

Week 12 (CNNs, part I)

Lectures

- *Artificial Neural Networks* (page 453), continued
  - *Convolutional Neural Networks* (page 498)
  - *CNNs with Keras* (page 511)

Self-study

Practice session

- *CNN for QMNIST* (page 936)

Week 13 (CNNs, part II)

Lectures

- *Artificial Neural Networks* (page 453), continued
  - *What did the CNN learn?* (page 521)
  - *Improving CNN performance* (page 570)

Self-study

Practice session

- *CNN Analysis for QMNIST* (page 936)
- *Hyperparameter Optimization for Cats and Dogs* (page 975) (bonus)

2.3 Data Science III (course at WHZ)

Part three of the data science lecture series continues discussion of supervised machine learning. Further methods like decision trees and support vector machines are introduced. Then we move on to unsupervised machine learning covering clustering methods and techniques for dimensionality reduction.

2.3.1 Supervised Learning

Week 1 (Decision Trees)

Lectures

- *Decision Trees* (page 587)
  - *Basics* (page 587)
  - *Regression Trees* (page 589)
  - *Classification Trees* (page 595)

Self-study

- *IBAN recognition* (page 937) (project)

Practice session

- *Forged Banknotes* (page 963)
  - *Decision Tree* (page 967) (project)

Week 2 (Ensemble Methods)

Lectures

- *Ensemble Methods* (page 599)
  - *The Idea* (page 599)
  - *Stacking* (page 600)
  - *Bagging* (page 600)
  - *Boosting* (page 601)

Self-study

- revisit *Worked Example: House Prices II* (page 421)

Practice session

- *Forged Banknotes* (page 963)
  - *Random Forest* (page 968) (project)
- *House Prices* (page 971)
  - *A Random Forest for House Prices* (page 973) (project)

Week 3 (Support-Vector Machines)

Lectures

- *Support-Vector Machines* (page 607)
  - *Hard Margin SVMs* (page 607)
  - *Soft Margin SVMs* (page 612)
  - *Kernel SVMs* (page 615)
  - *SVMs with Scikit-Learn* (page 616)

Self-study

- *Support-Vector Machines* (page 607)
  - *Support-Vector Regression (SVR)* (page 620) (bonus)

Practice session

- *Forged Banknotes* (page 963)
  - *Support-Vector Machine* (page 969) (project)
- *MNIST Character Recognition* (page 931)
  - *SVM for QMNIST* (page 939) (project)

Week 4 (Naive Bayes)

Lectures

- *Naive Bayes Classification* (page 623) (without kernel density estimates)

Self-study

- *Naive Bayes Classification* (page 623) (kernel density estimates, bonus)

Practice session

- *Forged Banknotes* (page 963)
  - *Naive Bayes Classification* (page 969) (project)

Week 5 (Text Classification)

Lectures

- *Text Classification* (page 627)
  - *Preprocessing Text Data* (page 627)
  - *Training* (page 647)

Self-study

Practice session

Week 6 (Text Classification)

Lectures

Self-study

Practice session

- *Blogs* (page 977)
  - *Blog Author Classification (Training)* (page 977) (project)
  - *Blog Author Classification (Test)* (page 980) (project)

2.3.2 Unsupervised Learning

Week 7 (Introduction, Centroid-based Clustering)

Lectures

- *Typical Tasks* (page 683)
- *Quality Measures for Clustering* (page 687)
- *Centroid-based Clustering (k-Means)* (page 691)

Self-study

Practice session

- *MNIST Character Recognition* (page 931)
  - *Semisupervised Classification* (page 940) (project)

Week 8 (Hierarchical Clustering)

Lectures

- *Hierarchical Clustering* (page 709)

Self-study

Practice session

- *Supermarket Customers* (page 983) (project)
- *Chinese Celadons* (page 987)
  - *Hierarchical Clustering* (page 987) (project)

Week 9 (Density-based Distribution-based Clustering)

Lectures

- *Density-based Clustering* (page 717)
  - *DBSCAN* (page 717)
  - *OPTICS* (page 720)
- *Distribution-based Clustering* (page 731)

Self-study

Practice session

- *Chinese Celadons* (page 987)
  - *Density-based Clustering* (page 988) (project)
- *MNIST Character Recognition* (page 931)
  - *Generating Handwritten Digits* (page 942) (project)

Week 10 (Autoencoders)

Lectures

- *Autoencoders* (page 739)

Self-study

Practice session

- *MNIST Character Recognition* (page 931)
  - *Autoencoder for QMNIST* (page 943) (project)

Week 11 (Nonlinear Dimensionality Reduction, Kernel PCA)

Lectures

- *Nonlinear Dimensionality Reduction Overview* (page 753)
- *Kernel PCA* (page 759)

Self-study

- *Feature Reduction* (page 343) (reread PCA section)

Practice session

Week 12 (Multidimensional Scaling)

Lectures

- *Multidimensional Scaling* (page 765)

Self-study

- *Feature Reduction* (page 343) (reread PCA section)

Practice session

- *Color Perception* (page 991) (project)
- *Forest Fires* (page 995) (project)

Week 13 (LLE, SNE, SOM)

Lectures

- *Locally Linear Embedding* (page 773)
- *Stochastic Neighbor Embedding* (page 777)
- *Self-Organizing Maps* (page 781)

Self-study

Practice session

- *MNIST Character Recognition* (page 931)
  - *t-SNE for QMNIST* (page 945) (project)
- *House Prices* (page 971)
  - *House Prices SOM* (page 973) (project)

2.4 Data Science IV (course at WHZ)

The last part of the data science lecture series is devoted to reinforcement learning. Next to very basic techniques we also discuss state-of-the-art deep reinforcement learning with artificial neural networks.

2.4.1 Reinforcement Learning

Week 1 (Overview and Stateless Tasks)

Lectures

- *Overview and Examples* (page 793)
  - *What is Reinforcement Learning?* (page 793)
  - *Examples* (page 795)
  - *Maximization of Return* (page 798)
- *Stateless Learning Tasks (Multi-armed Bandits)* (page 801)

Self-study

Projects and Exercises

- *Modeling* (page 895) (exercises)
- *Online Advertising* (page 997) (project)

Week 2 (Markov Decision Processes)

Lectures

- *Markov Decision Processes* (page 807)
  - *Basic Notions* (page 807)
  - *Bellman Equations and Optimal Policies* (page 810)

Self-study

Projects and Exercises

- *Markov Decision Processes* (page 896) (exercises)

Week 3 (Dynamic Programming)

Lectures

- *Dynamic Programming* (page 817)

Self-study

Projects and Exercises

- *Frozen Lake* (page 1001)
  - *Dynamic Programming* (page 1001) (project)

Week 4 (Monte Carlo Methods)

Lectures

- *Monte Carlo Methods* (page 819)

Self-study

Projects and Exercises

- *Frozen Lake* (page 1001)
  - *Monte Carlo Methods* (page 1002) (project)

Week 5 (Temporal Difference Learning)

Lectures

- *Temporal Difference Learning (TD Learning)* (page 825)

Self-study

Projects and Exercises

- *Frozen Lake* (page 1001)
  - *SARSA* (page 1003) (project)

Week 6 (Tic-Tac-Toe)

Lectures

Self-study

Projects and Exercises

- *Q-Learning for Tic-Tac-Toe* (page 1005)

Week 7 (Approximate Methods)

Lectures

- *Approximate Value Function Methods* (page 829)
  - *Principal Approach* (page 829)
  - *Policy Evaluation* (page 830)
  - *Policy Improvement* (page 832)

Self-study

Projects and Exercises

Week 8 (Cart Pole SARSA)

Lectures

Self-study

Projects and Exercises

- *Cart Pole* (page 1011)
  - *The Cart Pole Environment* (page 1011)
  - *SARSA* (page 1012)

Week 9 (Deep Q-Learning)

Lectures

- *Approximate Value Function Methods* (page 829)
  - *Deep Q-Learning* (page 837)

Self-study

Projects and Exercises

- *Cart Pole* (page 1011)
  - *Deep Q-Learning* (page 1012)

Week 10 (Policy Gradient Methods)

Lectures

- *Policy Gradient Methods* (page 839)

Self-study

Projects and Exercises

Week 11 (Policy Gradient Methods)

Lectures

Self-study

Projects and Exercises

- *Cart Pole* (page 1011)
  - *Policy Gradient Method* (page 1013)

Part II

Warm-Up

CHAPTER

THREE

DATA SCIENCE, AI, MACHINE LEARNING

Data Science comes in different flavors and sometimes denotes different things. Some clarification on the terms used in this book and on the subjects covered is mandatory.

3.1 Science With and Of Data

With the advent of cheap storage devices in the last decade of the 20th century companies, governments, other organizations and also private individuals started collecting data at large scale (*big data*). In a world full of data somebody has to think about how to make information accessible which is hidden in data. Computer Scientists and Mathematicians developed a bunch of methods for extracting information, more and more applications popped up, methods became more complex,… a new field of research was born. This new field matured, got the name 'data science' and now is accepted as serious field of research and teaching.

Data Science as a science field covers all technical aspects of data processing. There's large overlap with computer science and mathematics, but also with many other fields, depending on where data comes from. Mathematics provides advanced methods for extracting information from data. Computer science allows for their realization.

Data Science also touches law, ethics and sociology. May I use this data set for my project? Is it okay to collect and dig through personal data? What impact will extensive data collection and processing have on society?

Almost every data science project has four phases:

1. Collect Data

Data has to be recorded and stored somehow. Planning and realizing data collection processes is referred to as *data engineering*. Typical tasks in this phase are, for instance, installing and configuring sensors, setting up data base storage, and implementing techniques for supervising data flow.

2. Clean and Restructure Data

Raw data sets often contain errors, missing items or false items. They have to be cleaned. Almost always several data sets have to be combined to allow for succesful extraction of information. These preprocessing steps require lots of manual work and domain knowledge. Careful preprocessing will simplify subsequent processing steps and is at least as important as the modeling phase.

3. Create a Model

From recorded and preprocessed data a mathematical or algorithmic model is build. Depending on the concrete problem to solve from data, such a model may describe the data set (*descriptive model*) or it may be used to answer some question based on the data set (*predictive model*).

4. Communicate Results

Findings from the data have to be communicated to the client. Visualizations are the most important tool for delivering results.

In this book we focus on preprocessing and modeling. Data engineering and communication of results will be touched occasionally only. The visualization aspect of communication also plays an important role in preprocessing when exploring a new data set (*explorative data analysis*, short: *EDA*). So we will cover the full range of visualization tools and techniques there.

3.2 Example: Customer Segmentation

Brick-and-mortar stores as well as online shops collect as much customer data as they can to understand customer behavior. Knowing how many people buy which products at which time in which quantities is essential for efficient warehousing. But customer data is also used for targeted ad campaigns.

For targeted advertising one tries to identify groups of customers with similar behavior. For each group tailor-made ads are created. Customer segmentation is an example of descriptive modeling. The aim is to understand the collected data and to find structures not obvious at first glance.

Typical tasks in the four phases described above are:

1. Collect Data

- implement a network infrastructure to collect sales data from all stores in a central data base
- issue customer cards to know who comes to your shop (age, gender, location,…)
- think about buying external data about your customers (Schufa,…)
- check legal situation to know whether you are allowed to collect the data you want

2. Clean and Restructure Data

- throw away all the data not relevant for segmentation (for instance, data of customers not living in the targeted region)
- transform data (for instance, convert absolute quantities to relative quantities: milk made 5% of the shopping cart)
- restructure data to get per-customer data instead of per-shop or per-product

3. Create a Model

- apply some standard segmentation method
- try to understand the identified customer groups, find unique characteristics

4. Communicate Results

- present groups and their unique characteristics to the advertising department

3.3 Example: Weather Forecast

Weather forecasting is a typical example of predictive modeling. From past data we want to create a model which yields information on future weather parameters. In the past lots of experts analyzed recorded weather data and made predictions mainly from experience and classical mathematical and physical modeling. Data science allows to automate the forecasting process. Instead of handcrafted models and expert knowledge one creates a predictive data model based on all (or sufficiently much) recorded weather data.

1. Collect Data

- decide which weather parameters to record (temperature, humidity,…)
- implement a network infrastructure to collect weather data from across the world
- build and launch satellites
- build terrestrial weather stations

2. Clean and Restructure Data

- decide for a subset of data to use for forecasting (for instance, only use data from past 30 days)
- transform data (for instance, harmonize temperature units: Fahrenheit, Celsius)
- restructure data (for instance, downsample data from 5-minute periods to hourly values)

3. Create a Model

- apply some standard method for predictive modeling
- verify the quality of your model's predictions

4. Communicate Results

- from numerical outputs of the model make a human readable forecast (for instance, round temperatures to at most one decimal place)

3.4 Artificial Intelligence

Artificial intelligence to some extent is a buzzword. It's used for computer programs doing things we consider intelligent. Examples are image classification (what is shown on the image?), language processing (translate a text), autonomous driving (orient and move in a complex environment). Under the hood there's still a classical computer program, no intelligence.

Most, if not all, methods related to artifical intelligence are based on processing large data sets. Image and language processing systems are trained on large data sets of sample images and sample texts. Autonomous driving uses reinforcement learning, which can be understood as collecting large amounts of data while exploiting information extracted from previously collected data (data collection on demand). In this sence, artificial intelligence is a subfield of data science. In this book we also cover this vague field of articifial intelligence, including reinforcement learning.

There's also a strict mathematical definition of artificial intelligence: A computer system is intelligent if it passes the Turing test. In the Turing test a human chats with another human and the computer system in parallel. If the human cannot decide which of both chat partners is human, the computer passed the test. Up to now no computer passed the Turing tests. If interested, have a look at Wikipedia's article on the Turing test[26].

3.5 Machine Learning

By machine learning we denote the process of writing computer programs 'learning' to do something from data. In other words, we set-up a model with lots of unknowns and then fit the model to our data. So machine learning refers to a style of software development. We do not write a program line by line. Instead we use a general purpose program and fill in the details automatically based on some data set.

Machine learning may be regarded as the hard core of data science and artificial intelligence, where all the mathematics is contained in.

[26] https://en.wikipedia.org/wiki/Turing_test
[27] https://xkcd.com/1838

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Fig. 3.1: The pile gets soaked with data and starts to get mushy over time, so it's technically recurrent. Source: Randall Munroe, xkcd.com/1838[27]

CHAPTER

FOUR

PYTHON AND JUPYTER

In this book we use the Python[28] programming language for talking to the computer. Tools from the Jupyter[29] ecosystem allow for Python programming in a very comfortable graphical environment.

4.1 Data Science Tools

There are lots of software tools for data science and artificial intelligence. They can be devided into two groups:

Tailor-made GUI tools

For common tasks in data science and AI like clustering data or classifying images there exist (mostly commerical) tools with graphical user interface (GUI). Such tools are easy to use, but they have a very limited scope of application. Each task requires a different tool. Available methods are restricted to well known ones. Implementing new problem specific methods is not possible.

General Purpose Tools

To enjoy maximum freedom in choice of methods one has to leave the world of GUI tools. Creating data science models (that is, computer programs) without any restrictions requires the use of some high-level programming language. Examples are R[30] and Python[31]. Both languages are very common in the data science community because they ship with lots of extensions for simple usage in data science and AI.

Tailor-made tools come and go as time moves on. Programming languages are much more long-lasting. In this book we stick to the Python programming language and its ecosystem. The R programming language would be a good alternative, but sticks more to statistical tasks than to general purpose programming.

Tip: Some people feel frightend if someone says 'programming language'. Think of programming languages as usual software tools. The only difference is that they provide much more functionality than GUI tools. But there's not enough space on screen to have a button for each function. So we write text commands.

4.2 Why Python?

Python is a modern, free and open source programming language. It dates back to the early 1990s with a first official release in 1994. It's father and BDFL (benevolent dictator for life) is Guido van Rossum[32].

Python code is very readable and straight forward without too many cumbersome symbols like in most other programming languages. Many technical aspects of computer programming are managed by Python instead of by the programmer. With Python one may develop the full range of software, from simple scripts to fully featured web or desktop applications. Thousands of extensions allow for rapid development.

[28] https://www.python.org
[29] https://jupyter.org
[30] https://www.r-project.org/
[31] https://www.python.org
[32] https://en.wikipedia.org/wiki/Guido_van_Rossum
[33] https://xkcd.com/353

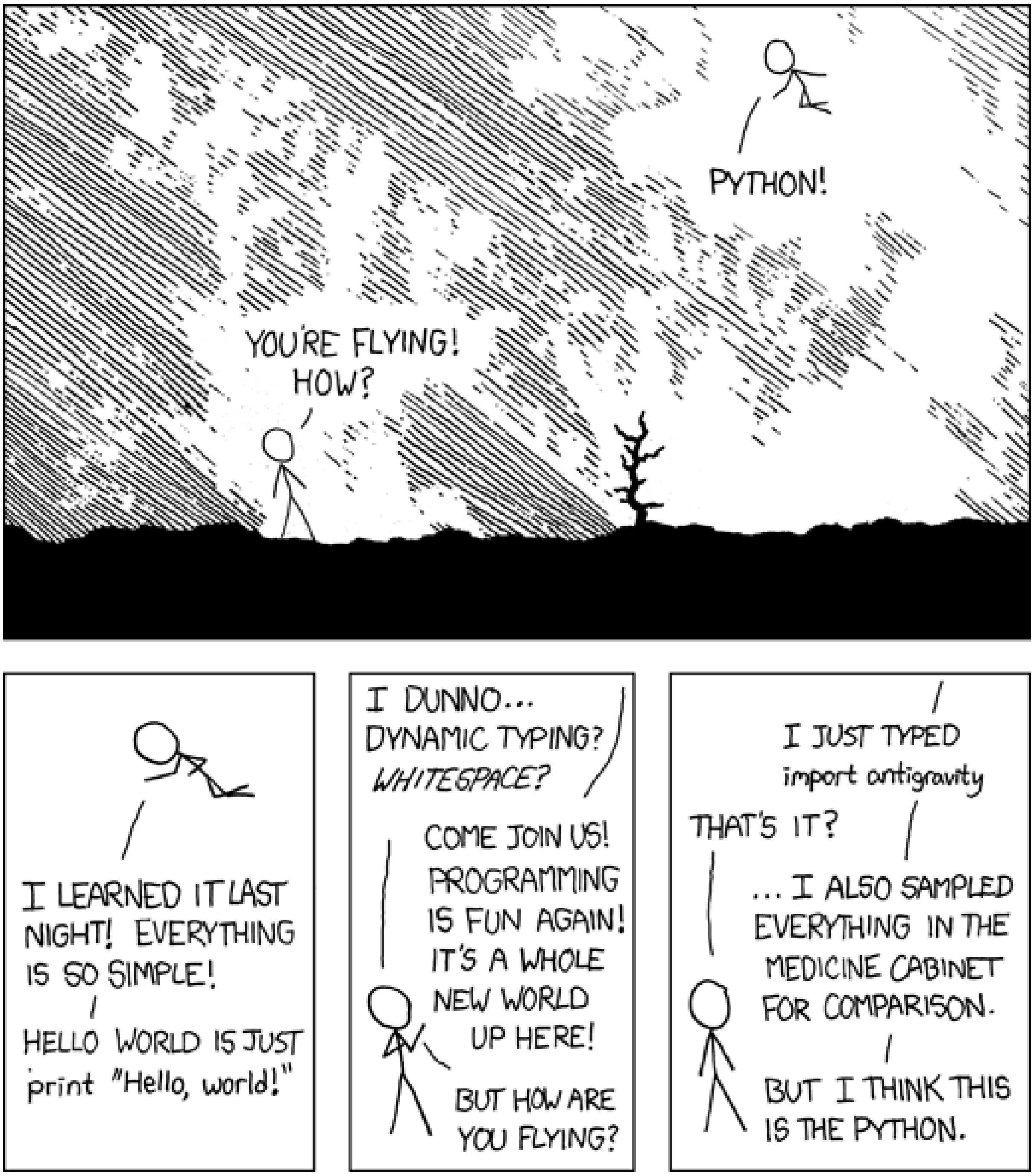

PYTHON!
YOU'RE FLYING! HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

Fig. 4.1: I wrote 20 short programs in Python yesterday. It was wonderful. Perl, I'm leaving you. Source: Randall Munroe, xkcd.com/353[33]

There's a large online community discussing Python topics. Almost every problem you'll encounter has already been solved. Simply use a search engine to find the answer.

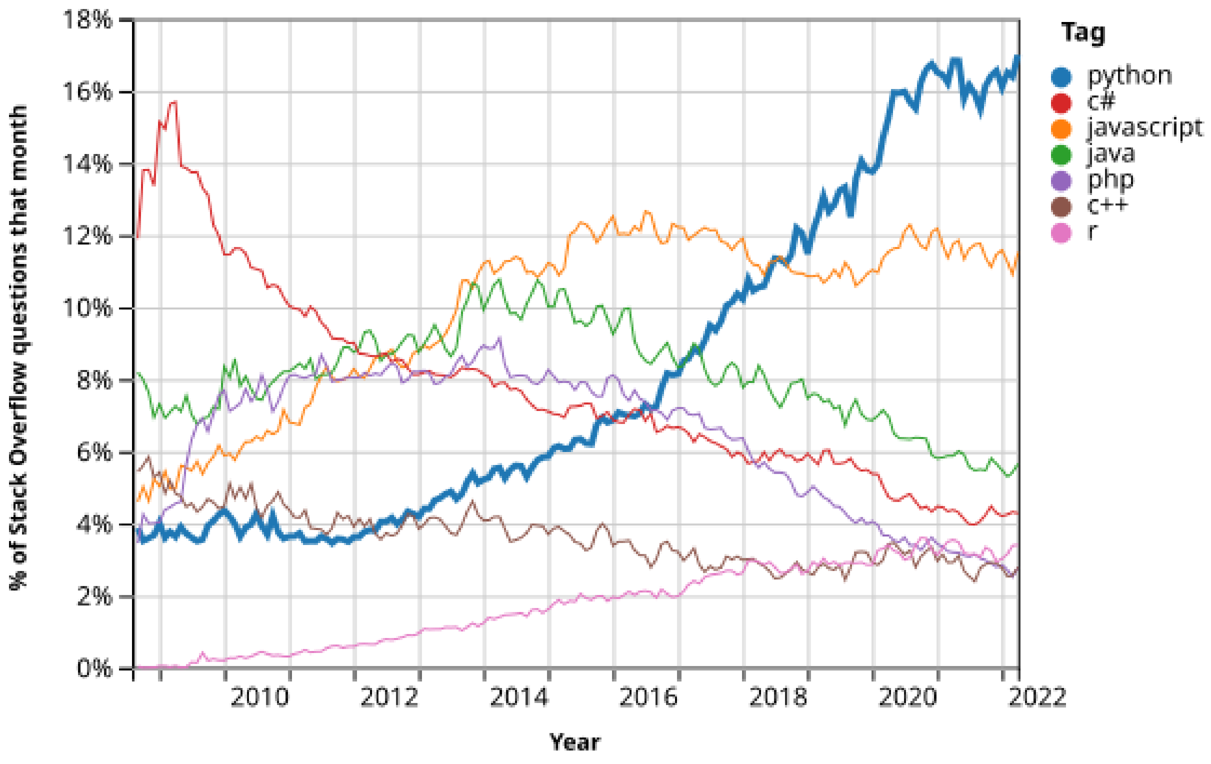


Fig. 4.2: Popuparity of programming languages on Stack Overflow[34]. Source: Stack Overflow Trends[35] (modified by the author)

Some rules followed by Python and its community are collected in the Zen of Python[36]. Here are some of them:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.
- There should be one – and preferably only one – obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.

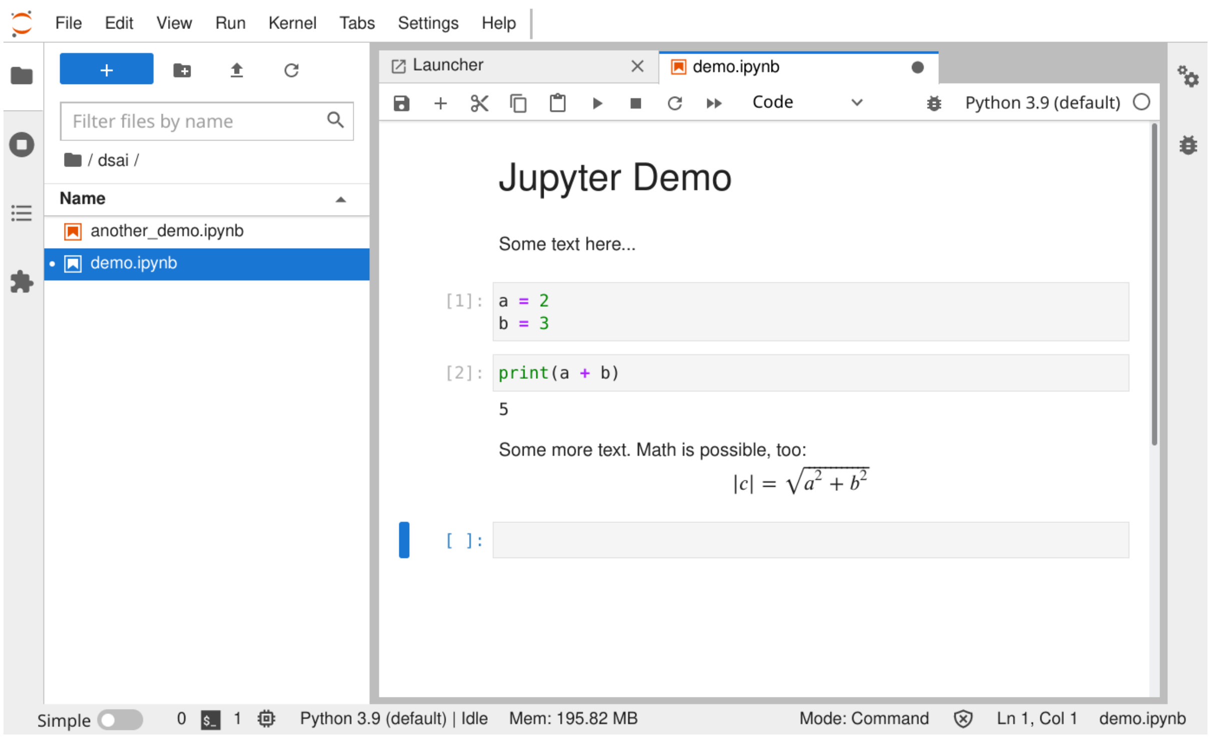
Last but not least Python is available on all platforms, Linux, macOS, Windows, and many more. Youtube's player is written in Python and many other tech giants use Python. But it's also not unlikely that a Python script controls your washing machine.

Hint: There are two versions of Python: Python 2 and Python 3. Source code is not compatible, that is, there are programs written in Python 2 which cannot be executed by a Python 3 interpreter. In this book we stick to Python 3. Python 2 is considered deprecated since January 2020[37].

[34] https://stackoverflow.com
[35] https://insights.stackoverflow.com/trends?tags=python%2Cjavascript%2Cjava%2Cc%23%2Cphp%2Cc%2B%2B%2Cr
[36] https://en.wikipedia.org/wiki/Zen_of_Python
[37] https://www.python.org/doc/sunset-python-2/

4.3 Jupyter

The Jupyter ecosystem is a collection of tools for Python programming with emphasis on data science. Jupyter allows for Python programming in a webbrowser. Outputs, including complex and interactive visualizations, can be put right below the code producing these outputs. Everything is in one document: code, outputs, text, images,…



Jupyter Demo

Some text here...

```
[1]: a = 2
     b = 3
```

```
[2]: print(a + b)
5
```

Some more text. Math is possible, too:

$$|c| = \sqrt{a^2 + b^2}$$

```
[ ]:
```

Fig. 4.3: JupyterLab is the most widely used member of the Jupyter ecosystem. It brings Python to the webbrowser.

In this book you'll meet at least four members of the Jupyter ecosystem.

JupyterLab[38]

JupyterLab is a web application bringing Python programming to the browser. It's the everyday tool for data science. JupyterLab may run on a remote server (cloud) or on your local machine.

Jupyter Notebook[39]

An alternative to JupyterLab is Jupyter Notebook. It's a predecessor of JupyterLab and provides almost identical functionality, but with different look and feel.

JupyterHub[40]

Running JupyterLab in the cloud requires user authentication and user management. JupyterHub provides everything we need to run several JupyterLabs on a server in parallel. Almost all JupyterLab providers (e.g., Gauss at Zwickau University, Binder) rely on JupyterHub.

Jupyter Book[41]

This book is being published using Jupyter Book. Each page is a Jupyter notebook file. Jupyter Book provides automatic generation of table of contents, handling bibliographies and rendering to different output formats.

[38] https://jupyter.org
[39] https://jupyter.org
[40] https://jupyter.org/hub
[41] https://jupyterbook.org

4.4 Install and Use

Work through the following projects to get up and running with Python and Jupyter:

- *Working with JupyterLab* (page 901)
- *Install Jupyter Locally* (page 905)
- *Python Without Jupyter* (page 909)

CHAPTER

FIVE

COMPUTERS AND PROGRAMMING

Computers are the main tool for data science and artificial intelligence. In this chapter we answer some basic questions:

- What is a computer?
- What are bits, bytes, kilobytes,...?
- What is software?
- What is programming and what are programming languages?

Although we don't have to know detailed answers to these questions, we should have some rudimentary understanding of what happens inside a computer.

Related exercises: *Computer Basics* (page 847).

5.1 CPU, Memory, IO

Each modern computer consists of three components: central processing unit (CPU), memory, input/output (IO) devices. These components are connected by many wires, which are organized together with some auxiliary stuff on the computer's mainboard.

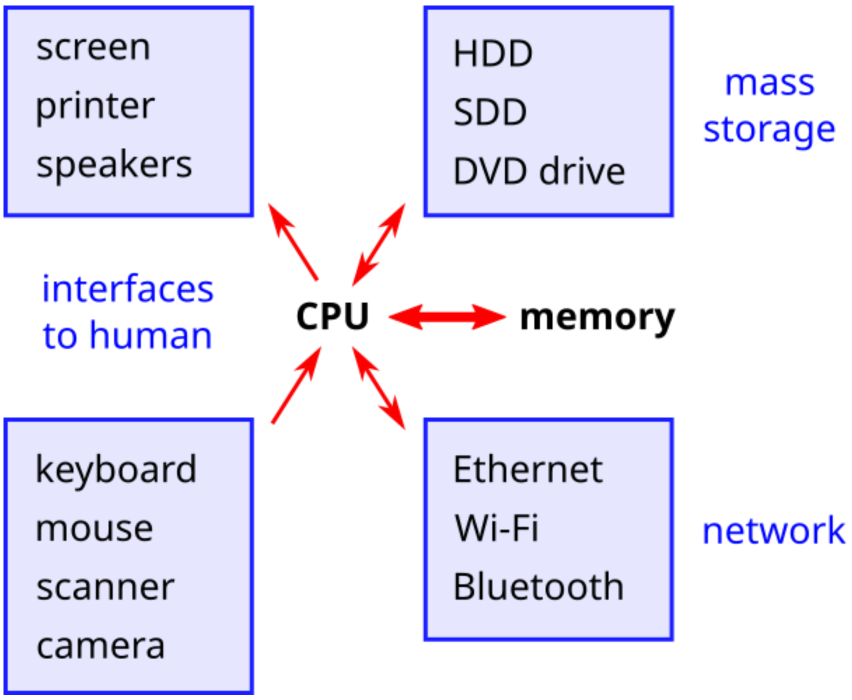


Fig. 5.1: There's a tight and fast data connection between CPU and memory. IO devices are connected to the CPU and in some cases also directly to memory.

IO devices are all parts of the computer which provide an interface to humans like screen, keyboard, printer, scanner. But also mass storage devices (hard disk drives, SSDs, DVD drives, card readers and so on) are IO devices. Another kind of IO devices are network adapters for Ethernet, Wi-Fi, Bluetooth and others. The common feature of all IO devices is that they produce and/or consume streams of binary data. 'Binary' means that there are only two different values, usually denoted by 0 and 1. Electrically, 0 might stand for low voltage and 1 for high voltage.

Memory can store streams of binary data. In some sense it is similar to mass storage IO devices, but it is used in a very different way. Most storage devices are very slow and access times for reading and writing data depend on the position of the data on the device. In contrast, memory access is very fast and access times are independent of the data's concrete location. Whenever data has to be stored for a short time only, memory is used. Due to technological reasons memory loses all data when power is turned off, whereas data on mass storage devices persists.

The **CPU** is a highly integrated circuit which processes streams of binary data. 'Processing' means that incoming data from memory and/or IO devices is transformed and then sent to memory and/or IO devices. If a binary stream from memory is interpreted as instructions by the CPU, then we say that the stream contains code. Data in the stricter sense refers to parts of binary streams that are processed by the CPU, but which do not tell the CPU what to do. Memory can contain code and data, whereas IO devices only produce non-code data.

5.2 Bits and Bytes

A **bit** is a piece of binary information. It either holds a one or a zero. Less information than a bit is no information. With a sequence of k bits we can express 2^k different values, for example the numbers $0, 1, \dots, 2^k - 1$.

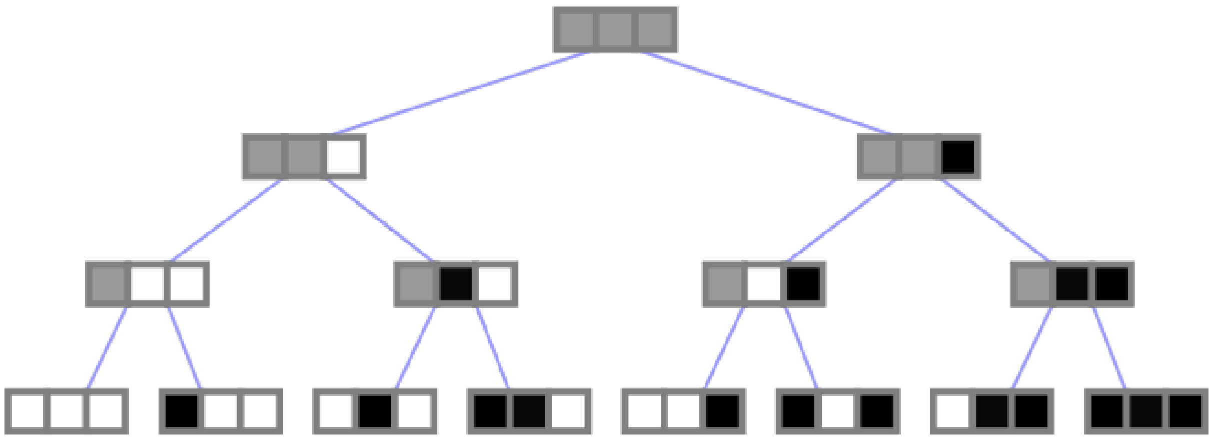


Fig. 5.2: Wtih 3 bits we may represent 8 different values. Each additional bit doubles the number of possible values.

bits	number of values	usual interpretation
1	2	0, 1
2	4	0, 1, 2, 3
3	8	0 … 7
4	16	0 … 15
5	32	0 … 31
6	64	0 … 63
7	128	0 … 127
8	256	0 … 255
16	65 536	0 … 65 535
24	16 777 216	0 … 16 777 215
32	4 294 967 296	0 … 4 294 967 295

By convention binary data in modern computers is organized in groups of 8 bits. A sequence of 8 bits is denoted as a **byte**.

Following the metric system, there are kilobytes (1000 byte), megabytes (1000 kilobyte), gigabytes (1000 megabytes), and so on with prefixes tera, peta, exa, zetta, yotta. Corresponding symbols are kB or KB, MB, GB, TB, PB, EB, ZB, YB.

In some hardware oriented fields of computer science it is common practice to use the factor 1024=210 instead of 1000. Thus, the size of a kilobyte may be 1000 or 1024 bytes. As a rule of thumb 1000 is used for data transmission and 1024 is used for memory and storage related things (except in adds for storage devices, because 1024 would give a lower number of gigabytes). Sometimes the prefixes kibi, mebi, gibi, tebi, pebi, exbi, zebi, yobi are used with corresponding symbols KiB, MiB, GiB, TiB, PiB, EiB, ZiB, YiB for factor 1024. One kibibyte, for instance, has 1024 bytes.

factor	name	symbol	bytes
1000	kilobyte	kB or KB	1 000
1024	kibibyte	KiB	1024
1000	megabyte	MB	1 000 000
1024	mebibyte	MiB	1 048 576
1000	gigabyte	GB	1 000 000 000
1024	gibibyte	GiB	1 073 741 824
1000	terabyte	TB	1 000 000 000 000
1024	tebibyte	TiB	1 099 511 627 776
1000	petabyte	PB	1 000 000 000 000 000
1024	pebibyte	PiB	1 125 899 906 842 624
1000	exabyte	EB	1 000 000 000 000 000 000
1024	exbibyte	EiB	1 152 921 504 606 846 976
1000	zettabyte	ZB	1 000 000 000 000 000 000 000
1024	zebibyte	ZiB	1 180 591 620 717 411 303 424
1000	yottabyte	YB	1 000 000 000 000 000 000 000 000
1024	yobibyte	YiB	1 208 925 819 614 629 174 706 176

Important: Computers only work with binary data. Everything has to be represented as sequences of zeros and ones. For integers, like 123, this is quite simple (see below). Rational numbers, like 0.123, may be represented by two integers, a numerator 123 and a denominator 1000 for instance. But what about text data? Or images?

Data which cannot be represented as sequence of zeros and ones cannot be processed by a computer. We'll come back to this representation issue several times in this book.

5.3 Representation of Numbers

Numbers may have a name, like *one*, *two*, *three*, *four*, *five*, *six*, *seven*, *eight*, *nine*, *ten*. There are even more named numbers: *eleven*, *twelve* and *zero*, for instance. Obviously, not all numbers can have an individual name. We need a system for automatically naming numbers and also for writing them down. At this point it is important to distinguish between numbers, which can be used for counting and computations, and their representation in spoken and written language.

In everyday life we use the decimal system based on 10 different digit symbols because we have 10 fingers. An octopus surely would invent a numbering system with only 8 digits. The Maya civilization used a 20 digits system (fingers plus toes). Computers would have invented number systems based on 2 digits, because they are representable by 1 bit, or 4 digits (2 bits) or 8 (3 bits) or 16 (4 bits).

[42] https://en.wikipedia.org/wiki/Dresden_Codex
[43] https://commons.wikimedia.org/wiki/File:Maya_Hieroglyphs_Plate_32.jpg

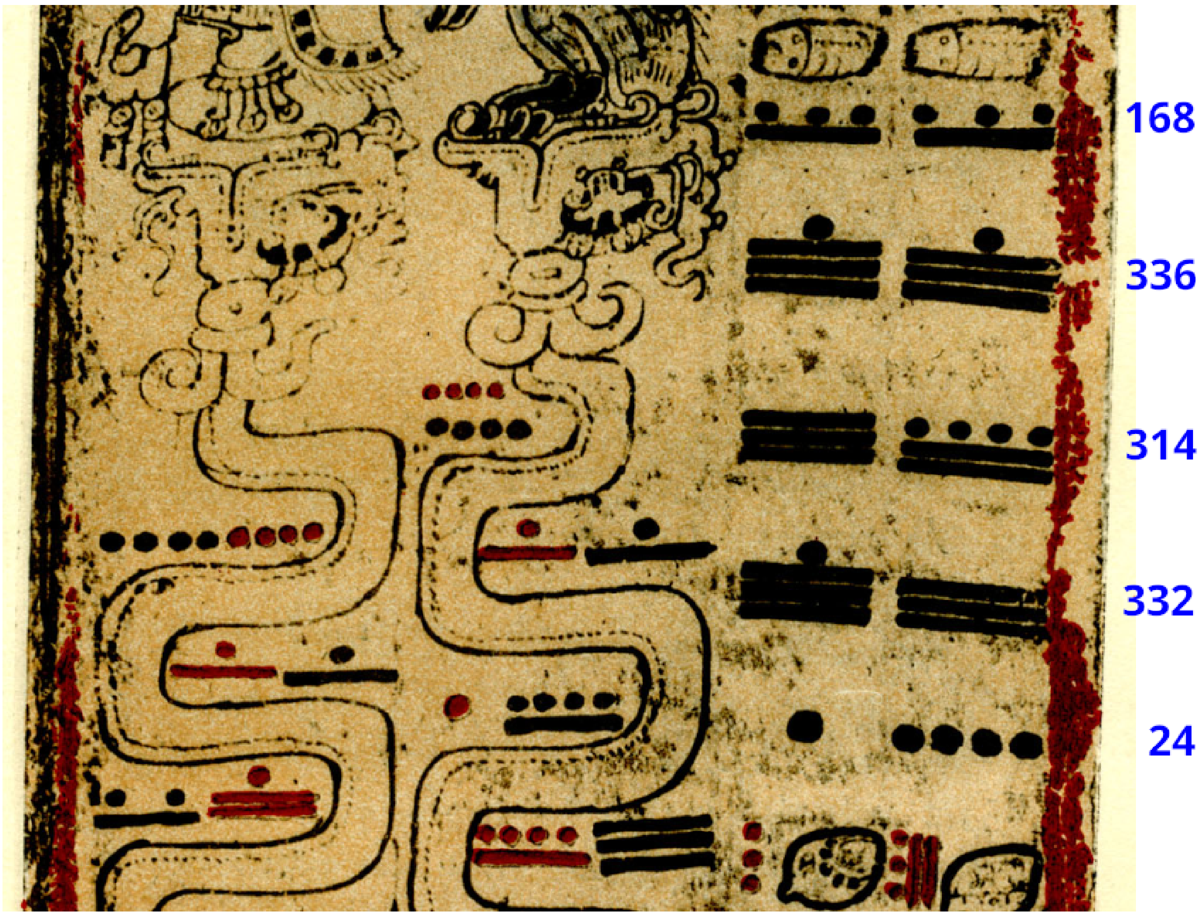


Fig. 5.3: Maya numerals on a page of a Maya book known as Dresden Codex[42]. Source[43]: Sylvanus Morley via Wikimedia Commons, modified by the author.

5.3.1 Positional Notation of Numbers

There are many systems for writing down (and naming) numbers. Today the most widely used ones are positional. An example for a non-positional system are Roman numerals.

Fix a number b, the **basis**, and take b symbols to denote the first b numbers. Here, we interpret zero as the first number, followed by one as the second number and so on. In case b is less than ten, we may use the symbols $0, 1, \ldots, 9$ for the numbers from zero to nine. Every number c has a unique representation of the form

$$c = a_n\,b^n + \cdots + a_2\,b^2 + a_1\,b^1 + a_0\,b^0,$$

where $a_0, a_1, \ldots, a_n \in \{0, 1, \ldots, 9\}$ are the **digits** and $n+1$ is the number of digits required to express the number c with respect to the basis b. With this unique representation at hand, we may write the number c as a list of its digits: $c = a_n\,\ldots\,a_0$. Keep in mind that the basis b has to be known to interpret a list a digits although b often is not written down explicitly.

Example: If we take, for instance, the number twelve, then with ten as base b we would have

$$\text{twelve} = 1 \cdot b^1 + 2 \cdot b^0 = 12.$$

Numbers given in base ten are denoted as **decimal** numbers. More exactly, one should say 'a number in decimal representation' since the number itself does not care about how we write it down.

To avoid confusion, each number we write down without explicitly specifying a basis is to be understood as a decimal number. Numbers in a basis other than ten always will come with some hint on the basis.

5.3.2 Binary Numbers

Numbers in positional representation with base 2 are called **binary** numbers. They frequently appear in computer engineering. Symbols for the two digits are 0, 1 and sometimes the letters O, I.

Example: Number twelve in binary representation is

$$\text{twelve} = 1 \cdot b^3 + 1 \cdot b^2 + 0 \cdot b^1 + 0 \cdot b^0 = 1100 \quad \text{(binary)}.$$

5.3.3 Octal Numbers

Base 8 yields **octal** numbers. For octal numbers the usual digits 0 to 8 can be used.

Example:

$$12 = 1 \cdot 8^1 + 4 \cdot 8^0 = 14 \quad \text{(octal)}.$$

Octal numbers occur for instance in file access permission on Unix-like systems because access is controlled by 3 sets (owner, group, all) of 3 bits (read, write, execute). Thus, all possible combinations can be conveniently expressed by three-digit octal numbers.

Example: Access right 750 (which is 111 101 000 in binary notation) says that the file's owner may read, write and execute the file. The owner's group is not allowed to write to the file (only read and execute). All other users do not have any access right.

5.3.4 Hexadecimal Numbers

Base 16 yields **hexadecimal** numbers. For hexadecimal numbers we use 0 to 9 followed by the symbols a, b, c, d, e, f to denote the digits.

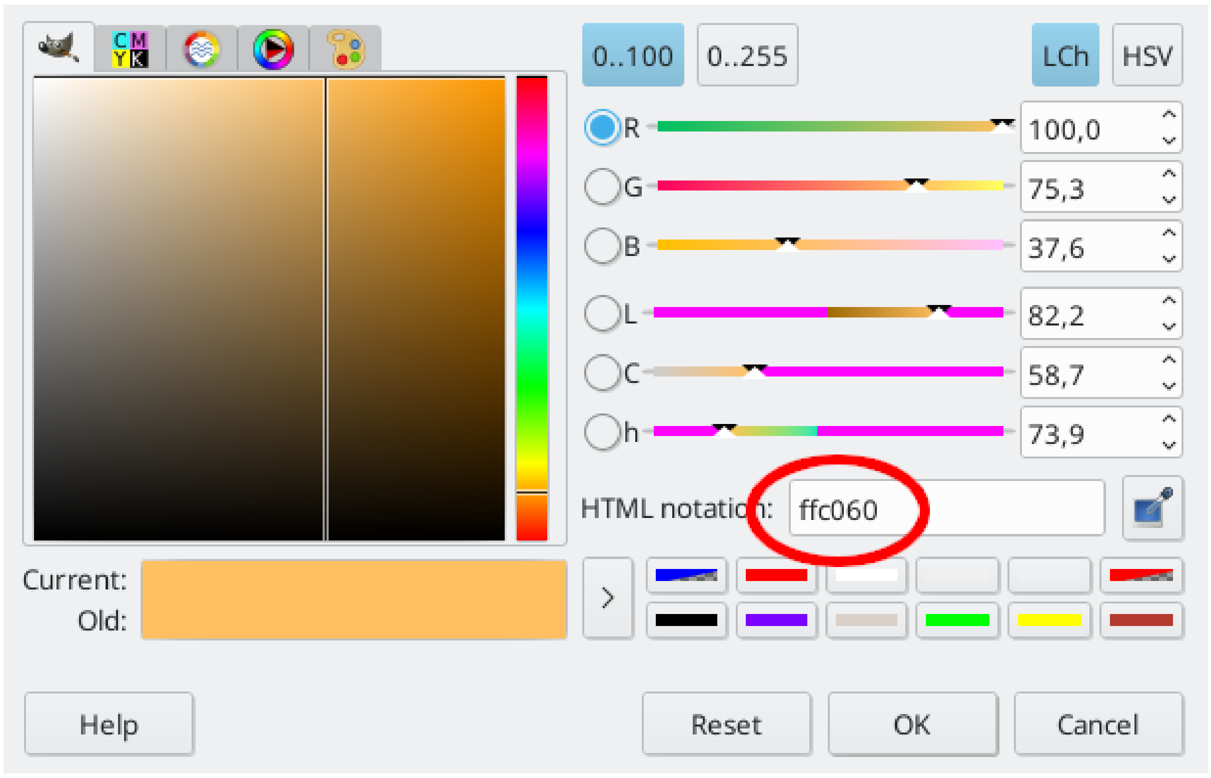
Examples:

$$12 = 12 \cdot 16^0 = \text{c} \quad \text{(hexadecimal)},$$
$$125 = 7 \cdot 16^1 + 13 \cdot 16^0 = \text{7d} \quad \text{(hexadecimal)}.$$

Note that letters a to f might be digits of a hexadecimal number as well as variable names. Have a look at the context to get the correct meaning. Sometimes capital letters A, B, C, D, E, F are used.

Hexadecimal numbers occur in many different situations because the range 0 to 255 of a byte value maps exactly to the set of all two-digit hexadecimal numbers: 00 to ff. We will meet this notation when specifying colors.

Example: The color value ff c0 60 yields a light orange (100% red intensity, 75% green intensity, 38% blue intensity).



R 100,0
G 75,3
B 37,6
L 82,2
C 58,7
h 73,9
HTML notation: ffc060

Fig. 5.4: Professional graphics programs show hexadecimal color values, often denoted as 'HTML notation', because hexadecimal color values frequently occur in HTML[44] code for websites.

5.4 Software and Programming Languages

Software is a stream of binary data to be read and processed by the CPU. The task of a software developer is to generate streams of binary data which make the CPU do what the software developer wants it to do.

Hint: 'Binary data' has at least two different meanings, depending on the context.

- In programming contexts, where we have to distinguish between computer and human readable data, data is considered 'binary' if it has no useful interpretation as text.
- In more general contexts, data is considered 'binary' if it is or can be represented as a sequence of zeros and ones. In this sense, a picture is not binary data, but a digital copy consisting of pixels instead of brushstrokes is binary data.

Modern software has a size of several megabytes or even gigabytes. It isn't impossible for humans to generate such large and complex amounts of binary data by hand. Instead, the process of software development has been automated step by step beginning from scratch with directly coding zeros and ones in the 1950s up to nowadays higher programming languages.

[44] https://en.wikipedia.org/wiki/HTML

5.4.1 Assemblers

A first step of automation has been the invention of **assemblers**. That are computer programs which transform a set of to some extent human readable codewords to a sequence of zeros and ones processable by a CPU. Here is an example:

```
mov 120, eax
mov 124, ebx
add ebx, eax
mov eax, 128
```

The first line tells the CPU to read 4 bytes from memory address 120 and to store them in one of its registers (a kind of internal memory). Second line does the same, but with memory address 124 and a second CPU register. Then the CPU is told to add both values. The CPU stores the result in its `eax` register. The last line makes the CPU write the result of addition to memory address 128.

Writing computer programs in assembler code made software development much easier. But due to the very limited instruction set reflecting one-to-one the instruction set of the CPU, programs are hard to read and tightly bound to the hardware they were designed for. The only advantage of assembler code compared to modern programming languages is its speed of execution and its small size after transforming it to binary code. The first initialization routine of modern operating systems is still written in assembler code, because it has to fit into a small predefined portion of a storage device called boot sector.

5.4.2 Structured Programming

A further step in the evolution of programming languages are languages for *structured programming*. Examples are **C**, **BASIC**, **Pascal**. Here the hardware is almost completely abstracted and a relatively complex program, the *compiler*, is needed to transform the source code written by the software developer to binary code for the CPU. Here is a snipped of a C program:

```
int a, b;
a = 5;
b = 10 * a + 7;
printf("result is %i", b);
```

The first line tells the compiler that we need two places in memory for storing integer values. The second line makes the CPU move the value 5 to the place in memory referenced by `a`. Third line makes the CPU do some calculations and store the result in memory referenced by `b`. Finally, the result shall be printed on screen. Writing this in assembler code would require some hundred lines of code and we would have to take care of memory organization (where is free space?) and of the instruction set of the CPU. Both is done by the compiler. Especially the C language is still of great importance. It is used, for example, for large parts of Linux and Windows.

5.4.3 Object-Oriented Programming

A further layer of abstraction is *object-oriented programming*. Instead of handling hundreds of variables and hundreds of functions for their processing, everything is organized in a well structured way reflecting the structure of the real world. Examples of programming languages allowing for object-oriented programming are **C++**, **Java**, **Python**.

5.4.4 Compiler vs. Interpreter

Source code of a computer program either is *compiled* or *interpreted*. Compiling means that the source code is translated to binary code and after finishing this translation it can be executed, that is, fed to the CPU. Interpretation means that the source code is translated line by line and each translated line is sent immediately to the processor.

Compiled programs run much faster than interpreted ones. But interpreted programs allow for simpler debugging and more intuitive elements in the programming language. Sometimes interpreted programs are called scripts and corresponding languages are denoted as scripting languages. C and C++ are compiled languages whereas Python is interpreted. Java is somewhere in between.

Part III

Python for Data Science

CHAPTER

SIX

CRASH COURSE

This chapter provides an overview of everyday Python features and their basic usage.

Related exercises:

- *Finding Errors* (page 851)
- *Basics* (page 857)
- *Simple List Algorithms* (page 915)
- *More Basics* (page 859)
- *Geometric Objects* (page 918)

6.1 A First Python Program

We start the Python crash course with a small program. The program will ask the user for some keyboard input. If the user types `bye` the program stops, else it asks again.

6.1.1 Source Code

```
code = None

while code != "bye":

    print("I'm a Python program. Type 'bye' to stop me:")
    code = input()     # get some input from user

    if code == "":
        print("To lazy to type?")
    print("")

print("Bye")
```

6.1.2 Line by Line

The first line

```
code = None
```

provides space in memory to store something (this is quite unprecise, but will be made precise soon). We name this piece of memory `code` since we want to store a code typed by the user. At the moment there is nothing to store, which is expressed by `None`.

The second line

```
while code != "bye":
```

is the head of a loop. The subsequent indented code block is executed again and again as long as the condition `code != "bye"` is satisfied. Here, `!=` means unequal. Thus the line

```
print("Bye")
```

is not executed before the variable `code` holds the value `bye`.

```
   print("I'm a Python program. Type 'bye' to stop me:")
```

prints a message on screen and

```
    code = input()     # get some input from user
```

waits for user input, which then is stored in the variable `code`. Text following a `#` symbol is ignored by the Python interpreter.

```
    if code == "":
```

checks whether the variable `code` contains an empty string. If so,

```
        print("To lazy to type?")
```

is executed. Else, execution continues printing a blank line:

```
    print("")
```

The line

```
print("Bye")
```

is only executed if the user types `bye`. Then, after executing this, the program stops. There is no other way for the user to stop the program (next to killing it with the operating system's help).

6.1.3 Some More Details

In our first Python script we can make several important observations:

- The symbol = does not mean that the stuff on its left-hand side is the same as on its right-hand side. Instead, the value on the right-hand side is assigned to the variable on the left-hand side. The process of assignment will be made more precise later on.
- Strings are surrounded by quotation marks. Again, details will follow.

- Contrary to most other programming languages, code indentation matters. Subblocks of code have to be indented to be recognized as such by the Python interpreter. Usual indentation width is 4 spaces, but other widths and also tabs may be used as long as indentation is consistent throughout the file.
- There is something we call a function. Functions in our script are `print` and `input`. We can pass arguments to functions to influence their behavior (what to print?) and functions may return some value. The latter is the case for the `input` function. The return value is stored in the variable `code`. Note, that even if we don't want to pass arguments to a function, we have to write parentheses: `input()`.

6.1.4 Errors

Programming is a very error-prone sport. There are two types of errors:

Syntax Errors

If the Python interpreter does not understand our code, we say that the code contains a syntax error. The interpreter will complain about a syntax error and stop program execution.

Semantic Errors

If the Python interpreter understands our code, but the program behaves differently from what we expected it to do, the code contains semantic errors. Some types of semantic errors, like division by zero, are detected by the Python interpreter. Other types won't be detected automatically. If the interpreter sees a semantic error, program execution stops with some hint on the problem.

The following code contains a syntax error: missing `)` in the first call to `print`.

```
code = None

while code != "bye":

    print("I'm a Python program. Type 'bye' to stop me:"
    code = input()    # get some input from user

    if code == "":
        print("To lazy to type?")
    print("")

print("Bye")
```

```
  Input In [2]
    print("I'm a Python program. Type 'bye' to stop me:"
         ^
SyntaxError: '(' was never closed
```

The next code example contains a semantic error not detectable by the interpreter: instead of == there is `!=` in the `if` statement, which checks for inequality instead of equality. Thus, the program will print the `To lazy to type?` message if the user has typed something. No message will appear if the user has provided no input. That's not what we want the program to do.

```
code = None

while code != "bye":

    print("I'm a Python program. Type 'bye' to stop me:")
    code = input()    # get some input from user

    if code != "":
        print("To lazy to type?")
    print("")
```

(continues on next page)

(continued from previous page)

```
print("Bye")
```

Important: Writing code is not too hard. But **writing code without errors is almost impossible**. Finding and correcting errors is an essential task and will consume considerable resources (time and nerves). Some advice for finding errors:

- Always read and understand the error message (if there is one).
- If you do not understand an error message, ask a search engine for explanation.
- Line numbers shown in an error message often are correct, but sometimes the erroneous code is located above (not below) the indicated position.
- Test run your code as often as possible. Write some lines, then test these lines. The less code you write between tests, the more obvious is the reason for an error.
- For each error there is a solution. You just have to find it.
- The Python interpreter is always right! If it says, that there is an error, then THERE IS AN ERROR.

6.2 Building Blocks

We start our quick run through Python with essential features which can be found in almost every high-level programming language. What we will meet here is known as *structured programming*. Later on we will move on to *object oriented programming*.

6.2.1 Comments

A Python source code file may contain text ignored by the Python interpreter. Such *comments* help to understand and document the source code. Everything following a # symbol is ignored by the interpreter.

```
a = 1    # here we could place some explanation

# this whole line is a comment and completely ignored by the interpreter
b = 2
```

For the Python interpreter the above code is equivalent to

```
a = 1
b = 2
```

Note that empty lines do not matter. We may place empty lines everywhere like comments to make the code more readable.

6.2.2 Assignments

Data (numbers, strings and other) in memory can be associated to a human readable string. Such a combination of a piece of data and a name for it is known as *variable*. To assign a name to a piece of data Python uses the = sign.

```
a = 1
```

The above code writes the number 1 to some location in memory and assigns the name `a` to it. Whenever we want to use or modify this value we simply have to provide its name `a`. The Python interpreter translates the name into a memory address.

```
print(a)
```

```
1
```

This prints the value of the variable `a` to screen.

6.2.3 Simple Data Types

We have to distiguish different types of data because each type comes with its own set of operations. Numbers can be added and multiplied, for example, whereas strings can be concatenated but not multiplied. In Python we do not have to care too much about choosing the correct data type, because the interpreter does much of the technical stuff (e.g., how much memory is required?) for us.

```
a = 2           # an integer
b = 2.1         # a floating point number
c = "Hello!"    # a string
d = True        # a boolean value

print(a)
print(b)
print(c)
print(d)
```

```
2
2.1
Hello!
True
```

Integers

Integers are the numbers …, -2, -1, 0, 1, 2,….

Note: In most programming languages there is a maxmimum value an integer can attain, like -2^{31}, …, $2^{31}+1$. In Python there is no limit on the size of an integer.

```
a = 5 + 2     # addition
b = 5 - 2     # substraction
c = 5 * 2     # multiplication
d = 5 // 2    # floor division
e = 5 % 2     # remainder of devision
f = 5 / 2     # division (yields a floating point number)
g = 2 ** 5    # power
```

(continues on next page)

(continued from previous page)

```
print(a)
print(b)
print(c)
print(d)
print(e)
print(f)
print(g)
```

```
7
3
10
2
1
2.5
32
```

Undefined operations will be identified by the interpreter.

```
a = 1 // 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [7], in <cell line: 1>()
----> 1 a = 1 // 0

ZeroDivisionError: integer division or modulo by zero
```

If we want the user to input an integer, we may use the following code

```
a = int(input("Give me an integer: "))
print(a)
```

Like `print` and `input` also `int` is a function. It takes a string and converts it to an integer. If this is not possible, an error message appears and program execution is stopped.

Floating Point Numbers

Python supports floating point numbers (also known as *floats*) in the approximate range 1e-308…1e+308 with 15 significant decimal places (*double precision* in IEEE 754 standard[45]). Floating point numbers are stored as a pair of coefficient and exponent of 2, where both coefficient and exponent are integers.

Example: $0.1875 = 3 \cdot 2^{-4}$ with coefficient 3 and exponent -4.

Important: Most decimal fractions cannot be represented exactly as float, which may cause tiny errors in computations.

Example:

$$\begin{aligned} 0.1 &\approx 3602879701896397 \cdot 2^{-55} \\ &= 0.1000000000000000055511151231257827021181583404541015625 \end{aligned}$$

See Python documentation[46] for more detailed explanation and additional examples.

[45] https://en.wikipedia.org/wiki/IEEE_754
[46] https://docs.python.org/3/tutorial/floatingpoint.html

```
a = 5        # integer (stored as is)
b = 5.0      # float (stored as coefficient and exponent)
c = 5.123    # float
d = c + 2    # float plus integer yields float

print(a)
print(b)
print(c)
print(d)
```

```
5
5.0
5.123
7.123
```

Note that Python converts data types automatically as needed. Destination type is chosen to prevent loss of data as far as possible (cf. line 4 in the code example above). If conversion is not possible, the interpreter will complain about.

Strings

In Python strings are as simple as numbers. Just enclose some characters in single or double quotation marks and they will become a Python string.

```
a = 'Hello'      # single quotation marks
b = 'my'
c = "friend!"    # double quotation marks

# strings may be concatenated using +
d = a + ' ' + b + ' ' + c

print(d)
```

```
Hello my friend!
```

Behavior of operators like + depends on the data type of the operands. Adding two integers `123 + 456` yields the integer `579`. Adding two strings `'123' + '456'` yields the string `'123456'`.

If a string contains single quotation marks, then use double quotation marks and vice versa. Alternatively, you may *escape* quotation marks in a string with a backslash.

```
a = "He isn't cool."
b = 'He isn\'t cool.'
c = 'He said: "Your are crazy"'
d = "He said: \"Your are crazy\""

print(a)
print(b)
print(c)
print(d)
```

```
He isn't cool.
He isn't cool.
He said: "Your are crazy"
He said: "Your are crazy"
```

Boolean Values

Boolean values or truth values can hold either `True` or `False`. Typically, they are the result of comparisons. Boolean values support logical operations like `and`, `or`, and `not` (see *Logic* (page 1017)).

```
a = True
b = a and False
c = not a
d = a or b

print(a)
print(b)
print(c)
print(d)
```

```
True
False
False
True
```

6.2.4 Functions

A function is a piece of Python code with a name. To execute the code we have to write its name, optionally followed by parameters (sometimes denoted as *arguments*) influencing the function's code execution. After executing the function some value can be returned to the caller.

This concept is required in two circumstances:

- a piece of code is needed several times,
- readability shall be increased by hiding some code.

Built-in Functions

Python has several *built-in functions*, like `print` and `input`. The `print` function takes one or more variables and prints them on screen. In case of multiple arguments outputs are separated by spaces. The `input` function may be called without arguments. It waits for user input and returns the input to the calling code.

```
a = input()
print('You typed:', a)
```

Above we also met the `int` function, which converts a string to an integer if possible. The `int` function behaves exactly in the same way as all other functions, but it is not a built-in function in the stricter sense. Instead, it's the constructor of a class, a concept we'll discuss later on.

Keyword Arguments

Functions accept different kinds of arguments. Some are passed as they are (like for `print`). Those are called *positional arguments* and we meet them in almost all programming languages.

In Python often we'll see function calls of the form `some_function(argument_name=passed_value)`. Such arguments are called *keyword arguments* and help to increase code readability. If a function accepts multiple keyword arguments, we do not have to care about which one to pass first, second and so on. Details will be discussed in a separate chapter on functions later on.

Function Definitions

We can define new functions with the `def` keyword.

```python
def say_hello(name):
    ''' Print a hello message. '''

    message = 'Hello ' + name + '!'
    print(message)


say_hello('John')
say_hello('Anna')
```

```
Hello John!
Hello Anna!
```

Note the indentation of the function's code and the *docstring* `'''...'''`. The indentation tells the Python interpreter which lines of code belong to the function. The docstring is ignored by the interpreter like a comment. But tools for automatic generation of software documentation extract the docstring and process it.

To return a value (like `input` does) we would have to add a line containing `return my_value`. The `return` keyword stops execution of the function and returns control to the calling code. We place `return` wherever appropriate for our purposes. Often, but not always, it's in the last line of the function's code.

Important: Variables introduced inside a function, like `message` above, are only accessible inside that function. But variables defined outside a function are accessible inside functions, too. It's considered good practice to keep inside and outside variables separated. That is, don't use outside variables inside a function. Instead pass all values required by the function as arguments and return results required outside a function with `return`. Exceptions prove the rule.

Errors in Functions

If there is an error in a function's code, the Python interpreter will show an error message together with a *traceback*. That's a list of code lines leading to the erroneous line. If a program calls a function which again calls a function which contains an error, the traceback will have three entries.

In the following example the variable name is incorrect in the `print` line.

```python
def say_hello(name):
    ''' Print a hello message. '''

    message = 'Hello ' + name + '!'
    print(mesage)


say_hello('John')
say_hello('Anna')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [15], in <cell line: 8>()
      4     message = 'Hello ' + name + '!'
      5     print(mesage)
----> 8 say_hello('John')
      9 say_hello('Anna')
```

(continues on next page)

(continued from previous page)

```
Input In [15], in say_hello(name)
      2 ''' Print a hello message. '''
      4 message = 'Hello ' + name + '!'
----> 5 print(mesage)

NameError: name 'mesage' is not defined
```

If there would be an error in `print` (because you passed an unexpected argument, for instance), then the traceback would have an additional entry showing the erroneous line in the definition of `print`.

Hint: Tracebacks may become very long if your code implies a problem in some built-in or library function. Check the traceback carefully to find the last entry referring to your code. That's the most likely location of the problem's cause.

6.2.5 Conditional Execution

Up to now program flow is linear. There is one path and the interpreter will follow this path. Here comes the first element of flow control: conditional execution.

```
a = int(input('Give me a number: '))
b = int(input('Give me another number: '))

if a > b:
    print('First number is greater.')
else:
    print('First number is not greater.')
```

If the condition is satisfied, then the first code block is executed. If it is not satisfied, the `else` block is executed. For equality use ==, for inequality use `!=`. Other comparison operators are <, >, <=, >=.

A comparison evaluates to a boolean value. Thus, more complex conditions can be constructed with the help of boolean operators.

```
a = int(input('Choose a number from 1 to 10: '))

if (a >= 1) and (a <= 10):
    print('Well done!')
else:
    print('You still have to learn a lot...')
```

The `else` part can be omitted, if nothing is to be done.

If more than two cases (`True` and `False`) have to be distinguished, use `elif`, which stands for 'else if':

```
a = int(input('Give me an integer: '))

if a < 0:
    print('It\'s a negative number.')
elif a == 0:
    print('It\'s zero.')
elif a < 10:
    print('It\'s a small positive number.')
else:
    print('It\'s a large positive number.')
```

6.2.6 Repeated Execution

The second element of flow control, next to conditional execution, is repeated execution. Python provides two techniques: *while loops* and *for loops*.

For Loops

A for loop repeats a code block a pre-specified number of times.

```python
for k in range(1, 10):
    print(k * k)
```

```
1
4
9
16
25
36
49
64
81
```

Note that 100 is not printed. The loop always stops before the final number is reached.

Note: Whenever you have to define a range of integers in Python the upper bound has to be the last value you need plus 1. If you already tried some other programming language, this peculiarity of Python needs getting used to.

While Loops

A while loop repeats a code block as long as a condition is met.

```python
my_number = 10

users_number = int(input('Guess my number: '))

while users_number != my_number:
    if users_number < my_number:
        print('Too small!')
    else:
        print('Too large!')
    users_number = int(input('One more chance: '))

print('Correct!')
```

No Do-While Loops

Many programming languages have a so called do-while loop. That's like a while loop, but the condition is checked at the loop's end. Thus, the loop's code block is executed at least once. Python does not have a while loop.

Guido van Rossum, Python's BDFL[47], rejected a Python enhancement proposal (PEP) which suggested to introduce a do-while loop[48] with the following words[49]:

[47] https://en.wikipedia.org/wiki/Benevolent_dictator_for_life
[48] https://peps.python.org/pep-0315/
[49] https://mail.python.org/pipermail/python-ideas/2013-June/021610.html

> Please reject the PEP. More variations along these lines won't make the language more elegant or easier to learn. They'd just save a few hasty folks some typing while making others who have to read/maintain their code wonder what it means.

Controlling Loop Execution

For and while loops provide the keywords `break` and `continue`. With `break` we can abort execution of the loop. With `continue` we can stop execution of the loop's code block and immediately begin the next iteration.

Loops may have an `else` code block. The `else` block is executed if iteration terminates regularly. It is skipped, if iteration is stopped by `break`.

```
for k in range(1, 10):
    print(k)
    a = input('Do you want to see the next number (y/n)?')
    if a == 'n':
        break
else:
    print('Now you\'ve seen all my numbers.')
print('Good bye!')
```

Note: Whereas for and while loops are available in almost all programming languages, the `else` block is a special feature of Python.

6.2.7 Lists

Next to the simple data types above there are more complex ones. Here we restrict our attention to lists. A list can hold a number of values. The length of a list is returned by the built-in function `len`. Square brackets `[` and `]` are used for defining a list and for accessing single elements of a list.

```
a = [2, -5, 4, 3, 2, -10, 3, 4]

print('List:', a)

print('Length of list:', len(a))

print('First element:', a[0])

print('Second element:', a[1])

print('Fifth element:', a[4])

print('Last element:', a[len(a) - 1])
```

```
List: [2, -5, 4, 3, 2, -10, 3, 4]
Length of list: 8
First element: 2
Second element: -5
Fifth element: 2
Last element: 4
```

List indices start with 0 in Python. Consequently, the last element of an n-element list has index n-1. The above code to access the last element is considered *non-pythonic*. Why this is the case and how to make it better will be discussed later on.

Lists may contain arbitrary types of data. Even lists of lists are allowed. This way we can construct two-dimensional data structures.

```
a = [[3, 4, 5], [-3, 7, 2], [4, 7, 5]]

print(a[0])

print(a[0][0], a[2][0])
```

```
[3, 4, 5]
3 4
```

Items of a list can be modified after creation of the list:

```
a = [3, 4, 5]
print(a)

a[1] = 1000
print(a)
```

```
[3, 4, 5]
[3, 1000, 5]
```

How to append elements to an existing list and many more list related topics will be discussed later on.

Note: A list may have length 0, that is, it may be empty. Empty lists occur frequently because often one wants to fill lists item by item, starting with an empty list. To get an empty list in Python write `[]`.

6.2.8 Make a Building from Building Blocks

With the above building blocks at hand we may write arbitrarily complex programs. There is nothing more we need. It's like Lego blocks[50]. Take lots of simple blocks, add some creativity, and think about how to reach your aim step by step.

All the other features of Python we'll discuss soon only exist to simplify programming, save some time and make programs more readable. But they won't add new possibilities.

To be correct: there are a small number of additional built-in functions we need to know, like `open` for accessing files.

Building everything from scratch is a long and winding road. So we'll use other people's code and combine it to new and larger projects. There's a large library of ready-to-use code snippets, called the Python standard library[51]. For specific tasks like data science and AI there are specialized libraries containg thousands of functions we may use without implementing them ourselves. Examples are Matplotlib[52], Pandas[53], Scikit-Learn[54] and Tensorflow[55].

[50] https://en.wikipedia.org/wiki/Lego
[51] https://docs.python.org/3/library/
[52] https://matplotlib.org/
[53] https://pandas.pydata.org/
[54] https://scikit-learn.org/stable/
[55] https://www.tensorflow.org/

6.3 Screen IO

Input from keyboard and output to screen are important for a program's interaction with the user. Here we discuss some frequently used features of terminal and Jupyter based screen IO. Graphical user interfaces (GUIs) are possible in Python, too, but won't be discussed here.

6.3.1 Input

The `input` function does not provide more functionality than we have already seen. It takes one argument, the text to be printed on screen, and returns a string with the user's input.

6.3.2 Output

The `print` function provides some fine-tuning. We already saw that it takes an arbitrary number of arguments. Each argument which is not a keyword argument will be send to screen. Outputs of multiple arguments are separated by one space. Non-string arguments are converted to strings automatically.

Instead of spaces, different separators can be specified with the keyword argument `sep='...'`.

```
a = 3
b = 2.34
c = 'some_string'

print(a, b, c, sep=' | ')
```

```
3 | 2.34 | some_string
```

Note: Note the difference to `print(a, b, c, ' | ')`, which yields `3 2.34 some_string |` .

The `print` function automatically adds a line break to the output. If this is not desired we may pass something else as keyword argument `end='...'`, an empty string for instance.

```
print('some text', end='')
print(' and some more text')
```

```
some text and some more text
```

Line breaks may be added wherever appropriate by writing `\n` in a string.

```
print('some text with line break\nin a string')
```

```
some text with line break
in a string
```

Note: The \ character is not printed but interpreted as `escape charater`. The character following \ is a command to the output algorithm. Next to `\n` we already met `\'` and `\"`. If you have to print a \ on screen use `\\`.

Calling `print` without any arguments prints a line break. Thus, `print()` and `print('')` are equivalent.

6.3.3 Automatic Ouput in Interactive Python

In JupyterLab and also in the Python interpreter's interactive mode we do not have to write `print(...)` everytime we want to see some result. JupyterLab automatically prints the value of the last line in a code cell. The Python interpreter automatically prints the value of the last command issued. Instead of

```
print(123 * 456)
```

```
56088
```

we may simply write

```
123 * 456
```

```
56088
```

While plain Python calls `print` on the value to output, JupyterLab calls it's own function `display`. For simple data like numbers and strings `display` does the same as `print`, but for complex data like tables or images `display` produces richer output. Even audio files and videos may be embedded into Jupyter notebooks by calling `display` on suitably prepared data (or leaving this to JupyterLab if data is produced in the last line of a cell).

6.4 Library Code

Source code libraries contain reusable code. In Python reusing code written by other people is very simple and there are lots of code libraries available for free. Code libraries for Python are organized in *modules* and *packages*.

6.4.1 Python Modules

Next to built-in functions like `print` and `input` Python ships with several *modules*, which can be loaded on demand. A module is a collection of additional functionality. Everybody can write and publish Python modules. How to do this will be explained later on. Modules either are written in Python or in some other laguage, mainly the C programming language.

Before we can use functionality of a module we have to import it:

```
import numpy as np
```

Hint: A number of modules comes pre-installed with Python (the Python standard library[56]). But many others have to be installed separately. Whenever Python shows `ModuleNotFoundError` you forgot to install the module you want to import. Install a module via Anaconda Navigator or with `conda install module_name` in a terminal, see *Install Jupyter Locally* (page 905) project for more details.

The code above imports the module `numpy` and makes it accessible under the name `np`, which is shorter than 'numpy'. NumPy[57] is a collection of functions and data types for advanced numerical computations. We will dive deeper into this module later on. To use NumPy's functionality we have to write `np.some_function` with `some_function` replaced by one of NumPy's functions.

```
np.sin(0.25 * np.pi)
```

[56] https://docs.python.org/3/library/
[57] https://numpy.org

```
0.7071067811865475
```

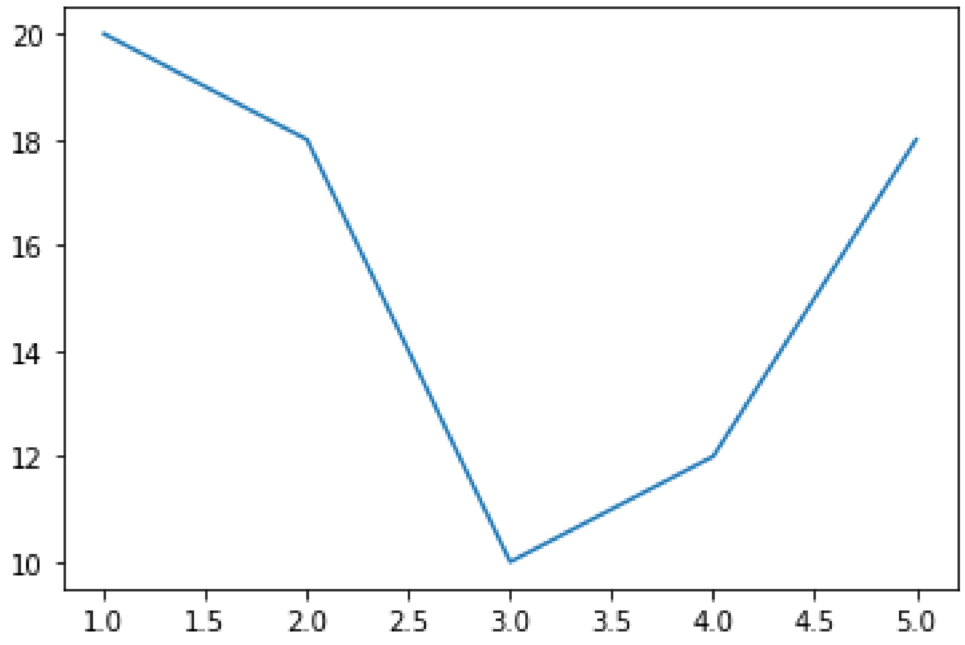
Here, `np.pi` is a floating point variable holding an approximation of π. The function `np.sin` computes the sine of its argument.

Note: The name `np` for accessing functionality of the module `numpy` can be choosen freely. But for widely used modules like NumPy there are standard names everybody should use to improve code readability. Names everybody should use are given in a module's documentation (look at code examples there). Keep in mind: that's a convention, `import numpy as wild_cat` would by okay, too, from the technical point of view.

6.4.2 Python Packages

A package is a collection of modules. A module from a package can be imported via `import package.module`. A very important Python package is Matplotlib[58] for scientific plotting:

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4, 5], [20, 18, 10, 12, 18])
plt.show()
```



With `plt.plot` we create a line plot and `plt.show` displays this plot. If the code runs in JupyterLab the plot is embedded into the notebook. If run by a plain Python interpreter a window opens showing the plot.

We will come back to Matplotlib when discussing data visualization.

[58] https://matplotlib.org/

6.5 Everything is an Object

The Python programming language follows some basic design principles, which make the language very clear and in some sense esthetic. One such principle is '**Everything is an object**'.

A Python object is a collection of variables and functions. We may use objects to bring structure into a program's variables and functions. Each object is of a certain type. An object's type specifies a minimum list of variables and functions an object contains or provides. Variables and functions belonging to an object are called *member variables* and *member functions*. A synonym for member function is *method*.

6.5.1 Object-Oriented Programming

Large programs have lots of variables and functions. By object-oriented programming we denote an approach to structure the morass of variables and functions into objects. Python supports this programming style.

For the moment we look at objects as containers for structuring source code. Later on we will discuss more advanced features of object-oriented programming like defining hierarchies of types (to untangle the morass of types…).

Example

Think of a Python program which does some geometrical computations and then plots a line and a circle. To specify geometric properties we could use variables

- `line_start_x`,
- `line_start_y`,
- `line_end_x`,
- `line_end_y`,
- `circle_x`,
- `circle_y`,
- `circle_radius`.

In addition, we could have functions

- `draw_line`,
- `draw_circle`,
- `move_line`,
- `move_circle`,
- `rotate_line`,

and so on.

Utilizing the idea of objects we could introduce objects of type `Point` with member variables `x` and `y` as well as an object of type `Line` with member variables `start` and `end` both of type `Point`. Analogously, an object of type `Circle` with member variables `center` (of type `Point`) and `radius` would be nice. Both the `Line` object and the `Circle` object could have member functions `draw` and `move`.

The object hierarchy could look like this:

- `my_line`
  - `start`
    - `x`
    - `y`
  - `end`

        * x
        * y
    - draw()
    - move(...)
    - rotate(...)
- my_circle
    - center
        * x
        * y
    - radius
    - draw()
    - move(...)

Objects Everywhere

Surprisingly, in Python there are no variables which are not an object. Even integers are objects. Most other object-oriented programming languages have some fundamental data types (integers, floats, characters) not represented as objects in the sense of object-oriented programming. In Python there are no such fundamental types!

6.5.2 Accessing an Object's Members

To get a list of all members (variables and functions) of an object, Python has the built-in function `dir`.

```
a = 2

dir(a)
```

```
['__abs__',
 '__add__',
 '__and__',
 '__bool__',
 '__ceil__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__float__',
 '__floor__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__index__',
 '__init__',
 '__init_subclass__',
 '__int__',
 '__invert__',
```

(continues on next page)

(continued from previous page)

```
 '__le__',
 '__lshift__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__neg__',
 '__new__',
 '__or__',
 '__pos__',
 '__pow__',
 '__radd__',
 '__rand__',
 '__rdivmod__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rfloordiv__',
 '__rlshift__',
 '__rmod__',
 '__rmul__',
 '__ror__',
 '__round__',
 '__rpow__',
 '__rrshift__',
 '__rshift__',
 '__rsub__',
 '__rtruediv__',
 '__rxor__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__sub__',
 '__subclasshook__',
 '__truediv__',
 '__trunc__',
 '__xor__',
 'as_integer_ratio',
 'bit_count',
 'bit_length',
 'conjugate',
 'denominator',
 'from_bytes',
 'imag',
 'numerator',
 'real',
 'to_bytes']
```

Note: The `dir` function returns a list of strings. Since it's the last line of a cell, this list is printed to screen by JupyterLab automatically.

Most of all these members won't be used directly. The `__add__` function, for instance, is called by the Python interpreter whenever we add integers. The following line of code is equivalent to `a + 3`:

```
a.__add__(3)
```

```
5
```

The notation `object.member` is the syntax to access members of an object.

6.5.3 Getting an Object's Type

Each object has a type. Objects of identical type provide identical functionality. An object's type is returned by Python's built-in function `type`.

```
type(a)
```

```
int
```

Note: As for `dir` above, the `type` function does not print anything. Screen output is done by JupyterLab automatically here. Note that Python's `print` yields `<class 'int'>` here whereas JupyterLab's `display` produces `int` to visualize `type`'s return value.

6.5.4 Really Everything is an Object

Note that the dot syntax `object.member` already appeared when accessing modules: `module.function`. This is not by chance. Everything is an object in Python!

```
import numpy as np

type(np)
```

```
module
```

Importing a module leads to a new object of type `module` providing all the functionality of the imported module in form of member functions and variables.

```
import numpy as np

dir(np)
```

```
['ALLOW_THREADS',
 'AxisError',
 'BUFSIZE',
 'CLIP',
 'ComplexWarning',
 'DataSource',
 'ERR_CALL',
 'ERR_DEFAULT',
 'ERR_IGNORE',
 'ERR_LOG',
 'ERR_PRINT',
 'ERR_RAISE',
 'ERR_WARN',
 'FLOATING_POINT_SUPPORT',
 'FPE_DIVIDEBYZERO',
 'FPE_INVALID',
 'FPE_OVERFLOW',
 'FPE_UNDERFLOW',
 'False_',
 'Inf',
 'Infinity',
```

(continues on next page)

(continued from previous page)

```
'MAXDIMS',
'MAY_SHARE_BOUNDS',
'MAY_SHARE_EXACT',
'ModuleDeprecationWarning',
'NAN',
'NINF',
'NZERO',
'NaN',
'PINF',
'PZERO',
'RAISE',
'RankWarning',
'SHIFT_DIVIDEBYZERO',
'SHIFT_INVALID',
'SHIFT_OVERFLOW',
'SHIFT_UNDERFLOW',
'ScalarType',
'Tester',
'TooHardError',
'True_',
'UFUNC_BUFSIZE_DEFAULT',
'UFUNC_PYVALS_NAME',
'VisibleDeprecationWarning',
'WRAP',
'_CopyMode',
'_NoValue',
'_UFUNC_API',
'__NUMPY_SETUP__',
'__all__',
'__builtins__',
'__cached__',
'__config__',
'__deprecated_attrs__',
'__dir__',
'__doc__',
'__expired_functions__',
'__file__',
'__getattr__',
'__git_version__',
'__loader__',
'__name__',
'__package__',
'__path__',
'__spec__',
'__version__',
'_add_newdoc_ufunc',
'_distributor_init',
'_financial_names',
'_from_dlpack',
'_globals',
'_mat',
'_pytesttester',
'_version',
'abs',
'absolute',
'add',
'add_docstring',
'add_newdoc',
'add_newdoc_ufunc',
'alen',
'all',
```

(continues on next page)

(continued from previous page)

```
 'allclose',
 'alltrue',
 'amax',
 'amin',
 'angle',
 'any',
 'append',
 'apply_along_axis',
 'apply_over_axes',
 'arange',
 'arccos',
 'arccosh',
 'arcsin',
 'arcsinh',
 'arctan',
 'arctan2',
 'arctanh',
 'argmax',
 'argmin',
 'argpartition',
 'argsort',
 'argwhere',
 'around',
 'array',
 'array2string',
 'array_equal',
 'array_equiv',
 'array_repr',
 'array_split',
 'array_str',
 'asanyarray',
 'asarray',
 'asarray_chkfinite',
 'ascontiguousarray',
 'asfarray',
 'asfortranarray',
 'asmatrix',
 'asscalar',
 'atleast_1d',
 'atleast_2d',
 'atleast_3d',
 'average',
 'bartlett',
 'base_repr',
 'binary_repr',
 'bincount',
 'bitwise_and',
 'bitwise_not',
 'bitwise_or',
 'bitwise_xor',
 'blackman',
 'block',
 'bmat',
 'bool8',
 'bool_',
 'broadcast',
 'broadcast_arrays',
 'broadcast_shapes',
 'broadcast_to',
 'busday_count',
 'busday_offset',
```

(continues on next page)

(continued from previous page)

```
 'busdaycalendar',
 'byte',
 'byte_bounds',
 'bytes0',
 'bytes_',
 'c_',
 'can_cast',
 'cast',
 'cbrt',
 'cdouble',
 'ceil',
 'cfloat',
 'char',
 'character',
 'chararray',
 'choose',
 'clip',
 'clongdouble',
 'clongfloat',
 'column_stack',
 'common_type',
 'compare_chararrays',
 'compat',
 'complex128',
 'complex256',
 'complex64',
 'complex_',
 'complexfloating',
 'compress',
 'concatenate',
 'conj',
 'conjugate',
 'convolve',
 'copy',
 'copysign',
 'copyto',
 'core',
 'corrcoef',
 'correlate',
 'cos',
 'cosh',
 'count_nonzero',
 'cov',
 'cross',
 'csingle',
 'ctypeslib',
 'cumprod',
 'cumproduct',
 'cumsum',
 'datetime64',
 'datetime_as_string',
 'datetime_data',
 'deg2rad',
 'degrees',
 'delete',
 'deprecate',
 'deprecate_with_doc',
 'diag',
 'diag_indices',
 'diag_indices_from',
 'diagflat',
```

(continues on next page)

(continued from previous page)

```
 'diagonal',
 'diff',
 'digitize',
 'disp',
 'divide',
 'divmod',
 'dot',
 'double',
 'dsplit',
 'dstack',
 'dtype',
 'e',
 'ediff1d',
 'einsum',
 'einsum_path',
 'emath',
 'empty',
 'empty_like',
 'equal',
 'errstate',
 'euler_gamma',
 'exp',
 'exp2',
 'expand_dims',
 'expm1',
 'extract',
 'eye',
 'fabs',
 'fastCopyAndTranspose',
 'fft',
 'fill_diagonal',
 'find_common_type',
 'finfo',
 'fix',
 'flatiter',
 'flatnonzero',
 'flexible',
 'flip',
 'fliplr',
 'flipud',
 'float128',
 'float16',
 'float32',
 'float64',
 'float_',
 'float_power',
 'floating',
 'floor',
 'floor_divide',
 'fmax',
 'fmin',
 'fmod',
 'format_float_positional',
 'format_float_scientific',
 'format_parser',
 'frexp',
 'frombuffer',
 'fromfile',
 'fromfunction',
 'fromiter',
 'frompyfunc',
```

(continues on next page)

(continued from previous page)

```
 'fromregex',
 'fromstring',
 'full',
 'full_like',
 'gcd',
 'generic',
 'genfromtxt',
 'geomspace',
 'get_array_wrap',
 'get_include',
 'get_printoptions',
 'getbufsize',
 'geterr',
 'geterrcall',
 'geterrobj',
 'gradient',
 'greater',
 'greater_equal',
 'half',
 'hamming',
 'hanning',
 'heaviside',
 'histogram',
 'histogram2d',
 'histogram_bin_edges',
 'histogramdd',
 'hsplit',
 'hstack',
 'hypot',
 'i0',
 'identity',
 'iinfo',
 'imag',
 'in1d',
 'index_exp',
 'indices',
 'inexact',
 'inf',
 'info',
 'infty',
 'inner',
 'insert',
 'int0',
 'int16',
 'int32',
 'int64',
 'int8',
 'int_',
 'intc',
 'integer',
 'interp',
 'intersect1d',
 'intp',
 'invert',
 'is_busday',
 'isclose',
 'iscomplex',
 'iscomplexobj',
 'isfinite',
 'isfortran',
 'isin',
```

(continues on next page)

(continued from previous page)

```
 'isinf',
 'isnan',
 'isnat',
 'isneginf',
 'isposinf',
 'isreal',
 'isrealobj',
 'isscalar',
 'issctype',
 'issubclass_',
 'issubdtype',
 'issubsctype',
 'iterable',
 'ix_',
 'kaiser',
 'kernel_version',
 'kron',
 'lcm',
 'ldexp',
 'left_shift',
 'less',
 'less_equal',
 'lexsort',
 'lib',
 'linalg',
 'linspace',
 'little_endian',
 'load',
 'loadtxt',
 'log',
 'log10',
 'log1p',
 'log2',
 'logaddexp',
 'logaddexp2',
 'logical_and',
 'logical_not',
 'logical_or',
 'logical_xor',
 'logspace',
 'longcomplex',
 'longdouble',
 'longfloat',
 'longlong',
 'lookfor',
 'ma',
 'mask_indices',
 'mat',
 'math',
 'matmul',
 'matrix',
 'matrixlib',
 'max',
 'maximum',
 'maximum_sctype',
 'may_share_memory',
 'mean',
 'median',
 'memmap',
 'meshgrid',
 'mgrid',
```

(continues on next page)

(continued from previous page)

```
 'min',
 'min_scalar_type',
 'minimum',
 'mintypecode',
 'mod',
 'modf',
 'moveaxis',
 'msort',
 'multiply',
 'nan',
 'nan_to_num',
 'nanargmax',
 'nanargmin',
 'nancumprod',
 'nancumsum',
 'nanmax',
 'nanmean',
 'nanmedian',
 'nanmin',
 'nanpercentile',
 'nanprod',
 'nanquantile',
 'nanstd',
 'nansum',
 'nanvar',
 'nbytes',
 'ndarray',
 'ndenumerate',
 'ndim',
 'ndindex',
 'nditer',
 'negative',
 'nested_iters',
 'newaxis',
 'nextafter',
 'nonzero',
 'not_equal',
 'numarray',
 'number',
 'obj2sctype',
 'object0',
 'object_',
 'ogrid',
 'oldnumeric',
 'ones',
 'ones_like',
 'os',
 'outer',
 'packbits',
 'pad',
 'partition',
 'percentile',
 'pi',
 'piecewise',
 'place',
 'poly',
 'poly1d',
 'polyadd',
 'polyder',
 'polydiv',
 'polyfit',
```

(continues on next page)

(continued from previous page)

```
 'polyint',
 'polymul',
 'polynomial',
 'polysub',
 'polyval',
 'positive',
 'power',
 'printoptions',
 'prod',
 'product',
 'promote_types',
 'ptp',
 'put',
 'put_along_axis',
 'putmask',
 'quantile',
 'r_',
 'rad2deg',
 'radians',
 'random',
 'ravel',
 'ravel_multi_index',
 'real',
 'real_if_close',
 'rec',
 'recarray',
 'recfromcsv',
 'recfromtxt',
 'reciprocal',
 'record',
 'remainder',
 'repeat',
 'require',
 'reshape',
 'resize',
 'result_type',
 'right_shift',
 'rint',
 'roll',
 'rollaxis',
 'roots',
 'rot90',
 'round',
 'round_',
 'row_stack',
 's_',
 'safe_eval',
 'save',
 'savetxt',
 'savez',
 'savez_compressed',
 'sctype2char',
 'sctypeDict',
 'sctypes',
 'searchsorted',
 'select',
 'set_numeric_ops',
 'set_printoptions',
 'set_string_function',
 'setbufsize',
 'setdiff1d',
```

(continues on next page)

(continued from previous page)

```
 'seterr',
 'seterrcall',
 'seterrobj',
 'setxor1d',
 'shape',
 'shares_memory',
 'short',
 'show_config',
 'sign',
 'signbit',
 'signedinteger',
 'sin',
 'sinc',
 'single',
 'singlecomplex',
 'sinh',
 'size',
 'sometrue',
 'sort',
 'sort_complex',
 'source',
 'spacing',
 'split',
 'sqrt',
 'square',
 'squeeze',
 'stack',
 'std',
 'str0',
 'str_',
 'string_',
 'subtract',
 'sum',
 'swapaxes',
 'sys',
 'take',
 'take_along_axis',
 'tan',
 'tanh',
 'tensordot',
 'test',
 'testing',
 'tile',
 'timedelta64',
 'trace',
 'tracemalloc_domain',
 'transpose',
 'trapz',
 'tri',
 'tril',
 'tril_indices',
 'tril_indices_from',
 'trim_zeros',
 'triu',
 'triu_indices',
 'triu_indices_from',
 'true_divide',
 'trunc',
 'typecodes',
 'typename',
 'ubyte',
```

(continues on next page)

(continued from previous page)

```
 'ufunc',
 'uint',
 'uint0',
 'uint16',
 'uint32',
 'uint64',
 'uint8',
 'uintc',
 'uintp',
 'ulonglong',
 'unicode_',
 'union1d',
 'unique',
 'unpackbits',
 'unravel_index',
 'unsignedinteger',
 'unwrap',
 'use_hugepage',
 'ushort',
 'vander',
 'var',
 'vdot',
 'vectorize',
 'version',
 'void',
 'void0',
 'vsplit',
 'vstack',
 'warnings',
 'where',
 'who',
 'zeros',
 'zeros_like']
```

If everything is an object in Python, then functions should be objects, too.

```
def my_function():
    print('Here we could do something useful.')

type(my_function)
```

```
function
```

The value returned by `type` should be an object, too.

```
type(type(my_function))
```

```
type
```

Built-in functions have their own type.

```
type(print)
```

```
builtin_function_or_method
```

6.5.5 Custom Object Types

We may define new objects with the `class` keyword. Instead of 'object type' one often says 'class'. To create a class for describing geometric points we could write

```
class Point:

    def __init__(self, x, y):
        self.x = x
        self.y = y
```

This defines a new object type or class `Point`. Here `__init__` is the only member function. The function with this special name is implicitly called by Python whenever a new `Point` object has been created. This initialization function expects the coordinates of the new point and stores them internally as member variables. The `self` argument provides access to the newly created object.

A class is like a blueprint. But the methods defined in the blueprint are called for concrete objects. The `self` argument gives access to the object for which a method has been called. Remember: there is only one class (blueprint), but there may be many objects of this type.

Note: From a technical point of view instead of `self` we may use any name for the first argument (bad practice!). But the first argument of a method always is the object for which the method has been called.

Note: Methods with two leading and two trailing underscores are called *magic methods* or *dunder methods* (dunder = double underscore). They are called by the Python interpreter for special purposes. We already met another dunder method: `__add__`, which is called whenever the `+` operator is used on an object. In contrast to most other programming languages virtually every operation in Python can be customized by implementing a suitable dunder method.

If we write

```
center = Point(3, 7)
```

the Python interpreter creates a new object of type `Point`. At this moment the new object has only one method, `__init__` and no member variables. Immediately after creation Python calls the object's `__init__` function. The arguments passed to `__init__` are the newly created object, 3 and 7.

Now we can work with our object as expected.

```
print(center.x)
center.x = center.x + 2
print(center.x)
```

```
3
5
```

Object info is as follows:

```
type(center)
```

```
__main__.Point
```

```
dir(center)
```

```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'x',
 'y']
```

The reason for `__main__` in the object type will be discussed later on. From the member list we see that there are many automatically created members. Some of them will be discussed later on, too.

CHAPTER

SEVEN

VARIABLES AND OPERATORS

Variables in Python follow a different, much more elegant approach than in other programming languages. In this chapter we discuss the details of variables in Python and of some consequences of Python's approach to variables.

Related exercises:

- *Variables and Operators* (page 861)
- *Memory Management* (page 864)

Related project: *Vector Multiplication* (page 919).

7.1 Names and Objects

In contrast to most other programming languages Python follows a very clean and simple approach to memory access and management. We now dive deeper into Python's internal workings to come up with a new understanding of what we called *variables* in the *Crash Course* (page 53).

7.1.1 Variables in the Non-Pythonian World

Most programming languages, C for instance, assign fixed names to memory locations. Such combinations of memory location and name are known as variables. Assigning a value to a variable then means, that the compiler or interpreter writes the value to the memory location to which the variable's name belongs. There is a one-to-one correspondence between variable names and memory locations.

Consider the following C code:

```c
int a;
int b;
a = 5;
b = a;
```

The first two lines tell the C compiler to reserve memory for two integer variables. The third line writes the value 5 to the location named `a`. The fourth line reads the value at the location named `a` and writes (copies) it to the location named `b`.

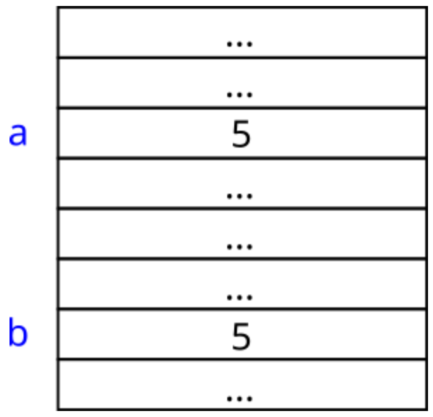


Fig. 7.1: Memory is organized as a linear sequence of bytes. Used and currently unused bytes are managed by the operating system and by compiler. In C programs there is a one-to-one correspondence between variable names and memory locations.

7.1.2 Variables in Python

Python allows for multiple names per memory location and adds a layer of abstraction.

In Python everything is an object and objects are stored somewhere in memory. If we use integers in Python, then the integer value is not written directly to memory. Instead, additional information is added and the resulting more complex data structure is written to memory.

A newly created Python object does not have a name. Instead, Python internally assigns a unique number to each object, the object identifier or *object ID* for short. Thus, there is a one-to-one correspondence between object IDs and memory locations.

In addition to a list of all object IDs (and corresponding memory locations), Python maintains a list of names occuring in the source code. Each name refers to exactly one object. But different names may refer to the same object. In this sense Python does not know variables as described above, but only objects and names tied to objects.

Consider the following code:

```
a = 5
b = a
```

The first line creates an integer object containing the value 5 and then ties the name `a` to this object. The second line takes the object referenced by the name `a` and ties a second name `b` to it.

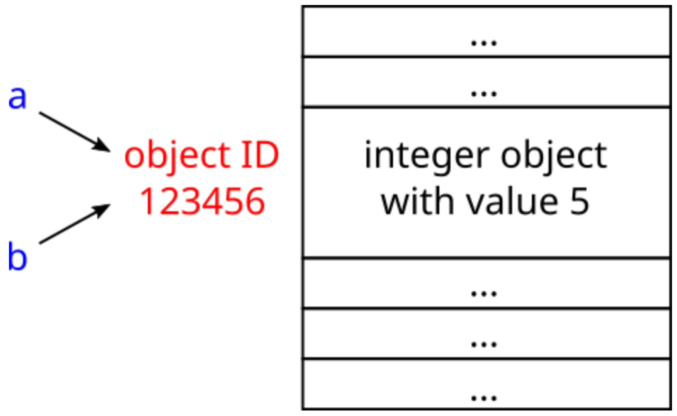


Fig. 7.2: In Python one memory location may have several names, but a unique object ID.

Important: Assignment operation = in Python is not about writing something to memory. Instead, Python takes the **existing** object on the right-hand side of = and ties an additional name to it.

The object on the right-hand side may have existed before or it may be created by some operation specified by the code following =.

It's also possible to create nameless objects. Simply omit `name =` before some object creation code.

Python has the built-in function `id` to get the ID of an object.

```
print(id(a))
print(id(b))
```

```
139672564187504
139672564187504
```

We see, that indeed `a` and `b` refer to the same object.

Clear distinction between names and objects in Python adds flexibility, but also requires much more care when accessing or modifying data in memory. We will have to discuss possible pitfalls resulting from this concept at several points later on.

7.1.3 Equality of Objects

In Python we have objects and we have values contained in the objects. Thus, there are two fundamentally different questions which might be relevant for controlling program flow:

- Do two names refer to the same object?
- Do two objects (refered to by two names) contain the same value?

Consider the following code:

```
a = 1.23
b = 1.23
```

It creates two `float` objects both holding the value 1.23. To see that there are two objects we can look at the object IDs:

```
print(id(a))
print(id(b))
```

```
139672516871888
139672519436656
```

So the answer to the first question is 'no', but the answer to the second question is 'yes'.

To compare equality of objects Python knows the `is` operator. To compare equality of values Python has the `==` operator. Both yield a boolean value as result.

```
print(a is b)
print(a == b)
```

```
False
True
```

Negations of both operators are `is not` and `!=`, respectively. Using `is` is equivalent to comparing object IDs:

```
print(id(a) == id(b))
```

```
False
```

Hint: Behavior of the `is` operator is hardwired in Python (use == on integer objects returned by `id`). But == simply calls the dunder method `__eq__` of the left-hand side object. Thus, what happens during comparison depends on an object's type. Writing your own classes (object types) you may implement the `__eq__` method whenever appropriate. Without custom implementation Python uses a default one behaving similarly to `is`.

7.1.4 Local versus Global Names

Names in Python have a scope, that is, a region of code where they are valid. Names defined outside functions and other structures are referred to as *global names* or *global variables* or simply *globals*. If a name is defined (that is, tied to some object) inside a function or some other structure, then the name is *local*. Local names are undefined outside the function or structure they are defined in.

```
def my_func():
    print(c)
    d = 456

c = 123
my_func()
print(d)
```

```
123
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [7], in <cell line: 7>()
      5 c = 123
      6 my_func()
----> 7 print(d)

NameError: name 'd' is not defined
```

If there is a local name which is also a global name, than it's local version is used and the global one is left untouched.

```
def my_func():
    c = 456
    print(c)

c = 123
my_func()
print(c)
```

```
456
123
```

But how to change a global variable from inside a function? The `global` keyword tells the interpreter that a name appearing in a function refers to a global variable. The interpreter then uses the global variable instead of creating a new local variable.

```
def my_func():
    global c
    c = 456
    print(c)
```

(continues on next page)

(continued from previous page)

```
c = 123
my_func()
print(c)
```

```
456
456
```

We cannot access a global variable from inside a function and then introduce a local variable with the same name. This leads to an error because each name appearing in an assignment in a function is considered local throughout the function. Consequently, accessing the value of a global variable before creating a corresponding local variable is interpreted as accessing an undefined name. The interpreter then complains about accessing a local variable before assignment.

```
def my_func():
    print(c)
    c = 456

c = 123
my_func()
print(c)
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
Input In [10], in <cell line: 6>()
      3     c = 456
      5 c = 123
----> 6 my_func()
      7 print(c)

Input In [10], in my_func()
      1 def my_func():
----> 2     print(c)
      3     c = 456

UnboundLocalError: local variable 'c' referenced before assignment
```

Important: It's considered bad practice to use lots of global variables. Global variables result in low readability of code. Exceptions prove the rule.

7.2 Types

Here we introduce two more very fundamental object types, discuss type conversion and introduce the Python-specific concept of immutability.

7.2.1 More Types

In the *Crash Course* (page 53) we met several data or object types (classes):

- integers
- floats
- booleans
- lists

We also met strings, which will be discussed in more detail later on. Python ships with some more data types. Next to complex numbers, which we do not consider here, and several list-like types Python knows two very special data types:

- None type
- NotImplemented type

Both types can represent only one value: `None` and `NotImplemented`, respectively. Thus, they can be considered as constants. But since in Python everything is an object, constants are objects, too. Objects have a type (class). Thus, there is a None type and a NotImplemented type.

```
print(type(None))
print(type(NotImplemented))
```

```
<class 'NoneType'>
<class 'NotImplementedType'>
```

Existence of `NotImplemented` will be justified soon. Typically it's used as return value of functions to signal the caller that some expected functionality is not available.

The value `None` is used whenever we want to express that a name is not tied to an object. In that case we simply tie the name to *the* `None` object. We write 'the' because the Python interpreter creates only one object of None type. Such an object can hold only one and the same value. So there is no reason to create several different `None` objects.

```
a = None
b = None
print(id(a))
print(id(b))
```

```
94287211210560
94287211210560
```

```
a = 'Some string'
b = 'Some string'
print(id(a))
print(id(b))
```

```
139871271436464
139871271437360
```

In the second code block two string objects are created although both hold the same value. For both `None` values in the first code block only one object is created. If you play around with this you may find, that for short strings and small integers Python behaves like for `None`. This issue will be discussed in detail soon.

Hint: `None` is a Python keyword like `if` or `else` or `import`. It is used to refer to the object of None type. But the memory occupied by this object does not neccessarily contain a string 'None' or something similar. In fact, this object does not contain something useful. Its mileage is its existence, which allows to tie (temporarily unused) names to it. Same holds for `NotImplemented`.

We already met this concept when introducing boolean values. `True` and `False` are Python keywords, too. They are used to refer to two different objects of type `bool`. But these objects do not contain a string 'True' or 'False' or something similar. Instead, a `bool` object stores an integer value: 1 for the `True` object and 0 for the `False` object. How to represent `None`, `True` and so on in memory depends on the concrete implementation of the Python interpreter and is not specified by the Python programming language.

7.2.2 Type Casting

Type casting means change of data type. An integer could be casted to a floating point number, for example. Python does not have a mechanism for type casting. Instead, dunder methods can be implemented to work with objects of different types.

A very prominent dunder method for handling different object types is the `__init__` method, which is called after creating a new object. Its main purpose is to fill the new object with data. For Python standard types like `int`, `float`, `bool` the `__init__` method accepts several different data types as argument.

We've already applied the function `int`, which creates an `int` object, to strings. Thus, we have seen that the `__init__` method of `int` objects accepts strings as argument and tries to convert them to an integer value. The other way round, `str` for creating string objects accepts integer arguments.

```
a = '123'     # a string
b = int(a)
print(type(b))
print(b)
```

```
<class 'int'>
123
```

```
a = 123     # an integer
b = str(a)
print(type(b))
print(b)
```

```
<class 'str'>
123
```

```
a = 2     # an integer
b = float(a)
print(type(b))
print(b)
```

```
<class 'float'>
2.0
```

Data may get lost due to type casting. The Python interpreter will **not** complain about possible data loss.

```
a = 2.34     # a float
b = int(a)
print(type(b))
print(b)
```

```
<class 'int'>
2
```

Hint: It's good coding style to use explicit type casting instead of relying on implicit conversions whenever this increases readability.

A counter example is `1.23 * 56`, where the integer `56` is converted to float implicitely. Explicit casting would decrease readability: `1.23 * float(56)`.

Note: If you define a custom object type, it depends on your implementation of the type's `__init__` method what data types can be cast to your type.

7.2.3 Casting to Booleans

Casting to `bool` maps 0, empty strings and similar values to `False`, all other values to `True`.

```
print(bool(None))
print(bool(0))
print(bool(123))
print(bool(''))
print(bool('hello'))
```

```
False
False
True
False
True
```

If we use non-boolean values where booleans are expected, Python implicitly casts to bool:

```
if not '':
    print('cumbersome condition satisfied')
```

```
cumbersome condition satisfied
```

For historical reasons boolean values internally are integers (0 or 1). This sometimes yields unexpected (but well-defined) results. An example is the comparison of integers to `True`.

```
a = 3

if a:
    print('first if')
else:
    print('first else')

if a == True:
    print('second if')
```

(continues on next page)

(continued from previous page)

```
else:
    print('second else')
```

```
first if
second else
```

The first condition is equivalent to `bool(3)`, which yields `True`, whereas the second is equivalent to `3 == 1`, yielding `False`. See PEP 285[59] for some discussion of that behavior (PEP 285 introduced `bool` to Python).

7.2.4 Immutability

Objects in Python can be either *mutable* or *immutable*. Mutable objects allow modifying the value they hold. Immutable objects do not allow changing their values. Objects of simple type like `int`, `float`, `bool`, `str` are immutable whereas lists and most others are mutable.

Understanding the concept of (im)mutability is fundamental for Python programming. Even if the source code suggests that an immutable object gets modified, a new object is created all the time:

```
a = 1
print(id(a))

a = a + 1
print(id(a))
```

```
139871335317744
139871335317776
```

This code snipped first creates an integer object holding the value 1 and then ties the name `a` to it. In line 3, sloppily speaking, `a` is increased by one. More precisely, a new integer object is created, holding the result 2 of the computation, and the name `a` is tied to this new object.

Mutable objects behave as expected:

```
a = [1, 2, 3]
print(id(a))

a[0] = 4
print(id(a))
```

```
139871271501888
139871271501888
```

Immutability of some data types allows the Python interpreter for more efficient operation and for code optimization during execution. We will discuss some of those efficiency related features later on.

Always be aware of (im)mutability of your data. The following two code samples show fundamentally different behavior:

```
a = 1     # immutable integer
b = a

a = a + 1
print(a, b)
```

[59] https://peps.python.org/pep-0285/#resolved-issues

```
2 1
```

Increasing a does not touch b, because the integer object a and b refer to is immutable. Increasing a creates a new object. Then a is tied to the new object and b still refers to the original one.

```
a = [1, 2, 3]     # mutable list
b = a

a[0] = 4
print(a, b)
```

```
[4, 2, 3] [4, 2, 3]
```

Modifying a also modifies b, because a and b refer to the same mutable object.

7.2.5 Getting the Type

Although rarely needed, we mention the built-in function isinstance. It takes an object and a type as parameters and returns True if the object is of the given type.

```
print(isinstance(8, int))
print(isinstance(8, str))
print(isinstance(8.0, float))
```

```
True
False
True
```

7.2.6 Useful Dunder Functions for Custom Types

There are a bunch of dunder functions one should implement when creating custom types (cf. *Custom Object Types* (page 83)):

- __str__ is called by the Python interpreter to get a text representation of an object. For instance, it's called by print and whenever one tries to convert an object to string via str(...).
- __repr__ is simlar to __str__ but should return a more informative string representation. In the best case, it returns the Python code to recreate the object. See Python's documentation[60] for details.
- __bool__ is called whenever an object has to be cast to bool.
- __len__ is called by the built-in function len to determine an object's length. This is useful for list-like objects.

[60] https://docs.python.org/3/reference/datamodel.html#object.__repr__

7.2.7 Types are Objects

Since everything in Python is an object, types are objects, too. Thus, types may provide member variables and methods in addition to the corresponding objects' member variables and methods. In some programming languages members of a type are called *static members*.

Member variables of types occur for instance if constants have to be defined (almost always for convenience):

```python
class ColorPair:

    red = (1, 0, 0)
    green = (0, 1, 0)
    blue = (0, 0, 1)
    yellow = (1, 1, 0)
    cyan = (0, 1, 1)
    magenta = (1, 0, 1)

    def __init__(self, color1, color2):
        self.color1 = color1
        self.color2 = color2


my_pair = ColorPair(ColorPair.red, ColorPair.yellow)
```

Member functions of types are rarely used. One usecase are very flexible contructors for complex types, which do not fit into the `__init__` method due to many different variants of possible arguments. Often such constructors are named `from_...` and corresponding object creation code looks like

```
my_object = SomeComplexType.from_other_type(arg1, arg2, arg3)
```

In such cases the `from_...` methods return a new object of corresponding type, that is, they implicitly call the `__init__` method.

Defining methods for types requires advanced syntax contructs we do not discuss here.

7.3 Operators

Like most other programming languages Python offers lots of operators to form new data from existing one. Important classes of operators are

- arithmetic operators (`+`, `-`, `*`, `/`,...),
- comparison operators (`==`, `!=`, `<`, `>`,...),
- logical operators (`and`, `or`, `not`,...).

7.3.1 Operator Precedence

Expressions containing more than one operator are evaluated in well-defined order. Python's operators can be listed from highest to lowest priority. Operators with identical priority are evaluated from left to right.

Syntax	Operator
`**`	exponentiation
+, − (unary)	sign
`*`, `/`, `//`, `%`	multiplication, division
+, − (binary)	addition, substraction
==, !=, <, >, <=, >=	comparison
`not`	logical not
`and`	logical and
`or`	logical or

See Python's documentation[61] for a complete list of all operators.

7.3.2 Chained Comparisons

We may write chained comparions like `a < b < c`. Python interprets them as single comparisons connected by `and`, that is, `a < b and b < c`.

Note: Unfortunate expressions like `a < b > c` are allowed, too. This example is equivalent to `a < b and b > c`. In a chain only neighboring operands are compared to each other! There is no comparison between `a` and `c` here.

7.3.3 Augmented Assignments

For arithmetic binary operators there's a shortcut for expressions like `a = a + b`. We may write such as `a += b`. The latter is called an *augmented assignment*. Although the result will look the same, there are two technical differences one should know:

- augmented assignment may work in-place,
- for augmented assignment the assignment target will be evaluated only once.

In-place Computations

For usual binary operators the Python interpreter calls corresponding dunder methods, like `__add__` for `+`. Augmented assignments have their own dunder methods starting with `i`. So `+=` calls `__iadd__`, for instance. The intention is that `+=` may work in-place, that is, without creating a new object. Of course, this is only possible for mutable objects. If there is no dunder method for augmented assignment the interpreter falls back to the usual binary operator's dunder method.

Example: The `+` operator applied to two lists concatenates both lists. With `a = a + b` a new list object holding the result is created and then the name `a` is tied to the new object. With `a += b` list `b` is appended to the existing list object referred to by `a`.

```
a = [1, 2, 3]
b = [4, 5, 6]

print(id(a))

a = a + b

print(a)
print(id(a))
```

[61] https://docs.python.org/3/reference/expressions.html#operator-precedence

```
139941989941376
[1, 2, 3, 4, 5, 6]
139941989942272
```

```
a = [1, 2, 3]
b = [4, 5, 6]

print(id(a))

a += b

print(a)
print(id(a))
```

```
139941989933632
[1, 2, 3, 4, 5, 6]
139941989933632
```

Only One Evaluation

If the assignment target is a more complex expression like for list items, the expression will be evaluated twice with usual binary operators, but only once if augmented assignment is used.

Example: In the following code an item of a list shall be incremented by 1. The item's index is computed by some complex function `get_index` (which for demonstration purposes is very simple here). The two code cells show different implementations, resulting in a different number of calls to `get_index`.

```
def get_index():
    print('get_index called')
    return 2

a = [1, 2, 3, 4]

a[get_index()] = a[get_index()] + 1

print(a)
```

```
get_index called
get_index called
[1, 2, 4, 4]
```

```
def get_index():
    print('get_index called')
    return 2

a = [1, 2, 3, 4]

a[get_index()] += 1

print(a)
```

```
get_index called
[1, 2, 4, 4]
```

Note: If efficiency matters you should prefer augmented assignments. Even `a += b` with integers `a` and `b` is more efficient than `a = a + b`, because the name `a` will be looked up only once in the table mapping names to object IDs.

7.3.4 Operators as Member Functions

All Python operators, == and + for instance, simply call a specially named member function of the involved objects, a so called dunder method. Lines 3 and 4 of the following code cell do exactly the same thing:

```
a = 5

b = a + 2
c = a.__add__(2)

print(b)
print(c)
```

```
7
7
```

Dunder methods allow to create new object types which can implement all the Python operators themselve. What an operator does depends on the operands' object type. For instance, + applied to numbers is usual addition, but + applied to strings is concatenation.

Dunder Methods for Binary Operators

For binary operators like + and == there is always the question which of both objects to use for calling the corresponding dunder method. In case of comparisons Python uses the dunder method of the left-hand side operand (up to one minor exception which we don't discuss here). For arithmetic operations Python always tries the left operand first (again, we omit a minor exception). If it does not have the required dunder method, then Python tries the operand on the right-hand side. If both objects do not have the dunder method, then then interpreter stops with an error.

Binary arithmetic operations might be unsymmetric. Thus, there are two variants of most arithmetic dunder methods: one for applying an operation as the left-hand side operand and one for applying an operation as the right-hand side operand. For addition the methods are called `__add__` and `__radd__`, for multiplication we have `__mul__` and `__rmul__`. Others follow the same scheme.

Operands of Different Types

Often binary operators shall be applied to objects of different types (adding integer and floating point values, for instance). Even if both objects have the corresponding dunder method, one or both of them could lack code for handling certain object types.

In such a case Python calls the dunder method and the method returns `NotImplemented` to signal that it doesn't know how to handle the other operand. Then the interpreter tries the dunder method of the other operand. If it returns `NotImplemented`, too, then the interpreter stops with an error. The `__add__` function of integer objects cannot handle `float` objects, but `__add__` of `float` objects can handle integers, for example:

```
a = 2
b = 1.23
print(a.__add__(b))
print(b.__add__(a))
```

```
NotImplemented
3.23
```

Writing `a + b` in the example above first calls `a.__add__(b)`, which returns `NotImplemented`, then `b.__radd__`. With `b + a` the first call goes to `b.__add__` and no second call (to `a.__radd__`) is required.

An example of beneficial use of Python's flexible mechanism for customizing operators is discussed in the project on *Vector Multiplication* (page 919).

7.4 Efficiency

Python's combination of the names/objects concept and (im)mutability tends to waste memory and CPU time:

- There might be many different objects holding all the same value. In principle, every time the number 1 occurs in the source code, a new integer object is created.
- Tying names to other objects may leave objects without name. Such objects are no more accessible but resist in memory.
- Modifying immutable objects requires to create new objects. Thus, even simple integer computations require relatively complex memory management operations.

To mitigate these drawbacks, the Python interpreter uses several optimization strategies. Although such issues are rather technical we briefly discuss them here, because they sometimes yield unexpected results.

7.4.1 Preloaded Integers

To avoid object creation every time a new integer is used, the Python interpreter pre-creates integer objects for all integers from -5 to 256. This saves CPU time. The somewhat cumbersome range stems from statistical considerations about integer usage.

In addition, the interpreter takes care that no integer in this range is created twice during program execution. This saves memory. The behavior is demonstrated in the following code snipped:

```python
a = 8
b = 4 + 4
print(id(a))
print(id(b))
```

```
140115940688336
140115940688336
```

Both object IDs are identical, thus only one integer object is used. Since integer objects are immutable, this cannot cause any trouble.

7.4.2 String Interning

As for integers, the Python interpreter tries to avoid multiple string objects with the same value. Since corresponding comparisons may require too much CPU time, this technique is only used for short strings. The rules controlling which strings get interned and which not are relatively complex.

```python
a = 'short'
b = 'sh' + 'ort'
print(id(a))
print(id(b))
```

```
140115859848048
140115859848048
```

```
a = 'very very long'
b = 'very' + ' very long'
print(id(a))
print(id(b))
```

```
140115859848944
140115899208176
```

7.4.3 Repeated Literals in Source Code

Before executing a Python program, the interpreter checks the syntax and creates a list of all literals. Here, literals are all types of explicit data appearing in the source code, like integers or strings. If some literal appears multiple times and if objects of the corresponding data type are immutable, only one object is created.

```
# Copy the following Python code to a text file and feed the file to the
# Python interpreter to see the effect of optimization of repreated literals.

a = 'a long string, which usually is not interned'
b = 12345678
c = 'a long string, which usually is not interned'
d = 12345678

print(id(a))
print(id(b))
print(id(c))
print(id(b))
```

The names `a` and `c` will point to the same string object, although the string is too long to be interned by the string interning mechanism. The names `b` and `d` will point to the same integer object, although they are outside the range of preloaded integers.

Care has to be taken when using interactive Python interpreters like Jupyter. If the above code snipped is executed line by line in an interactive interpreter, then four different objects will be created, because the interpreter does not parse the full code in advance.

Important: Executing Python code with an interactive interpreter may yield different results than executing the same code at once with a non-interactive interpreter! In particular, performance measures like memory consumption may differ.

```
a = 'a long string, which usually is not interned'
b = 12345678
c = 'a long string, which usually is not interned'
d = 12345678

print(id(a))
print(id(b))
print(id(c))
print(id(d))
```

```
140115894986160
140115859673552
140115894986352
140115859679312
```

7.4.4 Garbage Collection

As described above, there might be objects without names. Such objects resist in memory, but are no more accessible. To avoid filling up memory as time passes, the Python interpreter automatically removes nameless objects from memory. This mechanism is known as garbage collection and is a feature not available in all programming languages. In the C programming language, for instance, the programmer has to take care to free memory, if data isn't needed anymore.

Sometimes, especially when working with large data sets, one wants to get rid of some data in memory to have more memory available for other purposes. One way is to tie all names refering to the no more needed object to other objects, which is somewhat unintuitive. Alternatively, the `del` keyword can be used to untie a name from an object.

```
a = 5000
del a
print(a)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [5], in <cell line: 3>()
      1 a = 5000
      2 del a
----> 3 print(a)

NameError: name 'a' is not defined
```

The last line leads to an error message, because after executing line 2 the name `a` is no more valid. Note that `del` only deletes the name, not the object. In the following code snipped the object remains in memory, because it has another name:

```
a = 5000
b = a
del a
print(b)
```

```
5000
```

CHAPTER

EIGHT

LISTS AND FRIENDS

We already met lists in the *Crash Course* (page 53). Now it's time to discuss some details and to introduce further list-like objects types as well as Python features for efficiently working with lists and friends.

Related exercises: *Lists and Friends* (page 867).

8.1 Tuples

Tuples are *immutable* objects which hold a *fixed-size* list of other objects. Imagine tuples as a list of pointers to objects, where the number of pointers and the pointers themselve cannot be changed. But note, that the objects the tuple entries point to can change if they are mutable. Object type for tuples is `tuple`.

8.1.1 Syntax

Tuples are defined as a comma separated list. Often the list is surrounded by parentheses:

```
a = (1, 5, 9)                               # a tuple of integers
b = 2, 3, 4                                 # works also without parentheses
c = ('some string', 'another_string')       # tuple of strings
d = (42, 'some string', 1.23)               # tuple with mixed types
```

Tuples with only one item are allowed, too. To distinguish them from single objects, a trailing comma is required.

```
a = 1         # integer
b = (1)       # integer
c = 1,        # tuple containing one integer
d = (1,)      # tuple containing one integer
```

8.1.2 Indexing

Tuple items can be accessed by index. The first item has index 0, the second index 1, and so on. The length of a tuple is returned by the built-in function `len`. Indexing with negative numbers gives the items in reverse order.

```
colors = ('red', 'green', 'blue', 'yellow')

print(colors[2])
print(colors[-1])
print(len(colors))
print(colors)
```

```
blue
yellow
4
('red', 'green', 'blue', 'yellow')
```

Subtuples can be extracted by so called *slicing*. Simply provide a range of indices: `[2:5]` gives a new tuple consisting of the items 2, 3, 4. The new tuple is a new object and remains available even if the original tuple vanishes (cf. *Garbage Collection* (page 101)).

```
some_colors = colors[1:3]
print(some_colors[0])
print(some_colors)
```

```
green
('green', 'blue')
```

Extracting every second item can be done by `[3:10:2]`, which gives items with indices 3, 5, 7, 9. The general syntax is `[first_index:last_index_plus_one:step]`. Here are some more indexing examples:

```
print(colors[2:])               # from 2 to end
print(colors[2:len(colors)])    # from 2 to end
print(colors[2:-1])             # from 2 to last but one
print(colors[:3])               # from 0 to 3
print(colors[:])                # all
```

```
('blue', 'yellow')
('blue', 'yellow')
('blue',)
('red', 'green', 'blue')
('red', 'green', 'blue', 'yellow')
```

8.1.3 Tuple Assignments

Tuples can be used for returning more than one value from a function. For this purpose Python provides the following code construct:

```
a, b, c = 23, 42, 6
print(a, b, c)
```

```
23 42 6
```

That is, we can assigne the contents of a tuple to a tuple of names. Such constructions typically are used if a function returns several return values packed into a tuple.

```
def integer_division(a, b):

    c = a // b
    d = a % b

    return c, d

quotient, remainder = integer_division(100, 13)

print(quotient)
print(remainder)
```

```
7
9
```

8.1.4 Tuples From Lists

Lists may be converted to tuples via usual type convertion:

```
some_list = [1, 2, 3, 4, 5]
some_tuple = tuple(some_list)
```

Note that both list and tuple point to the some items. Items aren't copied! Modifying (mutable) list items will modify tuple items, too. See *Multiple Names and Copies* (page 106) for more details.

8.2 Lists

Lists are *mutable* objects which hold a *flexible number* of other objects. Lists can be regarded as mutable tuples. Indexing syntax is the same, including slicing. Type name for lists is `list`. Calling `list(...)` creates list from several other types (tuples, for instance).

In principle, Python lists can hold different object types. But it is considered bad practice to use this feature. It's better to have lists made up of objects of the same type.

8.2.1 List Methods

List objects come with several methods for modifying them.

The `append` method appends an item to a list.

```
a = []              # empty list
b = [2, 4, 6]       # list with three integer items
b.append(8)         # now the list has four items
print(len(b))       # length of list
```

```
4
```

To concatenate list use the + operator or the `extend` method.

```
[1, 2, 3, 4] + [9, 8, 7] + [23, 42]
```

```
[1, 2, 3, 4, 9, 8, 7, 23, 42]
```

```
a = [1, 2, 3, 4]
a.extend([9, 8, 7])
print(a)
```

```
[1, 2, 3, 4, 9, 8, 7]
```

With `sort` we may sort a list.

```
a = [3, 2, 5, 4, 1]
a.sort()
print(a)
```

```
[1, 2, 3, 4, 5]
```

To search a list use `index`. This method returns the index of the first occurrence of its argument in the list.

```
a = [3, 2, 5, 4, 1]
print(a.index(5))
```

```
2
```

To remove a list item either use the `del` keyword (remove by index) or the `remove` method (remove by value):

```
a = [1, 2, 3, 4]

del a[1]
print(a)

a.remove(3)
print(a)
```

```
[1, 3, 4]
[1, 4]
```

8.2.2 Multiple Names and Copies

Mutability of lists may cause troubles and care has to be taken:

```
a = [1, 2, 3, 4]
b = a

b[0] = 9

print(a)
print(b)
```

```
[9, 2, 3, 4]
[9, 2, 3, 4]
```

In this code snipped a list is created and the name `a` is tied to it. Then this list object gets `b` as a second name. Important: We have one (!) list object with two names, not two lists! Thus, modifying `b` also modifies `a`.

To copy a list use the `copy` method:

```
a = [1, 2, 3, 4]
b = a.copy()

b[0] = 9

print(a)
print(b)
```

```
[1, 2, 3, 4]
[9, 2, 3, 4]
```

This creates a so called *shallow copy*. A new list object is created and the items of the new list point to exactly the same objects as the original list. If the original list consists of immutable objects (like integers), then modifying the copy will not alter the original. But if list items point to mutable objects, then altering the objects of the copy will modify the original list. Copying a list including all objects the list points to, is known as *deep copying*. How to automatically deep-copy a list will be discussed later on.

```
a = [[1, 2], [3, 4]]
b = a.copy()

# alter b (not its items); works because b is a (shallow) copy of a
b.append([5, 6])
print('a:', a)
print('b:', b)

# alter items of b; also alters items of a because b isn't a deep copy of a
b[0][0] = 9
print('a:', a)
print('b:', b)
```

```
a: [[1, 2], [3, 4]]
b: [[1, 2], [3, 4], [5, 6]]
a: [[9, 2], [3, 4]]
b: [[9, 2], [3, 4], [5, 6]]
```

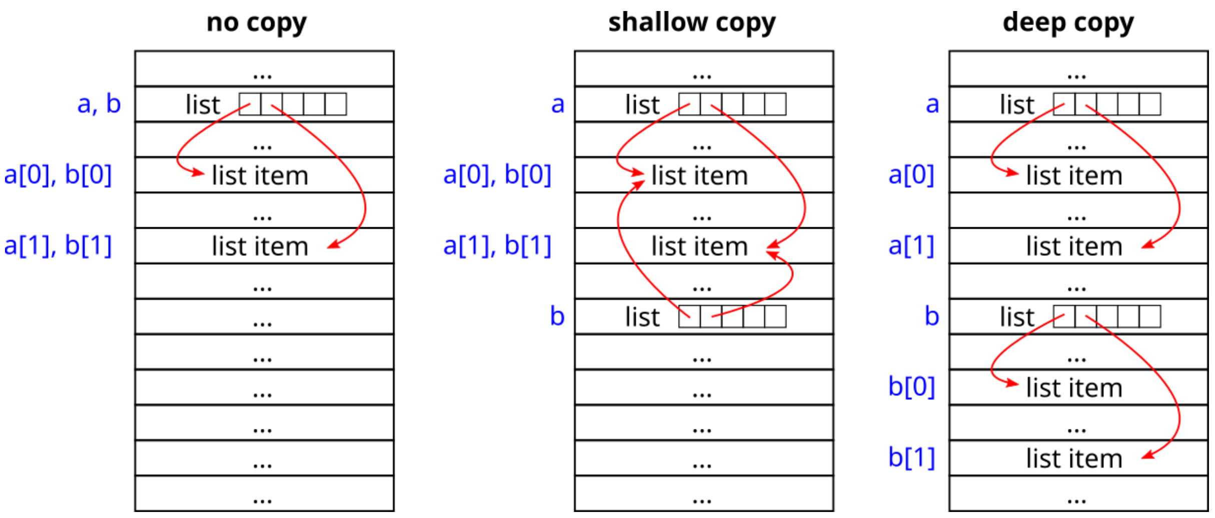


Fig. 8.1: Python lists are lists of memory locations (object IDs). Shallow copying only copies the list of memory locations. Deep copying also copies the data at those memory locations.

8.3 Dictionaries

Dictionaries are like lists, but indices (here denoted as *keys*) are not restricted to integers but can be of any immutable type. Even tuples are allowed as keys if they do not contain mutable items. Data types for keys can be mixed. Type name for dictionaries is `dict`.

8.3.1 Creating Dictionaries

Dictionary items are defined as colon separated pairs `key: value`.

```
person = {'name': 'John', 'surname': 'Doe', 'age': 42}
print(person['name'])

person['age'] += 1
print(person['age'])
```

```
John
43
```

To add data to an existing dictionary simply assign to the new key:

```
person['gender'] = 'male'
```

With `{}` we obtain an empty dictionary.

8.3.2 Dictionary Methods

Items may be removed with `del(key)`, like for lists.

The `keys` method returns a list-like object containing all keys used in the dictionary. Similarly, the `values` method returns all values in the dictionary and the `items` method returns all pairs (tuples) of keys and corresponding items.

Note: Python follows the duck typing[62] approach: If it looks like a duck, walks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.

In other words: There are many object types in Python which behave like a list, but aren't of type `list`. In particular, `len` and indexing syntax `[...]` may be used for such objects.

8.4 Iterable Objects

Tuples, lists, and dictionaries are examples of iterable objects. These are objects which allow for consecutive evalution of their items. There exist more types of iterable objects in Python and we may define new iterable objects by implementing suitable dunder methods.

[62] https://en.wikipedia.org/wiki/Duck_typing

8.4.1 For loops

We briefly mentioned for loops in the *Crash Course* (page 53). Now we add the details.

Basic Iteration

For loops allow to iterate through iterable objects of any kind, especially tuples, lists and dictionaries.

```
colors = ('red', 'green', 'blue', 'yellow')

for color in colors:
    print(color)
```

```
red
green
blue
yellow
```

The indented code below the `for` keyword is executed as long as there are remaining items in the iterable object. In the example above the name `color` first points to `colors[0]`, then, in the second run, to `colors[1]`, and so on.

This index-free iteration is considered *pythonic*: it's much more readable than indexing syntax and directly fulfills the purpose of iteration, while indexing in most cases is useless additional effort.

Note: The `range` built-in function we saw in the crash course is not part of the for loop's syntax. Instead, it's a built-in function returning an iterable object which contains a series of integers as specified in the arguments passed to the `range` function. The returned object is of type `range`.

The `range` function takes up to three arguments: the starting index, the stopping index plus 1, and the step size. Step size defaults to 1 if omitted. If only one argument is provided, it's interpreted as stopping index (plus 1) and start index is set to 0.

Iteration With Indices

If next to the items themselve also their index is needed inside the loop, use the built-in function `enumerate` and pass the iterable object to it. The `enumerate` function will return an iterable object which yields a 2-tuple in each iteration. The first item of each tuple is the index, the second one is the corresponding object.

```
colors = ('red', 'green', 'blue', 'yellow')

for index, color in enumerate(colors):
    print(str(index) + ': ' + color)
```

```
0: red
1: green
2: blue
3: yellow
```

Iterating Over Multiple Lists in Parallel

Being new to Python one is tempted to indexing for iteration over multiple lists in parallel:

```
# non-pythonic!

names = ['John', 'Max', 'Lisa']
surnames = ['Doe', 'Muller', 'Lang']

for i in range(len(names)):
    print(names[i] + ' ' + surnames[i])
```

```
John Doe
Max Muller
Lisa Lang
```

For iterating over multiple lists at the same time pass them to `zip`. This built-in function returns an iterable objects which yields tuples. The first tuple consists of the first elements of all lists, the second of the second elements of all lists and so on. The returned iterable object has as many items as the shortest list.

```
names = ['John', 'Max', 'Lisa']
surnames = ['Doe', 'Muller', 'Lang']

for name, surname in zip(names, surnames):
    print(name + ' ' + surname)
```

```
John Doe
Max Muller
Lisa Lang
```

8.4.2 Comprehensions

Applying some operation to each item of an iterable object can be done via for loops. But there are handy short-hands known as *list comprehensions* and *dictionary comprehensions*.

List Comprehensions

General syntax is

```
[new_item for item in some_list]
```

The following code snipped generates a list of squares from a list of numbers.

```
some_numbers = [2, 4, 6, 8]

squares = [x * x for x in some_numbers]

print(squares)
```

```
[4, 16, 36, 64]
```

Like in for loops also multiple lists are possible:

```
some_numbers = [2, 4, 6, 8]
more_numbers = [1, 2, 3, 4]

products = [x * y for x, y in zip(some_numbers, more_numbers)]

print(products)
```

```
[2, 8, 18, 32]
```

Nested for loops work, too:

```
[new_item for item_a in list_a for item_b in list_b]
```

Same principles (zipping and nesting) work for more than two lists, too.

Note: List comprehensions have two advantages compared to for loops:

- For small loops a one-liner is more readable than several lines.
- In most cases list comprehensions are faster.

Dictionary Comprehensions

Dict comprehensions look very similar to list comprehensions. General syntax:

```
{new_key: new_value for some_item in some_iterable}
```

Example modifying values only:

```
person = {'name': 'John', 'surname': 'Doe', 'eyes': 'brown'}

# enclose all values with star symbols
stars = {key: '*' + value + '*' for key, value in person.items()}

print(stars)
```

```
{'name': '*John*', 'surname': '*Doe*', 'eyes': '*brown*'}
```

Example modifying keys and values of a dictionary:

```
person = {'name': 'John', 'surname': 'Doe', 'eyes': 'brown'}

# enclose keys and values with star symbols
stars = {'*' + key + '*': '*' + value + '*' for key, value in person.items()}

print(stars)
```

```
{'*name*': '*John*', '*surname*': '*Doe*', '*eyes*': '*brown*'}
```

Note: Dictionary comprehensions may be used to create new dictionaries from arbitrary iterable objects. For instance we could loop over the `zip`ped lists, one holding the keys and one holding the values.

Conditional Comprehensions

List and dictionary comprehensions can be extended by a condition: simply append `if some_condition` to the comprehension.

```
some_numbers = [2, 4, 6, 8]
more_numbers = [1, 0, 2, 3]

quotients = [x / y for x, y in zip(some_numbers, more_numbers) if y != 0]

print(quotients)
```

```
[2.0, 3.0, 2.6666666666666665]
```

The list (or dictionary) comprehension drops all items not satisfying the condition.

8.4.3 Manual iteration

Python has a built-in function `next` which allows to iterate through iterable objects step by step. For this purpose we first have to create an iterator object from our iterable object. This is done by the built-in function `iter`. Then the iterator object is passed to `next`. The iterator object takes care about what the next item is.

```
a = iter([3, 2, 5])

print(next(a))
print(next(a))
print(next(a))
```

```
3
2
5
```

Creation of the intermediate iterator object is done automatically if iterable objects are used in for loops and comprehensions.

8.4.4 The `in` keyword

To test whether an object is contained in an iterable object, we my use the `in` keyword.

```
a = [1, 5, 6, 8]
print(1 in a)
print(10 in a)
```

```
True
False
```

```
a = {'name': 'Jon', 'age': 42}
print('name' in a)
print('Jon' in a)
print('Jon' in a.values())
```

```
True
False
True
```

The counterpart to `in` is `not in`.

Note: Conditions `a not in b` and `not a in b` are equivalent. The first uses the `not in` operator, while the second uses `in` and then applies the logical operator `not` to the result.

CHAPTER

NINE

STRINGS

We already learned basic usage of strings in the *Crash Course* (page 53). Here we add the details. Correct string handling is essential for loading data from files and also for writing to files.

Related exercises: *Strings* (page 870).

9.1 Basics

9.1.1 Substrings

Strings are sequence-type objects, that is, they can be regarded as lists of characters. Each character then is itself a string object.

```
a = 'some string'
print(a[0])
print(a[1])
print(len(a))
```

```
s
o
11
```

Even slicing is possible.

```
b = a[2:8]
print(b)
```

```
me str
```

Remember that strings are immutable. Thus, `a[3] = 'x'` does not replace the fourth character of `a` by `x`, but leads to an error message.

9.1.2 Line Breaks in String Literals

Sometimes it's necessary to use line breaks when specifying strings in source code. For this purpose Python knows triple quotes.

```
a = '''a very long string
spanning several lines
in source code'''

b = """another very long string
spanning several lines
in source code"""

print(a)
print(b)
```

```
a very long string
spanning several lines
in source code
another very long string
spanning several lines
in source code
```

As we see, line breaks in source code become part of the string. If this behavior is not desired, end each line with `\`.

```
a = '''a very long string\
spanning several lines\
in source code'''

print(a)
```

```
a very long stringspanning several linesin source code
```

A common pattern in Python source files is

```
a = '''\
first line
second line
last line\
'''
print(a)
```

```
first line
second line
last line
```

This way we get one-to-one correspondence between source code and screen output.

Note: A trailing \ tells Python that the souce code line continues on the next line, that is, that the next line break is not to be considered as line break. Usage is not restricted to strings. Long source code lines (longer than 80 characters) should be wrapped to multiple short lines.

9.1.3 Raw Strings

To have line breaks and special characters in string literals we have to use escape sequences like `\n`, `\"`, and so on. If we want to prevent the Python interpreter from translating escape sequences into corresponding characters, we may use *raw strings*. A raw string is taken as is by the interpreter. To mark a string literal as raw string prepend `r`.

```
a = r'Here is a line break:\nNow we are on a new line'
print(a)
print(type(a))
```

```
Here is a line break:\nNow we are on a new line
<class 'str'>
```

9.1.4 Useful Member Functions

Read Python's documentation of

- `count`[63]
- `find`[64]
- `replace`[65]
- `split`[66]
- `upper`[67]
- `is...`[68]

to get an idea of what's possible with string methods.

9.2 Special Characters

Strings may contain characters not available on some or all keyboards, like umlauts ä, ö, ü on US keyboards. To use such special characters in string literals Python provides escape sequences `\x`, `\u`, and `\U`.

9.2.1 Character Sets

The relation between characters in strings and their numerical representation in memory will be considered in detail in the chapter on *Text Files* (page 124). For the moment we content ourselves with the observation that there are several such mappings, the most prominent ones known as ASCII[69], the ISO 8859[70] family and several Unicode[71] variants.

ASCII knows 128 different characters, the ISO 8859 family some hundred, and Unicode several million ones. Nowadays, Unicode in its UTF-8[72] variant is the standard mapping for numerical representations of characters. Even Windows is adopting UTF-8 more and more after backing the wrong horse for two decades.

There are lots of searchable lists of Unicode characters in the web. Wikipedia's List of Unicode characters[73] is a good starting point.

[63] https://docs.python.org/3/library/stdtypes.html#str.count
[64] https://docs.python.org/3/library/stdtypes.html#str.find
[65] https://docs.python.org/3/library/stdtypes.html#str.replace
[66] https://docs.python.org/3/library/stdtypes.html#str.split
[67] https://docs.python.org/3/library/stdtypes.html#str.upper
[68] https://docs.python.org/3/library/stdtypes.html#str.isalnum
[69] https://en.wikipedia.org/wiki/ASCII
[70] https://en.wikipedia.org/wiki/ISO/IEC_8859
[71] https://en.wikipedia.org/wiki/Unicode
[72] https://en.wikipedia.org/wiki/UTF-8
[73] https://en.wikipedia.org/wiki/List_of_Unicode_characters

9.2.2 Unicode Characters in String Literals

To use Unicode characters in a string literal we either may type or copy them to the source code file or we may specify the character numerically. If the numercial representation has two hexadecimal digits, use `\x` followed by the two digits. In case of 4 digits use `\u` and for 8 digit characters use `\U`.

```
print('umlauts: \xe4, \xf6, \xfc')
print('Greek: \u03b1, \u03b2, \u03b3')
print('Chinese (?): \U0002070e, \U00028cd2')
```

```
umlauts: ä, ö, ü
Greek: α, β, γ
Chinese (?): 𠜎, 𨳒
```

Note: Not all Unicode characters are available in each font.

Some Unicode codes do not represent concrete characters, but control reading direction (left to right or right to left) and other properties like spacing.

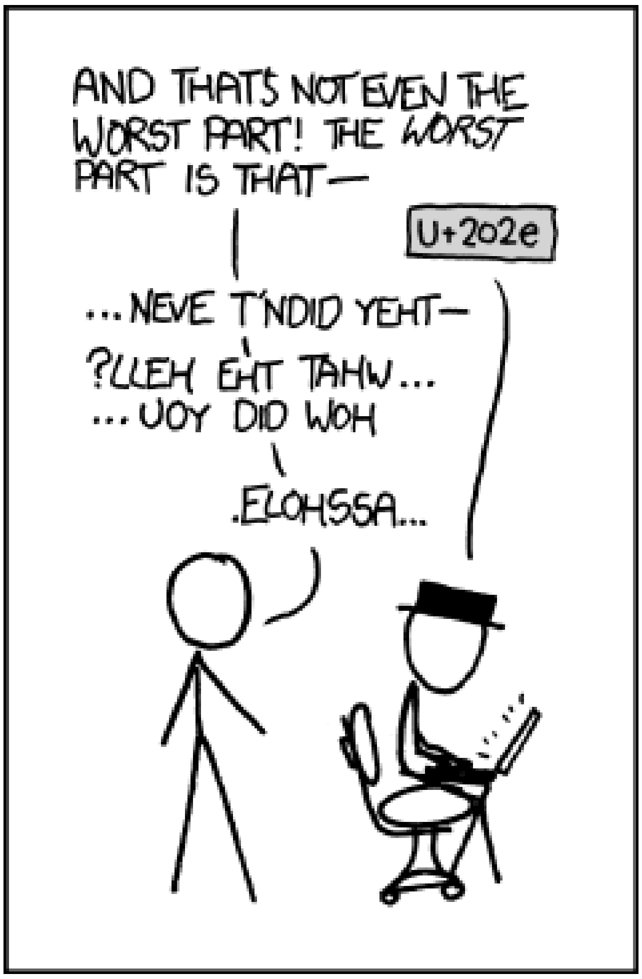


Fig. 9.1: Collaborative editing can quickly become a textual rap battle fought with increasingly convoluted invocations of U+202a to U+202e. Source: Randall Munroe, xkcd.com/1137[74]

[74] https://xkcd.com/1137

9.3 String Formatting

9.3.1 The `format` Method

String objects have a `format` method. This method allows for converting numbers and other data to strings.

```
a = 4
b = 5
c = a + b
nice_string = 'The result of {} plus {} is {}.'.format(a, b, c)
print(nice_string)
```

```
The result of 4 plus 5 is 9.
```

Calling the `format` method of a string replaces all pairs `{}` by the arguments passed to the format method. Next to integers also floats and strings can be passed to `format`. The original string is not modified because it's immutable. Instead, `format` returns a new string object.

The arguments of `format` can also be accessed by providing their index: the first argument has index 0, the second has index 1, and so on.

```
a = 4
b = a + a
print('The result of {0} plus {0} is {1}.'.format(a, b))
```

```
The result of 4 plus 4 is 8.
```

For more complex output (or more readable code) keyword arguments can be passed to format.

```
print('Mister {name} is {age} years old.'.format(name='Muller', age='42'))
```

```
Mister Muller is 42 years old.
```

Converting numbers to strings sometimes requires additional parameters: How many digits to use for floats? Shall numbers in several lines be aligned horizontally? There is a whole 'mini language' for writing formatting options. Here we provide only some examples. For details see Python documentation on format string syntax[75].

```
# print integer as float with 2 decimal places
print('The result is {:.2f}.'.format(3))

# right-align integers in fixed-width area
print('number of apples:  {:3d}'.format(2))
print('number of oranges: {:3d}'.format(145))

# same works for floats and strings
print('percentage of apples:  {:7.2f}'.format(2.1234))
print('percentage of oranges: {:7.2f}'.format(80.8976455))
print('percentage of bananas: {:7}'.format('unknown'))
```

```
The result is 3.00.
number of apples:    2
number of oranges: 145
percentage of apples:     2.12
percentage of oranges:   80.90
percentage of bananas: unknown
```

[75] https://docs.python.org/3/library/string.html#formatstrings

If indices or names are used for refering to `format`'s arguments, they have to be placed on the left-hand side of the collon: `{name:3.2f}`.

9.3.2 Formatted String Literals (f-Strings)

Starting with version 3.6 of Python there is a more comfortable way to format strings. It's very similar to formatting via `format` method, but requires less code and increases readability. The two major differences are:

- string literals have to be prefixed by `f`,
- the curly braces may contain a Python expression (an object name, for instance).

```python
a = 123
print(f'Here you see {a}.')

b = 4.56789
print(f'Formatting works as above: {b:.2f} is a rounded float.')
```

```
Here you see 123.
Formatting works as above: 4.57 is a rounded float.
```

For details see formatted string literals[76] in the Python documentation.

[76] https://docs.python.org/3/tutorial/inputoutput.html#formatted-string-literals

CHAPTER

TEN

ACCESSING DATA

There exist several sources of data: files, data bases, and web services, for instance. Here we consider basic file access and common file formats for storing large data sets. We also learn how to automatically download data from the web and how to scrape data from websites.

Related exercises: *File Access* (page 872).

Related projects:

- *Cafeteria* (page 947)
- *Weather* (page 923)
  - *DWD Open Data Portal* (page 923)
  - *Getting Forecasts* (page 925)

10.1 File IO

Next to screen IO, input from and output to files is the most basic operation related to data processing. Almost all data science projects start with reading data from one or more files. In this chapter we discuss basic file access. Later on there will be several specialized modules and functions to make things more straight forward. But from time to time, in case of uncommon file formats, one has to resort to the most basic operations.

10.1.1 Basics

Reading data from a file or writing data to a file requires three steps:

1. Open the file

Tell the operating system, that file access is required. The operating system checks permissions and, if everything is okay, returns a file identifier (usually a number), which has to be used for all subsequent file operations.

2. Read or write data

Tell the operating system to move data between the file and some place in memory which can be accessed by the Python interpreter.

3. Close the file

Tells the operating system, that file access is no longer required. The operating system, thus, knows that other applications now may read from or write to the file.

In Python all file related data and operations are encapsulated into a file object. There are different types of file objects depending on the file type (text or binary) and on some technical issues. All types of file objects provide identical member functions for reading and writing. Here is the basic procedure:

```
f = open('testdir/testfile.txt', 'r')
file_content = f.read()
f.close()

print(file_content)
```

```
Some text
in some file
splitted over
multiple lines.
```

This code snipped opens a file for reading (argument `'r'`) and assignes the name `f` to the resulting file object. Then the whole content of the file is stored in the string object `file_content`. Finally, the file is closed and it's content is printed to screen.

If something goes wrong, for instance the file does not exist, the Python interpreter stops execution with an error message. For the moment, we do not do any error checking when operating with files (this is very bad practice!).

Note: The `read` method and all other methods for reading and writing files can be used to process text data and binary data. Providing the `'r'` argument to `open` tells Python to open the file as text file. Reading data from the file results in a string object. If '`rb'` is used instead, then the file is handled as binary file and reading results in a list of bytes. Details will be discussed in the chapter on *Text Files* (page 124).

Default mode is `'r'`. So specifying no mode opens for reading in text mode.

Important methods for reading and writing files are `read`, `readline`, `readlines`, `write`, `writelines`, `seek`. See methods of file objects[77] in the Python documentation.

For more details on access modes see documention of `open`[78].

10.1.2 Paths are OS Dependent

Paths to a file are operating system dependent. Thus, using paths in the `open` function makes our code operating system dependent. This should be avoided and luckily there are techniques to avoid such OS dependence.

Linux/Unix/macOS

In Linux and other Unix like systems (macOS for instance), all files can be accessed via paths of the form `'/directory/subdirs/file'`. That is, a list of directory names separated by slashs and ending with the file name. If the path starts with a slash, then it's an absolute path, else a relative one.

Drives can be mounted as directory everythere in the file system's hierarchy. Thus, there is no need for special drive related path components.

Windows

Windows uses a different format: `'drive:\directory\subdirs\file'`. Instead of slashs backslashs are used as delimiters and there is an additional drive letter in absolute paths followed by a colon. The purpose of the drive letter is to select one of several physical (or even logical) drives.

From the programmer's point of view, additional effort is required to make code work in both worlds.

[77] https://docs.python.org/3/tutorial/inputoutput.html#methods-of-file-objects
[78] https://docs.python.org/3/library/functions.html#open

10.1.3 OS Independent Paths in Python

The Python module `os.path` provides the function `join`. This function takes directory names and a file name as arguments and returns a string containing the corresponding path with appropriate (OS dependent) delimiters. So output of the follow code snipped depends on the OS used for execution.

```
import os.path

test_path = os.path.join('testdir', 'testfile.txt')

print(test_path)
```

```
testdir/testfile.txt
```

The path separator (/ or \) used by the OS is available in `os.sep`.

```
import os

print('path separator:', os.sep)
```

```
path separator: /
```

10.1.4 Directory Listings

Often data sets are scattered over many files, for instance one file per customer, each file containing all the customers transactions in an online shop. In such cases we need to get a list of all files in a specified directory. Such functionality is provided by Python's `glob` module.

```
import glob

file_list = glob.glob('testdir/*')

for file in file_list:
    print(file)
```

```
testdir/test.zip
testdir/testwrite-windows.txt
testdir/iso8859-1.txt
testdir/umlauts.txt
testdir/testfile.txt
testdir/testwrite.txt
testdir/utf-8.txt
```

The `glob` module's `glob` function takes a path containing wildcards like `*` (arbitrary string) and `?` (arbitrary character), for instance, and returns a list of all files matching the specified path.

Note: To make above code snipped OS independent we should write `glob.glob(os.path.join('testdir', '*'))`.

10.2 Text Files

Text files are the most basic type of files. They contain string data. Historically there was a one-to-one mapping between byte values (0…255) and characters. Nowadays things are much more complex, because representing all the world's languages requires more than 256 different characters. When reading from and writing to text files the mapping between characters and there numerical representation in memory or storage devices is of uttermost importance.

Text file not only contain so called pritnable characters like letters and numbers, but also control characters like line breaks and tab stops. Related issues will be discussed in this chapter, too.

10.2.1 Encodings

Every kind of data has to be converted to a stream of bits. Else it cannot be processed by a computer. For strings we have to distinguish between their representation on screen (which symbol) and their representation in memory (which sequence of bits). Mapping between screen and memory representation is known as *encoding*. *Decoding* is mapping in opposite direction.

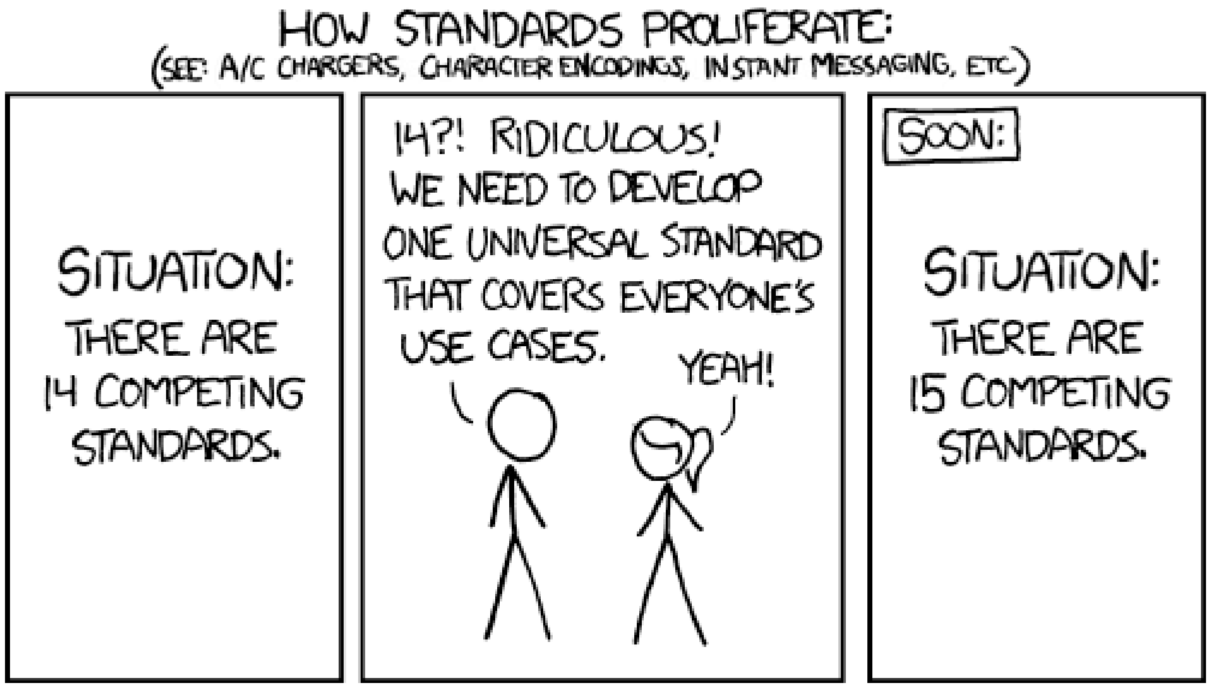

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Fig. 10.1: Fortunately, the charging one has been solved now that we've all standardized on mini-USB. Or is it micro-USB? Shit. Source: Randall Munroe, xkcd.com/927[79]

[79] https://xkcd.com/927

ASCII

Historically, each character of a string has been encoded as exactly one byte. A byte can hold values from 0 to 255. Thus, only 256 different characters are available, including so called control characters like tabs and new line characters.

The mapping between byte values and characters, the so called *character encoding*, has to be standardized to allow exchanging text files. For a long time, the most widespread standard has been *ASCII* (American Standard Code for Information Interchange). But since ASCII does not contain special characters like umlauts in other languages, several other encodings were developed. The ISO 8859[80] family is a very prominent set of ASCII derivates.

The first 128 characters of almost all encodings coincide with ASCII, but the remaining 128 contain different symbols. Thus, to read text files one has to know the encoding used for saving the file. Typically, the encoding is not (!) saved in the file, but has to be guessed or communicated along with the file. Have a look at the list of encodings[81] Python can process.

Unicode

Nowadays, *Unicode* is the standard encoding. More precisely, Unicode defines a group of encodings. We do not go into the details here. For our purposes it suffices to know that Unicode contains several hundred thousand symbols and the most important encoding of Unicode is called *UTF-8*. The eight means that most characters require only 8 bits. The symbols associated with the byte values 0 to 127 coincide with ASCII. A byte value above 127 indicates a multi-byte symbol comprising two, three, or four bytes.

Linux/Unix/macOS

Non-Windows systems (Linux, Unix, macOS) have native UTF-8 support for decades. It's the standard encoding for Websites and other internet related applications.

Windows

Windows, even Windows 10, uses a different Unicode encoding under the hood and supports UTF-8 at the surface only. Sometimes, if one has to dig deeper into the system, unexpected things may happen. Older Windows version did not have UTF-8 support at all. Always check the encoding if you work with text data generated on a Windows system!

Encodings in Python

Python uses UTF-8 and strictly distinguishes between strings and their encoded representation. The string is what we see on screen, whereas the encoded form is what is written to memory and storage devices.

String objects provide the `encode` member function. This function returns a sequence of bytes. This sequence is of type `bytes`. A `bytes` object is immutable. In essence, it's a tuple of integers between 0 and 255.

The other way round `bytes` objects provide a member function `decode` to transform them to strings.

```python
a = 'some string with umlauts: ä, ö, ü'
b = a.encode()
print(b)
```

```
b'some string with umlauts: \xc3\xa4, \xc3\xb6, \xc3\xbc'
```

As we see, `bytes` objects can be specified like strings, but prefixed by `b`. The only difference is that all bytes holding values above 127 or non-printable characters (line breaks, for instance) are replaced by their integer values in hexadecimal notation with the prefix `\x`, which is the escape sequence for specifying characters in hexadecimal notation. If we want to use octal notation, the escape sequence is `\000` where `000` is to be replaced by a three digit octal number.

[80] https://en.wikipedia.org/wiki/ISO/IEC_8859
[81] https://docs.python.org/3/library/codecs.html#standard-encodings

```
c = b.decode()
print(c)
```

```
some string with umlauts: ä, ö, ü
```

Note: The `encode` and `decode` methods accept an optional `encoding` parameter, which defaults to `'utf-8'`.

There is also a mutable version of `bytes` objects: `bytearray` objects. They provide a `decode` function, too.

Reading from a file opened in text mode is equivalent to reading after opening in binary mode followed by a call to `decode`. Similarly for writing. The `open` function has knowns an optional `encoding` parameter for text mode, defaulting to `'utf-8'`.

10.2.2 Line Breaks

Encoding line breaks in text files is done differently on different operating systems. The ASCII and Unicode standards define two symbols indicating a line break. One is symbol 10, known as line feed (LF for short). The other is symbol 13, known as carriage return (CR for short).

Historically, when typewriters were the standard text processing tools, starting a new line required two actions: move to next line without moving the carriage, then move the carriage to its rightmost position. Thus, there are two different symbols for these two actions.

Linux/Unix/macOS

Linux and other Unix like system (macOS, for instance) use single byte line breaks encoded by LF. Old versions of macOS used CR, but then developers switched to LF.

Windows

Windows adhers to the two-step legacy from pre-computer era. That is, on Windows line breaks in text data are encoded by the two bytes CR and LF.

Python can handle all three versions of line break codes (LF, CR, CR LF) and tries to hide the differences from the programmer. But be aware, that writing text files may produce different results on Windows and Linux/Unix/macOS machines.

10.2.3 Encoding Problem Examples

```
import os.path
```

Wrong Encoding

If we open an ISO 8859-1 encoded text file without specifying an encoding (that is, UTF-8 is used), the interpreter fails either fails to interpret some bytes or it shows wrong symbols.

```
f = open(os.path.join('testdir', 'iso8859-1.txt'), 'r')
text = f.read()
f.close()

print(text)
```

```
---------------------------------------------------------------------------
UnicodeDecodeError                        Traceback (most recent call last)
Input In [4], in <cell line: 2>()
      1 f = open(os.path.join('testdir', 'iso8859-1.txt'), 'r')
----> 2 text = f.read()
      3 f.close()
      5 print(text)

File ~/anaconda3/envs/ds_book/lib/python3.10/codecs.py:322, in␣
 ↪BufferedIncrementalDecoder.decode(self, input, final)
    319 def decode(self, input, final=False):
    320     # decode input (taking the buffer into account)
    321     data = self.buffer + input
--> 322     (result, consumed) = self._buffer_decode(data, self.errors, final)
    323     # keep undecoded input until the next call
    324     self.buffer = data[consumed:]

UnicodeDecodeError: 'utf-8' codec can't decode byte 0xe4 in position 24:␣
 ↪invalid continuation byte
```

If we open an UTF-8 encoded file with ISO 8859-1 decoding we see garbled symbols.

```
f = open(os.path.join('testdir', 'utf-8.txt'), 'r', encoding='iso-8859-1')
text = f.read()
f.close()

print(text)
```

```
Some umlauts: Ã¤, Ã¶, Ã¼.
This file is UTF-8 encoded.
```

Writing Line Breaks

The following code produces different files on Linux/Unix/macOS and Windows.

```
f = open(os.path.join('testdir', 'testwrite.txt'), 'w')
text = f.write('test\n\n\n\n\n\n\n\n\n\ntest')
f.close()
```

On Linux and Co. the file will have 18 bytes. On Windows it will have 28 bytes due to Windows' 2-byte line breaks. Opening the file in binary mode shows the line break encoding:

```
f = open(os.path.join('testdir', 'testwrite.txt'), 'rb')
text = f.read()
f.close()

print(tuple(text))
```

```
(116, 101, 115, 116, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 116, 101, 115, 116)
```

Using `print(text)` directly shows line breaks as `\n`, which is nice almost always, but not here. So we convert the bytes object to a tuple of integers before printing.

If the file has been writen on a Windows machine, it looks like that:

```
(116, 101, 115, 116, 13, 10, 13, 10, 13, 10, 13, 10, 13, 10, 13, 10, 13, 10, 13,
 ↪ 10, 13, 10, 13, 10, 116, 101, 115, 116)
```

10.3 ZIP Files

Large data sets usually ship as compressed files, mostly ZIP files. Extracting such files requires lots of disk space. Compressed text files, for instance, are much smaller than the original files (factor 5 to 10).

In Python we might use the `zipfile` module. This module allows to read single files from a ZIP archive without extracting the whole archive. We have to create an object of type `zipfile.ZipFile`. Such objects provide an `open` method. The return value of `open` is a file-like object, that is, it can be processed like usual files. Files from ZIP archives are always opened in binary mode by `ZipFile` objects' `open` method.

The `namelist` method returns a list of file names in the ZIP archive.

```python
import os.path
import zipfile

# open zip file
zf = zipfile.ZipFile(os.path.join('testdir', 'test.zip'))

# show contents of zip file
print(zf.namelist())

# read one specific file from zip file
f = zf.open('file.txt')
print('file contents:')
print(f.read().decode())     # opened in binary mode!
f.close()

# close zip file
zf.close()
```

```
['another_file.txt', 'file.txt']
file contents:
This is a file for testing zipfile module.
```

10.4 CSV Files

The simplest form for storing spreadsheet data are *comma separated values* (CSV) in a text file. Each line of a CSV file contains one row of the spreadsheet. The columns are separated by commas and sometimes by another symbol. CSV files may contain column headers in their first line(s).

A typical CSV file looks like that:

```
first_name,last_name,town
John,Miller,Atown
Ann,Abor,Betown
Bob,Builder,Cetown
Nina,Morning,Detown
```

CSV files are not standardized. Thus, there might be cumbersome deviations from what one expects to be a simple CSV file. The CSV format is used to move data between different sources which cannot read each others native file formats.

In Python we can use the module `csv` for reading data from CSV files. It provides the class `csv.reader`. When creating a `csv.reader` object we have to pass a file object of the CSV file as parameter. The `csv.reader` object then is an iterator object. It yields one line of the CSV file per iteration. More precisely, it yields a list of strings. Each string contains the data from the corresponding column.

See documentation of `csv` module[82] for details.

[82] https://docs.python.org/3/library/csv.html

10.5 HTML Files

HTML (*hypertext markup language*) files are text files containing additional information for rendering text like font type, font size, foreground and background colors. Also images, tables and other objects may be described or referenced by a HTML document. Typically, HTML files are interpreted and rendered by web browsers. Almost all websites consist of HTML files.

In data science knowing some basic HTML is important for webscraping, that is, for automatically extracting information from websites.

10.5.1 HTML fundamentals

A very basic HTML file looks like this:

```
<html>
    <head>
        <title>Title of webpage</title>
    </head>
    <body>
        <h1>Some heading</h1>
        <p>Text and text and more text in a paragraph.
        Here comes a <a href="http://some.where">link to somewhere</a>.</p>
    </body>
</html>
```

The file starts with `<html>` and ends with `</html>`. Then there is a head and a body. The head contains auxiliary information like the webpage's title, which is often shown in the browser window's title bar. The body contains the contents of the page.

There are many different *HTML tags* to influence rendering of the contents.

Headings from large to small: `h1`, `h2`, `h3`, `h4`, `h5`.

Paragraph: `p`.

Link: `a` with attribute `href`.

Table: `table`, `tr` (row inside table), `td` (cell inside row), and some more.

Image: `img` with attribute `src` (the URL of the image).

Invisible elements for layout control: `span` (inline element), `div` (box).

All tags have the attributes `style` (for specifying font size, colors and so on), `id` (a unique identifier for advanced style control and scripting), `class` (an identifier shared by several elements for advanced layout control).

Have a look at the HTML documentation[83] for details.

Modern browsers have tools to help understand a HTML file's structure. In Firefox or Chromium right-click some element of the webpage and click 'Inspect' in the pop-up menu. Then navigate through the HTML source. To see the whole HTML source code right-click and choose 'View Page Source'.

[83] https://html.spec.whatwg.org/

10.5.2 Parsing HTML files with Python

There are several modules available for parsing HTML files in Python. Here, parsing means to convert the textual representation into more structured Python objects. One such module is Beautiful Soup[84], which is not part of Python's standard library, but has to be installed manually.

For installation use `beautifulsoup4`. For importing `bs4` is the correct name.

```
import bs4
```

We have to create a `BeautifulSoup` object, whoes contructor takes a string or an opened file object as argument. The `BeautifulSoup` object then provides methods to find HTML tags by specifying tag name, id attribute, class attribute or one of several other properties. We do not have to write code for parsing HTML files. Instead we can search the file with BeautifulSoup's methods.

```
html = '''\
<html>
    <head>
        <title>Title of webpage</title>
    </head>
    <body>
        <h1>Some heading</h1>
        <p>Text and text and more text in a paragraph.
        Here comes a <a href="http://some.where">link to somewhere</a>.</p>
    </body>
</html>
'''

soup = bs4.BeautifulSoup(html)
```

The `find_all` method returns a list of objects representing subsets of the HTML file matching the arguments passed to `find_all`. In the following code snippet we search for `a` tags, that is, for links. But we could also search for certain attribute values and other criteria. There is also a `find` method which returns the first occurrence only.

The objects returned by `find_all` and `find` themselves provide corresponding methods to refine search.

```
# find all links
links = soup.find_all('a')

print('#links:', len(links))
print('last link:', links[-1])
```

```
#links: 1
last link: <a href="http://some.where">link to somewhere</a>
```

See Beautiful Soup's documentation[85] for details.

[84] https://www.crummy.com/software/BeautifulSoup
[85] https://beautiful-soup-4.readthedocs.io

10.6 XML Files

XML (*extensible markup language*) files look like *HTML Files* (page 129), but with custom tag names. Each XML file may use its own set of tags to describe data. In principle, HTML is a special case of XML.

Standard conforming XML files have some format specifications in there first lines. Content without such specifications could look like this:

```xml
<person>
    <first_name>John</first_name>
    <last_name>Miller</last_name>
    <town>Atown</town>
</person>
<person>
    <first_name>Ann</first_name>
    <last_name>Abor</last_name>
    <town>Betown</town>
</person>
<person>
    <first_name>Bob</first_name>
    <last_name>Builder</last_name>
    <town>Cetown</town>
</person>
<person>
    <first_name>Nina</first_name>
    <last_name>Morning</last_name>
    <town>Detown</town>
</person>
```

HTML files can be parsed like HTML files with Beautiful Soup.

```python
import bs4

xml = '''\
<person>
    <first_name>John</first_name>
    <last_name>Miller</last_name>
    <town>Atown</town>
</person>
<person>
    <first_name>Ann</first_name>
    <last_name>Abor</last_name>
    <town>Betown</town>
</person>
<person>
    <first_name>Bob</first_name>
    <last_name>Builder</last_name>
    <town>Cetown</town>
</person>
<person>
    <first_name>Nina</first_name>
    <last_name>Morning</last_name>
    <town>Detown</town>
</person>
'''

soup = bs4.BeautifulSoup(xml)

# find all towns
towns = soup.find_all('town')
```

(continues on next page)

(continued from previous page)

```
print('#towns:', len(towns))
print('last town:', towns[-1])
```

```
#towns: 4
last town: <town>Detown</town>
```

10.7 Web Access

Today's primary source of data is the world wide web. In the simplest case we may download a data set as one single file. Many data providers instead offer an API (application programming interface) for accessing and downloading data. The worst case is if we have to scrape data from a website's HTML and other files.

10.7.1 Server, Client, Browser

Websites and other web services are hosted on a *server* somewhere in the world. If we type an URL (web address) into a browser's address bar, the browser connects to the corresponding server and asks him to send the desired file to the user's computer. This process is referred to as *requesting a file* or *sending a request*. Our computer is the *client*, asking the server for some service (send a file). It's important to understand that we cannot simply collect a file from a remote server. We only may send a request to the server to send a file to us. The server may fulfill our request or send an error message or do not answer the request at all.

The technology behind is much more involved than one might think: How to find the correct server? Which language to speak with the server? What to do if the server does not answer the request? And so on. If you are interested in some background details, use DNS[86] and HTTP[87] as entry points.

The Python interpreter may take the role of the browser and request files from servers.

10.7.2 Downloading Files with Python

To download a webpage or some other file from the the web we may use the `requests` module from the Python standard library.

The module provides a function `get` which takes the URL and yields a `Response` object. The `Response` object contains information about the server's answer to our request. If the request has been succesful, the `content` member variable contains the requested file as `bytes` object.

```
import requests

response = requests.get('https://www.fh-zwickau.de/~jef19jdw/index.html')

print(response.content.decode())
```

[86] https://en.wikipedia.org/wiki/Domain_Name_System
[87] https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

10.7.3 Web APIs

Many webpages are to some extent dynamic. Their content can be influenced by passing parameters to them. Different techniques exist for this purpose. Most common are so-called 'GET' and 'POST'. We only consider the first method here.

Passing arguments via 'GET' is very simple. We just add them to the URL. If the webpage processes arguments with names `arg1`, `arg2`, `arg3` and if we want to pass corresponding values `value1`, `value2`, `value3`, we may request the URL

```
http://some.where/some_page.html?arg1=value1&arg2=value2&arg3=value3
```

The `requests.get` function knows the keyword argument `params` to increase readability. Instead of composing a long URL string we may write:

```
url = 'http://some.where/some_page.html'
params = {'arg1': 'value1',
          'arg2': 'value2',
          'arg3': 'value3'}
response = requests.get(url, params=params)
```

Most web services for data retrieval do not return HTML document, but more machine readable formats like CSV[88], JSON[89], or YAML[90]. There are Python modules for parsing all common formats.

10.7.4 Web Scraping

Sometimes data we want to analyze is scattered over a website. No direct connection to the underlying data base is available. Thus, we have to find ways to extract data from websites automatically. The process of extracting data from websites is referred as *web scraping*.

Legal considerations

There is no law which directly prohibits web scraping. But a website or part of it may be protected by copyright law. Almost all large websites have terms of use, which have to be respected by the user. Some websites explicitly prohibit automated data extraction. Some only prohibit commercial use of the provided data. Before starting a scraping project read the terms of use!

When in doubt ask the website provider for written permission to scrape data from the site or ask a lawyer!

Another issue is the web traffic caused by scrapers. A scraping project might require several thousand requests to a server within very short time. This may hurt the providers infrastructure. A common attack for getting down a website is to send thousands of requests fast enough to prevent the server from answering requests from other users (DoS attack, denial-of-service attack). We don't want to be attackers. Thus, whenever you start a scraping project, tell your script to wait a few seconds between consecutive requests to a server!

[88] https://en.wikipedia.org/wiki/Comma-separated_values
[89] https://en.wikipedia.org/wiki/JSON
[90] https://en.wikipedia.org/wiki/YAML

Scraping Media Files from Websites

Download a webpage via `requests.get` only yields the HTML document. Images and other media usually are not contained in HTML files. To download all images of a webpage we would have to find all `img` tags in the HTML file, then extract URLs from corresponding `src` attributes, and then download each URL separately.

Useful Little Helpers

Scraping data from websites often is tedious work and each scraping project requires different techniques for data extraction. Knowing some little helpers may save the day.

Regular Expressions

There is a mini language to describe query strings for text search. Corresponding search strings are called *regular expressions*. They can be used, for instance, in conjunction with Beautiful Soup. We do not go into the details here. Just an example:

```
import re    # Python's support for regular expressions

some_string = 'banana, apple, cucumber, orange'
pattern = '[aeiou].[aeiou]'    # vowel, some letter, vowel

result = re.findall(pattern, some_string)

print(result)
```

```
['ana', 'ucu', 'ora']
```

For details and more examples see documention of `re` module[91].

Dates and Times

Most data contains time stamps. Python ships with the modules `datetime` and `time` for handlung dates and times. The former provides tools for carrying out calculations with dates and times. The latter provides different time-related functionality.

`datetime` provides objects expressing a point in time (`date`, `time`, `datetime`) and objects expressing a duration (`timedelta`).

```
import datetime

some_date = datetime.date(2020, 6, 23)
some_delta = datetime.timedelta(weeks=2)

new_date = some_date + some_delta

print(f'It\'s {new_date.day:02}.{new_date.month:02}.{new_date.year}.')
```

```
It's 07.07.2020.
```

For details see documention of `datetime` module[92].

From `time` module we might use `time.sleep` to realize some delay between subsequent requests to a server.

[91] https://docs.python.org/3/library/re.html
[92] https://docs.python.org/3/library/datetime.html

```
import time

print('Have a break...')
time.sleep(5)     # seconds
print('...now I\'m back.')
```

```
Have a break...
...now I'm back.
```

For details see documention of `time` module[93].

[93] https://docs.python.org/3/library/time.html

CHAPTER

ELEVEN

FUNCTIONS

We already met functions in the *Crash Course* (page 53). Here we repeat the basics and add lots of details important for successful Python programming.

Related exercises: *Functions* (page 873).

11.1 Basics

A function has a name, which is used to call the function. A function can take arguments to control its behavior, and a function may return a value, which then can be used by the calling code.

11.1.1 Function Definition

Functions are defined as follows in Python:

```python
def my_function(argument, another_argument, more_arguments):
    # indented block of code
```

11.1.2 Returning Values

If a function shall return a value, then its code block has to contain the following line at least once (usually the last line):

```python
return some_value
```

The return keyword immediately stops execution of the function's code and hands control back to the calling code, which then can use the content of `some_value`.

If there is no `return` keyword in a function, then the function ends after executing its last line and returns `None`. In this sense, Python functions always return a value.

11.1.3 Function Calls

A function is called es follows:

```
a = my_function(value_for_argument, value_for_another_argument, values_for_more_
↪arguments)
```

After finishing execution of the function's code `a` contains the return value of the function. If the function does not return a value or the return value is not needed by the calling code, then the assignment `a = ...` can be omitted.

11.1.4 Documentation Strings (Docstrings)

In Python by convention every function definition contains a triple quoted documentation string. This string is ignored by the Python interpreter, but read by tools for automatic generation of source code documention.

```
def my_function():
    '''Does nothing.

    Takes no arguments und returns nothing.
    '''
    # some code
```

For some formatting conventions see Python documention[94]. More details: PEP 257[95]. There are many different conventions for docstring formatting. PEP 257 is only one of them.

11.2 Passing Arguments

In contrast to several other programming languages Python provides very flexible and readable syntax constructs for passing data to functions. Here we'll also discuss what happens in memory when passing data to functions.

11.2.1 Positional Arguments

Positional arguments have to be passed in exactly the same order as they appear in the function's definition. There can be as many positional arguments as needed. But a function may come without any positional arguments at all, too.

Positional arguments may have a default value, which is used if the argument is missing in a function call. Syntax:

```
def my_function(arg1=default_value, arg2=default_value2):
    # indented block of code
```

If there are mandatory arguments (that is, without default value) and optional arguments, then the latter have to follow the former.

[94] https://docs.python.org/3/tutorial/controlflow.html#tut-docstrings
[95] https://www.python.org/dev/peps/pep-0257/

11.2.2 Keyword Arguments

If there are several optional arguments and only some shall be passed to the function, they can be provided as keyword arguments. The order of keyword arguments does not matter when calling a function.

```
def my_function(arg1=default1, arg2=default2, arg3=default3):
    # indented block of code

my_function(arg3=12345, arg2=54321)
```

Keyword arguments are very common in Python, since they increase readability when calling functions with many arguments.

11.2.3 Arbitrary Number of Positional Arguments

If we need a function which can take an arbitrary number of arguments, we may use the following snytax:

```
def my_function(arg1, arg2, *other_args):
    # indented block of code
```

Then `other_args` contains a tuple of all arguments passed to the function, but without `arg1` and `arg2`.

11.2.4 Arbitrary Number of Keyword Arguments

If we need a function which can take an arbitrary number of keyword arguments, we may use the following snytax:

```
def my_function(arg1, arg2, kwarg1=default1, kwarg2=default2, **other_kwargs):
    # indented block of code
```

Then `other_kwargs` contains a dictionary of all keyword arguments passed to the function, but without `kwarg1` and `kwarg2`.

11.2.5 Argument Unpacking

If we have a list or tuple and all items shall be passed as single arguments to a function, then we should use argument unpacking:

```
my_list = [4, 3, 1]
some_function(*my_list)
some_function(my_list[0], my_list[1], my_list[2])
```

Both calls to `some_function` are equivalent.

Same works with keyword arguments and dictionaries, where keys are argument names and values are values to be passed to the function.

```
my_dict = {'kwarg1': 3, 'kwarg2': 5, 'kwarg3': 100}
some_function(**my_dict)
```

11.2.6 Memory Management

Passing Mutable Objects

Python never copies objects passed to a function. Instead, the argument names in a function definition are tied to the objects, whose names are given in the function call.

```
def some_function(arg1, arg2):
    print(id(arg1), id(arg2))

a = 5
b = 'some string'

print(id(a), id(b))

some_function(a, b)
```

```
139792935403888 139792871198064
139792935403888 139792871198064
```

Here we have to take care: if we pass mutable objects to a function, then the function may modify these objects!

```
def clear_list(l):
    for k in range(0, len(l)):
        l[k] = 0

my_list = [2, 5, 3]

clear_list(my_list)

print(my_list)
```

```
[0, 0, 0]
```

Always look up a function's documentation if you have to pass mutable objects to a function. If the function modifies an object, this fact should be provided in the documentation. For instance, several functions of the OpenCV[96] library, which we'll use later on, modify their arguments without proper documentation.

Mutable Default Values

A similar issue arises if we use mutable objects as default values for optional arguments. The name of an optional argument is tied to the object only once during execution (at time of function definition). If this object gets modified, then the default value changes for subsequent function calls.

```
def append42(l = []):
    l. append(42)
    print(l)

append42()
append42([1, 2, 3])
append42()
append42()
```

```
[42]
[1, 2, 3, 42]
```

(continues on next page)

[96] https://opencv.org

(continued from previous page)

```
[42, 42]
[42, 42, 42]
```

To prevent such problems use a construction similar to the following:

```
def append42(l = None):
    if l == None:
        l = []
    l. append(42)
    print(l)

append42()
append42([1, 2, 3])
append42()
append42()
```

```
[42]
[1, 2, 3, 42]
[42]
[42]
```

11.2.7 Restricting Argument Passing

We may restrict a function's arguments to one of the following types:

- positional only,
- positional and keyword,
- keyword only.

For this purpose we have to add `/` and `*` to the argument list in a function's definition:

```
def my_function(pos1, pos2, /, poskw1, poskw2, *, kw1, kw2):
    # indented block of code
```

In a call to the function the first group of arguments has to be passed without keyword, the second group may be passed with or without keyword, and the third group has to be passed by keyword.

The reason for existence of this technique is quite involved and, presumably, we won't need this feature. But we should know it to understand code written by others.

11.2.8 Functions in Python's Documentation

Flexibility of argument passing makes it hard to clearly document which variants a library function accepts. Python's documentation uses a special syntax to state type and number of arguments as well as default values of a function.

Example: The `glob` module's `glob` function (see *File IO* (page 121)) is shown in Python's documentation[97] as follows:

```
glob.glob(pathname, *, root_dir=None, dir_fd=None, recursive=False)
```

We see:

- `pathname` is the only positional argument.
- There are three arguments which have to be passed by keyword.

[97] https://docs.python.org/3/library/glob.html#glob.glob

- pathname is mandatory, whereas the other arguments have default values.

Another example: The built-in input[98] function.

```
input([prompt])
```

- There is only one argument.
- The argument is optional (indicated by [...]), that is, calling without any arguments is okay.

One more: The built-in print[99] function.

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- print accepts an arbitrary number of positional arguments.
- sep, end, file, flush have to be passed by keyword (because they follow a * argument).
- The four keyword arguments are optional (have default values).

11.3 Anonymous Functions (Lambdas)

Sometimes one has to call functions which take a function as argument. Passing a function as argument is very simple, just give the function's name as argument:

```
def my_function(arg1, arg2):
    # some code

some_function(my_function)
```

Often, functions passed to other functions are needed only once in the code and almost always they have very simple structure. Providing a full function definition and wasting a name for such throwaway functions, thus, should by avoided. The tool for avoiding this overhead are anonymous functions, in Python known as *lambdas*. Here is an example:

```
some_function(lambda arg1, arg2: SOME SHORT CODE)
```

The lambda keyword creates a function in the same way as def, but without assigning a name to it. Keyword arguments are allowed, too. In principle it is possible to define named functions with lambda:

```
my_function = lambda arg1, arg2: SOME SHORT CODE
```

But this should be avoided to keep code readable.

11.4 Function and Method Objects

Everything is an object in Python, even functions and member functions.

```
def my_function():
    print('Oh, you called me!')

print(type(my_function))
print(id(my_function))
print(dir(my_function))
```

[98] https://docs.python.org/3/library/functions.html#input
[99] https://docs.python.org/3/library/functions.html#print

```
<class 'function'>
140438181160432
['__annotations__', '__builtins__', '__call__', '__class__', '__closure__', '__
↪code__', '__defaults__', '__delattr__', '__dict__', '__dir__', '__doc__', '__
↪eq__', '__format__', '__ge__', '__get__', '__getattribute__', '__globals__',
↪'__gt__', '__hash__', '__init__', '__init_subclass__', '__kwdefaults__', '__
↪le__', '__lt__', '__module__', '__name__', '__ne__', '__new__', '__qualname__
↪', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__
↪str__', '__subclasshook__']
```

```
print(my_function.__name__)
```

```
my_function
```

Member functions are bit special.

```
class my_class:
    def some_method(self, some_string):
        print('You called me with {}'.format(some_string))

my_object = my_class()

print(type(my_object.some_method))
print(type(my_class.some_method))
```

```
<class 'method'>
<class 'function'>
```

If the method `my_object.some_method` is called, then the Python interpreter inserts the owning object as first argument and calls the corresponding function `my_class.some_method`. In other words, the following two lines are equivalent:

```
my_object.some_method('Hello')
my_class.some_method(my_object, 'Hello')
```

```
You called me with Hello
You called me with Hello
```

11.5 Recursion

A useful programming technique is recursion. That is, a function calls itself until some stopping criterion is satisfied.

To illustrate this approach let's have a list. Each item either is an integer or a list again. If it is a list, then each item of this list is an integer or another list, and so on. The task is to calculate the sum of all integers. This can't be solved by nested for loops, because we do not know the depth of the list nesting in advance.

```
def sum_list(l):
    ''' Sum up list items recursively. '''

    current_sum = 0

    for k in l:
        if type(k) == int:
            current_sum += k
        else:
```

(continues on next page)

(continued from previous page)

```
            current_sum += sum_list(k)

    return current_sum


a = [[1, 2, 3], 4, [5, [6, 7], 8]]

print(sum_list(a))
```

```
36
```

CHAPTER

TWELVE

MODULES AND PACKAGES

We already met modules and packages in the *Crash Course* (page 53). Here we add the details and learn how to write new modules and packages.

12.1 Importing Modules

To get access to the functionality of a module it has to be imported into the source code file or into the interactive interpreter session:

```
import module_name
```

This creates a Python object with name `module_name` (everything is an object in Python!) whoes methods are the functions defined on the module file. All names (functions, types, and so on) defined in the module then can be accessed this way:

```
module_name.some_function()
```

Use the built-in function `dir` to get a list of all names defined in an imported module.

```
import datetime
print(dir(datetime))
```

```
['MAXYEAR', 'MINYEAR', '__all__', '__builtins__', '__cached__', '__doc__', '__
↪file__', '__loader__', '__name__', '__package__', '__spec__', 'date',
↪'datetime', 'datetime_CAPI', 'sys', 'time', 'timedelta', 'timezone', 'tzinfo']
```

A module's name can appear very often in source code. The `as` keyword allows to abbreviate module names:

```
import module_name as mod

mod.some_function()
```

It is even possible to avoid typing module names at all. If only few functions of a module are needed, then they can be imported directly:

```
from module_name import some_function, some_other_function

some_function()
some_other_function()
```

The `as` keyword can be used to abbreviate function names, too:

```
from module_name import some_function as func

func()
```

To directly import all functions from a module use `*`:

```
from module_name import *

some_function()
```

But be careful; modules may contain hundreds of functions and importing all these functions may slow down your code.

Note: The `import` statement makes the Python interpreter look for a file `module_name.py`. If it cannot find a built-in module with this name (that is, a module integrated directly into the interpreter), then it looks in the directory containing the source code file. Then several other directories are taken into account. If the interpreter does not find the requested module, an error message is shown.

12.2 Importing Packages

Packages are collections of modules and can be imported in the same way as modules. Packages may contain subpackages. An `import` statement could look like this:

```
import package_name

package_name.subpackage_name.module_name.some_function()
```

The `import` statement creates a tree of objects and subobjects which reflects the structure of the package. To import only one subpackage or one module from a package, use `from`:

```
from package_name import subpackage_name

subpackage_name.module_name.some_function()
```

or

```
from package_name.subpackage_name import module_name

module_name.some_function()
```

The placeholder `*` and renaming with `as` are available, too.

12.3 The Python Standard Library

Python ships with a large number of modules and packages, known as Python standard library. Have a look at the complete list[100] of the standard library's contents and also at Brief Tour of the Standard Library[101] as well as Brief Tour of the Standard Library - Part II[102].

We already introduced some of the standard library's modules and packages (`datetime` and `os.path` for instance) and we will continue to introduce new functionality when needed for our purposes.

[100] https://docs.python.org/3/library/index.html
[101] https://docs.python.org/3/tutorial/stdlib.html
[102] https://docs.python.org/3/tutorial/stdlib2.html

12.4 Writing New Modules

Writing our own module is very simple: Put function definitions in a file `my_module.py` and import it with `import my_module`. Writing modules makes code reusable and increases readability.

When importing a module the Python interpreter executes the module file. If you want to use a Python source code file as script as well as as module, you might check the value of the pre-defined variable `__name__`. If `__name__ == '__main__'`, then the code is being executed as a script. If `__name__ == 'module_name'`, then the code is being run due to an `import` statement.

It's also possible to use compiled modules[103].

12.5 Writing New Packages

Of course you can write your own packages. A package then is a directory `package_name` which contains the Python files for all modules in the package and in addition a file `__init__.py`. This file might be empty, but is required to mark a directory as Python package. Subpackages are subdirectories with `__init__.py` file.

For details see Python's documentation[104].

12.6 Private Members

Python does not support hidden functions or variables in modules. Also hiding members of a class from the user of the class is not possible. But sometimes this would be quite useful. Variables needed only for internal calculations or little helper functions and methods shouldn't be visible from outside, because, if they were hidden, then we could change their name or remove them completely if desired without breaking source code which uses the module or class.

In Python there is a convention to mark private members: a leading underscore like in `_hidden_by_convention`. Variables, functions and methods preceded by an underscore should not be accessed or called from outside the module or class definition. But this is a convention. Nothing prevents you from violating this convention.

[103] https://docs.python.org/3/tutorial/modules.html#compiled-python-files
[104] https://docs.python.org/3/tutorial/modules.html#packages

CHAPTER

THIRTEEN

ERROR HANDLING AND DEBUGGING OVERVIEW

Up to now we did not care about error handling. If something went wrong, the Python interpreter stopped execution and printed some message. But Python provides techniques for more controlled error handling.

13.1 Error Handling

13.1.1 Syntax Versus Runtime Errors

The Python interpreter parses the whole souce code file before execution. In this phase the interpreter may encounter *syntax errors*. That is, the interpreter does not understand what we want him to do. The code does not look like Python code should look like. Syntax errors are easily recovered by the programmer.

The more serious types of errors are *runtime errors* (or *semantic errors*) which occur during program execution. Handling runtime errors is sometimes rather difficult.

13.1.2 Handling Runtime Errors

The traditional way for handling runtime errors is to avoid runtime errors at all. All user input and all other sources of possible trouble get checked in advance by incorporating suitable `if` clauses in the code. This approach decreases readability of code, because the important lines are hidden between lots of error checking routines.

The more pythonic way of handling runtime errors are *exceptions*. Everytime the interpreter encounters some problem, like division by zero, it *throws an exception*. The programmer may *catch the exception* and handle it appropriately or the programmer may leave exception handling to the Python interpreter. In the latter case, the interpreter usually stops execution and prints a detailed error message.

13.1.3 Basic Exception Handling Syntax

Here is the basic syntax for catching and handling exceptions:

```
try:
    # code which may cause troubles
except ExceptionName:
    # code for handling a certain exception caused by code in try block
except AnotherExceptionName:
    # code for handling a certain exception caused by code in try block
else:
    # code to execute after successfully finishing try block
```

The `try` block contains the code to be protected, that is, the code which might raise an exception. Then there is at least one `except` block. The code in the `except` block is only executed, if the specified exception has been raised. In this case, execution of the `try` block is stopped immediately and execution continues in the `except` block.

There can be several `except` blocks for handling different types of exceptions. Instead of an exception name also a tuple of names can be given to handle several different exceptions in one block.

The `else` block is executed after successfully finishing the `try` block, that is, if no exception occurred. Here is the right place for code which shall only be executed if no exception occurred, but for which no explicit exception handling shall be implemented.

Here is an example:

```
a = 0    # some number from somewhere (e.g., user input)

try:
    b = 1 / a
except ZeroDivisionError:
    print('Division by zero. Setting result to 1000.')
    b = 1000    # set b to some (reasonable) value
else:
    print('Everything okay.')

print('Result is {}.'.format(b))
```

```
Division by zero. Setting result to 1000.
Result is 1000.
```

Without using exception handling the interpreter would stop execution in the division line. By catching the exception we can avoid this automatic behavior and handle the problem in a way which does not prevent further program execution.

Note that exception names are not strings, but names of object types (classes). Thus, don't use quotation marks.

```
print(type(ZeroDivisionError))
print(dir(ZeroDivisionError))
```

```
<class 'type'>
['__cause__', '__class__', '__context__', '__delattr__', '__dict__', '__dir__',
↪'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '__
↪hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__ne__', '__
↪new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__setstate_
↪_', '__sizeof__', '__str__', '__subclasshook__', '__suppress_context__', '__
↪traceback__', 'args', 'with_traceback']
```

The Python documentation contains a list of built-in exceptions[105]. There is a kind of tree structure in the set of all exceptions and we may define new exceptions if we need them to express errors specific to our program. These topics will be discussed in detail when delving deeper into object oriented programming.

13.1.4 Clean-Up

Sometimes it's necessary to do some clean-up operations like closing a file no matter an exception occurred or not while keeping the file open for reading and writing. For this purpose Python provides the `finally` keyword:

```
try:
    # code which may cause troubles
except ExceptionName:
    # code for handling a certain exception caused by code in try block
else:
    # code to execute after successfully finishing try block
finally:
    # code for clean-up operations
```

[105] https://docs.python.org/3/library/exceptions.html#concrete-exceptions

The `finally` block is executed after the `try` block (if there is no `else` block) or after the `else` block if no exception occured. If an exception occurred, then the `finally` block is executed after the corresponding `except` clause. If `try` or `except` clauses contain `break`, `continue` or `return`, then the `finally` block is executed *before* `break`, `continue` or `return`, respectively. If a `finally` block executed before `return` contains a `return` itself, then `finally`'s `return` is used and the original `return` is ignored.

Note: As long as a file is opened by our program the operating system blocks file access for other programs. Thus, we should close a file as soon as possible. Forgetting to close a file is not too bad because the OS will close it for use after program execution stopped. But for long running programs with only short file access at start-up a non-closed file may block access by other programs for hours or days. Thus, always, especially in case of exception handling, make sure that in each situation (with or without exception) files get closed properly by the program.

13.1.5 Objects With Predefined Clean-Up Actions

Some object types, file objects for instance, include predefined clean-up actions. That is, for certain operations (e.g., opening a file) they define what should be done in a corresponding `finally` block (e.g., closing the file), if the operations would be placed in a `try` block.

To use this feature Python has the `with` keyword:

```python
with open('some_file') as f:
    # do something with file object f
```

If the `open` function is successful, then the indented code block is executed. If `open` fails, an exception is raised. In both cases, `with` ensures, that proper clean-up (closing the file) takes place.

Objects which can be used with `with` are said to support the *context management protocol*. Such objects can also be defined by the programmer using dunder methods, see Python's documentation[106] for details.

The purpose of `with` is to make code more readable by avoiding too many `try...except...finally` blocks.

13.2 Logging and Debugging

Up to now we considered syntax errors, which basically are typos in the code, and semantic errors, which are caused by unexpected user input or failed file access. But code may contain more involved semantic errors, which may be hard to identify. The process of finding and correcting semantic errors is known as *debugging*.

A simple approach to debugging is to print status information during program flow. For private scripts and a data scientist's everyday use this suffices. For higher quality programs the Python standard library provides the `logging` package, which allows to redirect some of the status information to a *log file*. Logging basics are described in the basic logging tutorial[107].

If looking at log messages does not suffice, there are programs specialized to debugging your code. We do not cover this topic here. But if you are interested in you should have a look at The Python Debugger[108] and at Debugging with Spyder[109].

[106] https://docs.python.org/3/library/stdtypes.html#context-manager-types
[107] https://docs.python.org/3/howto/logging.html#basic-logging-tutorial
[108] https://docs.python.org/3/library/pdb.html
[109] https://docs.spyder-ide.org/debugging.html

13.3 Profiling

Sometimes our code does what you want it to do, but it is too slow or consumes too much memory (out of memory error from the operating system). Then it's time for profiling.

You may use the Spyder Profiler[110] or import profiling functionality from suitable Python packages.

13.3.1 Profiling Execution Time

The `timeit` module provides tools for measuring a Python script's execution time in seconds.

```
import timeit

a = 1.23

code = """\
b = 4.56 * a
"""

timeit.timeit(stmt=code, number=1000000, globals=globals())
```

```
0.06846186006441712
```

This code snipped packs some code into the string `code` and passes it to the `timeit` function. This function executes the code `number` times to increase accuracy. The built-in function `globals` returns a list of all defined names. This list should be passed to the `timeit` function to provide access to all names.

Have a look at the The Python Profilers[111], too.

Note: If working in Jupyter you may use the `%timeit` and `%%timeit`[112] magics instead of the `timeit` module, the former for timing one line of code (`%timeit one_line_of_code`), the latter for timing the whole code cell (place it in the cell's first line).

13.3.2 Profiling Memory Consumption

From data science view also memory consumption is of interest, because handling large data sets requires lots of memory. There are many ways to obtain memory information. A simple one is as follows (install module `pympler` first):

```
from pympler import asizeof

my_string = 'This is a string.'
my_int = 23

print(asizeof.asizeof(my_string))
print(asizeof.asizeof(my_int))
```

```
72
32
```

This gives the size of the memory allocated for some object. This number also includes the size of 'subobjects', that is, for example, all the objects referenced by a list object are included.

[110] https://docs.spyder-ide.org/profiler.html
[111] https://docs.python.org/3/library/profile.html
[112] https://ipython.readthedocs.io/en/stable/interactive/magics.html#magic-timeit

CHAPTER

FOURTEEN

INHERITANCE

Inheritance is an important principle in object-oriented programming. Although we may reach our aims without using inheritance in our own code, it's important to know the concept and corresponding syntax constructs to understand other people's code. We'll meet inheritance related code in

- the documentation of modules and packages,
- when customizing and extending library code.

Related exercises: *Object-Oriented Programming* (page 875).

14.1 Principles of Object-Oriented Programming

Up to now we only considered three of four fundamental OOP principles:

- *encapsulation*: code is structured into small units (write functions for different tasks and group them together with relevant data into objects)
- *abstraction*:
  - similar tasks are handled in one and the same way (instead of several unrelated objects we define a class and instantiate several objects sharing the same interface)
  - implementation details are hidden behind interfaces (a class defines an interface by providing methods and (non-private) member variables; concrete implementation of methods and usage of member variables are not of importance and invisible from outside)
- *polymorphism* (functions and method accept different sets and types of arguments, that is, interfaces are flexible; something a Python programmer does not care about because it's a very native Python feature, in contrast to C/C++, for instance)

The missing principle is

- *inheritance* (create new classes by extending existing classes)

14.2 Idea and Syntax

Inheritance is a technique to create new classes by extending and/or modifying existing ones. A new class may have a *base class*. The new class inherits all methods and member variables from its base class and is allowed to replace some of the methods and to add new ones. Syntax:

```python
class NewClass(BaseClass):

    def additional_method(self, args):
        # do something

    def replacement_for_base_class_method(self, args):
        # do something
```

The only difference compared to usual class definitions is in the first line, where a base class can be specified. Defining methods works as before. If the method name does not exist in the base class, then a new method is created. If it already exists in the base class, the new one is used instead of the base class' method. In addition to explicitly defined methods, the new class inherits all methods from the base class.

Inheritance saves time for implementation and leads to a well structured class hierachy. Object-oriented programming is not solely about defining classes (encapsulation and abstraction), but also about defining meaningful relations between classes, thus, to some extent mapping real world to source code.

14.3 Example

Real-life examples of inheritance often are quite involved. For illustration we use a pathological example resampling relations between geometric objects.

Imagine a vector drawing program. Each geometric object shall be represented as object of a corresponding class. Say quadrangles are objects of type `Quad`, paraxial rectangles are objects of type `ParRect` and so on. Let's start with class `Point`:

```python
class Point:
    ''' represent a geometric point in two dimensions '''

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f'({self.x}, {self.y})'
```

Now we define `Quad`:

```python
class Quad:
    ''' represent a quadrangle '''

    def __init__(self, a, b, c, d):
        ''' make a quad from 4 Point objects '''
        self._a = a
        self._b = b
        self._c = c
        self._d = d

    def get_points(self):
        return (self._a, self._b, self._c, self._d)

    def __str__(self):
        return f'quadrangle with points' \
               f'({self._a.x}, {self._a.y}), ({self._b.x}, {self._b.x}), ' \
               f'({self._c.x}, {self._c.x}), ({self._d.x}, {self._d.x})'
```

The member variables `_a`, `_b`, `_c`, `_d` are hidden since we consider them implementation details. If the user wants access to the four points making the quadrangle, `get_points` should be called. This way we are free to store the quadrangle in a different format if it seems resonable in future when extending class' functionality. This is a design decision and is in no way related to inheritance.

Here comes `ParRect`: Note that a paraxial rectangle is defined by two Points.

```python
class ParRect(Quad):

    def __init__(self, a, c):
        ''' make a paraxial rect from two points '''
```

(continues on next page)

(continued from previous page)

```
        b = Point(c.x, a.y)
        d = Point(a.x, c.y)
        super().__init__(a, b, c, d)

    def __str__(self):
        return f'paraxial rect with points ({self._a.x}, {self._a.y}), ({self._c.
↪x}, {self._c.x})'

    def area(self):
        ''' return the rect's area '''
        return abs(self._b.x - self._a.x) * abs(self._d.y - self._a.y)
```

The `ParRect` class inherits everything from `Quad`. It has a new constructor with fewer arguments than in `Quad`, but calls the constructor of `Quad`.

Important: The built-in function `super` in principle returns `self` (that is, the current object), but redirects method calls to the base class.

We reimplement `__str__` and add the new method `area`. Note that `ParRect` objects have member variables `_a`, `_b`, `_c`, `_d` since those are created by the `Quad` constructor we call in the `ParRect` constructor. Also the `get_points` method is a member of `ParRect` since it gets inherited from `Quad`.

```
parrect = ParRect(Point(0, 0), Point(2, 1))

print(parrect)

print('area: {}'.format(parrect.area()))

a, b, c, d = parrect.get_points()
print('all points:', a, b, c, d)
```

```
paraxial rect with points (0, 0), (2, 2)
area: 2
all points: (0, 0) (2, 0) (2, 1) (0, 1)
```

14.4 Type Checking

Note that `isinstance` also returns `True` if we check against a base class of an object's class. In other words, each object is an instance of its class and of all base classes.

```
print(isinstance(parrect, ParRect))
print(isinstance(parrect, Quad))
```

```
True
True
```

In contrast, type checking with `type` returns `False` if checked against the base class:

```
print(type(parrect) == ParRect)
print(type(parrect) == Quad)
```

```
True
False
```

14.5 Every Class is a Subclass of `object`

In Python there is a built-in class `object` and every newly created class automatically becomes a subclass of `object`. The line

```
class my_new_class:
```

is equivalent to

```
class my_new_class(object):
```

and also to

```
class my_new_class():
```

by the way.

To see this in code we might use the built-in function `issubclass`. This function returns `True` if the first argument is a subclass of the second.

```
class MyClass:

    def __init__(self):
        print('Here is __init__()!')

print(issubclass(MyClass, object))
```

```
True
```

Alternatively, we may have a look at the `__base__` member variable, which stores the base class:

```
print(MyClass.__base__)
```

```
<class 'object'>
```

Objects of type `object` do not have real functionality. The `object` class provides some auxiliary stuff used by the Python interpreter for managing classes and objects.

```
obj = object()
dir(obj)
```

```
['__class__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__ne__',
 '__new__',
```

(continues on next page)

(continued from previous page)

```
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__']
```

14.6 Virtual Methods

Python does not allow for directly implementing so called *virtual methods*. A virtual method is a method in a base class which has to be (re-)implemented by each subclass. The typical situation is as follows: The base class implements some functionality, which for some reason has to call a method of a subclass. How to guarantee that the subclass provides the required method?

In Python a virtual method is a usual method which raises a `NotImplementedError`, a special exception type like `ZeroDivisionError` and so on. If everything is correct, this never happens, because the subclass overrides the base class' method. But if the creator of the subclass forgets to implement the method required by the base class, an error message will be shown.

14.7 Multiple Inheritance

A class may have several base classes. Just provide a tuple of base classes in the class definition:

```
class my_class(base1, base2, base3):
```

The new class inherits everything from all its base classes.

If two base classes provide methods with identical names, the Python interpreter has to decide which one to use for the new class. There is a well-defined algorithm for this decision. If you need this knowledge someday, watch out for *method resolution order (MRO)*.

14.8 Exceptions Inherit from `Exception`

Up to now we used built-in exceptions only, like `ZeroDivisionError`. But now we have gathered enough knowledge to define new exceptions. Exeptions are classes as we noted before. Each exception is a direct or indirect subclass of `BaseException`. Almost all exceptions also are a subclass of `Exception`, which itself is a direct subclass of `BaseException`. See Exception hierarchy[113] for exceptions' genealogy.

If we want to introduce a new exception, we have to create a new subclass of `Exception`.

```
class SomeError(Exception):

    def __init__(self, message):
        self.message = message

def my_function():
    print('I do something...')
    raise SomeError('Meaty error message!!!')

print('Entering my_function...')
```

(continues on next page)

[113] https://docs.python.org/3/library/exceptions.html#exception-hierarchy

(continued from previous page)

```
try:
    my_function()
except SomeError as error:
    print('Exception SomeError: {}'.format(error.message))
```

```
Entering my_function...
I do something...
Exception SomeError: Meaty error message!!!
```

At first we define a new exception class `SomeError`. The constructor takes an error message and stores it in the member variable `message`. The function `my_function` raises `SomeError`. The main program catches this exception and prints the error message. The `as` keyword provides access to a concrete `SomeError` object containing the error message.

Note that `except SomeBaseClass` also catches all subclasses of `SomeBaseClass`. If we want to handle a subclass exception separately we have to place its `except` line above the base class's `except` line. Contrary, a subclass `except` never handles a base class exception.

CHAPTER

FIFTEEN

FURTHER PYTHON FEATURES

Python provides much more features than we need. Here we list some of them, which might be of interest either because they simplify some coding tasks or because they frequently occur in other people's code.

15.1 Doing Nothing With `pass`

Python does not allow emtpy functions, classes, loops and so on. But there is a 'do nothing' command, the `pass`[114] keyword.

```python
def do_nothing():
    pass

do_nothing()
```

15.2 Checking Conditions With `assert`

Especially for debugging purposes placing multiple `if` statements can be avoided by using `assert`[115] instead. The condition following `assert` is evaluated. If the result is `False` an `AssertionError` is raised, else nothing happens. An optional error message is possible, too.

```python
a = 0   # some number from somewhere (e.g., user input)
assert a != 0, 'Zero is not allowed!'
b = 1 / 0
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Input In [2], in <cell line: 3>()
      1 a = 0   # some number from somewhere (e.g., user input)
----> 3 assert a != 0, 'Zero is not allowed!'
      5 b = 1 / 0

AssertionError: Zero is not allowed!
```

[114] https://docs.python.org/3/reference/simple_stmts.html#the-pass-statement
[115] https://docs.python.org/3/reference/simple_stmts.html#the-assert-statement

15.3 Structural Pattern Matching

Python 3.10 introduces two new keywords: `match` and `case`. In the simplest case they can be used to replace `if...elif...elif...else` constructs for discrimating many cases. But in addition they provide a powerful pattern matching mechanism. See PEP 636 - Structural Pattern Matching: Tutorial[116] for an introduction.

15.4 The `set` Type

Python knows a data type for representing (mathematical) sets. Its name is `set`. Typical operations like unions and intersections of sets are supported. The official Python Tutorial[117] shows basic usage.

15.5 Function Decorators

Function decorators are *syntactic suggar* (that is, not really needed), but very common. They precede a function definition and consist of an `@` character and a function name. Function decorators are used to modify a function by applying another function to it. The following two code cells are more or less equivalent:

```python
def double(func):
    return lambda x: 2 * func(x)

@double
def calc_something(x):
    return x * x
```

```python
def double(func):
    return lambda x: 2 * func(x)

def calc_something(x):
    return x * x

calc_something = double(calc_something)
```

See Function definitions[118] in Python's documention for details.

15.6 The `copy` Module

The `copy` module[119] provides functions for shallow and deep copying of objects. For discussion of the copy problem in the context of lists see *Multiple Names and Copies* (page 106).

[116] https://peps.python.org/pep-0636/
[117] https://docs.python.org/3/tutorial/datastructures.html#sets
[118] https://docs.python.org/3/reference/compound_stmts.html#function-definitions
[119] https://docs.python.org/3/library/copy.html

15.7 Multitasking

Reading in large data files or downloading data from some server may take a while. During this time the CPU is more or less idle and could do some heavy computations without slowing down data transfer. This a typical situation where one wants to have two Python programs or two parts of one and the same program running in parallel, possibly communicating with each other.

Python has the `threading`[120] module for real multitasking on operating system level. The `asynio`[121] module in combination with the `async` and `await` keywords provides a simpler multithreading approach completely controlled by the Python interpreter.

15.8 Graphical User Interfaces (GUIs)

The Python ecosystem provides lots of packages for creating and controlling graphical user interfaces (GUIs). Here are some widely used ones:

- `tkinter`[122] in Python's standard library provides support for classical desktop applications with a main window, subwindows, buttons, text fields, and so on.
- `ipywidgets`[123] is a very easy to use package for creating graphical user interfaces in Jupyter notebooks. For instance a slider widget could control some parameter of an algorithm.
- `flask`[124] is a package for building web apps with Python, that is, the user interacts via a website with the Python program running on a server.

[120] https://docs.python.org/3/library/threading.html
[121] https://docs.python.org/3/library/asyncio.html
[122] https://docs.python.org/3/library/tkinter.html
[123] https://ipywidgets.readthedocs.io
[124] https://flask.palletsprojects.com

Part IV

Managing Data with Python

CHAPTER

SIXTEEN

EFFICIENT COMPUTATIONS WITH NUMPY

Almost all data comes as tables with lots of numbers. NumPy[125] is a Python package for effciently handling large tables of numbers. NumPy also provides advanced and very efficient linear algebra operations on such tables, for instance solving systems of linear equations based on the data. Most machine learning algorithms boil down to moving large amounts of data and doing some linear algebra. Thus, it's a good idea to spend some time understanding NumPy's basic principles and discovering NumPy's functionality.

Related exercises:

- *NumPy Basics* (page 877)
- *Image Processing with NumPy* (page 879)

16.1 NumPy Arrays

NumPy provides two fundamental tools for data science purposes:

- a data type for storing tabular numerical data,
- very efficient functions for computations on large amounts of numerical data.

NumPy's basic data type is called `ndarray` (n-dimensional array), often called *NumPy array*.

NumPy's standard abbreviation is `np`.

```
import numpy as np
```

[125] https://numpy.org

16.1.1 Python Lists versus NumPy Arrays

From mathematics we know *Vectors* (page 1021) and *Matrices* (page 1022). A vector is a (one-dimensional) list of numbers. A matrix is a (two-dimensional) field of numbers. Vectors could be represented by lists in Python, whereas a matrix would be a list if lists (a list of rows or a list of columns).

Using Python lists for representing large vectors and matrices is very inefficient. Each item of a Python list has its own location somewhere in memory. When reading a whole list, to multiply a vector by some number, for instance, Python reads the first list item, then looks for the memory location of the second, then reads the second, and so on. A lot of memory management is involved.

To significantly improve performance, NumPy provides the `ndarray` data type. The most important property of an `ndarray` is its dimension. A one-dimensional array stores a vector. A two-dimensional array stores a matrix. Zero-dimensional arrays store nothing, but are valid Python objects. Visualization of arrays with dimension above two is somewhat difficult. A three-dimensional array can be visualized as cuboid of numbers, each number described by three indices (row, column, depth level). We will meet dimensions of three and above almost every day when diving into machine learning. One example are color images: two-dimensions for pixel positions, one dimension for color channels (red, green, blue, transparency).

Why are NumPy arrays more efficient?

- All items of a NumPy array have to have **identical data type**, mostly float or integer. This saves time and memory for handling different types and type conversions.
- All items of a NumPy array are stored in a **well-structured contiguous block of memory**. To find the next item or to copy a whole array or part of it much less memory management operations are required.
- NumPy provides **optimized mathematical operations for vectors and matrices**. Instead of processing arrays item by item, NumPy functions take the whole array and process it in compiled C code. Thus, the item-by-item part is not done by the (slow) Python interpreter, but by (very fast) compiled code.

16.1.2 Creating NumPy Arrays

Converting Python Lists to Arrays

There are several ways to create NumPy arrays. We start with conversion of Python lists or tuples by NumPy's `array` function.

Passing a list or a tuple to `array` yields a one-dimensional `ndarray`. The data type is determined by NumPy to be the simplest type which can hold all objects in the list or tuple.

```
a = np.array([23, 42, 7, 4, -2])

print(a)
print(a.dtype)
```

```
[23 42  7  4 -2]
int64
```

The member variable `ndarray.dtype` contains the array's data type. Here NumPy decided to use `int64`, that is, integers of length 8 byte. Available types will be discussed below. An example with floats:

```
b = np.array([2.3, 4.2, 7, 4, -2])

print(b)
print(b.dtype)
```

```
[ 2.3  4.2  7.   4.  -2. ]
float64
```

Important: NumPy ships with its own data types for numbers to allow for more efficient storage and computations. Python's `int` type allows for arbitrarily large numbers, whereas NumPy has different types for integers with different (and finite!) numerical ranges. NumPy also knows several types of floats differing in precision (number of decimal places) and range. Wherever possible conversion between Python types and NumPy types is done automatically.

Higher-Dimensional Arrays from Lists

To get higher-dimensional arrays use nested lists:

```
c = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])

print(c)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

New Arrays as Return Values

Next to explicit creation, NumPy arrays may be the result of mathematical operations:

```
d = a + b

print(type(d))
```

```
<class 'numpy.ndarray'>
```

To see that `d` is indeed a new `ndarray` and not an in-place modified `a` or `b`, we might look at the object ids, which are all different:

```
print(id(a), id(b), id(d))
```

```
140549050909872 140549364309744 140549050912656
```

Functions for Creating Special Arrays

A third way for creating NumPy arrays is to call specific NumPy functions returning new arrays. From `np.zeros` we get an array of zeros. From `np.ones` we get an array of ones. There are much more functions like `zeros` and `ones`, see Array creation routines[126] in Numpy's documentation.

```
a = np.zeros(5)
b = np.ones((2, 3))

print(a, '\n')
print(b)
```

```
[0. 0. 0. 0. 0.]

[[1. 1. 1.]
 [1. 1. 1.]]
```

[126] https://numpy.org/doc/stable/reference/routines.array-creation.html#routines-array-creation

NumPy almost always defaults to floats if no data type is explicitly provided.

16.1.3 Properties of NumPy Arrays

Objects of type `ndarray` have several member variables containing important information about the array:

- `ndim`: number of dimensions,
- `shape`: tuple of length `ndim` with array size in each dimension,
- `size`: total number of elements,
- `nbytes`: number of bytes occupied by the array elements,
- `dtype`: the array's data type.

```
a = np.zeros((4, 3))

print(a.ndim)
print(a.shape)
print(a.size)      # 4 * 3
print(a.nbytes)    # 4 * 3 * 8
print(a.dtype)
```

```
2
(4, 3)
12
96
float64
```

It's important to know that `shape` matters. In mathematics almost always we identify vectors with matrices having only one column. But in NumPy these are two different things. A vector has shape `(n, )`, that is `ndim` is 1, whereas a one-column matrix has shape `(n, 1)` with `ndim` of 2. Consequently, a vector neither is a row nor a column in NumPy. It's simply a list of numbers, nothing more.

```
a = np.zeros(5)
b = np.zeros((5, 1))
c = np.zeros((1, 5))

print(a, '\n')
print(b, '\n')
print(c)
```

```
[0. 0. 0. 0. 0.]

[[0.]
 [0.]
 [0.]
 [0.]
 [0.]]

[[0. 0. 0. 0. 0.]]
```

16.1.4 List-Like Indexing

Elements of NumPy arrays can be accessed similarly to items of Python lists. That is, the first item in a one-dimensional `ndarray` has index 0 and the last one has index `ndarray.size`. Slicing is allowed, too.

```
a = np.array([23, 42, -7, 3, 10])

print(a[0], '\n')
print(a[1:3], '\n')
print(a[::2])
```

```
23

[42 -7]

[23 -7 10]
```

In case of multi-dimensional arrays we have to provide an index for each dimension. Slicing is done per dimension.

```
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(a, '\n')

print(a[0, 1], '\n')
print(a[1:3, 0:2], '\n')
print(a[::2, ::2], '\n')
print(a[1, :])
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

2

[[4 5]
 [7 8]]

[[1 3]
 [7 9]]

[4 5 6]
```

Here `:` stands for ‘all indices of the dimension’.

Note: Selecting all elements in the last dimensions like in `a[1, :]` can be abbreviated to `a[1]`. Same holds for higher dimensions: `a[1, 3, :, :]` is equivalent to `a[1, 3]`. The drawback is that one doesn’t see immediately the array’s dimensionality.

16.1.5 Data Types

NumPy knows many different numerical data types. Often we do not have to care about types (NumPy will choose suitable ones), but sometimes we have to specify data types explicitly (see examples below).

Almost all NumPy functions accept the keyword argument `dtype` to specify the data type of the function's return value. Either pass a string with the desired type's name or pass a `type` object. Passing Python types like `int` makes NumPy choose the most appropriate NumPy type (here, `np.int64` or the string `'int64'`).

```
a = np.zeros((2, 3))
b = np.zeros((2, 3), dtype=np.int64)

print(a, '\n')
print(b)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]

[[0 0 0]
 [0 0 0]]
```

NumPy types for integers:

- `np.int8`, `np.int16`, `np.int32`, `np.int64` (signed integers of different range),
- `np.uint8`, `np.uint16`, `np.uint32`, `np.uint64` (unsigned integers of different range),

NumPy types for floats:

- `np.float16`, `np.float32`, `np.float64` (different precision and range)

For booleans there is `np.bool8`, which is very similar to Pythons `bool` (both using 8 times as much memory as required).

Types for complex numbers are available, too. See Built-in scalar types[127] in NumPy's documentation for details.

Hint: The `dtype` member of `ndarrays` and the `dtype` argument to NumPy functions carry more information than the bare type (e.g., 'signed integer of length 64 bits'). They also contain information about how data is organized in memory. This is important for efficient import of data from external sources. Details will be discussed in *Saving and Loading Non-Standard Data* (page 191).

Example: Saving Memory by Manually Choosing Types

Working with large NumPy arrays we have to save memory wherever possible. One important ingredient for memory efficiency is choosing small types, that is, types with small range. Often we work with arrays of zeros and ones or of small integers only. Then we should choose the smallest integer type:

```
a = np.ones(1000)     # defaults to np.int64
b = np.ones(1000, dtype=np.int8)

print(f'a has {a.nbytes} bytes')
print(f'b has {b.nbytes} bytes')
```

```
a has 8000 bytes
b has 1000 bytes
```

Having a data set with one billion numbers choosing the correct type decides about requiring 1 GB or 8 GB of memory!

[127] https://numpy.org/doc/stable/reference/arrays.scalars.html#built-in-scalar-types

Example: Unsuitable Default Type

Creating an array without explicitly providing a data type makes NumPy choose `np.int64` or `np.float64` depending on the presence of floats. This may lead to hard to find errors:

```
a = np.array([2, 4, 6, 1])     # defaults to np.int64
a[0] = 1.23
a[3] = 0.99

print(a)
```

```
[1 4 6 0]
```

Modifying values in integer arrays converts the new values to the array's data type, even if information will be lost. To avoid such errors always specify types if working with floats!

16.2 Array Operations

```
import numpy as np
```

All Python operators can be applied to NumPy arrays, where **all operations work elementwise**.

16.2.1 Mathematical Operations

For instance, we can easily add two vectors or two matrices by using the + operator.

```
a = np.array([1, 2, 3])
b = np.array([9, 8, 7])

c = a + b

print(c)
```

```
[10 10 10]
```

Important: Because all operations work elementwise, the `*` operator on two-dimensional arrays does NOT multiply the two matrices in the mathematical sense, but simply yields the matrix of elementwise products. Mathematical matrix multiplication will be discussed in *Linear Algebra Functions* (page 187).

NumPy reimplements almost all mathematical functions, like sine, cosine and so on. NumPy's functions take arrays as arguments and apply the mathematical functions elementwise. Have a look at Mathematical functions[128] in NumPy's documentation for a list of available functions.

```
a = np.array([1, 2 , 3])

b = np.sin(a)

print(b)
```

```
[0.84147098 0.90929743 0.14112001]
```

[128] https://numpy.org/doc/stable/reference/routines.math.html

Hint: Functions `min` and `amin` are equivalent. The `amin` variant exists to avoid confusion and name conflicts with Python's built-in function `min`. Writing `np.min` is okay.

Important: Functions `np.min` and `np.minimum` do different things. With `min` we get the smallest value in an array, whereas `minimum` yields the elementwise minimum of two equally sized arrays. With `np.argmin` we get the index (not the value) of the minimal element of an array.

16.2.2 Comparing Arrays

Comparisons work elementwise, too.

```
a = np.array([1, 2, 3])
b = np.array([-1, 3, 3])

print(a > b)
```

```
[ True False False]
```

Comparisons result in NumPy arrays of data type `bool`. The function `np.any` returns `True` if and only if at least one item of the argument is `True`. The function `np.all` returns `True` if and only if all items of the argument are `True`.

```
a = np.array([True, True, False])

print(np.any(a))
print(np.all(a))
```

```
True
False
```

Some of NumPy's functions also are accessible as methods of `ndarray` objects. Examples are `any` and `all`:

```
a = np.array([True, True, False])

print(a.any())
print(a.all())
```

```
True
False
```

Due to Python's internal workings for processing logical expressions efficiently it's not possible to redefine `and` and friends via dunder methods. Thus, there's no chance for NumPy to define it's own variant of `and`. There is a dunder method `__and__`, but that implements bitwise 'and' (Python operator `&`), which is something different than logical 'and'.

To combine several conditions you might use `logical_and`[129] and friends. Using Python's `and` (and friends) results in an error, because Python tries to convert a NumPy array to a `bool` value and it's not clear how to do this (any or all?).

[129] https://numpy.org/doc/stable/reference/generated/numpy.logical_and.html

16.2.3 Broadcasting

If dimensions of the operands of a binary operation do not fit (short vector plus long vector, for instance) an exception is raised. But in some cases NumPy uses a technique called *broadcasting* to make dimensions fit by cloning suitable subarrays. Examples:

```
a = np.array([[1, 2, 3]])     # 1 x 3
b = np.ones((4, 3))           # 4 x 3

c = a + b

print(a, '\n')
print(b, '\n')
print(c)
print(c.shape)
```

```
[[1 2 3]]

[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

[[2. 3. 4.]
 [2. 3. 4.]
 [2. 3. 4.]
 [2. 3. 4.]]
(4, 3)
```

```
a = np.array([[1, 2, 3]])               # 1 x 3
b = np.array([[1], [2], [3], [4]])      # 4 x 1

c = a + b

print(a, '\n')
print(b, '\n')
print(c, '\n')
print(c.shape)
```

```
[[1 2 3]]

[[1]
 [2]
 [3]
 [4]]

[[2 3 4]
 [3 4 5]
 [4 5 6]
 [5 6 7]]

(4, 3)
```

```
a = np.array([1, 2, 3])     #     3 (one-dimensional)
b = np.ones((4, 3))         # 4 x 3

c = a + b

print(a, '\n')
```

(continues on next page)

(continued from previous page)

```
print(b, '\n')
print(c)
print(c.shape)
```

```
[1 2 3]

[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

[[2. 3. 4.]
 [2. 3. 4.]
 [2. 3. 4.]
 [2. 3. 4.]]
(4, 3)
```

Broadcasting follows two simple rules:

- If one array has fewer dimensions than the other, add dimensions of size 1 till dimensions are equal.
- Compare array sizes in each dimension. If they equal, do nothing. If they differ and one array has size 1 in the dimension under consideration, clone the array with size 1 sufficiently often to fit the other array's size.

Broadcasting makes life much easier. On the one hand it allows for operations where one operand is a scalar:

```
a = np.array([[1, 2, 3], [4, 5, 6]])     # 2 x 3
b = 7                                     #     1 (one-dimensional)

c = a + b

print(a, '\n')
print(b, '\n')
print(c)
print(c.shape)
```

```
[[1 2 3]
 [4 5 6]]

7

[[ 8  9 10]
 [11 12 13]]
(2, 3)
```

On the other hand broadcasting allows for efficient column or row operations:

```
# multiply columns by different values

a = np.array([[1, 2, 3], [4, 5, 6]])     # 2 x 3
b = np.array([[0.5], [2]])                # 2 x 1

c = a * b

print(a, '\n')
print(b, '\n')
print(c)
print(c.shape)
```

```
[[1 2 3]
 [4 5 6]]

[[0.5]
 [2. ]]

[[ 0.5  1.   1.5]
 [ 8.  10.  12. ]]
(2, 3)
```

For higher-dimensional examples have a look at Broadcasting[130] in NumPy's documentation.

16.3 Advanced Indexing

NumPy supports different indexing techniques for accessing subarrays. We already discussed list-like indexing. Now we add boolean and integer indexing.

```
import numpy as np
```

16.3.1 Boolean Indexing

If the index to an array is a boolean array of the same shape as the indexed array, then a one-dimensional array of all items where the index is `True` is returned.

```
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
idx = np.array([[True, True, False], [False, True, True], [True, False, False]])

b = a[idx]

print(a, '\n')
print(idx, '\n')
print(b)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]

[[ True  True False]
 [False  True  True]
 [ True False False]]

[1 2 5 6 7]
```

A typical use case are elementwise bounds:

```
a = np.array([1, 4, 3, 5, 7, 6, 3, 2, 4, 5, 6, 7, 4, 1, 9])

b = a[a > 3]

print(b)
```

```
[4 5 7 6 4 5 6 7 4 9]
```

[130] https://numpy.org/doc/stable/user/basics.broadcasting.html

Here b is an array containing all numbers of a which are greater than 3. The comparison a > 3 returns a boolean array of the same shape as a. Note, that broadcasting is used to compare an array to a number. The resulting boolean array then is used as index to a.

16.3.2 Integer Indexing

Given an array we may provide an array of indices. The result of the corresponding indexing operation is an array of the same size as the index array, but with the items of the indexed array at corresponding positions.

```
a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
idx = np.array([[0, 5], [2, 3]])

print(a[idx])
```

```
[[1 6]
 [3 4]]
```

For indexing multi-dimensional arrays we need multiple index arrays (one per dimension).

```
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
idx0 = np.array([[0, 0], [2, 2]])
idx1 = np.array([[0, 2], [1, 0]])

print(a[idx0, idx1])
```

```
[[1 3]
 [8 7]]
```

16.4 Vectorization

NumPy is very fast if operations are applied to whole arrays instead of element-by-element. Thus, we should try to avoid iterating through array elements and processing single elements. This idea is known as *vectorization*.

```
import numpy as np
```

16.4.1 Example: Vectorization via Indexing

Imagine we have a vector of length n, where n is even. We would like to interchange each number at an even index with its successor. The result shall be stored in a new array.

Here is the code based on a loop:

```
def interchange_loop(a):
    result = np.empty_like(a)
    for k in range(0, int(a.size / 2)):
        result[2 * k] = a[2 * k + 1]
        result[2 * k + 1] = a[2 * k]
    return result

print(interchange_loop(np.array([1, 2, 3, 4, 5, 6, 7, 8])))
```

```
[2 1 4 3 6 5 8 7]
```

And here with vectorization:

```
def interchange_vectorized(a):
    result = np.empty_like(a)
    result[0::2] = a[1::2]
    result[1::2] = a[0::2]
    return result

print(interchange_vectorized(np.array([1, 2, 3, 4, 5, 6, 7, 8])))
```

```
[2 1 4 3 6 5 8 7]
```

Now let's look at the execution times:

```
%%timeit
interchange_loop(np.zeros(1000))
```

```
178 µs ± 5.66 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

```
%%timeit
interchange_vectorized(np.zeros(1000))
```

```
2.96 µs ± 266 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

The speed-up is of a factor of almost 75 for $n = 1000$ and becomes much better if n grows.

16.4.2 Example: Vectorization via Broadcasting

Given two lists of numbers we want to have a two-dimensional array containing all products from these numbers.

Here is the code based on a loop:

```
def products_loop(a, b):
    result = np.empty((len(a), len(b)))
    for i in range(0, len(a)):
        for j in range(0, len(b)):
            result[i, j] = a[i] * b[j]
    return result

print(products_loop(range(1000), range(1000)))
```

```
[[0.00000e+00 0.00000e+00 0.00000e+00 ... 0.00000e+00 0.00000e+00
  0.00000e+00]
 [0.00000e+00 1.00000e+00 2.00000e+00 ... 9.97000e+02 9.98000e+02
  9.99000e+02]
 [0.00000e+00 2.00000e+00 4.00000e+00 ... 1.99400e+03 1.99600e+03
  1.99800e+03]
 ...
 [0.00000e+00 9.97000e+02 1.99400e+03 ... 9.94009e+05 9.95006e+05
  9.96003e+05]
 [0.00000e+00 9.98000e+02 1.99600e+03 ... 9.95006e+05 9.96004e+05
  9.97002e+05]
 [0.00000e+00 9.99000e+02 1.99800e+03 ... 9.96003e+05 9.97002e+05
  9.98001e+05]]
```

And here with vectorization:

```
def products_vectorized(a, b):
    result = np.array([a]).T * np.array([b])
    return result

print(products_vectorized(range(1000), range(1000)))
```

```
[[      0       0       0 ...       0       0       0]
 [      0       1       2 ...     997     998     999]
 [      0       2       4 ...    1994    1996    1998]
 ...
 [      0     997    1994 ... 994009 995006 996003]
 [      0     998    1996 ... 995006 996004 997002]
 [      0     999    1998 ... 996003 997002 998001]]
```

Hint: The `T` member variable of a NumPy array provides the transposed array. It's not a copy (expensive) but a view (cheap). For detail see *Linear Algebra Functions* (page 187).

Execution times:

```
%%timeit
products_loop(range(1000), range(1000))
```

```
248 ms ± 3.36 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%%timeit
products_vectorized(range(1000), range(1000))
```

```
1.25 ms ± 11.4 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Speed-up is factor 200 for lists of lenth 1000.

16.4.3 Important

Whenever you see a loop in numerical routines, spend some time to vectorize it. Almost always that's possible. Often vectorization also increases readability of the code.

16.5 Array Manipulation Functions

NumPy comes with lots of functions for manipulating arrays. Some of them are needed more often, others almost never. A comprehensive list is provided in array manipulation routines[131]. Here we only mention some of the more important ones.

```
import numpy as np
```

[131] https://numpy.org/doc/stable/reference/routines.array-manipulation.html

16.5.1 Modifying Shape with `reshape`

A NumPy array's `reshape`[132] method yields an array of different shape, but with identical data. The new array has to have the same number of elements as the old one.

```
a = np.ones(5)           # 1d (vector)
b = a.reshape(1, 5)      # 2d (row matrix)
c = a.reshape(5, 1)      # 2d (column matrix)

print(a, '\n')
print(b, '\n')
print(c)
```

```
[1. 1. 1. 1. 1.]

[[1. 1. 1. 1. 1.]]

[[1.]
 [1.]
 [1.]
 [1.]
 [1.]]
```

One dimension may be replaced by `-1` indicating that the size of this dimension shall be computed by NumPy:

```
a = np.ones((8, 8))
b = a.reshape(4, -1)

print(a.shape, b.shape)
```

```
(8, 8) (4, 16)
```

16.5.2 Mirrowing with `fliplr` and `flipud`

To mirrow a 2d array on its vertical or horizontal axis use `fliplr`[133] and `flipud`[134], respectively.

```
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.fliplr(a)

print(a, '\n')
print(b)
```

```
[[1 2 3]
 [4 5 6]]

[[3 2 1]
 [6 5 4]]
```

[132] https://numpy.org/doc/stable/reference/generated/numpy.reshape.html
[133] https://numpy.org/doc/stable/reference/generated/numpy.fliplr.html
[134] https://numpy.org/doc/stable/reference/generated/numpy.flipud.html

16.5.3 Joining Arrays with `concatenate` and `stack`

Arrays of identical shape (except for one axis) may be joined along an existing axis to one large array with `concatenate`[135].

```
a = np.ones((2, 3))
b = np.zeros((2, 5))
c = np.full((2, 2), 5)
d = np.concatenate((a, b, c), axis=1)

print(d)
```

```
[[1. 1. 1. 0. 0. 0. 0. 0. 5. 5.]
 [1. 1. 1. 0. 0. 0. 0. 0. 5. 5.]]
```

If identically shaped array shall be joined along a new axis, use `stack`[136].

```
a = np.ones(2)
b = np.zeros(2)
c = np.full(2, 5)
d = np.stack((a, b, c), axis=1)

print(d)
```

```
[[1. 0. 5.]
 [1. 0. 5.]]
```

16.5.4 Appending Data with `append`

Like Python lists NumPy arrays may be extended by appending further data. The `append`[137] method takes the original array and the new data and returns the extended array.

```
a = np.ones((3, 3))
b = np.append(a, [[1, 2, 3]], axis=0)

print(b)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 2. 3.]]
```

[135] https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html
[136] https://numpy.org/doc/stable/reference/generated/numpy.stack.html
[137] https://numpy.org/doc/stable/reference/generated/numpy.append.html

16.6 Copies and Views

NumPy arrays may be very large. Thus, having too many copies of one and the same array (or subarrays) is expensive. NumPy implements a mechanism to avoid copying arrays if not necessary by sharing data between arrays. The programmer has to take care of which arrays share data und which arrays are independent from others.

```
import numpy as np
```

16.6.1 Views

A *view* of a NumPy array is a usual `ndarray` object, that shares data with another array. The other array is called the *base* array of the view.

Views can be created with an array's `view` method. The base object is accessible through a view's `base` member variable.

```
a = np.ones((100, 100))
b = a.view()

print('id of a:', id(a))
print('id of b:', id(b))
print('base of a:', a.base)
print('id of base of b:', id(b.base))
```

```
id of a: 139825110801392
id of b: 139825110801776
base of a: None
id of base of b: 139825110801392
```

The `view` method is rarely called directly (might be used for type conversions without copying), but views frequently originate from calling shape manipulation functions like `reshape` or `fliplr`:

```
b = a.reshape(10, 1000)
c = np.fliplr(a)

print(a.shape)
print(b.shape, b.base is a)
print(c.shape, c.base is a)
```

```
(100, 100)
(10, 1000) True
(100, 100) True
```

Operations on views alter the base array's (and other view's) data:

```
b[0, 0] = 5

print(a[0, 0], b[0, 0], c[0, -1])
```

```
5.0 5.0 5.0
```

Important: Writing data to views modifies the base array! This is a common source of errors, which are very hard to track down. Always keep track of which of your arrays are views!

16.6.2 Slicing Creates Views

Views may be smaller than the original array. Such views of subarrays originate from slicing operations:

```
a = np.ones((100, 100))
b = a[4:10, :]
c = a[5]

print(a.shape)
print(b.shape, b.base is a)
print(c.shape, c.base is a)
```

```
(100, 100)
(6, 100) True
(100,) True
```

Again modifying a view alters the original array:

```
b[1, 0] = 5

print(a[5, 0], b[1, 0], c[0])
```

```
5.0 5.0 5.0
```

16.6.3 Copies

A NumPy array's `copy` method yields a (deep) copy of an array.

```
a = np.ones((100, 100))
b = a.copy()

b[0, 0] = 5

print(a[0, 0], b[0, 0])
print(b.base)
```

```
1.0 5.0
None
```

Hint: NumPy arrays are mutable objects. Thus, assigning a new name to an array or passing an array to a function does not copy the array. Keeping this in mind is very important because functions you call in your code may alter you arrays. The other way round, writing functions other people might use, clearly indicate in the documentation if your function modifies arrays passed as parameters. If in doubt, use `copy`.

16.7 Efficiency Considerations

When working with large arrays the most expensive operation (longest execution time) is copying arrays. Thus, we should avoid making copies of arrays. But there are some more things to consider when optimizing execution time and memory consumption.

```
import numpy as np
```

16.7.1 Don't Use append in Loops

Sometimes data comes in in chunks and we have to build a large array step by step. We could start with an empty array and append each new chunk of data. If incoming chunks are single numbers, code could look as follows:

```
%%timeit

a = np.array([], dtype=np.int64)

for k in range(0, 100):
    a = np.append(a, k)
```

```
396 μs ± 55.2 μs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Each call to `append` creates a new (larger) array and copies the existing one into the new one. In the end we made 100 expensive copy operations.

If we know the final size of our array in advance, then we should create an array of final size before filling it with data:

```
%%timeit

a = np.empty(100, dtype=np.int64)

for k in range(0, 100):
    a[k] = k
```

```
9.12 μs ± 165 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

We get a speed-up of factor 40, because repeated copying is avoided.

16.7.2 Append to Lists Instead of Arrays

If data comes in in chunks and we do not know the final array's size in advance, we should use a Python list for temporarily storing data. Appending to a Python list is cheap, because existing list data won't be copied. Each list item has its own (more or less random) location in memory. If data is complete, we create a NumPy array of correct size and copy the list's items to the array.

```
%%timeit

a = []
for k in range(0, 100):
    a.append(k)

b = np.array(a, dtype=np.int64)
```

```
11.5 µs ± 1.41 µs per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

Speed-up compared to `np.append` is factor 35.

16.7.3 Use Multidimensional Indices

For multidimensional arrays we have two indexing variants:

- multidimensional indexing (e.g., `a[0, 0, 0]`),
- repeated onedimensional indexing (e.g., `a[0][0][0]`).

The latter creates a lower-dimensional slice `a[0]`, then indexes this slice, creating another slice, and so on. This process is less efficient than using multidimensional indices.

```
%%timeit

a = np.ones((10, 10, 10))
for k in range(0, 100):
    b = a[0][0][0]
```

```
37.3 µs ± 5.83 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

```
%%timeit

a = np.ones((10, 10, 10))
for k in range(0, 100):
    b = a[0, 0, 0]
```

```
13.2 µs ± 344 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

Speed-up is almost factor 3.

16.7.4 Remove Unused Arrays from Memory

Working with large arrays we should free memory as soon as possible (use `del`). A non-obvious situation, where memory can be freed, is when using small views of large arrays. Consider the following code:

```
a = np.ones((100, 100))    # large array resulting from some computation
b = a[0, :]    # we only need the first row
del a
```

Here the large array remains in memory although we only need the first row. Because the view `b` is based on the array object `a`, `del a` only removes the name `a`, but garbage collection cannot remove the array object. More efficient code:

```
a = np.ones((100, 100))    # large array resulting from some computation
b = a[0, :].copy()    # we only need the first row, make a copy
del a    # remove large array from memory
```

Here only the first (copied) row remains in memory. The original large array will be removed from memory by Python's garbage collection as soon as possible.

16.8 Special Floats

NumPy does not raise an exception if we use mathematical operations which lead to undefined results. Instead, NumPy prints a warning and returns one of several special floating point numbers.

```
import numpy as np
```

16.8.1 Infinity

Results of some undefined operations can be interpreted as plus or minus infinity. NumPy represents infinity by the special float `np.inf`:

```
a = np.array([1, -1])
b = a / 0

print(b)
print(type(b[0]))
```

```
[ inf -inf]
<class 'numpy.float64'>
```

```
/tmp/ipykernel_69952/2743290674.py:2: RuntimeWarning: divide by zero␣
↪encountered in true_divide
  b = a / 0
```

If there is some good reason we may also use `np.inf` in assignments:

```
a = np.inf

print(a)
```

```
inf
```

Also well defined operations may lead to infinity if the range of floats is exhausted:

```
print(np.array([1.23]) ** 10000)
```

```
[inf]
```

```
/tmp/ipykernel_69952/1071891269.py:1: RuntimeWarning: overflow encountered in␣
↪power
  print(np.array([1.23]) ** 10000)
```

Calculations with `np.inf` behave as expected:

```
print(np.inf + 1)
print(5 * np.inf)
print(0 * np.inf)
print(np.inf - np.inf)
```

```
inf
inf
nan
nan
```

16.8.2 Not a Number

Results of undefined operations which cannot be interpreted as infinity are represented by the special float `np.nan` (**n**ot **a** **n**umber).

```
a = np.array([-1, 1])
b = np.log(a)

print(b)
```

```
[nan  0.]
```

```
/tmp/ipykernel_69952/3972426452.py:2: RuntimeWarning: invalid value encountered␣
↪in log
  b = np.log(a)
```

Calculations with `np.nan` always lead to `np.nan`:

```
print(np.nan + 1)
print(5 * np.nan)
print(0 * np.nan)
```

```
nan
nan
nan
```

16.8.3 Comparison of and to Special Floats

Don't use usual comparison operators for testing for special floats. They show strange (but well-defined) behavior:

```
print(np.nan == np.nan)
print(np.inf == np.inf)
```

```
False
True
```

Instead, call `np.isnan`[138] or `np.isposinf`[139] (for $+\infty$) or `np.isneginf`[140] (for $-\infty$) or `np.isinf`[141] (for both).

Comparisons between finite numbers and `np.inf` are okay:

```
print(5 < np.inf)
```

```
True
```

[138] https://numpy.org/doc/stable/reference/generated/numpy.isnan.html
[139] https://numpy.org/doc/stable/reference/generated/numpy.isposinf.html
[140] https://numpy.org/doc/stable/reference/generated/numpy.isneginf.html
[141] https://numpy.org/doc/stable/reference/generated/numpy.isinf.html

16.8.4 NumPy Functions Handling `np.nan`

Some NumPy function come in two variants, which differ in handling `np.nan`. Look at amin[142] and nanmin[143], for instance.

16.9 Linear Algebra Functions

NumPy has a submodule `linalg` for functions related to linear algebra, but the base `numpy` module also contains some linear algebra functions. Linear algebra[144] in NumPy's documentation provides a comprehensive list of functions. Here we only provide few examples.

```
import numpy as np
```

For mathematical definitions see *Linear Algebra* (page 1021).

16.9.1 Vector Products

For inner products use `np.inner`[145]. For outer products use `np.cross`[146].

```
a = np.array([1, 2, 3])
b = np.array([1, 0, 2])

print(np.inner(a, b))
print(np.cross(a, b))
```

```
7
[ 4  1 -2]
```

16.9.2 Matrix Transpose

The `np.transpose`[147] function yields the transpose of a matrix. Alternatively, a NumPy array's member variable `T` holds the transpose, too. The transpose is a view (not a copy) of the original matrix.

```
A = np.array([[1, 2, 3],
              [4, 5, 6]])

print(np.transpose(A))
print(A.T)
```

```
[[1 4]
 [2 5]
 [3 6]]
[[1 4]
 [2 5]
 [3 6]]
```

[142] https://numpy.org/doc/stable/reference/generated/numpy.amin.html
[143] https://numpy.org/doc/stable/reference/generated/numpy.nanmin.html
[144] https://numpy.org/doc/stable/reference/routines.linalg.html
[145] https://numpy.org/doc/stable/reference/generated/numpy.inner.html
[146] https://numpy.org/doc/stable/reference/generated/numpy.outer.html
[147] https://numpy.org/doc/stable/reference/generated/numpy.transpose.html

16.9.3 Matrix Multiplication

NumPy introduces the `@` operator for matrix multiplication. It's equivalent to calling `np.matmul`[148].

```
A = np.array([[1, 2, 3],
              [4, 5, 6]])
B = np.array([[1, 0],
              [2, 1],
              [1, 1]])

print(A @ B)
print(np.matmul(A, B))
```

```
[[ 8  5]
 [20 11]]
[[ 8  5]
 [20 11]]
```

16.9.4 Determinants and Inverses

Determinants and inverses of square matrices can be computed with `np.linalg.det`[149] and `np.linalg.inv`[150], respectively.

```
A = np.array([[2, 0],
              [1, 1]])

print(np.linalg.det(A))
print(np.linalg.inv(A))
```

```
2.0
[[ 0.5  0. ]
 [-0.5  1. ]]
```

16.9.5 Solving Systems of Linear Equations

The `np.solve`[151] function solves a system of linear equations.

```
A = np.array([[2, 0],
              [1, 1]])
b = np.array([2, 3])

print(np.linalg.solve(A, b))
```

```
[1. 2.]
```

[148] https://numpy.org/doc/stable/reference/generated/numpy.matmul.html
[149] https://numpy.org/doc/stable/reference/generated/numpy.linalg.det.html
[150] https://numpy.org/doc/stable/reference/generated/numpy.linalg.inv.html
[151] https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html

16.10 Random Numbers

In data science contexts random numbers are important for simulating data and for selecting random subsets of data. NumPy provides a submodule `random` for creating arrays of (pseudo-)random numbers.

```
import numpy as np
```

16.10.1 Random Number Generators

NumPy supports several different algorithms for generating random numbers. For our purposes choice of algorithm does not matter (for crypto applications it matters!). Luckily NumPy provides a default one.

We first have to create a random number generator object (or get the default one) and initialize it with a seed. The seed determines the sequence of generated random numbers. Using a fixed seed is important if we need reproducable results (when testing things, for instance).

```
rng = np.random.default_rng(123)     # use some integer as seed here
```

16.10.2 Getting Random Numbers

Random numbers may follow different distributions. NumPy provides many standard distributions, see Random Generator[152] in NumPy's documentation.

```
# random integers (arguments: first, last + 1, shape)
a = rng.integers(23, 42, (4, 10))

print(a)
```

```
[[23 35 34 24 40 27 27 26 29 26]
 [29 38 31 40 31 28 37 38 39 39]
 [23 32 28 27 27 38 38 27 30 37]
 [25 34 31 40 37 27 38 38 27 32]]
```

```
# uniformly distributed floats in [0, 1)
a = rng.random((4, 4))

print(a)
```

```
[[0.23155562 0.16590399 0.49778897 0.58272464]
 [0.18433799 0.01489492 0.47113323 0.72824333]
 [0.91860049 0.62553401 0.91712257 0.86469025]
 [0.21814287 0.86612743 0.73075194 0.27786529]]
```

```
# permutation of an array
a = np.array([1, 2, 3, 4 ,5])
b = rng.permutation(a)

print(b)
```

```
[4 2 5 3 1]
```

[152] https://numpy.org/doc/stable/reference/random/generator.html#simple-random-data

CHAPTER

SEVENTEEN

SAVING AND LOADING NON-STANDARD DATA

There exist Python modules for almost all standard file formats. Readers and writers for several formats also are included in larger packages like `matplotlib`, `opencv`, `pandas`. To share data with others always use some standard file format (PNG or JPEG for images, CSV for tabulated data, and so one).

For storing temporary data like interim results NumPy and the `pickle` module from Python's standard library provide very convenient quick-and-dirty functions. Next to those functions, in this chapter we also discuss how to read custom binary file formats.

Related projects:

- *MNIST Character Recognition* (page 931)
  - *The xMNIST Family of Data Sets* (page 931)
  - *Load QMNIST* (page 933)

17.1 Saving and Loading NumPy Arrays

NumPy provides functions for saving arrays to files and for loading arrays from files.

```
import numpy as np
```

17.1.1 One Array per File

With `np.save`[153] we can write one array to a file.

```
a = np.array([1, 2, 3])

np.save('some_array.npy', a)
```

The `np.load`[154] functions reads an array from a file written with `np.save`:

```
a = np.load('some_array.npy')

print(a)
```

```
[1 2 3]
```

[153] https://numpy.org/doc/stable/reference/generated/numpy.save.html
[154] https://numpy.org/doc/stable/reference/generated/numpy.load.html

17.1.2 Multiple Arrays

To save multiple arrays to one file use `np.savez`[155] and provide each array as a keyword argument. The result is the same as calling `save` and creating an uncompressed (!) ZIP archive containing all files. File names in the ZIP archive correspond to keyword argument names.

```
a = np.array([1, 2, 3])
b = np.array([4, 5])

np.savez('many_arrays.npz', a=a, b=b)
```

Use `np.load`[156] to load multiple arrays written with `savez`. The returned object is dict-like, that is, it behaves like a dictionary, but isn't of type `dict`. Conversion to `dict` works as expected.

```
with np.load('many_arrays.npz') as data:     # data is dict-like
    a = data['a']
    b = data['b']

print(a)
print(b)
```

```
[1 2 3]
[4 5]
```

To get a compressed ZIP archive use `np.savez_compressed`[157].

17.2 Saving and Loading Arbitrary Python Objects

The `pickle` module provides functions for *pickling* (saving) and *unpickling* (loading) almost arbitrary Python objects to and from files, respectively. For details on what objects are picklable see documentation of the `pickle` module[158].

```
import pickle
```

There exist two interfaces: either use the functions `dump` and `load` or create a `Pickler` and an `Unpickler` object. Here we only discuss the former variant. For the latter see `pickle` module[159] in Python's documentation.

17.2.1 Pickling

Steps for pickling are:

1. Open a file for writing in binary mode.
2. Call `dump`[160] for each object to pickle.
3. Close the file.

```
some_object = [1, 2, 3, 4]
another_object = 'I\'m a string.'

with open('test.pkl', 'wb') as f:
```

(continues on next page)

[155] https://numpy.org/doc/stable/reference/generated/numpy.savez.html
[156] https://numpy.org/doc/stable/reference/generated/numpy.load.html
[157] https://numpy.org/doc/stable/reference/generated/numpy.savez_compressed.html
[158] https://docs.python.org/3/library/pickle.html#what-can-be-pickled-and-unpickled
[159] https://docs.python.org/3/library/pickle.html
[160] https://docs.python.org/3/library/pickle.html#pickle.dump

(continued from previous page)

```
    pickle.dump(some_object, f)
    pickle.dump(another_object, f)
```

17.2.2 Unpickling

Steps for unpickling are:

1. Open the file for reading in binary mode.
2. Call `load`[161] for each object to unpickle.
3. Close the file.

```
with open('test.pkl', 'rb') as f:
    some_object = pickle.load(f)
    another_object = pickle.load(f)

print(some_object)
print(another_object)
```

```
[1, 2, 3, 4]
I'm a string.
```

Unpickling objects from unknown sources is a **security risk**. See `pickle`'s documentation[162].

17.2.3 (Un)Pickling many Objects

If you have many objects to pickle, create a list of all objects and pickle the list. The advantage is, that for unpickling you do not have to remember how many objects you have pickled. Simply unpickle the list and look at its length.

17.3 Reading Custom Binary File Formats

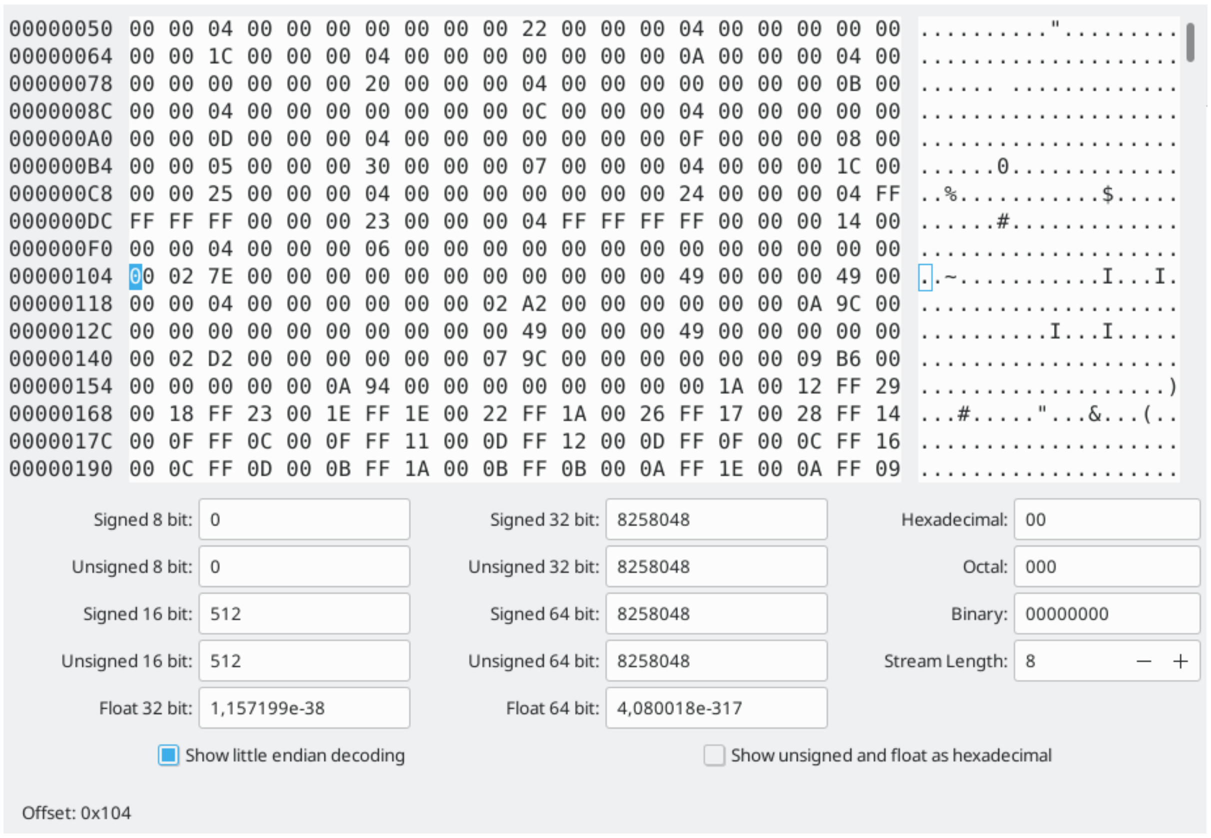
Sometimes data comes in custom binary formats for which no library functions exist. To read data from binary files we have to know how to interpret the data. Which bytes represent text? Which bytes represent numbers? And so on. Without format specification binary files are almost useless.

17.3.1 Viewing Binary Files

To view binary files use a hex editor. A hex editor shows a file byte by byte, where each byte is shown as two hexadecimal digits. If you do not have a hex editor installed, try wxHexEditor[163].

Most binary files are composed of strings, bit masks, integers, floats, and padding bytes. The hex editor shows common interpretations of bytes at current cursor position.

[161] https://docs.python.org/3/library/pickle.html#pickle.load
[162] https://docs.python.org/3/library/pickle.html
[163] https://www.wxhexeditor.org



```
00000050 00 00 04 00 00 00 00 00 00 00 22 00 00 00 04 00 00 00 00 00  ..........".........
00000064 00 00 1C 00 00 00 04 00 00 00 00 00 00 00 0A 00 00 00 04 00  ....................
00000078 00 00 00 00 00 00 20 00 00 00 04 00 00 00 00 00 00 00 0B 00  ...... .............
0000008C 00 00 04 00 00 00 00 00 00 00 0C 00 00 00 04 00 00 00 00 00  ....................
000000A0 00 00 0D 00 00 00 04 00 00 00 00 00 00 00 0F 00 00 00 08 00  ....................
000000B4 00 00 05 00 00 00 30 00 00 00 07 00 00 00 04 00 00 00 1C 00  ......0.............
000000C8 00 00 25 00 00 00 04 00 00 00 00 00 00 00 24 00 00 00 04 FF  ..%...........$.....
000000DC FF FF FF 00 00 00 23 00 00 00 04 FF FF FF FF 00 00 00 14 00  ......#.............
000000F0 00 00 04 00 00 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00  ....................
00000104 00 02 7E 00 00 00 00 00 00 00 00 00 00 00 49 00 00 00 49 00  ..~...........I...I.
00000118 00 00 04 00 00 00 00 00 00 02 A2 00 00 00 00 00 00 0A 9C 00  ....................
0000012C 00 00 00 00 00 00 00 00 00 00 49 00 00 00 49 00 00 00 00 00  ..........I...I.....
00000140 00 02 D2 00 00 00 00 00 00 07 9C 00 00 00 00 00 00 09 B6 00  ....................
00000154 00 00 00 00 00 0A 94 00 00 00 00 00 00 00 00 1A 00 12 FF 29  ...................)
00000168 00 18 FF 23 00 1E FF 1E 00 22 FF 1A 00 26 FF 17 00 28 FF 14  ...#....."...&...(..
0000017C 00 0F FF 0C 00 0F FF 11 00 0D FF 12 00 0D FF 0F 00 0C FF 16  ....................
00000190 00 0C FF 0D 00 0B FF 1A 00 0B FF 0B 00 0A FF 1E 00 0A FF 09  ....................
```

Signed 8 bit:	0	Signed 32 bit:	8258048	Hexadecimal:	00
Unsigned 8 bit:	0	Unsigned 32 bit:	8258048	Octal:	000
Signed 16 bit:	512	Signed 64 bit:	8258048	Binary:	00000000
Unsigned 16 bit:	512	Unsigned 64 bit:	8258048	Stream Length:	8 − +
Float 32 bit:	1,157199e-38	Float 64 bit:	4,080018e-317		

[✓] Show little endian decoding
[ ] Show unsigned and float as hexadecimal

Offset: 0x104

Fig. 17.1: A hex editor shows file contents in hexadecimal notation and as ASCII characters (right column) together with common interpretations (lower panel).

17.3.2 Reading Strings

We already discussed decoding binary data to strings in the chapter on *Text Files* (page 124). The only question is how to find the end of a string. This question should be answered in the format specification. Usually string data is terminated by a byte with value 0.

17.3.3 Reading Bit Masks

Bit masks are bytes in which each bit describes a truth value. To extract a bit from a byte all programming languages provide bitwise operators. Here we interpret a byte as sequence of 8 bits. Following bitwise operations can be used:

- `a & b` returns 1 at a bit position if and only if `a` and `b` are both 1 at this position (*bitwise and*).
- `a | b` returns 1 at a bit position if and only if at least one of `a` and `b` is 1 at this position (*bitwise or*)
- `a ^ b` returns 1 at a bit position if and only if exactly one of `a` and `b` is 1 at this position (*bitwise exclusive or*)
- `~a` returns 1 at a bit position if and only if `a` is 0 at this position (*bitwise not*)

Python implements these bitwise operators for signed integers, which results in somewhat unexpected results (but it's the only way since Python has no unsigned integers). Thus, better use NumPy's types.

To read the third bit use `& 0b00100000`:

```python
# some integer to be interpreted as bit mask (prefix 0b indicates binary notation)
bit_mask = np.uint8(0b10111100)

# get bit and convert result from int to bool
third_bit = bool(bit_mask & np.uint8(0b00100000))

third_bit
```

```
True
```

To set the third bit to 1 (when writing binary files) use `| 0b00100000`.

```python
# some integer to be interpreted as bit mask (prefix 0b indicates binary notation)
bit_mask = np.uint8(0b10011100)

# update bit mask (set third bit without modifying others)
bit_mask = bit_mask | np.uint8(0b00100000)

bin(bit_mask)
```

```
'0b10111100'
```

To set the third bit to 0 (when writing binary files) use `& ~0b00100000`.

```python
# some integer to be interpreted as bit mask (prefix 0b indicates binary notation)
bit_mask = np.uint8(0b10111100)

# update bit mask (set third bit without modifying others)
bit_mask = bit_mask & ~np.uint8(0b00100000)

bin(bit_mask)
```

```
'0b10011100'
```

17.3.4 Reading Integers

Integer values in a binary file may have different lengths, starting from 1 byte upto 8 byte. Reading a 1-byte-integer is very simple. Just read the byte. For two-byte integers things become more involved. There is a first (closer to begin of file) and a second byte and there is no universally accepted rule for converting two bytes to an integer. Denoting the first byte by a and the second by b there are two possibilities:

- $a + 256\,b$ (least significant byte first, *little endian*, Intel format)
- $256\,a + b$ (most significant byte first, *big endian*, Motorola format)

If we have 4-byte integers, the problem persists. With bytes a, b, c, d we have

- $a + 256\,b + 256^2\,c + 256^3\,d$ (little endian)
- $256^3\,a + 256^2\,b + 256\,c + d$ (big endian)

Analogously for 8-byte integers.

NumPy provides the `fromfile`[164] function to read integers and other numeric data from binary files. Next to `offset` (starting position) and `count` (number of items to read) it has a `dtype` keyword argument. Usual Python and NumPy types are allowed, but more detailed type control is possible by providing a string consisting of:

- `'<'` (little endian) or `'>'` (big endian) and
- `'i'` (signed integer) or `'u'` (unsigned integer) and
- length of item in bytes.

Reading unsigned 32-bit integers in little endian notation would require `'<u4'`, for instance.

If data is already in memory, use `frombuffer`[165] instead of `fromfile`.

```
data = bytes([200, 3, 4, 5])

# 4 unsigned 8-bit integers
a = np.frombuffer(data, 'u1')
print(a)

# 4 signed 8-bit integers
a = np.frombuffer(data, 'i1')
print(a)

# 2 unsigned 16-bit integers (little endian)
a = np.frombuffer(data, '<u2')
print(a)

# 2 unsigned 16-bit integers (big endian)
a = np.frombuffer(data, '>u2')
print(a)

# 1 unsigned 32-bit integer (big endian)
a = np.frombuffer(data, '>u4')
print(a)

# 1 signed 32-bit integer (big endian)
a = np.frombuffer(data, '>i4')
print(a)
```

```
[200   3   4   5]
[-56   3   4   5]
[ 968 1284]
```

(continues on next page)

[164] https://numpy.org/doc/stable/reference/generated/numpy.fromfile.html
[165] https://numpy.org/doc/stable/reference/generated/numpy.frombuffer.html

(continued from previous page)

```
[51203  1029]
[3355640837]
[-939326459]
```

See Byte-swapping[166] for more detailes on NumPy’s support of endianess.

[166] https://numpy.org/doc/1.19/user/basics.byteswapping.html

CHAPTER

EIGHTEEN

HIGH-LEVEL DATA MANAGEMENT WITH PANDAS

The Pandas[167] Python package makes NumPy's efficient computing capabilities more accessible for data science purposes and adds functionality for complex data types like timestamps, time periods and categories. Pandas provides powerful data structures and functions for managing, transforming and analyzing data.

Related exercises:

- *Pandas Basics* (page 881)
- *Pandas Indexing* (page 884)
- *Advanced Pandas* (page 886)
- *Pandas Vectorization* (page 889)

Related projects:

- *Public Transport* (page 949)
  - *Get Data and Set Up the Environment* (page 949)
  - *Find Connections* (page 954)
- *Corona Deaths* (page 959)
- *Weather* (page 923)
  - *Climate Change* (page 927)

[167] https://pandas.pydata.org/

18.1 Series

Pandas `Series` is one of two fundamental Pandas data types (the other is `DataFrame`). A `Series` object holds one-dimensional data, like a list, but with more powerful indexing capabilities. Data is stored in an underlying one-dimensional NumPy array. Thus, most operations are much more efficient than with lists.

```
import pandas as pd
```

18.1.1 Creation of `Series` Objects

A `Series` object can be created from a Python list or a dictionary, for instance. See Series constructor[168] in Pandas' documentation.

```
s = pd.Series([23, 45, 67, 78, 90])
s
```

```
0    23
1    45
2    67
3    78
4    90
dtype: int64
```

```
s = pd.Series({'a': 12, 'b': 23, 'c': 45, 'd': 67})
s
```

```
a    12
b    23
c    45
d    67
dtype: int64
```

A `Series` consists of an index (first column printed) and its data (second column printed). All data items have to be of identical type. The length of a `Series` is provided by the `size` member variable (you may also use Python's built-in function `len`).

```
s.size
```

```
4
```

18.1.2 Data Alignment

Data in a `Series` behaves like a one-dimensional `ndarray`, but Pandas' indexing mechanisms make things different from NumPy. Pandas implements automatic *data alignment*. That is, data items do not have fixed positions like in a NumPy array. Instead, only the (possibly non-integer) index matters. Here is a first example:

```
a = pd.Series({'a': 2, 'b': 4, 'c': 3, 'd': 6})
b = pd.Series({'a': 1, 'b': 5, 'd': 7, 'e': 9})
print(a, '\n')
print(b, '\n')
print(a + b)
```

[168] https://pandas.pydata.org/docs/reference/api/pandas.Series.html

```
a    2
b    4
c    3
d    6
dtype: int64

a    1
b    5
d    7
e    9
dtype: int64

a     3.0
b     9.0
c     NaN
d    13.0
e     NaN
dtype: float64
```

Both series have indices `a`, `b`, `d`. Thus, addition is defined. But `c` and `e` appear only in one of the series. Addition fails and the result is *not a number*.

Important: Note that data type now is float although every number is an integer. The reason is, that integers do not allow to represent the float `NaN`. Thus, Pandas has to change to data type of the result. We will come back to such `NaN` problems later on.

If we had used NumPy, then the result would be the sum of two vectors:

```
import numpy as np
```

```
a = np.array([2, 4, 3, 6])
b = np.array([1, 5, 7, 9])

a + b
```

```
array([ 3,  9, 10, 15])
```

18.1.3 Underlying Data Structures

Index and data are accessible via `index` and `array` members of `Series` objects:

```
s = pd.Series([23, 45, 67, 78, 90])

print(s.index, '\n')
print(s.array, '\n')
print(type(s.index), '\n')
print(type(s.array))
```

```
RangeIndex(start=0, stop=5, step=1)

<PandasArray>
[23, 45, 67, 78, 90]
Length: 5, dtype: int64

<class 'pandas.core.indexes.range.RangeIndex'>
```

(continues on next page)

(continued from previous page)

```
<class 'pandas.core.arrays.numpy_.PandasArray'>
```

The `index` member is one of several index types. Index objects will be discussed later on. The `array` member is an array type defined by Pandas. If we want to have a NumPy array, we should call `to_numpy()`:

```
a = s.to_numpy()

print(a, '\n')
print(type(a))
```

```
[23 45 67 78 90]

<class 'numpy.ndarray'>
```

18.1.4 Indexing

Accessing single items or subsets of a series works more or less the same way as for lists or dictionaries or NumPy arrays.

The flexibility of Pandas' multiple-items indexing mechanisms sometimes leads to confusion and unexpected erros. In addition, some features are not well documented and a transition to more predictable and more clearly structured indexing behavior is in progress.

Overview

There exist four widely used indexing mechanisms (here `s` is some series):

- `s[...]`: Python style indexing
- `s.ix[...]`: old Pandas style indexing (removed from Pandas in January 2020)
- `s.loc[...]` and `s.iloc[...]`: new Pandas style indexing
- `s.at[...]` and `s.iat[...]`: new Pandas style indexing for more efficient access to single items

Deprecated Indexing

Python style indexing and old Pandas style indexing (the `ix` *indexer*) allow for position based indexing and label based indexing. Position based means that, like for NumPy arrays, we refer to an item by its position in the series. The first item has position 0. Thus, the series' index object is completely ignored. Providing an item of the series' `index` member as index, is refered to as label based indexing.

Both `[...]` and `ix[...]` behave slightly differently when using slicing. A major problem is that sometimes it is not clear whether positional or label based indexing shall be used. Consider a series with an index made of id numbers, that is, integers:

```
s = pd.Series({123: 3, 45: 4, 542: 7, 2: 19})
print(s, '\n')

print(s[123], '\n')     # label based
print(s[2], '\n')       # label based
print(s[0:2])           # position based (October 2022 warning: will be label␣
 ↪based in future)
```

```
123     3
45      4
542     7
2      19
dtype: int64

3

19

123    3
45     4
dtype: int64
```

```
/tmp/ipykernel_234813/411904015.py:6: FutureWarning: The behavior of␣
↪`series[i:j]` with an integer-dtype index is deprecated. In a future version,␣
↪this will be treated as *label-based* indexing, consistent with e.g.␣
↪`series[i]` lookups. To retain the old behavior, use `series.iloc[i:j]`. To␣
↪get the future behavior, use `series.loc[i:j]`.
  print(s[0:2])          # position based (October 2022 warning: will be label␣
↪based in future)
```

Without knowing the exact mechanism behind `[...]`, which in fact calls the series' `__getitem__` method, code becomes unreadable. Same is true for `ix`. The `ix` indexer has been removed from Pandas since version 1.0.0 (January 2020). Indexing with `[...]` is still available, but should be avoided, at least for series with integer labels.

New Indexing Mechanism

Prefered indexing is via `loc[...]` and `iloc[...]`, the first for label based indexing, the second for positional indexing. Positional indexing is also known as *integer indexing*, thus the `i` in `iloc`. Slicing and boolean indexing are supported (see below).

If only a single item shall be accessed, then `loc[...]` and `iloc[...]` might be too slow due to the implementation of complex features like slicing. For single item access one should use `at[...]` and `iat[...]` providing label based and positional indexing, respectively.

Positional Indexing

Positional indexing via `iloc[...]` or `iat[...]` works like for one-dimensional NumPy arrays.

```
s = pd.Series({'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5})
print(s, '\n')

print(s.iloc[1:3], '\n')           # slicing
print(s.iloc[[3, 0, 2]], '\n')     # list of indices
print(s.iloc[[True, False, False, True, True]], '\n')     # boolean indexing
print(s.iat[3], '\n')              # efficient single element access
print(s.iloc[3])                   # less efficient single element access
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64

b    2
```

(continues on next page)

(continued from previous page)

```
c    3
dtype: int64

d    4
a    1
c    3
dtype: int64

a    1
d    4
e    5
dtype: int64

4

4
```

An important difference to NumPy indexing is, that the result is a series again. That is, the index of the selected items is returned, too.

Label Based Indexing

Label based indexing works like with dictionaries. But slicing is allowed.

```
s = pd.Series({'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5})
print(s, '\n')

print(s.loc['b':'d'], '\n')              # slicing
print(s.loc[['d', 'a', 'c']], '\n')     # list of labels
print(s.loc[[True, False, False, True, True]], '\n')     # boolean indexing
print(s.at['d'], '\n')                   # efficient single element access
print(s.loc['d'])                        # less efficient single element access
```

```
a    1
b    2
c    3
d    4
e    5
dtype: int64

b    2
c    3
d    4
dtype: int64

d    4
a    1
c    3
dtype: int64

a    1
d    4
e    5
dtype: int64

4

4
```

Important: Note that slicing with labels includes the stop item!

Different items with identical labels are allowed. In such case `loc[...]` returns all items with the specified label and `at[...]` returns an array of all values with the specified label.

```
s = pd.Series([1, 2, 3, 4], index=['a', 'b', 'b', 'c'])
print(s, '\n')

print(s.loc['b'], '\n')
print(s.at['b'])
```

```
a    1
b    2
b    3
c    4
dtype: int64

b    2
b    3
dtype: int64

b    2
b    3
dtype: int64
```

Indexing by Callables

Both `loc[...]` and `iloc[...]` accept a function as their argument. The function has to take a series as argument and has to return something allowed for indexing (list of indices/labels, boolean array and so on).

Scenarios justifying indexing by callables are relatively complex.

Views and Copies

As for NumPy arrays, indexing Pandas series may return a view of the series. That is, modifying the extracted subset of items might modify the original series. If you really need a copy of the items, use the `copy`[169] method of `Series` objects.

18.1.5 Some Useful Member Functions

A full list of member functions for `Series` objects[170] is provided in Pandas' documentation. Here we only list a few of them.

[169] https://pandas.pydata.org/docs/reference/api/pandas.Series.copy.html
[170] https://pandas.pydata.org/docs/reference/series.html

A First Look at a Series

If a series is read from a file we would like to get some basic information about the series.

With `describe`[171] we get statistical information about a series. The function returns a `Series` object containing the collected information.

First and last items are returned by `head`[172] and `tail`[173], respectively. Both take an optional argument specifying the number of items to return. Default is 5.

```
s = pd.Series([2, 4, 6, 5, 4, 3, -2, 3, 2, 5])

print(s.describe(), '\n')
print(s.head(), '\n')
print(s.tail(3))
```

```
count    10.000000
mean      3.200000
std       2.250926
min      -2.000000
25%       2.250000
50%       3.500000
75%       4.750000
max       6.000000
dtype: float64

0    2
1    4
2    6
3    5
4    4
dtype: int64

7    3
8    2
9    5
dtype: int64
```

Note that we did not specify labels explicitly. Thus, the `Series` constructor uses item positions as labels.

Iterating Over a Series

Iterating over the values of a series works like for Python lists:

```
s = pd.Series([2, 4, 6, 5, 4, 3, -2, 3, 2, 5])

for i in s:
    print(i)
```

```
2
4
6
5
4
3
-2
```

(continues on next page)

[171] https://pandas.pydata.org/docs/reference/api/pandas.Series.describe.html
[172] https://pandas.pydata.org/docs/reference/api/pandas.Series.head.html
[173] https://pandas.pydata.org/docs/reference/api/pandas.Series.tail.html

(continued from previous page)

```
3
2
5
```

If labels are required, too, call `items`[174]:

```
for lab, val in s.items():
    print(lab, val)
```

```
0 2
1 4
2 6
3 5
4 4
5 3
6 -2
7 3
8 2
9 5
```

If next to labels also positional indices are required use an additional `enumerate`:

```
for pos, (lab, val) in enumerate(s.items()):
    print(pos, lab, val)
```

```
0 0 2
1 1 4
2 2 6
3 3 5
4 4 4
5 5 3
6 6 -2
7 7 3
8 8 2
9 9 5
```

Vectorized Operators

Like NumPy arrays Pandas series implement most mathematical and comparison operators.

```
a = pd.Series([1, 2, 3, 4])
b = pd.Series([4, 0, 6, 3])

print(a * b, '\n')
print(a < b)
```

```
0     4
1     0
2    18
3    12
dtype: int64

0     True
1    False
```

(continues on next page)

[174] https://pandas.pydata.org/docs/reference/api/pandas.Series.items.html

(continued from previous page)

```
2     True
3    False
dtype: bool
```

Hint: Remember that Pandas uses data alignment, that is, labels matter, positions are irrelevant.

Functions `all`[175] and `any`[176] for boolean series are available, too.

```
s = pd.Series([True, True, False])

print(s.all())
print(s.any())
```

```
False
True
```

Removing and Adding Items

With `drop`[177] we can remove items from a series. Simply pass a list of labels to the function.

```
s = pd.Series([2, 4, 6, 5, 4, 3, -2, 3, 2, 5])
print(s, '\n')

t = s.drop([3, 4, 5])
print(t)
```

```
0    2
1    4
2    6
3    5
4    4
5    3
6   -2
7    3
8    2
9    5
dtype: int64

0    2
1    4
2    6
6   -2
7    3
8    2
9    5
dtype: int64
```

The `concat`[178] method concatenates two series.

[175] https://pandas.pydata.org/docs/reference/api/pandas.Series.all.html
[176] https://pandas.pydata.org/docs/reference/api/pandas.Series.any.html
[177] https://pandas.pydata.org/docs/reference/api/pandas.Series.drop.html
[178] https://pandas.pydata.org/docs/reference/api/pandas.concat.html

```
a = pd.Series({'a': 1, 'b': 2, 'c': 3, 'd': 4})
b = pd.Series({'d': 0, 'e': 5, 'f': 6, 'g': 7})

c = pd.concat([a, b])

print(a, '\n')
print(b, '\n')
print(c)
```

```
a    1
b    2
c    3
d    4
dtype: int64

d    0
e    5
f    6
g    7
dtype: int64

a    1
b    2
c    3
d    4
d    0
e    5
f    6
g    7
dtype: int64
```

Note that there is no check on duplicate index labels, since duplicates are no problem (see above).

Modifying Data in a Series

Important functions for modifying data in a series are:

- `apply`[179] (apply a function to each item or to the whole data array),
- `combine`[180] (choose items from two series to form a new one),
- `where`[181] (replace items which do not satisfy a condition),
- `mask`[182] (replace items which satisfy a condition)

[179] https://pandas.pydata.org/docs/reference/api/pandas.Series.apply.html
[180] https://pandas.pydata.org/docs/reference/api/pandas.Series.combine.html
[181] https://pandas.pydata.org/docs/reference/api/pandas.Series.where.html
[182] https://pandas.pydata.org/docs/reference/api/pandas.Series.mask.html

18.2 Data Frames

A Pandas data frame (a `DataFrame` object) is a collection of Pandas series with common index. Each series can be interpreted as a column in a two-dimensional table. There is a second index object for indexing columns.

Data frames are the most important data structure provided by Pandas.

```
import pandas as pd
```

18.2.1 Creation of `DataFrame` Objects

A `DataFrame` object can be created from lists/dictionaries of lists/dictionaries or from NumPy arrays or from lists/dictionaries of series, for instance. See DataFrame constructor[183] in Pandas' documentation. If necessary row labels and column labels can be provided via `index` and `columns` keyword arguments, respectively.

```
s1 = pd.Series({'a': 1, 'b': 2})
s2 = pd.Series({'b': 3, 'c': 4})

df = pd.DataFrame({'left': s1, 'right': s2})

df
```

```
   left  right
a   1.0    NaN
b   2.0    3.0
c   NaN    4.0
```

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                  index=['top', 'middle', 'bottom'],
                  columns=['left', 'middle', 'right'])

df
```

```
        left  middle  right
top        1       2      3
middle     4       5      6
bottom     7       8      9
```

Hint: JupyterLab shows data frames as graphical table. In other words, Jupyter's `display` function yields output different from `print`. With `print` we get a text representation of the data frame.

The `shape` member contains a tuple with number of rows and columns in the data frame:

```
df.shape
```

```
(3, 3)
```

[183] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

18.2.2 Data Alignment

Like for series data frames implement data alignment where possible. This applies to row indexing as well as to column indexing.

```
df1 = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                   index=['a', 'b', 'c'], columns=['A', 'B', 'C'])
df2 = pd.DataFrame([[11, 12, 13], [14, 15, 16], [17, 18, 19]],
                   index=['b', 'c', 'd'], columns=['A', 'C', 'D'])

display(df1)
display(df2)
display(df1 + df2)
```

```
   A  B  C
a  1  2  3
b  4  5  6
c  7  8  9
```

```
    A   C   D
b  11  12  13
c  14  15  16
d  17  18  19
```

```
      A   B     C   D
a   NaN NaN   NaN NaN
b  15.0 NaN  18.0 NaN
c  21.0 NaN  24.0 NaN
d   NaN NaN   NaN NaN
```

18.2.3 Underlying Data Structures

The index object for row indexing is accessible via `index` member and the index object for column indexing is accessible via `columns` member.

Data frames also have a `to_numpy`[184] method returning a data frame's data as NumPy array.

18.2.4 Indexing

Indexing data frames is very similar to indexing series. The major difference is that for data frames we have to specify a row and a column instead of only a row.

Overview

There exist four widely used mechanisms (here `df` is some data frame):

- `df[...]`: Python style indexing
- `df.ix[...]`: old Pandas style indexing (removed from Pandas in January 2020)
- `df.loc[...]` and `df.iloc[...]`: new Pandas style indexing
- `df.at[...]` and `df.iat[...]`: new Pandas style indexing for more efficient access to single items

[184] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_numpy.html

Python style indexing and old Pandas style indexing (the `ix` *indexer*) allow for position based indexing, label based indexing and mixed indexing (position for row, label for column, and vice versa). Both `[...]` and `ix[...]` behave slightly differently. As already discussed for series, sometimes it is not clear whether positional or label based indexing shall be used. Thus, `[...]` should be used with care. The `ix` indexer has been removed from Pandas in January 2020, but may appear in old code and documents.

Label Based Column Indexing with `[...]`

Indexing with `[...]` is mainly used for selecting columns by label. For other purposes new Pandas style indexing should be used.

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                  index=['a', 'b', 'c'], columns=['A', 'B', 'C'])
display(df)

s = df['A']
df_part = df[['B', 'C']]

display(s)
display(df_part)
```

```
   A  B  C
a  1  2  3
b  4  5  6
c  7  8  9
```

```
a    1
b    4
c    7
Name: A, dtype: int64
```

```
   B  C
a  2  3
b  5  6
c  8  9
```

If only one label is provided, then the result is a `Series` object. If a list of labels is provided, then the result is a `DataFrame` object. In case of integer labels, label based indexing is used when selecting columns:

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                  index=['a', 'b', 'c'], columns=[2, 23, 45])
display(df)
display(df[2])
```

```
   2  23  45
a  1   2   3
b  4   5   6
c  7   8   9
```

```
a    1
b    4
c    7
Name: 2, dtype: int64
```

Hint: Label based column indexing can be used to create new columns: `df['my_new_column'] = 0` creates a new column containing zeros, for instance.

Positional Indexing

Positional indexing via `iloc[...]` or `iat[...]` works like for two-dimensional NumPy arrays.

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                  index=['a', 'b', 'c'], columns=['A', 'B', 'C'])
display(df)

display(df.iloc[1:3, 0:2])              # slicing
display(df.iloc[[1, 0], [2, 1, 0]])     # list of indices
display(df.iloc[[True, False, True], [False, True, True]])     # boolean indexing
display(df.iat[2, 1])                   # efficient single element access
display(df.iloc[2, 1])                  # less efficient single element access
```

```
   A  B  C
a  1  2  3
b  4  5  6
c  7  8  9
```

```
   A  B
b  4  5
c  7  8
```

```
   C  B  A
b  6  5  4
a  3  2  1
```

```
   B  C
a  2  3
c  8  9
```

```
8
```

```
8
```

Variants (slicing, boolean and so on) may be mixed for rows and columns.

Label Based Indexing

Label based indexing works like with dictionaries. But slicing is allowed.

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                  index=['a', 'b', 'c'], columns=['A', 'B', 'C'])
display(df)

display(df.loc['c':'b':-1, 'A':'B'])        # slicing
display(df.loc[['c', 'a'], ['B', 'A']])     # list of labels
display(df.loc[[True, False, True], [False, True, True]])     # boolean indexing
display(df.at['b', 'B'])                    # efficient single element access
display(df.loc['b', 'B'])                   # less efficient single element access
```

```
   A  B  C
a  1  2  3
b  4  5  6
c  7  8  9
```

```
   A  B
c  7  8
b  4  5
```

```
   B  A
c  8  7
a  2  1
```

```
   B  C
a  2  3
c  8  9
```

```
5
```

```
5
```

Important: Like for series label based slicing includes the stop label!

Indexing by Callables

Both `loc[...]` and `iloc[...]` accept a function for row index and column index. The function has to take a data frame as argument and has to return something allowed for indexing (list of indices/labels, boolean array and so on).

Mixing Positional and Label Based Indexing

Indexing with `[...]` allows for mixing positional and label based indexing. But `[...]` should be avoided as discussed for series. The only exception is column selection. With Pandas' new indexing style mixed indexing is still possible but requires some more bytes of code:

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                  index=['a', 'b', 'c'], columns=['A', 'B', 'C'])
display(df)

display(df.loc['a':'b', df.columns[0:2]])
```

```
   A  B  C
a  1  2  3
b  4  5  6
c  7  8  9
```

```
   A  B
a  1  2
b  4  5
```

The idea is to use `loc[...]` and the `columns` member variable. With `columns` we get access to the index object for column indexing. This index object allows for usual positional indexing techniques (slicing, boolean and so on). The result of indexing an index object is an index object again. But the new index object only contains the desired subset of indices. This smaller index object than is passed to `loc[...]`. Same is possible for rows via `index` member.

Views and Copies

Like for NumPy arrays indexing Pandas data frames may return a view of the data frame. That is, modifying the extracted subset of items might modify the original data frame. If you really need a copy of the items, use the `copy`[185] method of `DataFrame` objects.

18.2.5 Some Useful Member Functions

A full list of member functions for `DataFrame` objects[186] is provided in Pandas' documentation. Here we only list a few.

A First Look at a Data Frame

With `describe`[187] we get basic statistical information about each column holding numerical values. The function returns a `DataFrame` object containing the collected information. Only columns with numerical data are considered by `describe`.

First and last rows are returned by `head`[188] and `tail`[189]. Both take an optional argument specifying the number of rows to return. Default is 5.

The `info`[190] method prints memory usage and other useful information.

```
df = pd.DataFrame([[1, 2, 3, 'some'], [4, 5, 6, 'string'], [7, 8, 9, 'here']],
                  index=['a', 'b', 'c'], columns=['A', 'B', 'C', 'D'])
display(df)

display(df.describe())
display(df.head(2))
display(df.tail(2))
df.info()
```

```
   A  B  C       D
a  1  2  3    some
b  4  5  6  string
c  7  8  9    here
```

```
         A    B    C
count  3.0  3.0  3.0
mean   4.0  5.0  6.0
std    3.0  3.0  3.0
min    1.0  2.0  3.0
25%    2.5  3.5  4.5
50%    4.0  5.0  6.0
75%    5.5  6.5  7.5
max    7.0  8.0  9.0
```

[185] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.copy.html
[186] https://pandas.pydata.org/docs/reference/dataframe.html
[187] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.describe.html
[188] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.head.html
[189] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.tail.html
[190] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.info.html

```
   A  B  C       D
a  1  2  3    some
b  4  5  6  string
```

```
   A  B  C       D
b  4  5  6  string
c  7  8  9    here
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3 entries, a to c
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   A       3 non-null      int64 
 1   B       3 non-null      int64 
 2   C       3 non-null      int64 
 3   D       3 non-null      object
dtypes: int64(3), object(1)
memory usage: 120.0+ bytes
```

Iterating Over a Data Frame

To iterate over columns use `items`[191], which returns tuples containing the column label and the column's data as `Series` object.

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                  index=['a', 'b', 'c'], columns=['A', 'B', 'C'])
display(df)

for label, s in df.items():
    print(label)
    print(s, '\n')
```

```
   A  B  C
a  1  2  3
b  4  5  6
c  7  8  9
```

```
A
a    1
b    4
c    7
Name: A, dtype: int64

B
a    2
b    5
c    8
Name: B, dtype: int64

C
a    3
b    6
c    9
Name: C, dtype: int64
```

[191] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.items.html

Alternativly, iterate over the `columns` member variable:

```
for label in df.columns:
    print(label)
    print(df[label], '\n')
```

```
A
a    1
b    4
c    7
Name: A, dtype: int64

B
a    2
b    5
c    8
Name: B, dtype: int64

C
a    3
b    6
c    9
Name: C, dtype: int64
```

Iteration over rows can be implemented via `iterrows`[192] method. Analogously to `items` it returns tuples containing the row label and a `Series` object with the data.

```
for label, s in df.iterrows():
    print(label)
    print(s, '\n')
```

```
a
A    1
B    2
C    3
Name: a, dtype: int64

b
A    4
B    5
C    6
Name: b, dtype: int64

c
A    7
B    8
C    9
Name: c, dtype: int64
```

Hint: Data in each column of a data frame has identical type, but types in a row may differ (from column to column). Thus, calling `iterrows` may involve type casting to get row data as `Series` object.

Important: Usually there is no need to iterate over rows, because Pandas provides much faster vectorized code for almost all operations needed for typical data science projects.

[192] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iterrows.html

Vectorized Operators

Mathematical operations and comparisons work in complete analogy to `Series` objects.

Removing and Adding Items

Functions for removing and adding items:

- `concat`[193] (concatenate data frames vertically or horizontally),
- `drop`[194] (remove rows or columns from data frame).

Modifying Data in a Data Frame

- `apply`[195] (apply function rowwise or columnwise to data frame),
- `combine`[196] (choose items from two data frames to form a new one),
- `where`[197] (replace items which do not satisfy a condition),
- `mask`[198] (replace items which satisfy a condition).

Data Frames from and to CSV Files

The Pandas function `read_csv`[199] reads a CSV file and returns a data frame.

The `DataFrame` method `to_csv`[200] writes data to a CSV file.

Missing Values

Missing values are a common problem in data science. Pandas provides several functions and mechanisms for handling missing values. Important functions:

- `isna`[201] (return boolean data frame with `True` at missing value positions),
- `notna`[202] (return boolean data frame with `False` at missing value positions),
- `fillna`[203] (fill all missing values, different fill methods are provided),
- `dropna`[204] (remove rows or columns containing missing values or consisting completely of missing values).

Details may be found in Pandas user guide[205].

[193] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.concat.html
[194] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.drop.html
[195] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.combine.html
[196] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.combine.html
[197] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.where.html
[198] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.mask.html
[199] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html
[200] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_csv.html
[201] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.isna.html
[202] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.notna.html
[203] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html
[204] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html
[205] https://pandas.pydata.org/pandas-docs/stable/user_guide/missing_data.html

18.3 Advanced Indexing

One of Pandas' most useful features is its powerful indexing mechanism. Here we'll discuss several types of index objects.

```
import pandas as pd
```

Series and data frames have one or two index objects, respectively. They are accessible via `Series.index` or `DataFrame.index` and `DataFrame.columns`. An index object is a list-like object holding all row or column labels.

To create a new index, call the constructor of the `Index` class:

```
new_index = pd.Index(['a', 'b', 'c', 'd', 'e'])
new_index
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

18.3.1 Reindexing

An index object may be replaced by another one. We have to take care whether data alignment between old and new index shall be applied or not.

With Data Alignment

`Series` and `DataFrame` objects provide the `reindex`[206] method. This method takes an index object (or a list of labels) and replaces the existing index by the new one. Data alignment is applied, that is, rows/columns with label in the intersection of old and new index remain unchanged, but rows/columns with old label not in the new index are dropped. If there are labels in the new index which aren't in the old one, then rows/columns are filled with `nan` or some specified value or by some more complex filling logic.

```
s = pd.Series({'a': 123, 'b': 456, 'e': 789})
print(s, '\n')

new_index = pd.Index(['a', 'b', 'c', 'd', 'e'])
s = s.reindex(new_index)
s
```

```
a    123
b    456
e    789
dtype: int64
```

```
a    123.0
b    456.0
c      NaN
d      NaN
e    789.0
dtype: float64
```

With fill value:

[206] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.reindex.html

```
s = pd.Series({'a': 123, 'b': 456, 'e': 789})
print(s, '\n')

new_index = pd.Index(['a', 'b', 'c', 'd', 'e'])
s = s.reindex(new_index, fill_value=0)
s
```

```
a    123
b    456
e    789
dtype: int64
```

```
a    123
b    456
c      0
d      0
e    789
dtype: int64
```

With filling logic:

```
s = pd.Series({'a': 123, 'b': 456, 'e': 789})
print(s, '\n')

new_index = pd.Index(['a', 'b', 'c', 'd', 'e'])
s = s.reindex(new_index, method='bfill')
s
```

```
a    123
b    456
e    789
dtype: int64
```

```
a    123
b    456
c    789
d    789
e    789
dtype: int64
```

The `align`[207] method reindexes two series/data frames such that both have the same index.

```
s1 = pd.Series({'a': 123, 'b': 456, 'e': 789})
s2 = pd.Series({'a': 98, 'c': 76, 'e': 54})
print(s1, '\n')
print(s2, '\n')

s1, s2 = s1.align(s2, axis=0)
print(s1, '\n')
print(s2)
```

```
a    123
b    456
e    789
dtype: int64
```

(continues on next page)

[207] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.align.html

(continued from previous page)

```
a    98
c    76
e    54
dtype: int64

a    123.0
b    456.0
c      NaN
e    789.0
dtype: float64

a    98.0
b     NaN
c    76.0
e    54.0
dtype: float64
```

Without Data Alignment

To simply replace an index without data alignment, that is to rename all the labels, there are two variants:

- replace the index object by a new one of same length via usual assignment,
- use an existing column as index.

```
s = pd.Series({'a': 123, 'b': 456, 'e': 789})
print(s, '\n')

new_index = pd.Index(['aa', 'bb', 'cc'])
s.index = new_index
s
```

```
a    123
b    456
e    789
dtype: int64
```

```
aa    123
bb    456
cc    789
dtype: int64
```

To use a column of a data frame as index call the `set_index`[208] method and provide the column label.

```
df = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                  index=['a', 'b','c'], columns=['A', 'B', 'C'])
display(df)

df = df.set_index('A')
df
```

```
   A  B  C
a  1  2  3
b  4  5  6
c  7  8  9
```

[208] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.set_index.html

```
   B  C
A
1  2  3
4  5  6
7  8  9
```

To convert the index to a usual column call `reset_index`[209]. The index will be replaced by the standard index (integers starting at 0).

```
df = df.reset_index()
df
```

```
   A  B  C
0  1  2  3
1  4  5  6
2  7  8  9
```

18.3.2 Index Sharing

Index objects may be shared between several series or data frames. Simply pass the index of an existing series or data frame to the constructor of a new series or data frame or assign it directly or use `reindex`.

```
s1 = pd.Series({'a': 123, 'b': 456, 'e': 789})
s2 = pd.Series([4, 6, 8], index=s1.index)

s1.index is s2.index
```

```
True
```

Note: Reindexing two series/data frames with `align` (see above) results in a shared index.

18.3.3 Enlargement by Assignment

To append data to a series or data frame we may use label based indexing:

```
s = pd.Series({'a': 123, 'b': 456})
print(s, '\n')

s.loc['e'] = 789
s
```

```
a    123
b    456
dtype: int64
```

```
a    123
b    456
e    789
dtype: int64
```

[209] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.reset_index.html

```
df = pd.DataFrame([[1, 2], [3, 4]], index=['a', 'b'], columns=['A', 'B'])
display(df)

df.loc['c', 'D'] = 5
df
```

```
   A  B
a  1  2
b  3  4
```

```
     A    B    D
a  1.0  2.0  NaN
b  3.0  4.0  NaN
c  NaN  NaN  5.0
```

18.3.4 Range Indices

Pandas' standard index is of type `RangeIndex`[210]. It's used whenever a series or a data frame is created without specifying an index.

```
index = pd.RangeIndex(5, 21, 2)
print(index, '\n')

for k in index:
    print(k)
```

```
RangeIndex(start=5, stop=21, step=2)

5
7
9
11
13
15
17
19
```

18.3.5 Interval Indices

The `IntervalIndex`[211] class allows for imprecise indexing. Each item in a series or data frame can be accessed by any number in a specified interval.

```
interval_list = [pd.Interval(2, 3), pd.Interval(6, 7), pd.Interval(6.5, 9)]
print(interval_list, '\n')

s = pd.Series([23, 45, 67], index=pd.IntervalIndex(interval_list, closed='left'))
s
```

```
[Interval(2, 3, closed='right'), Interval(6, 7, closed='right'), Interval(6.5,␣
↪9, closed='right')]
```

[210] https://pandas.pydata.org/docs/reference/api/pandas.RangeIndex.html
[211] https://pandas.pydata.org/docs/reference/api/pandas.IntervalIndex.html

```
[2.0, 3.0)    23
[6.0, 7.0)    45
[6.5, 9.0)    67
dtype: int64
```

Indexing by concrete numbers:

```
print(s.loc[2], '\n')
# print(s.loc[3], '\n')    # KeyError
print(s.loc[2.5], '\n')
print(s.loc[6.7])
```

```
23

23

[6.0, 7.0)    45
[6.5, 9.0)    67
dtype: int64
```

Indexing by intervals:

```
# print(s.loc[pd.Interval(2, 3)])    # KeyError
print(s.loc[pd.Interval(2, 3, 'left')])    # only exact matches!
```

```
23
```

`IntervalIndex` objects provide `overlaps`[212] and `contains`[213] methods for more flexible indexing:

```
mask = s.index.overlaps(pd.Interval(2.5, 6.4))
print(mask)
s.loc[mask]
```

```
[ True  True False]
```

```
[2.0, 3.0)    23
[6.0, 7.0)    45
dtype: int64
```

```
mask = s.index.contains(6.7)
print(mask)
s.loc[mask]
```

```
[False  True  True]
```

```
[6.0, 7.0)    45
[6.5, 9.0)    67
dtype: int64
```

Note: In principle `contains` should work with intervals instead of concrete numbers, too. But in Pandas 1.5.1 `NotImplementedError` is raised.

[212] https://pandas.pydata.org/docs/reference/api/pandas.IntervalIndex.overlaps.html
[213] https://pandas.pydata.org/docs/reference/api/pandas.IntervalIndex.contains.html

18.3.6 Multi-Level Indexing

Up to some details, multi-level indexing is indexing using tuples as labels. Corresponding indexing objects are of type `MultiIndex`[214]. Major application of multi-level indices are indices representing several dimensions. Thus, high dimensional data can be stored in a two-dimensional data frame.

Creating a Multi-Level Index

Let's start with a two-level index. First level contains courses of studies provided by a university. Second level contains some lecture series. Data is the number of students from each course attending a lecture and the average rating for each lecture.

Using the `MultiIndex` constructor is the most general, but not very straight forward way to create a multi-level index. We have to provide lists of labels for each level. In addition, we need lists of codes for each level indicating which label to use at each position. Each level may have a name.

```
courses = ['Mathematics', 'Physics', 'Philosophie']
lectures = ['Computer Science', 'Mathematics', 'Epistemology']

courses_codes =  [0, 0, 0, 1, 1, 1, 2, 2, 2]
lectures_codes = [0, 1, 2, 0, 1, 2, 0, 1, 2]

index = pd.MultiIndex(levels=[courses, lectures],
                      codes=[courses_codes, lectures_codes],
                      names=['course', 'lecture'])

data = zip([10, 15, 8, 20, 17, 3, 2, 1, 89],
           [2.1, 1.3, 3.6, 3.0, 1.6, 4.7, 3.9, 4.9, 1.1])

df = pd.DataFrame(data, index=index, columns=['students', 'rating'])

df
```

```
                              students  rating
course      lecture
Mathematics Computer Science        10     2.1
            Mathematics             15     1.3
            Epistemology             8     3.6
Physics     Computer Science        20     3.0
            Mathematics             17     1.6
            Epistemology             3     4.7
Philosophie Computer Science         2     3.9
            Mathematics              1     4.9
            Epistemology            89     1.1
```

Alternative creation methods are `MultiIndex.from_arrays`[215], `MultiIndex.from_tuples`[216], `MultiIndex.from_product`[217], `MultiIndex.from_frame`[218].

Hint: The mentioned creation methods are `static` methods. See *Types* (page 90) for some explanation of the concept.

The above index contains each combination of items from two lists. Thus `from_product` is applicable:

[214] https://pandas.pydata.org/docs/reference/api/pandas.MultiIndex.html
[215] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.MultiIndex.from_arrays.html
[216] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.MultiIndex.from_tuples.html
[217] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.MultiIndex.from_product.html
[218] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.MultiIndex.from_frame.html

```
courses = ['Mathematics', 'Physics', 'Philosophie']
lectures = ['Computer Science', 'Mathematics', 'Epistemology']

index = pd.MultiIndex.from_product([courses, lectures], names=['course', 'lecture
↪'])

data = zip([10, 15, 8, 20, 17, 3, 2, 1, 89],
           [2.1, 1.3, 3.6, 3.0, 1.6, 4.7, 3.9, 4.9, 1.1])

df = pd.DataFrame(data, index=index, columns=['students', 'rating'])

df
```

```
                                  students  rating
course      lecture
Mathematics Computer Science            10     2.1
            Mathematics                 15     1.3
            Epistemology                 8     3.6
Physics     Computer Science            20     3.0
            Mathematics                 17     1.6
            Epistemology                 3     4.7
Philosophie Computer Science             2     3.9
            Mathematics                  1     4.9
            Epistemology                89     1.1
```

Level information is stored in the `levels` member variable:

```
df.index.levels
```

```
FrozenList([['Mathematics', 'Philosophie', 'Physics'], ['Computer Science',
↪'Epistemology', 'Mathematics']])
```

Note, that multi-level indexing is not restricted to row indexing. Multi-level column indexing works in exactly the same manner.

Accessing Data

Accessing data works as for other types of indices. Labels now are tuples containing one item per level. But there exist additional techniques specific to multi-level indices.

Single Tuples

```
df.loc[('Physics', 'Computer Science'), :]
```

```
students    20.0
rating       3.0
Name: (Physics, Computer Science), dtype: float64
```

```
df.iloc[1, 1]
```

```
1.3
```

```
df.loc[[('Physics', 'Computer Science'), ('Mathematics', 'Epistemology')], :]
```

```
                               students  rating
course      lecture
Physics     Computer Science         20     3.0
Mathematics Epistemology              8     3.6
```

Slicing works as usual.

Slicing Inside Tuples

A new feature specific to multi-level indexing is slicing inside tuples. We would expect notation like

```
('Physics', :)
```

to get all rows with `Physics` at first level. But usual slicing syntax is not available here. Instead we have to use the built-in `slice` function. It takes start, stop and step values (start and step default to `None`) and returns a `slice` object. More precisely, the `slice` function is the constructor for `slice` objects. A slice object simply holds three values (start, stop, step).

```
df.loc[('Physics', slice(None)), :]
```

```
                           students  rating
course  lecture
Physics Computer Science         20     3.0
        Mathematics              17     1.6
        Epistemology              3     4.7
```

With `slice(None)` we create a slice object interpreted as *all* (analogously to `:`).

Slicing in the first level works, too.

```
df = df.sort_index()
df.loc[(slice('Mathematics', 'Physics'), 'Epistemology'), :]
```

```
                           students  rating
course      lecture
Mathematics Epistemology          8     3.6
Philosophie Epistemology         89     1.1
Physics     Epistemology          3     4.7
```

Note that label based slicing above requires a sorted index. Thus, we have to call `sort_index`[219] first.

An alternative to `slice` is creating a `pd.IndexSlice` object, which allows for natural slicing syntax:

```
df.loc[pd.IndexSlice['Mathematics':'Physics', 'Epistemology'], :]
```

```
                           students  rating
course      lecture
Mathematics Epistemology          8     3.6
Philosophie Epistemology         89     1.1
Physics     Epistemology          3     4.7
```

[219] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sort_index.html

Building a Mask Based on a Level

The `get_level_values`[220] method of an index object takes a level name or level index as argument and returns a simple `Index` object containing only the labels at the specified level. This object can then be used to create a boolean array for row indexing.

```
import numpy as np

no_physics_mask = df.index.get_level_values('course') != 'Physics'
no_epistemology_mask = df.index.get_level_values(1) != 'Epistemology'

mask = np.logical_and(no_physics_mask, no_epistemology_mask)

df.loc[mask, :]
```

```
                                    students  rating
course      lecture
Mathematics Computer Science              10     2.1
            Mathematics                   15     1.3
Philosophie Computer Science               2     3.9
            Mathematics                    1     4.9
```

Comparing an index object with a single value results in a one-dimensional boolean NumPy array with same length as the index object. NumPy's `logical_and` method implements elementwise *logical and*.

Cross-Sections

There's a shorthand for selecting all rows with a given label at a given level: `xs`[221]. This method takes a label and a level and returns corresponding rows as data frame.

```
df.xs('Mathematics', level='lecture')
```

```
             students  rating
course
Mathematics        15     1.3
Philosophie         1     4.9
Physics            17     1.6
```

The `level` Keyword Argument

Many Pandas functions accept a `level` keyword argument (like `xs` above or `drop`[222]) to provide functionality adapted to multi-level indices.

[220] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Index.get_level_values.html
[221] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.xs.html
[222] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.drop.html

More on Multi-Level Indexing

There are many different styles for using multi-level indexing. Some of them are very confusing for beginners, because same syntax may have different semantics depending on the objects (row/column label, tuple, list) passed as arguments. Here we only considered save and syntactically clear variants. To get an idea of other indexing styles have a look at MultiIndex / advanced indexing[223] in the Pandas user guide.

18.4 Dates and Times

We already met the `datetime` module in *Web Access* (page 132) for handling points of time and time durations. Pandas extends those capabilities by introducing time periods (durations associated with a point) and more advanced calendar arithmetics.

Pandas also provides date and time related index objects to easily index time series data: `DatetimeIndex`, `TimedeltaIndex`, `PeriodIndex`.

```
import pandas as pd
```

18.4.1 Time Stamps

The basic data structure for representing points in time are `Timestamp`[224] objects. They provide lots of useful methods for conversion from and to other date and time formats.

```
some_day = pd.Timestamp(year=2020, month=2, day=15, hour=12, minute=34)
some_day
```

```
Timestamp('2020-02-15 12:34:00')
```

18.4.2 Time Deltas

The basic data structure for representing durations are `Timedelta`[225] objects. They can be used in their own or to shift time stamps.

```
a_long_time = pd.Timedelta(days=10000, minutes=100)
a_long_time
```

```
Timedelta('10000 days 01:40:00')
```

18.4.3 Periods

A period in Pandas is a time interval paired with a time stamp. Interpretation is as follows:

- The interval is one of several preset intervals, like a calendar month or a week from Monday till Sunday. See Offset aliases[226] and Anchored aliases[227] for available intervals.
- The time stamp selects a concrete interval, the month or the week containing the time stamp, for instance.

The basic data structure for representing periods are `Period`[228] objects.

[223] https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html
[224] https://pandas.pydata.org/docs/reference/api/pandas.Timestamp.html
[225] https://pandas.pydata.org/docs/reference/api/pandas.Timedelta.html
[226] https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases
[227] https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#anchored-offsets
[228] https://pandas.pydata.org/docs/reference/api/pandas.Period.html

```
year2010 = pd.Period('1/1/2010', freq='A')
print(year2010.start_time)
print(year2010.end_time)
```

```
2010-01-01 00:00:00
2010-12-31 23:59:59.999999999
```

```
year2010oct = pd.Period('1/1/2010', freq='A-OCT')
print(year2010oct.start_time)
print(year2010oct.end_time)
```

```
2009-11-01 00:00:00
2010-10-31 23:59:59.999999999
```

18.4.4 Time Stamp Indices

Using time stamps for indexing offers lots of nice features in Pandas, because Pandas originally has been developed for handling time series.

Creating Time Stamp Indices

The constructor for `DatetimeIndex`[229] objects takes a list of time stamps and an optional frequency. Frequency has to match the passed time stamps. If there is no common frequency in the data, the frequency is `None` (default).

A more convenient method is `pd.date_range`[230]:

```
index = pd.date_range(start='2018-03-14', freq='2D12H', periods=10)
index
```

```
DatetimeIndex(['2018-03-14 00:00:00', '2018-03-16 12:00:00',
               '2018-03-19 00:00:00', '2018-03-21 12:00:00',
               '2018-03-24 00:00:00', '2018-03-26 12:00:00',
               '2018-03-29 00:00:00', '2018-03-31 12:00:00',
               '2018-04-03 00:00:00', '2018-04-05 12:00:00'],
              dtype='datetime64[ns]', freq='60H')
```

```
index = pd.date_range(start='2018-03-14', end='2018-03-20', freq='D')
index
```

```
DatetimeIndex(['2018-03-14', '2018-03-15', '2018-03-16', '2018-03-17',
               '2018-03-18', '2018-03-19', '2018-03-20'],
              dtype='datetime64[ns]', freq='D')
```

We may also use columns of an existing data frame to create a `DatetimeIndex`:

```
df = pd.DataFrame({'day': [1, 2, 3, 4], 'month': [4, 6, 9, 9], 'year': [2018,␣
↪2018, 2020, 2020]})
display(df)

pd.to_datetime(df)
```

[229] https://pandas.pydata.org/docs/reference/api/pandas.DatetimeIndex.html
[230] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.date_range.html

```
   day  month  year
0    1      4  2018
1    2      6  2018
2    3      9  2020
3    4      9  2020
```

```
0   2018-04-01
1   2018-06-02
2   2020-09-03
3   2020-09-04
dtype: datetime64[ns]
```

The `to_datetime`[231] function expects that columns are named `'day'`, `'month'`, `'year'`. It returns a series of type `datetime64` which may be converted to a `DatetimeIndex`.

Indexing

Exact Indexing

Using `Timestamp` objects for label based indexing yields items with the corresponding time stamp, if there are any. Slicing works as usual.

```
index = pd.date_range(start='2018-03-14', freq='D', periods=10)
s = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], index=index)
print(s, '\n')

print(s.loc[pd.Timestamp('2018-3-16')], '\n')
#print(s.loc[pd.Timestamp('2018-3-16 10:00')], '\n')    # KeyError
print(s.loc[pd.Timestamp('2018-3-16 00:00')], '\n')
print(s.loc[pd.Timestamp('2018-3-16'):pd.Timestamp('2018-3-20')])
```

```
2018-03-14     1
2018-03-15     2
2018-03-16     3
2018-03-17     4
2018-03-18     5
2018-03-19     6
2018-03-20     7
2018-03-21     8
2018-03-22     9
2018-03-23    10
Freq: D, dtype: int64

3

3

2018-03-16    3
2018-03-17    4
2018-03-18    5
2018-03-19    6
2018-03-20    7
Freq: D, dtype: int64
```

[231] https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html

Inexact Indexing

Passing strings containing partial dates/times selects time ranges. This technique is referred to as *partial string indexing*. Slicing is allowed.

```
index = pd.date_range(start='2018-03-14', freq='D', periods=100)
s = pd.Series(range(1, len(index) + 1), index=index)
print(s, '\n')

print(s.loc['2018-3'], '\n')
print(s.loc['2018-3':'2018-4'])
```

```
2018-03-14      1
2018-03-15      2
2018-03-16      3
2018-03-17      4
2018-03-18      5
             ...
2018-06-17     96
2018-06-18     97
2018-06-19     98
2018-06-20     99
2018-06-21    100
Freq: D, Length: 100, dtype: int64

2018-03-14     1
2018-03-15     2
2018-03-16     3
2018-03-17     4
2018-03-18     5
2018-03-19     6
2018-03-20     7
2018-03-21     8
2018-03-22     9
2018-03-23    10
2018-03-24    11
2018-03-25    12
2018-03-26    13
2018-03-27    14
2018-03-28    15
2018-03-29    16
2018-03-30    17
2018-03-31    18
Freq: D, dtype: int64

2018-03-14     1
2018-03-15     2
2018-03-16     3
2018-03-17     4
2018-03-18     5
2018-03-19     6
2018-03-20     7
2018-03-21     8
2018-03-22     9
2018-03-23    10
2018-03-24    11
2018-03-25    12
2018-03-26    13
2018-03-27    14
2018-03-28    15
2018-03-29    16
```

(continues on next page)

(continued from previous page)

```
2018-03-30    17
2018-03-31    18
2018-04-01    19
2018-04-02    20
2018-04-03    21
2018-04-04    22
2018-04-05    23
2018-04-06    24
2018-04-07    25
2018-04-08    26
2018-04-09    27
2018-04-10    28
2018-04-11    29
2018-04-12    30
2018-04-13    31
2018-04-14    32
2018-04-15    33
2018-04-16    34
2018-04-17    35
2018-04-18    36
2018-04-19    37
2018-04-20    38
2018-04-21    39
2018-04-22    40
2018-04-23    41
2018-04-24    42
2018-04-25    43
2018-04-26    44
2018-04-27    45
2018-04-28    46
2018-04-29    47
2018-04-30    48
Freq: D, dtype: int64
```

Inexact indexing has some pitfalls, which are described in Partial string indexing[232] and Slice vs. exact match[233] of the Pandas user guide.

Useful Functions for Time Stamp Indexed Data

Pandas provides lots of functions for working with time stamp indices. Some are:

- `asfreq`[234] (upsampling with fill values or filling logic)
- `shift`[235] (shift index or data by some time period)
- `resample`[236] (downsampling with aggregation, see below)

The `resample` method returns a `Resampler`[237] object, which provides several methods for calculating data values at the new time stamps. Examples are `sum`, `mean`, `min`, `max`. All these methods return a series or a data frame.

```
index = pd.date_range(start='2018-03-14', freq='D', periods=100)
s = pd.Series(range(1, len(index) + 1), index=index)

s2 = s.resample('5D').sum()
s2
```

232 https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#partial-string-indexing
233 https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#slice-vs-exact-match
234 https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.asfreq.html
235 https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.shift.html
236 https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.resample.html
237 https://pandas.pydata.org/pandas-docs/stable/reference/resampling.html

```
2018-03-14     15
2018-03-19     40
2018-03-24     65
2018-03-29     90
2018-04-03    115
2018-04-08    140
2018-04-13    165
2018-04-18    190
2018-04-23    215
2018-04-28    240
2018-05-03    265
2018-05-08    290
2018-05-13    315
2018-05-18    340
2018-05-23    365
2018-05-28    390
2018-06-02    415
2018-06-07    440
2018-06-12    465
2018-06-17    490
Freq: 5D, dtype: int64
```

18.4.5 Period indices

Period indices work analogously to time stamp indices. Corresponding class is `PeriodIndex`[238].

```
index = pd.period_range(start='2018-03-14', freq='D', periods=10)
print(index)

s = pd.Series(range(1, len(index) + 1), index=index)
s
```

```
PeriodIndex(['2018-03-14', '2018-03-15', '2018-03-16', '2018-03-17',
             '2018-03-18', '2018-03-19', '2018-03-20', '2018-03-21',
             '2018-03-22', '2018-03-23'],
            dtype='period[D]')
```

```
2018-03-14     1
2018-03-15     2
2018-03-16     3
2018-03-17     4
2018-03-18     5
2018-03-19     6
2018-03-20     7
2018-03-21     8
2018-03-22     9
2018-03-23    10
Freq: D, dtype: int64
```

Indexing with time stamps selects the appropriate period, like with `IntervalIndex` objects:

```
s.loc[pd.Timestamp('2018-03-15 12:34')]
```

```
2
```

[238] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.PeriodIndex.html

With time stamp index the above line would lead to a `KeyError`. But for periods it's interpreted as: select the period containing the time stamp.

Similar is possible with slicing:

```
s.loc[pd.Timestamp('2018-03-15 12:34'):pd.Timestamp('2018-03-18 23:45')]
```

```
2018-03-15    2
2018-03-16    3
2018-03-17    4
2018-03-18    5
Freq: D, dtype: int64
```

Methods `asfreq`, `shift`, `resample` also work for periods indices.

18.5 Categorical Data

Next to numerical and string data one frequently encounters *categorical data*. That is data of whatever type with finite range. Admissible values are called *categories*. There are two kinds of categorical data:

- *nominal data* (finitely many different values without any order)
- *ordinal data* (finitely many different values with linear order)

Examples:

- colors `red`, `blue`, `green`, `yellow` (nominal)
- business days `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday` (ordnial)

Pandas provides explicit support for categorical data and indices. Major advantages of categorical data compared to string data are lower memory consumption and more meaningful source code.

```
import pandas as pd
```

18.5.1 Creating Categorical Data

Pandas has a class `Categorical` to hold a list of categorical data with (ordinal) or without (nominal) ordering. Such `Categorical` objects can directly be converted to series or columns of a data frame. Almost always category labels are strings, but any other data type is allowed, too.

```
cat_data = pd.Categorical(['red', 'green', 'blue', 'green', 'green'],
                          categories=['red', 'green', 'blue'], ordered=False)

s = pd.Series(cat_data)
s
```

```
0      red
1    green
2     blue
3    green
4    green
dtype: category
Categories (3, object): ['red', 'green', 'blue']
```

Passing `dtype='category'` to series or data frame constructors works, too. Categories then are determined automatically.

```
s = pd.Series(['red', 'green', 'blue', 'green', 'green'], dtype='category')
s
```

```
0      red
1    green
2     blue
3    green
4    green
dtype: category
Categories (3, object): ['blue', 'green', 'red']
```

Or we may convert an existing series or data frame column to categorical type.

```
s = pd.Series(['red', 'green', 'blue', 'green', 'green'])
s = s.astype('category')
s
```

```
0      red
1    green
2     blue
3    green
4    green
dtype: category
Categories (3, object): ['blue', 'green', 'red']
```

Automatically determined categories always are unordered (nominal).

Advantage of ordered categories is that we may use `min` and `max` functions for corresponding data.

```
quality = pd.Series(pd.Categorical(['poor', 'good', 'excellent', 'good', 'very␣
↪good', 'poor'],
                                   categories=['very poor', 'poor', 'good', 'very␣
↪good', 'excellent'],
                                   ordered=True))

print(quality.min())
print(quality.max())
```

```
poor
excellent
```

18.5.2 Custom Categorical Types

Instead of using general `categorical` data type we may define new categorical types. Strictly speaking `categorical` isn't a well defined type because we have to provide the category labels to obtain a full-fledged data type. A more natural way for using categories is to define a data type for each set of categories via `CategoricalDtype`[239].

A further advantage is that the same set of categories can be used for several series and data frames simultaneously.

```
colors = pd.CategoricalDtype(['red', 'green', 'blue', 'yellow'], ordered=False)

s = pd.Series(['red', 'red', 'black', 'blue'], dtype=colors)
s
```

[239] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.CategoricalDtype.html

```
0     red
1     red
2     NaN
3    blue
dtype: category
Categories (4, object): ['red', 'green', 'blue', 'yellow']
```

Values not covered by the categorical type are set to `NaN`.

18.5.3 Encoding Categorical Data for Machine Learning

Most machine learning algorithms expect numerical input. Thus, categorical data has to be converted to numerical data first.

For ordinal data one might use numbers 1, 2, 3,... instead of the original category labels. But for nominal data the natural ordering of integers adds artificial structure to the data, which might affect an algorithm's behavior. Thus, *one hot encoding* usually is used for converting nominal data to numerical data.

The idea is to replace a variable holding one of n categories by n boolean variables. Each new variable corresponds to one category. Exactly one variable is set to `True`. Pandas supports this conversion via `get_dummies`[240] function.

```
colors = pd.CategoricalDtype(['red', 'green', 'blue', 'yellow'], ordered=False)

s = pd.Series(['red', 'red', 'green', 'blue'], dtype=colors)
print(s)

df = pd.get_dummies(s)
df
```

```
0      red
1      red
2    green
3     blue
dtype: category
Categories (4, object): ['red', 'green', 'blue', 'yellow']
```

```
   red  green  blue  yellow
0    1      0     0       0
1    1      0     0       0
2    0      1     0       0
3    0      0     1       0
```

18.5.4 Modifying Categories

Series or data frame columns with categorical data have a `cat` member providing access to the set of categories. Some member functions are:

- `rename_categories`[241] (modify category labels),
- `add_categories`[242] (add category; at the highest position, if ordinal),
- `remove_categories`[243] (remove category, replacing corresponding items by `nan`),
- `union_categories`[244] (join sets of categories).

[240] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html
[241] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.cat.rename_categories.html
[242] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.cat.add_categories.html
[243] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.cat.remove_categories.html
[244] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.api.types.union_categoricals.html

18.5.5 Categorical Data and CSV Files

Information about categories cannot be stored in CSV files. Instead, category labels are written to the CSV file in their native data type. When reading CSV data to a data frame, columns have to be converted to categorical types again, if desired.

18.5.6 Categorical Indices

Pandas supports categorical indices via `CategoricalIndex` objects. Simply pass a `Categorical` object as index when creating a series or a data frame.

```
quality = pd.Categorical(['poor', 'good', 'excellent', 'good', 'very good', 'poor
 ↪'],
                         categories=['very poor', 'poor', 'good', 'very good',
 ↪'excellent'],
                         ordered=True)
s = pd.Series([3, 4, 2, 23, 41, 5], index=quality)
print(s, '\n')

s = s.sort_index()
s
```

```
poor          3
good          4
excellent     2
good         23
very good    41
poor          5
dtype: int64
```

```
poor          3
poor          5
good          4
good         23
very good    41
excellent     2
dtype: int64
```

Data access works as usual.

```
print(s.loc['poor'], '\n')
print(s.loc['poor':'very good'])
```

```
poor    3
poor    5
dtype: int64

poor          3
poor          5
good          4
good         23
very good    41
dtype: int64
```

18.5.7 Categories by Binning

Continuous data or discrete data with too large range can be converted to categories by providing a list of intervals (bins) in which items shall be placed. Each bin can be regarded as a category. Binning is important for machine learning tasks which require discrete data. The `pd.cut`[245] function implements binning.

18.6 Restructuring Data

Restructuring and aggregation are two basic methods for extracting statistical information from data. We start with groupwise aggregation and then discuss several forms of restructuring without and with additional aggregation.

```
import pandas as pd
```

18.6.1 Grouping

Grouping is the first step in the so-called split-apply-combine procedure in data processing. Data is split into groups by some criterion, then some function is applied to each group, finally results get (re-)combinded. Typical functions in the apply step are sum or mean (more general: aggregation) or any type of transform or filtering functions (drop groups containing `nan` items, for instance).

This chapter follows the structure of the Pandas user guide[246], but leaves out sections on very specific details. Feel free to have a look at those details later on.

Splitting into Groups and Basic Usage

Grouping is done by calling the `groupby`[247] method of a series or data frame. It takes a column label or a list of column labels as argument and returns a `SeriesGroupBy` or `DataFrameGroupBy` object. The returned object represents a kind of list of groups, each group being a small series or data frame. All rows in a group have identical values in the columns used for grouping.

The `...GroupBy` object offers several methods for working with the determined groups. Iterating over such objects is possible, too.

Grouping by one column and subsequent aggregation yields an index with values from the column used for grouping:

```
df = pd.DataFrame({'age': [2, 3, 3, 2, 4, 5, 5, 5],
                   'score': [2.3, 4.5, 3.4, 2.0, 5.4, 7.2, 2.8, 3.9]})
display(df)

g = df.groupby('age')

for name, group in g:
    print('age:', name)
    display(group)

df_means  = g.mean()
df_means
```

```
   age  score
0    2    2.3
1    3    4.5
2    3    3.4
```

(continues on next page)

245 https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.cut.html
246 https://pandas.pydata.org/docs/user_guide/groupby.html
247 https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.groupby.html

(continued from previous page)

```
3     2     2.0
4     4     5.4
5     5     7.2
6     5     2.8
7     5     3.9
```

```
age: 2
```

```
   age  score
0    2    2.3
3    2    2.0
```

```
age: 3
```

```
   age  score
1    3    4.5
2    3    3.4
```

```
age: 4
```

```
   age  score
4    4    5.4
```

```
age: 5
```

```
   age  score
5    5    7.2
6    5    2.8
7    5    3.9
```

```
        score
age
2    2.150000
3    3.950000
4    5.400000
5    4.633333
```

Grouping by two columns and subsequent aggregation yields a multi-level index:

```
df = pd.DataFrame({'age': [2, 3, 3, 2, 4, 5, 5, 5],
                   'answer': ['yes', 'no', 'no', 'no', 'no', 'yes', 'yes', 'no'],
                   'score': [2.3, 4.5, 3.4, 2.0, 5.4, 7.2, 2.8, 3.9]})
display(df)

g = df.groupby(['age', 'answer'])

for name, group in g:
    print('age:', name[0])
    print('answer:', name[1])
    display(group)

df_means  = g.mean()
display(df_means)
```

```
   age answer  score
0    2    yes    2.3
1    3     no    4.5
2    3     no    3.4
3    2     no    2.0
4    4     no    5.4
5    5    yes    7.2
6    5    yes    2.8
7    5     no    3.9
```

```
age: 2
answer: no
```

```
   age answer  score
3    2     no    2.0
```

```
age: 2
answer: yes
```

```
   age answer  score
0    2    yes    2.3
```

```
age: 3
answer: no
```

```
   age answer  score
1    3     no    4.5
2    3     no    3.4
```

```
age: 4
answer: no
```

```
   age answer  score
4    4     no    5.4
```

```
age: 5
answer: no
```

```
   age answer  score
7    5     no    3.9
```

```
age: 5
answer: yes
```

```
   age answer  score
5    5    yes    7.2
6    5    yes    2.8
```

```
            score
age answer
2   no       2.00
    yes      2.30
```

(continues on next page)

(continued from previous page)

```
3   no       3.95
4   no       5.40
5   no       3.90
    yes      5.00
```

Grouping by levels of a multi-level index is possible by providing the `level` argument to `groupby`.

With `get_group` we have access to single groups:

```
g.get_group((5, 'yes'))
```

```
   age answer  score
5    5    yes    7.2
6    5    yes    2.8
```

`DataFrameGroupBy` objects allow for column indexing:

```
df = pd.DataFrame({'age': [2, 3, 3, 2, 4, 5, 5, 5],
                   'answer': ['yes', 'no', 'no', 'no', 'no', 'yes', 'yes', 'no'],
                   'score': [2.3, 4.5, 3.4, 2.0, 5.4, 7.2, 2.8, 3.9]})
display(df)

g = df.groupby('age')

g['answer'].get_group(5)
```

```
   age answer  score
0    2    yes    2.3
1    3     no    4.5
2    3     no    3.4
3    2     no    2.0
4    4     no    5.4
5    5    yes    7.2
6    5    yes    2.8
7    5     no    3.9
```

```
5    yes
6    yes
7     no
Name: answer, dtype: object
```

Aggregation

To apply a function to each column of each group use `aggregate`[248]. It takes a function or a list of functions as argument. Providing a dictionary of `column: function` pairs allows for column specific functions.

```
import numpy as np

df = pd.DataFrame({'age': [2, 3, 3, 2, 4, 5, 5, 5],
                   'answer': ['yes', 'no', 'no', 'no', 'no', 'yes', 'yes', 'no'],
                   'score': [2.3, 4.5, 3.4, 2.0, 5.4, 7.2, 2.8, 3.9]})
display(df)

```

(continues on next page)

[248] https://pandas.pydata.org/docs/reference/api/pandas.core.groupby.DataFrameGroupBy.aggregate.html#pandas.core.groupby.DataFrameGroupBy.aggregate

(continued from previous page)

```
g = df.groupby('age')

display(g.aggregate(np.min))
display(g.aggregate([np.min, np.max]))
display(g.aggregate({'answer': np.min, 'score': np.mean}))
```

```
   age answer  score
0    2    yes    2.3
1    3     no    4.5
2    3     no    3.4
3    2     no    2.0
4    4     no    5.4
5    5    yes    7.2
6    5    yes    2.8
7    5     no    3.9
```

```
    answer  score
age
2       no    2.0
3       no    3.4
4       no    5.4
5       no    2.8
```

```
    answer      score
      amin amax  amin amax
age
2       no  yes   2.0  2.3
3       no   no   3.4  4.5
4       no   no   5.4  5.4
5       no  yes   2.8  7.2
```

```
    answer     score
age
2       no  2.150000
3       no  3.950000
4       no  5.400000
5       no  4.633333
```

With `size` we get group sizes.

```
g.size()
```

```
age
2    2
3    2
4    1
5    3
dtype: int64
```

Many aggregation functions are directly accessible from the `...GroupBy` object. Examples are `...GroupBy.sum` and `...GroupBy.mean`. See Computations / descriptive stats[249] for a complete list.

[249] https://pandas.pydata.org/docs/reference/groupby.html#computations-descriptive-stats

Transformation

The `transform`[250] method allows to transform rows groupwise resulting in a data frame with same shape as the original one.

```
df = pd.DataFrame({'age': [2, 3, 3, 2, 4, 5, 5, 5],
                   'answer': ['yes', 'no', 'no', 'no', 'no', 'yes', 'yes', 'no'],
                   'score': [2.3, 4.5, 3.4, 2.0, 5.4, 7.2, 2.8, 3.9]})
display(df)

g = df.groupby('age')

# substract the groups mean score in each age group
df['score'] = g['score'].transform(lambda score: score - score.mean())
df
```

```
   age answer  score
0    2    yes    2.3
1    3     no    4.5
2    3     no    3.4
3    2     no    2.0
4    4     no    5.4
5    5    yes    7.2
6    5    yes    2.8
7    5     no    3.9
```

```
   age answer     score
0    2    yes  0.150000
1    3     no  0.550000
2    3     no -0.550000
3    2     no -0.150000
4    4     no  0.000000
5    5    yes  2.566667
6    5    yes -1.833333
7    5     no -0.733333
```

Filtering

To remove groups use `filter`[251] method. It takes a function as argument and returns a data frame with rows belonging to removed groups removed. The passed function gets the group (series or data frame) and has to return `True` (keep group) or `False` (remove group).

```
df = pd.DataFrame({'age': [2, 3, 3, 2, 4, 5, 5, 5],
                   'answer': ['yes', 'no', 'no', 'no', 'no', 'yes', 'yes', 'no'],
                   'score': [2.3, 4.5, 3.4, 2.0, 5.4, 7.2, 2.8, 3.9]})
display(df)

g = df.groupby('age')

g.filter(lambda dfg: dfg['score'].mean() > 4)
```

```
   age answer  score
0    2    yes    2.3
1    3     no    4.5
```

(continues on next page)

[250] https://pandas.pydata.org/docs/reference/api/pandas.core.groupby.DataFrameGroupBy.transform.html#pandas.core.groupby.DataFrameGroupBy.transform
[251] https://pandas.pydata.org/docs/reference/api/pandas.core.groupby.DataFrameGroupBy.filter.html

(continued from previous page)

```
2    3     no    3.4
3    2     no    2.0
4    4     no    5.4
5    5    yes    7.2
6    5    yes    2.8
7    5     no    3.9
```

```
   age answer  score
4    4     no    5.4
5    5    yes    7.2
6    5    yes    2.8
7    5     no    3.9
```

18.6.2 Restructuring Without Aggregation

There are three basic techniques for restructuring data in a data frame:

- `pivot`[252] (interprets two specified columns as row and column index)
- `stack`[253]/`unstack`[254] (move (level of) column index to (level of) row index and vice versa)
- `melt`[255] (create new column from some column labels)

Details and graphical illustrations of these technique may be found in Pandas' user guide[256] (first three sections).

18.6.3 Restructuring With Aggregation

Pandas supports pivot tables via `pivot_table`[257] function. Pivot tables are almost the same as pivoting with `pivot`[258] but allow for multiple values per data cell, which then are aggregated to one value.

Details may be found in Pandas' user guide[259].

Similar functionality is provided by `crosstab`[260]. See Pandas user guide[261], too.

18.7 Performance Issues

Similar to the discussion in *Efficiency Considerations* (page 183) for NumPy with Pandas we have to take care of how we implement certain operations, at least if performance matters. NumPy guidlines carry over to Pandas, but some additional remarks are in order.

```
import pandas as pd
```

[252] https://pandas.pydata.org/docs/reference/api/pandas.pivot.html
[253] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.stack.html#pandas.DataFrame.stack
[254] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.stack.html#pandas.DataFrame.unstack
[255] https://pandas.pydata.org/docs/reference/api/pandas.melt.html
[256] https://pandas.pydata.org/docs/user_guide/reshaping.html
[257] https://pandas.pydata.org/docs/reference/api/pandas.pivot_table.html
[258] https://pandas.pydata.org/docs/reference/api/pandas.pivot.html
[259] https://pandas.pydata.org/docs/user_guide/reshaping.html#pivot-tables
[260] https://pandas.pydata.org/docs/reference/api/pandas.crosstab.html
[261] https://pandas.pydata.org/docs/user_guide/reshaping.html#cross-tabulations

18.7.1 Vectorization

Analogously to NumPy, in Pandas we should avoid iterating over rows of series or data frames. Almost always vectorization is possible. For numeric columns Pandas relies on NumPy's vectorized function calls. For string and date/time data Pandas implements tailor-made vectorization techniques.

18.7.2 Vectorized String Operations

Indices, series and data frame columns containing string data have a member `str` providing typical string operations. Calling such a method applies the operation to each data item.

```
s = pd.Series(['abc', 'def', 'ghijklmn'])

s.str.upper()
```

```
0         ABC
1         DEF
2    GHIJKLMN
dtype: object
```

See Pandas' user guide[262] for a list of supported string operations.

18.7.3 Vectorized Date/Time Operations

Indices, series and data frame columns containing timestamp data have a member `dt` providing typical date/time operations. Calling such a method applies the operation to each data item.

```
s = pd.Series([pd.Timestamp(2022, 12, 24), pd.Timestamp(2022, 12, 25), pd.
↪Timestamp(2022, 12, 26)])

s.dt.dayofweek
```

```
0    5
1    6
2    0
dtype: int64
```

See Pandas' user guide[263] for available methods.

18.7.4 Accelerating Code Execution

Pandas has a function `eval`[264] which executes Python-like code provided as string. Due to (very complicated CPU caching and other) optimization techniques `eval` is faster for long expressions involving large data frames than standard Python code. The `DataFrame.query`[265] method provides a simplified interface to `eval` for selecting rows via boolean operations on columns.

Both methods should only be used for operations on very large data frames. For small data frames they are significantly slower than standard Python. Have look at Expression evaluation via `eval()`[266] in Pandas' user guide for details.

[262] https://pandas.pydata.org/docs/user_guide/text.html#method-summary
[263] https://pandas.pydata.org/docs/user_guide/basics.html#dt-accessor
[264] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.eval.html
[265] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.query.html
[266] https://pandas.pydata.org/pandas-docs/stable/user_guide/enhancingperf.html#expression-evaluation-via-eval

18.7.5 Very Large Data Sets

Sometimes data sets are too large to load the whole data set to memory. Pandas supports partial loading and there are other Pandas-like Python libraries supporting data sets larger than memory.

Partial Loading

The `pd.read_csv`[267] function supports chunking, that is, loading data in chunks. After processing a chunk it gets removed from memory and the next chunk can be read to memory. See Iterating through files chunk by chunk[268] in Pandas' user guide.

Other Libraries

`Dask`[269] is a parallel computing library with Pandas-like API. It allows for faster processing of large data sets. Have a look at Use other libraries[270] in Pandas' user guide for a quick introduction.

[267] https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html
[268] https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#iterating-through-files-chunk-by-chunk
[269] https://www.dask.org/
[270] https://pandas.pydata.org/pandas-docs/stable/user_guide/scale.html#use-other-libraries

Part V

Data Visualization

CHAPTER

NINETEEN

MATPLOTLIB

The Python standard library contains no modules for visualizing functions (plotting) and other types of data. But there is a de-facto standard: Matplotlib[271], providing lots of plot types and additional visualization features. It integrates well with *Efficient Computations with NumPy* (page 165) and is the Swiss army knife of visualization tools.

- *Matplotlib Basics* (page 251)
- *3D Plots* (page 282)
- *Animations* (page 287)
- *Seaborn* (page 292)
- *Maps* (page 296)

Related exercises:

- *Matplotlib Basics* (page 891)
- *Advanced Matplotlib* (page 893)

Related projects:

- *Chemnitz Trees* (page 961)
- *Weather* (page 923)
  - *Weather Animation* (page 928)

19.1 Matplotlib Basics

Matplotlib provides two interfaces for plotting:

- MATLAB[272] like state-based interface,
- object-oriented interface.

19.1.1 State-Based Plotting

The state-based interface is known as `pyplot`[273].

Data to be plotted is passed to Matplotlib as NumPy arrays or other array-like types. Thus, we need the following standard imports for plotting.

```
import numpy as np
import matplotlib.pyplot as plt
```

[271] https://matplotlib.org/
[272] https://en.wikipedia.org/wiki/MATLAB
[273] https://matplotlib.org/stable/api/pyplot_summary.html

We first need a place where all the plotting is done. Matplotlib calls this a *figure*. To create one call `figure`[274].

```
plt.figure()
```

```
<Figure size 640x480 with 0 Axes>
```

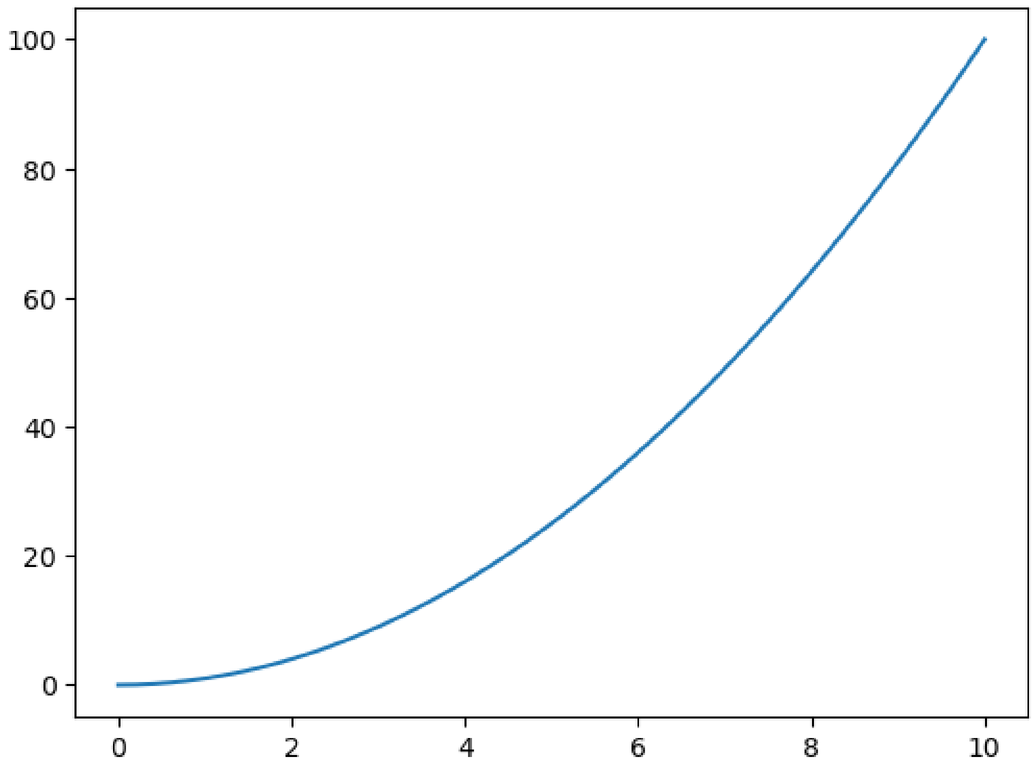
```
<Figure size 640x480 with 0 Axes>
```

We see two lines of output. The first line is the object returned by `plt.figure()`, which is printed by Jupyter because Jupyter always prints the result of the last line of code. The second line in the output is the drawing area, which is replaced by a describing string, because it's empty at the moment.

Simple line plots can be created with `plot`[275].

```
x = np.linspace(0, 10)
y = x ** 2

plt.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x7fa5c6789960>]
```

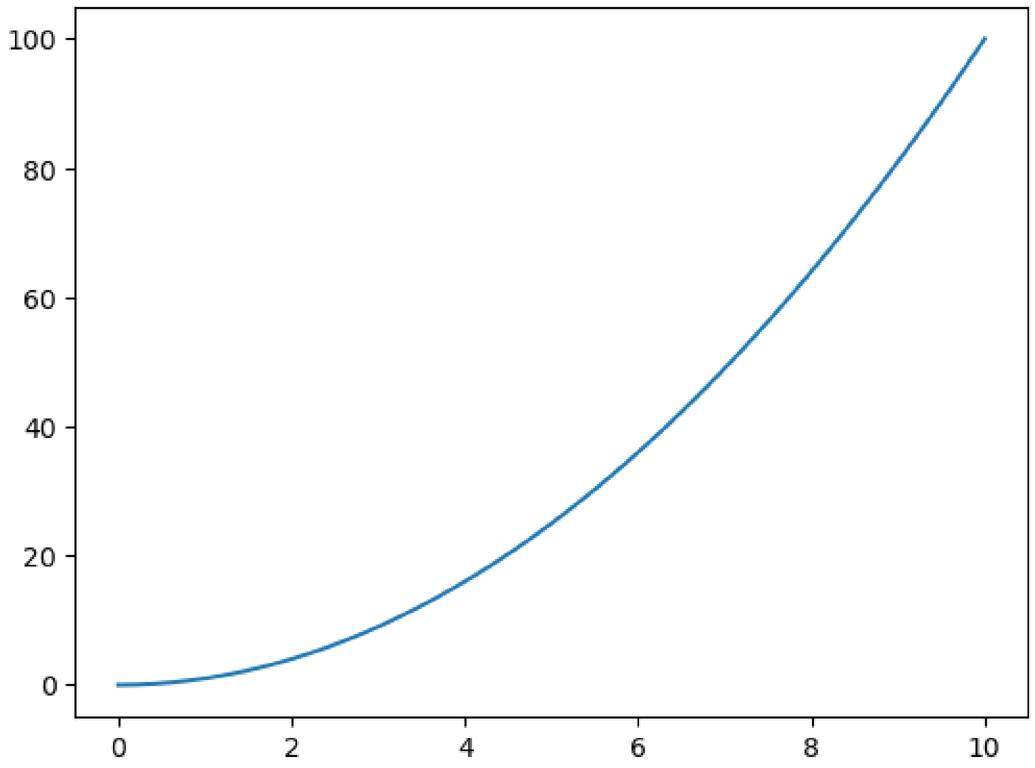


Jupyter automatically copies the drawing area created above by `plt.figure()` to the next code cell.

Note that `plot` only plots in the background. To see the plot on screen one has to call `show`[276]. This is automatically done by Jupyter at the end of each code block containing calls to `pyplot` functions. But before this call Jupyter prints the result of the last code line. If a code block ends with an explicit call to `show`, then jupyter does not produce any automatic output.

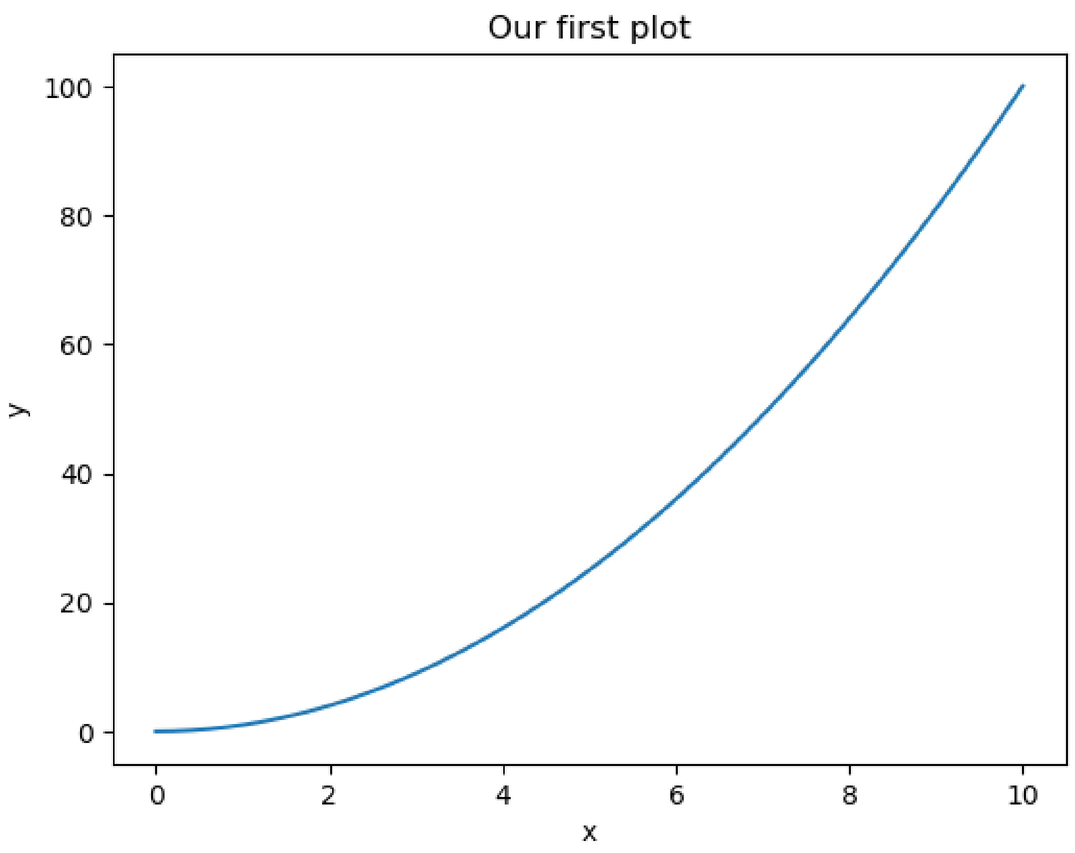
[274] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.figure.html
[275] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
[276] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.show.html

```
plt.plot(x, y)
plt.show()
```



We should add axes labels and a title to the plot.

```
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Our first plot')
plt.show()
```



Above we mentioned that `pyplot` provides a *state-based* interface to Matplotlib. That is, we do not have to tell `pyplot` to which plot we want to add a title. Instead every operation applies to the *current* plot. Having multiple plots (see below) we have to take care about what `pyplot`'s current plot is when calling functions like `xlabel`[277], `ylabel`[278] or `title`[279]. But for simple plotting tasks the state-based interface requires fewer lines of code than the object-oriented interface.

Hint: Plotting in simple Python shell

In a simple Python shell the plot does not automatically show up after calling `plt.plot`. To see the plot we have to call `show`.

The `show` function not only shows the plot, but also stops execution of the script (i.e., blocks the shell) until the plot window is closed.

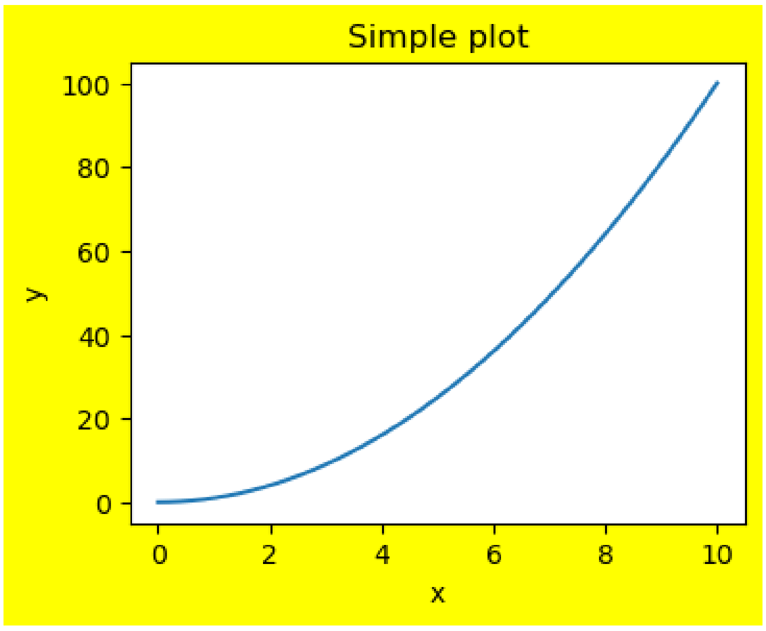
19.1.2 Object-Oriented Plotting

To use the object-oriented interface of Matplotlib one first creates an empty figure with `pyplot` and then starts to fill it with objects.

To get a simple line plot we first have to create an `Axes`[280] object, which encapsulates the coordinate system and all its surroundings. Then we can add a `Line2D`[281] object via `Axes.plot`[282].

[277] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.xlabel.html
[278] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.ylabel.html
[279] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.title.html
[280] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.html
[281] https://matplotlib.org/stable/api/_as_gen/matplotlib.lines.Line2D.html
[282] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.plot.html

```
fig = plt.figure(facecolor='yellow')
ax = fig.add_axes((0.25, 0.25, 0.5, 0.5))

ax.plot(x, y)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('Simple plot')

plt.show()
```



Important: The four values passed as a tuple to `Figure.add_axes`[283] describe the position of the left boundary, the right boundary, the width and the height of the `Axes` object relative to the figure's width and height. So we would expect to see an `Axes` object filling half the width and height of the yellow area and positioned at one quarter of width and also of height, that is, centered. To show the result of plotting operations Jupyter exports the figure to an image file and then displays that image file. For exporting the figure size is adapted to the figures content. Thus, in a Jupyter notebook we only see the `Axes` object without wide yellow boundary. Same code in a simple Python shell produces different output!

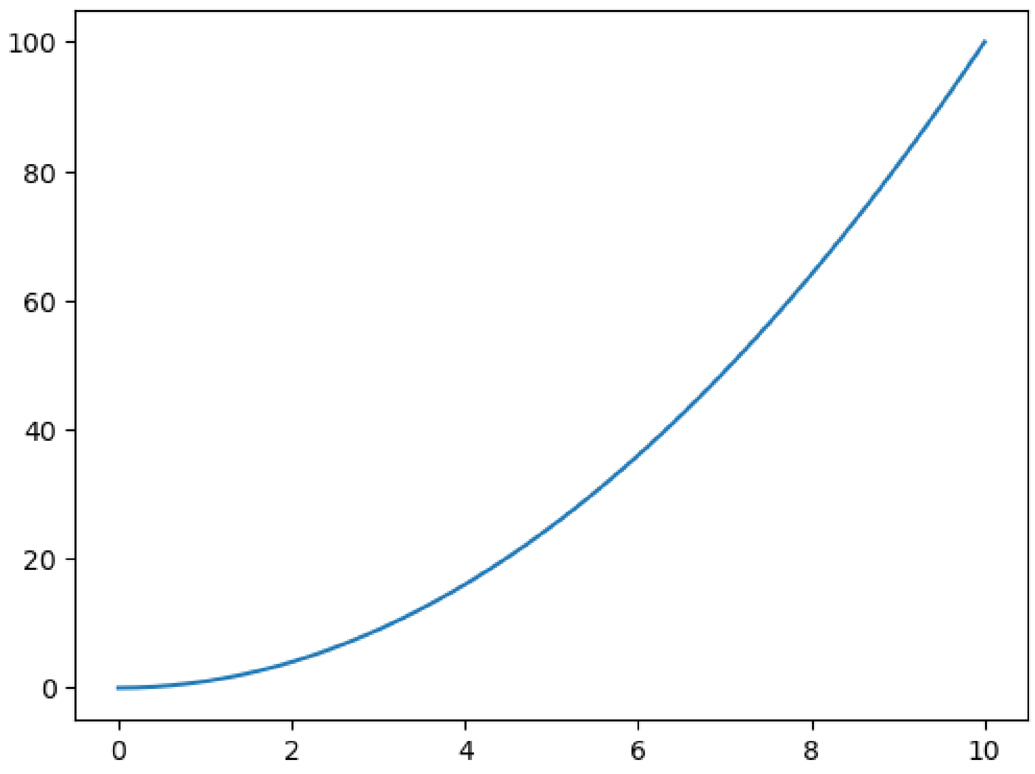
Similar issue: The four values passed in the first argument to `add_axes` specify position and dimensions of the drawing area. Ticks and labels lie outside this area. Consequently the tuple `(0, 0, 1, 1)` results in a drawing with invisible ticks and labels outside the figure. In Jupyter notebooks this does not work, because Jupyter automatically enlarges the figure to fit the whole `Axes` object including ticks and labels.

Note that the `Axes.plot` function returns a `Line2D` object which can be further processed if needed.

Often it's more convenient to create the figure and the `Axes` object in one step:

```
fig, ax = plt.subplots()
ax.plot(x, y)

plt.show()
```

[283] https://matplotlib.org/stable/api/figure_api.html#matplotlib.figure.Figure.add_axes



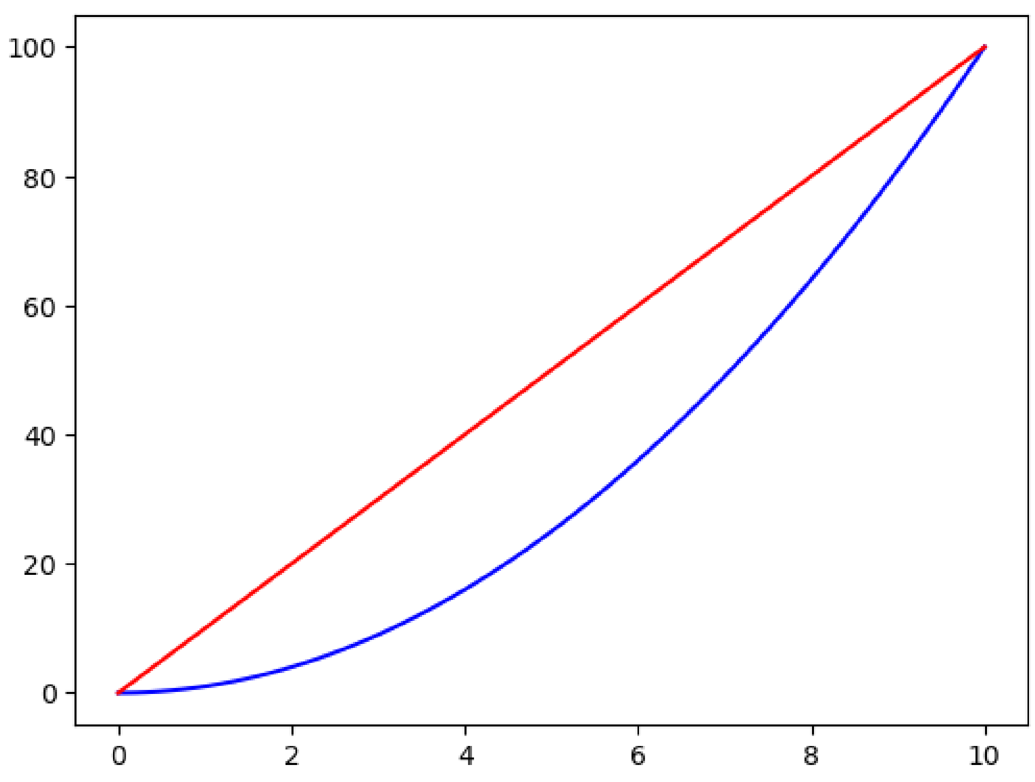
Details on `pyplot.subplots` will be given below.

19.1.3 Multiple Plots

Multiple Plots in One `Axes` Object

Placing more than one line plot (or any other type of plot) in one `Axes` object is straight forward.

```
x = np.linspace(0, 10, 100)
y1 = x ** 2
y2 = 10 * x

fig, ax = plt.subplots()
ax.plot(x, y1, '-b')
ax.plot(x, y2, '-r')

plt.show()
```



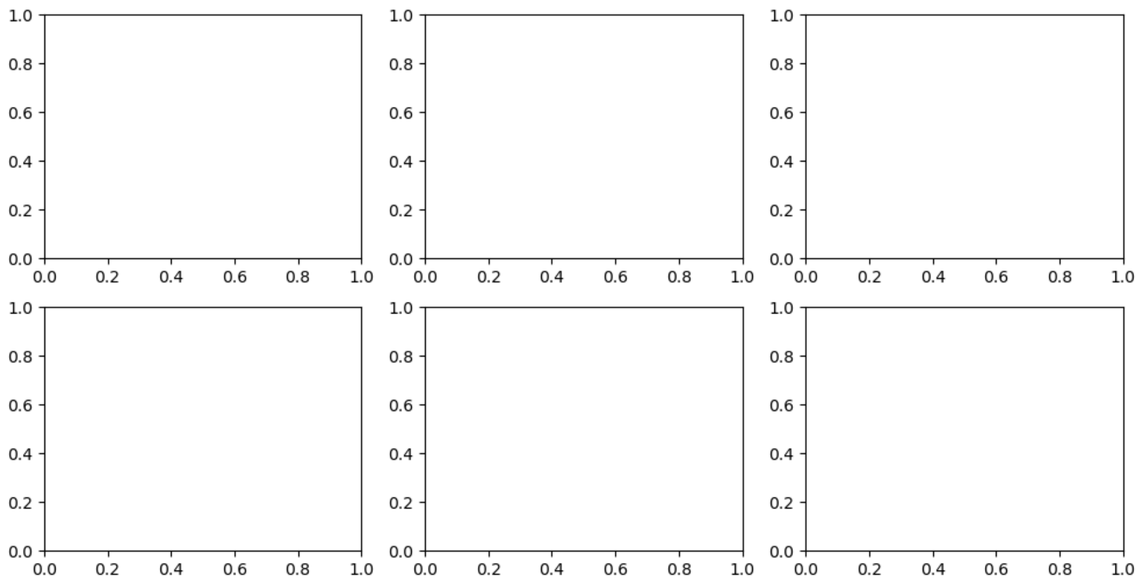
Multiple `Axes` Objects

Multiple `Axes` objects can be placed manually in a figure with `Figure.add_axes`. But there are methods in Matplotlib which support exact alignment of the `Axes` objects.

To get a grid of equally sized `Axes` objects call `Figure.add_subplot`[284].

```
m = 2    # rows
n = 3    # columns

fig = plt.figure(figsize=(12, 6))

ax = m * n * [None]    # will hold Axes objects of subplots
for k in range(m * n):
    ax[k] = fig.add_subplot(m, n, k+1)

plt.show()
```

[284] https://matplotlib.org/stable/api/figure_api.html#matplotlib.figure.Figure.add_subplot



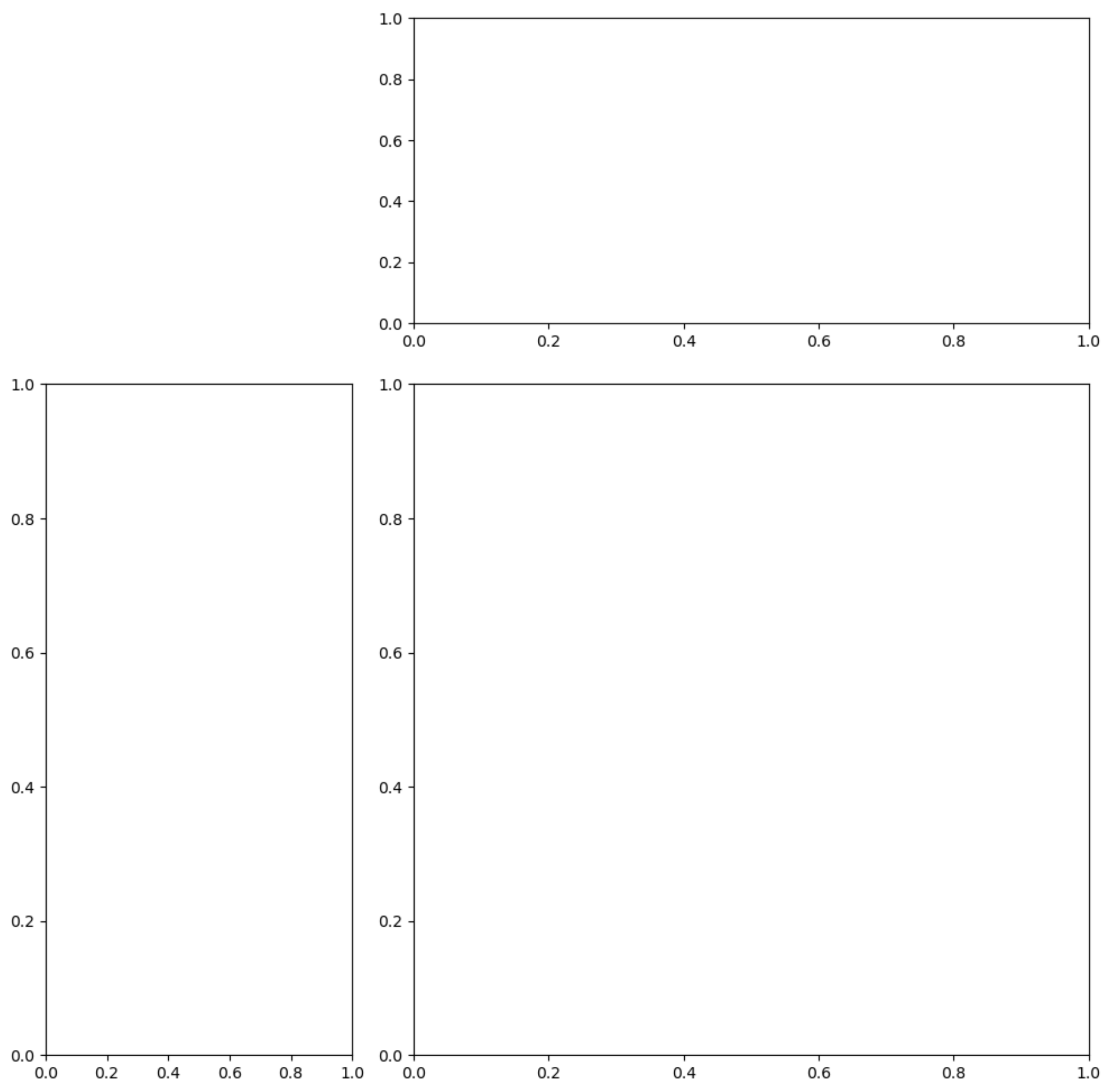
The third argument to `add_subplot` specifies the position in the grid. Subplots are numbered starting with 1 in the upper left corner, then continuing to the right and then to the next row.

Hint: The `pyplot.subplots`[285] method creates a new figure and grid of `Axes` objects. It returns the `Figure` object and a list of `Axes` objects.

More advanced grid layouts with subplots occupying more than one cell can be created with `Figure.add_gridspec`[286]. This method returns a `GridSpec`[287] object, which then can be used to specify the cells occupied by a subplot via NumPy style indexing and slicing.

```
fig = plt.figure(figsize=(12, 12))

gs = fig.add_gridspec(3, 3)

ax_left = fig.add_subplot(gs[1:, 0])
ax_top = fig.add_subplot(gs[0, 1:])
ax_center = fig.add_subplot(gs[1:, 1:])

plt.show()
```

[285] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html
[286] https://matplotlib.org/stable/api/figure_api.html#matplotlib.figure.Figure.add_gridspec
[287] https://matplotlib.org/stable/api/_as_gen/matplotlib.gridspec.GridSpec.html

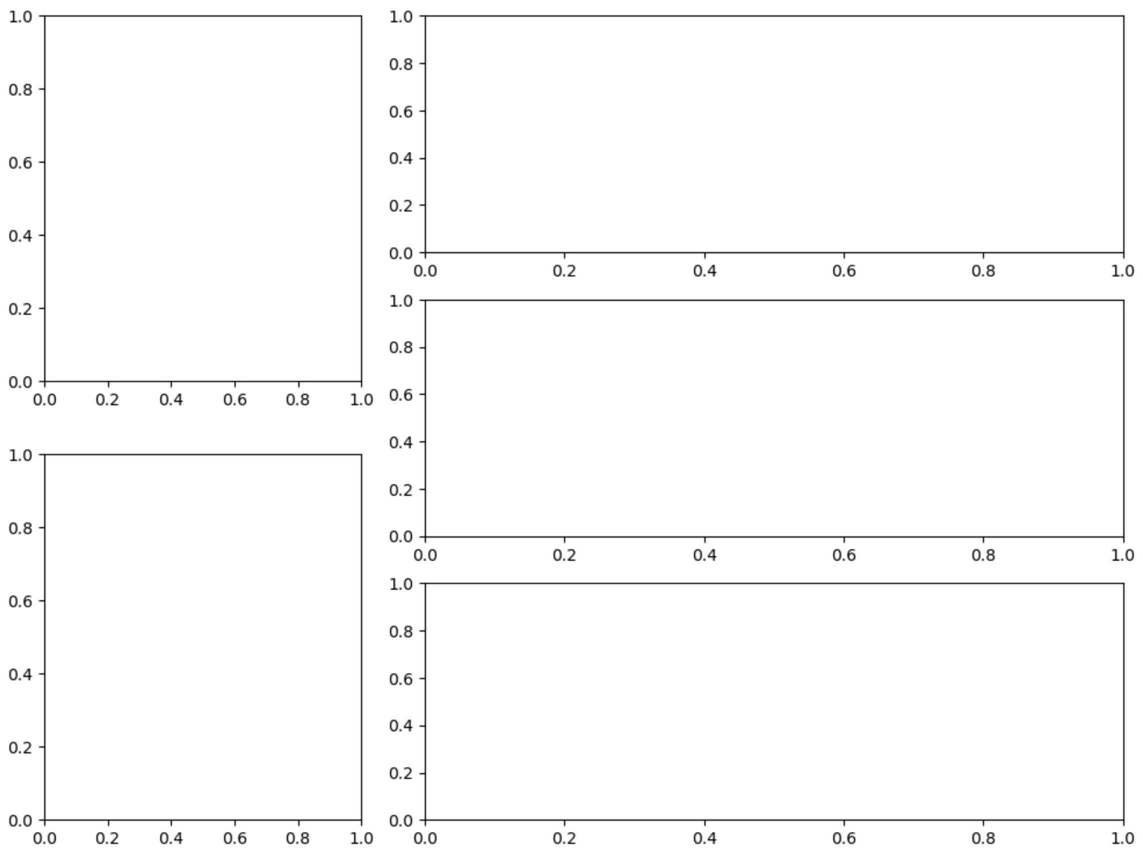


Indexing a `GridSpec` object returns a `SubplotSpec`[288] object which can be passed to `Axes.add_subplot`.

Subplots can be nested with `SubplotSpec.subgridspec`[289]. This method return a `GridSpecFromSubplotSpec`[290] object for which indexing returns `SubplotSpec` objects in the same way as for `GridSpec` objects.

```
fig = plt.figure(figsize=(12, 9))

gs = fig.add_gridspec(1, 3)
gs_left = gs[0, 0].subgridspec(2, 1)
gs_right = gs[0, 1:].subgridspec(3, 1)

ax_left_top = fig.add_subplot(gs_left[0, 0])
ax_left_bottom = fig.add_subplot(gs_left[1, 0])
ax_right_top = fig.add_subplot(gs_right[0, 0])
ax_right_middle = fig.add_subplot(gs_right[1, 0])
ax_right_bottom = fig.add_subplot(gs_right[2, 0])

plt.show()
```

[288] https://matplotlib.org/stable/api/_as_gen/matplotlib.gridspec.SubplotSpec.html
[289] https://matplotlib.org/stable/api/_as_gen/matplotlib.gridspec.SubplotSpec.html#matplotlib.gridspec.SubplotSpec.subgridspec
[290] https://matplotlib.org/stable/api/_as_gen/matplotlib.gridspec.GridSpecFromSubplotSpec.html



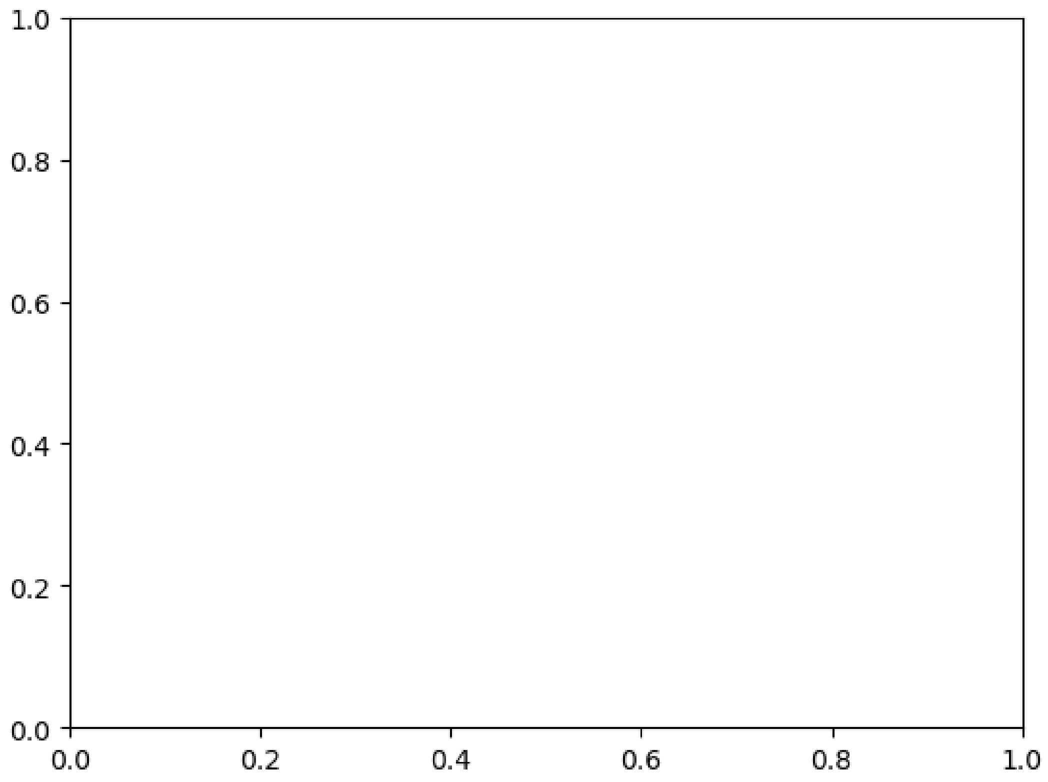
Note: The `Figure.suptitle`[291] methods adds a title to the whole figure, not only to an `Axes` object (linke `Axes.title`).

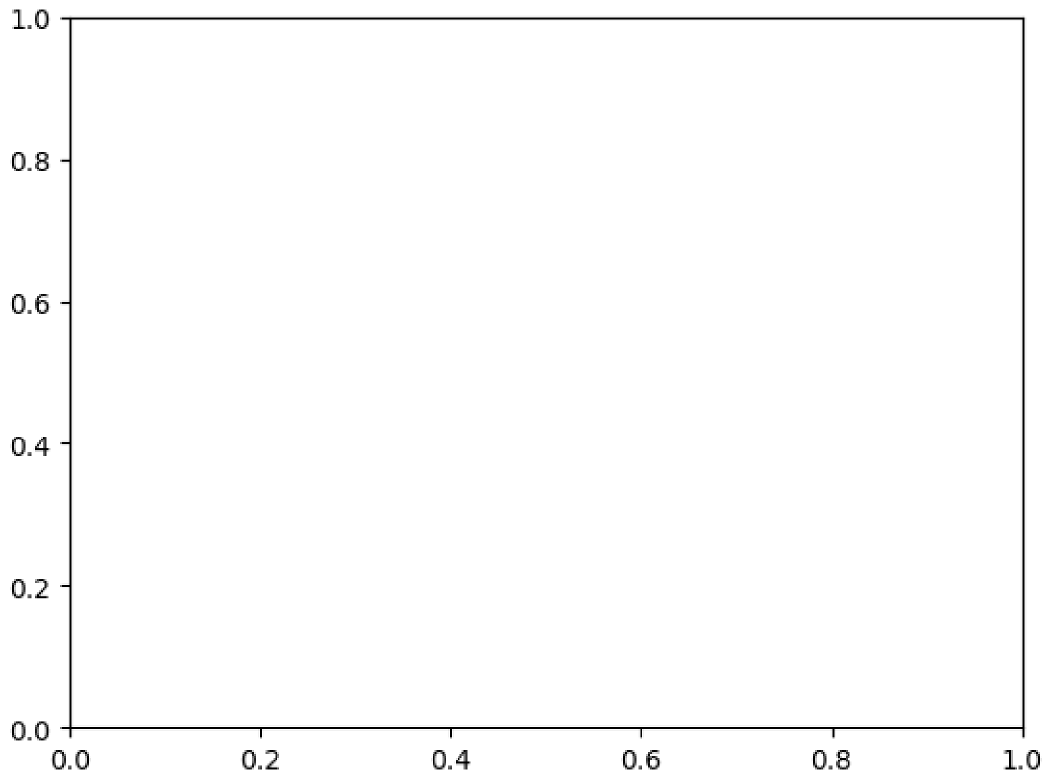
Multiple Figures

It's also possible to generate multiple `Figure` objects. In a simple Python shell this opens one window per figure. In a Jupyter notebook all figures are shown in the output cell.

```
fig1, ax1 = plt.subplots()

fig2, ax2 = plt.subplots()

plt.show()
```

[291] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.suptitle.html


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

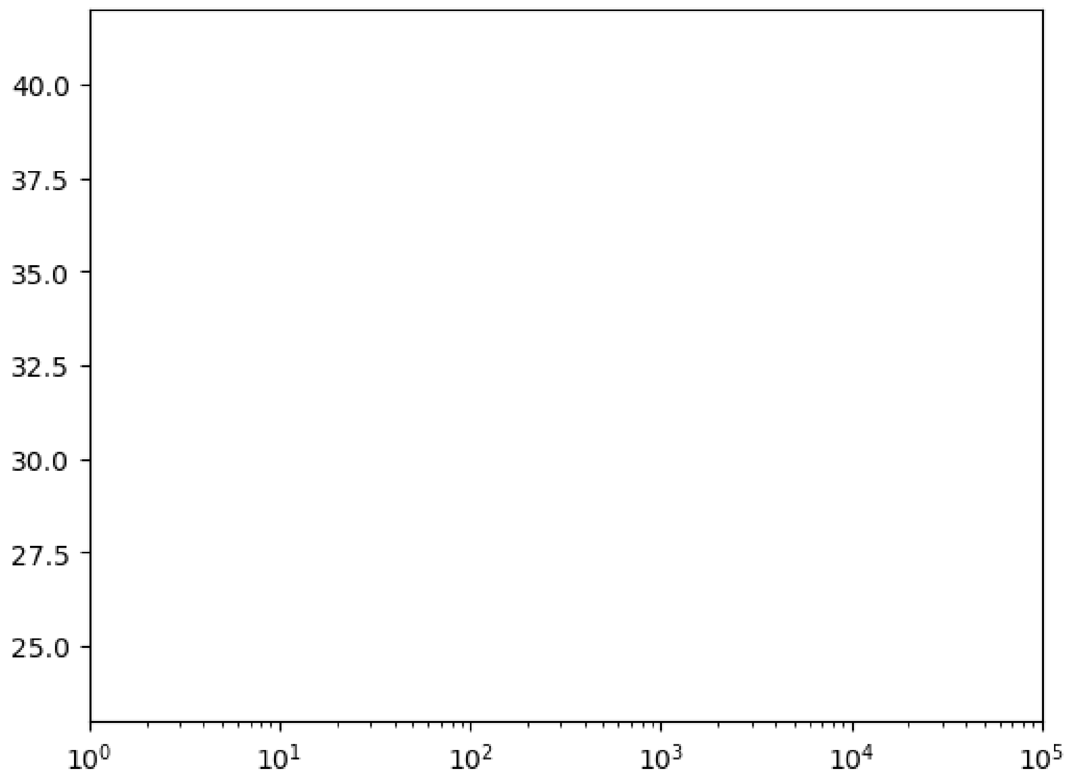
19.1.4 Axis Properties

Scaling and Limits

`Axes` objects provide several different methods for influencing axis scaling (linear, logarithmic) and axis limits (smallest and greatest value). With `Axes.axis`[292] scaling and limits for both axes can be set at once. The `Axes` methods `set_xlim`[293], `set_ylim`[294], `set_xscale`[295], `set_yscale`[296] allow for finer control.

Note that by default Matplotlib automatically sets axis limits to fit the plotted data. This behavior can be deactivated by calling `set_xlim` and `set_ylim` with parameter `auto=False`. Since `False` is the default value for `auto`, each call to `set_xlim` or `set_ylim` without providing the `auto` parameter deactivates automatic limits, too.

```
fig, ax = plt.subplots()

ax.set_xscale('log')
ax.set_xlim(1, 1e5)

ax.set_ylim(23, 42)

plt.show()
```



Direction of coordinate axes can be changed by exchanging upper and lower limits of the axes.

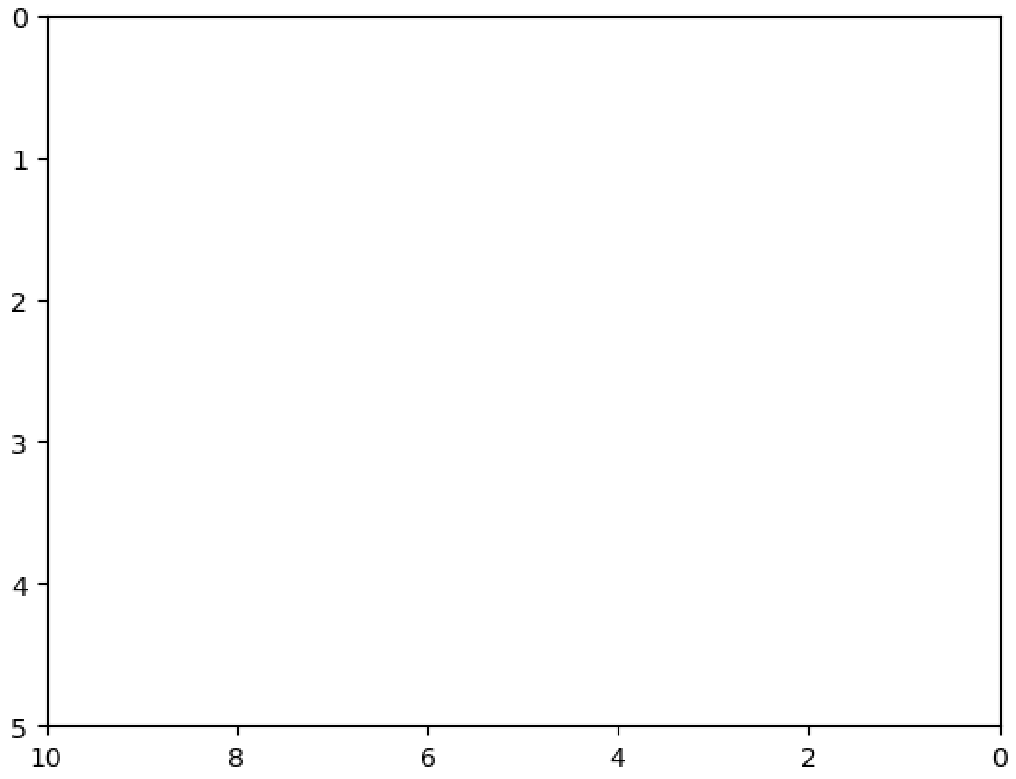
```
fig, ax = plt.subplots()

ax.set_xlim(10, 0)
ax.set_ylim(5, 0)
```

(continues on next page)

[292] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.axis.html
[293] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xlim.html
[294] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_ylim.html
[295] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xscale.html
[296] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_yscale.html

(continued from previous page)

```
plt.show()
```

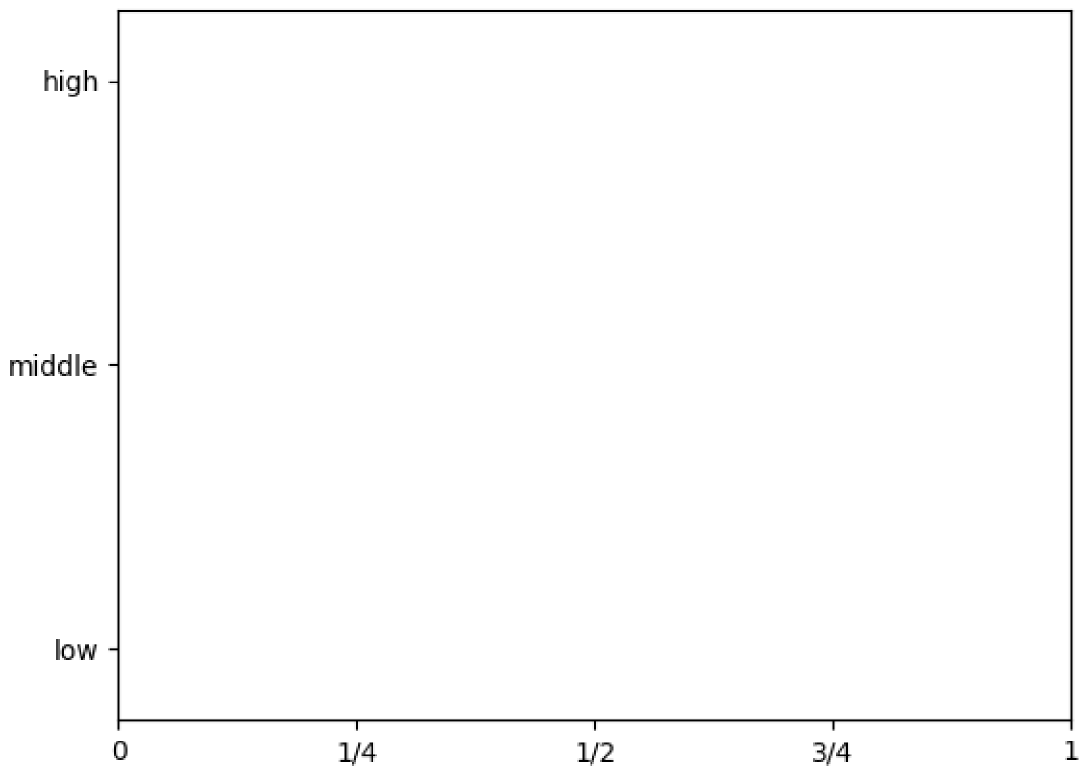


Tick Positions and Labels

To modify tick positions and labels `Axes` objects provide methods `set_xticks`[297], `set_yticks`[298], `set_xticklabels`[299], `set_yticklabels`[300].

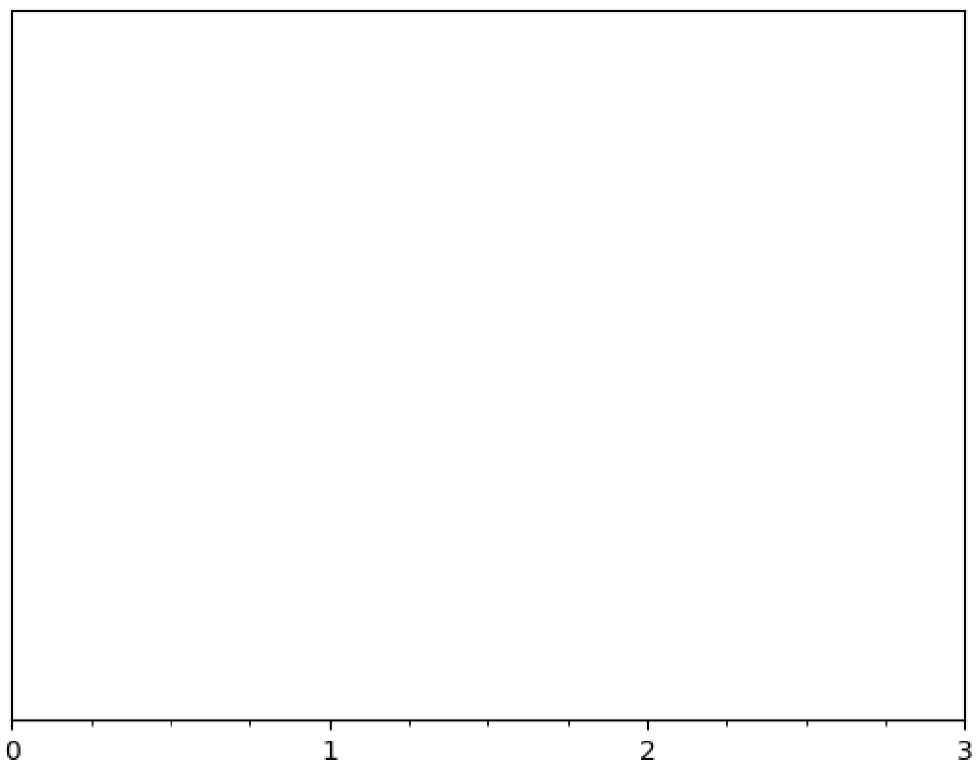
```
fig, ax = plt.subplots()

ax.set_xlim(0, 1)
ax.set_xticks([0, 0.25, 0.5, 0.75, 1])
ax.set_xticklabels(['0', '1/4', '1/2', '3/4', '1'])

ax.set_ylim(0, 1)
ax.set_yticks([0.1, 0.5, 0.9])
ax.set_yticklabels(['low', 'middle', 'high'])

plt.show()
```

[297] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xticks.html
[298] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_yticks.html
[299] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xticklabels.html
[300] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_yticklabels.html



Matplotlib distinguishes minor and major ticks. Passing the parameter `minor` with value `True` or `False` (default) switches between both variants. Passing an empty tick list removes all ticks from the axis.

```
fig, ax = plt.subplots()

ax.set_xlim(0, 3)
ax.set_xticks([0, 1, 2, 3])
ax.set_xticks([0.25, 0.5, 0.75, 1.25, 1.5, 1.75, 2.25, 2.5, 2.75], minor=True)
ax.set_xticklabels([], minor=True)

ax.set_ylim(0, 1)
ax.set_yticks([])

plt.show()
```



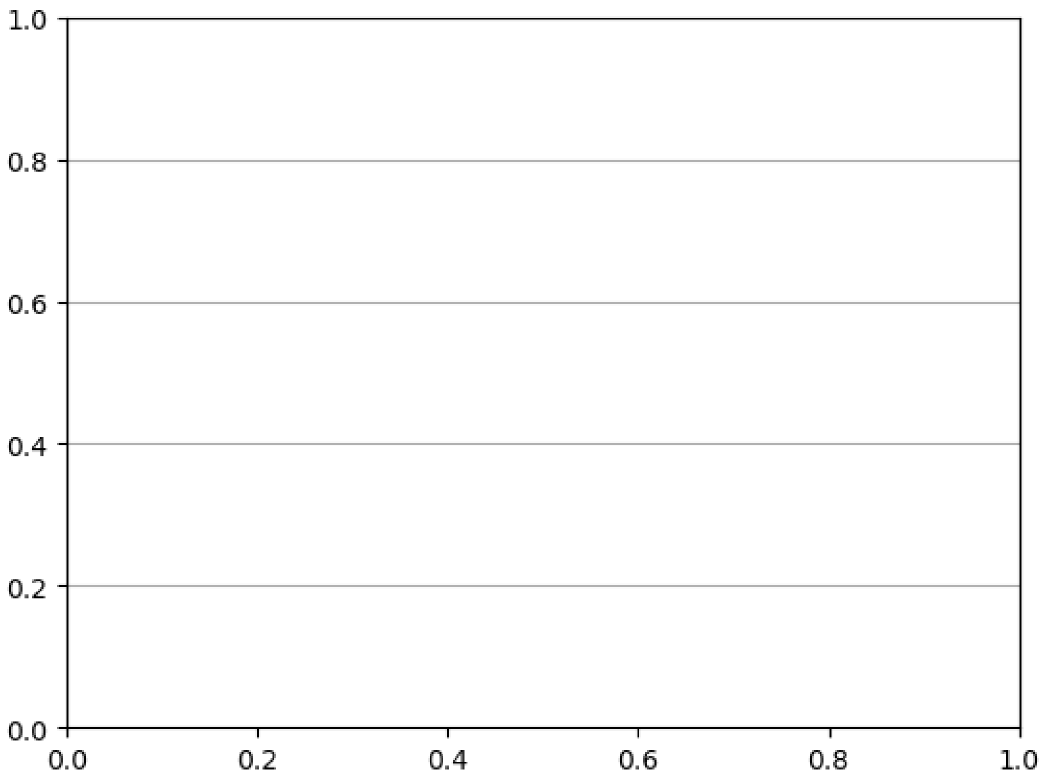
More avanced control of tick and tick label properties is provided by `Axes.tick_params`[301]. There we can specify tick size and color, font and color for labels, rotation of labels and much more.

Grid Lines

Grid lines enhance readability of plots. They can be added and modified with `Axes.grid`[302]. More detailed control is provided by `Axes.tick_params`. Grid line positions always coincide with tick positions.

```
fig, ax = plt.subplots()

ax.grid(axis='y')

plt.show()
```

[301] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.tick_params.html
[302] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.grid.html

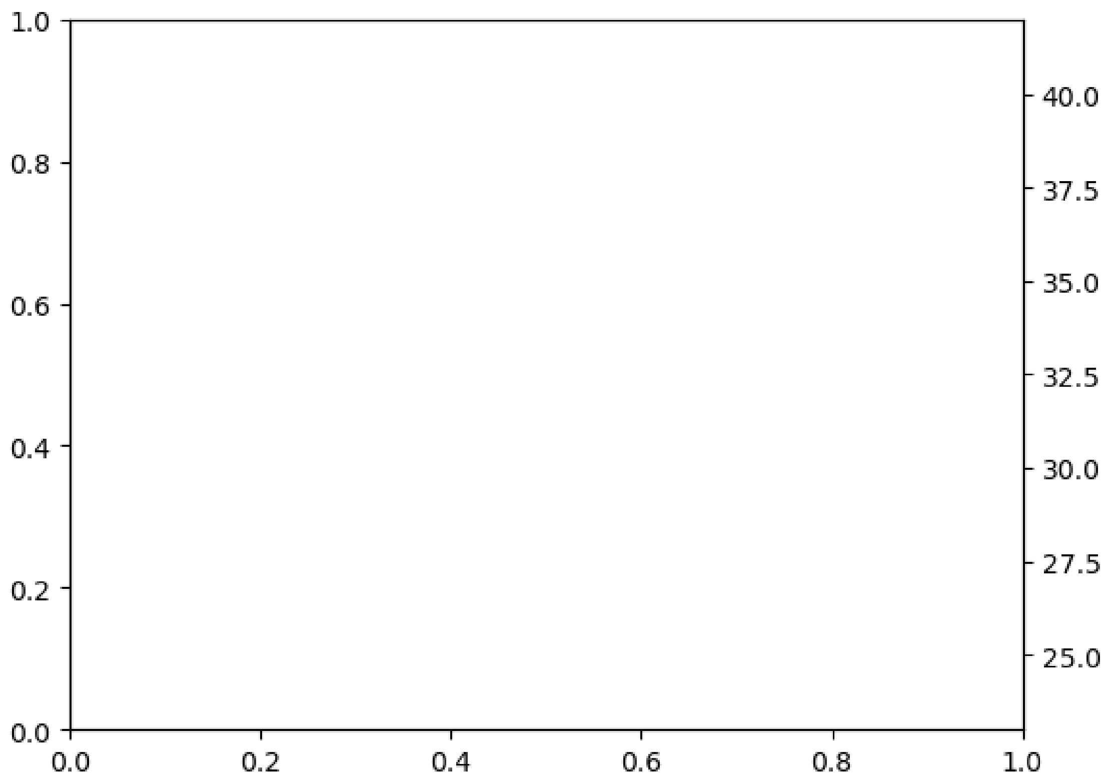


Different Scales

Sometimes one wants to have two plots with different y axis limits in one figure. This can be achieved with `Axes.twiny`[303] (there is also a `twinx`[304]). Such figures then have two different y axes, one with ticks and labels at the left boundary of the drawing area and one with ticks and labels at the right boundary. The `twiny` methods sets this all up for us and returns a new `Axes` object overlaying the original one in a way which gives the desired result.

```
fig, ax1 = plt.subplots()
ax2 = ax1.twinx()

ax1.set_ylim(0, 1)
ax2.set_ylim(23, 42)

plt.show()
```

303 https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.twiny.html
304 https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.twinx.html



Both `Axes` objects share their x axis. Thus, to modify x axis properties it doesn't matter which of both `Axes` objects is modified. To modify y axis properties, corresponding `Axes` object has to be modified. For plotting the plotting methods of the `Axes` object with the correct y axis have to be called.

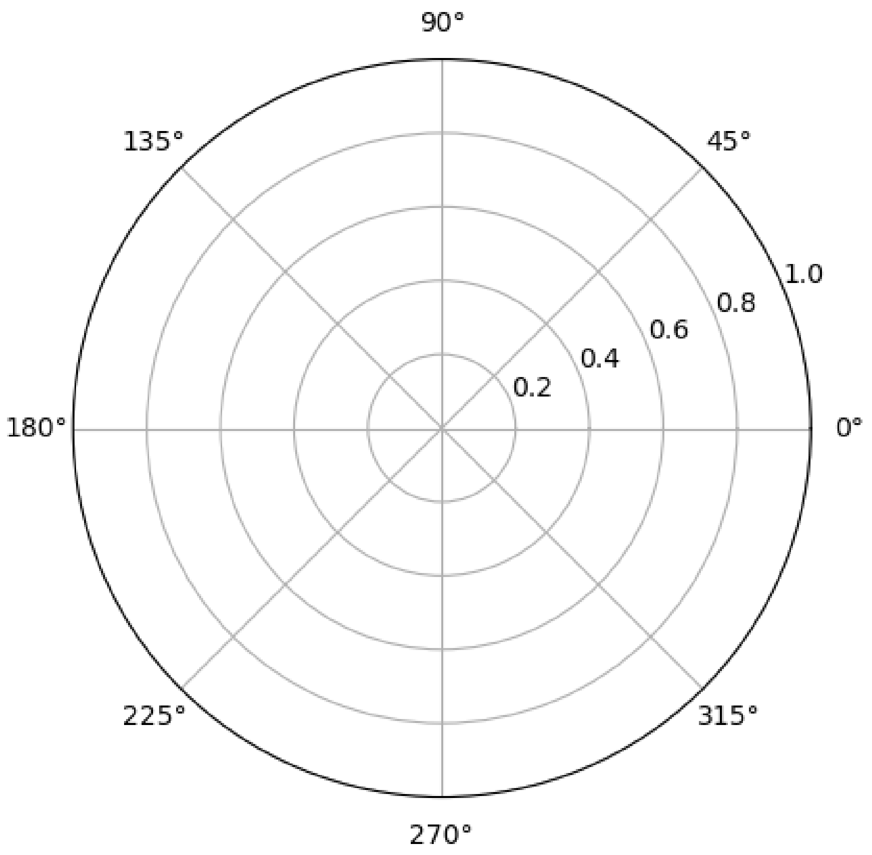
Polar Plots

Matplotlib also provides support for plotting in polar coordinates. To create a polar plot pass the parameter `projection='polar'` when creating an `Axes` object (not supported by all variants for creating `Axes` objects).

Note that the object returned is not really an instance of the `Axes` class, but of `PolarAxes`[305], which is derived from `Axes` and partly comes with different methods.

```
fig = plt.figure()
ax = fig.add_axes((0.1, 0.1, 0.8, 0.8), projection='polar')

plt.show()
```

[305] https://matplotlib.org/stable/api/projections/polar.html#matplotlib.projections.polar.PolarAxes



By default grid lines are turned on and tick labels are adapted to polar coordinates. But everything can be custimized with similar methods as before.

19.1.5 Colors and Colorbars

19.1.6 Specifying Colors

Whenever a Matplotlib function takes a color as argument different formats are accepted. Some of them are:

- tuples with 3 floats between 0 and 1 for red, green, blue components
- tuples with 4 floats between 0 and 1 for red, green, blue components and opacity
- string `'#rrggbb'` where rr, gg, bb are integers from 0 to 255 in hexadecimal notation for red, green, blue component
- string with only one character out of b (blue), g (green), r (red), c (cyan), m (magenta), y (yellow), k (black), w (white)
- string with pre-defined color name like `'white'` or `'red'` (lists of available color names: with prefix `'tab:'`[306], without prefix[307], with prefix `'xkcd:'`[308])

By default Matplotlib uses the `'tab:'` prefixed colors and cycles through them if multiple lines are plotted.

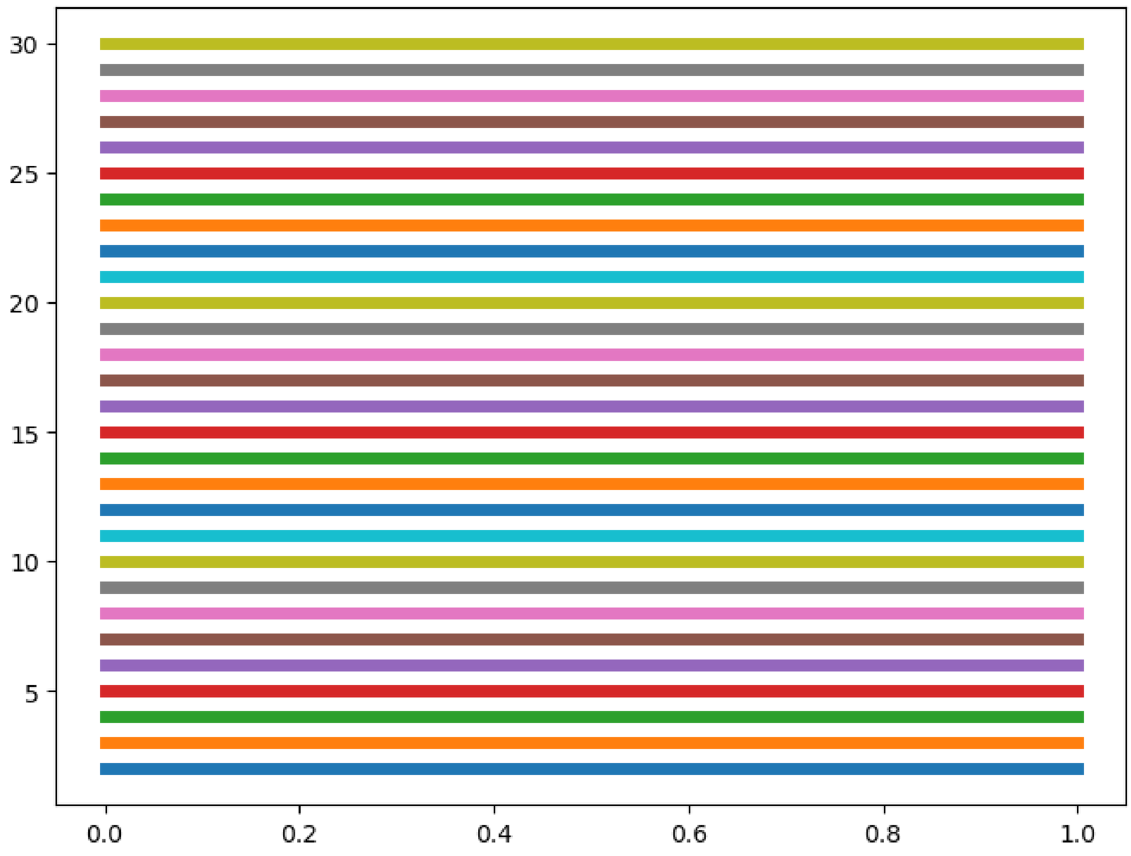
```
fig, ax = plt.subplots(figsize=(8, 6))

x = np.array([0, 1])
y = np.array([1, 1])
```

(continues on next page)

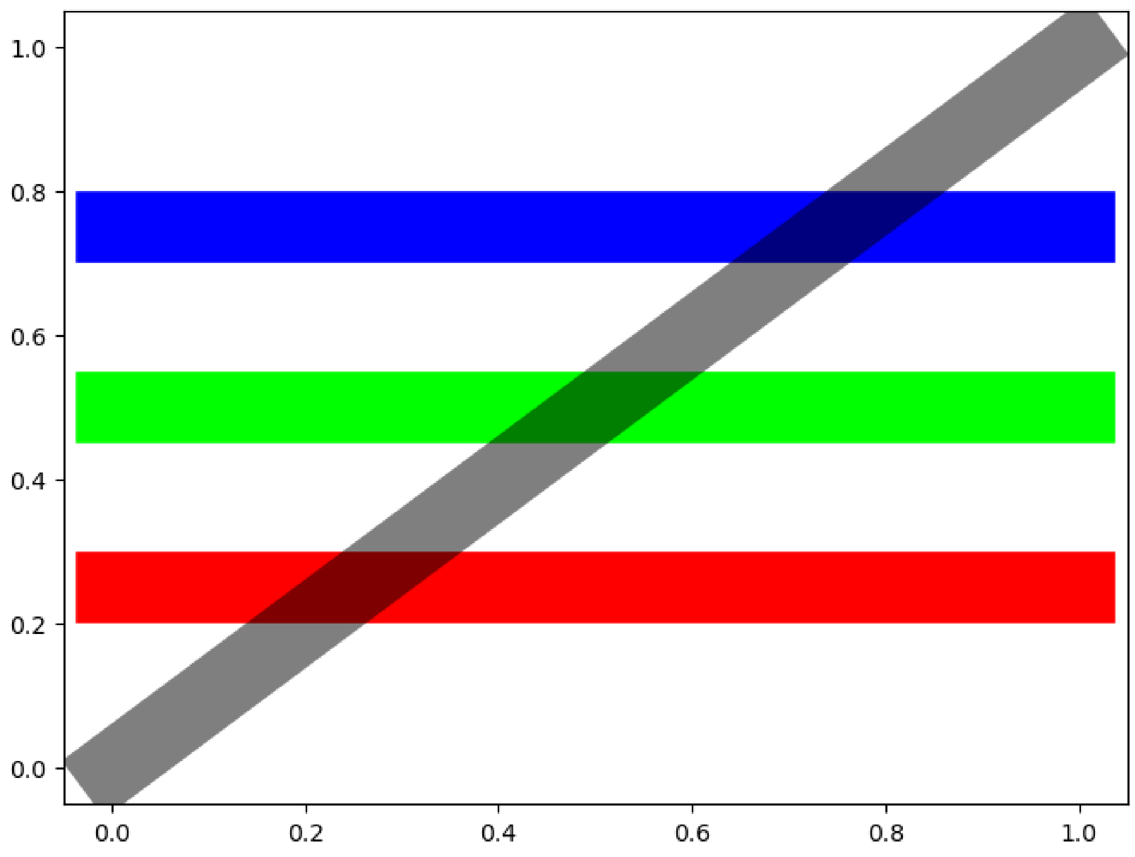
[306] https://matplotlib.org/stable/_images/sphx_glr_named_colors_002.png
[307] https://matplotlib.org/stable/_images/sphx_glr_named_colors_003.png
[308] https://i.stack.imgur.com/nCk6u.jpg

(continued from previous page)

```
for n in range(1, 30):
    ax.plot(x, y + n, '-', linewidth=5)

plt.show()
```



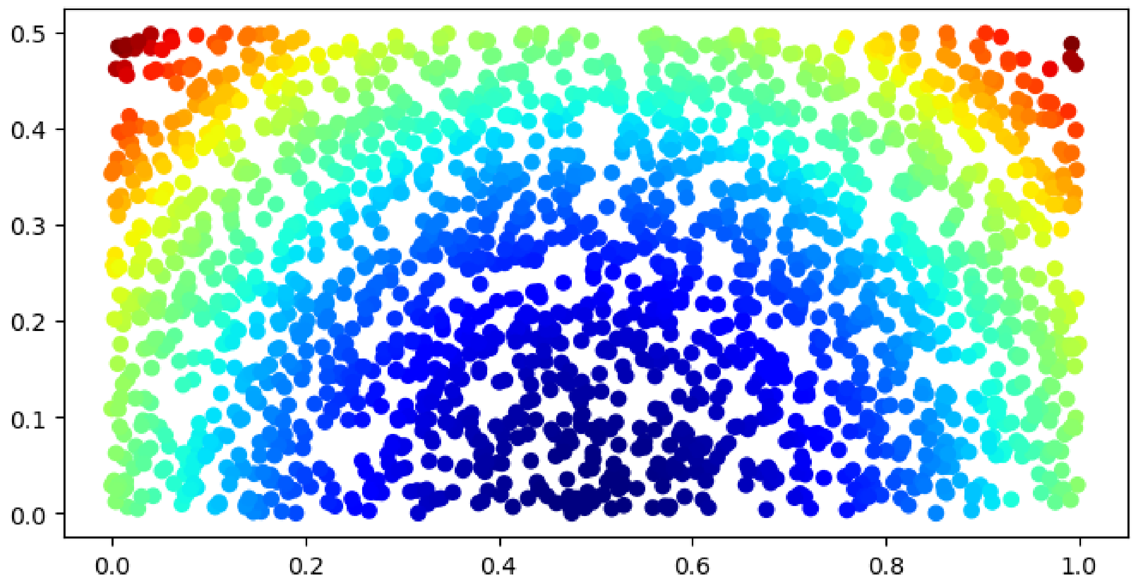
```
fig, ax = plt.subplots(figsize=(8, 6))

x = np.linspace(0, 1, 100)
y = np.ones(x.shape)

ax.plot(x, 0.25 * y, '-', linewidth=30, color='red')
ax.plot(x, 0.5 * y, '-', linewidth=30, color='#00ff00')
ax.plot(x, 0.75 * y, '-', linewidth=30, color=(0, 0, 1))
ax.plot(x, x, '-', linewidth=30, color=(0, 0, 0, 0.5))

plt.show()
```



Converting Data to Colors

Some plot types allow color selection depending on the plotted data.

```
n = 2000     # number of samples

rng = np.random.default_rng(0)

x = rng.uniform(0, 1, n)
y = rng.uniform(0, 0.5, n)
z = (x - 0.5) ** 2 + y ** 2

fig, ax = plt.subplots(figsize=(8, 4))

ax.scatter(x, y, c=z, cmap='jet')

plt.show()
```



The `Axes.scatter`[309] method plots a list of points and colors them according to the values assigned to parameter `c`. Conversion from data to colors requires two steps:

- convert data values to values in the interval $[0, 1]$,
- map the interval $[0, 1]$ to a list of colors.

Default behavior for the first step is to map the maximal data value to 1, the minimal value to 0, and all values in between in a linear manner. This normalization process can be customized by creating a `Ǹormalize`[310] object and passing it as parameter `norm` to `scatter` and similar methods.

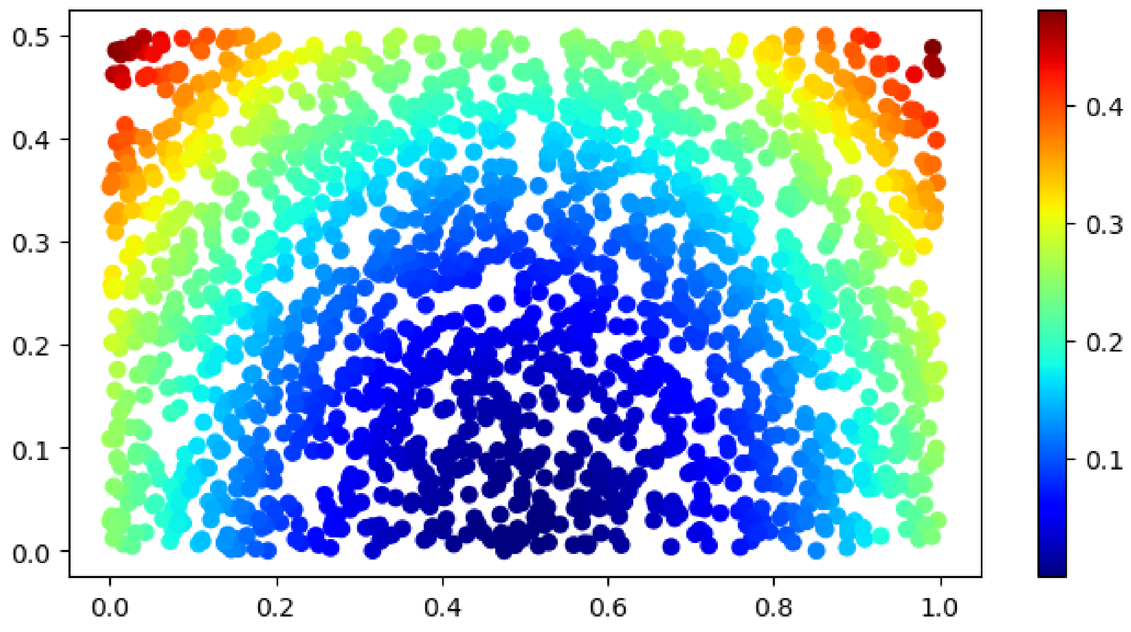
Mapping the interval $[0, 1]$ to colors is done via colormaps. There are many pre-defined colormaps, which can be passed as string to the `cmap` parameter (list of pre-defined color maps[311]). But custom colormaps can be created, too.

Colorbars

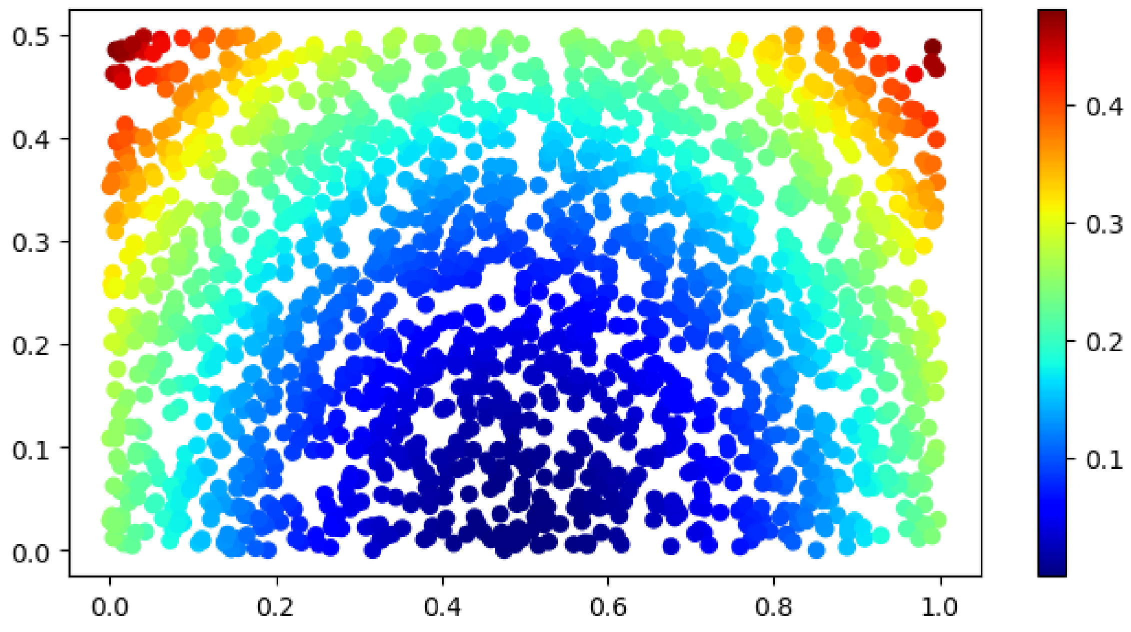
Colorbars visualize the connection between data values and colors. They can be created with `pyplot.colorbar`[312] or `Figure.colorbar`[313].

```
fig, ax = plt.subplots(figsize=(8, 4))

plt.scatter(x, y, c=z, cmap='jet')
plt.colorbar()

plt.show()
```

[309] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.scatter.html
[310] https://matplotlib.org/stable/api/_as_gen/matplotlib.colors.Normalize.html
[311] https://matplotlib.org/stable/gallery/color/colormap_reference.html
[312] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.colorbar.html
[313] https://matplotlib.org/stable/api/figure_api.html#matplotlib.figure.Figure.colorbar



```
fig, ax = plt.subplots(figsize=(8, 4))

scatter_plot = ax.scatter(x, y, c=z, cmap='jet')
fig.colorbar(scatter_plot, ax=ax, orientation='vertical')

plt.show()
```

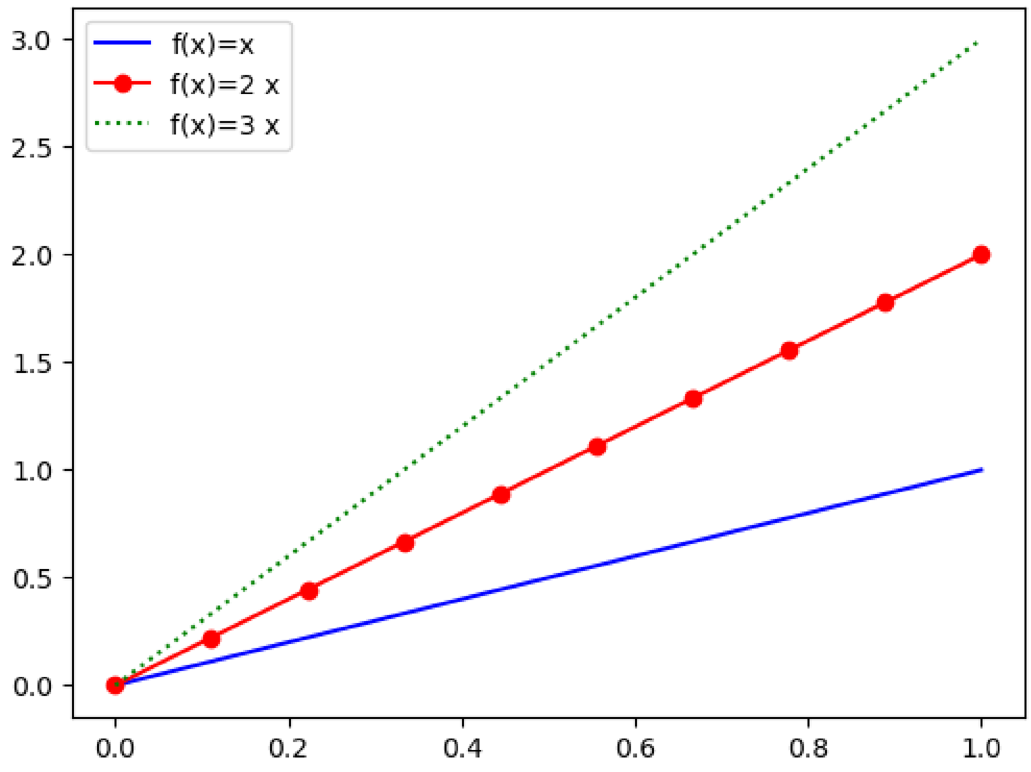


19.1.7 Legends and Text

Legends

Simple legends can be generated automatically with `Axes.legend`[314] in connection with the `label` argument to plotting methods. Each labeled line of a multi-line plot is represented in the legend and the legend is placed optimally, that is, covering as few data points as possible.

```
x = np.linspace(0, 1, 10)

fig, ax = plt.subplots()

ax.plot(x, x, '-b', label='f(x)=x')
ax.plot(x, 2 * x, '-or', label='f(x)=2 x')
ax.plot(x, 3 * x, ':g', label='f(x)=3 x')

ax.legend()

plt.show()
```



Alternatively, legend entries can be added manually.

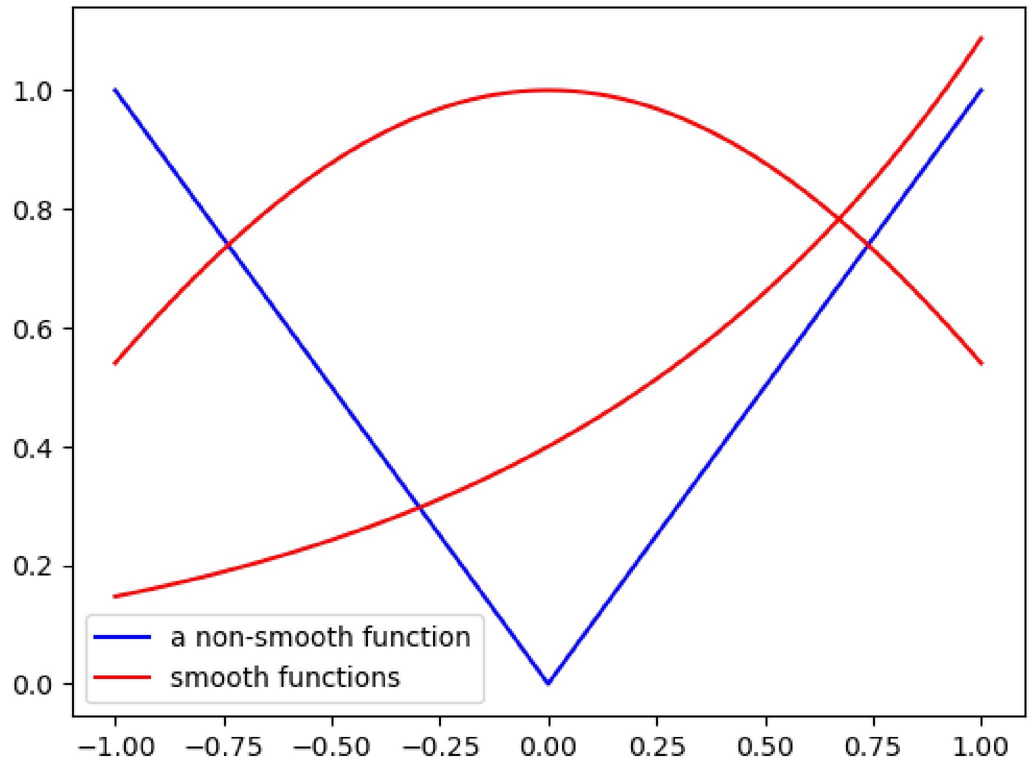
```
x = np.linspace(-1, 1, 101)

fig, ax = plt.subplots()

line1 = ax.plot(x, np.abs(x), '-b')[0]
line2 = ax.plot(x, np.cos(x), '-r')[0]
line3 = ax.plot(x, 0.4 * np.exp(x), '-r')[0]

ax.legend((line1, line2), ('a non-smooth function', 'smooth functions'))
```

(continues on next page)

[314] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.legend.html

(continued from previous page)

```
plt.show()
```



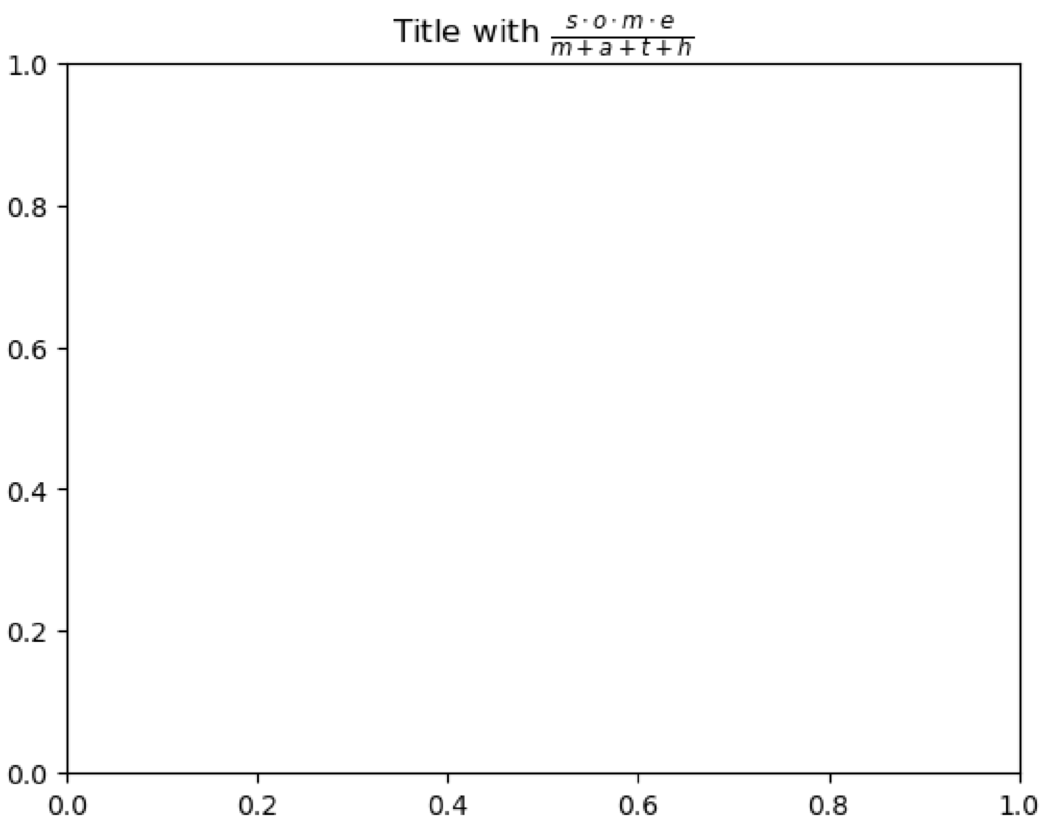
The `legend` method takes *handles* as arguments to refer to lines and other graphical objects (called *artists* in Matplotlib). In Matplotlib the objects themselve are used as handles. Because `plot` returns a list of all `Line2D` objects created, we have to extract the first (and only) element of this list.

TeX in Matplotlib Text

Mathematical formula can be used in Matplotlib via TeX[315] commands. Matplotlib comes with its own TeX interpreter and supports many but not all TeX commands. TeX commands have to be enclosed in dollar signs. Because TeX commands almost always contain backslashs and Python interprets backslashs as special characters in strings, we have to use raw strings to pass TeX commands to Matplotlib.

```
fig, ax = plt.subplots()

ax.set_title(r'Title with $\frac{s \cdot o \cdot m \cdot e}{m+a+t+h}$')

plt.show()
```

[315] https://en.wikipedia.org/wiki/TeX



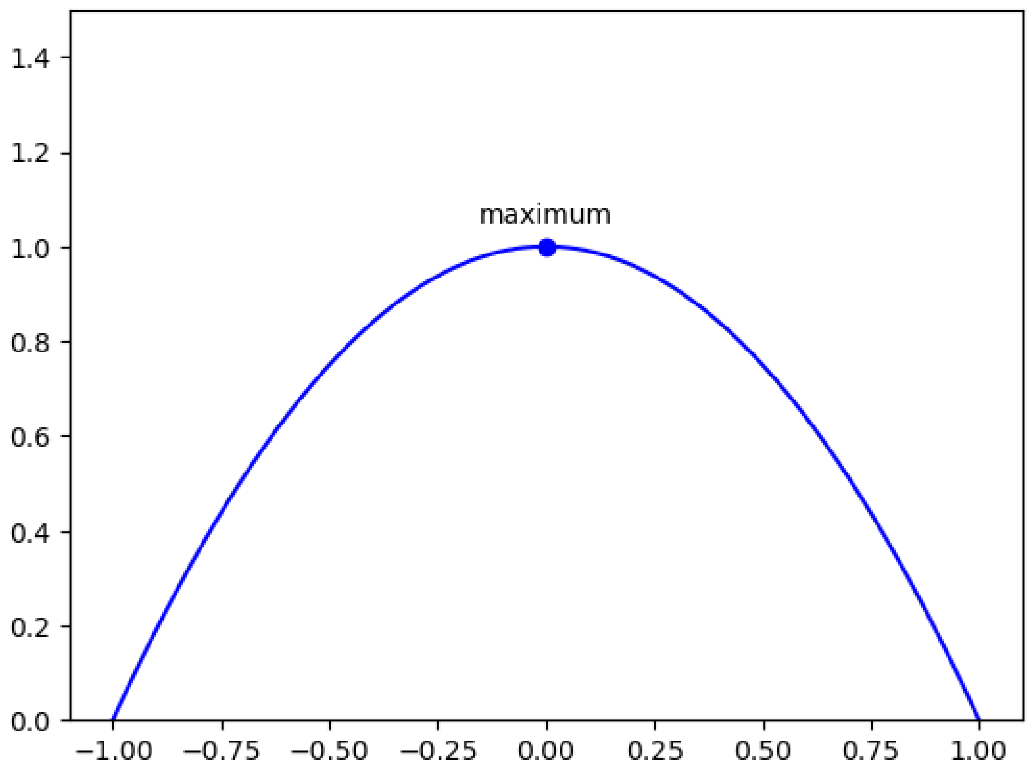
Full LaTeX[316] typesetting is available, too, but requires an external LaTeX installation.

Annotations

To add text to a plot use `Axes.text`[317] method.

```
fig, ax = plt.subplots()

x = np.linspace(-1, 1, 100)
y = 1 - x ** 2

ax.plot(x, y, '-b')
ax.plot(0, 1, 'ob')
ax.set_ylim(0, 1.5)

ax.text(0, 1.05, 'maximum', ha='center')

plt.show()
```

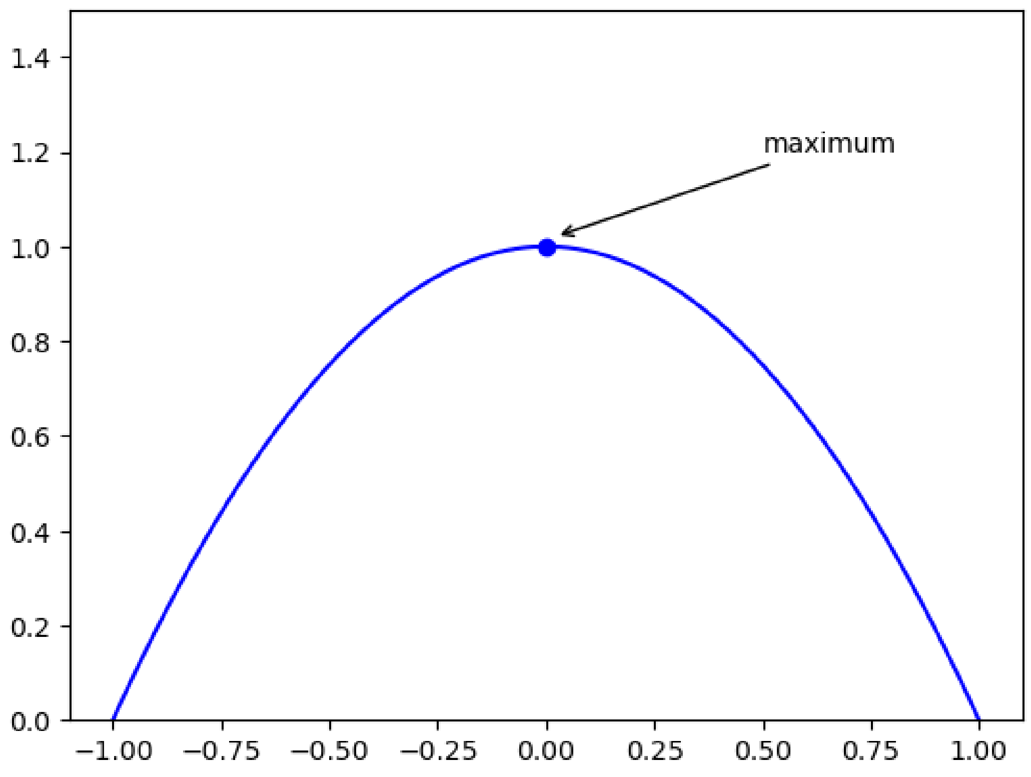
[316] https://en.wikipedia.org/wiki/LaTeX
[317] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.text.html



More advanced annotation is provided by `Axes.annotate`[318].

```
fig, ax = plt.subplots()

ax.plot(x, y, '-b')
ax.plot(0, 1, 'ob')
ax.set_ylim(0, 1.5)

ax.annotate('maximum', (0.02, 1.02), xytext=(0.5, 1.2), arrowprops={'arrowstyle':
↪'->'})

plt.show()
```

318 https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.annotate.html

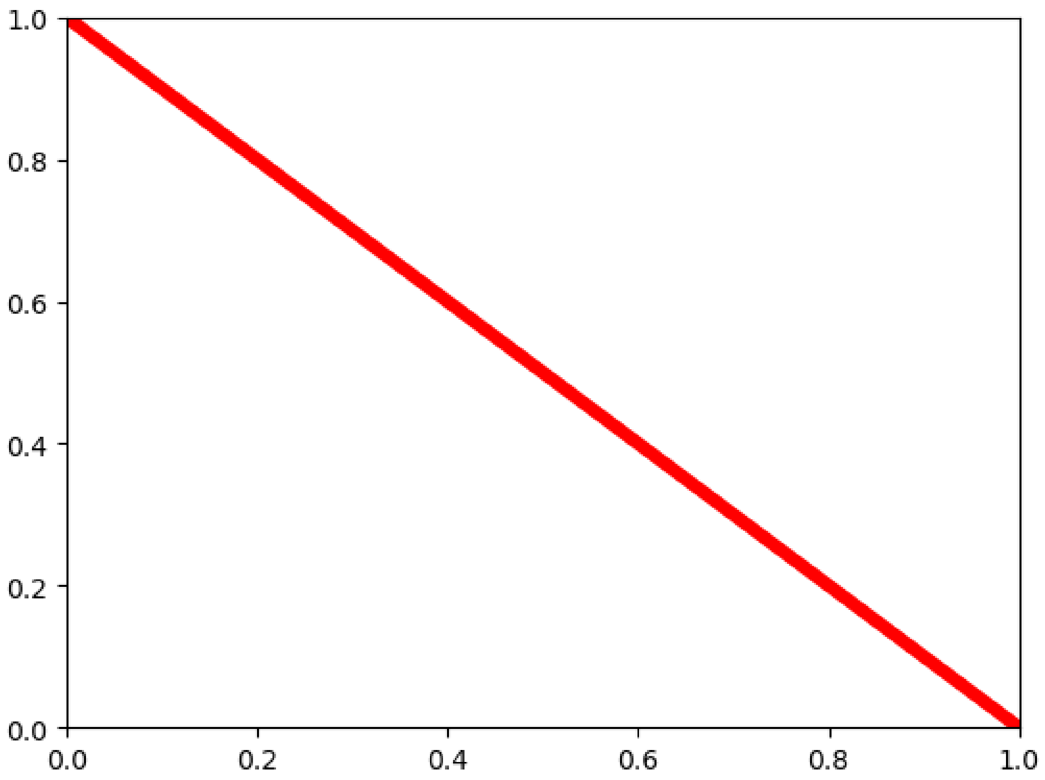


19.1.8 Geometric Objects

Matplotlib can create many types of geometric objects. Workflow is as follows: Create an object by instanciation. Then add it with `Axes.add_artist`[319] to an `Axes` object.

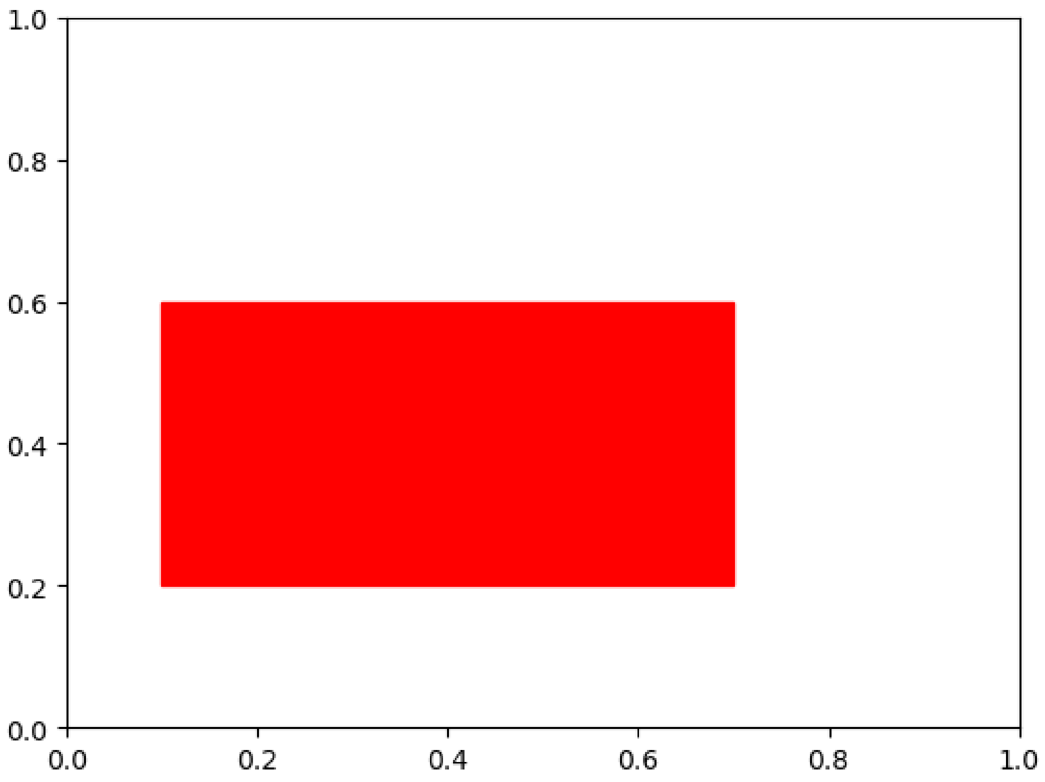
```
import matplotlib.lines

fig, ax = plt.subplots()

my_line = matplotlib.lines.Line2D([0, 1], [1, 0], color='red', linewidth=5)
ax.add_artist(my_line)

plt.show()
```

[319] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.add_artist.html



`Line2D` objects are created and returned by `Axes.plot`. They offer all the features (markers, different styles,...) known from function plotting.

```
import matplotlib.patches

fig, ax = plt.subplots()

my_rect = matplotlib.patches.Rectangle((0.1, 0.2), 0.6, 0.4, color='red')
ax.add_artist(my_rect)

plt.show()
```



Have a look at this list of classes in `matplotlib.patches`[320] or at this example[321] to get an idea of what shapes are provided next to rectangles.

19.1.9 Raster Images

Raster images can be represented in different ways:

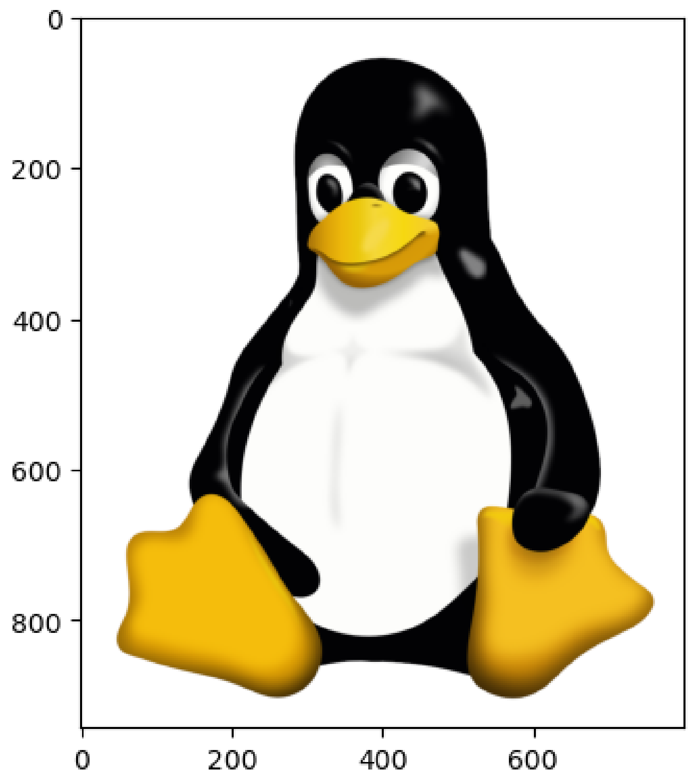
- (m, n) dimensional NumPy array (each entry is interpreted as grey level or, more generally, as argument to a colormap)
- (m, n, 3) dimensional NumPy array (the third dimension contains red, green, blue components for each pixel)
- (m, n, 4) dimensiona NumPy array (red, green, blue, opacity)

Plotting raster images is done with `Axes.imshow`[322]. To load an image from a file use `matplotlib.image.imread`[323] or `pyplot.imread`[324].

```
img = plt.imread('tux.png')
print(img.shape)

fig, ax = plt.subplots()

ax.imshow(img)

plt.show()
```

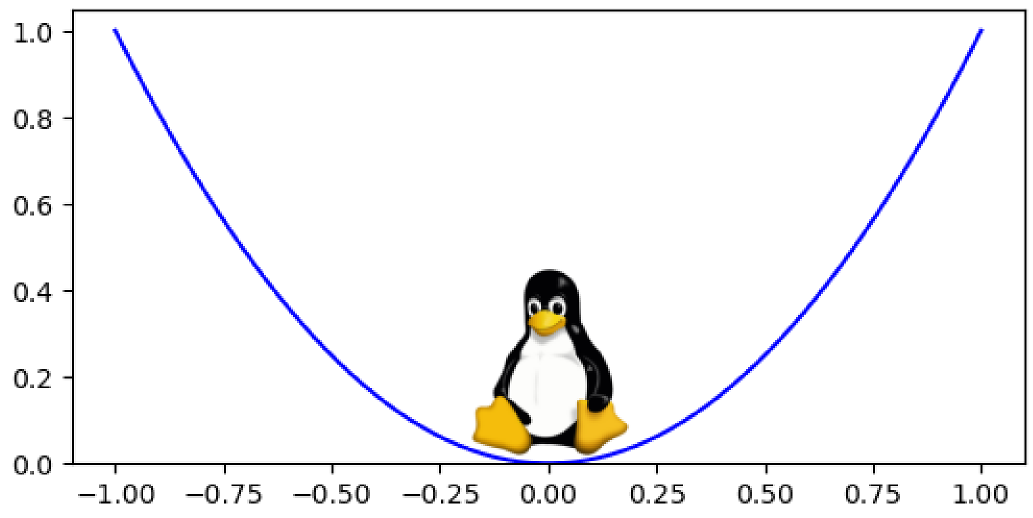
```
(943, 800, 4)
```

[320] https://matplotlib.org/stable/api/patches_api.html
[321] https://matplotlib.org/examples/shapes_and_collections/artist_reference.html
[322] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.imshow.html
[323] https://matplotlib.org/stable/api/image_api.html#matplotlib.image.imread
[324] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imread.html



The `imshow` method has several parameters. With `extent` we can accurately place an image in an existing plot.

```
x = np.linspace(-1, 1, 100)
y = x ** 2

m = img.shape[0]
n = img.shape[1]

fig, ax = plt.subplots()

ax.imshow(img, extent=(-0.2, 0.2, 0, 0.4 / n * m))
ax.plot(x, y, '-b')

plt.show()
```



Hint: For grayscale images provide `cmap`, `vmin`, `vmax` arguments to `imshow` in the same way as for scatter plots.

19.1.10 Complex Visualizations

Plot Types

Up to now we mainly used `Axes.plot` and `Axes.scatter`. But there are many more very useful plot types we do not discuss in detail. Here are some of them:

- plot[325] (line plots)
- scatter[326] (point clouds)
- bar[327] (bar plots)
- pie[328] (pie charts)
- hist[329] (histograms)
- contour[330] (contour plots)

More plot types are listed in `Axes`'s documentation[331].

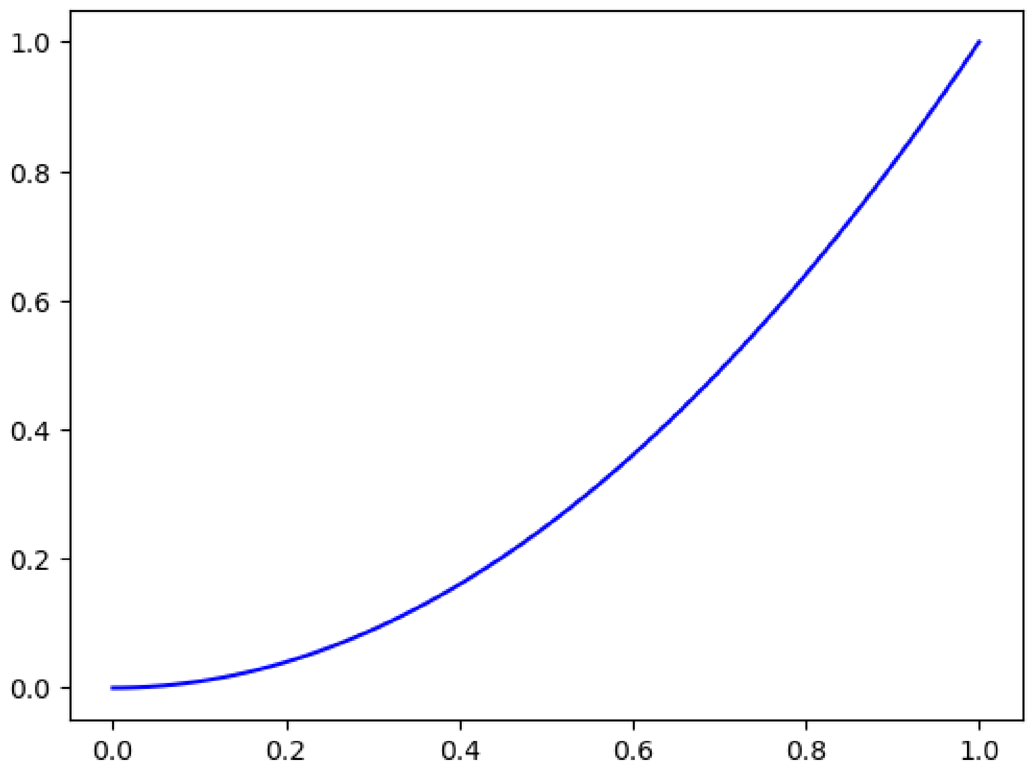
To get an idea of what is possible with Matplotlib have a look at the gallery[332].

Saving Plots and Postprocessing

Plots can be saved to image files with `Figure.savefig`[333].

```
fig_svgfile = 'saved.svg'
fig_pngfile = 'saved.png'

x = np.linspace(0, 1, 100)
y = x ** 2

fig, ax = plt.subplots()

ax.plot(x, y, '-b')

plt.show()

fig.savefig(fig_svgfile)
fig.savefig(fig_pngfile, dpi=200)
```

[325] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.plot.html
[326] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.scatter.html
[327] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.bar.html
[328] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.pie.html
[329] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.hist.html
[330] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.contour.html
[331] https://matplotlib.org/stable/api/axes_api.html#plotting
[332] https://matplotlib.org/stable/gallery/index.html
[333] https://matplotlib.org/stable/api/figure_api.html#matplotlib.figure.Figure.save_fig



After saving a figure postprocessing with external tools is possible. For raster images use, for instance, GIMP[334]. For vector graphics Inkscape[335] is a good choice. Next to addition of advanced graphical effects postprocessing also includes composing several plots to factsheets.

19.2 3D Plots

Matplotlib comes without native support for 3-dimensional plots. But there is an extension ('toolkit') for 3D plots: Mplot3d[336]. This toolkit is part of the Matplotlib standard installation.

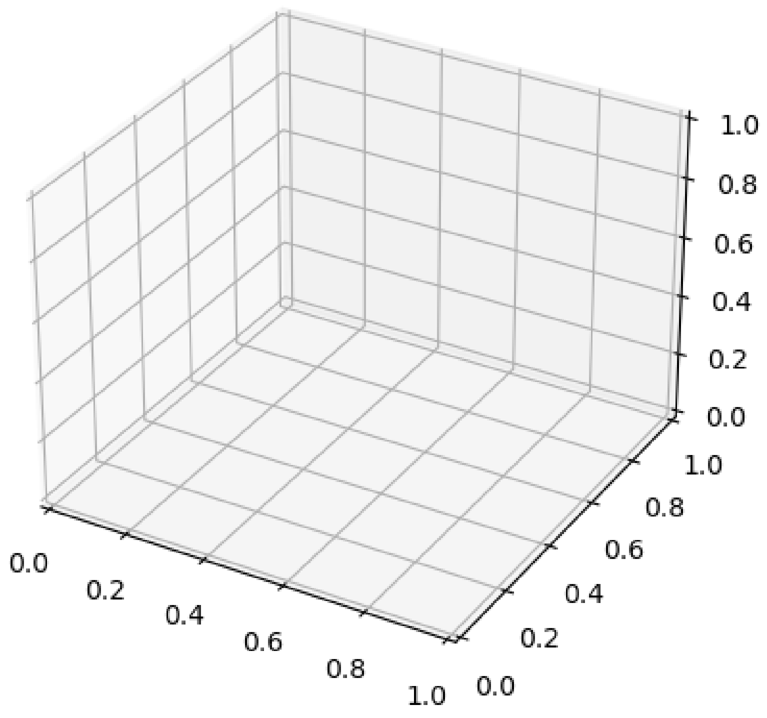
```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d as plt3d
```

19.2.1 Basics

For 3d plotting we need an `Axes3D`[337] object. This can either be created explicitly by `plt3d.axes3d.Axes3D(fig)` or implicitly by choosing `projection='3d'` in `Figure.add_axes` and similar functions. Automatic creation is preferred in newer versions of Mplot3d.

```
fig = plt.figure()
ax = fig.add_axes((0.1, 0.1, 0.8, 0.8), projection='3d')
plt.show()

print('Type of ax: ' + str(type(ax)))
```

[334] https://www.gimp.org
[335] https://inkscape.org
[336] https://matplotlib.org/stable/api/toolkits/mplot3d.html
[337] https://matplotlib.org/stable/api/toolkits/mplot3d/axes3d.html

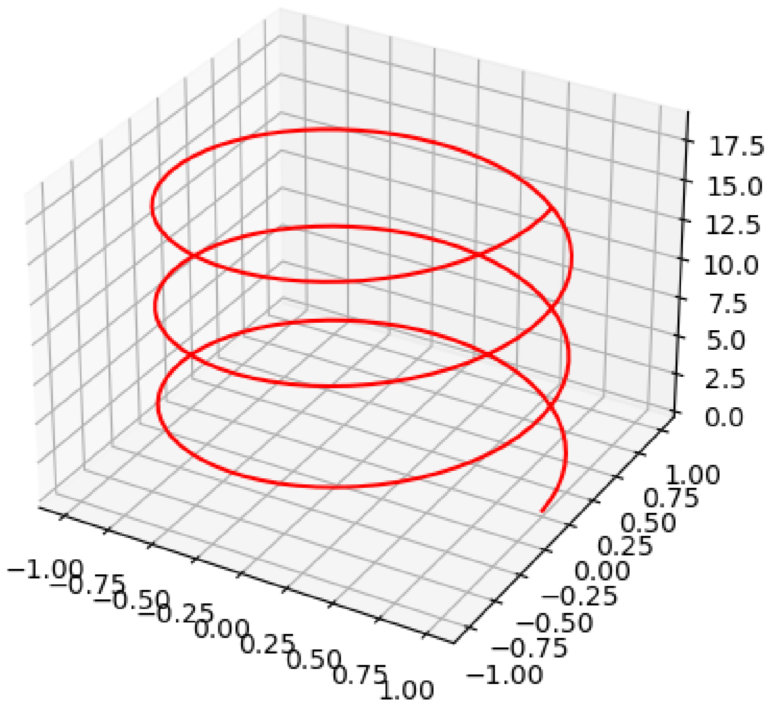


```
Type of ax: <class 'mpl_toolkits.mplot3d.axes3d.Axes3D'>
```

Now `ax` is of type `Axes3D` instead of `Axes`.

Plotting is very similar to 2d plots. Here is a 3d line plot:

```
fig = plt.figure()
ax = fig.add_axes((0.1, 0.1, 0.8, 0.8), projection='3d')

t = np.linspace(0, 6 * np.pi, 200)
x = np.cos(t)
y = np.sin(t)
z = t

ax.plot(x, y, z, '-r')
plt.show()
```



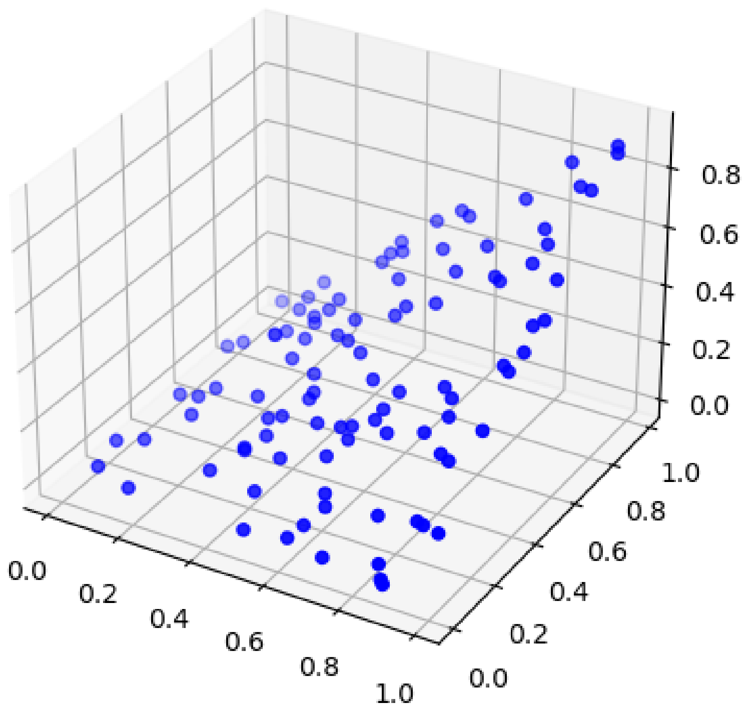
Mplot3d uses the Matplotlib API to draw its 3d plots. All arguments not processed by Mplot3d are passed on to Matplotlib. Thus, many properties like line style and color can be set in exactly the same way as in Matplotlib.

`Axes3D` objects can be configured in the same way as `Axes` objects. Functions `set_xlim`, `set_ylim`, `set_zlim`, `set_title` and axis labeling work as expected.

Several plot types are available. Some of them provide additional features in their 3d variant. For example, `scatter`[338] has a boolean argument `depthshade`, which by default is `True` and modifies the scatter points' color to give an appearance of depth. If color encodes an important feature, set `depthshade=False`.

```
fig = plt.figure()
ax = fig.add_axes((0.1, 0.1, 0.8, 0.8), projection='3d')

x = np.random.rand(100)
y = np.random.rand(100)
z = x * y

ax.scatter(x, y, z, c='blue')
plt.show()
```

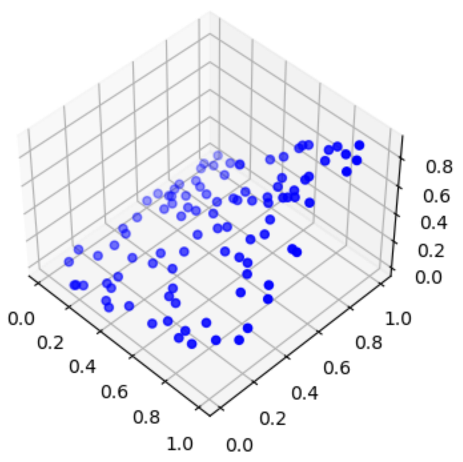
[338] https://matplotlib.org/stable/api/_as_gen/mpl_toolkits.mplot3d.axes3d.Axes3D.scatter.html

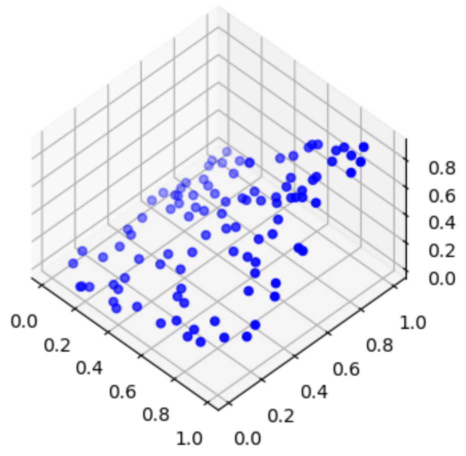


19.2.2 Camera Configuration

Initial position of the view can be set with `azim` and `elev` arguments when creating the `Axes3D` object. Simply pass them to `Figure.add_axes` or similar functions. The `proj_type` arguments switches between perspective (default) and orthogonal projection.

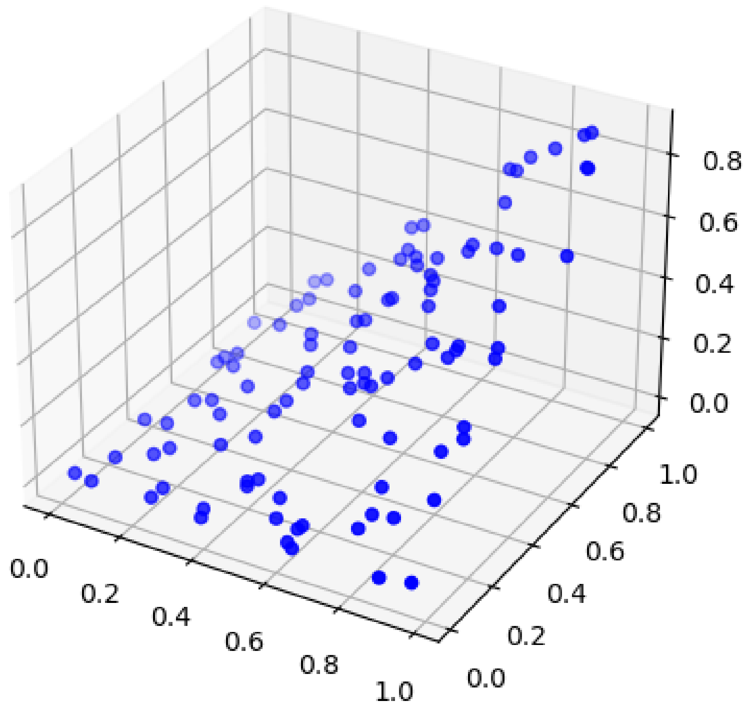
```
x = np.random.rand(100)
y = np.random.rand(100)
z = x * y

fig = plt.figure(figsize=(12, 4))
ax_left = fig.add_subplot(1, 2, 1, projection='3d', elev=45, azim=-45)
ax_right = fig.add_subplot(1, 2, 2, projection='3d', elev=45, azim=-45, proj_type=
↪'ortho')

ax_left.scatter(x, y, z, c='blue')
ax_right.scatter(x, y, z, c='blue')

plt.show()
```





When plotting in a simple Python shell, Mplot3d/Matplotlib create a figure window which allows for rotating the plot by mouse. To get this feature in a Jupyter notebook add the IPython magic `%matplotlib widget`, which requires installation of the `ipympl` Python module[339].

```
%matplotlib widget

fig = plt.figure()
ax = fig.add_axes((0.1, 0.1, 0.8, 0.8), projection='3d')

x = np.random.rand(100)
y = np.random.rand(100)
z = x * y

ax.scatter(x, y, z, c='blue')
plt.show()
```

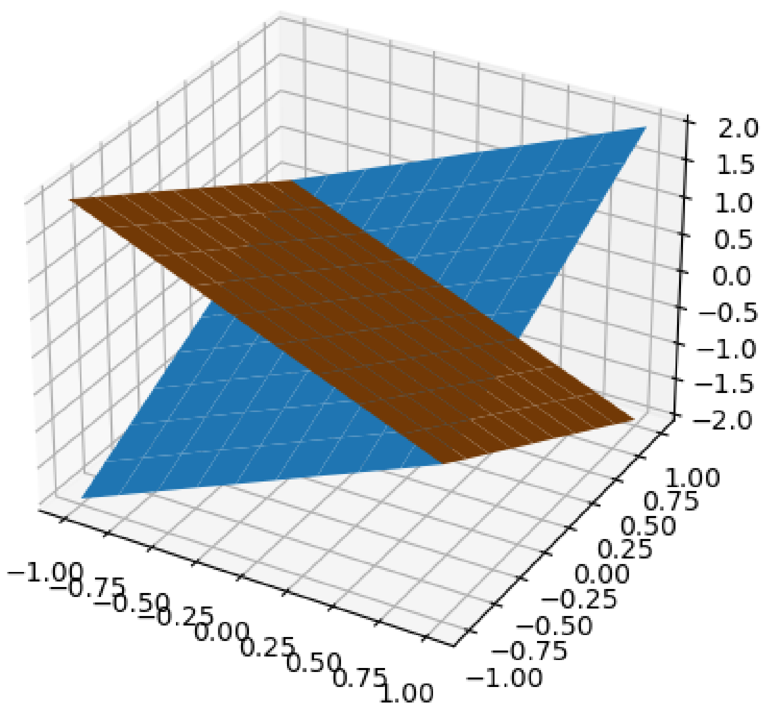


[339] https://github.com/matplotlib/ipympl

19.2.3 Limitations

Mplot3d is a very good choice for simple 3d plots. But more complex visualizations may contain rendering errors, because Mplot3d builds upon Matplotlib's 2d plotting facilities not allowing for correct overlap calculations.

```
%matplotlib widget

fig = plt.figure()
ax = fig.add_axes((0.1, 0.1, 0.8, 0.8), projection='3d')

X, Y = np.meshgrid(np.linspace(-1, 1, 10), np.linspace(-1, 1, 10))
Z1 = X + Y
Z2 = -X - Y

ax.plot_surface(X, Y, Z1)
ax.plot_surface(X, Y, Z2)

plt.show()
```



19.3 Animations

Matplotlib comes with different techniques for animated plots. Here we concentrate on `FuncAnimation`[340] objects.

```
%matplotlib widget

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as anim
```

[340] https://matplotlib.org/stable/api/_as_gen/matplotlib.animation.FuncAnimation.html

An animation is a sequence of *frames*. A frame is an image (a Matplotlib plot). Typical frame rates range from 15 to 30 frames per second (fps).

Basic steps for creating an animation are:

1. Create figure and axes.
2. Create plot.
3. Write an update function which modifies the animated objects every time step.
4. Create a `FuncAnimation` object.

Matplotlib supports different update techniques:

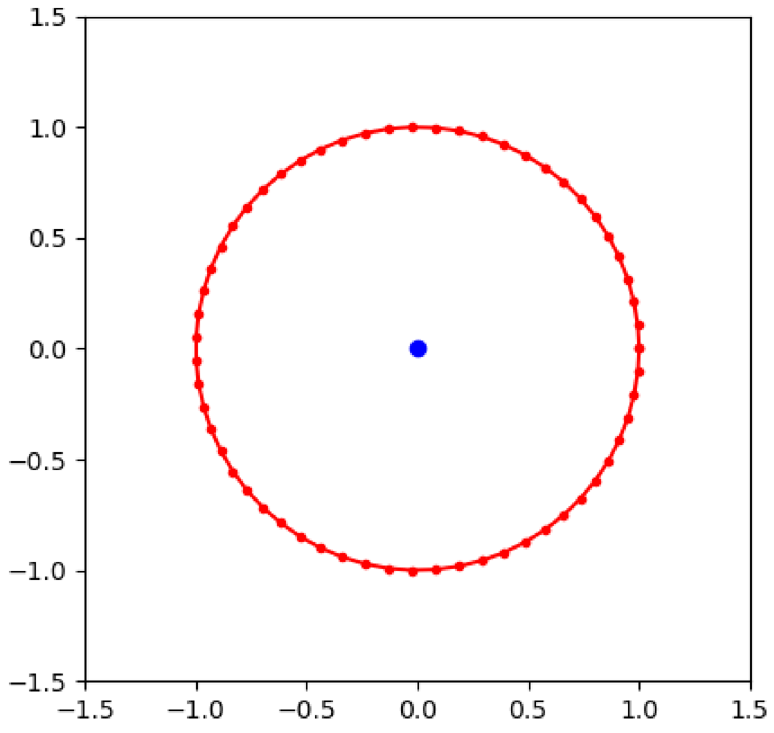
- Take the plot of the previous frame and modify it to get the next frame.
- Redraw all objects in every frame.
- Redraw all objects in every frame, but use a background image that contains all unanimated objects. This is called *blitting*.

For simple animations the first technique is a good choice. For more complex animations redrawing everything simplifies code, but may slow down the animation. Blitting speeds up complex animations, but requires some additional lines of code.

19.3.1 Animations without Blitting

For figures with only few animated objects updating only these objects is efficient and yields readable code.

```python
duration = 1000    # length of animation in milliseconds
fps = 20    # frames per second
n_frames = int(fps * duration / 1000)    # total number of frames

# figure and axes
fig, ax = plt.subplots()
ax.set_xlim(-1.5, 1.5)
ax.set_ylim(-1.5, 1.5)
ax.set_aspect('equal')

# circle (not animated)
t = np.linspace(0, 2 * np.pi, 60)
ax.plot(np.cos(t), np.sin(t), '-r.')

# line (animated)
line = ax.plot(0, 0, '-ob')[0]

# update function
def update_animation(frame):
    angle = -2 * np.pi / n_frames * frame
    line.set_data([0, np.cos(angle)], [0, np.sin(angle)])

# start animation
fa = anim.FuncAnimation(fig, update_animation, frames=n_frames, interval=1000/fps)

plt.show()
```



```
<IPython.core.display.HTML object>
```

Hint: Getting Matplotlib animations to render correctly in Jupyter Lab is sometimes more difficult than expected. Install the `ipympl` package and use the `%matplotlib widget` magic. If this doesn't work try HTML/JS export of the animation and render it's output in Jupyter Lab:

```
from IPython.display import HTML

display(HTML(fa.to_jshtml()))
```

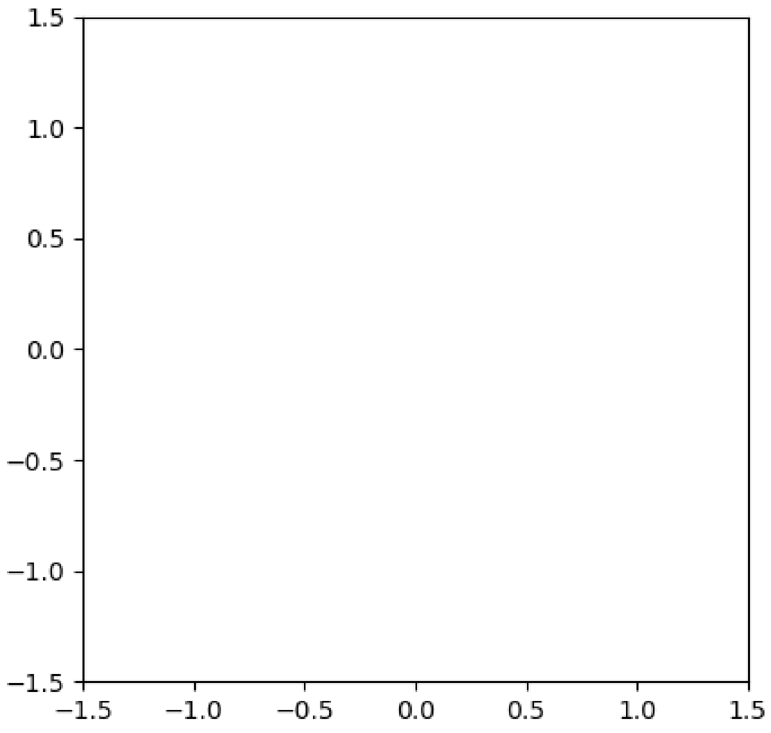
If there are many animated objects or if objects have to be created or removed during animation, then redrawing everything is a good choice to keep code readable. For easy comparison we use the same animation as above, but with complete redrawing.

```
duration = 1000    # length of animation in milliseconds
fps = 20    # frames per second
n_frames = int(fps * duration / 1000)    # total number of frames

# figure and axes
fig, ax = plt.subplots()
ax.set_xlim(-1.5, 1.5)
ax.set_ylim(-1.5, 1.5)
ax.set_aspect('equal')

# update function
def update_animation(frame):

    ax.clear()
```

(continues on next page)

(continued from previous page)

```
    t = np.linspace(0, 2 * np.pi, 60)
    ax.plot(np.cos(t), np.sin(t), '-r.')

    angle = -2 * np.pi / n_frames * frame
    ax.plot([0, np.cos(angle)], [0, np.sin(angle)], '-ob')

# start animation
fa = anim.FuncAnimation(fig, update_animation, frames=n_frames, interval=1000/fps)

plt.show()
```



```
<IPython.core.display.HTML object>
```

19.3.2 Animations with Blitting

When blitting is enabled, Matplotlib starts each frame with a fixed background image and adds the animated objects.

To get the background image, Matplotlib calls the update function once (draws the first frame) and removes the animated objects. To tell Matplotlib which objects are not part of the background the update function has to return a list of all animated objects.

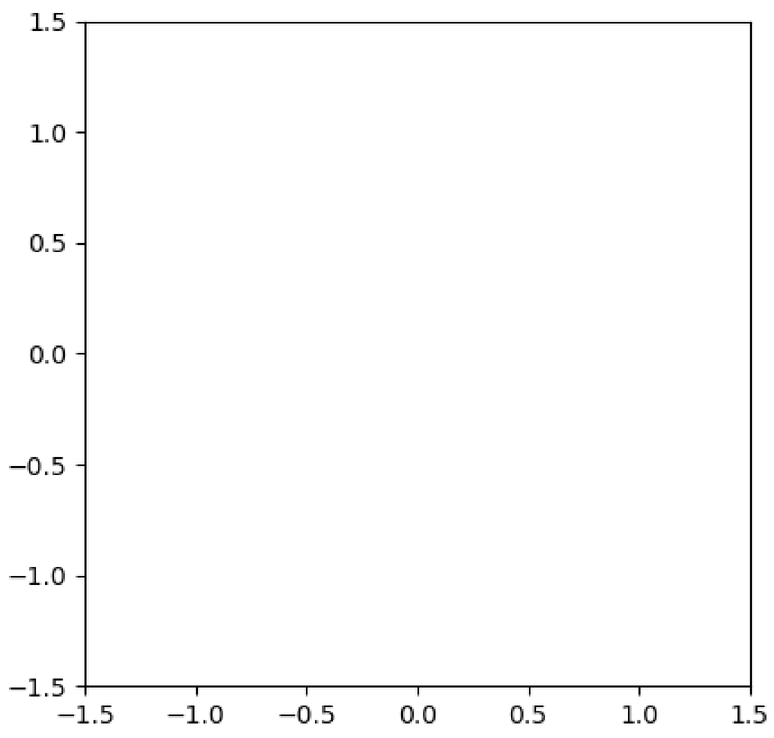
If Matplotlib shall not use the first frame to figure out the background image, an additional initialization function can be provided, which has to return a list of objects drawn but not belonging to the background. Usually that is empty.

In both cases the update function for each frame has to return a list of all animated (that is, created or modified) objects. Objects not in the list will disappear.

Important: Note that some Matplotlib backends do not support blitting and play the animation without blitting, resulting in a messed up animation. This is the case for animations in JupyterLab! To see correct results of the code below, run it in a simple Python shell.

```
duration = 1000    # length of animation in milliseconds
fps = 20    # frames per second
n_frames = int(fps * duration / 1000)    # total number of frames

# figure and axes
fig, ax = plt.subplots()
ax.set_xlim(-1.5, 1.5)
ax.set_ylim(-1.5, 1.5)
ax.set_aspect('equal')

# check blitting support
if not fig.canvas.supports_blit:
    print('Blitting not supported by backend.')

# initialization function (draws circle)
def init_animation():
    t = np.linspace(0, 2 * np.pi, 60)
    ax.plot(np.cos(t), np.sin(t), '-r.')
    return []

# update function (draws line)
def update_animation(frame):
    angle = -2 * np.pi / n_frames * frame
    line = ax.plot([0, np.cos(angle)], [0, np.sin(angle)], '-ob')[0]
    return [line]

# start animation
fa = anim.FuncAnimation(fig, update_animation, init_func=init_animation,
                        frames=n_frames, interval=1000/fps, blit=True)

plt.show()
```

```
Blitting not supported by backend.
```



```
<IPython.core.display.HTML object>
```

19.3.3 Saving animations

Matplotlib animations can be saved as animated GIFs or in different video formats.

Important: Blitting does not work for saved animations. If you intend to save an animation, don't use blitting.

```
fa.save('anim.gif', writer='imagemagick')
fa.save('anim.mp4', writer='ffmpeg')
```

19.4 Seaborn

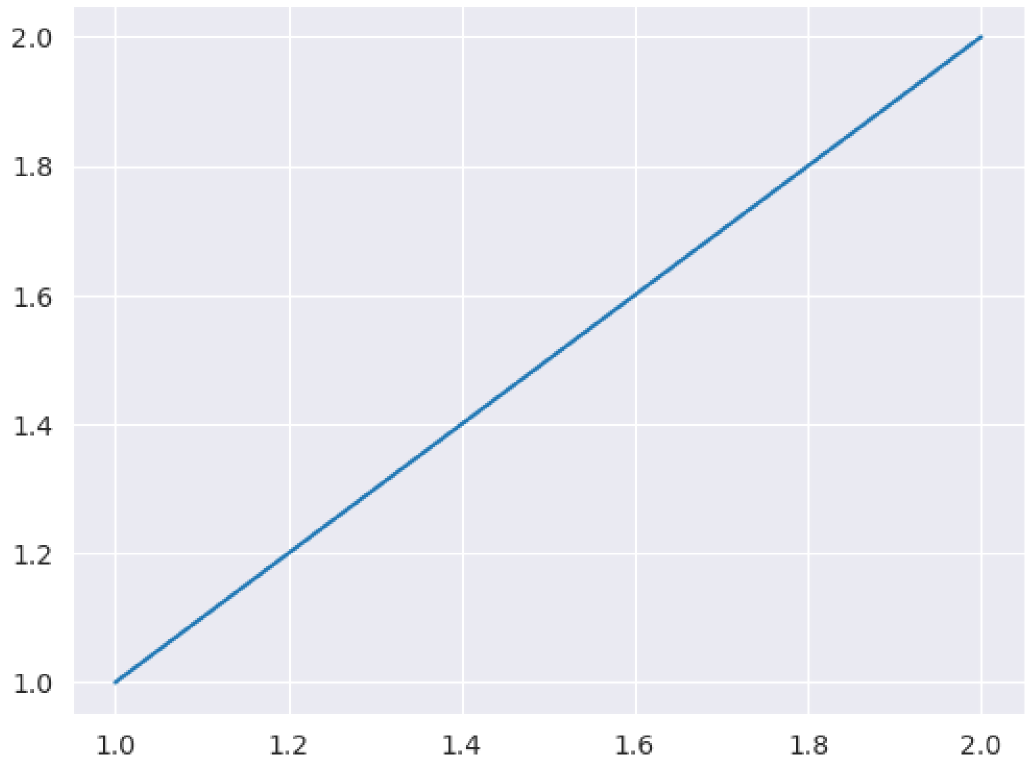
Seaborn is a Python library based on Matplotlib. It provides two useful features:

- different pre-defined styles for Matplotlib figures,
- lots of functions for visualizing complex datasets.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

The reason for importing as `sns` is somewhat vague. `sns` are the initials of Samuel Norman Seaborn[341], a fictional television character. See also issue #229 in Seaborns Github repository[342].

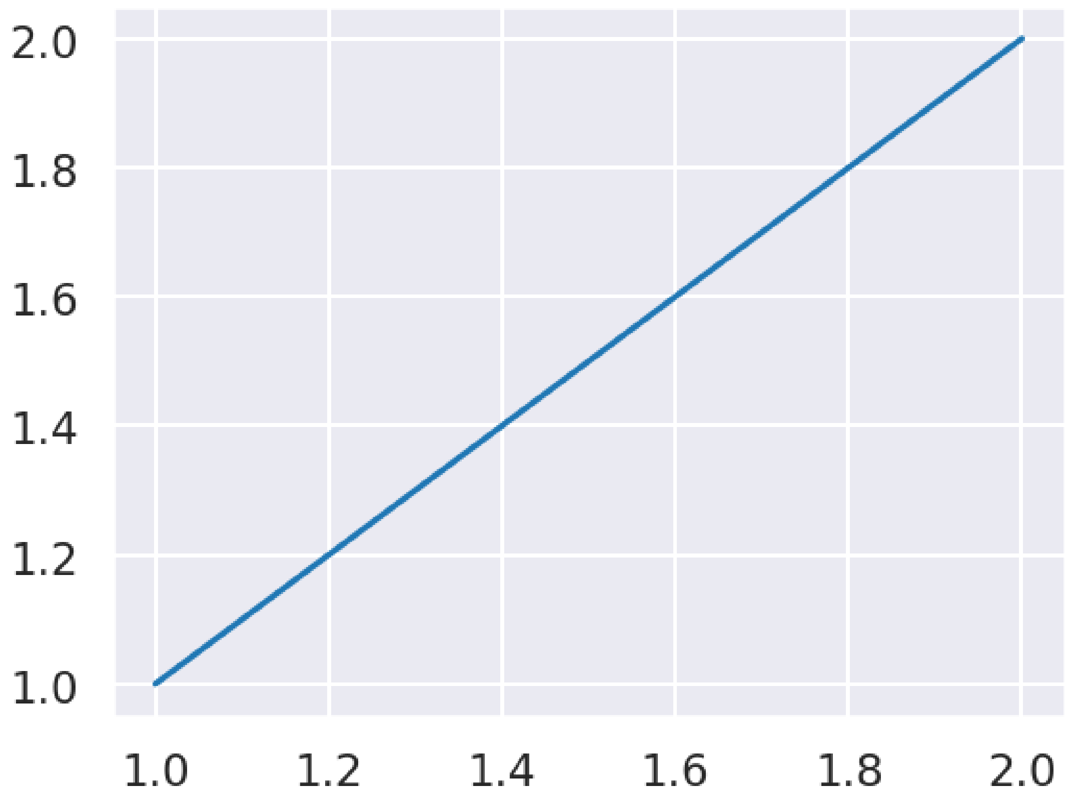
```
sns.set_style('darkgrid')
fig, ax = plt.subplots()
x = [1, 2]
y = [1, 2]
ax.plot(x, y)
plt.show()
```



Seaborn also supports different scalings for different usecases. Scaling is set with `sns.set_context` and one of the string arguments `paper`, `notebook`, `talk`, `poster`, where `notebook` is the default. Different scalings allow for almost identical code to create figures for different channels of publication.

```
sns.set_context('talk')
fig, ax = plt.subplots()
x = [1, 2]
y = [1, 2]
ax.plot(x, y)
plt.show()
```

[341] https://en.wikipedia.org/wiki/Sam_Seaborn
[342] https://github.com/mwaskom/seaborn/issues/229



19.4.1 Plots for Exploring Data Sets

Seaborn comes with lots of functions which take a whole data set (Pandas data frame) and create complex visualizations of the dataset. To get an overview have a look at the official Seaborn tutorials[343] and at the Seaborn gallery[344].

```
sns.set_context('notebook')
rng = np.random.default_rng(0)

# parameters for point clouds
means = [[5, 0, 0], [-5, 2, 0], [0, -3, 5]] # mean vectors
covs = [[[1, 1, 0], [1, 1, 0], [0, 0, 1]],
        [[10, 2, 0], [2, 10, 2], [0, 2, 10]],
        [[0.1, 0, 0], [0, 3, 3], [0, 3, 7]]] # covariance matrices
names = ['cloud A', 'cloud B', 'cloud C'] # names
ns = [100, 1000, 100] # samples per cloud

# create data frame with named samples from each cloud
clouds = []
for (mean, cov, name, n) in zip(means, covs, names, ns):
    x, y, z = rng.multivariate_normal(mean, cov, n).T
    cloud_data = pd.DataFrame(np.asarray([x, y, z]).T, columns=['x', 'y', 'z'])
    cloud_data['name'] = name
    clouds.append(cloud_data)
data = pd.concat(clouds)

# show data frame structure
display(data)

# plot pairwise relations with Seaborn
```

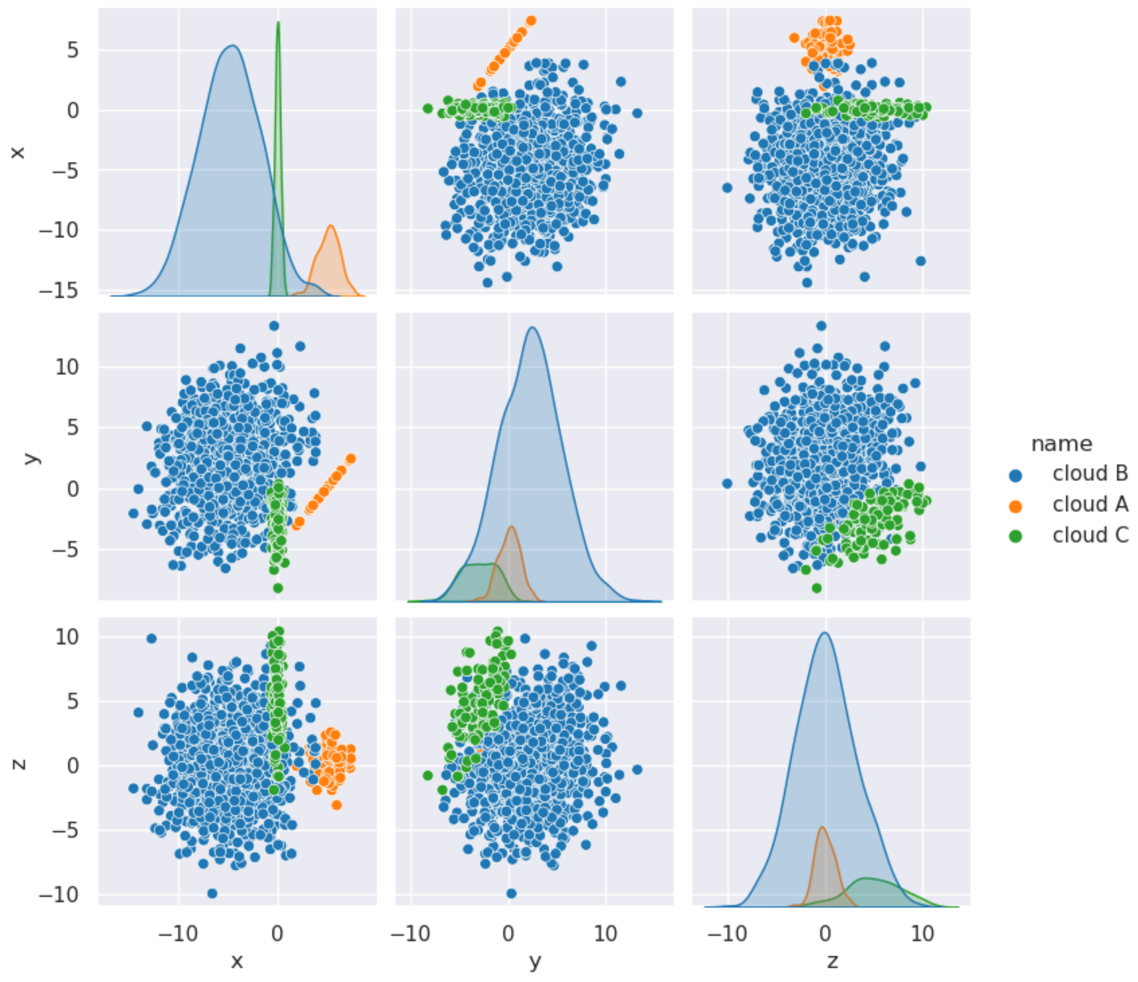
(continues on next page)

343 https://seaborn.pydata.org/tutorial.html
344 https://seaborn.pydata.org/examples/index.html

(continued from previous page)

```
sns.pairplot(data, hue='name', hue_order=['cloud B', 'cloud A', 'cloud C'])
plt.show()
```

```
            x         y         z     name
0    4.874270 -0.125730 -0.132105  cloud A
1    4.895100 -0.104900 -0.535669  cloud A
2    3.696000 -1.304000  0.947081  cloud A
3    6.265421  1.265421 -0.623274  cloud A
4    7.325031  2.325031 -0.218792  cloud A
..        ...       ...       ...      ...
95   0.044085 -4.658415  6.014424  cloud C
96   0.133556  0.038378  9.619508  cloud C
97   0.401736 -4.324301  3.687626  cloud C
98  -0.202251 -1.378106  3.291303  cloud C
99   0.110040 -2.580220  4.072073  cloud C

[1200 rows x 4 columns]
```



19.5 Maps

There exist several Python modules to plot maps with Matplotlib. Here we cover Cartopy[345] and Mplleaflet[346]. Cartopy is good for schematic maps and supports different coordinate and mapping systems. Mplleaflet combines Matplotlib drawings with interactive maps based on OpenStreetMap[347] data.

In a separate chapter we'll meet Folium[348] allowing for more advanced interactive maps.

19.5.1 Cartopy Basics

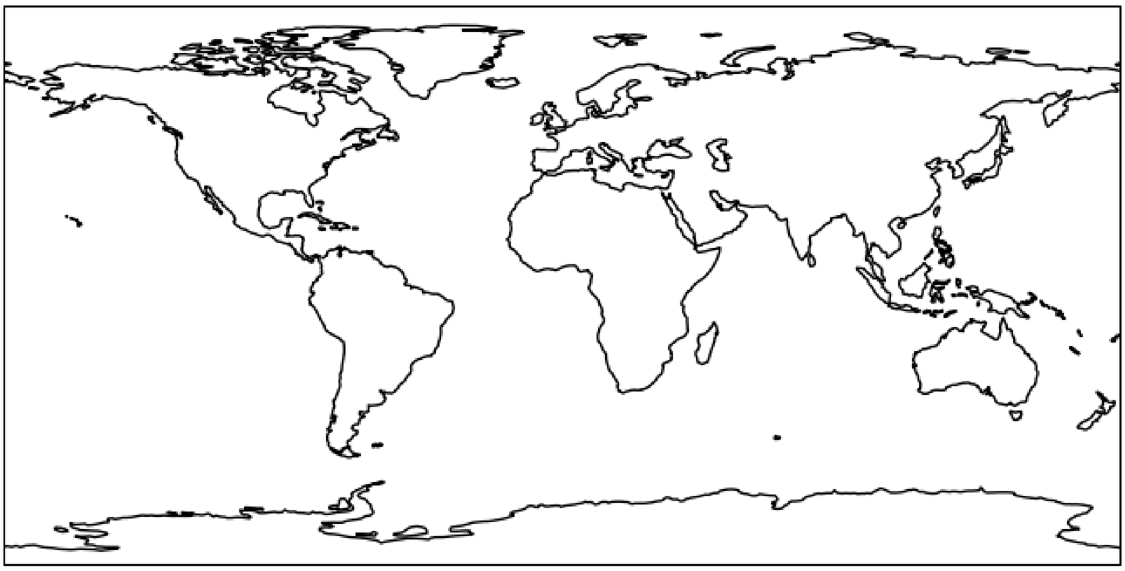
```
import matplotlib.pyplot as plt
import cartopy.crs as ccrs
import cartopy.feature as cfeature
```

The letters `crs` are the abbreviation of *coordinate reference system*. Cartopys `crs` module handles all coordinate transforms. In particular, it tells Matplotlib how to transform geographical coordinates to screen coordinates.

Creating a map requires only few steps:

- create a Matplotlib figure,
- create a `GeoAxes`[349] object,
- add content to the map.

```
fig = plt.figure()
ax = fig.add_axes((0, 0, 1, 1), projection=ccrs.PlateCarree())

ax.coastlines()

plt.show()
```



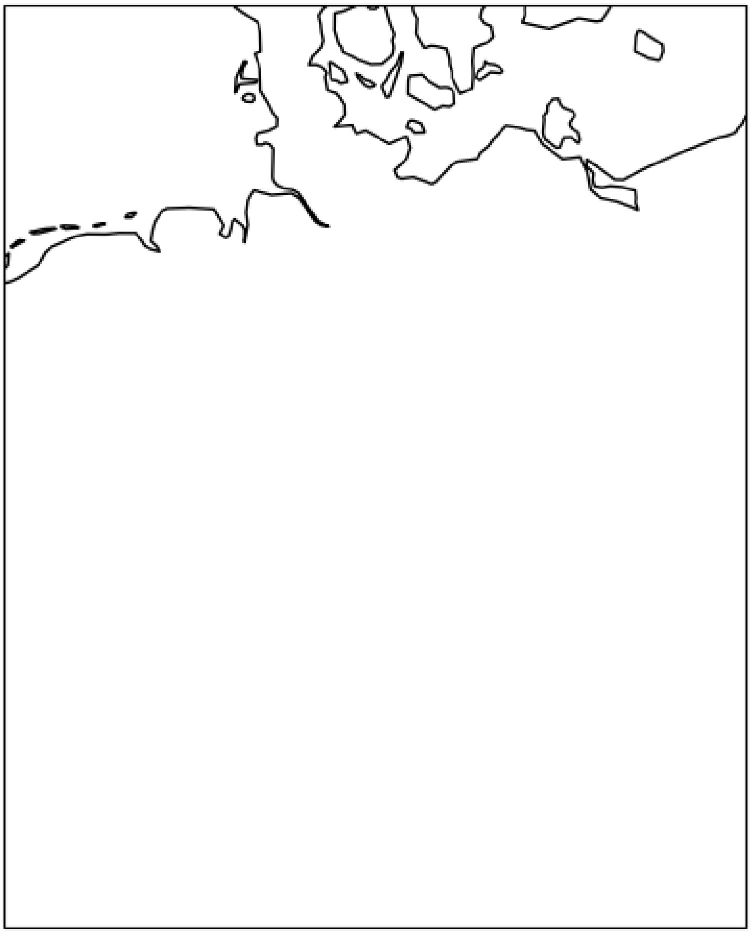
`ax` is a Cartopy `GeoAxes` object, because the `projection` is a Cartopy projection. There are many different projection types available, see Cartopy projection list[350]. Note that `GeoAxes` is derived from the usual Matplotlib

[345] https://scitools.org.uk/cartopy/docs/latest/
[346] https://github.com/jwass/mplleaflet
[347] https://www.openstreetmap.org
[348] https://python-visualization.github.io/folium/
[349] https://scitools.org.uk/cartopy/docs/latest/reference/generated/cartopy.mpl.geoaxes.GeoAxes.html
[350] https://scitools.org.uk/cartopy/docs/v0.15/crs/projections.html

Axes and thus provides very similar functionality.

Most projection types take arguments for specifing map center and some parameters. To get a map of a smaller geographic region use GeoAxes.set_extent[351]. Detail level of coastlines can be controlled with resolution argument, which has to be 110m (default), 50m or 10m. Here is a map of Germany:

```
fig = plt.figure()
ax = fig.add_axes((0, 0, 1, 1), projection=ccrs.Orthographic(10.5, 51.25))
ax.set_extent([5.5, 15.5, 47, 55.5])

ax.coastlines(resolution='50m')

plt.show()
```



19.5.2 Adding more Content

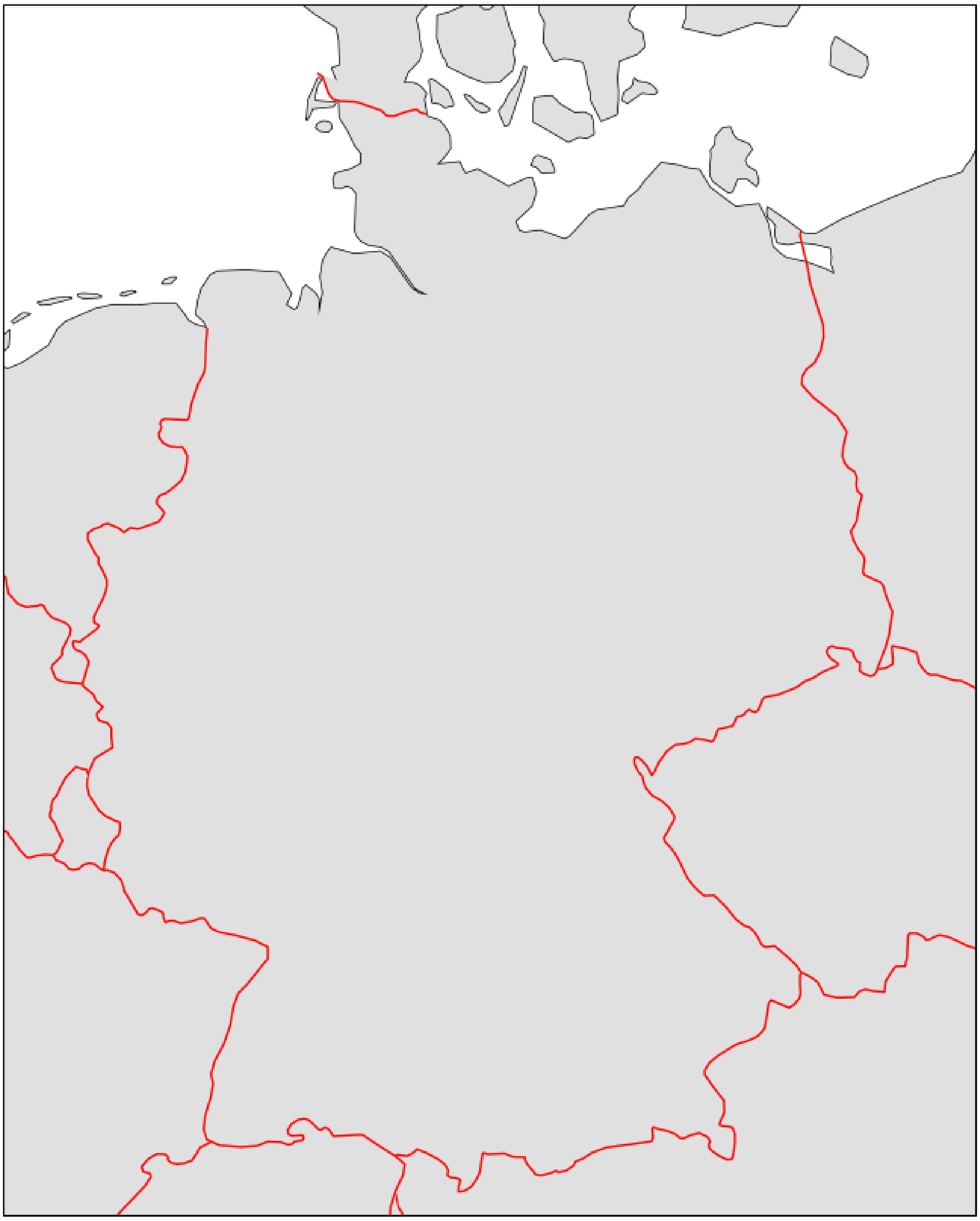
Cartopy gets its data from Natural Earth[352], which provides public domain map data. But other sources can be used, too. Map data is downloaded by Cartopy as needed and then reused if needed again. To add Natural Earth content to a map call GeoAxes.add_feature[353].

[351] https://scitools.org.uk/cartopy/docs/latest/reference/generated/cartopy.mpl.geoaxes.GeoAxes.html#cartopy.mpl.geoaxes.GeoAxes.set_extent

[352] https://www.naturalearthdata.com

[353] https://scitools.org.uk/cartopy/docs/latest/reference/generated/cartopy.mpl.geoaxes.GeoAxes.html#cartopy.mpl.geoaxes.GeoAxes.add_feature

```
fig = plt.figure(figsize=(8,8))
ax = fig.add_axes((0, 0, 1, 1), projection=ccrs.Orthographic(10.5, 51.25))
ax.set_extent([5.5, 15.5, 47, 55.5])

ax.coastlines(resolution='50m')

ax.add_feature(
    cfeature.NaturalEarthFeature(
        name='land',
        scale='50m',
        category='physical'
    ),
    facecolor='#e0e0e0'
)
ax.add_feature(
    cfeature.NaturalEarthFeature(
        name='admin_0_boundary_lines_land',
        scale='50m',
        category='cultural'
    ),
    edgecolor='r',
    facecolor='#e0e0e0'
)

plt.show()
```



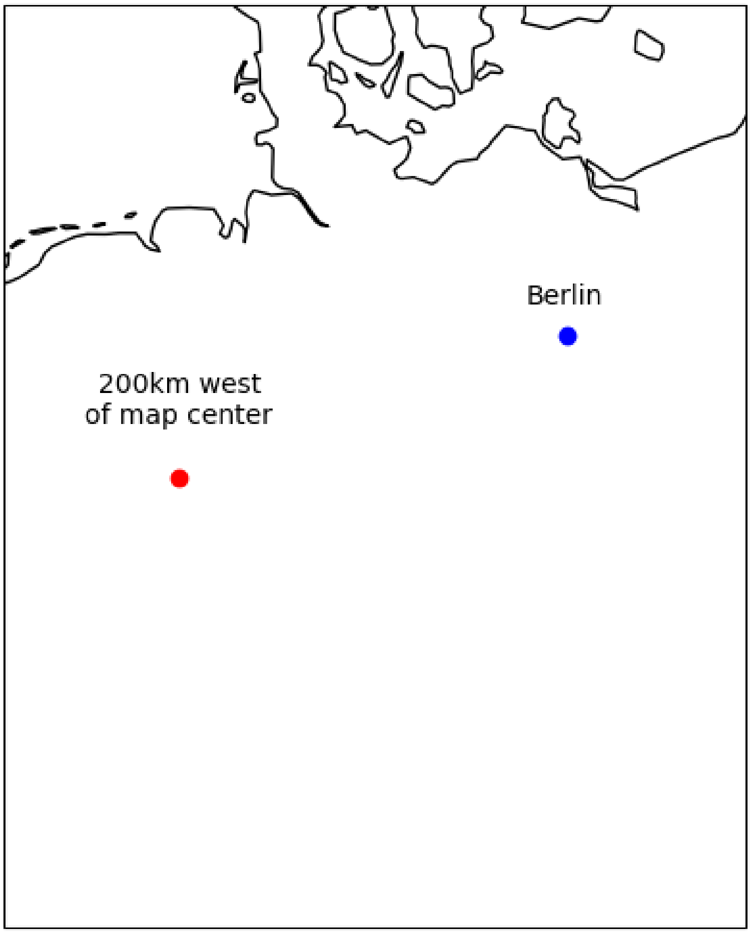
Available options for the `name` argument can be guessed from the download section of Natural Earth[354]. Look at the filenames in the download links. Arguments other than `name`, `resolution` and `category` are passed on to Matplotlib.

[354] https://www.naturalearthdata.com/downloads

19.5.3 Plotting onto the Map

All Matplotlib plotting functions are available for extending the map. Coordinates can be provided with respect to arbitrary cartographic projections, if the correct transform is chosen. If latitude, longitude values are used, `PlateCarree` is the right choice. For all other projection types coordinates have to be provided in meters.

```
fig = plt.figure()
ax = fig.add_axes((0, 0, 1, 1), projection=ccrs.Orthographic(10.5, 51.25))
ax.set_extent([5.5, 15.5, 47, 55.5])

ax.coastlines(resolution='50m')

ax.plot(13.404, 52.518, 'ob', transform=ccrs.PlateCarree())
ax.text(13.404, 52.9, 'Berlin', ha='center', va='center', transform=ccrs.
 ↪PlateCarree())

ax.plot(-200000, 0, 'or', transform=ccrs.Orthographic(10.5, 51.25))
ax.text(-200000, 80000, '200km west\nof map center', ha='center', va='center',
        transform=ccrs.Orthographic(10.5, 51.25))

plt.show()
```



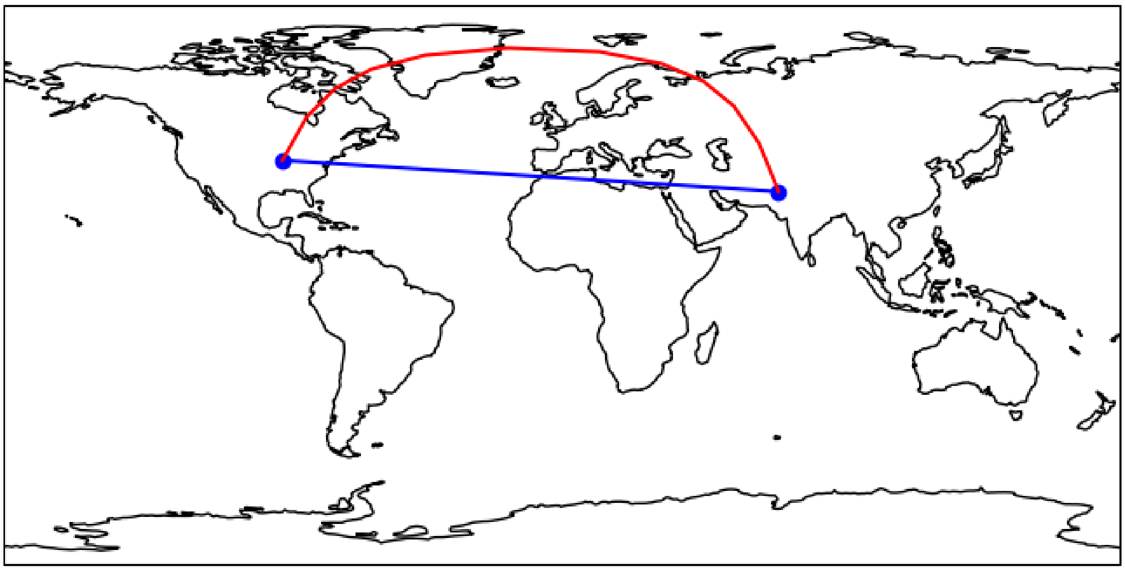
Take care: On Cartopy maps `plot` does not connect two points by a straight line, but by the shortest path with respect to the chosen coordinate system.

```
fig = plt.figure()
ax = fig.add_axes((0, 0, 1, 1), projection=ccrs.PlateCarree())
```

(continues on next page)

(continued from previous page)

```
ax.coastlines()

lat1 = 40
lon1 = -90
lat2 = 30
lon2 = 70

ax.plot([lon1, lon2], [lat1, lat2], '-ob', transform=ccrs.PlateCarree())
ax.plot([lon1, lon2], [lat1, lat2], '-r', transform=ccrs.Geodetic())

ax.set_global()

plt.show()
```



When plotting Matplotlib adjusts axes limits to the plotted objects. With `GeoAxes.set_global`[355] we reset limits to show the whole map.

19.5.4 Interactive Maps with Mplleaflet

```
import matplotlib.pyplot as plt
import mplleaflet
```

The Python library Mplleaflet allows to show Matplotlib drawings on an interactive map based on OpenStreetMap[356] or other map services. The result is a webpage, which can be embedded into an Jupyter notebook.

Use longitude and latitude values for plotting and then call `mplleaflet.display` to show the map inside the Jupyter notebook. Alternatively, call `mplleaflet.show` to open the map in new window.

Hint: Mplleaflet seems to be unmaintained for several years, resulting in broken compatibility with Matplotlib (Matplotlib changed some variable names). Thus, usage of Mplleaflet may require some tweaking of Matplotlib variable names. See GitHub issue[357] and this solution[358].

[355] https://scitools.org.uk/cartopy/docs/latest/reference/generated/cartopy.mpl.geoaxes.GeoAxes.html#cartopy.mpl.geoaxes.GeoAxes.set_global
[356] https://www.osm.org
[357] https://github.com/jwass/mplleaflet/issues/80
[358] https://github.com/plotly/plotly.py/issues/2913#issuecomment-730619757

```
fig = plt.figure(figsize=(10,6))
ax = fig.add_axes((0, 0, 1, 1))

ax.plot([12, 13, 13, 12, 12], [51, 51, 50, 50, 51], '-ob')

# some Mplleaflet patching
ax.xaxis._gridOnMajor = ax.xaxis._major_tick_kw['gridOn']
ax.yaxis._gridOnMajor = ax.yaxis._major_tick_kw['gridOn']

mplleaflet.display(fig=fig)
```

```
/home/jef19jdw/anaconda3/envs/ds_book/lib/python3.10/site-packages/IPython/core/
↪display.py:431: UserWarning: Consider using IPython.display.IFrame instead
  warnings.warn("Consider using IPython.display.IFrame instead")
```

```
<IPython.core.display.HTML object>
```

CHAPTER

TWENTY

PLOTLY

Plotly[359] is a relatively new JavaScript based plotting library for Python and several other languages[360]. It's free and open source, but developed by Plotly Technology Inc providing commercial tools and services around Plotly. Plotly started as an online-only service (rendering done on a server), but now may be used offline without any data transfer to plotly.com[361].

Here we only cover 3d plots to have a good alternative to Matplotlib's limited 3d capabilities. But plotly allows for 2d plotting, too. A major advantage of Plotly is, that interactive Plotly plots can easily be integrated into websites.

The Plotly python package may be installed via `conda` from Plotly's channel:

```
conda install -c plotly plotly
```

Plotly integrates well with NumPy and Pandas.

```python
import numpy as np
import pandas as pd
```

20.1 Plotly Express

The Plotly express[362] interface to Plotly provides commands for complex standard tasks:

```python
import plotly.express as px

a = np.random.rand(100)
b = np.random.rand(100)
c = a * b
df = pd.DataFrame({'some_feature': a, 'another_feature': b, 'result': c})

fig = px.scatter_3d(df, x='some_feature', y='another_feature', z='result', color=
↪'result', size='result', width=800, height=600)

fig.show()
```

```
<IPython.core.display.HTML object>
```

[359] https://plotly.com/python/
[360] https://plotly.com/graphing-libraries/
[361] http://plotly.com
[362] https://plotly.com/python/plotly-express

20.2 Graph Object Interface

For more advanced usage Plotly provides the graph object interface[363]. Here we first have to create a Figure[364] object, which then is filled with content step by step.

```
import plotly.graph_objects as go

fig = go.Figure()
fig.layout.width = 800
fig.layout.height = 600

x, y = np.meshgrid(np.linspace(0, 1, 30), np.linspace(0, 1, 30))
z = x * y

fig.add_trace(
    go.Surface(x=x, y=y, z=z, colorscale='Jet')
)

u = x.reshape(-1)
v = y.reshape(-1)
w = z.reshape(-1) + 0.3 * (np.random.rand(z.size) - 0.5)

fig.add_trace(
    go.Scatter3d(
        x=u, y=v, z=w,
        marker={'size': 2, 'color': 'rgba(255,0,0,0.5)'},
        line={'width': 0, 'color': 'rgba(0,0,0,0)'},
        hoverinfo = 'none'
    )
)

fig.show()
```

```
<IPython.core.display.HTML object>
```

20.3 Exporting Figures

Plotly plots can easily be exported to HTML with Figure.write_html[365]:

```
fig.write_html('fig.html', include_plotlyjs='directory', auto_open=False)
```

The `include_plotlyjs='directory'` tells Plotly that the exported HTML file finds the Plotly JavaScript Library in the HTML file's directory (for offline use). To make this work download Plotly's JS bundle[366] from Plotly's GitHub repository.

Defautl behavior is that the exported file is opened automatically after export. To prevent this, use `auto_open=False`.

[363] https://plotly.com/python-api-reference/plotly.graph_objects.html
[364] https://plotly.com/python-api-reference/generated/plotly.graph_objects.Figure.html
[365] https://plotly.com/python-api-reference/generated/plotly.graph_objects.Figure.html#plotly.graph_objects.Figure.write_html
[366] https://github.com/plotly/plotly.js/blob/master/dist/plotly.min.js

CHAPTER

TWENTYONE

FOLIUM

Folium[367] is a Python package for generating interactive maps. It's based on OpenStreetMap[368] and the Leaflet[369] JavaScript library.

```
import folium
```

Related projects:

- *Public Transport* (page 949)
  - *Interactive Map* (page 956)

21.1 Basic Usage

Usage is straight-forward.

```
m = folium.Map(location=[50.7161, 12.4956], zoom_start=16)

folium.Marker(
    [50.7161, 12.4956],
    popup='<b>Zwickau University of Technology</b><br />Kornmark 1',
    tooltip='Zwickau University of Technology',
    icon=folium.Icon(color="red", icon="info-sign")
).add_to(m)

display(m)
```

```
<folium.folium.Map at 0x7f203ef4fa90>
```

For more features have a look at Folium's Quickstart Guide[370].

[367] https://python-visualization.github.io/folium/
[368] https://www.osm.org
[369] https://leafletjs.com/
[370] https://python-visualization.github.io/folium/quickstart.html

21.2 Advanced Features

Of particular interest for data science purposes are:

- image overlays[371] (position raster image on a map)
- marker clusters[372] (place many markers on a map)
- fast marker clusters[373] (position many markers on a map very efficiently)

Data on Folium maps is organized in layers. Layer visibility may be toggled by the user, if we add layer controls[374]:

```
import folium.plugins

m = folium.Map(location=[50.7161, 12.4956], zoom_start=17)

folium.plugins.MarkerCluster(
    [[50.7161, 12.4956], [50.7161, 12.4966], [50.7171, 12.4956]],
    name='some marker'
).add_to(m)

folium.plugins.MarkerCluster(
    [[50.7151, 12.4946], [50.7161, 12.4946], [50.7151, 12.4956]],
    name='more marker'
).add_to(m)

folium.LayerControl().add_to(m)

display(m)
```

```
<folium.folium.Map at 0x7f203ef4fe80>
```

21.3 Exporting a Map

To generate an HTML file containing the interactive Folium map, call `save`[375]. The `save` method is inherited by Foliums `Map` from `branca.elements.Element` as we see from `Map`'s documentation[376]. Branca[377] is used by Folium for HTML and JavaScript generation.

```
m.save('folium.html')
```

The generated HTML file may be edited to add some more (HTML and JavaScript) content. Alternatively, we may inject some HTML from Python source via 'get_root'[378]:

```
m.get_root().html.add_child(folium.Element(
'''
<div style="position: absolute; z-index: 10000; top: 10px; left: 60px;␣
↪background: white; padding:5px 10px;">
<span style="font-size: 20pt; font-weight: bold; color: black;">Some Title</span>
↪<br/>
<span style="font-size: 10pt; color: blue;">some text</span>
```

(continues on next page)

[371] https://python-visualization.github.io/folium/modules.html#folium.raster_layers.ImageOverlay
[372] https://python-visualization.github.io/folium/plugins.html#folium.plugins.MarkerCluster
[373] https://python-visualization.github.io/folium/plugins.html#folium.plugins.FastMarkerCluster
[374] https://python-visualization.github.io/folium/modules.html#folium.map.LayerControl
[375] https://python-visualization.github.io/branca/element.html
[376] https://python-visualization.github.io/folium/modules.html#folium.folium.Map
[377] https://github.com/python-visualization/branca
[378] https://python-visualization.github.io/branca/element.html#branca.element.Element.get_root

(continued from previous page)

```
</div>
'''
))

display(m)
m.save('folium.html')
```

```
<folium.folium.Map at 0x7f203ef4fe80>
```

Part VI

Supervised Learning

CHAPTER

TWENTYTWO

MACHINE LEARNING OVERVIEW

Machine learning is the core technique behind data science in general and todays artificial intelligence. Revisit *Data Science, AI, Machine Learning* (page 33) for details. There are three major types of machine learning:

- supervised learning,
- unsupervised learning,
- reinforcement learning.

Supervised learning is also known as *learning by examples*. The task is to predict properties of input data. For instance, input data could be pet images and the computer shall decide whether there is a dog or a cat in the image. Predicting values of a discrete variable is called *classification*. If we want to predict continuous variables, then it's a *regression problem*. To *train* a supervised learning method, we need lots of training examples, that is, input data with corresponding outputs. The computer then gathers knowledge from the examples and tries to generalize that knowledge to input data not contained in the training data set.

Unsupervised learning tries to find structures in large data sets without the need for examples. It's mostly concerned with *clustering*. Given a data set, find subsets (clusters) of data points with common properties. In some sense it's also about classification, but with classes not known in advance. A typical example is the identification of different customer types in e-commerce. Unsupervised learning also covers techniques for anomaly detection and generative learning (create new data with same properties like exisiting data).

Reinforcement learning is learning by *trial and error*. Thus, it's very similar to a child's learning process. Again, no examples of how to do it right are required. The *agent* (the learning algorithm) can choose between several actions. Depending on the outcome of it's actions, the agent changes its behavior until the correct outcome is observed. Examples are mobile robots finding their way through a building, but also artificial intelligences playing classical board games (see Google's AlphaGo Zero[379] for a prominent example).

There exist less prominent subclasses of machine learning, like **semi-supervised learning**, mixing techniques from supervised learning and unsupervised learning.

Today's most advanced artificial intelligence products like ChatGPT[380] or Stable Diffusion[381] use a complex mix of many different techniques. To understand such complex products we have to understand the basic methods and algorithms first. Only then we may go on designing more complex systems.

[379] https://en.wikipedia.org/wiki/AlphaGo_Zero
[380] https://en.wikipedia.org/wiki/ChatGPT
[381] https://en.wikipedia.org/wiki/Stable_Diffusion

CHAPTER

TWENTYTHREE

GENERAL CONSIDERATIONS

Before we consider concrete classes of supervised learning methods, we have a look at basic principles common to (almost) all methods.

Related projects:

23.1 Problem and Workflow

Here we introduce basic terminology used throughout the book and we present major problems arising when implementing supervised learning methods.

23.1.1 Statement of the Problem

Abstract Formulation

Let $f: X \to Y$ be a function between two sets X and Y and assume we only know f at finitely many elements of X. The aim is to find an approximation f_{approx} of f, which can be evaluated at arbitrary elements of X.

X is the set of *inputs*, Y is the set of outputs or *labels*. The finitely many elements $x_1, \ldots, x_n$ in X at which f is known are called *training inputs*. Corresponding labels $y_1 := f(x_1), \ldots, y_n := f(x_n)$ are called *training labels*. The pairs $(x_1, y_1), \ldots, (x_n, y_n)$ are the *training examples* or *training data*. Ultimately, the function f_{approx} will be represented by a Python program or a Python function: it takes an input and yields some output.

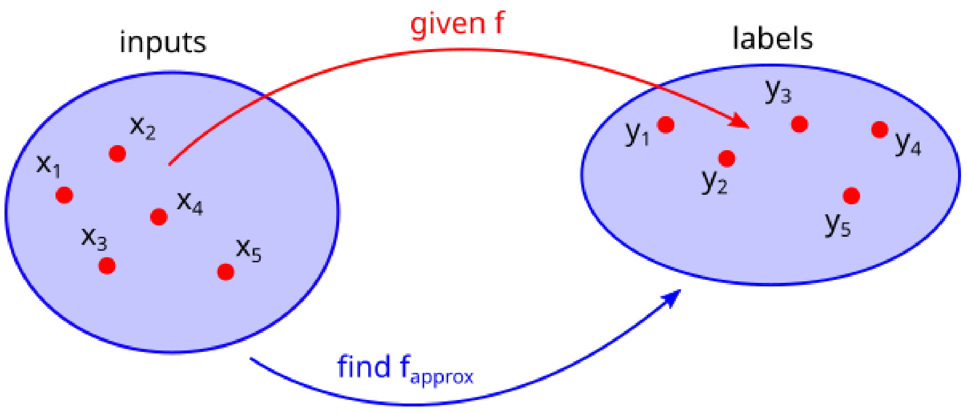


Fig. 23.1: Given the a mapping f on finitely many points supervised learning aims at finding a mapping f_{approx} on the whole space.

Standard Situation

We only consider $X = \mathbb{R}^m$, that is, m-dimensional vectors as inputs. Here, m is a positive integer. The elements of X are called *feature vectors*. Each feature vector contains values for m *features*. In addition, we restrict our attention to labels in finite or infinite subsets $Y \subseteq \mathbb{R}$.

If there are only finitely many labels, then the supervised learning problem of finding f_{approx} is a *classification problem*. With infinitely many labels it's a *regression problem*.

Example: Classification of Handwritten Digits

Recognizing handwritten digits is a typical classification task. Given an image of a handwritten digit we have to decide which digit it contains. Assuming gray scale images, where each pixel's color is described by a real number, the set of inputs has dimension $m =$ width $\times$ height. If the images are 28 by 28 pixels in size (like in QMNIST data set), then each input has 784 features. The label set is $Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

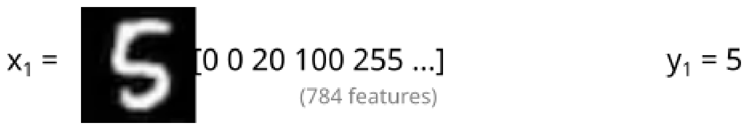


Fig. 23.2: The sample x_1 is a vector with 784 components, one component (or feature) per pixel.

Example: Predicting House Prices

Given several properties of a house we would like to predict the price we have to pay for buying that house. The properties we take into account yield the feature vectors. The output is the price, a real number. Thus, it's a regression problem. Observing house prices on the market yields training examples, which then can be used to predict prices for houses not sold during our market observation.

Hypotheses, Parameters, Models

An approximate function f_{approx} is sometimes called *hypothesis* in machine learning contexts. The set of all functions taken into consideration is the *hypotheses space*.

Usually one does not consider general functions. Instead one restricts search to some family of functions, e.g. linear functions, described by a number of parameters. If that family has p parameters, then we have a p-dimensional hypotheses space. Fitting the parameters of such *parameterdependent hypotheses* to training examples is much easier than fitting a general hypothesis.

We have to distinguish between parameters of the hypothesis and parameters of the whole approach. The latter are called *hyperparameters*. Dimension of the hypotheses space is a hyperparameter, for instance.

A parameterdependent hypothesis sometimes is called a *model*. If a fixed set of parameters has been chosen, then it's a *trained model*.

23.1.2 Principal Steps in Supervised Learning

The starting point for all supervised machine learning methods is a finite collection of correctly labeled inputs:

$$\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}.$$

Based on experience, technical limitations, and more involved considerations, which will be discussed later on, we have to choose a model we want to fit to the examples. From this point on the process of supervised learning almost always follows a fixed scheme:

1. **Split** the data into training, validation and test sets.
2. **Train** the model (fit the parameters to the training set).
3. Optimize hyperparameters based on **validation** set.
4. Evaluate the model on the **test** set.
5. **Use** the trained model.

Training-Validation-Test Split

At first we split the set of examples into three disjoint sets: training set, validation set, test set. The training set usually is the largest subset and will be used for training the model. The validation set is used for optimizing hyperparameters, and the test set is used in the evaluation phase. Typical ratios are 60:20:20 or 40:30:30, but others might be appropriate depending on the complexity of the model and of the underlying true function f. Also the number of hyperparameters should be taken into account when splitting the set of examples. We will come back to this issue when dealing which concrete learning tasks.

Training

By training we mean determination of parameters in a parameterized hypothesis. This typically involves mathematical optimization methods. The result of training a model is a concrete hypothesis f_{approx}, which can be used to predict labels for unknow feature vectors.

Hyperparameter Optimization

The trained model might contain hyperparameters, the number of parameters, for instance. Choosing different hyperparameters could yield more accurate predictions. Thus, we should optimize prediction accuracy of the trained model with respect to the hyperparameters.

For this purpose we evaluate the trained model on the validation set and compare the predictions to the true labels. It's important to use examples different from the training examples, since on the training set the model always yields very good predictions (if our training was succesful). With a separate validation set, results are much more realistic. There are many different error measures for expressing the difference between predictions and true labels on a validation set. We will come back to this topic later on.

Choosing different sets of hyperparameters and training and evaluating corresponding models, we see what set of hyperparameters yields the best predictions.

Evaluation

After choosing optimal hyperparameters we have to test the final model on an independent data set: the test set. Now we see, whether the model also yields good predictions on feature vectors which did not take part in the training and validation process.

23.1.3 Python Packages for Supervised Learning

There exist many different Python packages implementing standard methods of supervised machine learning. In this book we focus on few packages only. The focus will be on understanding methods and algorithms. Packages we use:

- Scikit-Learn[382] is a very well structured and easy to use package. It provides insight to all algorithms and their working. Thus, it's best choice for learning and understanding new methods. For several years Scikit-Learn becomes more and more efficient. For instance, parallel computing is supported.
- Tensorflow[383] is a library for exploiting computational power of GPUs (graphics processing units). Typically, it's used for fast matrix computations in neural network training.
- Keras[384] provides a user friendly interface to Tensorflow (and other backends). It's the standard tool for deep learning with neural networks.

Other tools not covered here:

- PyTorch[385] is an alternative to Keras, which provides a more object-oriented programming interface.

[382] https://scikit-learn.org/stable/
[383] https://www.tensorflow.org/
[384] https://keras.io/
[385] https://pytorch.org/

23.1.4 Non-Numeric Data

All machine learning methods expect purely numeric data (because computers do so). Thus, we have to convert everything to numbers.

We already discussed one-hot encoding for categorical data in Pandas, see *Categorical Data* (page 235). Scikit-Learn supports one-hot encoding, too, with its `OneHotEncoder`[386] class in the `preprocessing` module.

Ordinal categegories (weekdays, for instance) may also be convert to sequences of integers (0 to 6 or 1 to 7 for weekdays). But beware of additional or incorrect structure we may add to data by conversion to numbers. Sometimes it's more difficult than one might expect (distance between weekdays 0 and 6 is the same as between 0 and 1, for instance).

Later on we will have to discuss about conversion of text to numbers and other non-obvious conversion problems.

23.2 Introductory Example (k-NN)

To demonstrate basic principles of supervised learning and their realization with Scikit-Learn we introduce the simplest supervised learning method here, the *k-nearest neighbors method* (k-NN).

23.2.1 Method

k-NN is a method without trainable parameters. Such non-parametric methods do not need a training phase (but may require hyperparameter optimization). All computations are done in the prediction phase. Thus, in contrast to almost all other machine learning methods, training is ver cheap and predictions are computationally expensive.

Given a feature vector x we look for the k training samples closest to x and make a prediction from those k neighbors:

- For classification problems we use the class appearing most often among the neighbors as prediction y.
- For regression problems we use the mean of the labels of all neighbors as prediction y.

Here k is a hyperparameter determining the size of the neighborhood predictions are based on. Different metrics for calculating distances in the feature space may be used, but Euclidean distance is the standard choice. The metric can be regarded as a hyperparameter, too.

The k-NN method per prediction requires as many distance computations in the feature space as there are samples in the training data set. Thus, for very large data sets k-NN might be computationally infeasible.

23.2.2 Scaling

To ensure comparable influence of all features on the computed distances all features should have similar numerical range. Else features with high numerical values would dominate the distance measure. This scaling problem arises for almost all machine learning methods and will be discussed in detail in *Scaling* (page 342).

Example

Consider classification of used cars with features age (in years) and kilometers driven. Classes might be 'low quality' and 'high quality', but do not matter here. We look at three samples:

$$x_1 = (1, 100000), \quad x_2 = (11, 100000), \quad x_3 = (1, 99990).$$

To predict the quality of sample x_1 with k-NN we have to compute distances to all other samples. For both x_2 and x_3 the distance is $\sqrt{10^2 + 0^2} = 10$. That is, an age difference of 10 years (with equal kilometers) has the same influence on predictions like a kilometer difference of 10 (with equal age). The age difference should be very relevant for the car's quality, but the tiny kilometer difference won't.

[386] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

Rescaling the kilometer feature by factor 1000, that is, expressing the feature as multiples of 1000 kilometers, the samples look like

$$x_1 = (1, 100), \quad x_2 = (11, 100), \quad x_3 = (1, 99.99)$$

and distances of x_1 to x_2 and x_3 are 10 and 0.01, respectively. Now the old car x_2 is a lot further away from x_1 than the young car x_3. Consequently, x_3 is much more likely to influence k-NN predictions for x_1 than x_2.

Important: Prepare your data well!

The example above shows that correct data preprocessing is of utmost importance. Even simple steps like scaling may alter results obtained from machine learning methods significantly.

23.2.3 Weights

Often k-NN includes some kind of weighting. A common approach is to weight the feature vectors by their distance to the feature vector we want to predict the label for. Denote by $d_1, \ldots, d_k$ the distances of the k nearest neighbors and $y_1, \ldots, y_k$ are corresponding labels.

For regressions tasks the prediction then is

$$y_{\text{pred}} = \frac{\sum\limits_{\kappa=1}^{k} \frac{1}{d_\kappa}\, y_\kappa}{\sum\limits_{\kappa=1}^{k} \frac{1}{d_\kappa}}.$$

Note that weights always have to add up to 1 for regression tasks.

For classification tasks distance-weighted k-NN predicts the class for which the sum of all weights in the neighborhood is maximal:

$$y_{\text{pred}} = \underset{c\in\{1,\ldots,C\}}{\operatorname{argmax}} \sum_{\substack{\kappa\in\{1,\ldots,k\}\\ y_\kappa = c}} \frac{1}{d_\kappa}$$

Here classes are denoted $1, \ldots, C$.

Whenever Scikit-Learns wants to have a data set, it expects a Numpy array of shape $n \times m$, where n is the number of samples in the data set and m is the number of features. Corresponding labels (for training) have to be provided in a one-dimensional array of length n.

23.2.4 k-NN with Scikit-Learn

All methods implemented in Scikit-Learn follow the same approach:

1. Create a suitable Scikit-Learn object, which provides the method and encapsulates parameters and hyperparameters as well as results.
2. Call the object's `fit` method (that's the training step).
3. Investigate results (parameters and so on, depending on the method).
4. Call the object's `predict` method to predict labels for non-training data.

Scikit-Learn implements k-NN regression in `KNeighborsRegressor`[387] and k-NN classification in `KNeighborsClassifier`[388]. The `fit` method stores training data internally (remember, k-NN requires no training) and `predict` then uses stored training data for calculating the distances and choosing a prediction.

The import name of Scikit-Learn is `sklearn`. The installation name is `scikit-learn`.

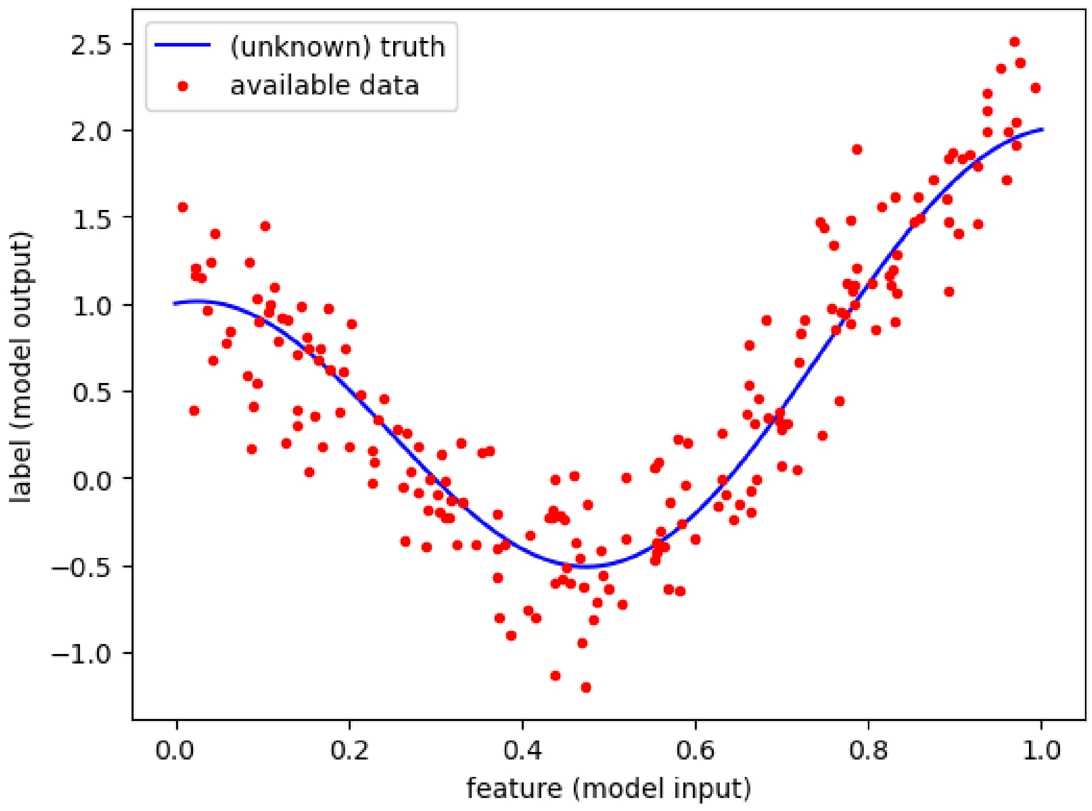
[387] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html
[388] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Regression Example

For demonstration we use a small one-feature synthetic data set.

```python
import numpy as np
import matplotlib.pyplot as plt

import sklearn.neighbors as neighbors
import sklearn.model_selection as model_selection
import sklearn.metrics as metrics

rng = np.random.default_rng(42)
```

```python
def truth(x):
    return x + np.cos(2 * np.pi * x)

xmin = 0
xmax = 1
x = np.linspace(xmin, xmax, 1000)

n = 200    # number of data points to generate
noise_level = 0.3    # standard deviation of artificial noise

# simulate data
X = (xmax - xmin) * rng.random((n, 1)) + xmin
y = truth(X).reshape(-1) + noise_level * rng.standard_normal(n)

# plot truth and data
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='(unknown) truth')
ax.plot(X.reshape(-1), y, 'or', markersize=3, label='available data')
ax.legend()
ax.set_xlabel('feature (model input)')
ax.set_ylabel('label (model output)')
plt.show()
```



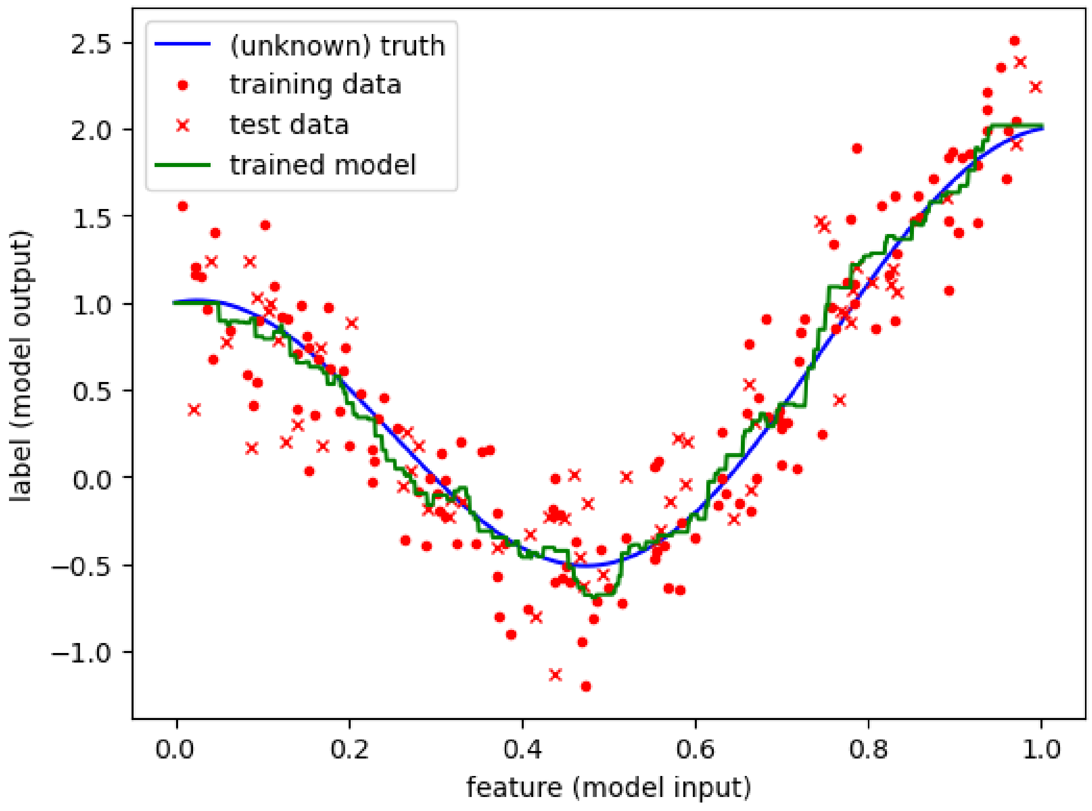
Weightless k-NN (`weights='uniform'`) yields piecewise constant hypotheses, which can be seen by using small `n`. This is not true if weighting by distance is used (`weights='distance'`).

To evaluate the model's prediction quality after training we have to split our data set into a training set and a test set. Training and test samples should be chosen at random to avoid bias due to sorted or otherwise structured samples. Scikit-Learn provides `train_test_split`[389] in its `model_selection` module for this purpose. For the moment we do not consider hyperparameter optimization. Thus, no validation set is needed.

```
# split data into train and test sets
X_train, X_test, y_train, y_test \
    = model_selection.train_test_split(X, y, test_size=0.3, random_state=0)
print(y_train.size, y_test.size)

# regression
knn = neighbors.KNeighborsRegressor(10, weights='uniform')
knn.fit(X_train, y_train)

# get predictions in grid for plotting the hypothesis
y_knn = knn.predict(x.reshape(-1, 1))

# plot truth, data, hypothesis
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='(unknown) truth')
ax.plot(X_train.reshape(-1), y_train, 'or', markersize=3, label='training data')
ax.plot(X_test.reshape(-1), y_test, 'xr', markersize=5, label='test data')
ax.plot(x, y_knn, '-g', label='trained model')
ax.legend()
ax.set_xlabel('feature (model input)')
ax.set_ylabel('label (model output)')
plt.show()
```

[389] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

```
140 60
```



In the evaluation phase we calculate some suitable metric on the test set. Here we use mean squared error (MSE)[390].

```
print('test error:', metrics.mean_squared_error(y_test, knn.predict(X_test)))
print('train error:', metrics.mean_squared_error(y_train, knn.predict(X_train)))
```

```
test error: 0.10849356915074192
train error: 0.07957991672205268
```

Now we may use the trained model to get predictions for arbitrary inputs. Simply call the model's `predict`[391] method.

Classification Example

Here is a synthetic two-feature binary classification example.

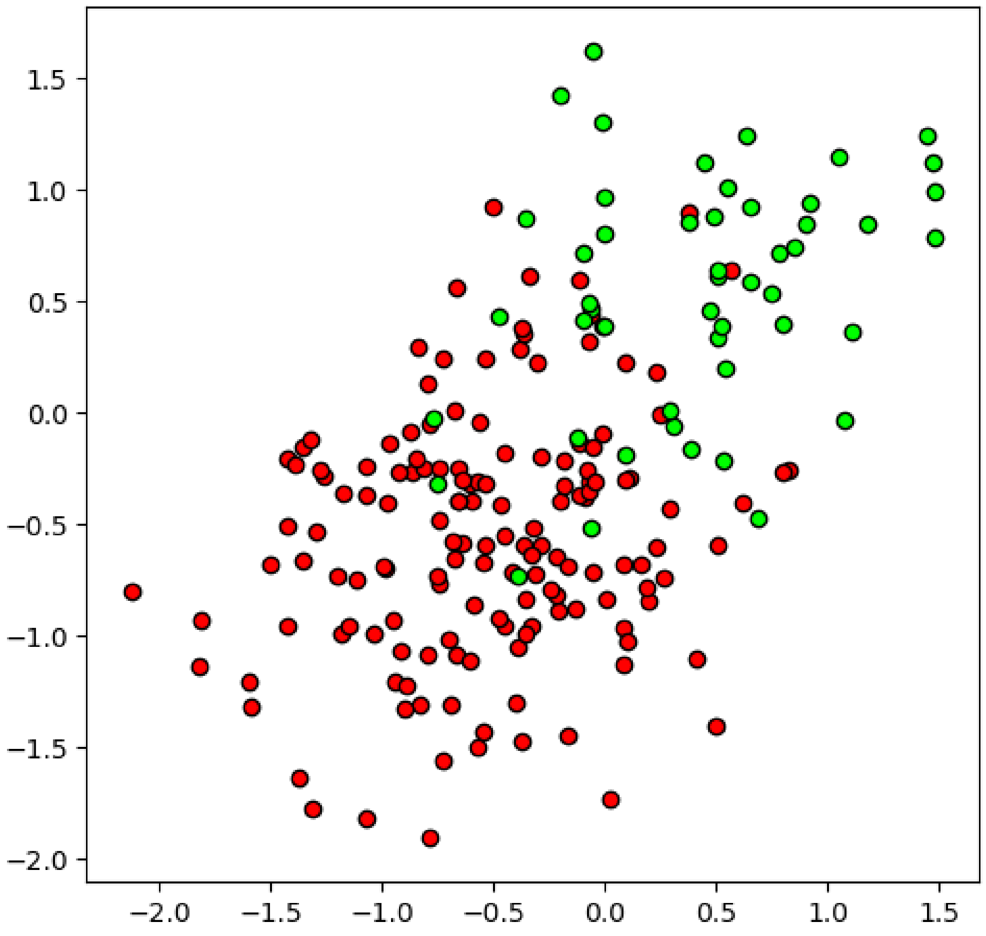
```
n0 = 150    # training samples in class 0
n1 = 50    # training samples in class 1

# generate two point clouds (classes)
X0 = rng.multivariate_normal([-0.5, -0.5], [[0.3, 0], [0, 0.3]], size=n0)
X1 = rng.multivariate_normal([0.5, 0.5], [[0.3, 0], [0, 0.3]], size=n1)
X = np.concatenate((X0, X1))

# set labels
```

(continues on next page)

[390] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html
[391] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbors.KNeighborsRegressor.predict

(continued from previous page)

```
y0 = np.zeros(n0, dtype=np.uint8)
y1 = np.ones(n1, dtype=np.uint8)
y = np.concatenate((y0, y1))

# set plotting region
x0_min = X[:, 0].min() - 0.2
x0_max = X[:, 0].max() + 0.2
x1_min = X[:, 1].min() - 0.2
x1_max = X[:, 1].max() + 0.2

# plot data set
fig, ax = plt.subplots(figsize=(6, 6))
ax.scatter(X[y == 0, 0], X[y == 0, 1], c='#ff0000', edgecolor='black')
ax.scatter(X[y == 1, 0], X[y == 1, 1], c='#00ff00', edgecolor='black')
ax.set_xlim(x0_min, x0_max)
ax.set_ylim(x1_min, x1_max)
ax.set_aspect('equal')
plt.show()
```



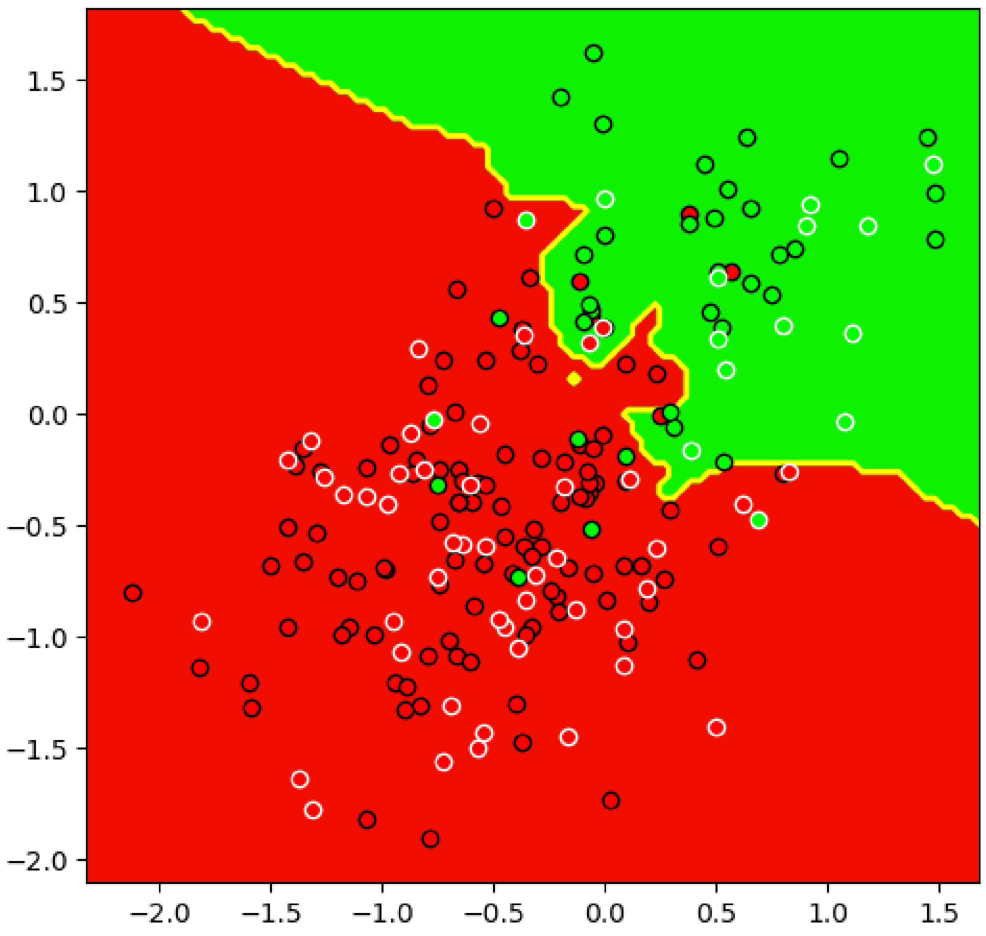
```
from matplotlib.colors import LinearSegmentedColormap

X_train, X_test, y_train, y_test \
    = model_selection.train_test_split(X, y, test_size=0.3, random_state=0)
print(y_train.size, y_test.size)

knn = neighbors.KNeighborsClassifier(5, weights='uniform')
knn.fit(X_train, y_train)
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots(figsize=(6, 6))

# plot trained model (function values color-coded)
x0, x1 = np.meshgrid(np.linspace(x0_min, x0_max, 100), np.linspace(x1_min, x1_max,
 ↪ 100))
X_grid = np.concatenate((x0.reshape(-1, 1), x1.reshape(-1, 1)), axis=1)
y_grid = knn.predict(X_grid).reshape(100, 100)
cm = LinearSegmentedColormap.from_list('rg', ['#ff0000', '#00ff00'])
ax.contourf(x0, x1, y_grid, cmap=cm, levels=np.linspace(0, 1, 10))

# plot decision boundary
ax.contour(x0, x1, y_grid, levels=[0.5], linewidths=[2], colors=['#ffff00'])

# plot training and test data
ax.scatter(X_train[y_train == 0, 0], X_train[y_train == 0, 1], c='#ff0000',␣
 ↪edgecolor='black', zorder=1000)
ax.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], c='#00ff00',␣
 ↪edgecolor='black', zorder=1000)
ax.scatter(X_test[y_test == 0, 0], X_test[y_test == 0, 1], c='#ff0000', edgecolor=
 ↪'white', zorder=1000)
ax.scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], c='#00ff00', edgecolor=
 ↪'white', zorder=1000)

ax.set_xlim(x0_min, x0_max)
ax.set_ylim(x1_min, x1_max)
ax.set_aspect('equal')

plt.show()

# evaluation
print('test accuracy:', metrics.accuracy_score(y_test, knn.predict(X_test)))
print('train accuracy:', metrics.accuracy_score(y_train, knn.predict(X_train)))
```

```
140 60
```



```
test accuracy: 0.9166666666666666
train accuracy: 0.9214285714285714
```

Here we used accuracy[392] as quality measure.

23.2.5 Further Aspects to Consider

Odd k for Binary Classification

In binary classification even k may result in deadlocks, which typically are handled by random choice of one class. Obviously, it's a good idea to choose odd k.

Imbalanced Classes

Care has to be taken with imbalanced classes. If one class has much more samples than other classes, then that class might dominate almost all neighborhoods and samples from smaller classes get outshined.

Imbalanced classes are a serious problem in machine learning if one wants to classify rare events or objects like failures in production lines, for instance (inputs are sensor data, output is failture/no failture). It's easy to get lots of positive (no failure) samples. But there will be only few training samples from the failure class.

Possible solutions are dropping samples from the larger class or to introduce weights based on expected or actual class sizes (in the production line example: multiply distances to 'no failure' samples by expected failure rate). Details heavily depend on the concrete context.

[392] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

Try

```
n0 = 300
n1 = 10

X0 = rng.multivariate_normal([-0.5, -0.5], [[0.3, 0], [0, 0.3]], size=n0)
X1 = rng.multivariate_normal([-0.5, -0.5], [[0.01, 0], [0, 0.01]], size=n1)
```

in the classification example above. The small region with green samples is surrounded and permeated by lots of red samples. Thus, even in the green region neighborhoods contain more red than green samples.

Important: There is no standard solution! and **Know your data!**

Each machine learning task comes with its own set of difficulties. Even the application of simple methods like k-NN requires a considerable amount of brain-work to get useful and reliable results. The most important step is to know and understand your data.

The Curse of Dimensionality

If the hyperparameter k is too large, then predictions will be stable (altering/removing individual neighbors won't change predictions) but inaccurate (very distant samples influence predictions). If k is too small, then predictions are very sensitive to modifications and noise in the training data.

Exponential Growth of k

It's important to note that the choice of k heavily depends on the dimension m of the feature space. In higher dimensions we have to look at much more neighbors to get sufficient information about the local behavior of the approximated function. More precisely, k should be proportional to 2^m (this fact is of little use for choosing concrete k, but it's a good starting point for developing some intuition on choosing good hyperparameters).

Imagine the vertices of a hypercube in m-dimensional space. For $m = 1$ it's an interval with two vertices (end points). For $m = 2$ it's a square with four vertices. For $m = 3$ it's a usual cube with 8 vertices. For general m a hypercube has 2^m vertices. If (in average) we want to have one neighbor on each 'side' of an input (more precisely: one neighbor per orthant[393]), we have to set $k = 2^m$. Such heavy influence of space dimension is sometimes called the *curse of dimensionality*.

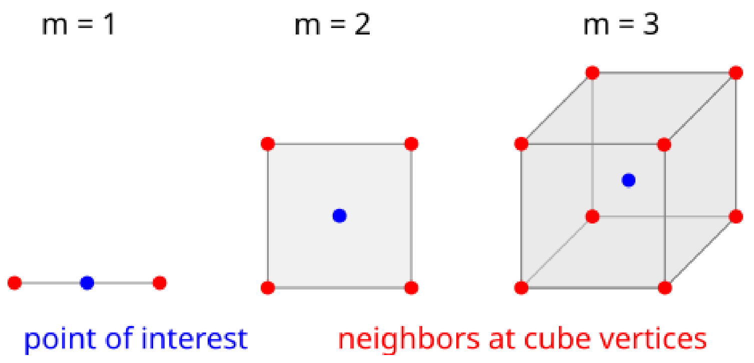


Fig. 23.3: Even if a point of interest only has neighbors at the vertices of a surrounding cube, number of neighbors grows exponentially with dimension.

[393] https://en.wikipedia.org/wiki/Orthant

Distances in High Dimensions

The curse of dimensionality causes also another problem: in high dimensions almost all distances between points are large and almost identical. The higher the space dimension the less dense the data set.

To understand the problem in more detail we consider n feature vectors uniformly distributed in the m-dimensional hypercube $[0,1]^m$. Fix some feature vector (the one we want to predict the label for) and a hypersphere with radius R centered at that feature vector. Denote the number of feature vectors inside the hypersphere by k. Then, due to uniformly distributed data, $\frac{k}{n}$ will be very close to the volume of a hypersphere divided by the hypercube's volume. The hypercube has volume one. Thus,

$$\frac{k}{n} \approx \text{volume of hypersphere} \qquad \text{or} \qquad k \approx n \cdot \text{volume of hypersphere.}$$

Have a look at the volume of the hypersphere (Wikipedia)[394] with radius R:

dimension	volume	volume for $R = 0.1$
1	$2\,R$	0.2
2	$3.142\,R^2$	0.0342
3	$4.189\,R^3$	0.004189
4	$4.935\,R^4$	0.0004934
5	$5.264\,R^5$	0.00005264

The volume of a hypersphere drastically decreases (compared to the cubes volume) if dimension grows. Thus, the number of feature vectors inside the sphere decreases with growing dimension. Or the other way round: the higher the dimension the farther away are neighboring points. Consequently, almost all points seem to be close to the surface of the hypercube and the interior is more or less empty.

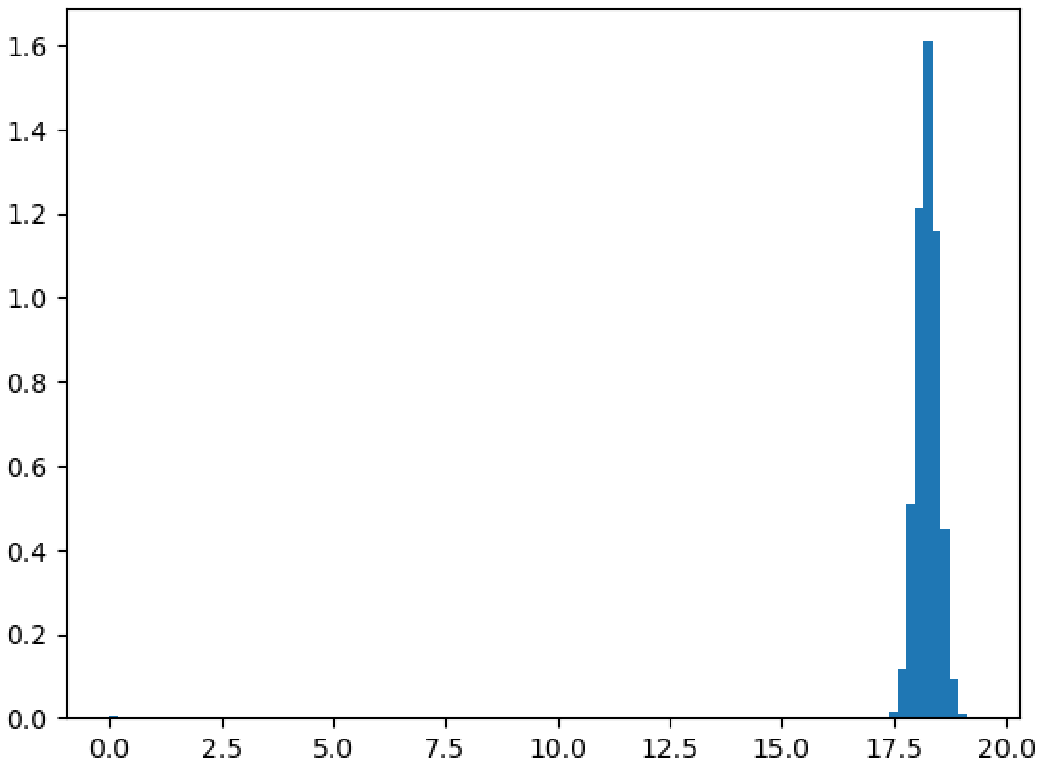
But remember: throughout the dimensionality discussion we assumed that points are uniformly (!) distributed in space. The problem is the Euclidean distance which is not able to grasp this uniform distribution in high dimensions.

The following code illustrates the theoretical considerations numerically (Scikit-Learn's `pairwise_distances`[395] saves some work here):

```
from sklearn.metrics import pairwise_distances

dim = 2000     # dimension of data space
n = 1000     # number of samples

# generate samples uniformly distributed in unit (hyper)cube
X = rng.uniform(dim * (0,), dim * (1,), (n, dim))

# calculate pairwise distances between samples
dists = pairwise_distances(X).reshape(-1)

# plot histogram of pairwise distances
fig, ax = plt.subplots()
ax.hist(dists, 100, density=True)
plt.show()
```

[394] https://en.wikipedia.org/wiki/Volume_of_an_n-ball

[395] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise_distances.html



Try different values for `dim`: 2, 20, 200, 200.

Don't Buy High-Dimensional Melons

From the above discussion we may deduce that high-dimensional melons only consist of green skin with only very little red interior. We may compute this result directly. To simplify calculations we assume that a melon is a cube and one tenth of the edge length l is green skin. Then in m dimensions the red interiors volume compared to the melon's total volume is

$$\frac{V_{\text{red}}}{V_{\text{total}}} = \frac{(0.9\,l)^m}{l^m} = 0.9^m,$$

which goes to zero for $m \to \infty$.

For spheric melons similar computations yield exactly the same result 0.9^m.

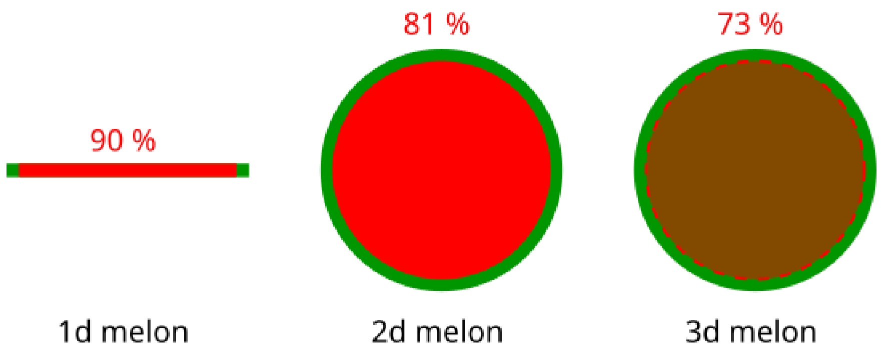


Fig. 23.4: The best melon is a 1d melon. But 3d melons are okay, too.

Practical Implications

Increasing the size of the training data set might help to overcome the curse of dimensionality. But we would have to collect much more data than possible if dimension of feature space is high. Some concrete numbers: In a 30-dimensional feature space (say a hypercube) we have 1 billion vertices. Thus, we would need 1 billion samples to have at least one sample per vertex in average. In two dimensions equivalent data set size would be 4 samples (one per vertex).

There is also some good news. Most data sets are not uniformly distributed. Instead samples concentrate around lower-dimensional manifolds. Thus, we do not have to cover the entire space with samples. Feature/dimensionality reduction is an important preprocessing step for k-NN, see *Feature Reduction* (page 343).

Important: Math matters!

Human imagination fails in high dimensions. Math still works.

23.3 Quality Measures

To evaluate and sometimes also to train a machine learning model we have to express approximation quality on given set of samples numerically. Choosing a suitable quality measure heavily depends on the underlying task. A standard measures for regression problems is the mean of the squared Euclidean distances between correct and predicted output, in this context usually referred to as *mean squared error*. For classification tasks the *correct classification rate* (sometimes denoted as *accurcy*) is a straight-foward quality measure.

23.3.1 Quality Measures for Regression Problems

For regression tasks all quality measures have the form

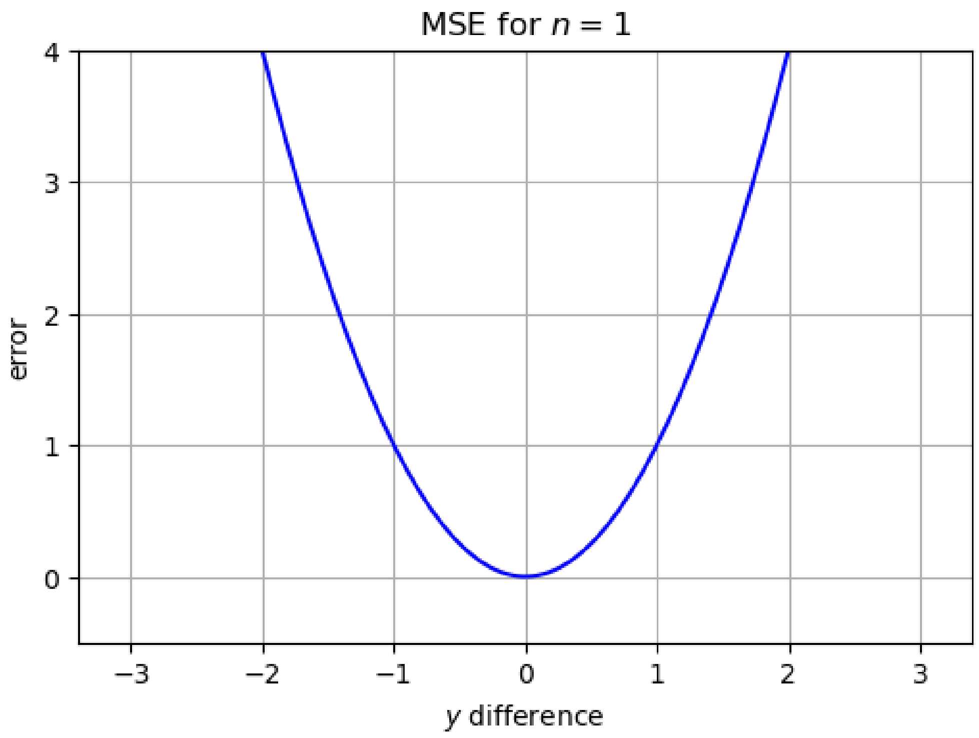
$$\frac{1}{n}\sum_{l=1}^{n} l(f_{\text{approx}}(x_l), y_l),$$

where $(x_1, y_1), \ldots, (x_n, y_n)$ are the samples (inputs and correct labels) on which to compute f_{approx}'s predition quality and $l : \mathbb{R} \to \mathbb{R}$ is some (usually nonnegative) function. Quality measures of this form often are denoted as *loss functions*.

Mean Squared Error

Mean squared error (MSE) is the Euclidean distance between the vectors of correct and predicted outputs:

$$\frac{1}{n}\sum_{l=1}^{n} \left(f_{\text{approx}}(x_l) - y_l\right)^2.$$



```
<IPython.core.display.HTML object>
```

Use of this loss function is motivated by three major advantages:

- It is differentiable and, thus, accessible to efficient optimization algorithms.
- Solutions to corresponding minimization problems in many cases can be made explicit.
- One can prove mathematically that mean squared error is the (in some sense) optimal loss function if labels follow a Gaussian distribution. Details will be addressed in statistics lectures (*maximum aposteriory probability estimation* or *MAP estimation* for short).

A drawback is its sensitivity to outliers. Due to its quadratic nature, small deviations between predicted and correct labels result in very small changes of the loss function only. In contrast, large deviations (outliers!) result in extremely large values of the corresponding summand in the loss function.

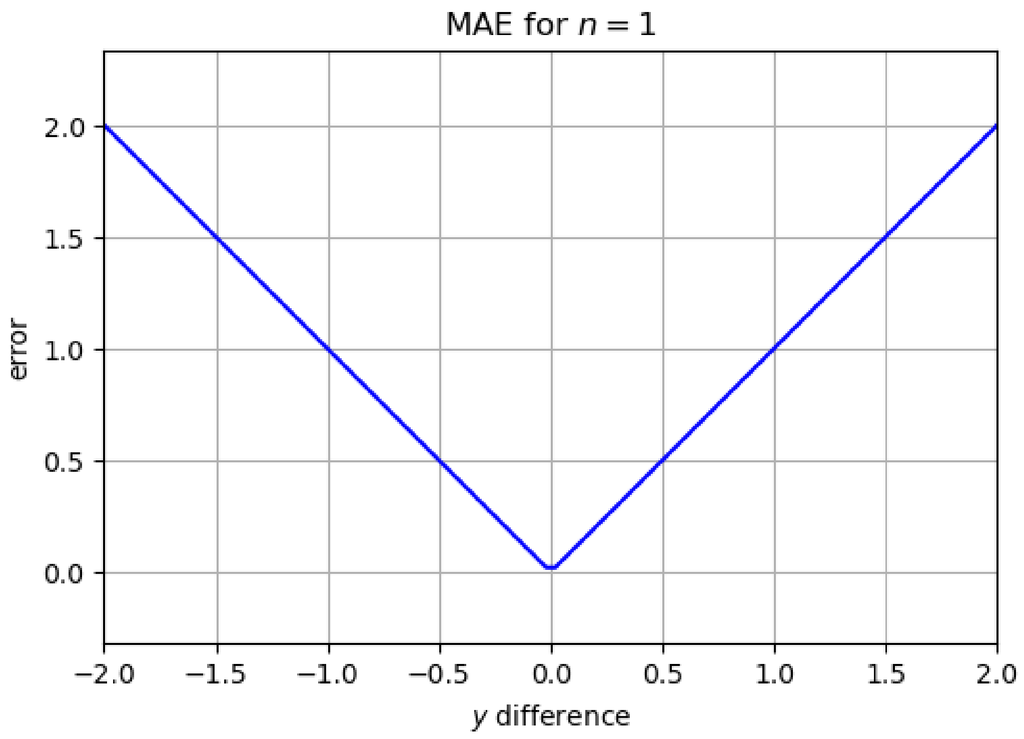
Scikit-Learn provides `mean_squared_error`[396] in its `metrics` module.

Mean Absolute Error

Sensitivity to outliers of mean squared error stems from its quadratic nature. So it's reasonable to have a look on non-quadratic loss functions. The simplest non-quadratic loss function is the mean absolute error (MAE):

$$\frac{1}{n}\sum_{l=1}^{n}\left|f_{\text{approx}}(x_l)-y_l\right|.$$

[396] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html



```
<IPython.core.display.HTML object>
```

Its value is proportional to the deviation of predictions from correct labels and does not overemphasize large deviations. A major drawback is its non-differentiability. Neither can we compute explicit solutions to corresponding minimization problems, nor can we use simple gradient based optimization algorithms.

Scikit-Learn provides `mean_absolute_error`[397] in its `metrics` module.

Huber Loss

Huber loss is a mixture of mean squared error and mean absolute error. It tries to combine the advantages of both. At small arguments Huber loss is quadratic (and thus differentiable). At larger arguments Huber loss is linear (and thus less sensitive to outliers). Huber loss is of the form

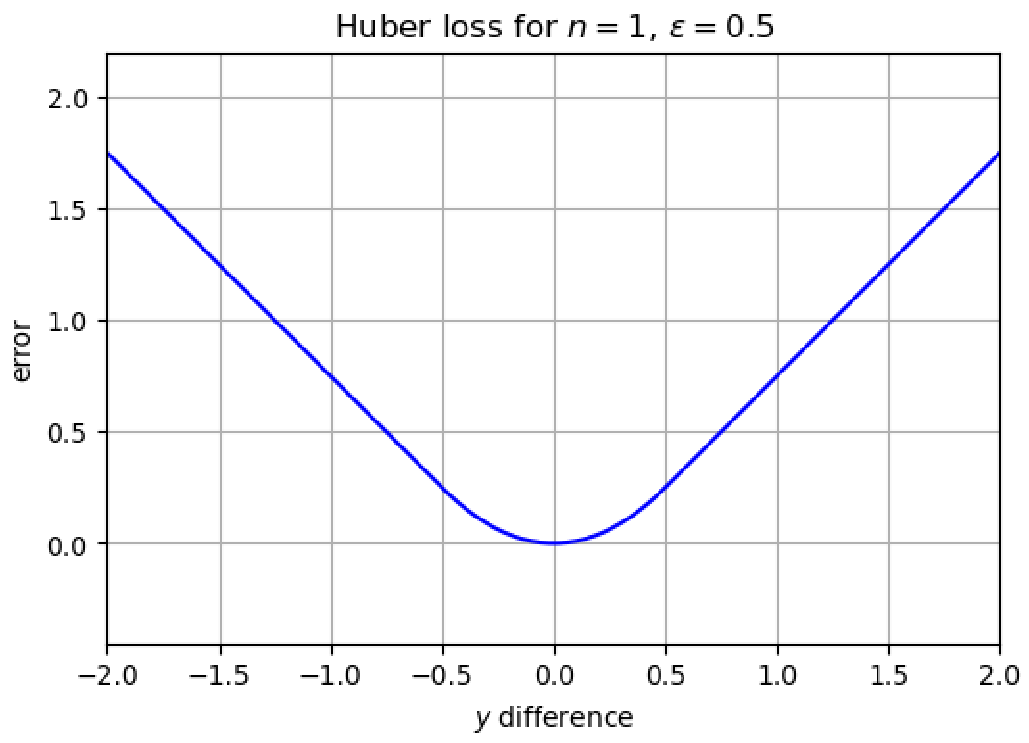
$$\frac{1}{n}\sum_{l=1}^{n} h(f_{\text{approx}}(x_l) - y_l)$$

with

$$h(z) = \begin{cases} z^2, & \text{if } |z| \leq \varepsilon, \\ 2\,\varepsilon\,|z| - \varepsilon^2, & \text{else.} \end{cases}$$

The parameter $\varepsilon > 0$ determines the value at which quadratic growth is replaced by linear growth. Statistical considerations suggest $\varepsilon = 1.35$. But other choice may be more appropriate in non-standard situations. Huber loss typically includes scaling the data to make the choice of ε independent from the data's range.

[397] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html#sklearn.metrics.mean_absolute_error



```
<IPython.core.display.HTML object>
```

Scikit-Learn does not provide the Huber loss in its `metrics` module, but it implements some machine learning methods using that loss implicitly.

23.3.2 Classification Outputs

The final output of classification methods should be class labels. But many algorithms do not yield a concrete label but for each class a probability value that the input belongs the that class. Alternatively algorithms may use a scoring scheme, where each class is assigned a score. High scores indicate high probability that the input belongs to that class.

In total we have three different output variants from classification algorithms: labels, probabilities, scores. Thus, we have to think about conversion methods between them and we will meet quality measures suitable for one variant but not for others. Thus, choice of quality measures also depends on the classification method's output 'format'.

Before we come to concrete quality measures we discuss conversion methods between the three output variants.

From Probabilities to Labels

Although classification problems deal with discrete targets most classification algorithms yield values in the continuous range $[0,1]$. Given an input the algorithm may provide as many values from $[0,1]$ as there are classes. Each such value can be interpreted as the probability that the input belongs to the corresponding class. In some cases the values do not add up to 1, thus it might not be a probability in the mathematical sense. In addition, if we do not specify the underlying probabilistic framework, we should not speek about probabilities. Thus, a better alternative is *probability-like score*.

For binary (that is, 2-class) classification problems there is usually only one output $p \in [0,1]$. This can be seen as the probability for the input to belong to the one class. The probability for the other class then is $1-p$. To derive a concrete class from p we may use a threshold value $t \in (0,1)$:

$$\begin{aligned} p \leq t \quad &\Rightarrow \quad \text{class A,} \\ p > t \quad &\Rightarrow \quad \text{class B.} \end{aligned}$$

Next to the canonical choice $t = 0.5$ other choice may be appropriate in some situations.

For multiclass problems the class with the highest probability-like score is chosen. If there are several classes with high probaility, then the algorithm wasn't able to make a clear decision. Knowledge about class probabilities can be used to improve the algorithm or to inform the user about a less trustworthy result.

From Scores to Probabilities

If an algorithm outputs scores not lying in $[0, 1]$ we may transform them to get probability-like scores. For bounded scores, say in $[a, b]$, the simple linear transform

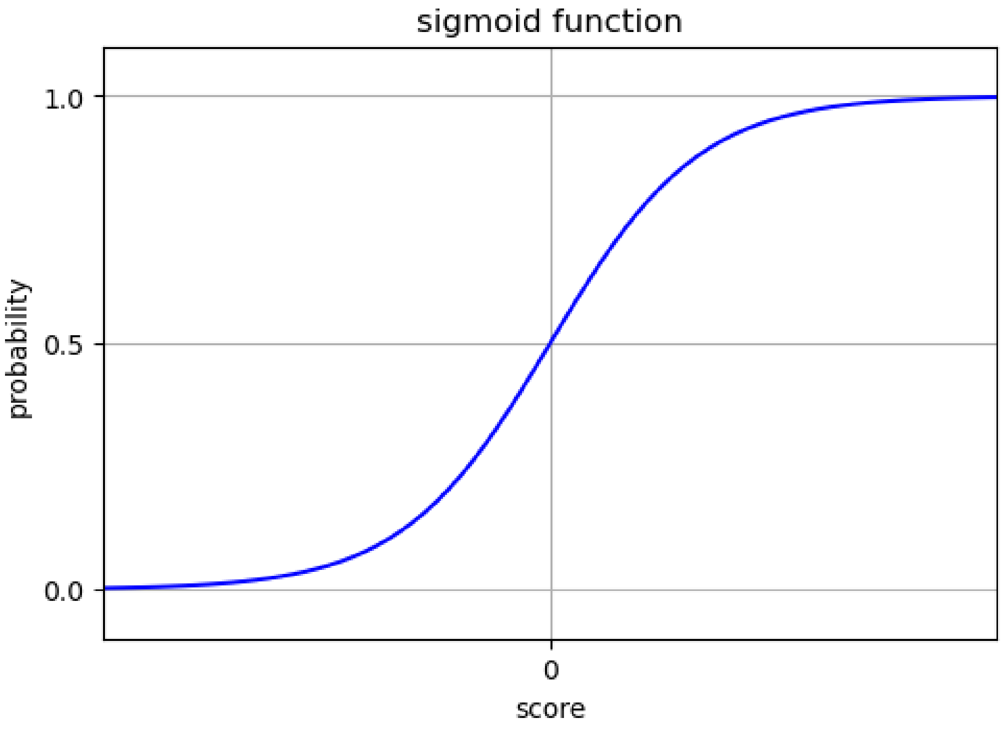
$$z \mapsto \frac{z - a}{b - a}$$

does a good job. But the unbounded case is not as simple as the bounded one.

If scores are arbitrary (unbounded) values in $\mathbb{R}$, the transform should be defined on the whole real axis, it should be monotonically increasing, and it should map negative numbers to $[0, \frac{1}{2})$ and positive numbers to $(\frac{1}{2}, 1]$. The *sigmoid function*

$$z \mapsto \frac{1}{1 + \mathrm{e}^{-z}}, \qquad z \in \mathbb{R},$$

shows all those properties.

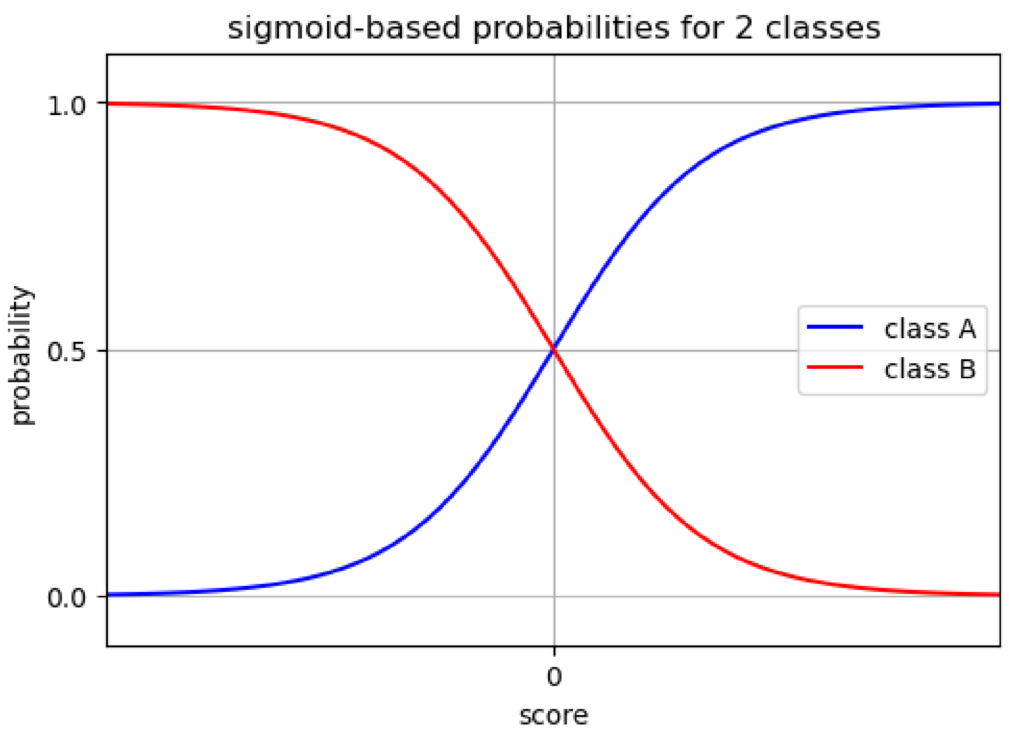


Note that 0 and 1 do not belong to the range of the sigmoid function, but

$$\lim_{z \to -\infty} \frac{1}{1 + \mathrm{e}^{-z}} = 0 \qquad \text{and} \qquad \lim_{z \to \infty} \frac{1}{1 + \mathrm{e}^{-z}} = 1.$$

In case of binary classification with classes A and B we typically only have one output $z \in \mathbb{R}$ with high values indicating that a sample belongs to class A and low values indicating that a sample belongs to class B. Predicted class probabilities can be defined as

$$\text{probability for class A} := \frac{1}{1 + \mathrm{e}^{-z}}$$
$$\text{probability for class B} := 1 - \frac{1}{1 + \mathrm{e}^{-z}} = \frac{\mathrm{e}^{-z}}{1 + \mathrm{e}^{-z}} = \frac{1}{1 + \mathrm{e}^{z}}.$$



For multiclass classification with classes $1,\dots,C$ and class scores $z_1,\dots,z_C$ we may use the *softmax function*:

$$\text{probability for class } i := \frac{\mathrm{e}^{z_i}}{\mathrm{e}^{z_1}+\cdots+\mathrm{e}^{z_C}},\qquad i=1,\dots C.$$

These values lie in $[0,1]$ and sum up to 1.

```
<IPython.core.display.HTML object>
```

Note: It's a nice little math exercise to prove that softmax-based probabilites for two classes with two scores solely depend on the difference of the two scores, not on their absolute values.

As by-product one sees that for two classes computing probabilities from two independent scores (via softmax) is the same as computing probabilities from one score (via sigmoid) if one identifies the one score with the difference of the two scores.

We'll meet sigmoid and softmax functions in slightly different contexts when discussing machine learning methods like artificial neural networks and logistic regression.

23.3.3 Quality Measures for Classification

There are many different measures for prediction quality of classification algorithms. Here we mention only the most important ones. Given a data set each measure either compares predicted labels to correct labels or it's based on class probabilities. In all cases the outcome is a positive real number expressing some kind of prediction quality.

Correct Classification Rate (Accuracy)

Correct classification rate (or accuracy) counts the number of correct predictions and normalizes the result to $[0, 1]$, where 0 indicates no correct predictions and 1 indicates that all predictions are correct. Formula:

$$\text{accuracy} := \frac{\text{correct predictions}}{\text{number of samples}}.$$

Care has to be taken in interpretation, because accuracy above 0 does not mean that the classifier does anything useful. We have to compare the accuracy of the model under consideration to the accuracy of a purely random classifier. A purely random classifier assigns labels by chance equally distributed over all classes. Thus, accuracies close to or below

$$\frac{1}{\text{number of classes}}$$

are very bad.

Important: Always think twice! If we have spent hours in training a classifier on cat and dog images and evaluation on independent data shows a correct classification rate of 55 per cent, is this a good result? No! Assiging labels 'cat' and 'dog' by chance we would get nearly the same result (50 per cent accuracy) with almost no computational effort.

There is another, more severe issue. Correct classification rate only yields reasonable results if the test set is balanced, that is, if the number of samples per class does not depend on the class.

Example: Consider three classes A, B, C and a test set with 100 samples;

$$5 \text{ in A}, \quad 10 \text{ in B}, \quad 85 \text{ in C}.$$

If a (very simple) classifier always chooses class C, then accuracy is 85 per cent, which sounds pretty good! If we have a test set with

$$50 \text{ in A}, \quad 40 \text{ in B}, \quad 10 \text{ in C},$$

then accuracy drops to 10 per cent, although it's still the same classifier. With a balanced test set, say

$$33 \text{ in A}, \quad 33 \text{ in B}, \quad 34 \text{ in C},$$

accuracy is 34 per cent, which is close to the accuracy of a purely random classifier. Thus we see that the classifier does not do anything useful.

Scikit-Learn provides `accuracy_score`[398] in its `metrics` module.

Balanced accuracy

For imbalanced test sets correct classification rate does not yield sensible results. But imbalanced test sets are much more common than balanced ones. To obtain an accuracy-like quality measure for imbalanced test sets we measure the accuracy on each class and then take the mean over all classes. Here accuracy on a class is the number of correctly classified samples from a class devided by the class size. If we have C classes, then

$$\text{balanced accuracy} := \frac{1}{C}\sum_{i=1}^{C}\frac{\text{number of samples correctly labeled as class } i}{\text{total number of samples in class } i}.$$

[398] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

Example: Consider the three classes problem with a classifier always predicting class C again. If 5 samples belong to class A, 10 to B, 85 to C, then we have

$$\begin{aligned}
&\text{5 in A,} \quad \text{10 in B,} \quad \text{85 in C} \quad \Rightarrow \quad \text{bal. acc.} = \frac{1}{3}\left(\frac{0}{5}+\frac{0}{10}+\frac{85}{85}\right)=\frac{1}{3}.\\
&\text{50 in A,} \quad \text{40 in B,} \quad \text{10 in C} \quad \Rightarrow \quad \text{bal. acc.} = \frac{1}{3}\left(\frac{0}{50}+\frac{0}{40}+\frac{10}{10}\right)=\frac{1}{3}.\\
&\text{33 in A,} \quad \text{33 in B,} \quad \text{34 in C} \quad \Rightarrow \quad \text{bal. acc.} = \frac{1}{3}\left(\frac{0}{33}+\frac{0}{33}+\frac{34}{34}\right)=\frac{1}{3}.
\end{aligned}$$

The example shows that balanced accuracy does not depend on possibly imbalanced class sizes in the test set. In all cases we obtain results indicating that the classifier is not better than a purely random classifier. Note that with a purely random classifier for each test set we would have

$$\text{bal. acc.} = \frac{1}{3}\left(\frac{1}{3}+\frac{1}{3}+\frac{1}{3}\right)=\frac{1}{3}.$$

Scikit-Learn provides `balanced_accuracy_score`[399] in its `metrics` module.

Confusion Matrix

Although confusion matrices are not a numeric quality measure they offer a good overview of classification results. For classification with classes $1,\dots,C$ the confusion matrix is a $C \times C$ matrix showing in row i and column j the number of samples belonging to class i and classified as j.

If all samples were classified correctly, then the confusion matrix is purely diagonal.

Scikit-Learn provides `confusion_matrix`[400] in its `metrics` module.

Log Loss

Log loss (or logistic regression loss or cross-entropy loss) is a loss, not a score. That is, best prediction quality is indicated by zero log loss and the higher the loss the lower the prediction quality (to get a score, use negative log loss). Log loss requires predicted class probabilities, not class labels.

Given a test set with n samples we denote the true labels by $y_1,\dots,y_n$ and the predicted probabilities that sample l belongs to class y_l (its true class) by $p_1,\dots,p_n$. Then

$$\text{log loss} := -\frac{1}{n}\sum_{l=1}^{n}\log p_l = -\log\left(\left(\prod_{l=1}^{n} p_l\right)^{\frac{1}{n}}\right).$$

The product $\prod p_l$ is the predicted probability that each sample belongs to its true class. The n-th root ensures independence from test set size (like the factor $\frac{1}{n}$ for sums over all samples). The negative logarithm transforms the interval $[0,1]$ to $[0,\infty]$ by mapping 0 to ∞ and 1 to 0. In other words, the negative logarithm transforms a probability (high is good) into a loss (low is good). Other functions for such tranform may be used, too, but negative logarithm is the most widely used one.

Note: Always try to explain formulas in few words without mathematical symbols. The log loss formula could be explained as follows: **Log loss expresses (up to scaling) how convinced the model is that each sample belongs to the class it really belongs to.**

Then consider edge cases: Log loss is zero if the model assignes probability 1 to the correct classes for all samples. Log loss tends to infinity if the model assignes only very small probabilities to the correct classes for all samples.

[399] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html
[400] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

Often log loss is written as a more complex formula explicitly showing the dependence on the true labels $y_1,\ldots,y_n$. With classes $1,2,\ldots,C$ set

$$y_{l,i}:=\begin{cases}1, & \text{if } y_l=i,\\ 0, & \text{else}\end{cases}\qquad\text{for}\quad l=1,\ldots,n\quad\text{and}\quad i=1,\ldots,C$$

(one-hot encoding) and denote by $p_{l,i}$ the predicted probability that sample l belongs to class i. Then

$$\text{log loss}=-\frac{1}{n}\sum_{l=1}^{n}\sum_{i=1}^{C}y_{l,i}\,\log p_{l,i},$$

where we set

$$0\,\log 0:=0.$$

For binary classification with class labels 0 and 1 and predicted probabilities $p_{1,1},\ldots,p_{n,1}$ for samples to belong to class 1, log loss reduces to

$$\text{binary log loss}=-\frac{1}{n}\sum_{l=1}^{n}\bigl(y_l\,\log p_{l,1}+(1-y_l)\,\log(1-p_{l,1})\bigr).$$

Note: Nice little math exercise: Consider binary classification with class labels 0 and 1. Show that if predicted probabilities $p_{1,1},\ldots,p_{n,1}$ for class 1 result from a sigmoid transform of scores $z_1,\ldots,z_n$, then

$$\text{binary log loss}=\frac{1}{n}\sum_{l=1}^{n}\log\begin{cases}1+\mathrm{e}^{z_l}, & \text{if } y_l=0,\\ 1+\mathrm{e}^{-z_l}, & \text{if } y_l=1.\end{cases}$$

Scikit-Learn provides `log_loss`[401] in its `metrics` module.

Precision and Recall

Precision and recall are quality measures for binary classification problems. Almost always the two classes in a binary classification problem are not on a par. Instead they are based on answering the question whether a sample shows some prespecified property ('positive') or not ('negative'). Usage of precision and recall relies on this interpretation of binary classification.

Precision is the amount of correctly classified positives in the set of all samples classified as 'positive':

$$\text{precision}:=\frac{\text{number of samples correctly classified 'positive'}}{\text{total number of samples classified 'positive'}}.$$

In constrast, recall (or true positive rate (TPR) or sensitivity) counts the number of correctly classified positives in the set of all positives:

$$\text{recall}:=\frac{\text{number of samples correctly classified 'positive'}}{\text{total number of positive samples}}.$$

Both, precision and recall yield numbers in $[0,1]$; the higher the better the prediction quality. High precision tells us that the classifier does not label to many negatives as 'positive'. High recall tells us that the classifier is able to detect sufficiently many positives.

Example: If a doctor (the classifier) shall diagnose some disease, high precision means that the doctor has only very few wrong diagnoses (but may declare some ill clients as healthy). High recall means that the doctor identifies almost all ill persons (but may declare some healthy persons as ill).

Whether precision or recall are sensible measures for prediction qualitiy heavily depends on the context of the classification problem.

Example: We want to detect some rare event, say some rare disease, and we have 100 samples, 1 positive, 99 negative. If our classifier classifies all samples as positive, then precision is 0.01 and recall is 1. An always-negative

[401] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html

classifier would have undefined precision and recall 0. If our classifier labels by chance (50 samples labeled 'positive', 50 'negative'), then precision and recall are 0.02 and 1 if the positive sample is labeled correctly. If it is labled as 'negative', then precision and recall both are 0.

Scikit-Learn provides `precision_score`[402] and `recall_score`[403] in its `metrics` module.

F1-score

F1-score (or F-score) combines precision and recall by taking their harmonic mean[404]:

$$\text{F1-score} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

The F1-score is always in $[0, 1]$. If one of both, precision or recall, is close to 0, then the F1-score is close to 0. If both are close to 1, then the F1-score is close to 1. Thus, high F1-score guarantees good prediction quality (most 'positive' labels are correct and only few positive samples are missed).

Scikit-Learn provides `f1_score`[405] in its `metrics` module.

Area Under Receiver Operator Characteristic Curve (AUC)

AUC (area under curve) measures the area under a curve called reveiver operator characteristic (ROI) curve. This quality measure works for classification algorithms yielding a probability-like score with high values indicating positive samples and low values indicating negative samples. AUC does not depend on the threshold for converting scores to class labels. Instead, it expresses the **probability that a randomly drawn positive sample gets a higher score than a randomly drawn negative sample**.

Related Scikit-Learn methods are

- `roc_auc_score`[406]
- `roc_curve`[407]
- `RocCurveDisplay.from_predictions`[408]

Definition of AUC

Given true labels $y_1, \ldots, y_n$ ('positive' or 'negative') and corresponding scores $p_1, \ldots, p_n \in [0, 1]$ consider for each threshold $t \in (0, 1)$ the *false positive rate*

$$\text{FPR}(t) := \frac{\text{number of negative samples with } p_l > t}{\text{total number of negative samples}}$$

and the *true positive rate* (precision)

$$\text{TPR}(t) := \frac{\text{number of positive samples with } p_l > t}{\text{total number of positive samples}}.$$

Then the ROC 'curve' is the (finite) set of points

$$\{(FPR(t), TPR(t)) : t \in (0, 1)\}.$$

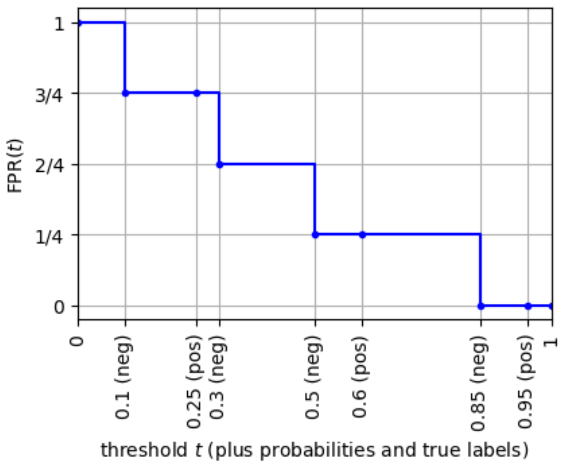
Often these points are connected by straight lines and the resulting curve is denoted as ROC curve. AUC is the area under that curve.

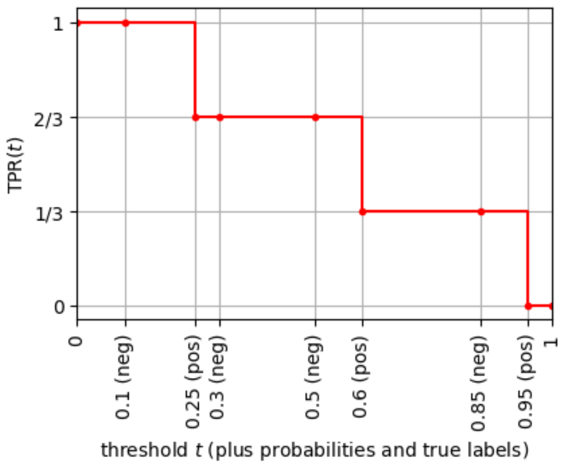
402 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
403 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
404 https://en.wikipedia.org/wiki/Harmonic_mean
405 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
406 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html
407 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
408 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html#sklearn.metrics.RocCurveDisplay.from_predictions

Example: Assume we have $n = 7$ samples and a trained binary classification model yields probability-like scores as follows:

true label	predicted probability
negative	0.1
negative	0.5
negative	0.3
negative	0.85
positive	0.6
positive	0.25
positive	0.95

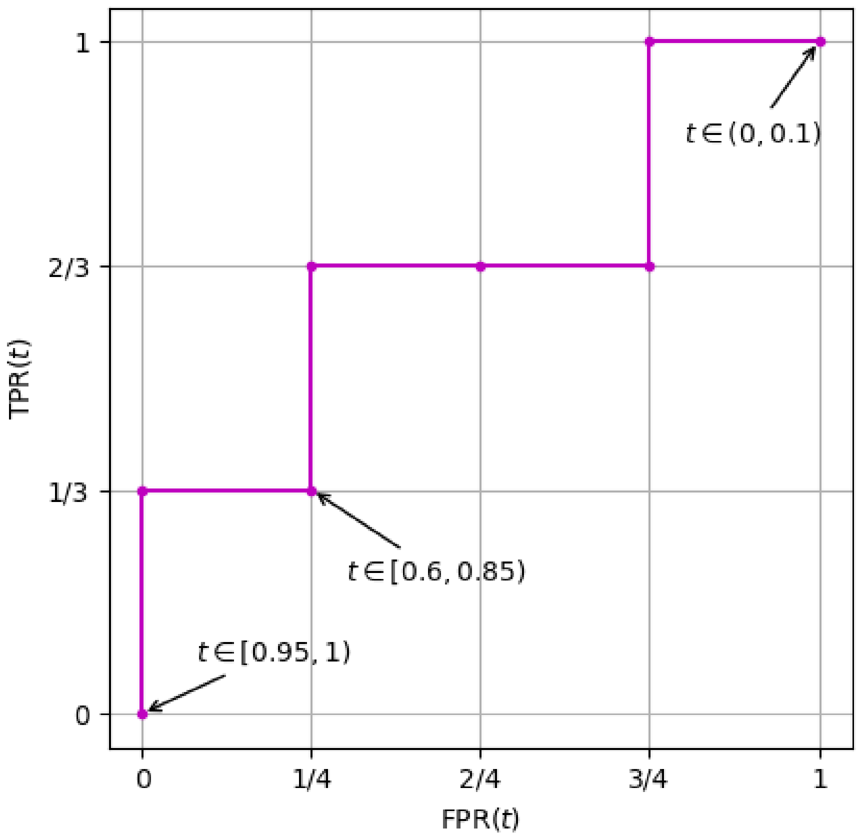
Then FPR and TPR look as follows:





FPR drops whenever t hits the predicted probability of a negative sample. Thus, we may identify steps in FPR (or areas between horizontal grid lines) with the negative samples. Analogously, we may identify steps in TPR with the positive samples.

The ROC curve is:



Now AUC is

$$\text{AUC} = 8 \cdot \frac{1}{3} \cdot \frac{1}{4} = \frac{2}{3}.$$

Properties of the ROC curve

From the definition of the ROC curve we easily deduce following properties:

- If there are no samples with equal scores, then the ROC curve is composed of horizontal and vertical line segments.
- It always connects $(0, 0)$ with $(1, 1)$ if scores are strictly between 0 and 1. If some scores are 0 or 1, then the endpoints are at least close to $(0, 0)$ and $(1, 1)$.
- The ROC curve is monotonically increasing.

Properties of AUC

From the definition of AUC we easily deduce following properties:

- AUC is always in $[0, 1]$.
- AUC is 1 if and only if the classification algorithm labels all samples correctly.
- AUC is 0 if and only if the classification algorithm labels all positive samples ‘negative’ and all negative samples ‘positive’.
- For a classification algorithm drawing scores at random equally distributed in $[0, 1]$ AUC is approximately 0.5 (the ROC curve is close to the diagonal).

Proof of AUC Interpretation

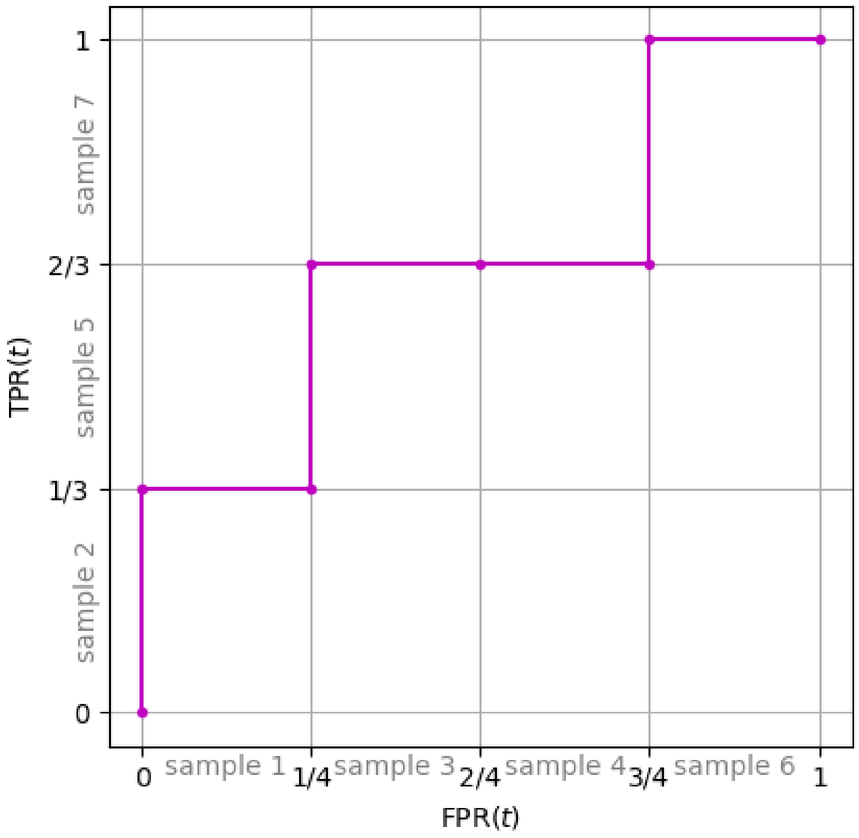
We stated above that AUC is the probability that a randomly chosen positive sample has higher score than a randomly chosen negative sample. To see this we look at the grid implicitly defined by the ROC curve. This idea is taken from The Probabilistic Interpretation of AUC[409], where missleadingly grid points are counted instead of boxes (areas between grid lines).

To avoid discussion of certain special cases we assume that all scores $p_1, \ldots, p_n$ are different and no score is 0 or 1. To simplify notation further we assume that samples are sorted by predicted probability, that is,

$$p_1 < p_2 < \cdots < p_n.$$

Thus, samples 2, 5, 7 are positive and samples 1, 3, 4, 6 are negative in the example above.

Given a positive sample and a negative sample we identify the positive sample with corresponding step in TPR and the negative sample with corresponding step in FPR. Thus, each positive-negative may identified with a box in the grid:



For some fixed positive-negative pair we want to show that the probability-like score of the positive sample is higher than the score of the negative sample if the box is below the curve and vice versa if the box is above the curve.

Assume the box is below the curve. Then we may go to the left and upwards until we are at a box which has two sides on the ROC curve. While going to the left FPR decreases, that is, score of corresponding negative samples increases. Going upwards TPR increases, that is, score of corresponding positive samples decreases. Thus, it remains to show that for a box with left and upper side on the curve corresponding positive sample has higher score than corresponding negative sample. But this can be seen from looking at a range of thresholds t covering scores of both samples. Increasing the threshold FPR drops before TPR drops (else lower and right side of the box would be on the curve). Thus, score of the negative sample has to be smaller than for the positive sample.

For boxes above the curve we may use analog reasoning to see that score of corresponding negative sample is above corresponding positive sample. The probability that a randomly chosen positive sample is scored higher than a ran-

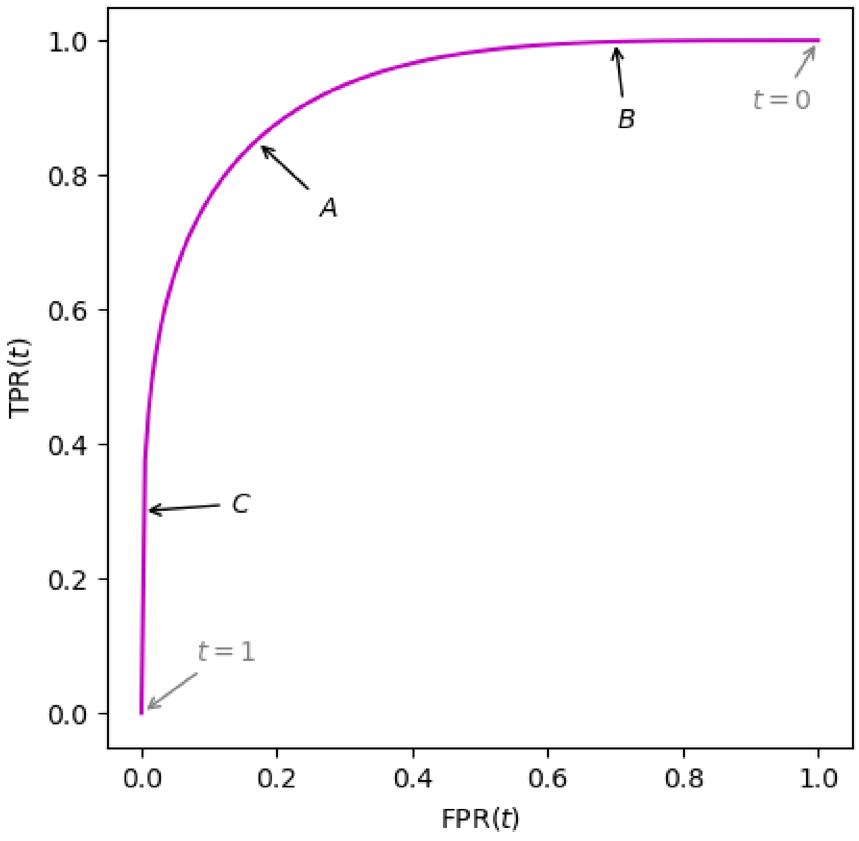
[409] http://madrury.github.io/jekyll/update/statistics/2017/06/21/auc-proof.html

domly chosen negative sample thus is the number of boxes below the curve divided by the total number of boxes or, equivalently (because all boxes have same area), the area under the curve divided by the total area, which is 1.

Choosing a Threshold

AUC does not depend on the threshold used for converting scores to labels. Thus, it cannot be used to find a good threshold but for comparing different classification methods. Finding a good threshold heavily depends on the underlying problem. We have to decide what is more important: high TPR or low FPR.

There exist several heuristic methods to derive a threshold form the ROC curve (not from AUC). One idea is to choose a threshhold where the ROC curve has high curvature or where the graph's slope is close to 1 (region A in figure below). Then the loss/gain in TPR is nearly the same as the loss/gain in FPR when modifying the threshold slightly. If, in contrast, we would choose a threshold where the curve is very steep (region B), then lowering the threshold would significantly increase TPR without increasing FPR too much. If the curve is very flat (region C), then larger thresholds would yield much smaller FPR while preserving TPR.



Another idea for choosing a threshold from the ROC curve is to choose the point (better: the corresponding theshold) closest to $(0, 1)$ (region A in figure). The point $(0, 1)$ corresponds to perfect classification (all positive samples labeled 'positive', no negative samples labeled 'positive'). Thus, getting as close as possible to this point is quite reasonable.

Binary Metrics for Multiclass Classification

Quality measures for binary classification (precision, recall, F1-score, AUC) can be extended to multiclass classification. For each class we look at the canonical binary problem (sample belongs to class or not) and calculate corresponing prediction quality for each such binary problem. The results then are averaged somehow.

Simplest way is to take the mean of all binary quality measures. If classes are very different in size, some weighting might be appropriate to lower influence of performance on very small classes. This approach is known as *(weighted) macro averaging*. To make this precise let $n_1, \ldots, n_C$ be the sizes of the the C classes and denote by $Q_1, \ldots, Q_C$ the binary metrics computed for corresponding C binary classification problems. Then

$$\text{weighted macro average} = \frac{n_1}{n}\,Q_1 + \cdots + \frac{n_C}{n}\,Q_C.$$

An alternative is *micro averaging*, which implements averaging over classes into the concrete structure of a binary metric. For precision, recall and F1-score micro averaging simply yields (unbalanced) accuracy. For AUC it's not clear how to define micro averaging. Thus, for us it doesn't add any value.

23.4 Scaling

Scaling of numeric data may influence results obtained from supervised learning methods. Often this influence is not obvious. The method itself might be sensitive to scaling, but more often scaling issues arise from underlying numerical algorithms (e.g., minimization procedures) for implementing a method.

We already met an example showing the importance of scaling in *Introductory Example (k-NN)* (page 317). More will follow when discussing more machine learning techniques. Here we have a look at two standard approaches to scaling: normalization and standardization.

23.4.1 Normalization

One common method for scaling data is to choose an interval, often $[0, 1]$, and to linearly transform values to fit this interval. If a feature's values are in the interval $[a, b]$, then transformation to $[0, 1]$ is done by

$$x_{\text{new}} = \frac{x_{\text{old}} - a}{b - a}.$$

Care has to be taken if data contains outliers: a very large value in the date would force values in the usual range to be mapped very close to zero.

Scikit-Learn offers normalization as `MinMaxScaler`[410] class in the `preprocessing` module. `MinMaxScaler` objects (like most of Scikit-Learn's objects) offer the three methods `fit`, `transform`, `fit_transform`. The latter is simple a convinience method which calls `fit` and then `transform`. The `fit` method looks at the passed data and determines its range. The `transform` method applies the actual transformation. Thus, if multiple data sets shall be transformed, call `fit` only once and then apply the transform to all data sets:

```python
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(X_train)     # get range (no transform here)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Alternatively:

```python
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[410] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

23.4.2 Standardization

More often than normalization the following approach is used for scaling data: First substract the mean, then divide by standard deviation. The result are features whose values have mean 0 and standard deviation 1. That is, values are centered at 0 and their mean deviation from 0 is 1.

Given values $x_1, \ldots, x_n$ the mean μ is

$$\mu = \frac{1}{n}\sum_{l=1}^{n} x_l$$

and standard deviation σ is

$$\sigma = \sqrt{\frac{1}{n}\sum_{l=1}^{n}(x_l - \mu)^2}.$$

Corresponding transform reads

$$x_{\text{new}} = \frac{x_{\text{old}} - \mu}{\sigma}.$$

From the mathematical statistics view we are slighlt imprecise here. Our μ is not the mean of the data's underlying probability distribution, but an estimate for it, known as *emperical* mean in statistics. Same holds for σ. But in addition, our estimate σ in some sence is worse than the usual emperical standard deviation in statistics, because it's not *unbiased* (see statistics lecture).

Scikit-Learn offers `StandardScaler`[411] in the `preprocessing` module for standardizing date. Usage is exactly the same as described above for normalization.

23.4.3 Scaling of Interdependent Features

In many cases features may be scaled independently (age and kilometers driven for cars, for instance). But in other cases information isn't solely contained in isolated features but differences between features may carry information, too. The most important example here are images. If we have a set of images and if we scale each pixel/feature independently, we may destroy information contained in the images.

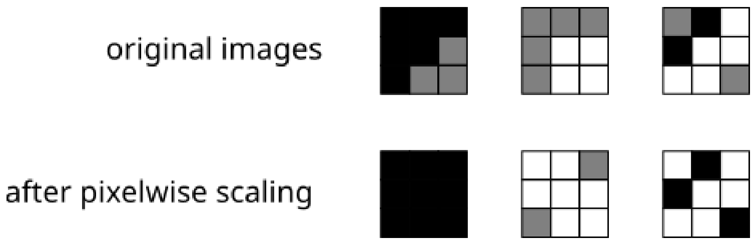


Fig. 23.5: Pixelwise normalization of images my destroy content. Pixels not covering the full color range will get modified while pixels with values in the full range will remain untouched.

Thus, for images and similar data, we have to apply same scaling to all pixels/features to keep information encoded as differences between features.

23.5 Feature Reduction

Feature reduction or *dimensionality reduction* aims at reducing the number of features in a data set without deterioration results of supervised learning methods. This saves resources (memory, CPU time), but also may help interpreting the data. We distinguish two classes of methods for feature reduction:

- feature selection,
- feature transform.

[411] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

A straight forward idea is feature selection or removal. One tries to identify relevant features and removes features of little or no relevance. Relevance here means influnce on the accuracy of the trained model. If we remove a feature, but the model trained on the reduced data set yields almost identical predictions as before, then the feature is considered irrelevant.

Feature transform refers to more complex methods which map the whole feature space to a lower dimensional space. Here, resulting new features lack easy interpretability because each contains information from many different original features.

Here we consider two methods for feature selection and one feature transforms method.

23.5.1 Manual Feature Removal

Exploratory data analysis gives a first impression of feature properties and relations between features. Observations made during this phase of doing data science should be used to reduce the number of features.

One observation frequenly made in raw data sets are features, that contain (almost) the same value in each sample. Such features can be removed without loss of information.

Another frequent observation are highly correlated features. Plotting one feature versus another we see whether there is a functional dependence between them or whether they are uncorrelated. In the first case the plot shows a line, in the second case we see a cloud of points.

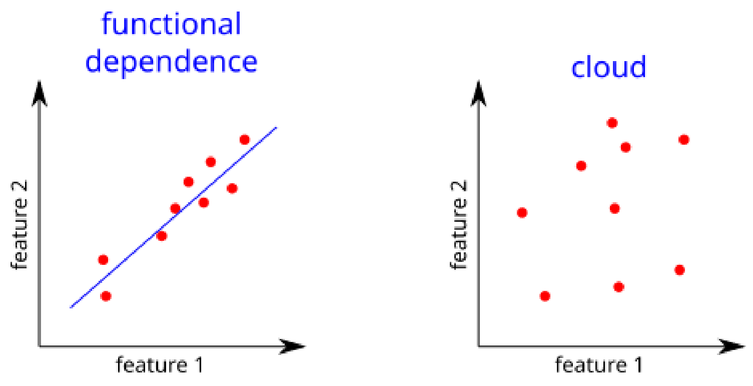


Fig. 23.6: Left image shows linear dependence, but functional dependency may also nonlinear (quadratic, for instance). On the right-hand side there's no obvious functional dependence.

In case of functional dependence one of the two considered features can be removed, because its values can be deduced from the remaining one. In other words, both features contain the same information, but in different representation.

There might also be features uncorrelated to the targets. If the plot of a feature against the targets shows a very homogeneous distribution of points all over the plotting area, then the feature is likely to have no influence on the targets. Thus, it can be removed. If in doubt, don't remove the feature because eyes may trick you about equal distribution of points.

23.5.2 Permutation Feature Importance

Given a trained model we ask for the importance of each feature. We could remove one feature, retrain the model and compare prediction quality to the original model. Doing this for all features we see which features can be removed without deteriorating results. The drawback of that approach is that we have to train as many models as we have features.

A similar idea, which does not require additional training, is calculating *permutation feature importance*. Given a data set we permutate the values of a fixed feature and look at corresponding predictions. The smaller the deviation from the original predictions the less important is the feature. So deviation in predictions after permutating feature values is a measure of feature importance.

Scikit-Learn supports permutation feature importance via `permutation_importance`[412] function in the `inspection` module. We have to pass the trained model and a data set. The object returned by the function has a member `importances` containing a NumPy array with the feature importances. By default the function calculates feature importances for several different permutations. Mean and standard deviation can be used to get a summary for all permuations. Both are accessible via the `importances_mean` and `importances_std` members, respectively.

23.5.3 Principal Component Analysis (PCA)

Principal component analysis is a powerful technique for deriving new features from existing ones (feature transformation). Original features may suffer from two problems:

- A feature might have low variance, that is, it contains few information. The extreme case is a feature with variance 0. Then all values of the feature are identical.
- Features might be correlated, that is, they contain redundant information. The extreme case is a feature whoes values are a fixed multiple of another feature's values.

Principal component analysis tackles both problems:

- PCA creates features with high variance and removes low variance features.
- Features created by PCA are uncorrelated.

We now go into the details of this important technique.

Prerequisites

PCA only works for numerical features and **all features should be centered**. More sloppily we may say that PCA assumes that the data set is a point cloud centered at the origin.

Let $x_1, \ldots, x_n$ be the feature vectors of the data set and assume we have m features. Writing the feature vectors as rows of a matrix we obtain a matrix X of size $n \times m$. Denote the columns of X by $\xi_1, \ldots, \xi_m$, that is, ξ_1 contains all values of the first feature, ξ_2 all values of the second feature, and so on; in formulas:

$$\xi_1 = \begin{bmatrix} x_1^{(1)} \\ \vdots \\ x_n^{(1)} \end{bmatrix}, \quad \ldots, \quad \xi_m = \begin{bmatrix} x_1^{(m)} \\ \vdots \\ x_n^{(m)} \end{bmatrix}.$$

Let's call $\xi_1, \ldots, \xi_m$ value vectors.

Linear Feature Transforms

Given numbers $a^{(1)}, \ldots, a^{(m)}$ the linear combination

$$a^{(1)}\,\xi_1 + \cdots + a^{(m)}\,\xi_m = X\,a$$

of the value vectors $\xi_1, \ldots, \xi_m$ can be considered the value vector of a new feature. Taking more such linear combinations we derive more and more features from the original ones.

Say we create $\tilde{m}$ new features with value vectors $\tilde{\xi}_1, \ldots, \tilde{\xi}_{\tilde{m}}$, then the corresponding matrix $\tilde{X}$ of size $n \times \tilde{m}$ may be written as

$$\tilde{X} = X\,A,$$

where each column of the *transform matrix* A contains the coefficients of the linear combination yielding the corresponding new feature. The first column of A contains the coefficients for $\tilde{\xi}_1$, for instance. Dimensions of A are $m \times \tilde{m}$. This can be seen by carrying out usual matrix multiplication: the first column of $X\,A$ is X times the first column of A and so on.

[412] https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html

If the transform matrix A is invertible, we may write

$$\tilde{X}\,A^{-1} = X.$$

That is, we may switch back and forth between both feature sets as we like. Obviously, invertibility of A requires at least $m = \tilde{m}$.

If the transform matrix A is invertible, then both feature sets contain identical information, but in different representation. There exist several approaches for contructing transform matrices. PCA is only one. Like PCA, most approaches aim at removing features carrying only few information. Thus, A will not be invertible.

Variance Maximization

By assumption the original value vectors $\xi_1, \ldots, \xi_m$ are centered (mean 0). One easily verifies, that derived value vectors then are centered (mean 0), too.

The idea of PCA is to find coefficient vectors a such that the corresponding new value vector $X\,a$ has maximum variance with respect to all vectors a of length 1. Considering other lengths, too, would add no value. Looking at length 2, for instance, would yield a factor 4 in all variances (factor 2 for standard deviation), but the maximizing vector a would have the same direction as for length 1.

Since $X\,a$ is centered, its variance is simply the sum of the squares of its components (up to constant factor); in other words: its squared length. Taking the square root does not change the maximizer. Thus, maximizing variance boils down to solving

$$|X\,a| \to \max_{|a|=1}.$$

Some deeper mathematics reveals that maximum variance is given by the largest eigenvalue of the matrix $X^{\mathrm{T}}\,X$ and that the maximizer a^* is a corresponding eigenvector. The square matrix $X^{\mathrm{T}}\,X$ is known as *covariance matrix* in statistics.

The optimal coefficient vector a^* yields a derived feature which contains a maximum of information compared to all other derivable features. Such a feature is called a *principal component* of the original features.

Uncorrelated Features

Given the variance maximizing coefficient vector a^* we would like to restrict our attention to only those derivable features which are completely uncorrelated to the new feature described by a^*. In this reduced set of features we then may look for maximum variance again. Repeating this process would yield a list of features with descending variance (list of *principal components*) and all features in the list would be mutually uncorrelated.

The only question to answer is: How to restrict the set of features to avoid any correlation with already extracted features? The answer is: We have to restrict all considerations to coefficient vectors yielding features orthogonal to all previously found variance maximizing features. Two derived features are uncorrelated if corresponding value vectors $\tilde{\xi}_i$ and $\tilde{\xi}_j$ have zero covariance. Because both are centered, covariance is $\tilde{\xi}_i^{\mathrm{T}}\,\tilde{\xi}_j$. Zero covariance thus is nothing else than orthogonality.

Assume that step by step we have found k variance maximizing coefficient vectors $a_1^*, \ldots, a_k^*$. Then we restrict attention to vectors a for which $X\,a$ is orthorgonal to all $X\,a_1^*, \ldots, X\,a_k^*$.

Starting with $k = 1$ we go on until the space of derivable features (that is, the set of all linear combinations of original value vectors) is exhausted. We end up with at most m new features. If we obtain less features, then the original feature set contained redundant information, which now got removed.

Geometric Interpretation

Although we derived the PCA procedure from statistical considerations PCA has a nice geometric interpretation.

A value vector $\xi = X\,a$ of a derived feature contains the inner products of all samples with the coefficient vector a. The coefficient vector a 'lives' in the same space as the samples: $\mathbb{R}^m$. Thus, maximizing $X\,a$ can be regarded as looking for a direction (1d subspace) such that the orthogonal projections of all samples onto that subspace have maximum range.

To get the second principal component we project all samples onto the $m-1$ dimensional subspace orthogonal to the first principal component and then look for the projection range maximizing direction inside this subspace.

Following this procedure until the space is exhausted we obtain a set of orthogonal unit vectors in the original feature space. In other words, the $a_1^*, a_2^*, \ldots$ define a new coordinate system. The transformed samples $\tilde{x}_1, \ldots, \tilde{x}_n$ in $\tilde{X} = X\,A$ simply are the original samples but expressed with respect to the new coordinate system.

Removing Low Variance Features

PCA yields a list of uncorrelated centered features and their variances. Thus, me may remove low variance features from the list to reduce resource consumption when applying machine learning algorithms.

The remaining coefficient vectors in the list make up the transform matrix A introduced above.

Relation to Singular Value Decomposition (SVD)

PCA looks for eigenvalues and eigenvectors of $X^{\mathrm{T}}\,X$. This matrix is square, symmetric, and positive semidefinite. Thus, all eigenvalues are nonnegative real numbers.

The matrix X itself lacks all those nice properties. Nonetheless, there is a transform very similar to PCA which applies directly to X, the singular value decomposition. SVD yields two orthonormal systems $\{u_1, \ldots, u_n\}$ and $\{v_1, \ldots, v_m\}$ and a list of nonnegative real numbers $s_1, \ldots, s_m$ such that

$$X = U\,S\,V^{\mathrm{T}}.$$

Here S is the 'diagonal' matrix of size $n \times m$ with $s_1, \ldots, s_m$ on its diagonal, U and V are the matrices having $u_1, \ldots, u_n$ and $v_1, \ldots, v_m$ as columns, respectively.

We have the following relations to PCA:

- $v_\kappa = a_\kappa^*$ (normalized eigenvectors of $X^{\mathrm{T}}\,X$),
- $u_\kappa = \frac{X\,a_\kappa^*}{|X\,a_\kappa^*|}$ (normalized principal components or eigenvectors of $X\,X^{\mathrm{T}}$),
- $s_\kappa = |X\,a_\kappa^*|$ (length of principal components or square roots of eigenvalues of $X^{\mathrm{T}}\,X$ and $X\,X^{\mathrm{T}}$).

Note that U and V are orthonormal matrices, implying $U^{-1} = U^{\mathrm{T}}$ and $V^{-1} = V^{\mathrm{T}}$. Thus,

$$A = V, \qquad \tilde{X} = X\,A = U\,S.$$

Some of the $s_1, \ldots, s_m$ may be zero, indicating that the number of features can be reduced without loss of information. If some of the $s_1, \ldots, s_m$ are very small, then setting them to zero reduces the number of features while neglecting only very few information (*truncated singular value decomposition or TSVD*).

PCA with Scikit-Learn

Scikit-Learn implements a PCA[413] class in its `decomposition` module. The constructor takes the number `n_components` of principal components to keep (`None` for all). The `fit` method computes the principle components and `transform` yields transformed data.

After fitting the PCA object offers principal components and corresponding variances as member variables `components_` and `explained_variance_`.

Scikit-Learn's PCA automatically centers the input data and stores each component's mean in the `PCA` object's `mean_` member.

We start an illustrating example with all necessary imports and a function for plotting a point cloud.

```python
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import sklearn.preprocessing as preprocessing
import sklearn.decomposition as decomposition

from numpy.random import default_rng
rng = default_rng(0)

def plot_data(X, xlabel, ylabel, zlabel):
    ''' Scatter plot of data with properly configured axes. '''

    fig = go.Figure()
    fig.layout.width = 800
    fig.layout.height = 600

    fig.add_trace(go.Scatter3d(
        x=X[:, 0], y=X[:, 1], z=X[:, 2],
        mode='markers',
        marker={'color': '#0000ff', 'size': 1}
    ))

    data_min = X.min()
    data_max = X.max()
    fig.update_scenes(
        xaxis_title_text=xlabel,
        yaxis_title_text=ylabel,
        zaxis_title_text=zlabel,
        xaxis_range=[data_min, data_max],
        yaxis_range=[data_min, data_max],
        zaxis_range=[data_min, data_max]
    )
    fig.update_layout(scene_aspectmode='cube', showlegend=False)

    return fig
```

We use simulated data with 3 features. For simple simulation we generate data following a multivariate normal distribution.

```python
n = 500    # number of data points to generate

# generate data
X = rng.multivariate_normal([1, 2, 3], [[1, 0.2, 0.3], [0.2, 2, -0.1], [0.3, -0.1,
↪ 0.2]], n)

# plot data
```

(continues on next page)

413 https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

(continued from previous page)

```
fig = plot_data(X, 'feature 1', 'feature 2', 'feature 3')
fig.show()
```

```
<IPython.core.display.HTML object>
```

Now we do a full PCA and plot the principal components. That is, we keep all principle components. In this case we do not have to pass an argument to the `PCA` constructor. Note that we multiply the length of the principle components by 3 for better visibility.

```
pca = decomposition.PCA()
pca.fit(X)

# plot
fig = plot_data(X, 'feature 1', 'feature 2', 'feature 3')

# plot principal components
vec = np.stack((pca.mean_, np.empty(3)), axis=1)
for i in range(0, 3):
    vec[:, 1] = vec[:, 0] + 3 * np.sqrt(pca.explained_variance_[i]) * pca.
↪components_[i, :]
    fig.add_trace(go.Scatter3d(
        x=vec[0, :], y=vec[1, :], z=vec[2, :],
        mode='lines',
        line={'color': '#ff0000', 'width': 5}
    ))

fig.show()
```

```
<IPython.core.display.HTML object>
```

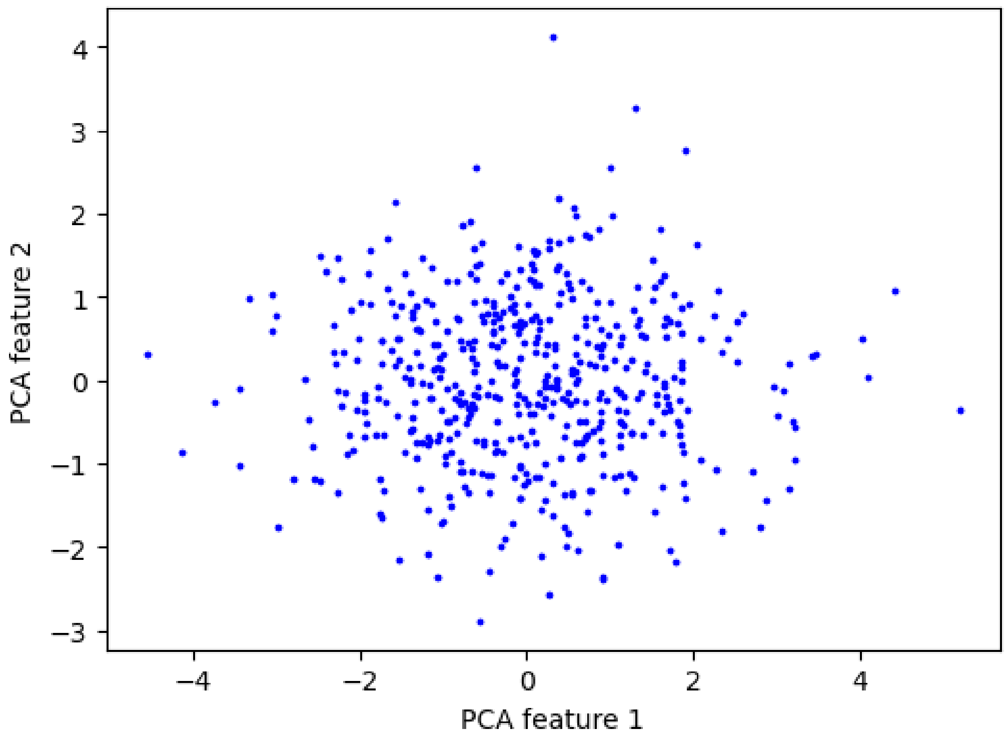
It remains to transform the data according to the principal components.

```
X_transformed = pca.transform(X)

# plot
fig = plot_data(X_transformed, 'PCA feature 1', 'PCA feature 2', 'PCA feature 3')

# plot principal components
vec = np.stack((np.zeros(3), np.empty(3)), axis=1)
for i in range(0, 3):
    vec[:, 1] = 0
    vec[i, 1] = 3 * np.sqrt(pca.explained_variance_[i])
    fig.add_trace(go.Scatter3d(
        x=vec[0, :], y=vec[1, :], z=vec[2, :],
        mode='lines',
        line={'color': '#ff0000', 'width': 5}
    ))

fig.show()
```

```
<IPython.core.display.HTML object>
```

If we want to reduce the number of features we may omit the last component (the one with smallest variance), resulting in a two-dimensional data set. To make `PCA.transform` do this for us we have to restart PCA with `n_components=2`.

```
pca = decomposition.PCA(n_components=2)
pca.fit(X)

X_transformed = pca.transform(X)

# plot transformed data
fig, ax = plt.subplots(figsize=(6, 6))
ax.scatter(X_transformed[:, 0], X_transformed[:, 1], c='#0000ff', s=3)
ax.set_xlabel('PCA feature 1')
ax.set_ylabel('PCA feature 2')
ax.set_aspect('equal')

plt.show()
```



23.5.4 Nonlinear Feature Reduction Methods

PCA is a linear method, that is, we apply a linear transform to the data. There exist many nonlinear methods for feature reduction. Such nonlinear methods are considered a separate branch of unsupervised machine learning and will be considered later on.

23.6 Hyperparameter Tuning

Hyperparameters are parameters of a model which are not fit to the data. Instead, they are choosen in advance, mainly to control model complexity. An example is the number of neighbors considered in k-NN method. But preprocessing steps may depend on parameters, too. Remember the number of components to keep in PCA, for instance. This is a hyperparameter of the overall model, too. In this broader sense a model is an algorithm which takes raw data and yields predictions.

Choice of hyperparameters is difficult and heavily builds upon experience of the data scientist. From the computational point of view it's very difficult to find optimal hyperparameters. Corresponding optimization problems are non-differentiable and, thus, not accessible to standard optimization procedures.

We will consider techniques for automatically choosing hyperparameters below. But getting some experience in manually choosing hyperparameters is an indispensable skill of data scientists.

23.6.1 Pipelines

Scikit-Learn allows to combine several processing steps into one estimator object. Thus, applying a chain of transformations and fitting procedures to several data sets becomes very simple. This functionality is provided by the `Pipeline` class[414] of the `sklearn.pipeline` module.

Hint: Pipelines aren't needed for hyperparameter optimization. But they simplify code by encapsulating all processing steps and their parameters in one Python object. Thus, hyperparameter optimization algorithms do not have to cope with several and different objects defining the overall model to be optimized.

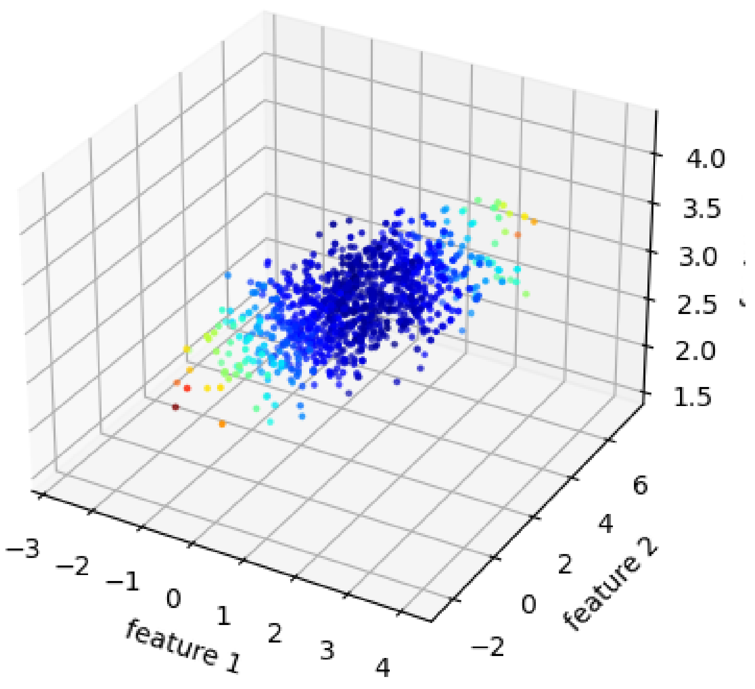
Consider standardization followed by PCA and k-NN regression for the following example with synthetic data. We have three features and want to predict some quantity depending an these feature.

First the imports and creation of synthetic data.

```
import numpy as np
from numpy.random import default_rng
rng = default_rng(0)

import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d as plt3d

import sklearn.preprocessing as preprocessing
import sklearn.decomposition as decomposition
import sklearn.neighbors as neighbors
import sklearn.pipeline as pipeline
```

```
n = 1000    # number of data points to generate

# generate data
X = rng.multivariate_normal([1, 2, 3], [[1, 0.2, 0.3], [0.2, 2, -0.1], [0.3, -0.1,
↪ 0.2]], n)
y = (0.89 * X[:, 0] + 0.78 * X[:, 1] + 0.84 * X[:, 2] - 5.34) ** 2 + 0.5 * rng.
↪normal(size=n)

# plot data (color encodes y)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(X[:, 0], X[:, 1], X[:, 2], s=3, c=y, cmap='jet')
ax.set_xlabel('feature 1')
ax.set_ylabel('feature 2')
ax.set_zlabel('feature 3')
plt.show()
```

[414] https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html



Now we create a pipeline containing all processing steps. The `Pipeline` constructor takes a list of tuples. Each tuple consists of a string and a Scikit-Learn object. The string will be used to refer to parts of the pipeline later on. The Scikit-Learn objects have to provide `fit` and `transform` methods, where the whole pipeline provides the same methods as the last object in the pipeline. Thus, if the last object provides a `predict` method, then the whole pipeline can be used for prediction.

```
steps = [('standardize', preprocessing.StandardScaler()),
         ('pca', decomposition.PCA(n_components=2)),
         ('knn', neighbors.KNeighborsRegressor(n_neighbors=5))]

pipe = pipeline.Pipeline(steps)

pipe.fit(X, y)
pipe.predict([[0, 0, 3]])
```

```
array([6.71318598])
```

Calling `fit` of the pipeline object calls `fit` and `transform` for the first object in the pipeling, then for the second, and so on. Parameters of the processing steps can be set when creating the objects or afterwards by calling `Pipeline.set_params()`. This method takes keyword arguments of the form `step__param`, where `step` is a step's name and `param` is the name of the parameter of the corresponding object.

```
pipe.set_params(knn__n_neighbors=10)
pipe.fit(X, y)
pipe.predict([[0, 0, 3]])
```

```
array([6.96541741])
```

All objects contained in a pipeline are accessible through the `Pipeline.steps` object. It's a list of tuples, each containing a step's name and the corresponding object. This way all parameters of all processing steps are available.

```
pipe.steps[1][1].components_[0, :]    # largest principal component
```

```
array([-0.70940987, -0.02120454, -0.70447712])
```

23.6.2 Grid Search

The simplest but often also the only applicable form of hyperparameter optimization is grid search: for each hyperparameter we specify finitely many values and train a model for each combination of these values. Then the best model is chosen. To judge about a model's quality we need a scoring function, for regression problems usually the mean squared error on a validation set.

It's important to split the data into three sets: one for training the models, one for validating (that is, scoring) the models, and one for testing the final model's prediction quality on data not involved in the training and model selection process.

Splitting into training and validation data has not to be fixed. After having a closer look at grid search we will discuss advanced techniques for selecting training and validation data.

We start grid search with data splitting: 50% for training, 30% for validation, 20% for testing.

```
import sklearn.model_selection as model_selection

X_train_val, X_test, y_train_val, y_test \
    = model_selection.train_test_split(X, y, test_size=2/10, random_state=0)

X_train, X_val, y_train, y_val \
    = model_selection.train_test_split(X_train_val, y_train_val, test_size=3/8,␣
↪random_state=0)

print(y_train.size, y_val.size, y_test.size)
```

```
500 300 200
```

Now steps are as follows:

- specify grid for each hyperparameter,
- generate all combinations of parameter values with `ParameterGrid`[415] from Scikit-Learn's `model_selection` module,
- loop over all combinations and find best model.

```
import sklearn.metrics as metrics

param_grid = {'pca__n_components': [1, 2, 3],
              'knn__n_neighbors': range(1, 11)}

best_err = None
best_params = None

for params in model_selection.ParameterGrid(param_grid):

    pipe.set_params(**params)
    pipe.fit(X_train, y_train)

    y_val_pred = pipe.predict(X_val)
    err = metrics.mean_squared_error(y_val_pred, y_val)

    if (best_err == None) or (err < best_err):
        best_err = err
```

(continues on next page)

[415] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.ParameterGrid.html

(continued from previous page)

```
        best_params = params

print(best_params)
```

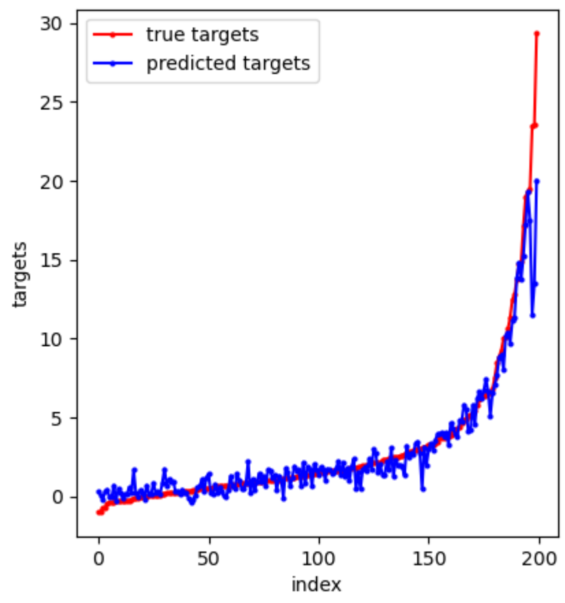
```
{'knn__n_neighbors': 4, 'pca__n_components': 2}
```

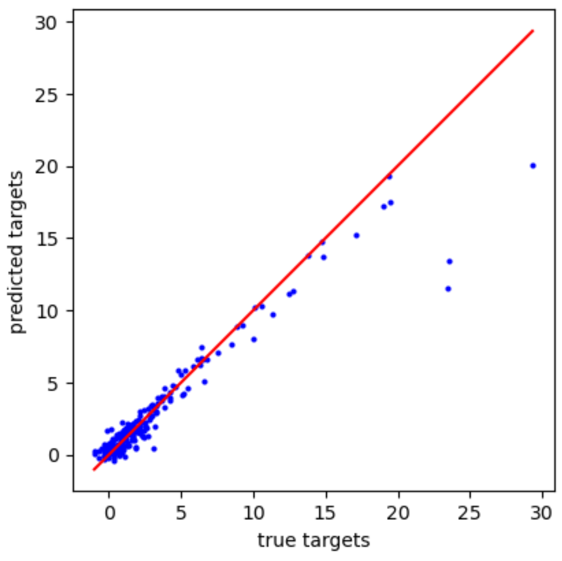
Now we may fit the final model. Because we do not have to adjust hyperparameters anymore, we may **use both training and validation data for fitting**.

Metrics and plots demonstrate the performance of the model.

```
pipe.set_params(**best_params)
pipe.fit(X_train_val, y_train_val)

y_test_pred = pipe.predict(X_test)
mse = metrics.mean_squared_error(y_test_pred, y_test)

print('root of MSE:  ', np.sqrt(mse))
print('standard deviation of true targets:  ', np.std(y_test))

fig, (ax_left, ax_right) = plt.subplots(1, 2, figsize=(10, 5))

s = y_test.argsort()    # sort for getting a continuous line
ax_left.plot(y_test[s], '-or', markersize=2, label='true targets')
ax_left.plot(y_test_pred[s], '-ob', markersize=2, label='predicted targets')
ax_left.set_xlabel('index')
ax_left.set_ylabel('targets')
ax_left.legend()

ax_right.plot(y_test, y_test_pred, 'ob', markersize=2)
min_value = np.minimum(y_test.min(), y_test_pred.min())
max_value = np.maximum(y_test.max(), y_test_pred.max())
ax_right.plot([min_value, max_value], [min_value, max_value], '-r')
ax_right.set_aspect('equal')
ax_right.set_xlabel('true targets')
ax_right.set_ylabel('predicted targets')

plt.show()
```

```
root of MSE:   1.4555318931691366
standard deviation of true targets:   4.641821625794403
```





23.6.3 Cross Validation

Especially for small data sets different splits into training and validation sets may yield different results in hyperparameter optimization. To avoid influence of the splitting process, we may use many different splits and calculate their mean score. *Cross validation* implements this idea.

Data available for training and validation is split into ν more or less equally sized subsets. Then ν different training validation splits are considered: one of the ν subsets is used for validation and the other $\nu - 1$ comprise the test set. For each such split we train the model and calculate the score on the validation set. Finally, we calculate the mean of all ν scores. This way all available data is used for training as well as for validation.

Scikit-Learn implements grid search with cross validation in the `GridSearchCV` class[416]. The constructor takes a Scikit-Learn object (a pipeline, for instance) and a parameter grid as arguments. `GridSearchCV` objects then provide a `fit` method for doing the grid search and a `predict` method for getting predictions from the optimal model. The constructur accepts several other important arguments:

- `scoring`: How to calculate the score for each model? The higher the score, the better the model. We may pass a predefined string, `'neg_mean_squared_error'`, for instance. See list of predefined scoring parameters[417].
- `n_jobs`: How many processors to use? If Scikit-Learn shall use all available processors, pass `-1`.
- `cv`: How many subsets to use?. Typical choice are $2, \ldots, 10$.

```
param_grid = {'pca__n_components': [1, 2, 3],
              'knn__n_neighbors': range(1, 11)}

gs = model_selection.GridSearchCV(pipe, param_grid, scoring='neg_mean_squared_
↪error', cv=5, n_jobs=-1)
gs.fit(X_train_val, y_train_val)

y_test_pred = gs.predict(X_test)
mse = metrics.mean_squared_error(y_test_pred, y_test)

print(gs.best_params_)
print()
print('root of MSE:  ', np.sqrt(mse))
print('standard deviation of true targets:  ', np.std(y_test))
```

[416] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
[417] https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

```
{'knn__n_neighbors': 3, 'pca__n_components': 2}

root of MSE:   1.4311176903230793
standard deviation of true targets:   4.641821625794403
```

If there are only very few samples for training and validation choosing as many subsets as samples are available can be useful. Then the validation set consists of only one sample. This approach is known as `leave one out (LOO) cross validation`.

23.6.4 Randomized Search

If training of a model takes much time, then searching the whole parameter space for optimal hyperparameters via grid search is not feasible. In such cases choosing hyperparameters randomly may yield good results in less time. Scikit-Learn provides random search functionality with `RandomizedSearchCV`[418].

23.6.5 Bias versus Variance

There are two main sources for bad model performance:

- The model is too simple to properly represent the data.
- The model is overfitted to the training data.

In the first case we say that the model is too *biased*. That is, it contains assumptions about the data the data does not satisfy. Approximating nonlinear data by a linear function is a typical example.

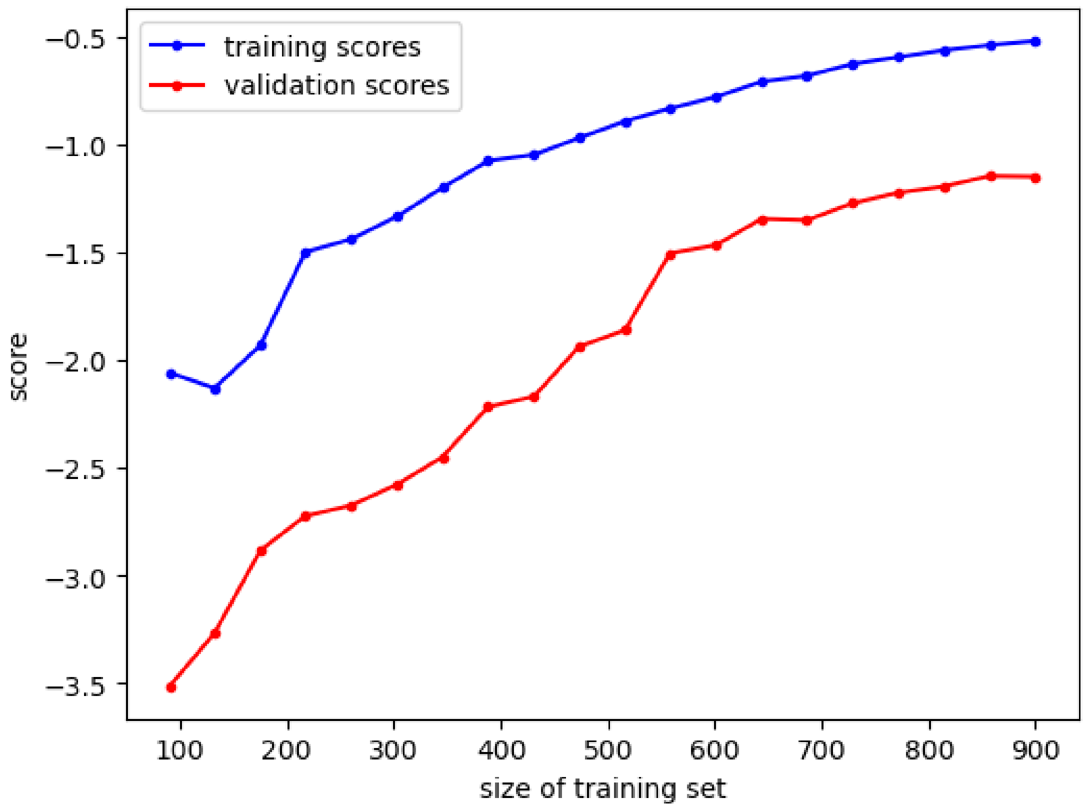
The second situation stems from too complex models. The model itself imposes only very few assumptions on the data and gathers all its information from the training data. Here we say that the model has high *variance*. High variance results in low *generalization power*. That is, the model performance is bad on data not included in the training procedure.

To find good models we have to look for a compromise between bias and variance of a model. This is what we do when tuning hyperparameters. To reduce variance, obtaining more training data is a choice, too. But often getting more data is impossible or at least expensive.

Scikit-Learn has the `learning_curve`[419] function to plot the dependence between size of training data set and prediction errors. The function trains and scores a model for different amounts of training data. For scoring cross validation is used.

```
pipe.set_params(pca__n_components=3, knn__n_neighbors=3)

train_sizes, train_scores, val_scores \
    = model_selection.learning_curve(pipe, X, y, train_sizes=np.linspace(0.1, 1,␣
 ↪20), cv=10,
                                     scoring='neg_mean_squared_error')

fig, ax = plt.subplots()
ax.plot(train_sizes, np.mean(train_scores, axis=1), '-ob', markersize=3, label=
 ↪'training scores')
ax.plot(train_sizes, np.mean(val_scores, axis=1), '-or', markersize=3, label=
 ↪'validation scores')
ax.legend()
ax.set_xlabel('size of training set')
ax.set_ylabel('score')
plt.show()
```

418 https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
419 https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html



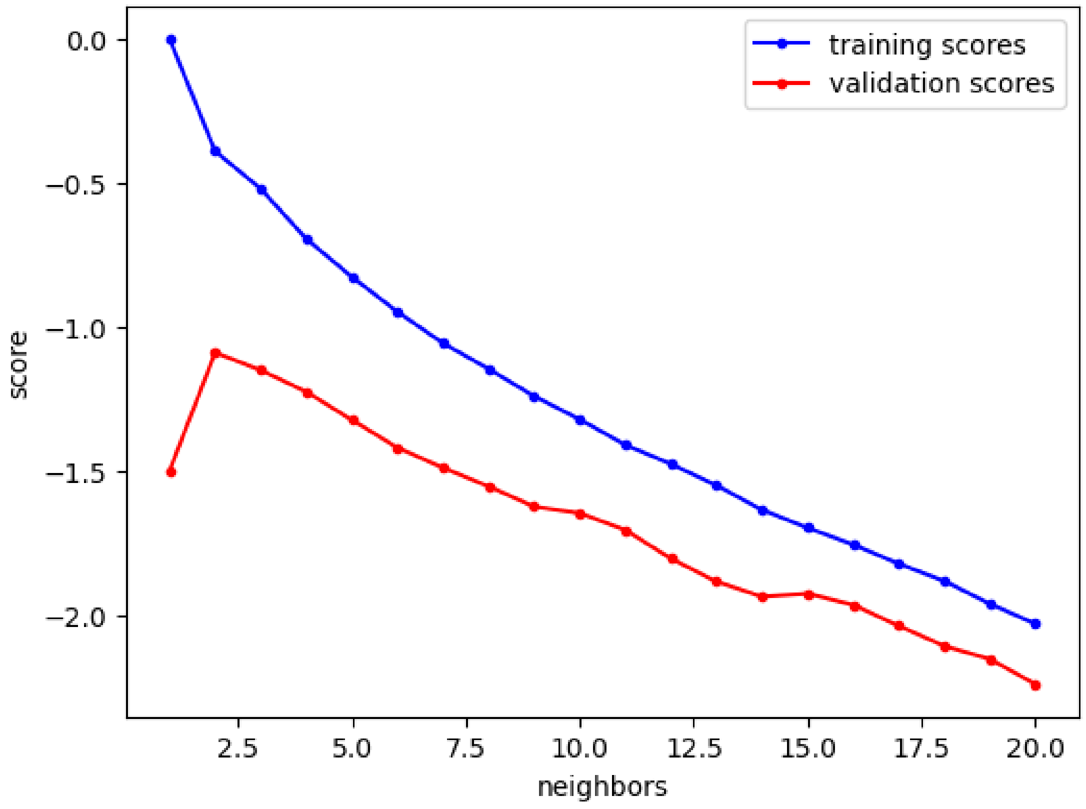
We see that 600 training samples suffice to train the model. Using more samples increases performance only slightly. Thus, it is very unlikely that collecting even more data would improve our model.

Similarly to the learning curve we may plot training and validation error depending on a hyperparameter. Usually, on the one end of the parameter scale the model is too simple (thus, high training and validation error), on the other end the model is too complex and overfits the training data (thus, low training error, high validation error). From the so called *validation plot* we may find a good hyperparameter between both extreme cases.

Scikit-Learn provides the `validation_curve`[420] function for this purpose. Usage is very similar to `learning_curve`.

```
pipe.set_params(pca__n_components=3, knn__n_neighbors=3)

param_range = range(1, 21)
train_scores, val_scores \
    = model_selection.validation_curve(pipe, X, y, param_name='knn__n_neighbors',
                                       param_range=param_range, cv=10,
                                       scoring='neg_mean_squared_error')

fig, ax = plt.subplots()
ax.plot(param_range, np.mean(train_scores, axis=1), '-ob', markersize=3, label=
 ↪'training scores')
ax.plot(param_range, np.mean(val_scores, axis=1), '-or', markersize=3, label=
 ↪'validation scores')
ax.legend()
ax.set_xlabel('neighbors')
ax.set_ylabel('score')
plt.show()
```

[420] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.validation_curve.html



Obviously, number of neighbors should be 2 to get a relatively good score (score close to 0 means that the mean squared error is close to 0). More neighbors reduce model complexity. Thus, the model isn't able to fit training (and validation) data. With only one neighbor the model tends to overfit training data.

CHAPTER

TWENTYFOUR

LINEAR REGRESSION

Linear regression denotes a class of relatively simple yet powerful regression methods. In this chapter we study the basics as well as advanced techniques and special cases.

Related projects:

- *House Prices GUI* (page 971)

24.1 Approach

Linear regression is a classical method in mathematics for constructing functions which are somehow close to a given set of points. In Data Science linear regression does a pretty good job, thought it's relatively simple.

24.1.1 Linear Regression with Linear Functions

The Principle

We first consider the simplest regression model. The parameter dependent hypothesis is of the form

$$f_{\text{approx}}(x)=a_0+a_1\,x^{(1)}+a_2\,x^{(2)}+\cdots+a_m\,x^{(m)}=a_0+\sum_{k=1}^{m}a_k\,x^{(k)},$$

where $x=\left(x^{(1)},\ldots,x^{(m)}\right)$ is a feature vector and $a_0,a_1,\ldots,a_m$ are the parameters of the hypothesis f_{approx}.

Given n training samples $(x_1,y_1),\ldots,(x_n,y_n)$ we want to choose $a_0,a_1,\ldots,a_m$ such that

$$f_{\text{approx}}(x_l)\approx y_l\qquad\text{for } l=1,\ldots,n.$$

To achive this we solve the minimization problem

$$\boxed{\frac{1}{n}\sum_{l=1}^{n}\left(f_{\text{approx}}(x_l)-y_l\right)^2\to\min_{a_0,\ldots,a_m}}.$$

The function g defined by

$$g(u,v)=\frac{1}{n}\sum_{l=1}^{n}\left(u_l-v_l\right)^2$$

is a so called *loss function* expressing the distance between two vectors $u=(u_1,\ldots,u_n)$ and $v=(v_1,\ldots,v_n)$. There are several other loss functions in machine learning like *mean absolute error* and *Huber loss*, see *Quality Measures* (page 328). For the moment we content ourselves with the simplest one, the *mean squared error*.

Note that we cannot expect $f_{\text{approx}}(x_l)=y_l$, because our model, that is, our assumption of a linear function, is likely to be too simplistic. In addition, observations y_l in regression problems often are corrupted by noise.

Solving the Minimization Problem Analytically

The minimization problem

$$h(a_0,\ldots,a_m):=\frac{1}{n}\sum_{l=1}^{n}\left(f_{\text{approx}}(x_l)-y_l\right)^2=\frac{1}{n}\sum_{l=1}^{n}\left(a_0+\sum_{k=1}^{m}a_k\,x_l^{(k)}-y_l\right)^2\to\min_{a_1,\ldots,a_m}$$

is known to have only global minima (no other stationary points), which can be found via differential calculus. We simply have to find $a_0,\ldots,a_m$ where the gradient of h is the zero vector. To keep formulas as simple as possible we introduce an additional artificial zeroth feature $x_l^{(0)}:=1$ for $l=1,\ldots,n$. This allows to write

$$h(a_0,\ldots,a_m)=\frac{1}{n}\sum_{l=1}^{n}\left(\sum_{k=0}^{m}a_k\,x_l^{(k)}-y_l\right)^2.$$

Taking the derivative with respect to a_i for $i=0,1,\ldots,m$ we obtain

$$\begin{aligned}\frac{\partial}{\partial a_i}h(a_1,\ldots,a_m)&=\frac{1}{n}\sum_{l=1}^{n}2\left(\sum_{k=0}^{m}a_k\,x_l^{(k)}-y_l\right)x_l^{(i)}\\&=\frac{2}{n}\sum_{l=1}^{n}\sum_{k=0}^{m}a_k\,x_l^{(k)}\,x_l^{(i)}-\frac{2}{n}\sum_{l=1}^{n}y_l\,x_l^{(i)}\\&=\frac{2}{n}\sum_{k=0}^{m}a_k\sum_{l=1}^{n}x_l^{(k)}\,x_l^{(i)}-\frac{2}{n}\sum_{l=1}^{n}y_l\,x_l^{(i)}.\end{aligned}$$

Thus, the gradient is zero if and only if

$$\sum_{k=0}^{m}a_k\sum_{l=1}^{n}x_l^{(k)}\,x_l^{(i)}=\sum_{l=1}^{n}y_l\,x_l^{(i)}\qquad\text{for } i=0,1,\ldots,m.$$

This is a system of linear equations for the unknowns $a_0,a_1,\ldots,a_m$. Sometimes these equations are called *normal equations* of the minimization problem. Taking the feature vectors $x_1,\ldots,x_n$ as rows of a matrix $B\in\mathbb{R}^{m\times n}$ (including the artificial zeroth feature 1), that is,

$$B:=\begin{bmatrix}1 & x_1^{(1)} & \cdots & x_1^{(m)}\\ \vdots & \vdots & & \vdots\\ 1 & x_n^{(1)} & \cdots & x_n^{(m)}\end{bmatrix}$$

we see that the gradient of h is zero if and only if

$$\boxed{B^{\mathrm{T}}\,B\,a=B^{\mathrm{T}}\,y.}$$

Here a and y are the vectors of the unknowns and of the labels, respectively. Note that the system matrix $B^{\mathrm{T}}\,B$ is a square matrix of size $m+1$ with m being the number of features in our data.

Example: Used Cars Prices

We look at a data set containing information about used cars offered at the online platform CarDekho[421]. The data set contains age, selling price, kilimeters driven and several other parameters of about 4000 cars.

The dataset has been obtained via Kaggle[422] and is licenced under Database Contents License (DbCL) v1.0[423], including Open Database License (ODbL) v1.0[424].

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

cars = pd.read_csv('cars.csv')

cars
```

```
                                 name  year  selling_price  km_driven  \
0                       Maruti 800 AC  2007          60000      70000
1            Maruti Wagon R LXI Minor  2007         135000      50000
2                Hyundai Verna 1.6 SX  2012         600000     100000
3              Datsun RediGO T Option  2017         250000      46000
4               Honda Amaze VX i-DTEC  2014         450000     141000
...                               ...   ...            ...        ...
4335  Hyundai i20 Magna 1.4 CRDi (Diesel)  2014         409999      80000
4336           Hyundai i20 Magna 1.4 CRDi  2014         409999      80000
4337                  Maruti 800 AC BSIII  2009         110000      83000
4338     Hyundai Creta 1.6 CRDi SX Option  2016         865000      90000
4339                     Renault KWID RXT  2016         225000      40000

        fuel seller_type transmission         owner
0     Petrol  Individual       Manual   First Owner
1     Petrol  Individual       Manual   First Owner
2     Diesel  Individual       Manual   First Owner
3     Petrol  Individual       Manual   First Owner
4     Diesel  Individual       Manual  Second Owner
...      ...         ...          ...           ...
4335  Diesel  Individual       Manual  Second Owner
4336  Diesel  Individual       Manual  Second Owner
4337  Petrol  Individual       Manual  Second Owner
4338  Diesel  Individual       Manual   First Owner
4339  Petrol  Individual       Manual   First Owner

[4340 rows x 8 columns]
```

We concentrate on one car model. Let's take the most frequent one.

```
grouped_by_name = cars.groupby('name')
name = grouped_by_name.size().idxmax()
cars_subset = grouped_by_name.get_group(name)

cars_subset
```

```
                       name  year  selling_price  km_driven    fuel  \
117   Maruti Swift Dzire VDI  2014         434000      79350  Diesel
327   Maruti Swift Dzire VDI  2018         800000      25000  Diesel
338   Maruti Swift Dzire VDI  2015         490000      60000  Diesel
```

(continues on next page)

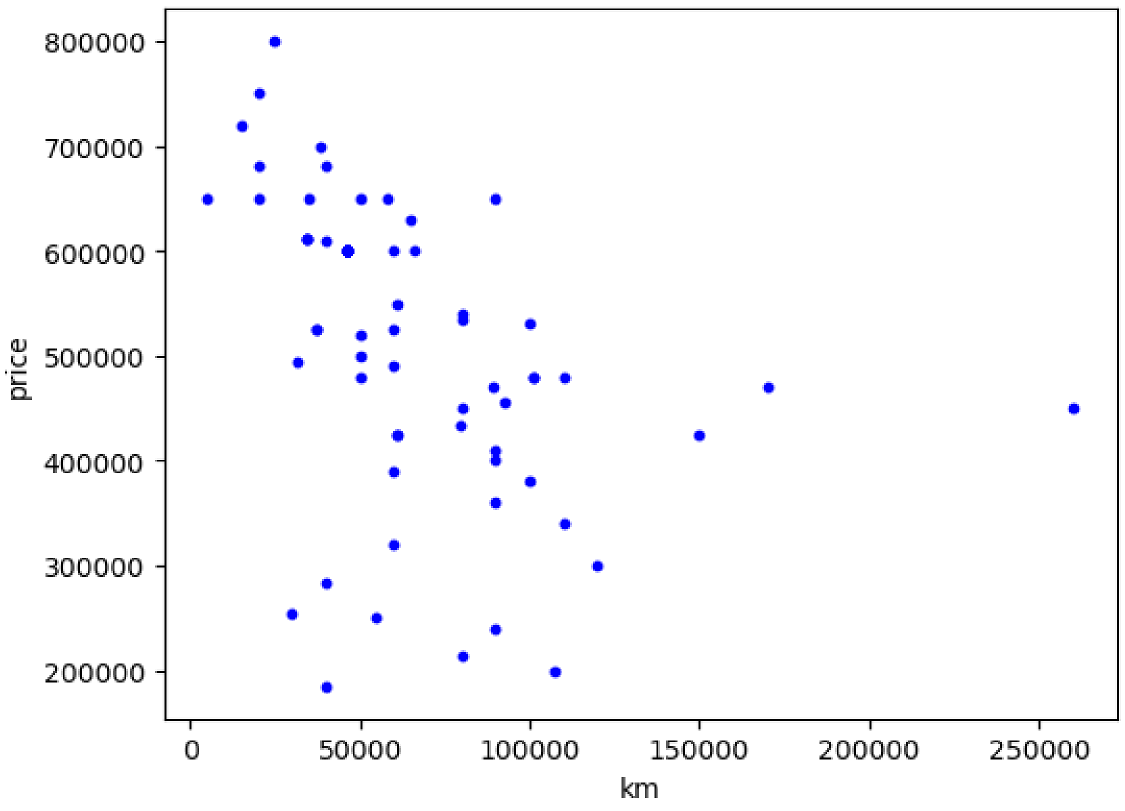
[421] https://www.cardekho.com
[422] https://www.kaggle.com
[423] http://opendatacommons.org/licenses/dbcl/1.0
[424] https://opendatacommons.org/licenses/odbl/summary

(continued from previous page)

```
365   Maruti Swift Dzire VDI  2014         400000      90000  Diesel
409   Maruti Swift Dzire VDI  2012         215000      80000  Diesel
...                      ...   ...            ...        ...     ...
4189  Maruti Swift Dzire VDI  2013         425000      61083  Diesel
4245  Maruti Swift Dzire VDI  2014         480000     101000  Diesel
4265  Maruti Swift Dzire VDI  2014         480000     101000  Diesel
4289  Maruti Swift Dzire VDI  2019         680000      40000  Diesel
4314  Maruti Swift Dzire VDI  2015         470000     170000  Diesel

     seller_type transmission         owner
117   Individual       Manual  Second Owner
327   Individual       Manual   First Owner
338   Individual       Manual  Second Owner
365   Individual       Manual   First Owner
409   Individual       Manual   Third Owner
...          ...          ...           ...
4189      Dealer       Manual   First Owner
4245      Dealer       Manual   First Owner
4265      Dealer       Manual   First Owner
4289  Individual       Manual   First Owner
4314  Individual       Manual   First Owner

[69 rows x 8 columns]
```

Before we start any computation we should have a look at the data.

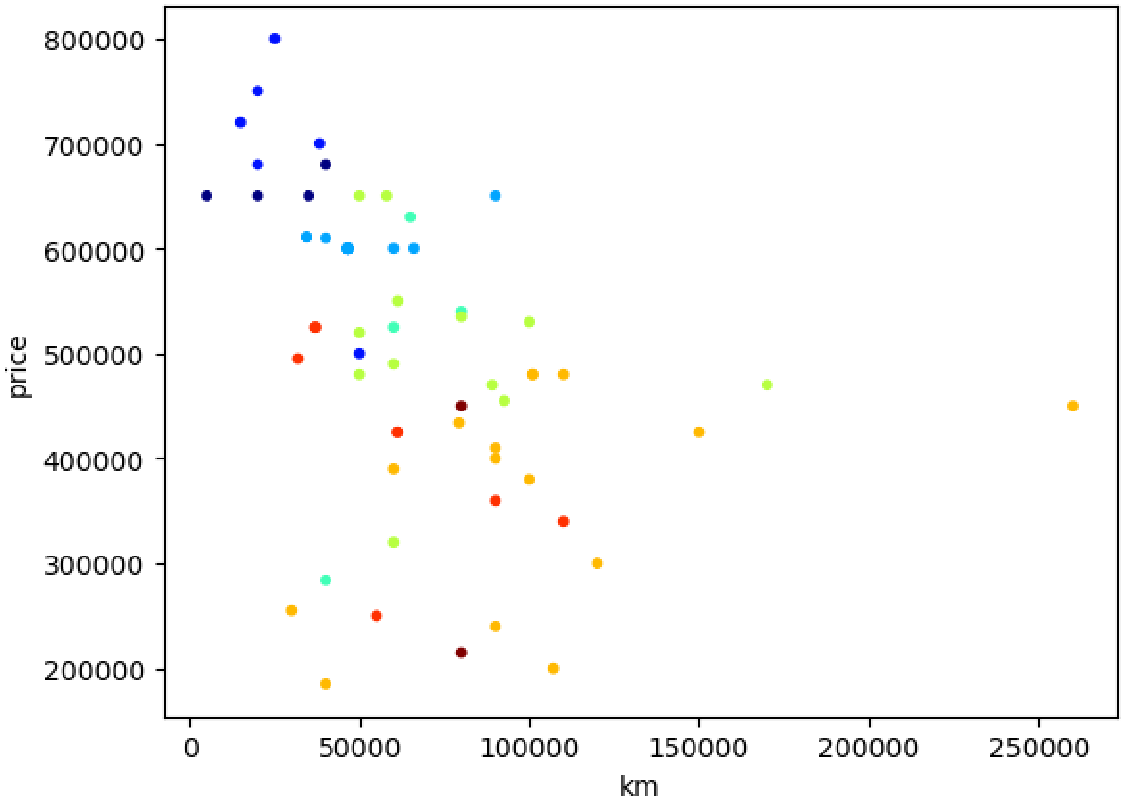
```
fig, ax = plt.subplots()

ax.scatter(cars_subset['km_driven'].to_numpy(), cars_subset['selling_price'].to_
↪numpy(), s=10, c='b')
ax.set_xlabel('km')
ax.set_ylabel('price')

plt.show()
```



This does not look like a linear dependence between price and kilometers driven. There seems to be no systematic dependence between both quantities at all. But intuition suggests that there should be one. Thus, we should take into account other features. Isn't it possible that the dependence between price and kilometers driven depends on the age of the car?

Let's visualize different ages.

```
fig, ax = plt.subplots()

ax.scatter(cars_subset['km_driven'].to_numpy(), cars_subset['selling_price'].to_
↪numpy(),
           s=10, c=2020 - cars_subset['year'].to_numpy(), cmap='jet')
ax.set_xlabel('km')
ax.set_ylabel('price')

plt.show()
```



The blue region shows some linear structure and the reddish region, too. Thus, age should be taken into account for predicting prices. Now let's do two regressions: one for young cars and one for old ones.

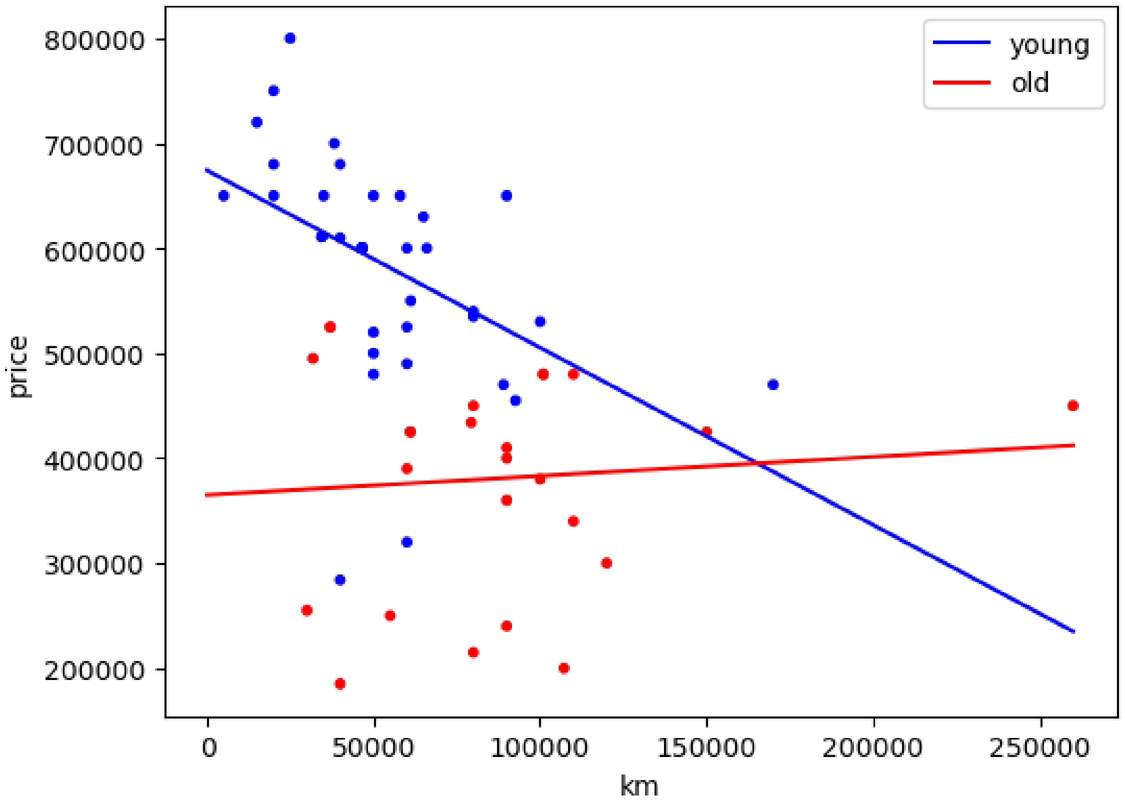
```
from matplotlib import colors

def simple_linear_regression(X, y):
    B = np.ones((X.size, 2))
    B[:, 1] = X
    return np.linalg.solve(np.matmul(B.T, B), np.matmul(B.T, y))

year = 2014   # where to split the data set into young and old

mask_young = cars_subset['year'] > year
X_young = cars_subset.loc[mask_young, 'km_driven'].to_numpy()
y_young = cars_subset.loc[mask_young, 'selling_price'].to_numpy()

mask_old = cars_subset['year'] <= year
X_old = cars_subset.loc[mask_old, 'km_driven'].to_numpy()
y_old = cars_subset.loc[mask_old, 'selling_price'].to_numpy()

a_young = simple_linear_regression(X_young, y_young)
a_old = simple_linear_regression(X_old, y_old)

print(a_young)
print(a_old)

fig, ax = plt.subplots()

ax.scatter(cars_subset['km_driven'].to_numpy(), cars_subset['selling_price'].to_
↪numpy(),
           s=10, c=(mask_young.to_numpy()), cmap=colors.ListedColormap(['r', 'b
↪']))
```

(continues on next page)

(continued from previous page)

```
xmin = 0
xmax = 260000
ax.plot([xmin, xmax], [a_young[0] + a_young[1] * xmin, a_young[0] + a_young[1] *␣
↪xmax], '-b', label='young')
ax.plot([xmin, xmax], [a_old[0] + a_old[1] * xmin, a_old[0] + a_old[1] * xmax], '-
↪r', label='old')

ax.set_xlabel('km')
ax.set_ylabel('price')
ax.legend()
plt.show()
```

```
[ 6.74201542e+05 -1.68902485e+00]
[3.64966780e+05 1.81825528e-01]
```



Given the kilometers driven, the simple formula

$$\boxed{y = 674202 - 1.69\,x}$$

yields a prediction for the price we could get for a young car, where x are the kilometers and y is the price. For old cars we have the model

$$\boxed{y = 364967 + 0.18\,x}$$

Note that this holds only for the selected car model. In addition, we only considered very simple hypotheses with only two parameters.

Instead of doing two one-dimensional regressions for young and old cars, we could do a two-dimensional regression including both features, km driven and age.

At first we look at the data.

```
import plotly.graph_objects as go

xcoords = cars_subset['km_driven'].to_numpy()
ycoords = 2020 - cars_subset['year'].to_numpy()
zcoords = cars_subset['selling_price'].to_numpy()

fig = go.Figure()
fig.layout.width = 800
fig.layout.height = 600

fig.add_trace(go.Scatter3d(
    x=xcoords, y=ycoords, z=zcoords,
    mode='markers',
    marker={'color': '#0000ff', 'size': 2}
))
fig.update_scenes(
    xaxis_title_text='km',
    yaxis_title_text='age',
    zaxis_title_text='price',
)

fig.show()
```

```
<IPython.core.display.HTML object>
```

Then we do the regression.

```
X = np.stack((cars_subset['km_driven'].to_numpy(),
              2020 - cars_subset['year'].to_numpy()), axis=1)
y = cars_subset['selling_price'].to_numpy()

B = np.ones((X.shape[0], 3))
B[:, 1:] = X
a = np.linalg.solve(np.matmul(B.T, B), np.matmul(B.T, y))

print(a)

fig = go.Figure()
fig.layout.width = 800
fig.layout.height = 600

fig.add_trace(go.Scatter3d(
    x=X[:, 0], y=X[:, 1], z=y,
    mode='markers',
    marker={'color': '#0000ff', 'size': 2}
))

xmin = 0
xmax = 250000
ymin = 0
ymax = 10
xcoords, ycoords = np.meshgrid([xmin, xmax], [ymin, ymax])
zcoords = a[0] + a[1] * xcoords + a[2] * ycoords
fig.add_trace(
    go.Surface(x=xcoords, y=ycoords, z=zcoords, showscale=False,
               colorscale=[[0, 'rgba(255, 0, 0, 0.8)'], [1, 'rgba(255, 0, 0, 0.8)
↪']])
)

fig.update_scenes(
```

(continues on next page)

(continued from previous page)

```
    xaxis_title_text='km',
    yaxis_title_text='age',
    zaxis_title_text='price',
)

fig.show()
```

```
[ 7.69448384e+05 -2.07046783e-01 -5.47861786e+04]
```

```
<IPython.core.display.HTML object>
```

Now our model for the price y depending on kilometers x_1 and age x_2 is

$$\boxed{y = 769448 - 0.21\,x_1 - 54786\,x_2.}$$

Limitations

Linear regression with linear functions yields inaccurate results if the data set does not show a linear structure. Also outliers may distort results. Thus, linear regression has to be used with care and preceeding visual analysis is mandatory.

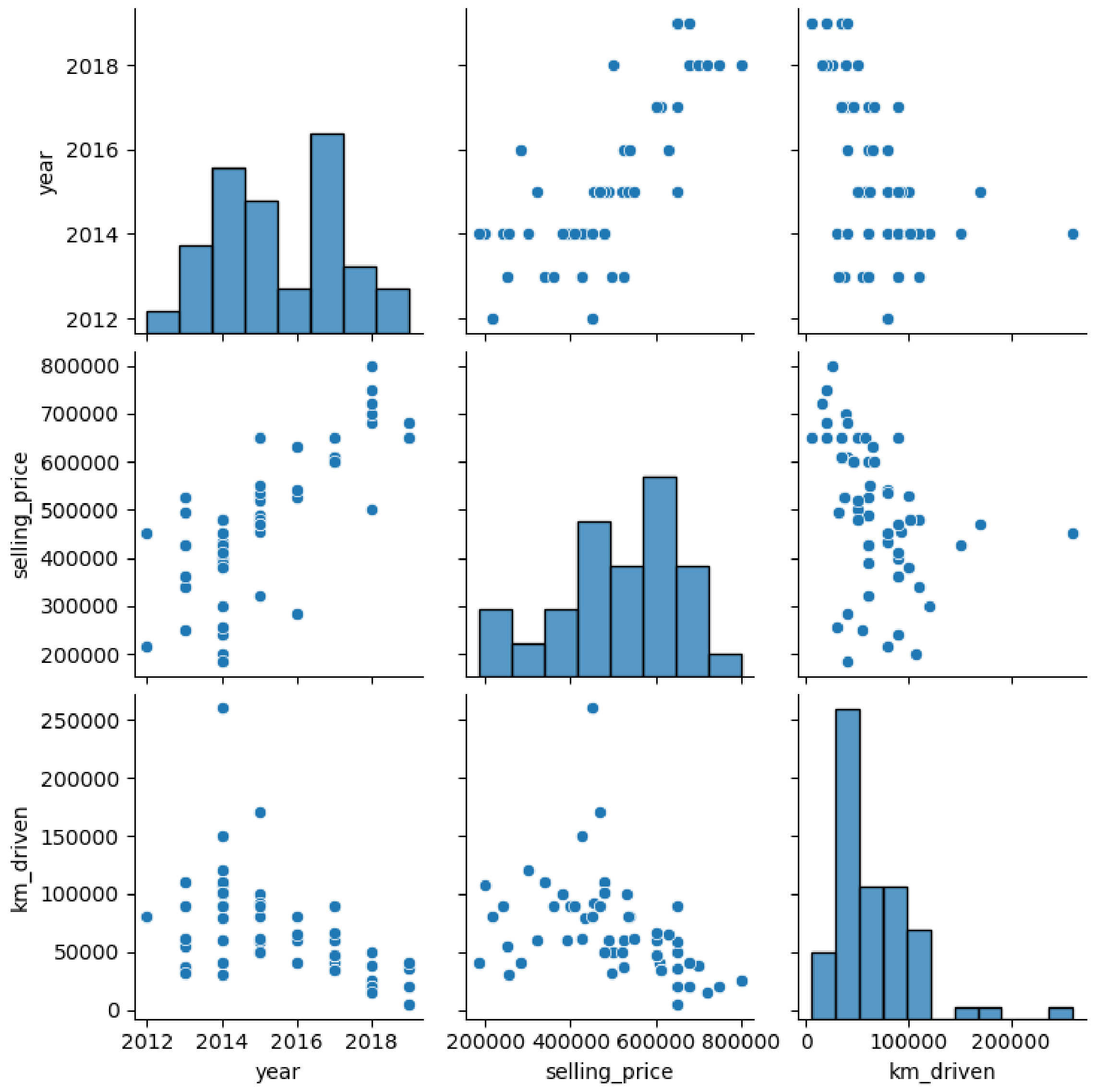
Have a look at Ascombe's quartet[425] for very different data sets all yielding the same result if linear regression with linear functions is used.

Linear Regression with Seaborn

Seaborn offers `pairplot` for getting a first overview of mutual dependence of all variables. See Plotting pairwise data relation ships[426] in the Seaborn documentation for more details.

```
import seaborn as sns

sns.pairplot(cars_subset)
plt.show()
```

425 https://en.wikipedia.org/wiki/Anscombe%27s_quartet
426 https://seaborn.pydata.org/tutorial/axis_grids.html#plotting-pairwise-data-relationships

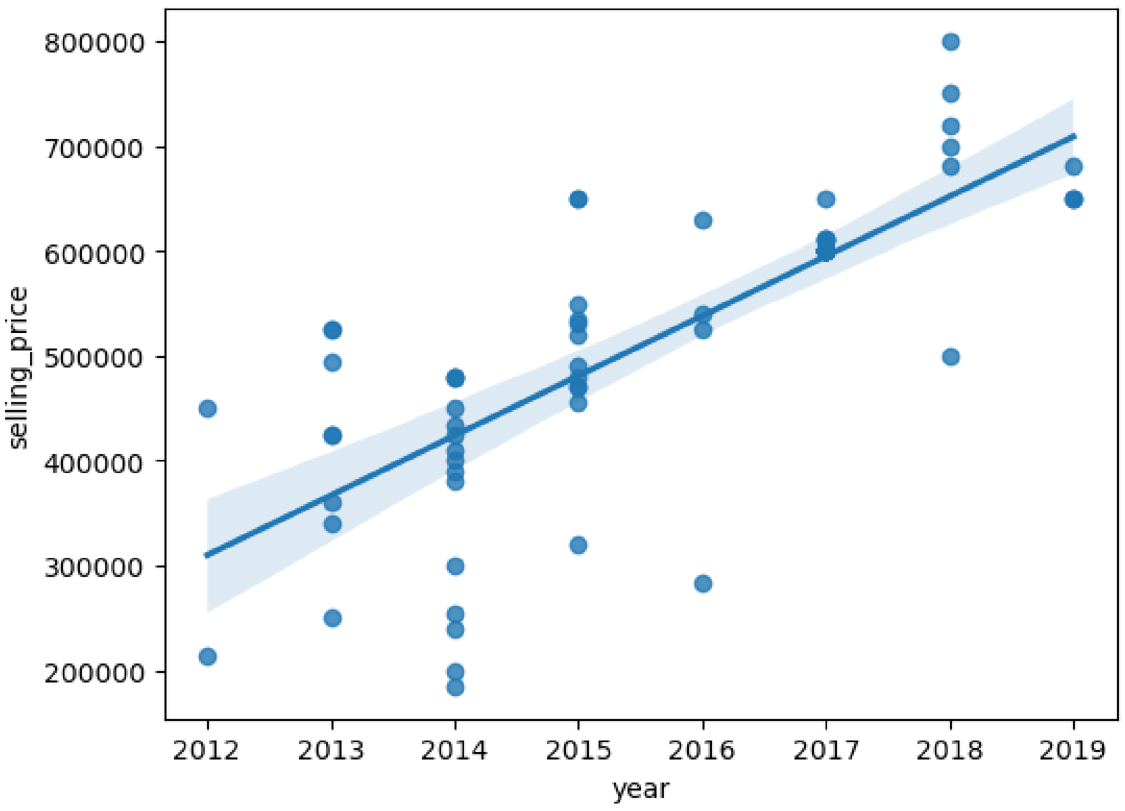


Seaborn also provides linear regression directly. See Estimating regression fits[427] for more details.

```
sns.regplot(x='year', y='selling_price', data=cars_subset)
```

```
<Axes: xlabel='year', ylabel='selling_price'>
```

[427] https://seaborn.pydata.org/tutorial/regression.html



Linear Regression with Scikit-Learn

Linear regression is implemented by almost all machine learning libraries. Scikit-Learn is a good choice due to its simple API. It offers a module `linear_model`.

We reimplement linear regression with two features for the above example of used car prices.

The workflow is as follows:

- create a `LinearRegression` object,
- fit the object to the data (that is, do the regression),
- extract the coefficients from the object or use the object for prediction.

```
# X, y from above

import sklearn.linear_model as lm

regression = lm.LinearRegression()

regression.fit(X, y)

print(regression.intercept_)     # a[0]
print(regression.coef_)          # a[1], a[2]
```

```
769448.3841336307
[-2.07046783e-01 -5.47861786e+04]
```

From Scikit-Learn we obtain identical coefficients as above. To evaluate the regression function call `predict`.

```
age = 3.5
km = 40000

price = regression.predict([[km, age]])

print(price)
```

```
[569414.88787459]
```

For comparison with more complex models below we calculate (root) mean squared error on the training set.

```
import sklearn.metrics as metrics

metrics.mean_squared_error(y, regression.predict(X), squared=False)
```

```
89976.37031916517
```

24.1.2 Linear Regression with Nonlinear Functions

Idea

We consider more general parameter dependent hypotheses of the form

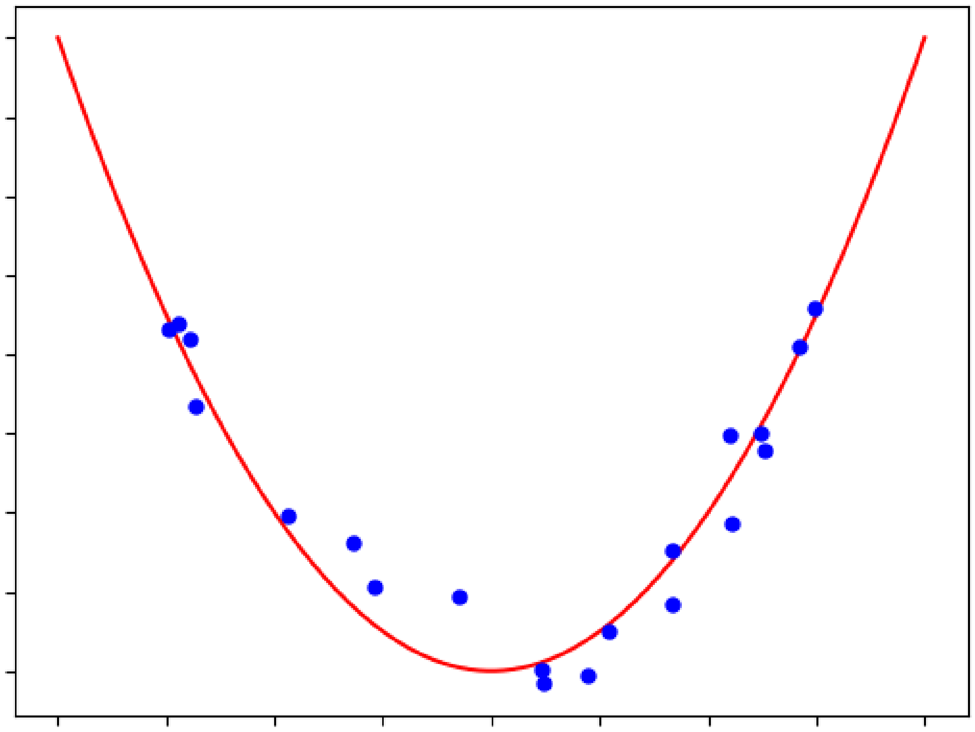
$$f_{\text{approx}}(x) = a_1\,\varphi_1(x^{(1)},\ldots,x^{(m)}) + \cdots + a_\mu\,\varphi_\mu(x^{(1)},\ldots,x^{(m)}) = \sum_{\kappa=1}^{\mu} a_\kappa\,\varphi_\kappa(x^{(1)},\ldots,x^{(m)}),$$

where $x = (x^{(1)},\ldots,x^{(m)})$ is a feature vector, $a_1,\ldots,a_\mu$ are the parameters of the hypothesis f_{approx}, and $\varphi_1,\ldots,\varphi_\mu$ are prescribed real-valued functions on $\mathbb{R}^m$.

Note that f_{approx} depends linearly on the parameters $a_1,\ldots,a_\mu$. Thus *linear* regression. In contrast, in *nonlinear* regression we would consider hypotheses containing the searched for parameters in a nonlinear fashion. This would be the most general case, which is rarely needed in practise.

Example: Fitting a Parabola

Consider data with only one feature, that is, $m = 1$. We would like to fit a parabola to such a data set.



In mathematical notation our hypotesis shall have the form

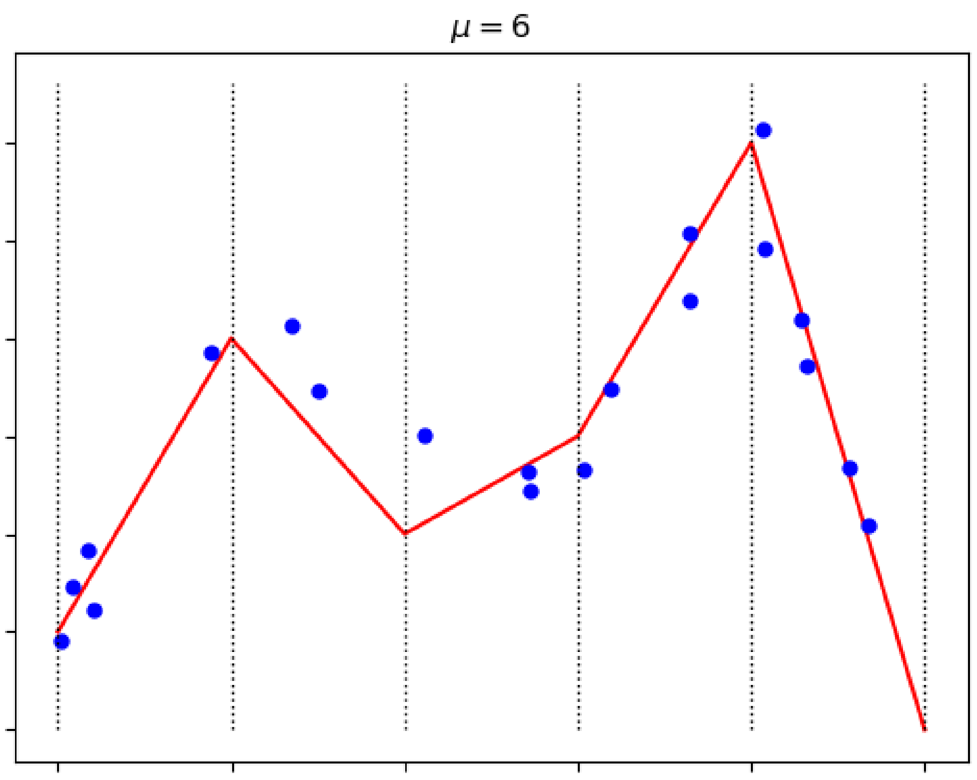
$$f_{\text{approx}}(x) = a_1\,x^2 + a_2\,x + a_3,$$

where we replaced $x^{(1)}$ by x since we only have one feature. Thus, fitting a parabola is a special case of our general linear regression approach with

$$\mu = 3, \qquad \varphi_1(x) = x^2, \qquad \varphi_2(x) = x, \qquad \varphi_3(x) = 1.$$

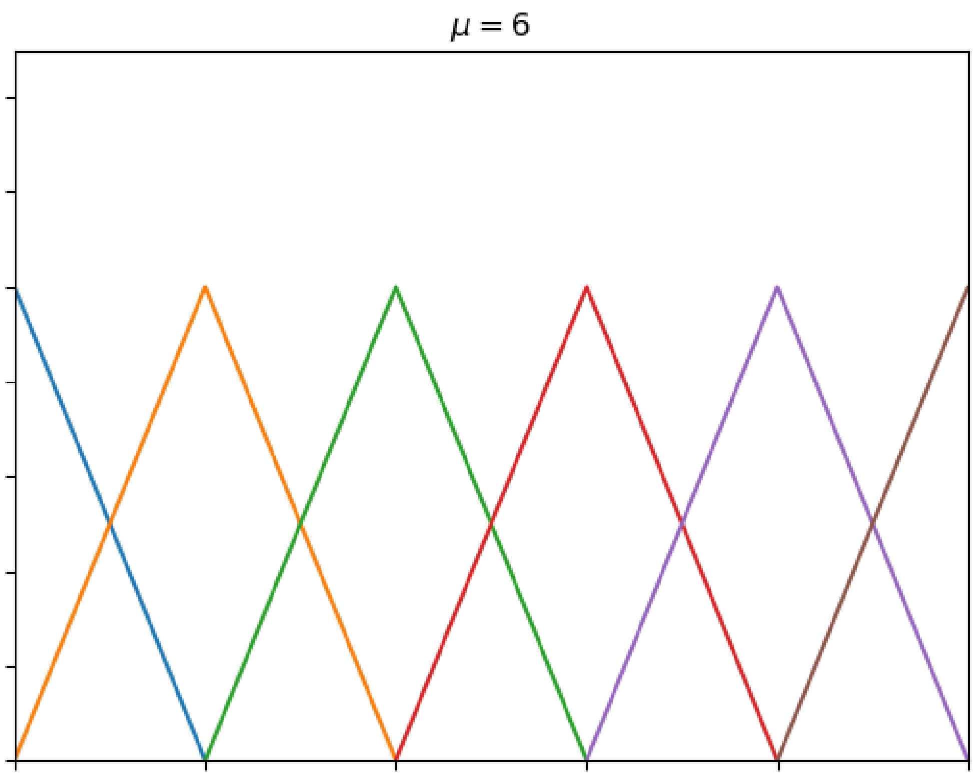
Example: Fitting Piecewise Linear Functions

Again consider data with only one feature. In addition, assume that the feature takes values in $[0, 1]$. Then it might be a good idea to divide $[0, 1]$ into $\mu - 1$ equaly spaced subintervals for some fixed μ. On each subinterval we could do a linear regression, but require that resulting lines are connected at the interval boundaries.



In our linear regression framework we could choose $\varphi_1, \ldots, \varphi_\mu$ to be hat functions of width $\frac{2}{\mu-1}$ centered at the interval boundaries $\frac{\kappa-1}{\mu-1}$ for $\kappa = 1 \ldots, \mu$:

$$\varphi_\kappa(x) = \begin{cases} (\mu - 1)\,x - \kappa + 2, & \text{if } x \in [\frac{\kappa-2}{\mu-1}, \frac{\kappa-1}{\mu-1}], \\ -(\mu - 1)\,x + \kappa, & \text{if } x \in [\frac{\kappa-1}{\mu-1}, \frac{\kappa}{\mu-1}], \\ 0, & \text{else.} \end{cases}$$



The parameters $a_1, \ldots, a_\mu$ describe the high of the hats or the function values of f_{approx} at the interval boundaries.

Example: RBF Regression

Using several copies of a hat function and placing them at fixed grid points is a special case of RBF regression. Here RBF is the abbreviation of *radial basis function*. An RBF is a function that depends only on the length $|x|$ of a feature vector x and not on the individual features. Thus, it is symmetric for one-dimensional feature vectors, cone shaped for two-dimensional feature vectors, and so on. RBFs typically have their maximum at 0 (the center) and then decay when distance from the center increases.

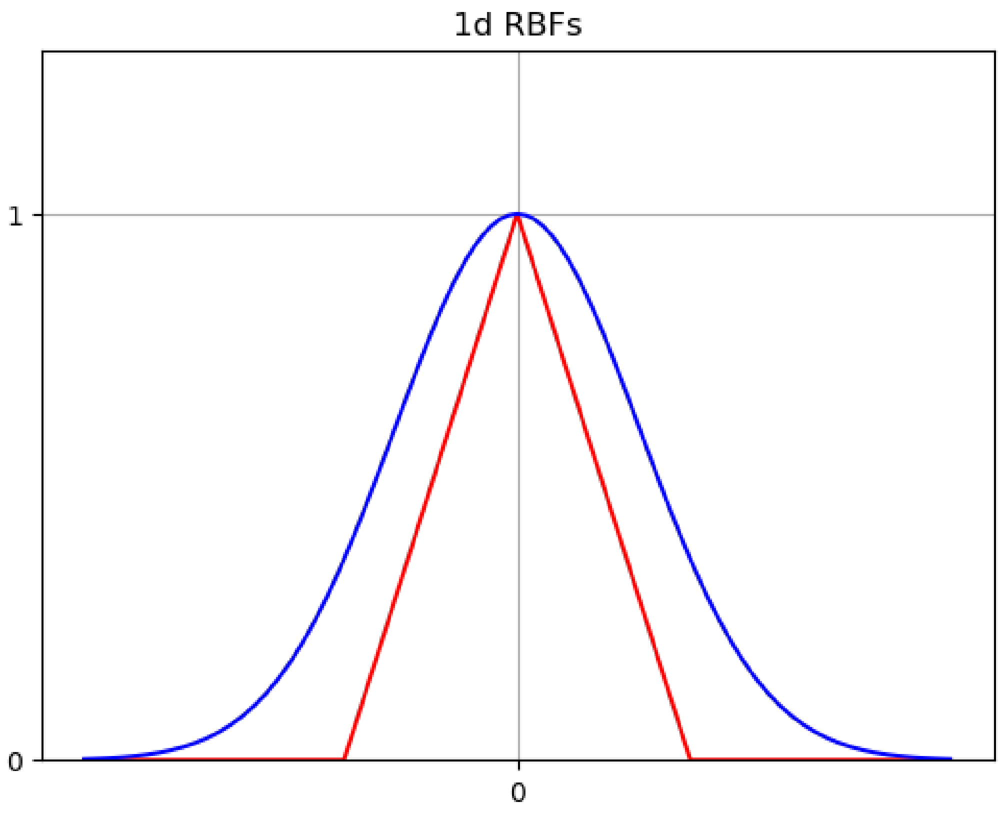
Next to hat functions

$$\varphi(x) = \begin{cases} 1 - \frac{1}{c}\,|x|, & \text{if } |x| \leq c, \\ 0, & \text{else,} \end{cases}$$

where $c > 0$ controls the width of the hat ($c = \frac{1}{\mu-1}$ in the previous example), Gaussian RBFs

$$\varphi(x) = \mathrm{e}^{-c\,|x|^2}$$

are widely used. Again, $c > 0$ controls the width of the bell shaped curve.



```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

Given fixed grid points $u_1, \ldots, u_\mu$ in the feature space we choose

$$\varphi_1(x) = \varphi(x - u_1), \quad \ldots, \quad \varphi_\mu(x) = \varphi(x - u_\mu).$$

Relation to Linear Regression with Linear Functions

Linear regression with nonlinear functions can be reduced to linear regression with linear functions studied above. We simply have to transform all feature vectors by applying the functions $\varphi_1,\ldots,\varphi_\mu$.

Given feature vectors $x_1,\ldots,x_n$ with m features each we define new feature vectors $\tilde{x}_1,\ldots,\tilde{x}_n$ with μ features by

$$\tilde{x}_1^{(\kappa)}:=\varphi_\kappa(x_1),\quad\ldots,\quad\tilde{x}_n^{(\kappa)}:=\varphi_\kappa(x_n)\qquad\text{for }\kappa=1,\ldots,\mu.$$

Now our hypothesis reads

$$f_{\text{approx}}(\tilde{x})=\sum_{\kappa=1}^{\mu}a_\kappa\,\tilde{x}^{(\kappa)}$$

with $\tilde{x}$ from $\mathbb{R}^\mu$.

Implementation with Scikit-Learn

Scikit-Learn provides a module for preprocessing data. This module contains a `PolynomialFeatures`[428] class for creating transformer objects. The transformer object then provides a method `fit_transform`[429] doing the actual transformation.

Care has to be taken at the following point: Scikit-Learn's polynomial feature transform adds a feature which is always one. This feature corresponds to x^0. But the linear regression model adds this feature again. Thus, we end up with too many parameters. From the mathematical point of view this isn't a problem. But there might be confusion when evaluating the coefficients for some reason. Either set `include_bias` to `False` when creating the transformer object or set `fit_intercept` to `False` when creating the `LinearRegression` object.

Again we reuse the used car price example with two-dimensional feature space. Now we want to fit a second-order function to the data.

```python
# X, y from above

from sklearn.preprocessing import PolynomialFeatures

transformer = PolynomialFeatures(degree=2)
transformed_X = transformer.fit_transform(X)

print('shape of original data:    ', X.shape)
print('shape of transformed data:', transformed_X.shape)
```

```
shape of original data:     (69, 2)
shape of transformed data: (69, 6)
```

Starting with two features $x^{(1)}$ and $x^{(2)}$, we now have six features:

$$\tilde{x}^{(1)}=1,\quad\tilde{x}^{(2)}=x^{(1)},\quad\tilde{x}^{(3)}=x^{(2)},\quad\tilde{x}^{(4)}=\left(x^{(1)}\right)^2,\quad\tilde{x}^{(5)}=x^{(1)}\,x^{(2)},\quad\tilde{x}^{(6)}=\left(x^{(2)}\right)^2.$$

From here on regression works as usual.

```python
regression = lm.LinearRegression(fit_intercept=False)
regression.fit(transformed_X, y)

# predict price
age = 3.5
km = 40000
price = regression.predict(transformer.fit_transform([[km, age]]))
```

(continues on next page)

[428] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html
[429] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html#sklearn.preprocessing.PolynomialFeatures.fit_transform

(continued from previous page)

```
print(price)

fig = go.Figure()
fig.layout.width = 800
fig.layout.height = 600

fig.add_trace(go.Scatter3d(
    x=X[:, 0], y=X[:, 1], z=y,
    mode='markers',
    marker={'color': '#0000ff', 'size': 2}
))

xmin = 0
xmax = 250000
ymin = 0
ymax = 10
grid_size = 20

xcoords, ycoords = np.meshgrid(np.linspace(xmin, xmax, grid_size), np.
↪linspace(ymin, ymax, grid_size))
xyfeatures = np.stack((xcoords.reshape(-1), ycoords.reshape(-1)), axis=1)
zcoords = regression.predict(transformer.fit_transform(xyfeatures)).reshape(grid_
↪size, grid_size)

fig.add_trace(
    go.Surface(x=xcoords, y=ycoords, z=zcoords, showscale=False,
               colorscale=[[0, 'rgba(255, 0, 0, 0.8)'], [1, 'rgba(255, 0, 0, 0.8)
↪']])
)

fig.update_scenes(
    xaxis_title_text='km',
    yaxis_title_text='age',
    zaxis_title_text='price',
)

fig.show()

metrics.mean_squared_error(y, regression.predict(transformed_X), squared=False)
```

```
[569498.41773257]
```

```
<IPython.core.display.HTML object>
```

```
88734.00776114204
```

24.2 Regularization

Whenever we try to fit a model to a finite data set we have to find a compromise between two competing aims:

- Fit the data as good as possible.
- Generalize information from data to regions in feature space without data (fit the truth).

One problem is that data usually contains noise and thus does not contain arbitrarily precise information about the underlying truth. On the other hand, in most applications there is not the one underlying truth. If some relevant features are not contained in the data set, then even a complete data set does not allow to recover underlying truth.

An example for the second issue is prediction of prices, say house prices. The price depends on many features, which cannot be recorded completely. Thus, corresponding data set might contain a feature vector twice, but with different target values (prices). Which is the better one, that is, which one contains more truth?

Fitting data as good as possible is quite easy. The hard part is to avoid *overfitting*. By overfitting we mean neglecting the second aim. That is, our hypothesis fits the data very well, but does not represent essential properties of the underlying truth.

24.2.1 Example

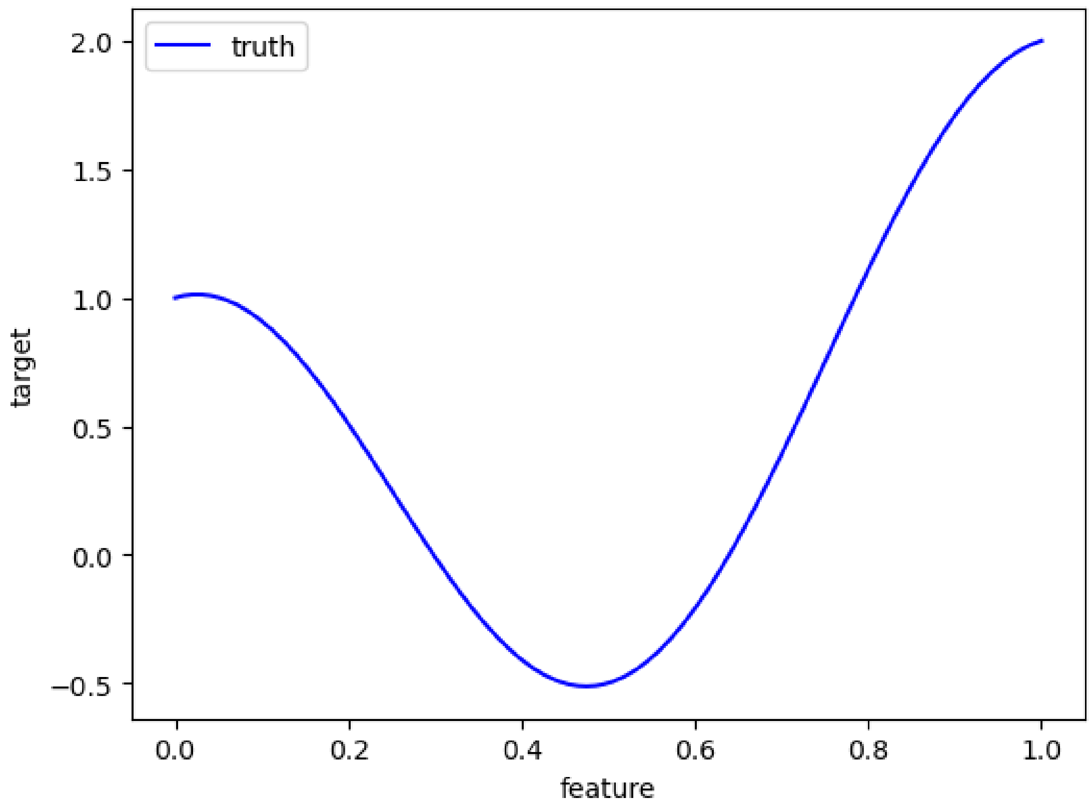
Let's have a look at an illustrating example. We consider data with only one feature, so we can plot everything and visualize the problem.

First some standard imports and initialization of the random number generator.

```
import numpy as np
import matplotlib.pyplot as plt

import sklearn.linear_model as lm
from sklearn.preprocessing import PolynomialFeatures

from numpy.random import default_rng
rng = default_rng(0)
```

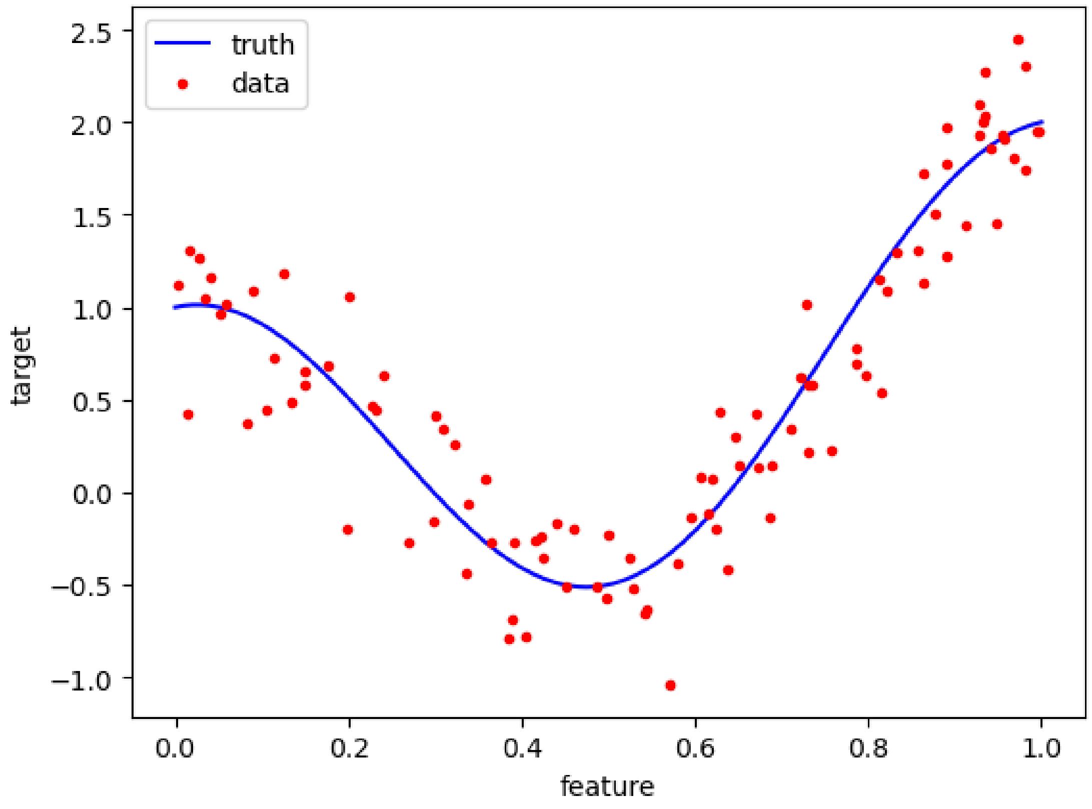
To investigate overfitting we choose an underlying truth and simulate data based on this truth. This way we have access to the in practice unknown truth and can compare predictions to the truth.

```
# function to reconstruct from data ('underlying truth')
def truth(x):
    return x + np.cos(2 * np.pi * x)

# range and grid for plotting
xmin = 0
xmax = 1
x = np.linspace(xmin, xmax, 100)

# plot truth
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='truth')
ax.legend()
ax.set_xlabel('feature')
ax.set_ylabel('target')
plt.show()
```



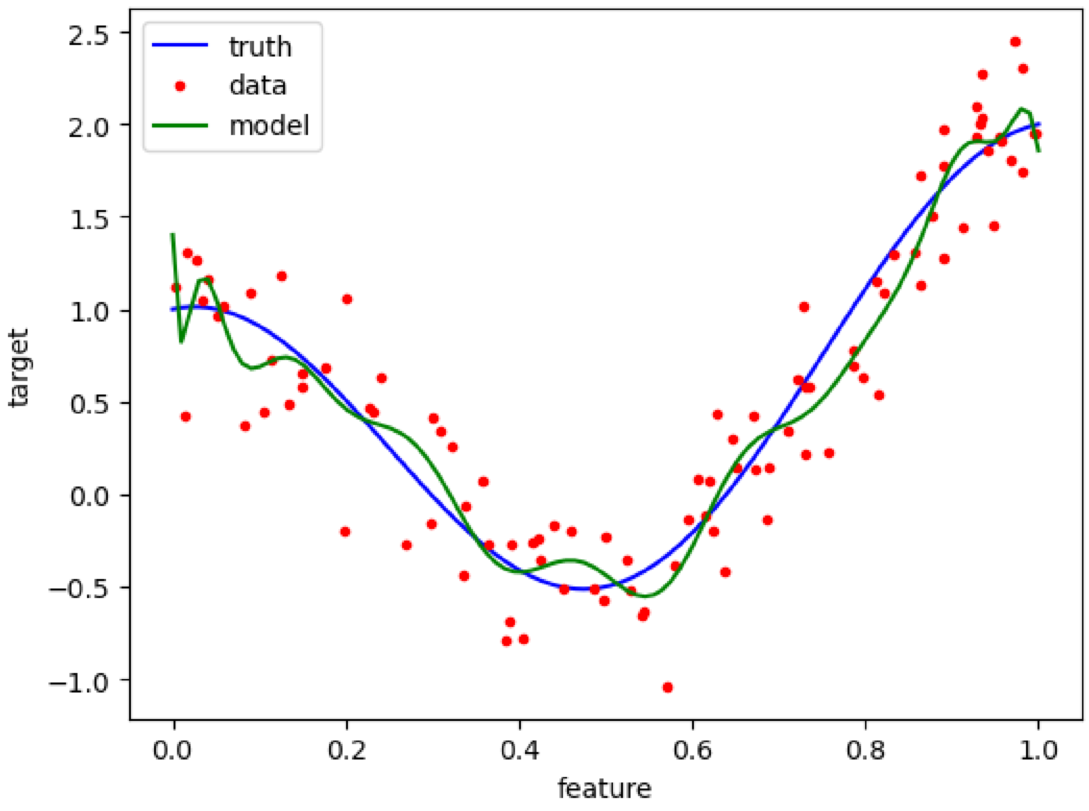
To simulate data we generate uniformly distributed arguments, calculate corresponding true function values, and add some noise. Noise almost always follows a normal distribution.

```
n = 100     # number of data points to generate
noise_level = 0.3     # standard deviation of artificial noise

# simulate data
X = (xmax - xmin) * rng.random((n, 1)) + xmin
y = truth(X).reshape(-1) + noise_level * rng.standard_normal(n)

# plot truth and data
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='truth')
ax.plot(X.reshape(-1), y, 'or', markersize=3, label='data')
ax.set_xlabel('feature')
ax.set_ylabel('target')
ax.legend()
plt.show()
```



We use polynomial regression for obtaining a model explaining our data. Different degrees of the polynomial will yield very different results (try 1, 2, 5, 10, 15, 20, 25).

```
degree = 20    # degree for polynomial regression

# regression
regression = lm.LinearRegression()
transform = PolynomialFeatures(degree=degree).fit_transform
regression.fit(transform(X), y)

# get hypothesis for plotting
y_reg = regression.predict(transform(x.reshape(-1, 1)))

# plot truth, data, hypothesis
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='truth')
ax.plot(X.reshape(-1), y, 'or', markersize=3, label='data')
ax.plot(x, y_reg, '-g', label='model')
ax.set_xlabel('feature')
ax.set_ylabel('target')
ax.legend()
plt.show()
```



Obviously, there is an optimal degree, say 4 or 5 or 6. The degree in polynomial regression is a hyperparameter.

For lower degrees our model is not versatile enough to grasp the truth's structure. For higher degrees we observe overfitting: the model adapts very well to the data points, but tends to oscillate to reach as many data points as possible. These oscillations are an artifact and not a characteristic of the underlying truth.

Before we discuss how to avoid overfitting, we have to think about a different issue: How to detect overfitting? In our illustrating example we know the underlying truth and can compare the hypothesis to the truth. But in practice we do not know the truth!

24.2.2 Detecting overfitting

We split our data set into two subsets: one for fitting the model (training set) and one for detecting overfitting (validation set). If our model is close to the (unknown) truth, then the error on both subsets should be almost identical. In case of overfitting the error on the training set will be much smaller than on the validation set.

Here, the error is the *mean squared error*:

$$\frac{1}{n}\sum_{k=1}^{n}\left(f_{\text{approx}}(x_k)-y_k\right)^2,$$

where $(x_1, y_1), \ldots, (x_n, y_n)$ are the samples from the considered subset.

Let's test this with the above example. First we split the data set.

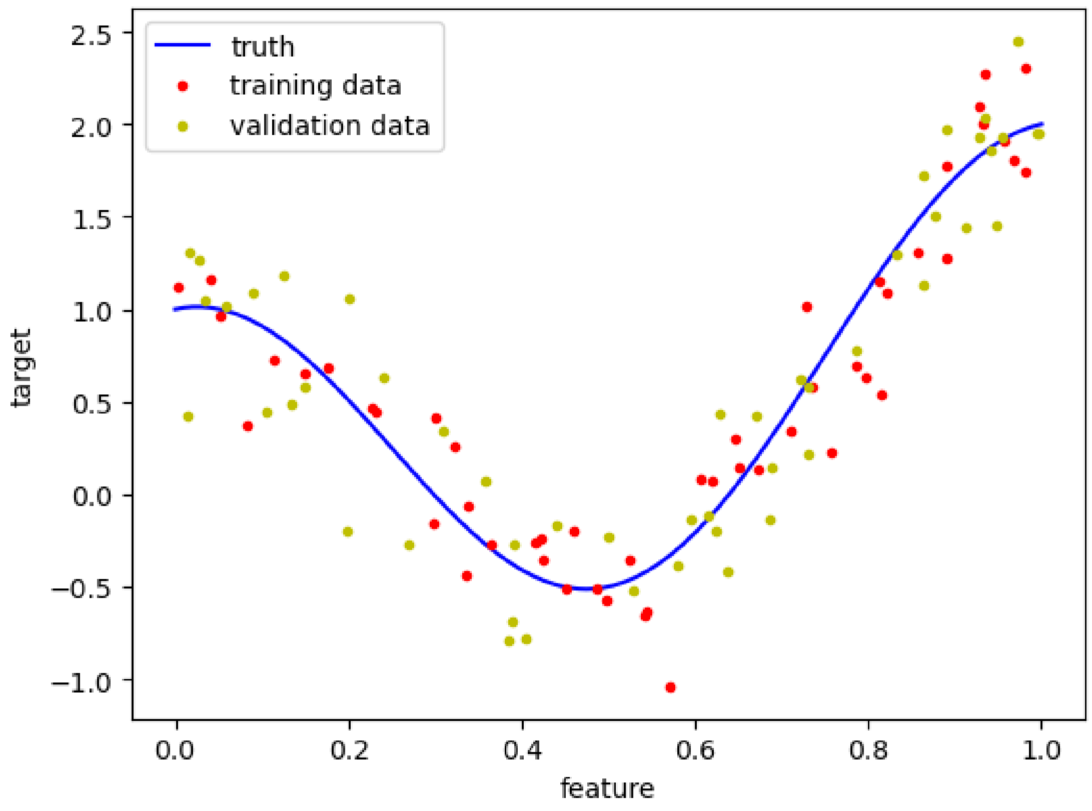
```
from sklearn.model_selection import train_test_split

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.5)

print(X_train.shape, X_val.shape)

# plot truth, training data, validation data
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='truth')
ax.plot(X_train.reshape(-1), y_train, 'or', markersize=3, label='training data')
ax.plot(X_val.reshape(-1), y_val, 'oy', markersize=3, label='validation data')
ax.set_xlabel('feature')
ax.set_ylabel('target')
ax.legend()
plt.show()
```

```
(50, 1) (50, 1)
```



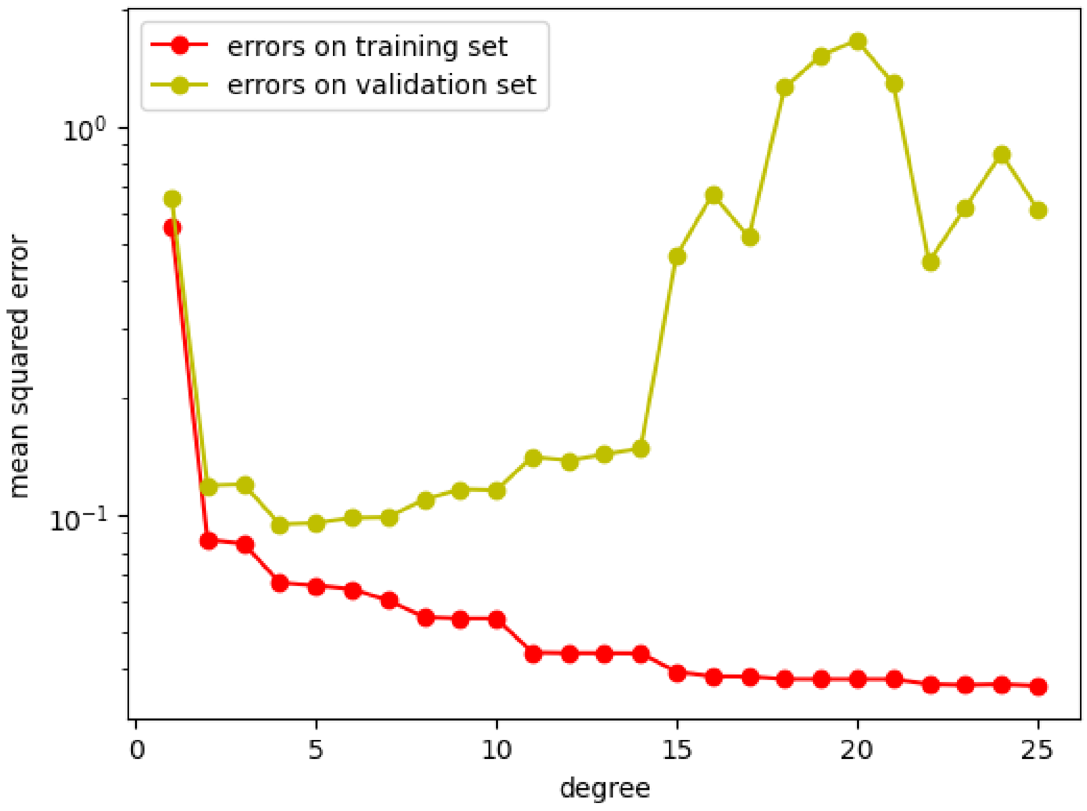
Now we fit models for different degrees and plot corresponding errors on the training set and on the validation set.

```
from sklearn import metrics

max_degree = 25

regression = lm.LinearRegression()

train_errors = np.zeros(max_degree)
val_errors = np.zeros(max_degree)

degrees = range(1, max_degree + 1)

for degree in degrees:

    # regression
    transform = PolynomialFeatures(degree=degree).fit_transform
    regression.fit(transform(X_train), y_train)

    # predictions on subsets
```

(continues on next page)

(continued from previous page)

```
    y_train_pred = regression.predict(transform(X_train))
    y_val_pred = regression.predict(transform(X_val))

    # errors
    train_errors[degree - 1] = metrics.mean_squared_error(y_train_pred, y_train)
    val_errors[degree - 1] = metrics.mean_squared_error(y_val_pred, y_val)

# plot errors
fig, ax = plt.subplots()
ax.semilogy(degrees, train_errors, '-or', label='errors on training set')
ax.semilogy(degrees, val_errors, '-oy', label='errors on validation set')
ax.set_xlabel('degree')
ax.set_ylabel('mean squared error')
ax.legend()
plt.show()

# print errors
print('    training     validation')
print(np.stack((train_errors, val_errors), axis=1))
```



```
   training     validation
[[0.55327756 0.65483872]
 [0.08634214 0.119108  ]
 [0.08442359 0.11991263]
 [0.06694953 0.09463578]
 [0.06584899 0.09539557]
 [0.06440824 0.09831239]
 [0.06030987 0.09869475]
 [0.05459956 0.10948717]
 [0.05416856 0.11645479]
 [0.05416567 0.11578297]
```

(continues on next page)

(continued from previous page)

```
 [0.04427975 0.14101537]
 [0.04413737 0.13811564]
 [0.04411521 0.1431831 ]
 [0.04407453 0.14827712]
 [0.03944999 0.46436207]
 [0.03846396 0.66928752]
 [0.03839572 0.51795306]
 [0.0378066  1.25822297]
 [0.03778724 1.5208766 ]
 [0.03778158 1.66348658]
 [0.03777532 1.28780466]
 [0.03667487 0.44913459]
 [0.03650158 0.61691911]
 [0.03667436 0.85090129]
 [0.03632425 0.61247783]]
```

The higher the degree, the smaller the error on the training set, but the higher the error on the validation set. Starting at degree about 15 the difference between both errors grows significantly. This shows that the small error on the training set is not a result of a well approximated truth, but stems from overfitting.

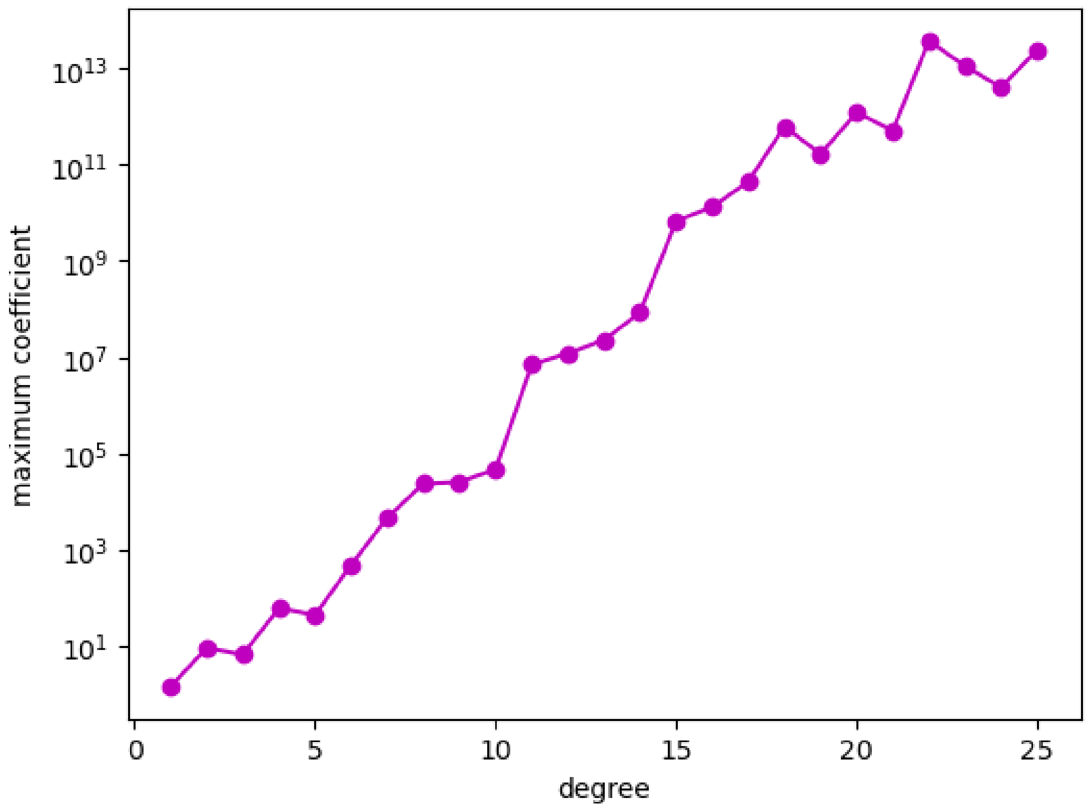
Here we also see that the error on the validation set is slightly larger than on the training set, because the hypothesis has been fitted to the training data.

24.2.3 Avoiding Overfitting

Overfitting almost always correlates with very large parameters after fitting the model. Thus, penalizing parameter values should be a good idea.

Let's have a look at the parameters in our illustrative example for both cases good fit and overfitting.

```python
max_degree = 25

regression = lm.LinearRegression()

max_coeffs = np.zeros(max_degree)

degrees = range(1, max_degree + 1)

for degree in degrees:

    # regression
    transform = PolynomialFeatures(degree=degree).fit_transform
    regression.fit(transform(X_train), y_train)

    # maximum coefficient
    max_coeffs[degree - 1] = np.max(np.abs(regression.coef_))

# plot errors
fig, ax = plt.subplots()
ax.semilogy(degrees, max_coeffs, '-om',)
ax.set_xlabel('degree')
ax.set_ylabel('maximum coefficient')
plt.show()
```



In linear regression and most other method one minimizes a loss function expressing the distance between the hypothesis f_{approx} and the targets in the training data:

$$\frac{1}{n}\sum_{l=1}^{n}\left(f_{\text{approx}}(x_l)-y_l\right)^2 \to \min_{a_1,\ldots,a_\mu},$$

where $a_1,\ldots,a_\mu$ are the parameters of the model. If we add the squares of the parameters to this function, then we not only force the hypothesis to be close to the data, but we also ensure that the parameters cannot become too large. As we mentioned above, large parameters correlate with overfitting. Modifying a minimization problem in this way is known as *regularization*.

To control the trade-off between data fitting and regularization, we introduce a *regularization parameter* $\alpha \geq 0$:

$$\frac{1}{n}\sum_{l=1}^{n}\left(f_{\text{approx}}(x_l)-y_l\right)^2 + \alpha\,\frac{1}{\mu}\sum_{\kappa=1}^{\mu}a_\kappa^2 \to \min_{a_1,\ldots,a_\mu}.$$

The regularization parameter α is an additional hyperparameter of the model.

There are several other penalty terms, which will be discussed below. Adding squares of the model parameters is the simplest version from the view of computational efficiency. Linear regression regularized this way is also known as *Ridge regression*.

Scikit-Learn implements Ridge regression in the `linear_model` module: `Ridge`[430].

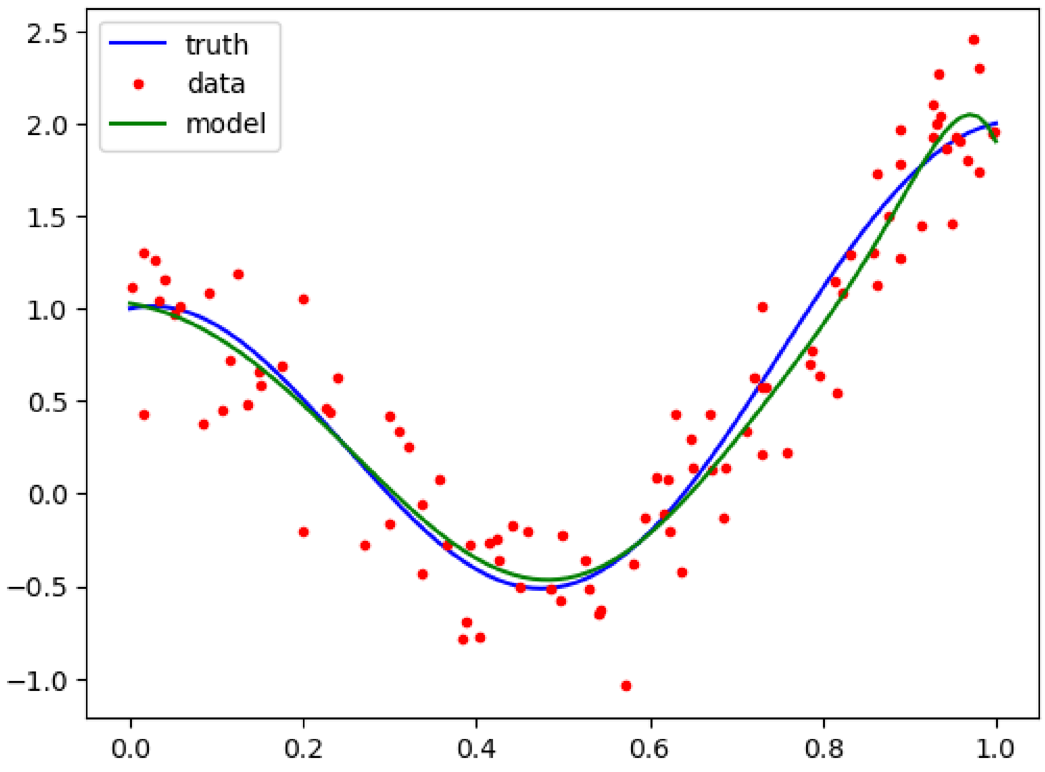
```
degree = 20     # degree for polynomial regression
alpha = 1e-5     # regularization parameter

# regression
regression = lm.Ridge(alpha=alpha)
transform = PolynomialFeatures(degree=degree).fit_transform
regression.fit(transform(X), y)
```

(continues on next page)

[430] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

(continued from previous page)

```
# get hypothesis for plotting
y_reg = regression.predict(transform(x.reshape(-1, 1)))

# plot truth, data, hypothesis
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='truth')
ax.plot(X.reshape(-1), y, 'or', markersize=3, label='data')
ax.plot(x, y_reg, '-g', label='model')
ax.legend()
plt.show()
```



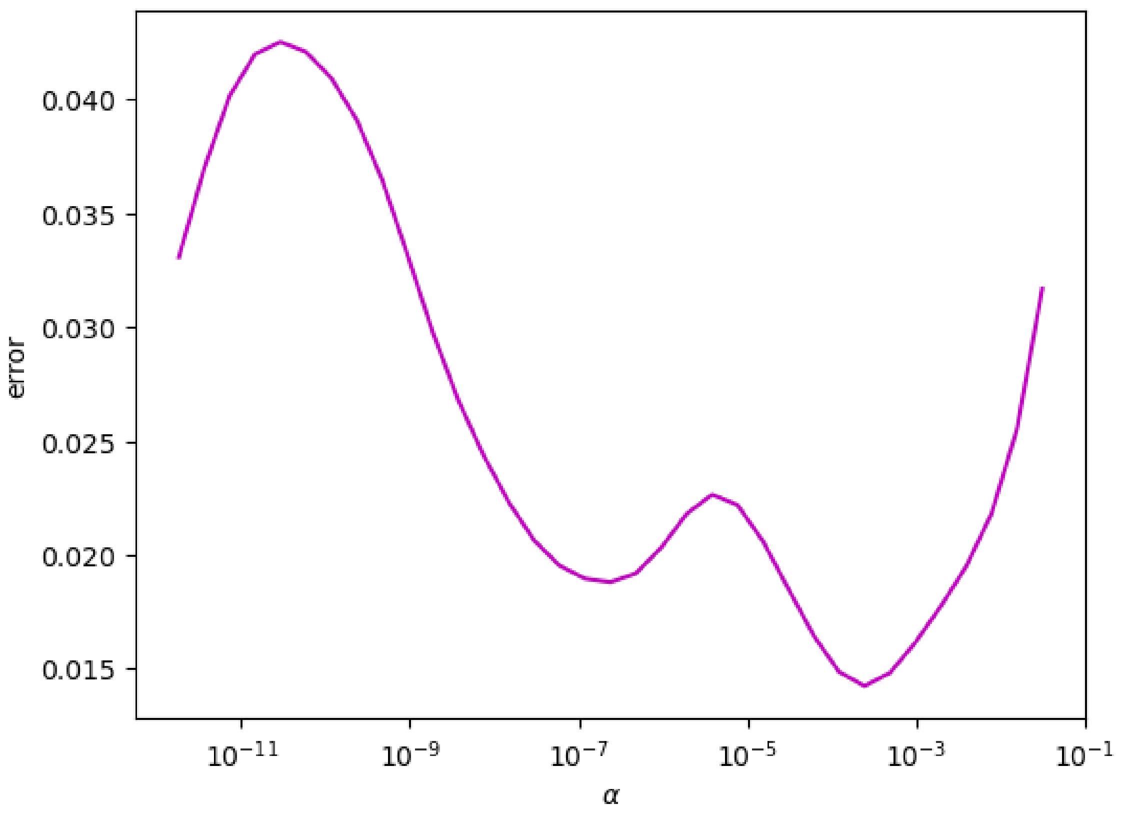
In our exmaple we know the underlying truth. Thus, we may compare predictions from the regularized model to the truth for different regularization parameters.

```
degree = 20    # degree for polynomial regression
alphas = [2 ** (-k) for k in range(5, 40)]    # regularization parameters

errors = np.zeros(len(alphas))

for idx, alpha in enumerate(alphas):

    # regression
    regression = lm.Ridge(alpha=alpha)
    transform = PolynomialFeatures(degree=degree).fit_transform
    regression.fit(transform(X_train), y_train)

    # get mean squared error for equispaced grid (same as for plotting)
    y_reg = regression.predict(transform(x.reshape(-1, 1)))
    y_true = truth(x)
    errors[idx] = metrics.mean_squared_error(y_reg, y_true)

# plot errors
```

(continues on next page)

(continued from previous page)

```
fig, ax = plt.subplots()
ax.semilogx(alphas, errors, '-m')
ax.set_xlabel('$\\alpha$')
ax.set_ylabel('error')
plt.show()
```



For α close to zero overfitting leads to large errors. For large α model parameters are close to zero, which leads to very bad data fitting and, thus, to large errors, too (*overregularization*). Between both ends there is a local minimum, yielding the optimal α.

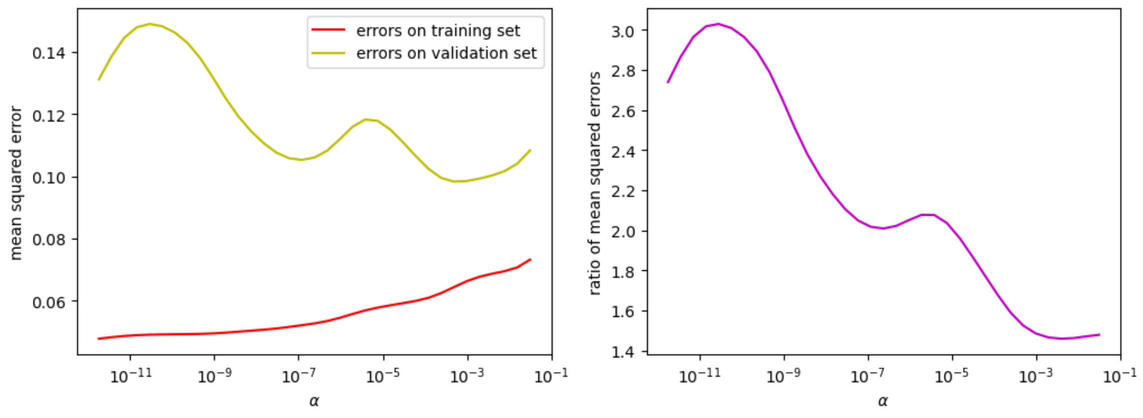
In practice we do not know the truth. But analogously to detecting overfitting we may find values for α, where overfitting vanishes. We simply have to start with very small α leading to overfitting. Then we increase α until training and validation data yield similar mean squared errors when compared to the hypothesis.

```
degree = 20    # degree for polynomial regression
alphas = [2 ** (-k) for k in range(5, 40)]    # regularization parameters

train_errors = np.zeros(len(alphas))
val_errors = np.zeros(len(alphas))

for idx, alpha in enumerate(alphas):

    # regression
    regression = lm.Ridge(alpha=alpha)
    transform = PolynomialFeatures(degree=degree).fit_transform
    regression.fit(transform(X_train), y_train)

    # predictions on subsets
    y_train_pred = regression.predict(transform(X_train))
    y_val_pred = regression.predict(transform(X_val))
```

(continues on next page)

(continued from previous page)

```
    # errors
    train_errors[idx] = metrics.mean_squared_error(y_train_pred, y_train)
    val_errors[idx] = metrics.mean_squared_error(y_val_pred, y_val)

# plot errors
fig, (ax_left, ax_right) = plt.subplots(1, 2, figsize=(12, 4))
ax_left.semilogx(alphas, train_errors, '-r', label='errors on training set')
ax_left.semilogx(alphas, val_errors, '-y', label='errors on validation set')
ax_left.set_xlabel('$\\alpha$')
ax_left.set_ylabel('mean squared error')
ax_left.legend()
ax_right.semilogx(alphas, val_errors / train_errors, '-m')
ax_right.set_xlabel('$\\alpha$')
ax_right.set_ylabel('ratio of mean squared errors')
plt.show()

# print errors
print('  training       validation')
print(np.stack((train_errors, val_errors), axis=1))
```



```
  training       validation
[[0.07326089 0.10832048]
 [0.07075901 0.10408418]
 [0.06954605 0.10167556]
 [0.06871829 0.10026634]
 [0.06773215 0.09926158]
 [0.06630425 0.09848887]
 [0.06443948 0.09830744]
 [0.06251508 0.09948163]
 [0.06100575 0.10238347]
 [0.06000687 0.10645098]
 [0.05927804 0.1108613 ]
 [0.05859948 0.11495806]
 [0.05785644 0.11780122]
 [0.0569407  0.11823884]
 [0.05579468 0.11589774]
 [0.05456294 0.11190603]
 [0.05352215 0.10822899]
 [0.05277435 0.10602967]
 [0.05218768 0.10528655]
 [0.05163829 0.10579957]
 [0.05114153 0.10762232]
 [0.0507495  0.11063545]
 [0.05043095 0.11450807]
 [0.05011846 0.11923569]
```

(continues on next page)

(continued from previous page)

```
[0.04980821 0.12506099]
[0.0495608  0.13166127]
[0.04941321 0.1379058 ]
[0.0493391  0.14279624]
[0.04929224 0.14614947]
[0.04923822 0.14819351]
[0.04915017 0.14892117]
[0.04899594 0.1478916 ]
[0.04873589 0.14449768]
[0.04834478 0.13855119]
[0.04785498 0.13111712]]
```

Note that this way only values for α leading to overfitting can be detected. But overregularization does not lead to large differences in the errors. Thus overregularization is indistinguishable from good fitting if only errors on training and validation sets are compared.

24.2.4 Example: Scaling is Important

Consider regularized linear regression (Ridge regression) with two features. This example is just for illustration; never use regularization if your model has only three parameters! Values of feature 1 are between 0 and 1, values of feature 2 are between 1900 and 2100. With

$$f_{\text{approx}}(x)=a_0+a_1\,x^{(1)}+a_2\,x^{(2)}$$

we have to solve

$$\frac{1}{n}\sum_{l=1}^{n}\left(f_{\text{approx}}(x_l)-y_l\right)^2+\alpha\,\frac{1}{3}\,\left(a_0^2+a_1^2+a_2^2\right)\to\min_{a_0,a_1,a_2},$$

where $(x_1,y_1),\dots,(x_n,y_n)$ are the training samples and α is the regularization parameter.

If the target variable y is small, say between -1 and 1, then a_1 is likely to take a value between -1 and 1, too. But a_2 will be much smaller than 1, say between $-\frac{1}{2000}$ and $\frac{1}{2000}$. Thus, a_2 has almost no influence on the penalty term for regularization. The result is, that feature 1 is suppressed by regularization (a_1 much smaller than without regularization) and feature 2 is left untouched (a_2 of same magnitude as without regularization).

Such imbalance should be avoided. Of course, one feature might be more relevant for explaining the data than other features. But here the imbalance stems from penalizing all features with the same factor α regardless of their range of values.

24.2.5 Other Regularization Methods

Next to adding squares of the model parameters there are several other choices for the penalty. Here we only consider two of them.

- *LASSO* (Least Absolute Shrinkage and Selection Operator):

$$\alpha\,\frac{1}{\mu}\sum_{\kappa=1}^{\mu}|a_\kappa|.$$

- *Elastic Net*:

$$\alpha\,\frac{1}{\mu}\left(\beta\sum_{\kappa=1}^{\mu}|a_\kappa|+(1-\beta)\sum_{\kappa=1}^{\mu}a_\kappa^2\right),$$

where $\beta\in[0,1]$ is a further hyperparameter, also known as mixing parameter. Solving minimization problems with LASSO penalty is numerically more challenging, but leads to hypotheses with only very few non-zero parameters (this is not obvious, but can rigorously be proven). Elastic Net penalties are a mixture of the standard penalty and the LASSO penalty, yielding numerically more tractable minimization problems, but still enforcing many parameters to be zero.

Regularized linear regression with LASSO and Elastic Net penalties is available in Scikit-Learn's `linear_model` module: `Lasso`[431], `ElasticNet`[432].

24.3 Worked Example: House Prices I

To test techniques for supervised learning discussed so far we train a model for predicting house prices in Germany. Inputs are properties of a house and of the plot of land it has been built on. Output is the selling price.

Training data exists in form of advertisements on specialized websites for finding a buyer for a house. In principle we could scrape data from such a website, but usually its not allowed by the website operator and we would have to write lots of code. Erdogan Seref[433] (unreachable in 2023) already did this job at www.immobilienscout24.de[434] and published the data set at www.kaggle.com[435] (unreachable in 2023) under a Attribution-NonCommercial-ShareAlike 4.0 International License[436].

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn.linear_model as linear_model
import sklearn.metrics as metrics
import sklearn.model_selection as model_selection
import sklearn.preprocessing as preprocessing
import sklearn.pipeline as pipeline

data_path = 'german_housing.csv'
```

24.3.1 The Data Set

At first we load the data set and try to get an overview of features and data quality.

```
data = pd.read_csv(data_path)
```

If a data frame has many columns Pandas by default does not show all columns. But we want to see all. Thus, we have to adjust the number of columns shown by setting corresponding option[437] to `None` (that is, unlimited).

```
pd.set_option('display.max_columns', None)
data.head(10)
```

```
   Unnamed: 0      Price                Type  Living_space     Lot  \
0           0   498000.0  Multiple dwelling        106.00   229.0
1           1   495000.0  Mid-terrace house        140.93   517.0
2           2   749000.0          Farmhouse        162.89    82.0
3           3   259000.0          Farmhouse        140.00   814.0
4           4   469000.0  Multiple dwelling        115.00   244.0
5           5  1400000.0  Mid-terrace house        310.00   860.0
6           6  3500000.0             Duplex        502.00  5300.0
7           7   630000.0             Duplex        263.00   406.0
8           8   364000.0             Duplex        227.00   973.0
```

(continues on next page)

[431] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html
[432] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html
[433] https://www.kaggle.com/scriptsultan
[434] https://www.immobilienscout24.de
[435] https://www.kaggle.com/scriptsultan/german-house-prices
[436] https://creativecommons.org/licenses/by-nc-sa/4.0
[437] https://pandas.pydata.org/pandas-docs/stable/user_guide/options.html#frequently-used-options

(continued from previous page)

```
9          9  1900000.0            Duplex       787.00   933.0

   Usable_area     Free_of_Relation  Rooms  Bedrooms  Bathrooms  Floors  \
0          NaN           01.10.2020    5.5       3.0        1.0     2.0
1        20.00           01.01.2021    6.0       3.0        2.0     NaN
2        37.62           01.07.2020    5.0       3.0        2.0     4.0
3          NaN   nach Vereinbarung    4.0       NaN        2.0     2.0
4          NaN               sofort    4.5       2.0        1.0     NaN
5       100.00               sofort    8.0       NaN        NaN     3.0
6       163.16      nach Absprache   13.0       NaN        4.0     NaN
7       118.00           01.04.2020   10.0       NaN        NaN     3.0
8        83.00      nach Absprache   10.0       4.0        4.0     2.0
9          NaN                  NaN   30.0       NaN        NaN     3.0

   Year_built Furnishing_quality  Year_renovated    Condition  \
0      2005.0             normal             NaN   modernized
1      1994.0              basic             NaN   modernized
2      2013.0                NaN             NaN  dilapidated
3      1900.0              basic          2000.0  fixer-upper
4      1968.0            refined          2019.0  refurbished
5      1969.0              basic             NaN   maintained
6      2004.0              basic             NaN  dilapidated
7      1989.0              basic             NaN   modernized
8      1809.0             normal          2015.0   modernized
9      1920.0              basic             NaN   modernized

           Heating            Energy_source   Energy_certificate  \
0  central heating                      Gas            available
1    stove heating                      NaN  not required by law
2    stove heating   Fernwärme, Bioenergie            available
3  central heating                    Strom            available
4  central heating                       Öl            available
5              NaN                       Öl            available
6    stove heating   Erdwärme, Holzpellets            available
7    stove heating                      Gas            available
8  central heating                    Strom            available
9    stove heating    Gas, Fernwärme-Dampf            available

   Energy_certificate_type  Energy_consumption Energy_efficiency_class  \
0       demand certificate                 NaN                       D
1                      NaN                 NaN                     NaN
2       demand certificate                 NaN                       B
3       demand certificate                 NaN                       G
4       demand certificate                 NaN                       F
5  consumption certificate                 NaN                     NaN
6  consumption certificate                35.9                       A
7       demand certificate                 NaN                       E
8  consumption certificate               183.1                       F
9       demand certificate                 NaN                       D

               State                 City                     Place  Garages  \
0  Baden-Württemberg        Bodenseekreis              Bermatingen      2.0
1  Baden-Württemberg      Konstanz (Kreis)                   Engen      7.0
2  Baden-Württemberg     Esslingen (Kreis)              Ostfildern      1.0
3  Baden-Württemberg      Waldshut (Kreis)  Bonndorf im Schwarzwald      1.0
4  Baden-Württemberg     Esslingen (Kreis)  Leinfelden-Echterdingen      1.0
5  Baden-Württemberg            Stuttgart                      Süd      2.0
6  Baden-Württemberg     Göppingen (Kreis)                   Wangen      7.0
7  Baden-Württemberg  Freiburg im Breisgau               Munzingen      2.0
8  Baden-Württemberg             Enzkreis               Neuenbürg      8.0
9  Baden-Württemberg             Mannheim                 Rheinau     12.0
```

(continues on next page)

(continued from previous page)

```
    Garagetype
0  Parking lot
1  Parking lot
2       Garage
3       Garage
4       Garage
5       Garage
6  Parking lot
7       Garage
8  Parking lot
9  Parking lot
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10552 entries, 0 to 10551
Data columns (total 26 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Unnamed: 0               10552 non-null  int64  
 1   Price                    10552 non-null  float64
 2   Type                     10150 non-null  object 
 3   Living_space             10552 non-null  float64
 4   Lot                      10552 non-null  float64
 5   Usable_area              5568 non-null   float64
 6   Free_of_Relation         6983 non-null   object 
 7   Rooms                    10552 non-null  float64
 8   Bedrooms                 6878 non-null   float64
 9   Bathrooms                8751 non-null   float64
 10  Floors                   7888 non-null   float64
 11  Year_built               9858 non-null   float64
 12  Furnishing_quality       7826 non-null   object 
 13  Year_renovated           5349 non-null   float64
 14  Condition                10229 non-null  object 
 15  Heating                  9968 non-null   object 
 16  Energy_source            9325 non-null   object 
 17  Energy_certificate       9797 non-null   object 
 18  Energy_certificate_type  7026 non-null   object 
 19  Energy_consumption       2433 non-null   float64
 20  Energy_efficiency_class  5733 non-null   object 
 21  State                    10551 non-null  object 
 22  City                     10551 non-null  object 
 23  Place                    10262 non-null  object 
 24  Garages                  8592 non-null   float64
 25  Garagetype               8592 non-null   object 
dtypes: float64(12), int64(1), object(13)
memory usage: 2.1+ MB
```

We should drop irrelevant columns and adjust data types.

- `Unnamed: 0`: Seems to be an integer index. We don't need it, so drop it.
- `Price`: This is our target variable.
- `Type`: An important column, because house prices are likely to depend on the type of house. We should convert this to categorical type.
- `Living_space` and `Lot`: Important features, keep them.
- `Usable_area`: Likely to have influence on the selling price, but available only for half the samples. If we want to use this for regression, we would have to drop half the training samples. Alternatively we could impute

values, but it's very hard to guess usable area from other features. We should drop the column.

- `Free_of_Relation`: Not related to the selling price. Drop it.
- `Rooms`, `Bedrooms`, `Bathrooms`: Should have influence on prices, but not available for all samples. For the moment we keep all three columns. Later we should have a look on correlations between the three columns and possibly only keep the first one, which is available for all samples.
- `Floors`: Important feature, keep it.
- `Year_built`: Important feature, keep it.
- `Furnishing_quality`: Important, convert to categorical and keep.
- `Year_renovated`: Important, but half the data is missing. There is good chance that missing values indicate that there the house has not been renovated until today. Thus, a reasonable fill value is the year of construction.
- `Condition`: Important, convert to categorical and keep.
- `Heating` and `Energy_source`: Could be important, convert to categorical and keep.
- `Energy_certificate`, `Energy_certificate_type`, `Energy_consumption`: The first contains more or less only the value `'available'` (since energy certificates are required by law). The second is irrelevant and the third is missing for most samples. Drop them all.
- `Energy_efficiency_class`: Likely to have influence on the selling price, although classification procedure is very unreliable in practice. Keep and convert to categorical.
- `State`, `City`, `Place`: Geolocation surely influences selling prices. But it's hard to use location data for regression. For the moment we keep these columns.
- `Garages`: Could be important, keep.
- `Garagetype`: If we keep `Garages` then we also have to keep this column. Convert to categorical and rename to `Garage_type` to fit naming convention used for the other columns.

```
data = data.drop(columns=['Unnamed: 0', 'Usable_area', 'Free_of_Relation',
                          'Energy_certificate', 'Energy_certificate_type',
 ↪'Energy_consumption'])

data['Type'] = data['Type'].astype('category')
data['Furnishing_quality'] = data['Furnishing_quality'].astype('category')
data['Condition'] = data['Condition'].astype('category')
data['Heating'] = data['Heating'].astype('category')
data['Energy_source'] = data['Energy_source'].astype('category')
data['Energy_efficiency_class'] = data['Energy_efficiency_class'].astype('category
 ↪')
data['Garagetype'] = data['Garagetype'].astype('category')

data = data.rename(columns={'Garagetype': 'Garage_type'})

nan_mask = data['Year_renovated'].isna()
data.loc[nan_mask, 'Year_renovated'] = data.loc[nan_mask, 'Year_built']
```

Categorical columns `Furnishing_quality`, `Condition` and `Energy_efficiency_class` should have a natural ordering, which should be represented by the data type.

```
print(data['Furnishing_quality'].cat.categories)
print(data['Condition'].cat.categories)
print(data['Energy_efficiency_class'].cat.categories)
```

```
Index(['basic', 'luxus', 'normal', 'refined'], dtype='object')
Index(['as new', 'by arrangement', 'dilapidated', 'first occupation',
       'first occupation after refurbishment', 'fixer-upper', 'maintained',
       'modernized', 'refurbished', 'renovated'],
```

(continues on next page)

(continued from previous page)

```
      dtype='object')
Index([' A ', ' A+ ', ' B ', ' C ', ' D ', ' E ', ' F ', ' G ', ' H '], dtype=
 ↪'object')
```

We should rename same categories and sort them as good as possible.

```
data['Furnishing_quality'] = data['Furnishing_quality'] \
    .cat.rename_categories({'luxus': 'luxury'}) \
    .cat.reorder_categories(['basic', 'normal', 'refined', 'luxury'])

data['Condition'] = data['Condition'].cat.reorder_categories([
    'first occupation',
     'first occupation after refurbishment',
     'as new',
     'maintained',
     'renovated',
     'modernized',
     'refurbished',
     'by arrangement',
     'fixer-upper',
     'dilapidated'
])

data['Energy_efficiency_class'] = data['Energy_efficiency_class'] \
    .cat.rename_categories({
        ' A ': 'A',
        ' A+ ': 'A+',
        ' B ': 'B',
        ' C ': 'C',
        ' D ': 'D',
        ' E ': 'E',
        ' F ': 'F',
        ' G ': 'G',
        ' H ': 'H'
    }) \
    .cat.reorder_categories(['A+', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H'])
```

Now let's see how many complete samples we have.

```
len(data.dropna())
```

```
1591
```

That's very few. So we should drop some columns with many missing values.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10552 entries, 0 to 10551
Data columns (total 20 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Price                      10552 non-null  float64
 1   Type                       10150 non-null  category
 2   Living_space               10552 non-null  float64
 3   Lot                        10552 non-null  float64
 4   Rooms                      10552 non-null  float64
 5   Bedrooms                   6878 non-null   float64
```

(continues on next page)

(continued from previous page)

```
 6   Bathrooms               8751 non-null   float64 
 7   Floors                  7888 non-null   float64 
 8   Year_built              9858 non-null   float64 
 9   Furnishing_quality      7826 non-null   category
 10  Year_renovated          10211 non-null  float64 
 11  Condition               10229 non-null  category
 12  Heating                 9968 non-null   category
 13  Energy_source           9325 non-null   category
 14  Energy_efficiency_class 5733 non-null   category
 15  State                   10551 non-null  object  
 16  City                    10551 non-null  object  
 17  Place                   10262 non-null  object  
 18  Garages                 8592 non-null   float64 
 19  Garage_type             8592 non-null   category
dtypes: category(7), float64(10), object(3)
memory usage: 1.1+ MB
```

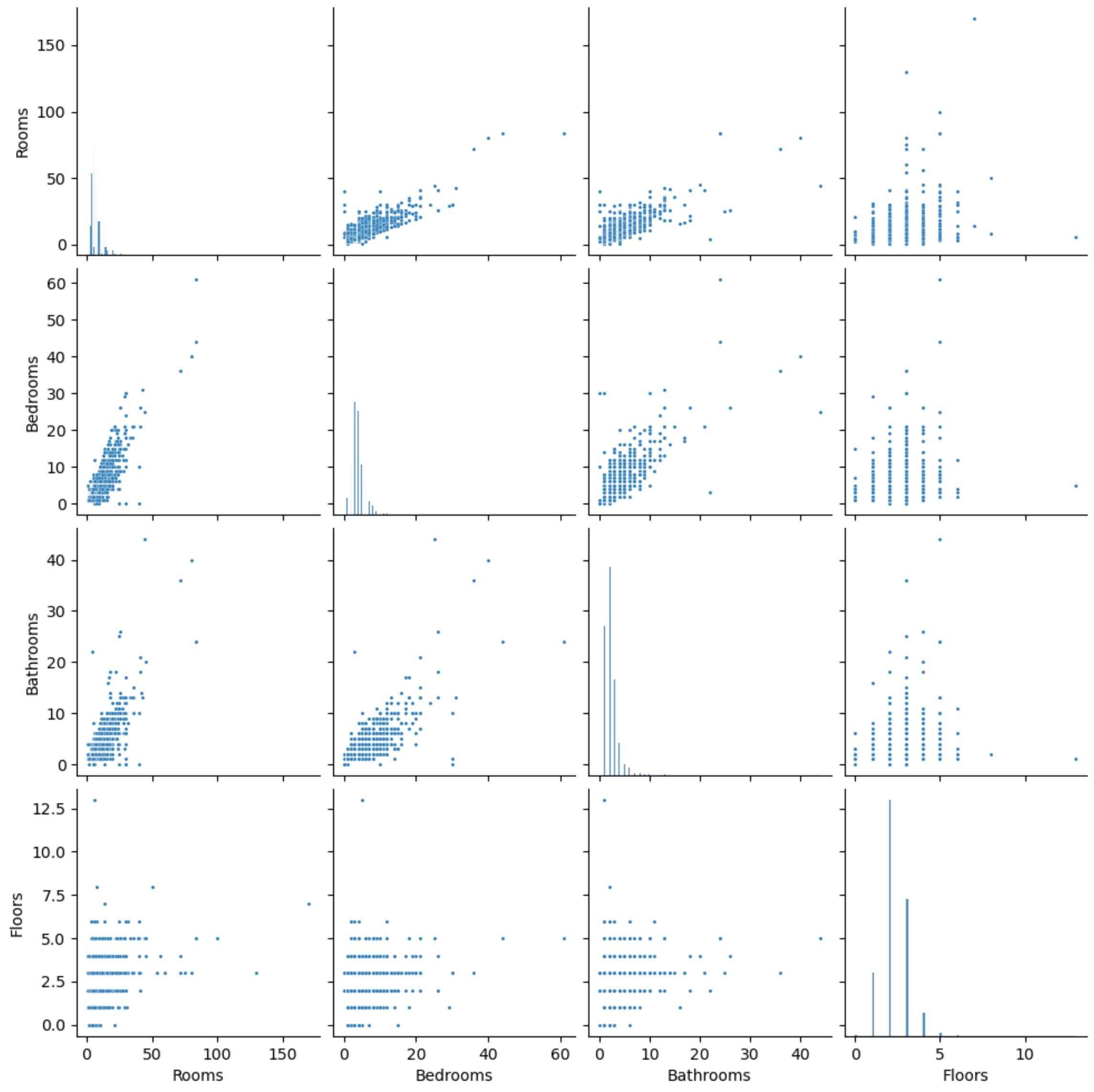
`Energy_efficiency_class` is relatively unreliable and not too important for selling prices.

```
len(data.drop(columns=['Energy_efficiency_class']).dropna())
```

```
2615
```

Better, but not good. The `Bedrooms` column has many missing values, too, and it's likely to be correlated to `Rooms`. So let's look at correlations between `Rooms`, `Bedrooms`, `Bathrooms`, `Floors`.

```
sns.pairplot(data[['Rooms', 'Bedrooms', 'Bathrooms', 'Floors']], plot_kws={"s": 5}
↪)
plt.show()
```



`Floors` is not correlated to the other columns, so keep it. `Bedrooms` show correlation to `Rooms` and `Bathrooms`, so drop `Bedrooms`. `Bathroom` shows some correlation to `Rooms`. Wether to drop `Bathrooms` should be decided by the increase in sample counts.

```
len(data.drop(columns=['Energy_efficiency_class', 'Bedrooms']).dropna())
```

```
3174
```

```
len(data.drop(columns=['Energy_efficiency_class', 'Bedrooms', 'Bathrooms']).
↪dropna())
```

```
3479
```

We should keep `Bathrooms`, because dropping it only yields 300 more samples while neglecting possibly important information. Note that the number of bath rooms can be regarded as a measure for overall furnishing quality. Thus, there should be some correlation to `Furnishing_quality`.

```
data.groupby('Furnishing_quality')['Bathrooms'].mean()
```

```
Furnishing_quality
basic      2.207496
normal     2.363720
refined    1.826739
luxury     2.778761
Name: Bathrooms, dtype: float64
```

In addition, judging about furnishing quality of a house is highly subjective. Thus, we should drop the column to get more samples without missing data.

```
len(data.drop(columns=['Energy_efficiency_class', 'Bedrooms', 'Furnishing_quality
↪']).dropna())
```

```
4526
```

The `Energy_source` is another candidate for dropping, because it has more than 1000 missing values and its influence on selling prices should be rather low.

```
for cat in data['Energy_source'].cat.categories:
    print(cat)
```

```
 Bioenergie
 Erdgas leicht
 Erdgas leicht, Erdgas schwer
 Erdgas schwer
 Erdgas schwer, Bioenergie
 Erdgas schwer, Holz
 Erdwärme
 Erdwärme, Fernwärme
 Erdwärme, Gas
 Erdwärme, Holzpellets
 Erdwärme, Solar
 Erdwärme, Solar, Holzpellets, Holz
 Erdwärme, Solar, Umweltwärme
 Erdwärme, Strom
 Erdwärme, Umweltwärme
 Fernwärme
 Fernwärme, Bioenergie
 Fernwärme, Flüssiggas
 Fernwärme, Nahwärme, KWK fossil
 Fernwärme-Dampf
 Flüssiggas
 Flüssiggas, Holz
 Gas
 Gas, Bioenergie
 Gas, Fernwärme
 Gas, Fernwärme-Dampf
 Gas, Holz
 Gas, Holz-Hackschnitzel
 Gas, KWK fossil
 Gas, Kohle, Holz
 Gas, Strom
 Gas, Strom, Holz
 Gas, Strom, Kohle, Holz
 Gas, Wasserenergie
 Gas, Öl
 Gas, Öl, Holz
 Gas, Öl, Kohle
 Gas, Öl, Kohle, Holz
```

(continues on next page)

(continued from previous page)

```
Gas, Öl, Strom
Holz
Holz, Bioenergie
Holz-Hackschnitzel
Holzpellets
Holzpellets, Gas
Holzpellets, Gas, Öl
Holzpellets, Holz
Holzpellets, Holz-Hackschnitzel
Holzpellets, Kohle, Holz
Holzpellets, Strom
Holzpellets, Öl
KWK erneuerbar
KWK fossil
KWK regenerativ
Kohle
Kohle, Holz
Kohle/Koks
Nahwärme
Solar
Solar, Bioenergie
Solar, Erdgas schwer
Solar, Gas
Solar, Gas, Holz
Solar, Gas, Strom
Solar, Gas, Strom, Holz
Solar, Gas, Wasserenergie
Solar, Gas, Öl
Solar, Gas, Öl, Holz
Solar, Holz
Solar, Holz-Hackschnitzel
Solar, Holzpellets
Solar, Holzpellets, Holz
Solar, Holzpellets, Strom
Solar, Holzpellets, Öl
Solar, Strom
Solar, Strom, Bioenergie
Solar, Umweltwärme
Solar, Öl
Solar, Öl, Bioenergie
Solar, Öl, Holz
Solar, Öl, Holz-Hackschnitzel
Solar, Öl, Strom
Solar, Öl, Strom, KWK fossil
Strom
Strom, Bioenergie
Strom, Flüssiggas
Strom, Holz
Strom, Holz-Hackschnitzel
Strom, Kohle
Strom, Kohle, Holz
Strom, Umweltwärme
Umweltwärme
Wasserenergie
Windenergie
Wärmelieferung
Öl
Öl, Bioenergie
Öl, Fernwärme
Öl, Holz
Öl, Kohle
```

(continues on next page)

(continued from previous page)

```
Öl, Kohle, Holz
Öl, Strom
Öl, Strom, Holz
Öl, Strom, Kohle, Holz
Öl, Umweltwärme
```

Values are very diverse and hard to preprocess for regression. We would have to convert the column to several boolean columns. In addition, some grouping would be necessary (`Holz` is a subcategory of `Bioenergie` and so on).

```
len(data.drop(columns=['Energy_efficiency_class', 'Bedrooms', 'Furnishing_quality
↪', 'Energy_source']).dropna())
```

```
4854
```

Now we have almost 5000 complete samples. Should be a good compromise between completeness and level of detail.

```
data = data.drop(columns=['Energy_efficiency_class', 'Bedrooms', 'Furnishing_
↪quality', 'Energy_source'])
data = data.dropna()
```

24.3.2 Outliers and Further Preprocessing

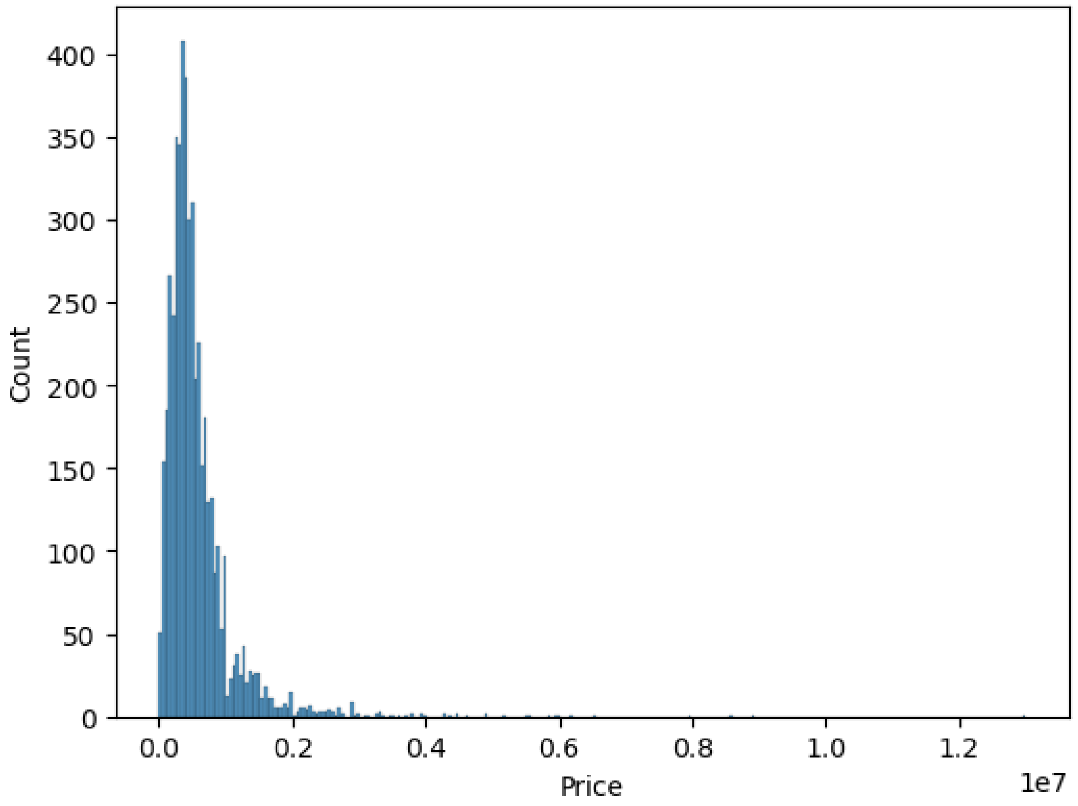
Now that we have a cleaned data set we should remove outliers. The simplest method of detecting outliers is to look at the ranges of all feature. With `describe` we get a first overview for numerical features.

```
data.describe()
```

```
              Price  Living_space           Lot        Rooms    Bathrooms  \
count  4.854000e+03   4854.000000   4854.000000  4854.000000  4854.000000
mean   5.739566e+05    209.305740   1240.636904     7.051504     2.316028
std    5.880211e+05    118.252688   3806.518099     3.834865     1.595327
min    0.000000e+00      0.000000      0.000000     1.000000     0.000000
25%    2.800000e+05    135.000000    401.000000     5.000000     1.000000
50%    4.400000e+05    180.000000    675.000000     6.000000     2.000000
75%    6.850000e+05    248.000000   1042.000000     8.000000     3.000000
max    1.300000e+07   1742.240000 143432.000000    84.000000    26.000000

            Floors   Year_built  Year_renovated      Garages
count  4854.000000  4854.000000     4854.000000  4854.000000
mean      2.256696  1964.252369     1995.626700     2.518541
std       0.776769    49.065052       35.389067     2.719901
min       0.000000  1430.000000     1430.000000     1.000000
25%       2.000000  1950.000000     1991.000000     1.000000
50%       2.000000  1974.000000     2008.000000     2.000000
75%       3.000000  1997.750000     2016.000000     3.000000
max       8.000000  2021.000000     2206.000000    65.000000
```

`Price` Column

```
sns.histplot(data['Price'])
plt.show()
```

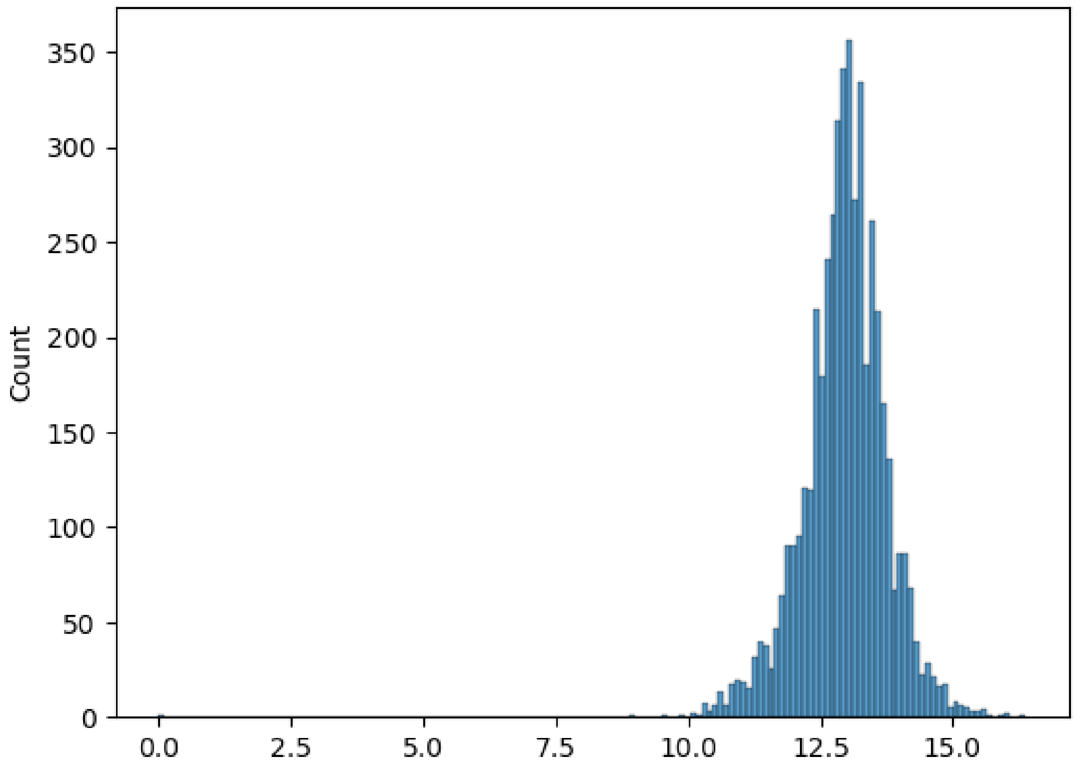


There are only very few high prices and price distribution concentrates on low prices. If the target variable has wide range, but most samples concentrate on a small portion of the range, then 'learning' the target is much more difficult than for more uniformly distributed data.

A common trick is to use nonlinear scaling. Especially for market prices it is known from experience that they follow a log-normal distribution[438], that is, after applying the logarithm we see a normal distribution. Before applying the logarithm we should drop samples with zeros in the `Price` column to avoid undefined results. A price of zero indicates that the seller did not provide a price in the advertisement. Thus, dropping such sample even is a good idea if we wouldn't want to apply the logarithm.

```
data = data.loc[data['Price'] > 0, :]
sns.histplot(np.log(data['Price'].to_numpy()))
plt.show()
```

[438] https://en.wikipedia.org/wiki/Log-normal_distribution



There seem to be same very small values.

```
data.loc[data['Price'] <= np.exp(7), :]
```

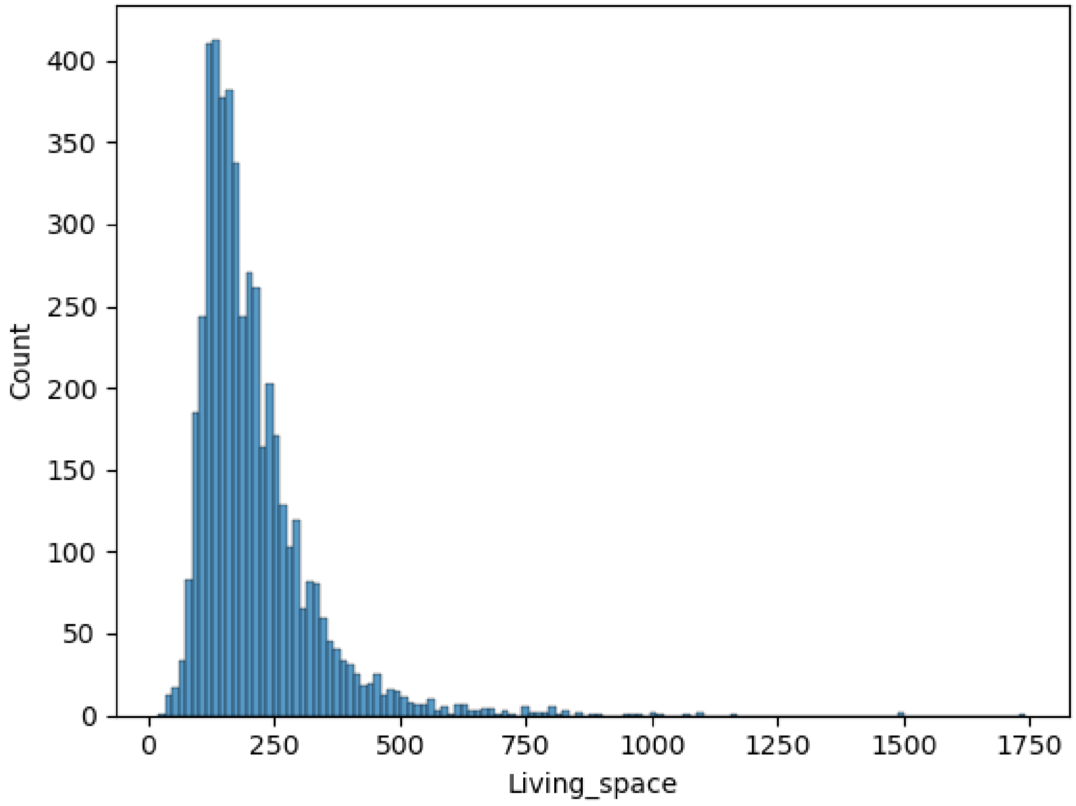
```
      Price    Type  Living_space     Lot  Rooms  Bathrooms  Floors  \
6987    1.0  Duplex         459.0  2742.0   23.0        8.0     3.0   

      Year_built  Year_renovated   Condition        Heating  \
6987      1957.0          1957.0  refurbished  stove heating   

                   State              City   Place  Garages Garage_type  
6987  Nordrhein-Westfalen  Märkischer Kreis  Altena      1.0      Garage  
```

Those samples should be dropped because house prices below $e^7 \approx 1000$ EUR are very uncommon.

```
data['Price'] = np.log(data['Price'].to_numpy())
data = data.loc[data['Price'] > 7, :]
```

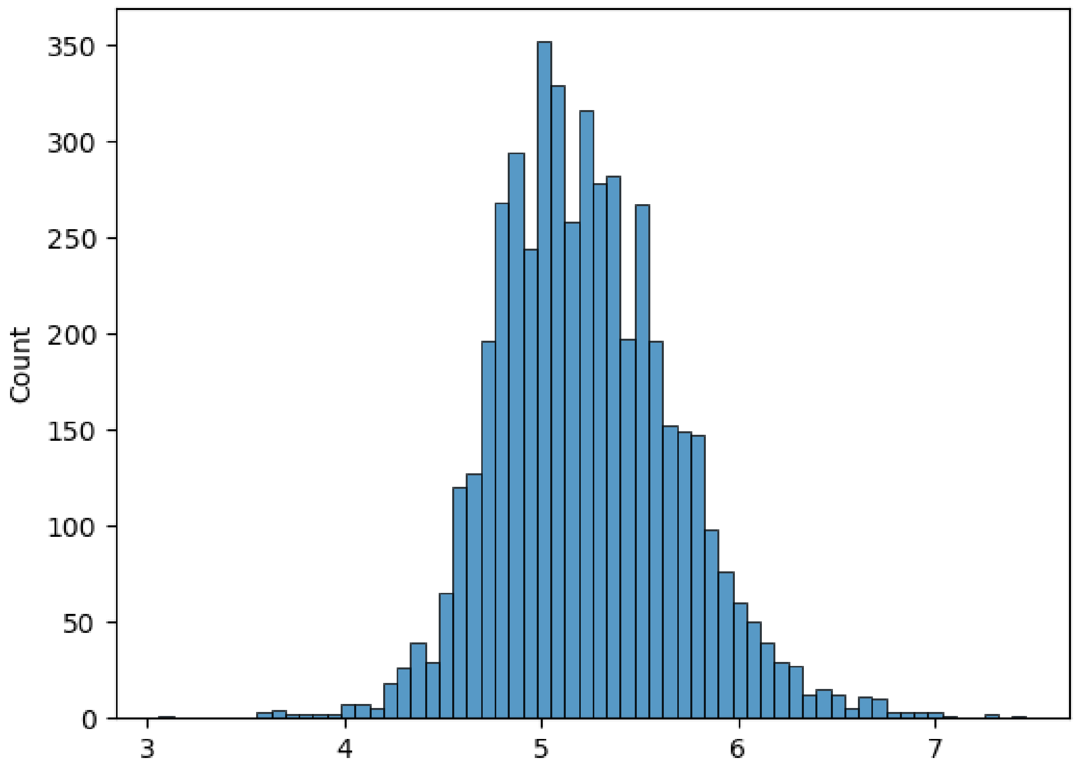
`Living_space` Column

```
sns.histplot(data['Living_space'])
plt.show()
```



Same here as for `Price`.

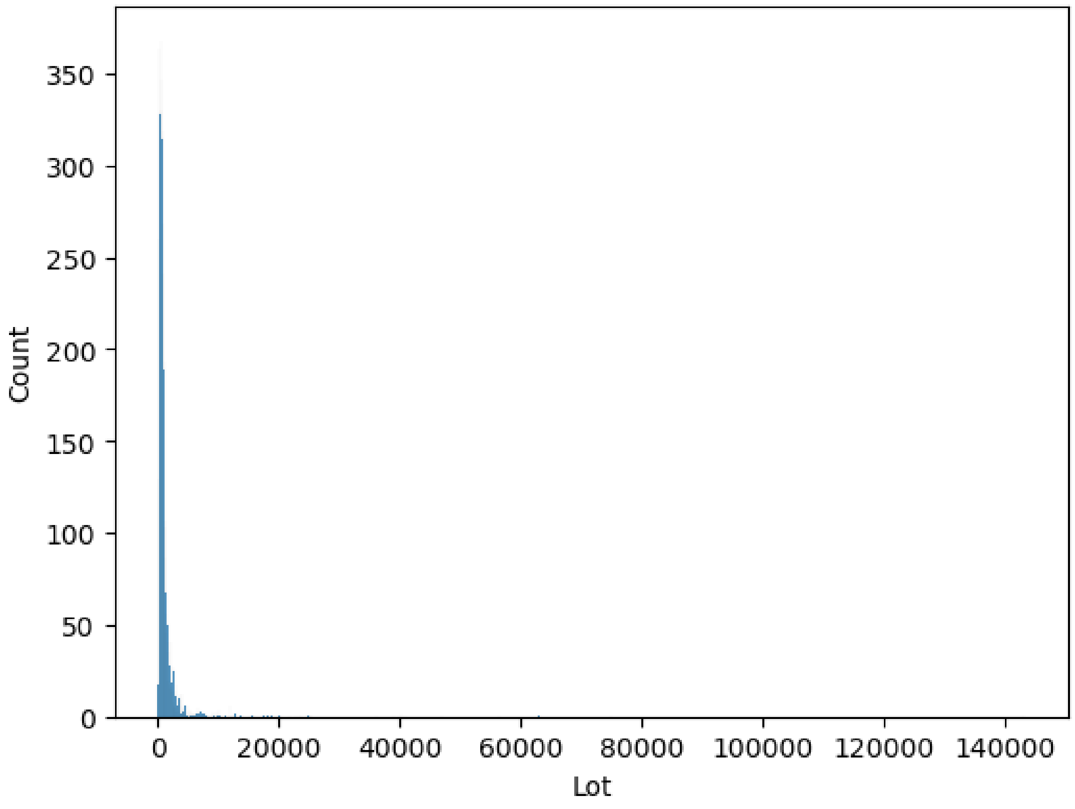
```
data = data.loc[data['Living_space'] > 0, :]
sns.histplot(np.log(data['Living_space'].to_numpy()))
plt.show()
```



```
data['Living_space'] = np.log(data['Living_space'].to_numpy())
```

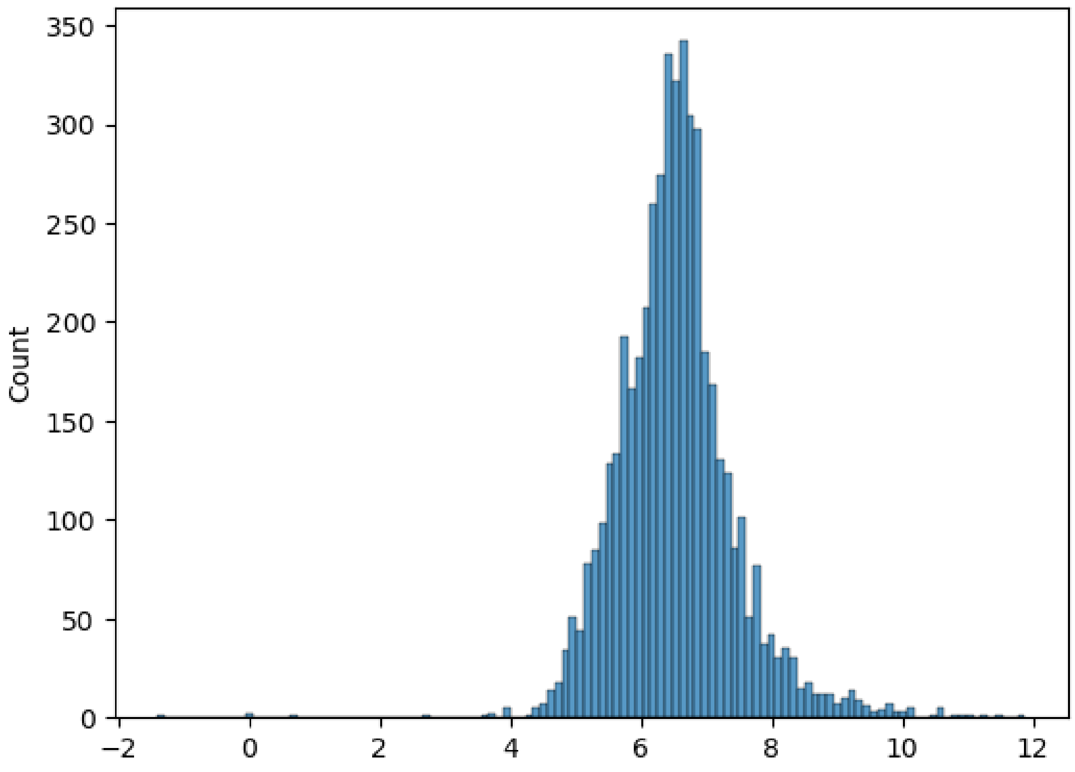
`Lot` Column

```
sns.histplot(data['Lot'])
plt.show()
```



Same here as for `Price` again.

```
data = data.loc[data['Lot'] > 0, :]
sns.histplot(np.log(data['Lot'].to_numpy()))
plt.show()
```



```
data.loc[data['Lot'] <= np.exp(2), :]
```

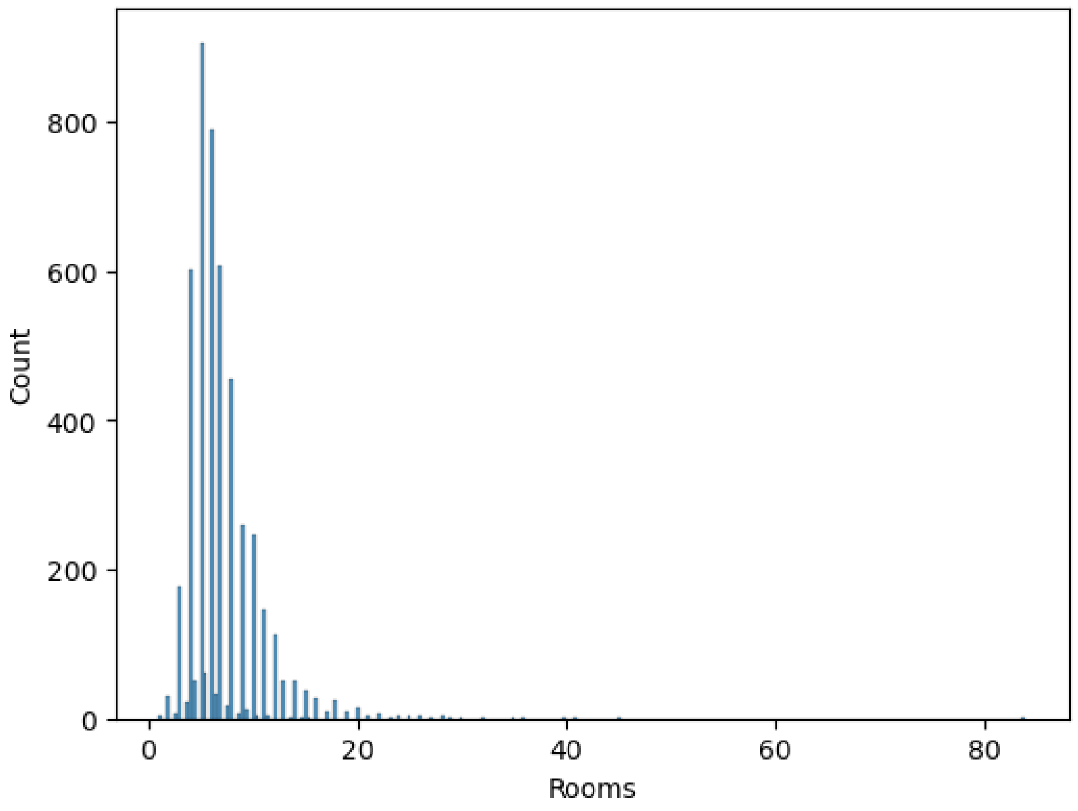
```
          Price                    Type  Living_space   Lot  Rooms  Bathrooms  \
722   11.492723      Mid-terrace house      4.491441  0.25    4.0        1.0
1876  12.860362   Residential property      5.342334  1.00    5.0        1.0
6435  11.707834                 Duplex      5.056246  1.96    5.0        2.0
7887  12.594731                 Duplex      5.857933  1.00    8.0        5.0

      Floors  Year_built  Year_renovated        Condition  \
722      3.0      1900.0          1900.0       modernized
1876     1.0      2019.0          2019.0  first occupation
6435     3.0      1905.0          1981.0      refurbished
7887     3.0      1965.0          2019.0       modernized

                  Heating               State               City     Place  \
722   wood-pellet heating   Baden-Württemberg   Main-Tauber-Kreis     Ahorn
1876        stove heating              Bayern  Regensburg (Kreis)     Hemau
6435        stove heating  Nordrhein-Westfalen      Höxter (Kreis)    Höxter
7887        stove heating     Rheinland-Pfalz     Neuwied (Kreis)  Dierdorf

      Garages          Garage_type
722       1.0               Garage
1876      1.0   Outside parking lot
6435      3.0   Outside parking lot
7887      1.0   Outside parking lot
```

Lot size below $e^2 < 8$ m^2 is very unlikely.

```
data['Lot'] = np.log(data['Lot'].to_numpy())
data = data.loc[data['Lot'] > 2, :]
```

`Rooms` Column

```
sns.histplot(data['Rooms'])
plt.show()
```



```
data.loc[data['Rooms'] >= 30, :]
```

```
          Price                 Type  Living_space       Lot  Rooms  \
1863  14.430696  Residential property      6.371612  7.578145   41.0
4322  14.346139       Single dwelling      6.745236  7.783224   30.0
4346  15.150512                Duplex      6.690842  6.311735   36.0
4557  12.896717                Duplex      6.653534  7.970395   36.0
4614  13.906265                Duplex      7.306531  8.234034   84.0
4615  13.906265                Duplex      7.306531  8.234034   84.0
6895  13.639966                Duplex      6.516193  7.090077   32.0
7397  13.963931                Duplex      7.057898  6.952729   45.0
7904  13.805460                Duplex      6.682109  8.505121   35.0
8705  14.506155              Bungalow      7.003065  8.575462   40.0

      Bathrooms  Floors  Year_built  Year_renovated  \
1863       21.0     3.0      1907.0          2019.0
4322        6.0     3.0      2009.0          2009.0
4346       13.0     5.0      1961.0          2016.0
4557       15.0     3.0      1970.0          2000.0
4614       24.0     5.0      1991.0          2018.0
4615       24.0     5.0      1989.0          2018.0
6895        8.0     3.0      2003.0          2003.0
7397       20.0     4.0      1950.0          1950.0
7904       13.0     3.0      2004.0          2019.0
8705        0.0     3.0      2012.0          2012.0

                         Condition        Heating  \
1863                    maintained  stove heating
4322                   dilapidated  stove heating
4346                    modernized  stove heating
4557                    maintained  stove heating
4614                    modernized  stove heating
```

(continues on next page)

(continued from previous page)

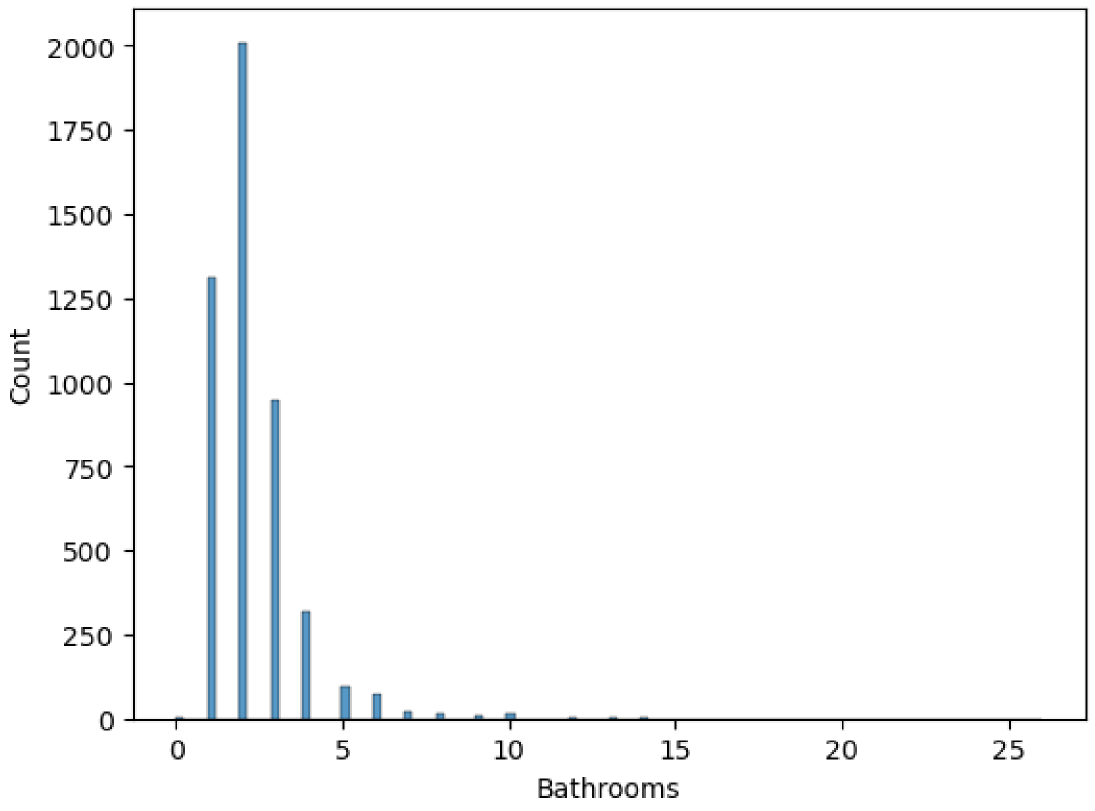
```
4615                          modernized     stove heating
6895                         dilapidated  electric heating
7397                          maintained         heat pump
7904  first occupation after refurbishment     stove heating
8705                         fixer-upper         heat pump

                       State                    City       Place  Garages  \
1863                  Bayern        Eichstätt (Kreis)  Dollnstein     16.0
4322                  Hessen  Limburg-Weilburg (Kreis)     Hadamar      4.0
4346                  Hessen        Frankfurt am Main   Rödelheim      3.0
4557  Mecklenburg-Vorpommern           Demmin (Kreis)     Rosenow      1.0
4614  Mecklenburg-Vorpommern           Demmin (Kreis)     Kruckow     16.0
4615  Mecklenburg-Vorpommern           Demmin (Kreis)     Kruckow     16.0
6895     Nordrhein-Westfalen            Gelsenkirchen      Hassel      8.0
7397     Nordrhein-Westfalen            Gelsenkirchen       Horst     12.0
7904         Rheinland-Pfalz    Trier-Saarburg (Kreis)    Saarburg     15.0
8705         Rheinland-Pfalz       Vulkaneifel (Kreis)  Gerolstein     40.0

              Garage_type
1863  Outside parking lot
4322  Outside parking lot
4346               Garage
4557  Outside parking lot
4614  Outside parking lot
4615  Outside parking lot
6895          Parking lot
7397  Outside parking lot
7904  Outside parking lot
8705          Parking lot
```

There are only very few sample with high number of rooms. There is no chance to get good predictions from those few samples.

```
data = data.loc[data['Rooms'] < 30, :]
```

`Bathrooms` Column

```
sns.histplot(data['Bathrooms'])
plt.show()
```



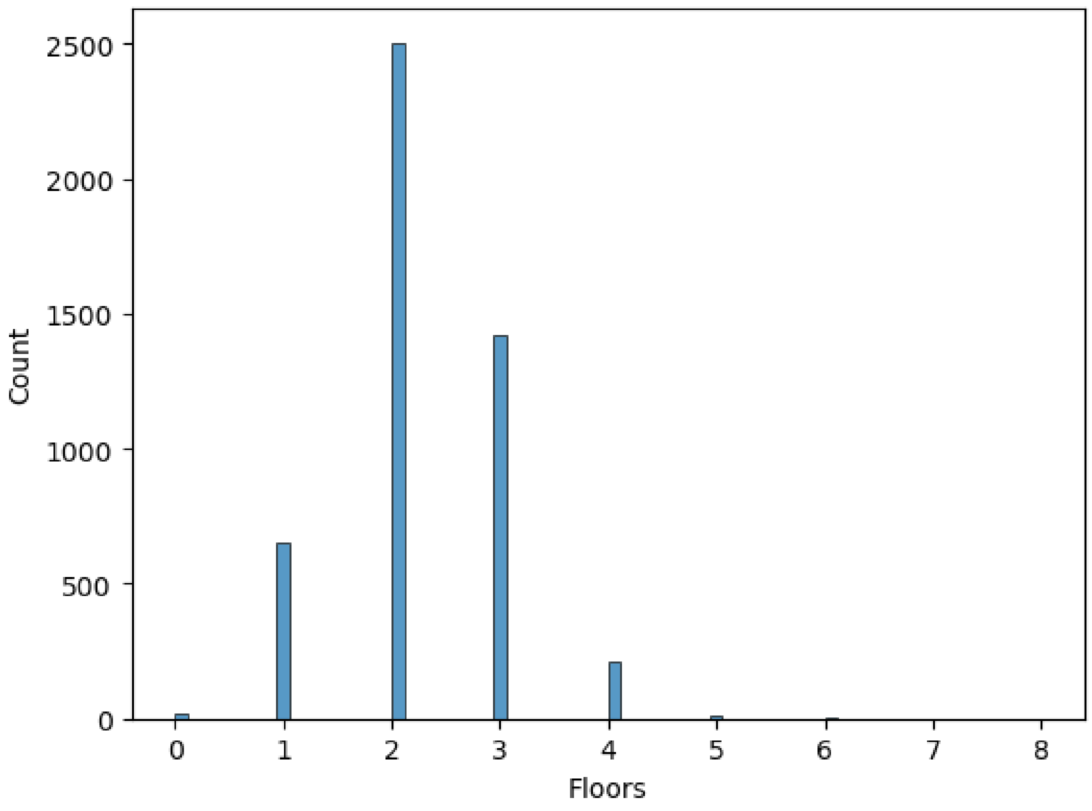
```
data.loc[data['Bathrooms'] >= 15, :]
```

```
           Price               Type  Living_space       Lot  Rooms  Bathrooms  \
3270   15.264780             Duplex      7.462927  7.693605   26.0       26.0
6006   13.457406           Bungalow      6.514713  8.750366   25.0       25.0
7224   14.038654             Duplex      6.476972  6.063785   18.0       18.0
10088  12.037654  Multiple dwelling      4.499810  5.777652    4.0       22.0

       Floors  Year_built  Year_renovated   Condition             Heating  \
3270      4.0      1994.0          1994.0  modernized           heat pump
6006      3.0      1874.0          2016.0   renovated  cogeneration units
7224      4.0      1962.0          1962.0  modernized       stove heating
10088     2.0      1961.0          2000.0  modernized           heat pump

                     State                         City                Place  \
3270           Brandenburg       Teltow-Fläming (Kreis)  Blankenfelde-Mahlow
6006         Niedersachsen    Lüchow-Dannenberg (Kreis)               Küsten
7224   Nordrhein-Westfalen              Herford (Kreis)              Herford
10088   Schleswig-Holstein  Schleswig-Flensburg (Kreis)                 Tarp

       Garages          Garage_type
3270      26.0           Duplex lot
6006      30.0  Outside parking lot
7224       5.0               Garage
10088      1.0  Outside parking lot
```

```
data = data.loc[data['Bathrooms'] < 15, :]
```

`Floors` Column

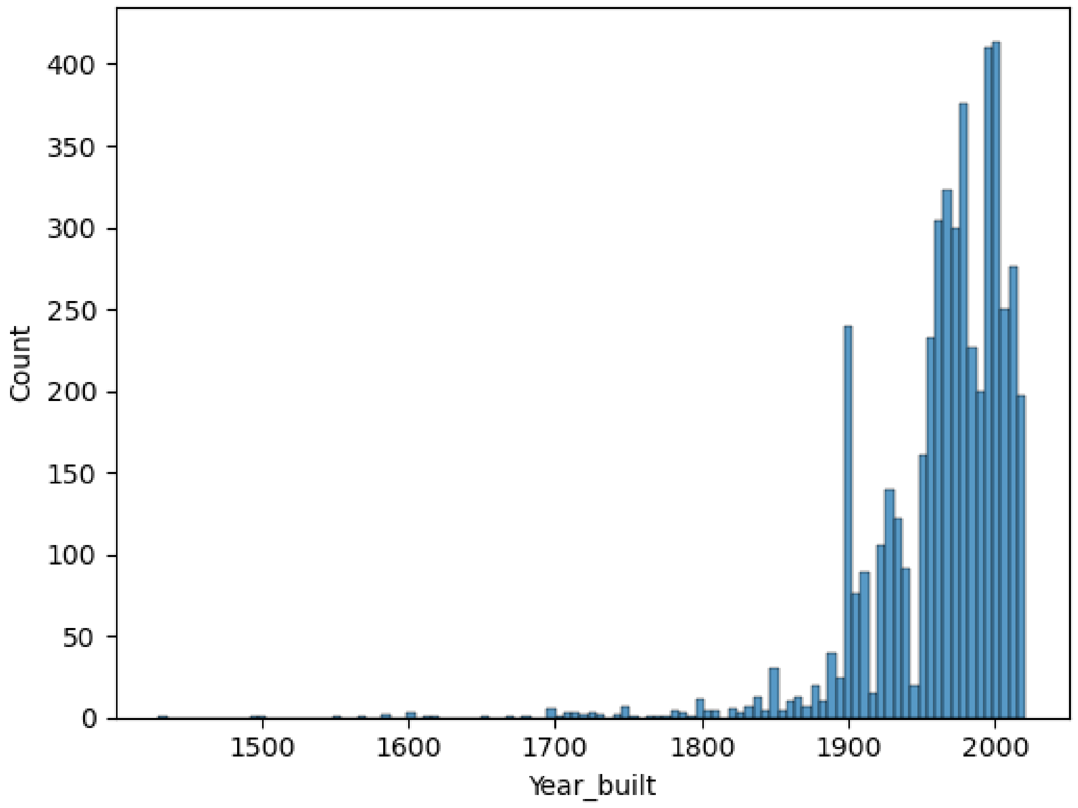
```
sns.histplot(data['Floors'])
plt.show()
```



Nothing to do here.

`Year_built` Column

```
sns.histplot(data['Year_built'])
plt.show()
```



```
data.loc[data['Year_built'] <= 1500, :]
```

```
          Price          Type  Living_space        Lot  Rooms  Bathrooms  \
1311  14.580978  Corner house      6.182085  11.522876   13.0        7.0   
1458  14.077875        Duplex      6.522093   6.361302   19.0        9.0   
8174  12.203570      Bungalow      5.703782   5.634790   10.0        2.0   

      Floors  Year_built  Year_renovated    Condition             Heating  \
1311     3.0      1430.0          1430.0  fixer-upper       stove heating   
1458     3.0      1500.0          2014.0   modernized  underfloor heating   
8174     4.0      1492.0          1492.0  refurbished       stove heating   

                State                         City             Place  \
1311           Bayern  Berchtesgadener Land (Kreis)  Marktschellenberg   
1458           Bayern                      Ansbach             Stadt   
8174  Rheinland-Pfalz          Bad Kreuznach (Kreis)      Bad Kreuznach   

      Garages Garage_type  
1311      2.0     Carport  
1458      9.0     Carport  
8174      2.0      Garage  
```

```
data = data.loc[data['Year_built'] > 1500, :]
```

Values above 2020 obviously are wrong (data set is from 2020).

```
data.loc[data['Year_built'] > 2020, :]
```

```
          Price             Type  Living_space       Lot  Rooms  Bathrooms  \
2437  13.623139  Single dwelling      4.897840  5.609472    4.5        1.0   
```

(continues on next page)

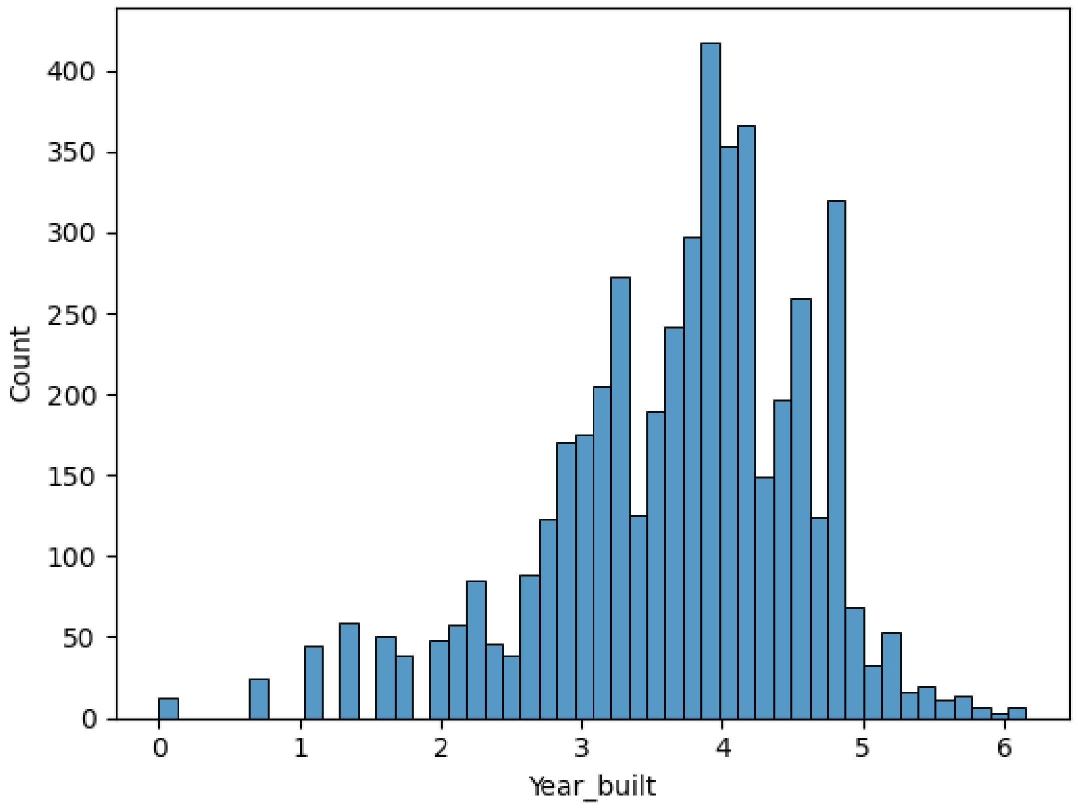
(continued from previous page)

```
2515  13.704579    Single dwelling      4.875197  5.796058    4.0        1.0
2516  13.704579    Single dwelling      4.875197  5.963579    4.0        1.0
2519  13.928839  Mid-terrace house      4.962845  6.061457    4.5        2.0
2520  13.981025  Mid-terrace house      4.962845  6.165418    4.5        2.0
2521  13.652992    Single dwelling      4.897840  5.700444    4.5        1.0

      Floors  Year_built  Year_renovated    Condition        Heating   State  \
2437     3.0      2021.0          2021.0  dilapidated  stove heating  Bayern
2515     3.0      2021.0          2021.0  dilapidated  stove heating  Bayern
2516     3.0      2021.0          2021.0  dilapidated  stove heating  Bayern
2519     3.0      2021.0          2021.0  dilapidated  stove heating  Bayern
2520     3.0      2021.0          2021.0  dilapidated  stove heating  Bayern
2521     3.0      2021.0          2021.0  dilapidated  stove heating  Bayern

                    City       Place  Garages  Garage_type
2437  Rosenheim (Kreis)  Kolbermoor      2.0  Parking lot
2515  Ebersberg (Kreis)   Zorneding      2.0  Parking lot
2516  Ebersberg (Kreis)   Zorneding      2.0  Parking lot
2519  Rosenheim (Kreis)  Kolbermoor      2.0  Parking lot
2520  Rosenheim (Kreis)  Kolbermoor      2.0  Parking lot
2521  Rosenheim (Kreis)  Kolbermoor      2.0  Parking lot
```

```
data = data.loc[data['Year_built'] <= 2020, :]
```

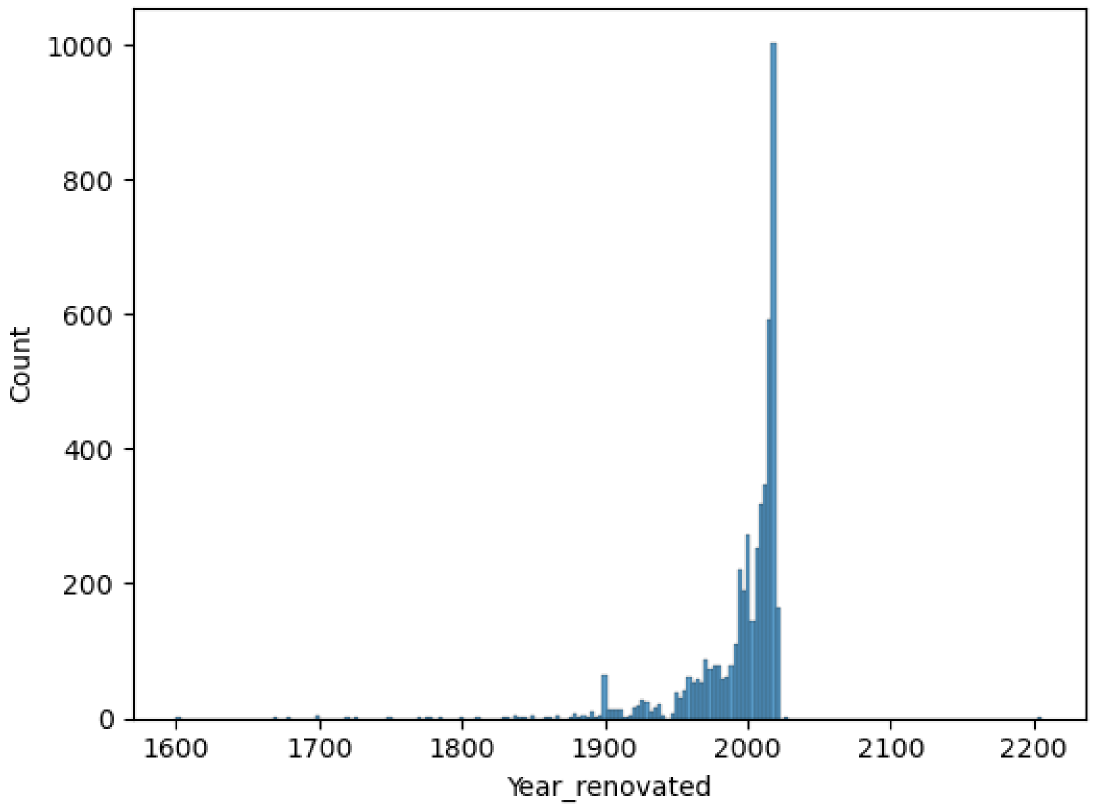
To get a better distribution of the samples over the range, we again apply a logarithmic transform.

```
data.loc[:, 'Year_built'] = np.log(2021 - data['Year_built'].to_numpy())
sns.histplot(data['Year_built'])
plt.show()
```



Year_renovated Column

```
sns.histplot(data['Year_renovated'])
plt.show()
```



There seem to be renovations before 1900, which seems somewhat strange. But remember that we filled missing values with values from `Year_built`. Values above 2020 obviously are wrong.

```
data.loc[data['Year_renovated'] > 2020, :]
```

```
          Price              Type  Living_space       Lot  Rooms  Bathrooms  \
7803  12.992255            Duplex      6.969791  6.818924   26.0       13.0   
9324  12.971540  Mid-terrace house     5.075174  6.486161    6.0        2.0   

      Floors  Year_built  Year_renovated       Condition    Heating  \
7803     2.0    4.025352          2026.0  by arrangement  heat pump   
9324     3.0    4.727388          2206.0     modernized   heat pump   

                State                    City         Place  Garages  \
7803  Rheinland-Pfalz  Trier-Saarburg (Kreis)     Reinsfeld     15.0   
9324          Sachsen                 Zwickau  Nordvorstadt      2.0   

             Garage_type  
7803         Parking lot  
9324  Outside parking lot  
```

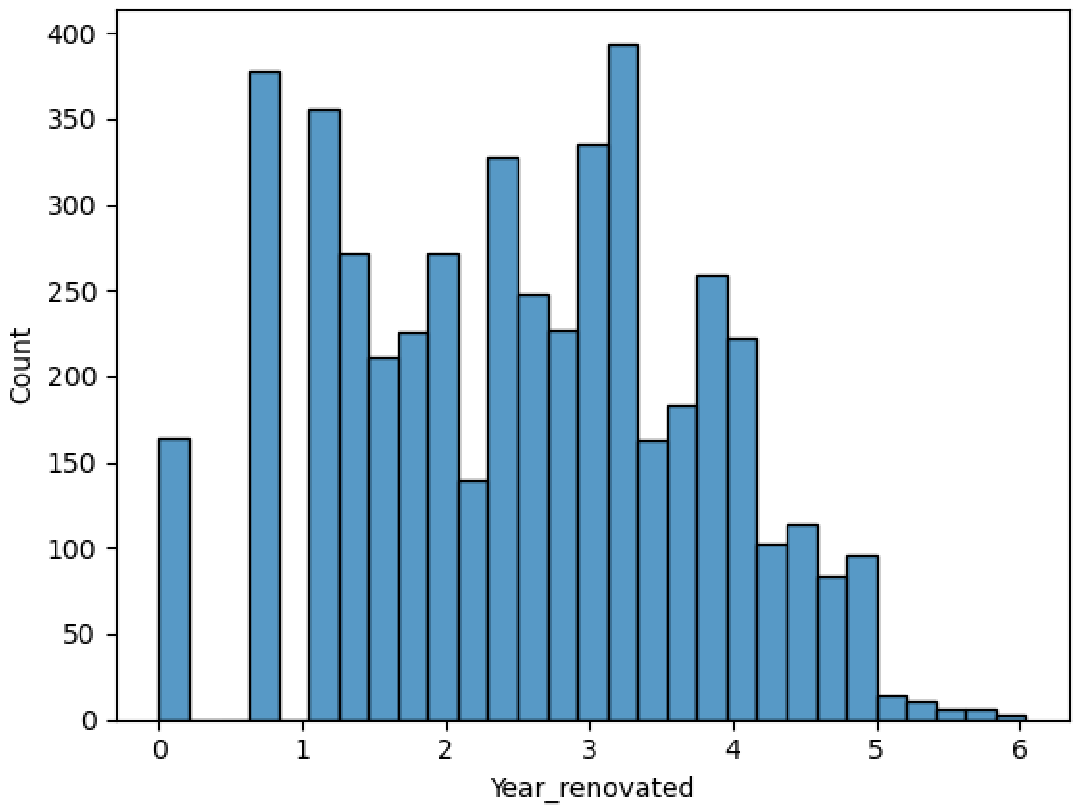
```
data = data.loc[data['Year_renovated'] <= 2020, :]
```

```
data.loc[:, 'Year_renovated'] = np.log(2021 - data['Year_renovated'].to_numpy())
sns.histplot(data['Year_renovated'])
```

(continues on next page)

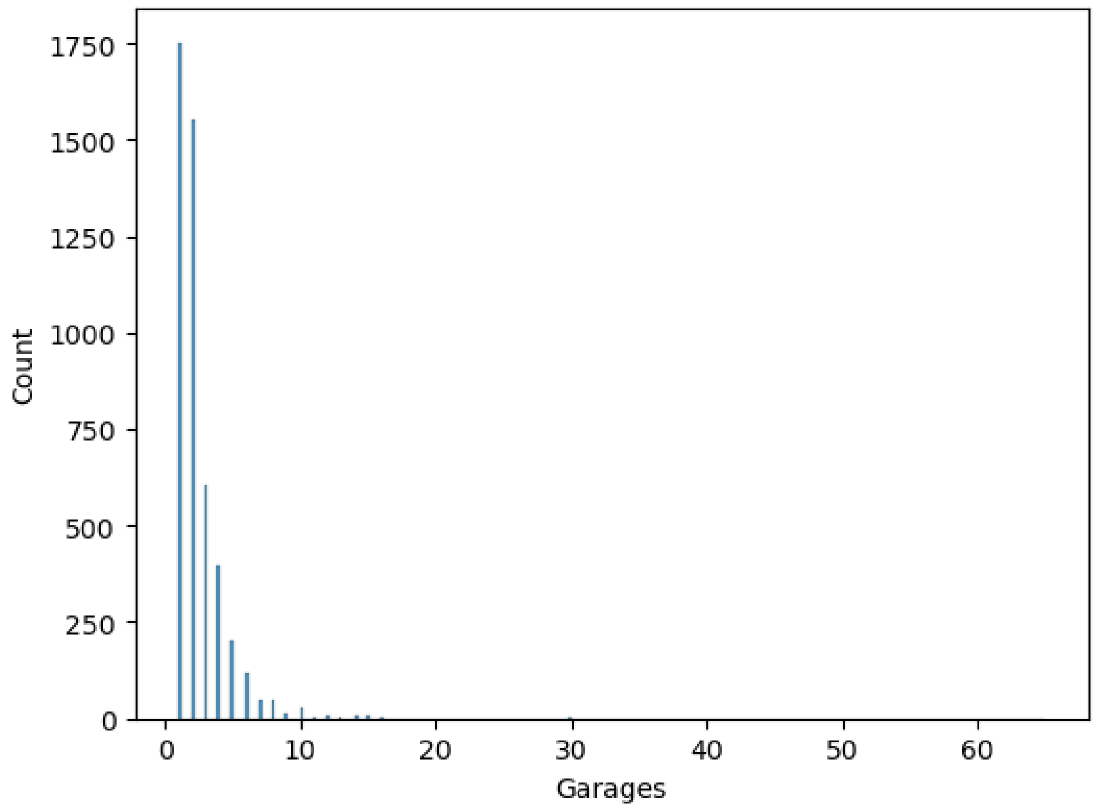
(continued from previous page)

```
plt.show()
```



`Garages` Column

```
sns.histplot(data['Garages'])
plt.show()
```



```
data.loc[data['Garages'] >= 20, :]
```

```
          Price          Type  Living_space        Lot  Rooms  Bathrooms  \
461   14.343193      Bungalow      5.347108   8.932609    8.0        2.0
2579  14.220976        Duplex      6.278521   8.027150   15.0        4.0
3761  15.009130      Bungalow      6.476972   8.116716   10.0        2.0
3932  13.384728        Duplex      5.375278   8.809714   15.0        7.0
4175  14.066269        Duplex      5.777652   6.884487   11.0        5.0
4243  11.407565        Duplex      3.058707   7.342132    1.0        1.0
7166  13.102161      Bungalow      7.003065   9.239899   12.0        3.0
7870  12.959844  Corner house      5.828946  10.596635   14.0        2.0
9535  13.910821        Duplex      6.415097   8.097122   22.0        8.0

      Floors  Year_built  Year_renovated   Condition             Heating  \
461      3.0    3.332205        3.332205  modernized       stove heating
2579     2.0    3.761200        2.397895  modernized  underfloor heating
3761     3.0    5.293305        1.609438   renovated           heat pump
3932     4.0    4.510860        4.510860  maintained       stove heating
4175     2.0    3.496508        0.000000  modernized       stove heating
4243     4.0    3.912023        3.912023  modernized           heat pump
7166     1.0    3.970292        2.639057  modernized         gas heating
7870     3.0    4.795791        1.609438 dilapidated       stove heating
9535     3.0    3.091042        3.091042  modernized       stove heating

                    State                              City  \
461     Baden-Württemberg              Ludwigsburg (Kreis)
2579               Bayern               Rottal-Inn (Kreis)
3761               Hessen                Frankfurt am Main
3932               Hessen          Darmstadt-Dieburg (Kreis)
4175               Hessen           Limburg-Weilburg (Kreis)
4243               Hessen                Main-Taunus-Kreis
7166  Nordrhein-Westfalen            Minden-Lübbecke (Kreis)
```

(continues on next page)

(continued from previous page)

```
7870      Rheinland-Pfalz  Altenkirchen (Westerwald) (Kreis)
9535       Sachsen-Anhalt                      Salzlandkreis

                     Place  Garages           Garage_type
461   Vaihingen an der Enz     30.0  Outside parking lot
2579           Eggenfelden     22.0          Parking lot
3761           Niederursel     20.0  Outside parking lot
3932               Otzberg     60.0  Outside parking lot
4175           Bad Camberg     21.0  Outside parking lot
4243    Hattersheim am Main     30.0          Parking lot
7166             Espelkamp     65.0          Parking lot
7870              Birnbach     50.0  Outside parking lot
9535        Bernburg (Saale)     58.0  Outside parking lot
```

```
data = data.loc[data['Garages'] < 20, :]
```

`Type` Column

```
data['Type'].value_counts()
```

```
Mid-terrace house       2198
Duplex                   868
Single dwelling          565
Farmhouse                265
Villa                    213
Multiple dwelling        209
Special property         168
Residential property     124
Bungalow                 113
Corner house              78
Castle                     2
Name: Type, dtype: int64
```

```
data = data.loc[data['Type'] != 'Castle', :]
data['Type'] = data['Type'].cat.remove_categories('Castle')
```

`Condition` Column

```
data['Condition'].value_counts()
```

```
modernized                            2146
dilapidated                            663
refurbished                            565
renovated                              509
maintained                             361
fixer-upper                            268
first occupation after refurbishment   210
first occupation                        50
by arrangement                          26
as new                                   3
Name: Condition, dtype: int64
```

We should remove `'as new'` and `'by arrangement'` because only few samples use these categories and both are somewhat dubious.

```
data = data.loc[~data['Condition'].isin(['as new', 'by arrangement']), :]
data['Condition'] = data['Condition'].cat.remove_categories(['as new', 'by␣
↪arrangement'])
```

`Heating` Column

```
data['Heating'].value_counts()
```

```
stove heating            2900
heat pump                 563
oil heating               420
central heating           248
underfloor heating        162
night storage heater      138
district heating          117
wood-pellet heating        62
floor heating              55
electric heating           52
gas heating                32
cogeneration units         13
solar heating              10
Name: Heating, dtype: int64
```

Something is wrong here! More than every second house sold in 2020 has stove heating? And what about `'floor heating'`? Is it gas powered or oil powered or what else? What's the difference between `'floor heating'` and `'underfloor heating'`. It's better to drop this column.

```
data = data.drop(columns=['Heating'])
```

`Garage_type` Column

```
data['Garage_type'].value_counts()
```

```
Garage                      2647
Outside parking lot          897
Parking lot                  739
Carport                      409
Underground parking lot       53
Duplex lot                    26
Car park lot                   1
Name: Garage_type, dtype: int64
```

There are many similar categories. We should join some.

```
data.loc[data['Garage_type'] == 'Car park lot', 'Garage_type'] = 'Outside parking␣
↪lot'
data.loc[data['Garage_type'] == 'Duplex lot', 'Garage_type'] = 'Outside parking␣
↪lot'
data.loc[data['Garage_type'] == 'Parking lot', 'Garage_type'] = 'Outside parking␣
↪lot'
data['Garage_type'] = data['Garage_type'].cat.remove_categories(['Car park lot',
↪'Duplex lot', 'Parking lot'])

data['Garage_type'].value_counts()
```

```
Garage                        2647
Outside parking lot           1663
Carport                        409
Underground parking lot         53
Name: Garage_type, dtype: int64
```

24.3.3 Save Cleaned Data

We save cleaned data for future use.

```
data.to_csv(data_path.replace('.csv', '_preprocessed.csv'))
```

24.3.4 Linear Regression

Now data is almost ready for training a model. It remains to convert categorical data to numerical data. `Condition` is ordered and numeric representation is accessible with `Series.cat.codes`[439]. Columns `Type` and `Garage_type` should be one-hot encoded.

```
data['Condition_codes'] = data['Condition'].cat.codes
data = pd.get_dummies(data, columns=['Type', 'Garage_type'], drop_first=True)
```

```
data.head()
```

```
        Price  Living_space       Lot  Rooms  Bathrooms  Floors  Year_built  \
0   13.118355      4.663439  5.433722    5.5        1.0     2.0    2.772589   
2   13.526494      5.093075  4.406719    5.0        2.0     4.0    2.079442   
3   12.464583      4.941642  6.701960    4.0        2.0     2.0    4.795791   
8   12.804909      5.424950  6.880384   10.0        4.0     2.0    5.356586   
10  14.375126      5.347108  7.286192    6.0        2.0     3.0    4.406719   

    Year_renovated    Condition               State               City  \
0         2.772589   modernized  Baden-Württemberg      Bodenseekreis   
2         2.079442  dilapidated  Baden-Württemberg  Esslingen (Kreis)   
3         3.044522  fixer-upper  Baden-Württemberg   Waldshut (Kreis)   
8         1.791759   modernized  Baden-Württemberg          Enzkreis   
10        1.945910   modernized  Baden-Württemberg         Stuttgart   

                    Place  Garages  Condition_codes  Type_Corner house  \
0             Bermatingen      2.0                4                  0   
2              Ostfildern      1.0                7                  0   
3   Bonndorf im Schwarzwald      1.0                6                  0   
8               Neuenbürg      8.0                4                  0   
10              Schönberg      2.0                4                  0   

    Type_Duplex  Type_Farmhouse  Type_Mid-terrace house  \
0             0               0                       0   
2             0               1                       0   
3             0               1                       0   
8             1               0                       0   
10            0               0                       1   

    Type_Multiple dwelling  Type_Residential property  Type_Single dwelling  \
0                        1                         0                     0   
2                        0                         0                     0   
```

(continues on next page)

439 https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.cat.codes.html

(continued from previous page)

```
3                        0                          0                     0
8                        0                          0                     0
10                       0                          0                     0

    Type_Special property  Type_Villa  Garage_type_Garage  \
0                       0           0                   0
2                       0           0                   1
3                       0           0                   1
8                       0           0                   0
10                      0           0                   1

    Garage_type_Outside parking lot  Garage_type_Underground parking lot
0                                 1                                    0
2                                 0                                    0
3                                 0                                    0
8                                 1                                    0
10                                0                                    0
```

We drop columns not used for regression and convert the data frame to NumPy arrays suitable for Scikit-Learn.

```
y = data['Price'].to_numpy()
X = data.drop(columns=['Price', 'Condition', 'State', 'City', 'Place']).to_numpy()

print(X.shape, y.shape)
```

```
(4772, 21) (4772,)
```

We have relatively few data. Thus, test set should be small to have more training samples.

```
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_
↪size=0.2)
print(y_train.size, y_test.size)
```

```
3817 955
```

We use polynomial regression with regularization.

```
steps = [('poly', preprocessing.PolynomialFeatures()),
         ('ridge', linear_model.Ridge())]

pipe = pipeline.Pipeline(steps)

param_grid = {'poly__degree': [1, 2, 3],
              'ridge__alpha': [0] + [2 ** k for k in range(5, 15)]}

gs = model_selection.GridSearchCV(pipe, param_grid=param_grid,
                                   scoring='neg_mean_squared_error', n_jobs=-1,␣
↪cv=5)

gs.fit(X_train, y_train)
best_params = gs.best_params_
```

24.3.5 Evaluating the Model

Now we use the test set to evaluate prediction quality of the model.

```
print(best_params)

pipe.set_params(**best_params)
pipe.fit(X_train, y_train)

y_test_pred = pipe.predict(X_test)
```

```
{'poly__degree': 2, 'ridge__alpha': 64}
```

Root mean squared error between predicted and exact targets on its own does not tell much about fitting quality. We have to compare the value to standard deviation of the targets. Standard deviation is the root mean squared error of the exact targets and their mean. In other words, standard deviation tells us the prediction error if we would use constant predictions for all inputs. Obviously the constant should be the mean of the training (!) targets, but the mean of the training targets should be very close the mean of the test targets if test sample have been selected randomly.

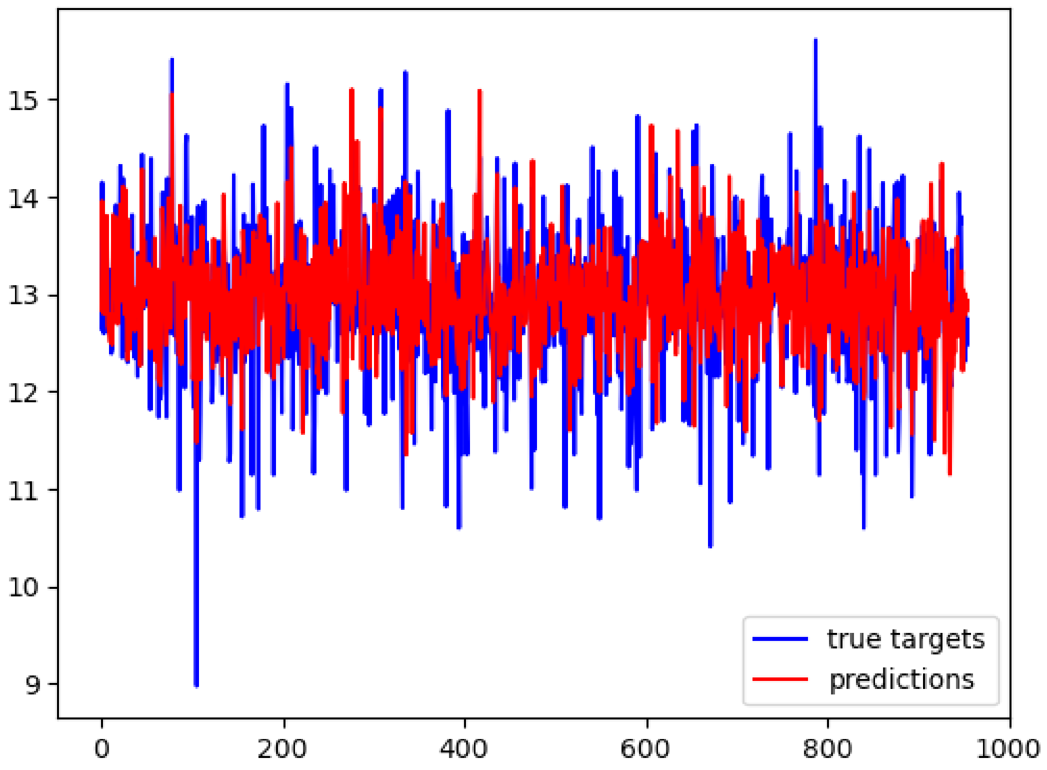
```
rmse = metrics.mean_squared_error(y_test, y_test_pred, squared=False)
sigma = np.std(y_test)
print('RMSE:', rmse)
print('standard deviation:', sigma)
print('ratio:', rmse / sigma)
```

```
RMSE: 0.549529269010328
standard deviation: 0.7553775997302192
ratio: 0.7274894955934497
```

We see that the model's prediction is better than constant prediction, but not so much.

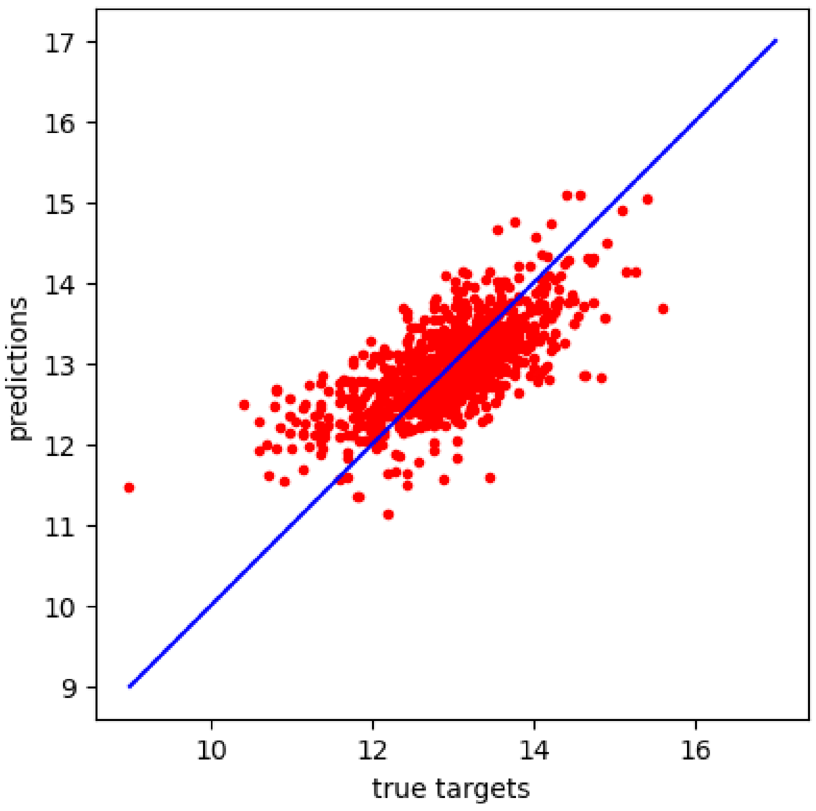
We should have a closer look at the predictions. Since there is no natural ordering in the set of samples plotting `y_test` and `y_test_pred` with `plot` does not help much.

```
fig, ax = plt.subplots()
ax.plot(y_test, '-b', label='true targets')
ax.plot(y_test_pred, '-r', label='predictions')
ax.legend()
plt.show()
```



A better idea is to plot `y_test` versus `y_test_pred`. If true and predicted labels are close, then points should concentrate along the diagonal. Else they are far away from the diagonal.

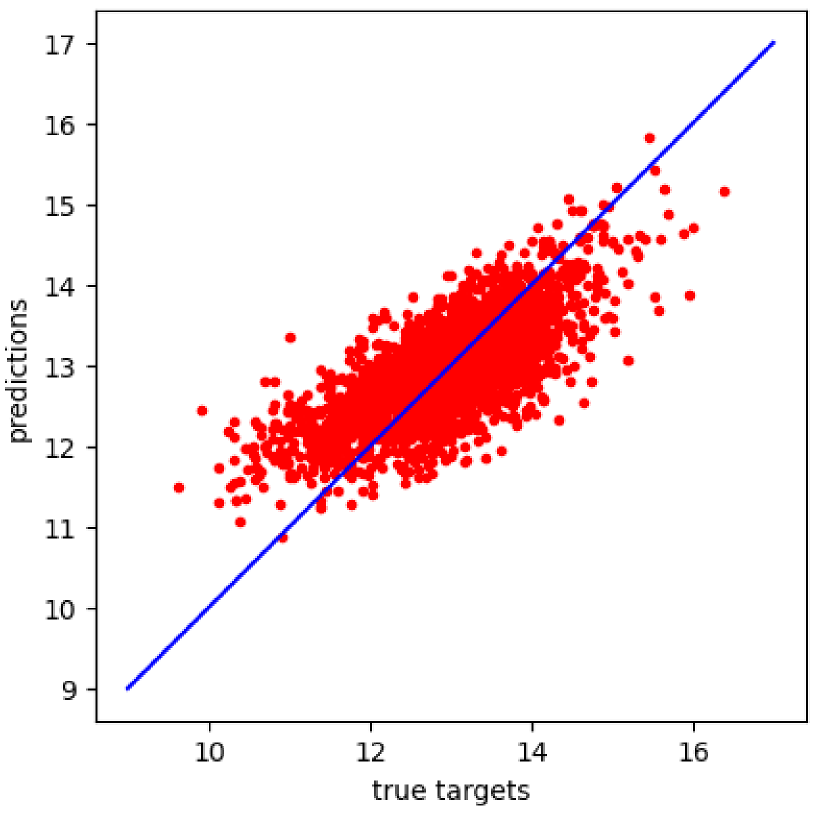
```
fig, ax = plt.subplots()
ax.plot(y_test, y_test_pred, 'or', markersize=3)
ax.plot([9, 17], [9, 17], '-b')
ax.set_xlabel('true targets')
ax.set_ylabel('predictions')
ax.set_aspect('equal')
plt.show()
```



The red cloud shows some rotation compared to the blue line. Small target values get too high predictions and high target values get too low predictions. In other words, predictions tend to be too close to the target's mean. Such behavior is typically observed if there are many similar samples with different targets in the training data. Then there is no clear functional dependence of the targets on the inputs and models tend to predict the mean targets.

To further investigate this issue we should look at the predictions on the training set. If we are right, then predictions on the training set should show similar behavior (predictions close to mean).

```
y_train_pred = pipe.predict(X_train)

fig, ax = plt.subplots()
ax.plot(y_train, y_train_pred, 'or', markersize=3)
ax.plot([9, 17], [9, 17], '-b')
ax.set_xlabel('true targets')
ax.set_ylabel('predictions')
ax.set_aspect('equal')
plt.show()
```



Again we see slight rotation. To summarize: our input data has too few details to explain the targets. There are simlar inputs with different targets leading to underestimation of high values and over estimation of low values. The only way out is gathering more data, either by dropping less columns or by getting relevant data from additional sources. We will come back to this issue soon.

24.3.6 Predictions

When using our model for predicting house prices we have to keep in mind that we transformed some of the input data. All those transforms have to be applied to new inputs, too.

```
living_space = 80
lot = 3600
rooms = 5
bathrooms = 0
floors = 2
year_built = 1948
year_renovated = 1948
garages = 2
condition_codes = 7     # 0 = 'first occupation', 7 = 'dilapidated'
type_corner_house = 0
type_duplex = 0
type_farmhouse = 1
type_midterrace_house = 0
type_multiple_dwelling = 0
type_residential_property = 0
type_single_dwelling = 0
type_special_property = 0
type_villa = 0
garage_type_garage = 0
garage_type_outside_parking_lot = 1
garage_type_underground_parking_lot = 0
```

(continues on next page)

(continued from previous page)

```
X = np.asarray([np.log(living_space), np.log(lot), rooms, bathrooms, floors,
                np.log(2021 - year_built), np.log(2021 - year_renovated),
                garages, condition_codes, type_corner_house, type_duplex, type_
↪farmhouse,
                type_midterrace_house, type_multiple_dwelling, type_residential_
↪property,
                type_single_dwelling, type_special_property, type_villa,
                garage_type_garage, garage_type_outside_parking_lot,
                garage_type_underground_parking_lot]).reshape(1, -1)

y = np.exp(pipe.predict(X))

print('predicted price: {:.0f} EUR'.format(y[0]))
```

```
predicted price: 115455 EUR
```

24.4 Worked Example: House Prices II

We try to improve prediction of house prices based on Erdogan Seref's[440] (unreachable in 2023) German housing dataset from www.immobilienscout24.de[441] published at www.kaggle.com[442] (unreachable in 2023) under a Attribution-NonCommercial-ShareAlike 4.0 International License[443].

We load preprocessed data and adjust data types.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn.linear_model as linear_model
import sklearn.metrics as metrics
import sklearn.model_selection as model_selection
import sklearn.preprocessing as preprocessing
import sklearn.pipeline as pipeline

data_path = 'german_housing_preprocessed.csv'
regions_path = 'regions.csv'
```

```
data = pd.read_csv(data_path, index_col=0)

data['Type'] = data['Type'].astype('category')
data['Condition'] = data['Condition'].astype('category')
data['Garage_type'] = data['Garage_type'].astype('category')

data['Condition'] = data['Condition'].cat.reorder_categories([
    'first occupation',
    'first occupation after refurbishment',
    'maintained',
    'renovated',
    'modernized',
    'refurbished',
```

(continues on next page)

[440] https://www.kaggle.com/scriptsultan
[441] https://www.immobilienscout24.de
[442] https://www.kaggle.com/scriptsultan/german-house-prices
[443] https://creativecommons.org/licenses/by-nc-sa/4.0

(continued from previous page)

```
    'fixer-upper',
    'dilapidated'
])
```

```
data.head()
```

```
        Price                Type  Living_space       Lot  Rooms  Bathrooms  \
0   13.118355  Multiple dwelling      4.663439  5.433722    5.5        1.0   
2   13.526494           Farmhouse      5.093075  4.406719    5.0        2.0   
3   12.464583           Farmhouse      4.941642  6.701960    4.0        2.0   
8   12.804909              Duplex      5.424950  6.880384   10.0        4.0   
10  14.375126  Mid-terrace house      5.347108  7.286192    6.0        2.0   

    Floors  Year_built  Year_renovated    Condition              State  \
0      2.0    2.772589        2.772589   modernized  Baden-Württemberg   
2      4.0    2.079442        2.079442  dilapidated  Baden-Württemberg   
3      2.0    4.795791        3.044522  fixer-upper  Baden-Württemberg   
8      2.0    5.356586        1.791759   modernized  Baden-Württemberg   
10     3.0    4.406719        1.945910   modernized  Baden-Württemberg   

                 City                      Place  Garages           Garage_type
0        Bodenseekreis                Bermatingen      2.0  Outside parking lot
2   Esslingen (Kreis)                  Ostfildern      1.0               Garage
3    Waldshut (Kreis)  Bonndorf im Schwarzwald      1.0               Garage
8             Enzkreis                 Neuenbürg      8.0  Outside parking lot
10           Stuttgart                 Schönberg      2.0               Garage
```

24.4.1 More Data

Results obtained from linear regression showed that input variables do not suffice to explain the targets. Thus, we should add more input variables. When preprocessing the data we dropped several columns. Keeping them could increase prediction quality slightly, but there were several good reasons to drop those columns. The main reason were lots of missing values in those columns.

A far better idea is to collect additional data. What features of a house influence the selling price? Of course its location! Up to now we did not use location information at all, but we have location information available. There are columns `State`, `City`, `Place`. But city names do not help. We need something like proximity to big cities or nice landscape. Adding a layer of abstraction we might ask for the demand for houses and the whealth of potential buyers. So we should head out for statistical information about local real estate markets and about economic power of different regions in Germany.

Everything we need is publicly available at www.regionalstatistik.de[444] provided by *Statistische Ämter des Bundes und der Länder* under the license Datenlizenz Deutschland – Namensnennung – Version 2.0[445]. Clicking here and there we find two interesting tables:

- annual income per inhabitant[446]
- prices for construction ground[447]

From those tables we may compile a table with 4 columns for region id, region name, income, ground prices.

[444] https://www.regionalstatistik.de
[445] https://www.govdata.de/dl-de/by-2-0
[446] https://www.regionalstatistik.de/genesis/online?operation=abruftabelleBearbeiten&levelindex=1&levelid=1614168819692&auswahloperation=abruftabelleAuspraegungAuswaehlen&auswahlverzeichnis=ordnungsstruktur&auswahlziel=werteabruf&code=AI-S-01&auswahltext=&werteabruf=Werteabruf#abreadcrumb
[447] https://www.regionalstatistik.de/genesis/online?operation=abruftabelleBearbeiten&levelindex=1&levelid=1614168973527&auswahloperation=abruftabelleAuspraegungAuswaehlen&auswahlverzeichnis=ordnungsstruktur&auswahlziel=werteabruf&code=61511-01-03-4&auswahltext=&werteabruf=Werteabruf#abreadcrumb

The difficult part is matching region names in German housing data set with region names in the region table. Here is some code doing the job:

```
# remove rows with missing location information
data = data.dropna(subset=('State', 'City'))

# reindex to remove gaps in the index
data.index = pd.RangeIndex(0, len(data))
```

```
regions = pd.read_csv(regions_path)

regions.head(5)
```

```
     id                                    region  income  prices
0     0                               Deutschland   22623  137.67
1     1                        Schleswig-Holstein   22864   85.30
2  1001                 Flensburg, Kreisfreie Stadt   19296   85.30
3  1002   Kiel, Landeshauptstadt, Kreisfreie Stadt   19263   85.30
4  1003       Lübeck, Hansestadt, Kreisfreie Stadt   20363  110.84
```

```
data['city_short'] = data['City'].str.replace(' (Kreis)', '', regex=False)
data['region_idx'] = 0

for (idx, city_short) in enumerate(data['city_short']):
    find_results = regions['region'].str.find(str(city_short))
    if not (find_results > -1).any():
        #print(city_short)
        if data.loc[idx, 'State'] == 'Hamburg':
            find_results = regions['region'].str.find('Hamburg')
            data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
        elif data.loc[idx, 'State'] == 'Bremen':
            find_results = regions['region'].str.find('Bremen')
            data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
        elif data.loc[idx, 'State'] == 'Berlin':
            district = city_short.split('(')[-1][0:-1]
            if district == 'Weißensee':
                district = 'Pankow'
            if district == 'Prenzlauer Berg':
                district = 'Pankow'
            if district == 'Hohenschönhausen':
                district = 'Lichtenberg'
            if district == 'Wedding':
                district = 'Berlin-Mitte'
            find_results = regions['region'].str.find(district)
            #print('***', city_short, ':', district, '-->', regions['Region
↪'][find_results > -1])
            data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
        elif city_short == 'Neuss (Rhein-Kreis)':
            find_results = regions['region'].str.find('Neuss')
            data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
        elif city_short == 'Sankt Wendel':
            find_results = regions['region'].str.find('Wendel')
            data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
        elif city_short == 'Stadtverband Saarbrücken':
            find_results = regions['region'].str.find('Saarbrücken,␣
↪Regionalverband')
            data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
        elif city_short.split()[1] in ('in', 'im', 'an', 'am'):
            if city_short.split()[0] == 'Neustadt':
                find_results = regions['region'].str.find(city_short.split()[-1])
```

(continues on next page)

(continued from previous page)

```
            else:
                find_results = regions['region'].str.find(city_short.split()[0])
            #print('***', city_short, '-->', regions.loc[find_results > -1,
↪'Region'])
            data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
        else:
            print('NOT FOUND:', city_short)

    else:
        # take last match (smallest region)
        data.loc[idx, 'region_idx'] = regions.index[find_results > -1][-1]
```

```
data['Region_id'] = regions.loc[data['region_idx'], 'id'].values
data['Income'] = regions.loc[data['region_idx'], 'income'].values
data['Land_prices'] = regions.loc[data['region_idx'], 'prices'].values
```

```
data = data.drop(columns=['city_short', 'region_idx'])
```

```
data.head(5)
```

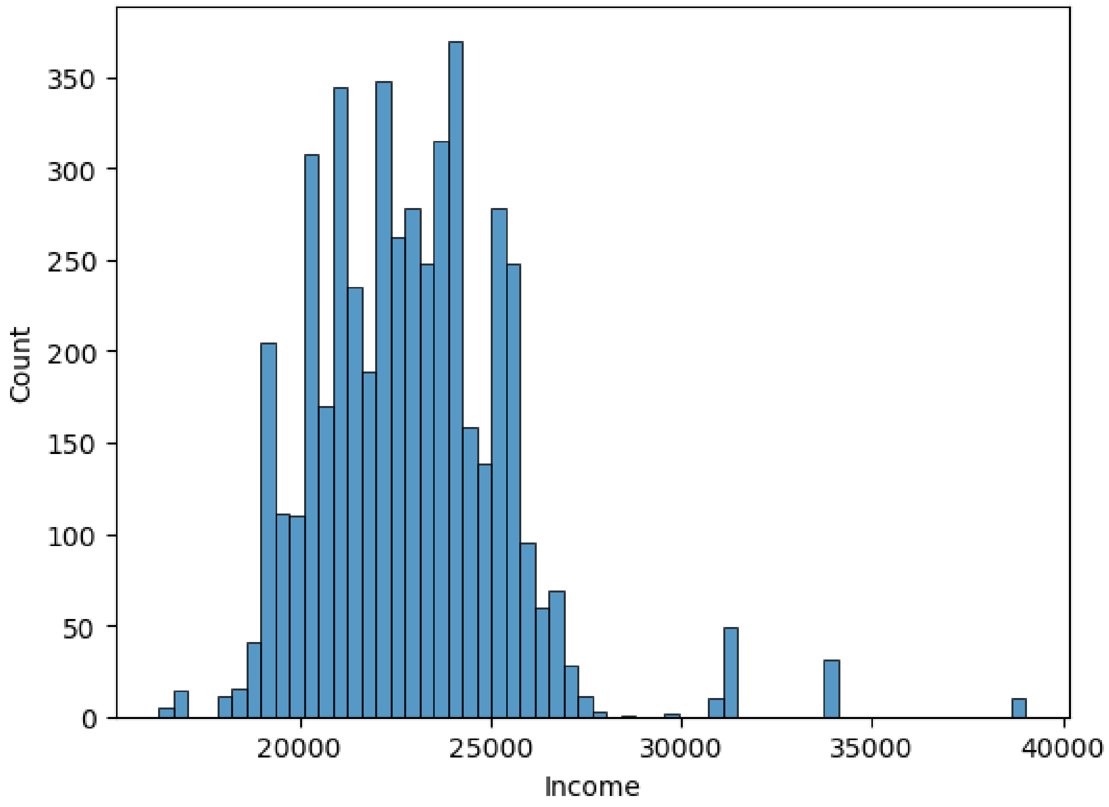
```
       Price                Type  Living_space       Lot  Rooms  Bathrooms  \
0  13.118355  Multiple dwelling      4.663439  5.433722    5.5        1.0
1  13.526494          Farmhouse      5.093075  4.406719    5.0        2.0
2  12.464583          Farmhouse      4.941642  6.701960    4.0        2.0
3  12.804909             Duplex      5.424950  6.880384   10.0        4.0
4  14.375126  Mid-terrace house      5.347108  7.286192    6.0        2.0

   Floors  Year_built  Year_renovated    Condition             State  \
0     2.0    2.772589        2.772589   modernized  Baden-Württemberg
1     4.0    2.079442        2.079442  dilapidated  Baden-Württemberg
2     2.0    4.795791        3.044522  fixer-upper  Baden-Württemberg
3     2.0    5.356586        1.791759   modernized  Baden-Württemberg
4     3.0    4.406719        1.945910   modernized  Baden-Württemberg

                City                   Place  Garages          Garage_type  \
0      Bodenseekreis              Bermatingen      2.0  Outside parking lot
1  Esslingen (Kreis)              Ostfildern      1.0               Garage
2   Waldshut (Kreis)  Bonndorf im Schwarzwald      1.0               Garage
3           Enzkreis              Neuenbürg      8.0  Outside parking lot
4          Stuttgart              Schönberg      2.0               Garage

   Region_id  Income  Land_prices
0       8435   26548       172.95
1       8116   25449       387.06
2       8337   25304       111.64
3       8236   25496       192.18
4       8111   25559      1500.34
```

24.4.2 Preprocessing New Columns

Now that we have two now columns (`Income` and `Land_prices`) we should look at their values.

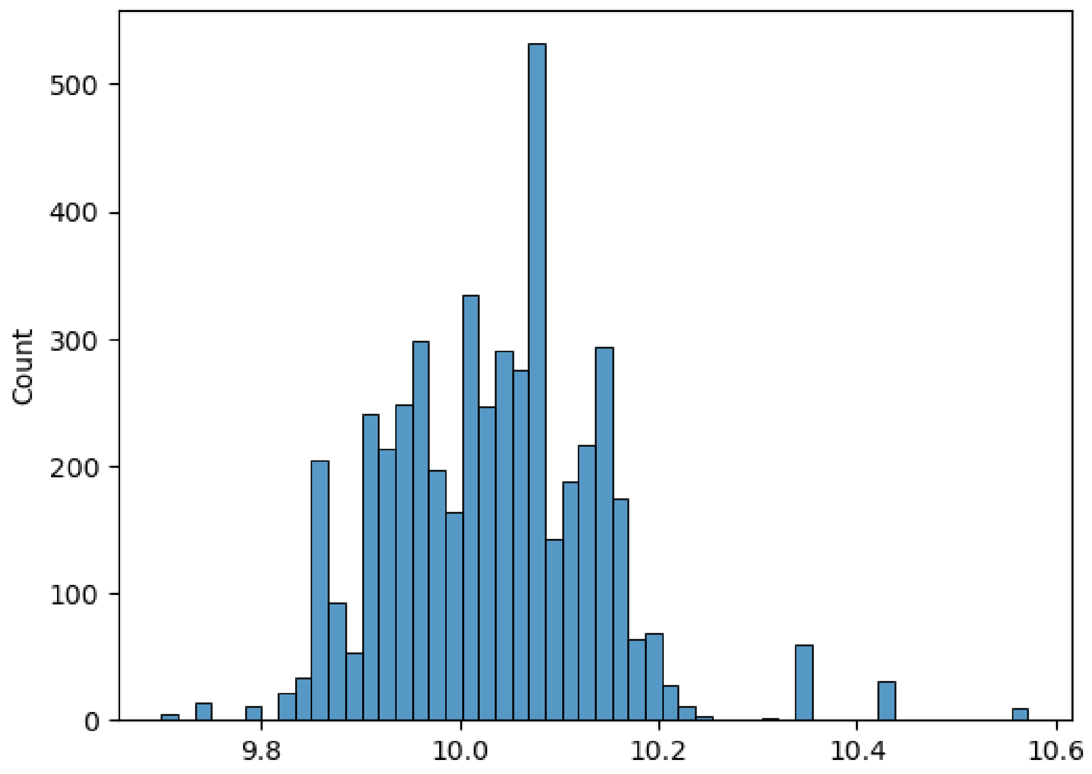
`Income` Column

```
sns.histplot(data['Income'])
plt.show()
```



Log-scaling is not mandadory here, but it brings values to a similar range like the other columns. Without log-scaling we would have a column with very large range, which may result in problems when using regularization (see *Regularization* (page 375)). Alternatively we could standardize all columns before doing linear regression.

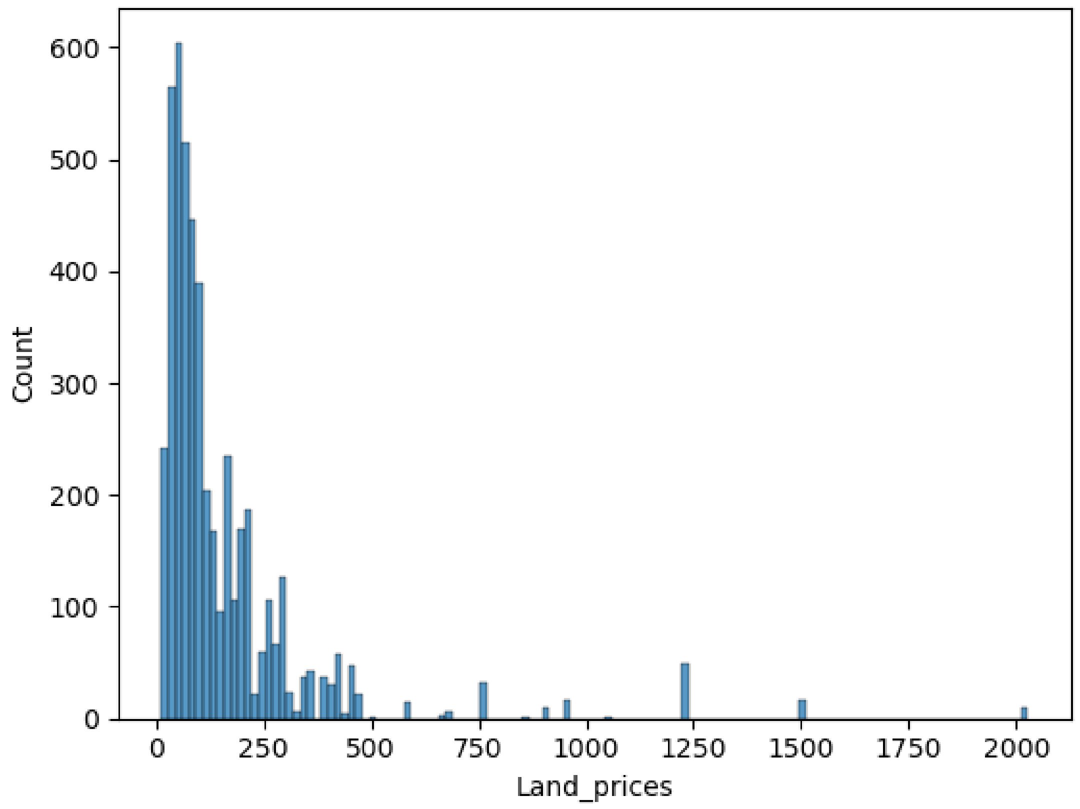
```
sns.histplot(np.log(data['Income'].to_numpy()))
plt.show()
```



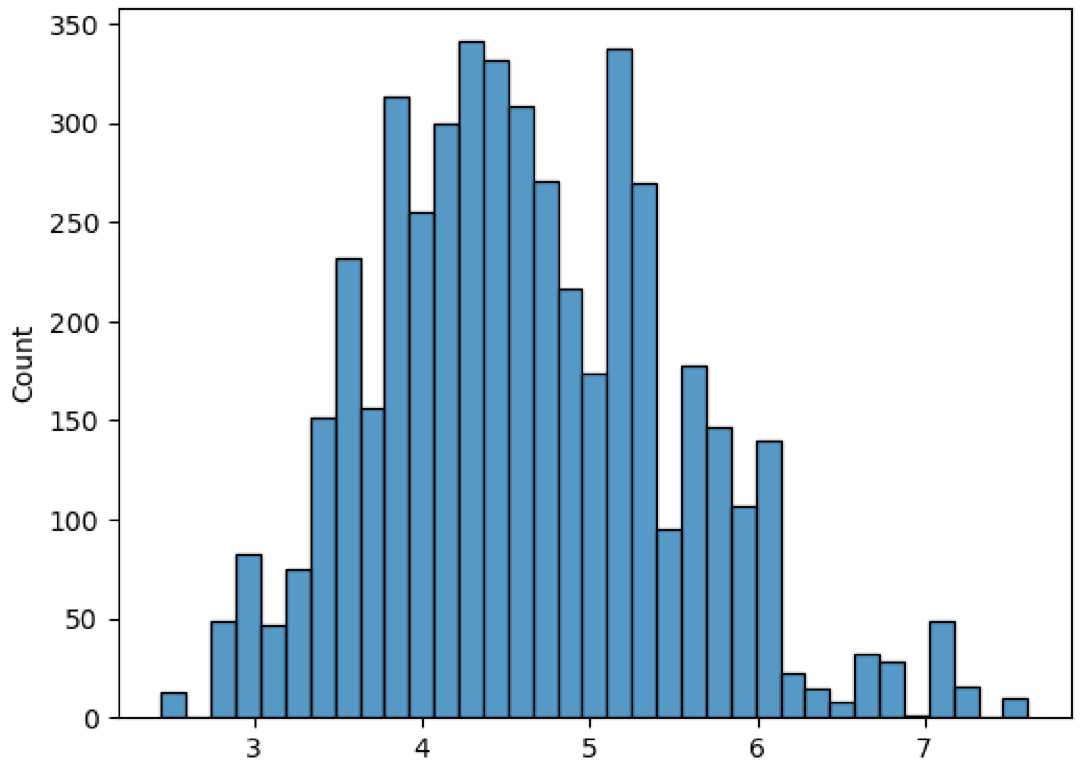
```
data['Income'] = np.log(data['Income'].to_numpy())
```

`Land_prices` Column

```
sns.histplot(data['Land_prices'])
plt.show()
```



```
sns.histplot(np.log(data['Land_prices'].to_numpy()))
plt.show()
```



```
data['Land_prices'] = np.log(data['Land_prices'].to_numpy())
```

24.4.3 Save Data

```
data.to_csv(data_path.replace('preprocessed', 'extended'))
```

24.4.4 Linear Regression

Now we do linear regression as before, but with two additional columns.

```
data['Condition_codes'] = data['Condition'].cat.codes
data = pd.get_dummies(data, columns=['Type', 'Garage_type'], drop_first=True)
```

```
y = data['Price'].to_numpy()
X = data.drop(columns=['Price', 'Condition', 'State', 'City', 'Place', 'Region_id
↪']).to_numpy()

print(X.shape, y.shape)
```

```
(4772, 23) (4772,)
```

```
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_
↪size=0.2)
print(y_train.size, y_test.size)
```

```
3817 955
```

```
steps = [('poly', preprocessing.PolynomialFeatures()),
         ('ridge', linear_model.Ridge())]

pipe = pipeline.Pipeline(steps)

param_grid = {'poly__degree': [1, 2, 3],
              'ridge__alpha': [0] + [2 ** k for k in range(5, 15)]}

gs = model_selection.GridSearchCV(pipe, param_grid=param_grid,
                                  scoring='neg_mean_squared_error', n_jobs=-1,␣
↪cv=5)

gs.fit(X_train, y_train)
best_params = gs.best_params_
```

24.4.5 Evaluation

Now the interesting part. Do we see an increase in prediction quality?

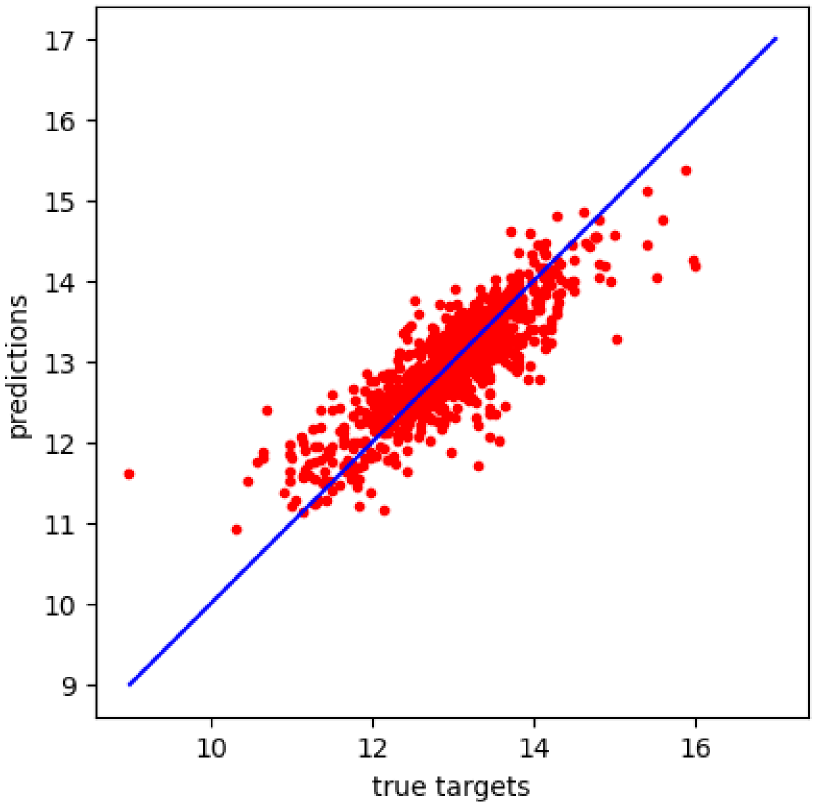
```
print(best_params)

pipe.set_params(**best_params)
pipe.fit(X_train, y_train)

y_test_pred = pipe.predict(X_test)
```

```
{'poly__degree': 2, 'ridge__alpha': 512}
```

```
rmse = metrics.mean_squared_error(y_test, y_test_pred, squared=False)
sigma = np.std(y_test)
print('RMSE:', rmse)
print('standard deviation:', sigma)
print('ratio:', rmse / sigma)
```

```
RMSE: 0.41784051720522897
standard deviation: 0.7969255717596182
ratio: 0.5243156099039885
```

```
fig, ax = plt.subplots()
ax.plot(y_test, y_test_pred, 'or', markersize=3)
ax.plot([9, 17], [9, 17], '-b')
ax.set_xlabel('true targets')
ax.set_ylabel('predictions')
ax.set_aspect('equal')
plt.show()
```



Looks much better!

24.4.6 Feature Importance

With a trained model we may look at feature importances to see which features have high influence on the selling price.

```
import sklearn.inspection as inspection
```

```
result = inspection.permutation_importance(pipe, X, y, n_jobs=-1)
```

```
cols = data.drop(columns=['Price', 'Condition', 'State', 'City', 'Place', 'Region_
↪id']).columns
imp = pd.Series(result.importances_mean, index=cols)
imp = imp.sort_values(ascending=False)
imp
```

```
Land_prices                            0.402403
Living_space                           0.313045
Year_built                             0.169028
Lot                                    0.064657
Income                                 0.029009
Rooms                                  0.019429
Bathrooms                              0.019242
Type_Duplex                            0.017057
Type_Villa                             0.015990
Year_renovated                         0.011070
Condition_codes                        0.008705
Type_Mid-terrace house                 0.007865
Garage_type_Outside parking lot        0.007603
Floors                                 0.005987
Garages                                0.004356
Garage_type_Garage                     0.003244
Type_Corner house                      0.002825
Type_Multiple dwelling                 0.002599
Type_Single dwelling                   0.002530
Type_Farmhouse                         0.000998
Type_Special property                  0.000996
Garage_type_Underground parking lot    0.000359
Type_Residential property              0.000302
dtype: float64
```

24.5 Outliers

Data sets almost always contain outliers, that is, samples showing very different behavior compared to most other samples. Outliers may influence regressions results. Thus, we have to cope with them somehow.

One approach is to remove outliers in advance, so they do not take part in the regression process. But how to detect outliers? Exploratory data analysis might expose some outliers. Alternatively we may apply some advanced statistical methods to automatically detect outliers. But such methods are beyond the scope of this lecture series.

Another approach is to modify regression methods in a way which makes them more *robust* to outliers. We will consider two ideas in detail:

- Choose loss functions other than mean squared error.
- Run regression on subsets of the data to find subsets without outliers.

24.5.1 Example

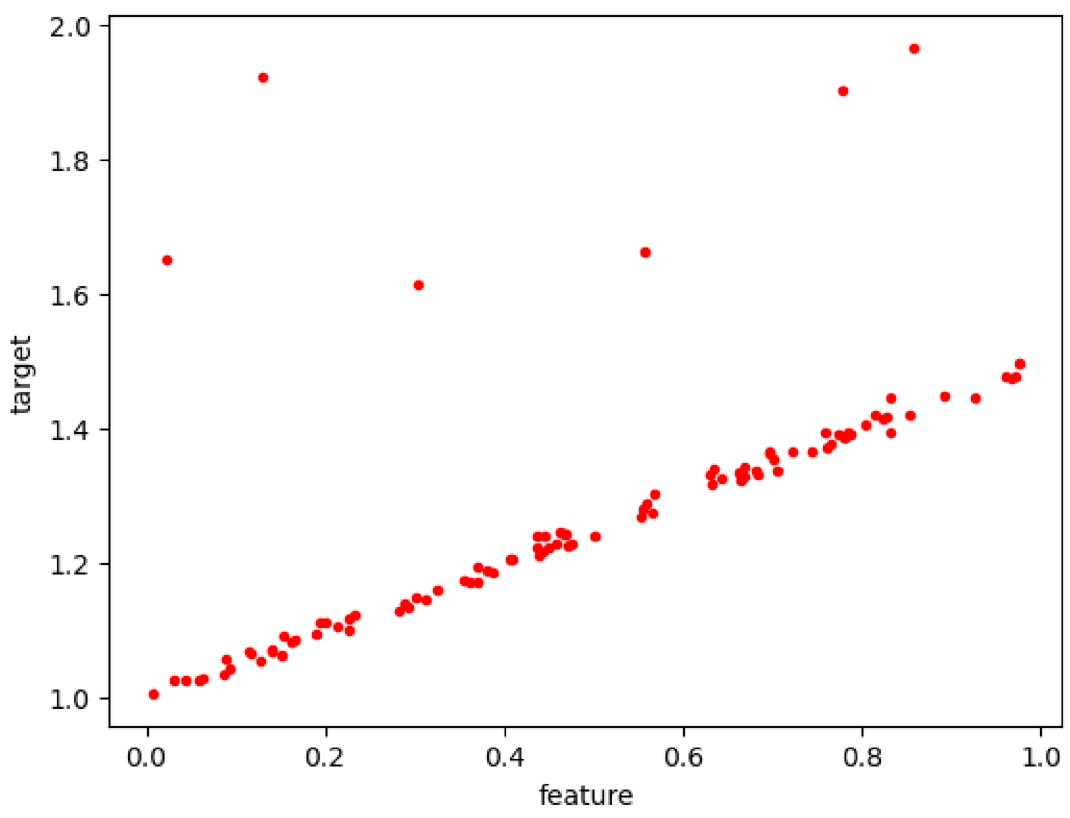
To illustrate the influence of outliers we consider linear regression for a synthetic data set with one feature.

```
import numpy as np
import matplotlib.pyplot as plt

import sklearn.linear_model as linear_model
import sklearn.metrics as metrics

from numpy.random import default_rng
rng = default_rng(42)
```

```
# feature range
xmin = 0
xmax = 1

# true coefficients: y = a * x + b
a = 0.5
b = 1

# parameters of data set
n = 100
noise_level = 0.01
outlier_rate = 0.05
outlier_noise_level = 1
```

To simulate data with outliers we take exact data, add some Gaussian noise, and then randomly select several data points. The selected data points are moved upwards by some random distance.

```
# simulate data without outliers
X = (xmax - xmin) * rng.random((n, 1)) + xmin
y = (a * X + b).reshape(-1) + noise_level * rng.standard_normal(n)

# add some outliers
outlier_mask = rng.choice([True, False], size=y.size, p=[outlier_rate, 1 -␣
↪outlier_rate])
inlier_mask = np.logical_not(outlier_mask)
y[outlier_mask] = (a * X[outlier_mask, 0] + b).reshape(-1) \
                  + outlier_noise_level * rng.random(np.count_nonzero(outlier_
↪mask))

# plot data
fig, ax = plt.subplots()
ax.plot(X.reshape(-1), y, 'or', markersize=3)
ax.set_xlabel('feature')
ax.set_ylabel('target')
plt.show()
```



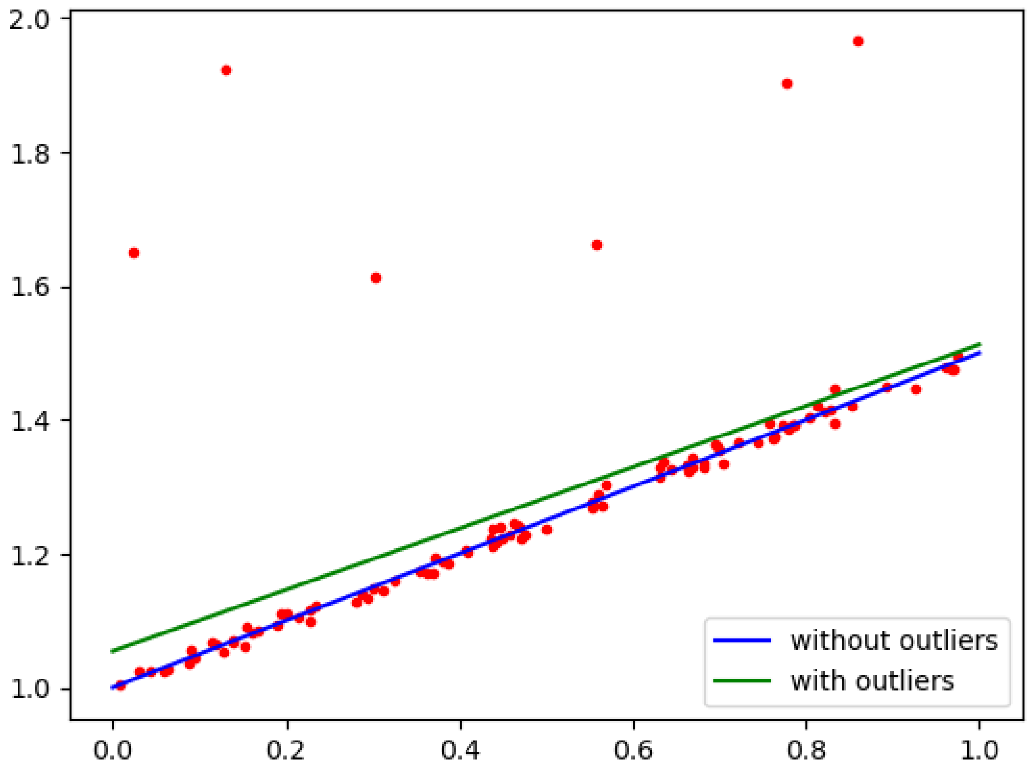
To investigate the influence of outliers we do two regressions: one with outliers removed from the data set and one with the full data set.

```
# regression without outliers
reg = linear_model.LinearRegression()
reg.fit(X[inlier_mask, :], y[inlier_mask])

# regression with full data
outreg = linear_model.LinearRegression()
outreg.fit(X, y)

# plot results
fig, ax = plt.subplots()
ax.plot(X.reshape(-1), y, 'or', markersize=3)
ax.plot(
    [xmin, xmax],
    [reg.intercept_ + xmin * reg.coef_[0], reg.intercept_ + xmax * reg.coef_[0]],
    '-b', label='without outliers'
)
ax.plot(
    [xmin, xmax],
    [outreg.intercept_ + xmin * outreg.coef_[0], outreg.intercept_ + xmax *␣
↪outreg.coef_[0]],
    '-g', label='with outliers')
ax.legend()
plt.show()

# errors
err = metrics.mean_squared_error(y[inlier_mask], reg.predict(X[inlier_mask, :]),
                                  squared=False)
outerr = metrics.mean_squared_error(y[inlier_mask], outreg.predict(X[inlier_mask,␣
↪:]),
                                     squared=False)
```

(continues on next page)

(continued from previous page)

```
exact_err = metrics.mean_squared_error(y[inlier_mask], a * X[inlier_mask, 0] + b,
                                       squared=False)
print('RMSE exact solution:   ', exact_err)
print('RMSE without outliers:', err)
print('RMSE with outliers:    ', outerr)
```



```
RMSE exact solution:   0.009791289366682039
RMSE without outliers: 0.009790327732097473
RMSE with outliers:    0.03698110315752278
```

We clearly see, that the small number of outliers deteriorate results significantly.

24.5.2 Loss Functions

We already discussed three different loss functions in *Quality Measures* (page 328):

- mean squared error (differentiable, sensitive to outliers),
- mean absolut error (non-differentiable, more robust w.r.t. outliers),
- Huber loss (differentiable, more robust w.r.t. to outliers).

If we try to minimize the mean squared error, (large) deviations from outliers get much more penalized than (small) deviations from inliers. In other words, outliers attract the optimal solution much more than inliers. Thus, even few outliers lead to deviations of the minimizing regression line towards the outliers.

MSE is favorable in view of computation time (linear regression requires solving only one system of linear equations). MAE requires very advanced optimization algoritms to solve the linear regression problem. Huber loss is somewhere in between.

MAE with Scikit-Learn

Scikit-Learn does not implement regression with mean absolute error loss directly. But we may interpret mean absolute error regression as a special case of a more general regression routine: the `SGDRegressor`[448]. This is not a regression technique on its own, but a specific algorithm for minimizing a wide class of functionals typically occuring in regression problems. We do not go into the details her, but we only use it to test mean absolute error regression for our simple illustrating example. Take care, that `SGDRegressor` uses gradient based optimization here, which might yield non-optimal results due to non-differentiability.

```
maereg = linear_model.SGDRegressor('epsilon_insensitive', epsilon=0, alpha=0)
maereg.fit(X, y)

maeerr = metrics.mean_squared_error(y[inlier_mask], maereg.predict(X[inlier_mask,␣
↪:]),
                                    squared=False)
print('RMSE exact solution:              ', exact_err)
print('RMSE for MSE without outliers:    ', err)
print('RMSE for MSE with outliers:       ', outerr)
print('RMSE for MAE loss with outliers: ', maeerr)
```

```
RMSE exact solution:              0.009791289366682039
RMSE for MSE without outliers:    0.009790327732097473
RMSE for MSE with outliers:       0.03698110315752278
RMSE for MAE loss with outliers:  0.009839046162356173
```

Huber Loss with Scikit-Learn

For details on linear regression with Huber loss see Scikit-Learn's User Guide[449].

Scikit-Learn provides the `HuberRegressor`[450].

```
huberreg = linear_model.HuberRegressor()
huberreg.fit(X, y)

hubererr = metrics.mean_squared_error(y[inlier_mask], huberreg.predict(X[inlier_
↪mask, :]),
                                    squared=False)
print('RMSE exact solution:                ', exact_err)
print('RMSE for MSE without outliers:      ', err)
print('RMSE for MSE with outliers:         ', outerr)
print('RMSE for MAE loss with outliers:    ', maeerr)
print('RMSE for Huber loss with outliers: ', hubererr)
```

```
RMSE exact solution:                0.009791289366682039
RMSE for MSE without outliers:      0.009790327732097473
RMSE for MSE with outliers:         0.03698110315752278
RMSE for MAE loss with outliers:    0.009839046162356173
RMSE for Huber loss with outliers:  0.00980351846505725
```

[448] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html
[449] https://scikit-learn.org/stable/modules/linear_model.html#huber-regression
[450] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html

24.5.3 RANSAC Algorithm

If one does not want to change the loss function to lower the influence of outliers, one can apply the RANSAC algorithm to a given regression method. RANSAC is the abbreviation of *random sample consensus*. The idea is to randomly choose a small subset of the data and train the model on this subset. The smaller the subset the higher the chance to have no outliers in the subset. But the subset has to be large enough to determine all model parameters.

In a second step all other data points are compared to corresponding model predictions. If the model provides a good prediction, the data point is considered as inlier. If the prediction if far away from the true target, then the data point is marked as outlier. Here, we have to choose a threshold value to separate inliers from outliers.

If there are too few inliers compared to the size of the data set (and to an estimate of the outlier rate), the model is rejected. Else the model is refit to the whole inlier set and the fitting error is computed.

This process is repeated with different random subsets several times. The model with the lowest fitting error is chosen as final model. Probabilistic considerations suggest up to

$$\frac{\ln(1-p)}{\ln(1-(1-r)^s)},$$

iterations, where s is the number of data points in the subset, $r \in (0,1)$ is the estimated outlier rate for the full data set, and p is the prescribed probability to have at least one subset containing no outliers. Alternatively, one can prescribe an error level to stop the iteration if the fitting error of a model is below this level.

Example

To get a better understanding of the algorithm we implement it without using Scikit-Learn routines.

First we set the parameters and calculate the number of iterations required.

```
estimated_outlier_rate = 0.05    # ~ outlier_rate
threshold_value = 0.04    # ~ noise_level
subset_size = 2    # two points determine a straight line (2 model parameters)
min_inliers = int(0.9 * n)    # reliable lower bound on number of inliers in full␣
↪data set

max_iter = int(np.log(1 - 0.99) / np.log(1 - (1 - estimated_outlier_rate) **␣
↪subset_size))
print('max_iter =', max_iter)
```

```
max_iter = 1
```

```
max_iter = 10
```

To be able to investigate each step of the algorithm we store all information about each single iteration.

```
subset_masks = np.empty((max_iter, n), dtype=bool)    # each row is a mask␣
↪determining a subset
initial_subset_masks = np.empty((max_iter, n), dtype=bool)    # initial subset␣
↪used for first fit
errors = np.empty(max_iter)    # MSE for all models
models_a = np.empty(max_iter)    # parameter a for all models
models_b = np.empty(max_iter)    # parameter b for all models
initial_models_a = np.empty(max_iter)    # parameter a for all models (first fit)
initial_models_b = np.empty(max_iter)    # parameter b for all models (first fit)

ordered_mask = np.full(n, False)      # will be shuffled to get a random mask...
ordered_mask[0:subset_size] = True    # ...for subset selection

reg = linear_model.LinearRegression()
```

(continues on next page)

(continued from previous page)

```
best_error = None
best_index = None
best_model = None

for k in range(0, max_iter):

    # determine subset
    subset_mask = rng.permutation(ordered_mask)
    initial_subset_masks[k] = subset_mask.copy()

    # fit model to subset
    reg.fit(X[subset_mask, :], y[subset_mask])
    initial_models_a[k] = reg.coef_[0]
    initial_models_b[k] = reg.intercept_

    # get points supporting the model
    not_mask = np.logical_not(subset_mask)
    subset_mask[not_mask] = (np.abs(reg.predict(X[not_mask, :]) - y[not_mask])
                             <= threshold_value)

    # refit model to extended subset
    reg.fit(X[subset_mask, :], y[subset_mask])
    models_a[k] = reg.coef_[0]
    models_b[k] = reg.intercept_
    errors[k] = metrics.mean_squared_error(reg.predict(X[subset_mask, :]),␣
 ↪y[subset_mask])
    subset_masks[k, :] = subset_mask

    # find best model
    if np.count_nonzero(subset_mask) >= min_inliers:
        if (best_error == None) or (errors[k] < best_error):
            best_error = errors[k]
            best_index = k
            best_model = reg
```

The fitting error is very good:

```
ransacerr = metrics.mean_squared_error(y[inlier_mask], best_model.
 ↪predict(X[inlier_mask, :]),
                                       squared=False)
print('RMSE exact solution:               ', exact_err)
print('RMSE for MSE without outliers:     ', err)
print('RMSE for MSE with outliers:        ', outerr)
print('RMSE for MAE loss with outliers:   ', maeerr)
print('RMSE for Huber loss with outliers: ', hubererr)
print('RMSE for RANSAC with outliers:     ', ransacerr)
```

```
RMSE exact solution:                0.009791289366682039
RMSE for MSE without outliers:      0.009790327732097473
RMSE for MSE with outliers:         0.03698110315752278
RMSE for MAE loss with outliers:    0.009839046162356173
RMSE for Huber loss with outliers:  0.00980351846505725
RMSE for RANSAC with outliers:      0.00980412170408581
```

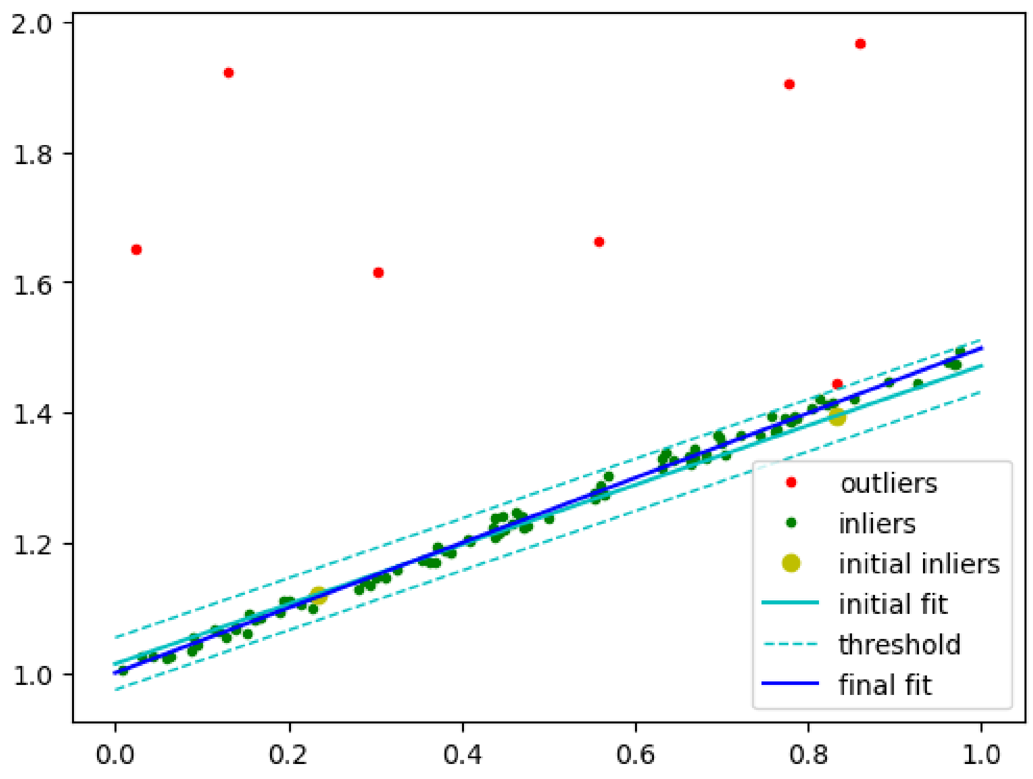
Since we have stored all information, we now may plot the fitting procedure for each interation.

```
index = best_index

init_mask = initial_subset_masks[index]
```

(continues on next page)

(continued from previous page)

```
mask = subset_masks[index]
not_mask = np.logical_not(mask)
init_a = initial_models_a[index]
init_b = initial_models_b[index]

fig, ax = plt.subplots()

# points not supporting the model
ax.plot(X[not_mask, 0], y[not_mask], 'or', markersize=3, label='outliers')

# points supporting the model
ax.plot(X[mask, 0], y[mask], 'og', markersize=3, label='inliers')

# initial subset
ax.plot(X[init_mask, 0], y[init_mask], 'oy', markersize=6, label='initial inliers
↪')

# initial fit
ax.plot([xmin, xmax], [init_b + xmin * init_a,
                       init_b + xmax * init_a], '-c', label='initial fit')

# treshold value
ax.plot([xmin, xmax], [threshold_value + init_b + xmin * init_a,
                       threshold_value + init_b + xmax * init_a], '--c', label=
↪'threshold', linewidth=1)
ax.plot([xmin, xmax], [-threshold_value + init_b + xmin * init_a,
                       -threshold_value + init_b + xmax * init_a], '--c',␣
↪linewidth=1)

# final fit
ax.plot([xmin, xmax], [models_b[index] + xmin * models_a[index],
                       models_b[index] + xmax * models_a[index]], '-b', label=
↪'final fit')

ax.legend()
plt.show()
```



RANSAC with Scikit-Learn

Scikit-Learn offers the `RANSACRegressor`[451].

```
reg = linear_model.RANSACRegressor(linear_model.LinearRegression(),
                                   min_samples=subset_size,
                                   residual_threshold=threshold_value,
                                   max_trials=max_iter,
                                   stop_n_inliers=min_inliers)
reg.fit(X, y)

ransac2err = metrics.mean_squared_error(y[inlier_mask], reg.predict(X[inlier_mask,
 ↪ :]),
                                        squared=False)
print('RANSAC:                ', ransacerr)
print('RANSAC (Scikit-Learn):', ransac2err)
```

```
RANSAC:                0.00980412170408581
RANSAC (Scikit-Learn): 0.00980412170408581
```

[451] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RANSACRegressor.html

Discussion

Quality of results obtained by RANSAC heavily depends on chosen parameters and on the number of iterations. The more iterations, the better the results, but the more computation time is required.

A major advantage is that RANSAC can be applied to almost all regression methods and is not restricted to linear regression. In addition, RANSAC also works if there are many outliers (up to 50 per cent) and there exist extensions of the algorithm which can cope with even more outliers. Thus, RANSAC is the first choice for problems with many outliers, like image stitching and motion reconstruction in computer vision.

CHAPTER

TWENTYFIVE

LOGISTIC REGRESSION

Logistic regression is a standard classification technique for binary and multiclass problems. It predicts class probabilities via nonlinear regression. Probabilities are modeled as sigmoid of a linear function of the feature values (or softmax of linear functions for multiclass). The model is fit to data by minimizing the log loss. Logistic regression almost always requires regularization, which is easily incorporated by adding some penalty (cf. linear regression) to the loss. Minimization is done numerically, e.g. with gradient descent.

25.1 Binary Logistic Regression

25.1.1 The Method

Given a binary classification task with classes 0 and 1 we model the probability that a sample $x\in\mathbb{R}^m$ belongs to class 1 as

$$\frac{1}{1+\mathrm{e}^{-(a_0+a_1\,x^{(1)}+\cdots+a_m\,x^{(m)})}}$$

with real parameters $a_0,a_1,\ldots,a_m$. This is the sigmoid function applied to score which is a linear function of the inputs. With

$$\mathring{x}:=\left(1,x^{(1)},\ldots,x^{(m)}\right)\qquad\text{and}\qquad\mathring{a}:=(a_0,a_1,\ldots,a_m)$$

it reads

$$\frac{1}{1+\mathrm{e}^{-\mathring{a}^{\mathrm{T}}\,\mathring{x}}}.$$

For training samples $(x_1,y_1),\ldots,(x_n,y_n)$ corresponding log loss is

$$\begin{aligned}\text{log loss}&=-\frac{1}{n}\sum_{l=1}^{n}\left(y_l\,\log\frac{1}{1+\mathrm{e}^{-\mathring{a}^{\mathrm{T}}\,\mathring{x}_l}}+(1-y_l)\,\log\left(1-\frac{1}{1+\mathrm{e}^{-\mathring{a}^{\mathrm{T}}\,\mathring{x}_l}}\right)\right)\\&=\frac{1}{n}\sum_{l=1}^{n}\left(y_l\,\log\left(1+\mathrm{e}^{-\mathring{a}^{\mathrm{T}}\,\mathring{x}_l}\right)-(1-y_l)\,\left(-\mathring{a}^{\mathrm{T}}\,\mathring{x}_l-\log\left(1+\mathrm{e}^{-\mathring{a}^{\mathrm{T}}\,\mathring{x}_l}\right)\right)\right)\\&=\frac{1}{n}\sum_{l=1}^{n}\left((1-y_l)\,\mathring{a}^{\mathrm{T}}\,\mathring{x}_l+\log\left(1+\mathrm{e}^{-\mathring{a}^{\mathrm{T}}\,\mathring{x}_l}\right)\right).\end{aligned}$$

Minimizing the loss function with respect to $\mathring{a}$ yields the model

$$\frac{1}{1+\mathrm{e}^{-\mathring{a}^{\mathrm{T}}\,\mathring{x}}}.$$

for predicting the probability that some sample x belongs to class 1. From that probability we may derive a class table by thresholding.

25.1.2 Interpretation

Thresholding predicted probabilities at some level $t \in (0,1)$ yields class 1 if and only if

$$\frac{1}{1+\mathrm{e}^{-\mathring{a}^{\mathrm{T}}\,\mathring{x}}} \geq t \quad\Leftrightarrow\quad \frac{1-t}{t} \geq \mathrm{e}^{-\mathring{a}^{\mathrm{T}}\,\mathring{x}} \quad\Leftrightarrow\quad \log\frac{t}{1-t} \leq \mathring{a}^{\mathrm{T}}\,\mathring{x}.$$

The equation

$$\log\frac{t}{1-t} = \mathring{a}^{\mathrm{T}}\,\mathring{x} \quad\Leftrightarrow\quad a_1\,x^{(1)} + \cdots + a_m\,x^{(m)} = \log\frac{t}{1-t} - a_0$$

defines a hyperplane in $\mathbb{R}^m$ (the set of points x satisfying the equation is a hyperplane). Thus, logistic regression (plus thresholding) splits the feature space into two half spaces and assigns samples in one half space the label 0 and samples in the other half space the label 1. Logistic regression determines the direction of the splitting hyperplane, the threshold controls parallel displacement.

From this observation it is obvious that logistic regression only yields good predictions if classes are (almost) linearly separable.

25.1.3 Why not Fitting Scores Directly?

Logistic regression uses linear scoring, but fits sigmoids of the scores to the data. Sigmoids convert the linear ansatz to a nonlinear one. Why not transforming data and then fit a linear model to the transformed data? If p is the predicted probability for x belonging to class 0 we have

$$p = \frac{1}{1+\mathrm{e}^{-\mathring{a}^{\mathrm{T}}\,\mathring{x}}} \quad\Leftrightarrow\quad \log\frac{p}{1-p} = a_0 + a_1\,x^{(1)} + \cdots + a_m\,x^{(m)}.$$

Thus, if we apply the function g defined by

$$g(v) := \log\frac{v}{1-v}$$

to the labels we have linear regression with linear functions. The problem is that class labels are either 0 or 1, but the domain of g is $(0,1)$. We only have

$$\lim_{v\to 0} g(v) = -\infty \qquad \text{and} \qquad \lim_{v\to 1} g(v) = \infty,$$

so we would have to transform labels 0 and 1 to $\pm\infty$, which cannot be handled in a linear regression setting.

25.2 Multiclass Logistic Regression

For multiclass tasks we may use one of the following two standard (that is, not restricted to logistic regression) approaches to reduce the problem to several binary tasks.

25.2.1 One-versus-Rest Approach

Given a multiclass classification problem with $C > 2$ classes we consider C binary classification problems, one per class. In each binary problem we decide whether an input belongs to the corresponding class or not. This yields c scores for the multiclass classification problem.

The one-versus-rest approach requires training C binary classification models on the full data set, which is only feasible if sufficient computational resources are available. One-versus-rest is mainly used in combination with logistic regression.

25.2.2 One-versus-One Approach

Given a multiclass classification problem with $C > 2$ classes we consider binary classification problems for all combinations of classes. With C classes there are $\frac{C\,(C-1)}{2}$ class pairs. Each binary model is trained on the subset of the training set containing samples with targets in one of the two classes. Given an input we obtain a number of hits for each class. The class with most hits, that is, the class chosen in most of the binary problems, is chosen as output of the multiclass task.

With the one-versus-one approach we have to train more models than for the one-versus-rest approach and training data sets are smaller. One-versus-one is typically used in combination with models not very sensitive to training set size. An example are so called support vector machines (SVM) considered later on.

25.2.3 Alternative to Standard Approaches

An alternative to one-versus-rest and one-versus-one specific to logistic regression is to replace the sigmoid by softmax for transforming lineare scores. Then logistic regression minimizes the log loss of softmax of linear score models. Because probabilities sum to 1 we only have to predict $C - 1$ probabilities in a C classes setting. For instance we could set the score of class C to 0. Then the model for the C probabilities is

$$\frac{\mathrm{e}^{\mathring{a}_1^{\mathrm{T}}\,\mathring{x}}}{\mathrm{e}^{\mathring{a}_1^{\mathrm{T}}\,\mathring{x}}+\cdots+\mathrm{e}^{\mathring{a}_{C-1}^{\mathrm{T}}\,\mathring{x}}+1},\quad \ldots,\quad \frac{\mathrm{e}^{\mathring{a}_{C-1}^{\mathrm{T}}\,\mathring{x}}}{\mathrm{e}^{\mathring{a}_1^{\mathrm{T}}\,\mathring{x}}+\cdots+\mathrm{e}^{\mathring{a}_{C-1}^{\mathrm{T}}\,\mathring{x}}+1},\quad \frac{1}{\mathrm{e}^{\mathring{a}_1^{\mathrm{T}}\,\mathring{x}}+\cdots+\mathrm{e}^{\mathring{a}_{C-1}^{\mathrm{T}}\,\mathring{x}}+1}$$

with $(C-1)\,(m+1)$ parameters a_i^l, $i = 1, \ldots, C-1$, $l = 0, 1, \ldots, m$.

25.3 Logistic Regression with Scikit-Learn

Scikit-Learn implements three classes for logistic regression in the `linear_model` module:

- LogisticRegression[452] implements logistic regression with regularization. Strength of regularization can be controlled via the parameter C, which is the inverse of the usual regularization parameter α. The higher C the less regularization is applied. Different algorithms for solving the minimization problem can be chosen.
- SGDClassifier[453] implements stochastic gradient descent (very efficient in case of many samples) for several loss functions. Passing `loss='log'` yields regularized logistic regression. Regulatization is controlled by parameter `alpha`.
- LogisticRegressionCV[454] combines `LogisticRegression` with cross validation for choosing the regularization parameter.

All three classes support different penalties (squares/l2, LASSO/l1, elastic net).

25.4 The Need for Regularization

Logistic regression almost always is run with regularization. There are two reasons for regularizing logistic regression:

- avoid overfitting,
- guarantee existence of a solution.

Although logistic regression uses linear models with relatively few parameters overfitting may occur. An example will be given below.

A more serious problem is that for linearly separable classes (that is, perfect classification is possible with logistic regression), the unregularized objective function of logistic regression has no minimizer. We may drive its value arbitrarily close to zero, but we cannot reach zero. Model parameters will grow to infinity und numerical minimization

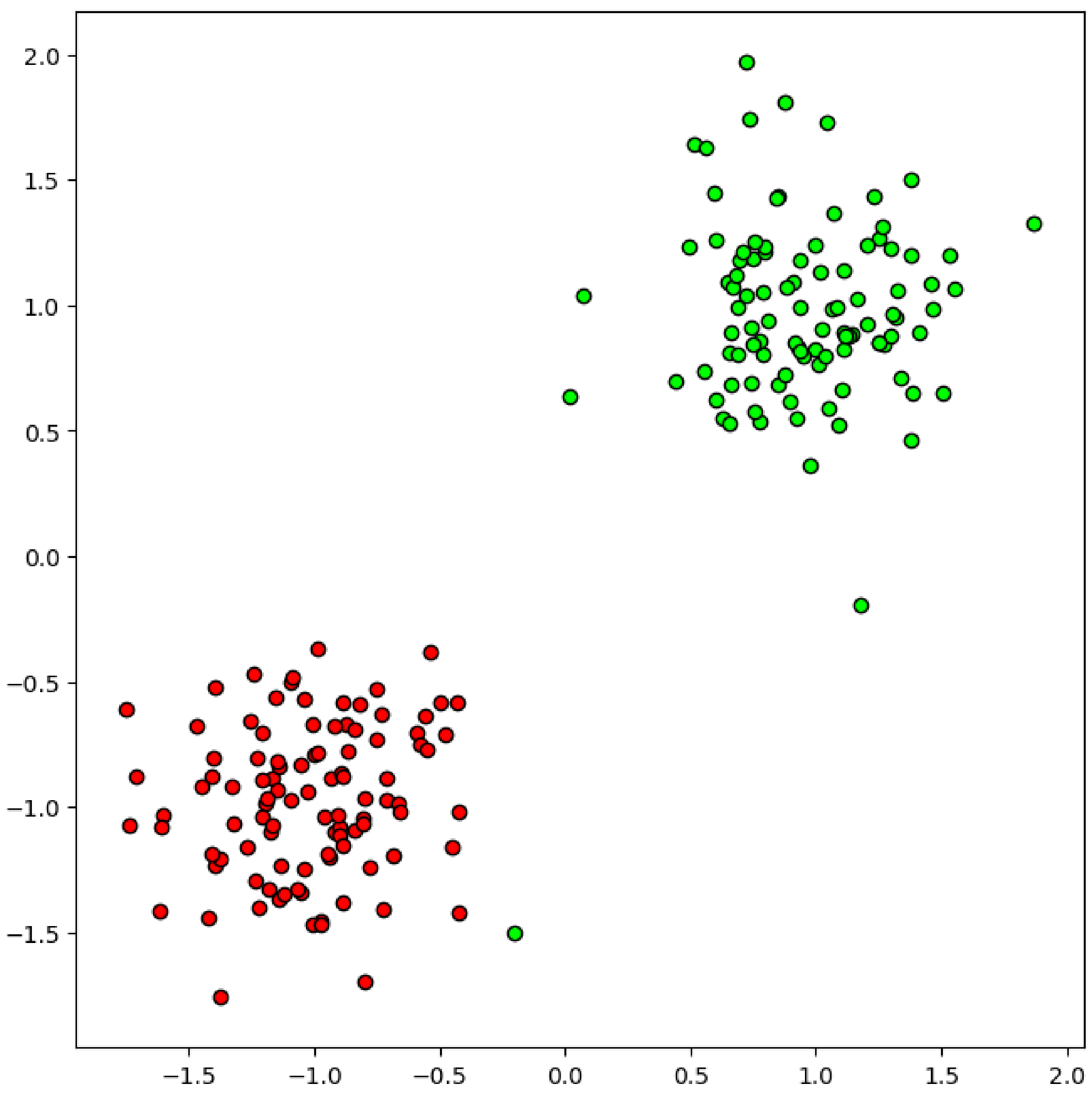
[452] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[453] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
[454] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html

procedures never satisfy a stopping criterion. Regularization guarantees existence of a minimizer und, thus, numerical stability. Linearly separable classes frequently occur for problems with few training data but many features.

To demonstrate the influence of regularization we consider binary classification with two classes 0 (red) and 1 (green) and two features. We use synthetic data with well separated classes and one outlier.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors
import sklearn.linear_model as linear_model

rng = np.random.default_rng(0)
```

```
n0 = 100    # samples in class 0
n1 = 100    # samples in class 1

# generate two point clouds and one outlier
X0 = rng.multivariate_normal([-1, -1], [[0.1, 0], [0, 0.1]], size=n0)
X1 = rng.multivariate_normal([1, 1], [[0.1, 0], [0, 0.1]], size=n1)
Xout = np.array([[-0.2, -1.5]])
X = np.concatenate((X0, X1, Xout))

# set labels
y0 = np.zeros(n0)
y1 = np.ones(n1)
yout = np.ones(1)
y = np.concatenate((y0, y1, yout))

# set plotting region
x0_min = X[:, 0].min() - 0.2
x0_max = X[:, 0].max() + 0.2
x1_min = X[:, 1].min() - 0.2
x1_max = X[:, 1].max() + 0.2

# plot data set
fig, ax = plt.subplots(figsize=(8,8))
ax.scatter(X[y == 0, 0], X[y == 0, 1], c='#ff0000', edgecolor='black')
ax.scatter(X[y == 1, 0], X[y == 1, 1], c='#00ff00', edgecolor='black')
ax.set_xlim(x0_min, x0_max)
ax.set_ylim(x1_min, x1_max)
ax.set_aspect('equal')
plt.show()
```



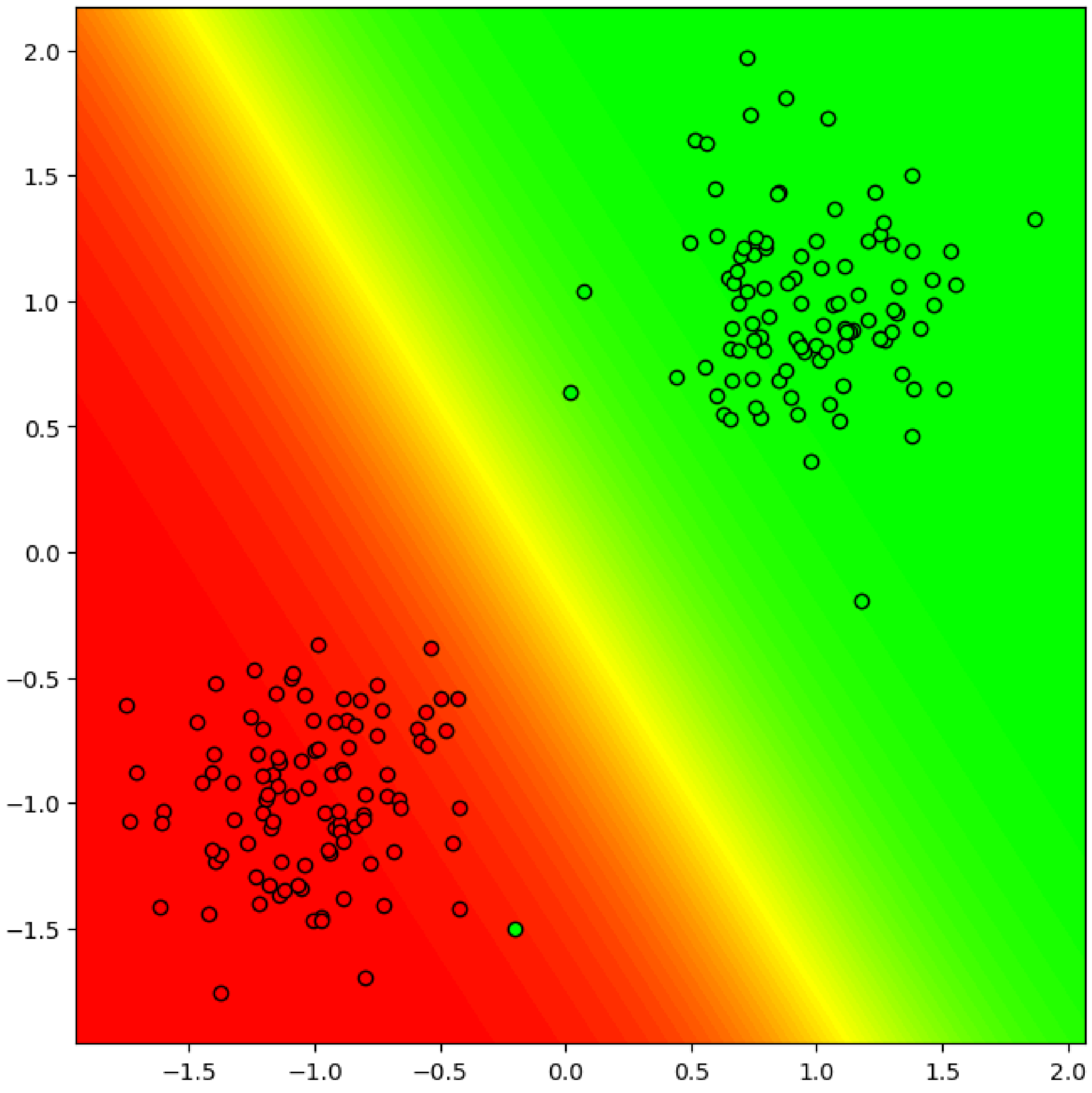
Logistic regression for binary classification results in a list of coefficients defining the model, which maps inputs to $(0, 1)$. We may visualizing the model with the following function.

```
def plot_linreg(ax, X, y, a, b, c):
    ''' a, b, c are the coefficients of the linear model: a + b * x0 + c * x1 '''

    # plot model (function values color-coded)
    x0, x1 = np.meshgrid(np.linspace(x0_min, x0_max, 100), np.linspace(x1_min, x1_
↪max, 100))
    y_grid = 1 / (1 + np.exp(-(a + b * x0 + c * x1)))
    cm = matplotlib.colors.LinearSegmentedColormap.from_list('ryg', ['#ff0000', '
↪#ffff00', '#00ff00'])
    ax.contourf(x0, x1, y_grid, cmap=cm, levels=np.linspace(0, 1, 50))

    # plot data set
    ax.scatter(X[y == 0, 0], X[y == 0, 1], c='#ff0000', edgecolor='black')
    ax.scatter(X[y == 1, 0], X[y == 1, 1], c='#00ff00', edgecolor='black')

    ax.set_xlim(x0_min, x0_max)
    ax.set_ylim(x1_min, x1_max)
    ax.set_aspect('equal')
```

With high regularization we obtain a good separating hyperplane neglecting the outlier. Relatively small coefficients result in a wide region in which predictions are close to $\frac{1}{2}$.

```
alpha = 1
logreg = linear_model.LogisticRegression(C=1/alpha)
logreg.fit(X, y)

a = logreg.intercept_[0]
b, c = logreg.coef_[0, :]
print(a, b, c)

fig, ax = plt.subplots(figsize=(8, 8))
plot_linreg(ax, X, y, a, b, c)
plt.show()
```

```
0.3275158741724573 2.742734564616664 1.7586424065927262
```



With almost no regularization we get perfect separation of both classes, but samples close to the red cloud might be missclassified as green (overfitting). Almost all inputs get classified close to 0 or close to 1. The region with mean predictions is very small.

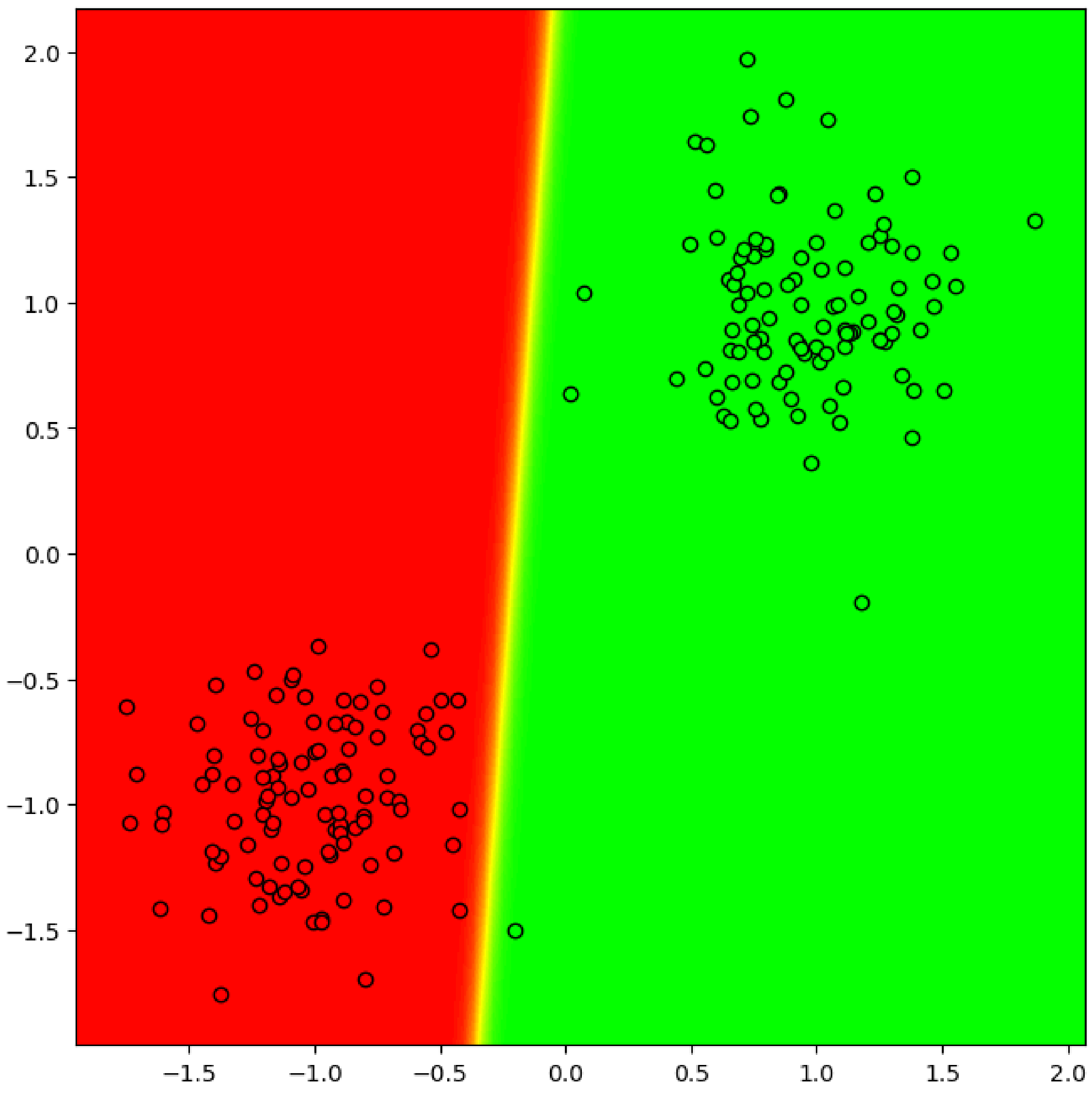
```
alpha = 1e-4
```

(continues on next page)

(continued from previous page)

```
logreg = linear_model.LogisticRegression(C=1/alpha)
logreg.fit(X, y)

a = logreg.intercept_[0]
b, c = logreg.coef_[0, :]
print(a, b, c)

fig, ax = plt.subplots(figsize=(8, 8))
plot_linreg(ax, X, y, a, b, c)
plt.show()
```

```
7.8650652464787765 38.74250199983403 -2.765877255810703
```



If we further decrease regularization we run into numerical difficulties, because coefficients become arbitrarily large. The fact that for linearly separable classes the minimization problem has no solution can be restated as: the minimizer lies at infinity. Note that Scikit-Learn stops minimization if a maximum number of iterations is reached or decrease of the objective function is below some tolerance level. Thus, to see the influence of too low regularization we have to adjust those parameters. Else minimization stops to early.

```
alpha = 1e-15
logreg = linear_model.LogisticRegression(C=1/alpha, tol=1e-7)
logreg.fit(X, y)

a = logreg.intercept_[0]
b, c = logreg.coef_[0, :]
print(a, b, c)

np.exp(-(a + b * -2.1 + c * 1))
```

```
33.381878479995414 149.96785067058389 -9.427364096624917
```

```
2.3449177053285104e+126
```

25.5 Decision Boundaries for Multiclass Classification

Classification methods divide the feature space into a set of mutually disjoint subsets, one subset per class. The boundaries between those subsets are called decision boundaries. For binary logistic regression with threshold-based classification boundary the decision boundary between the two classes is a hyperplane.

In multiclass classification with classes $1, \ldots, C$ the class with the highest predicted probability is choses as predicted class. Applying softmax to scores does not change the ordering of the scores. Thus, the class with the highest score is chosen. In logistic regression scores are linear functions of the feature values. If we want to decide whether class 1 is chosen as prediction for a given feature vector x we have to compare corresponding score to scores of all other classes. The prediction for x is class 1 if and only if

$$\mathring{a}_1^{\mathrm{T}}\,\mathring{x} \geq \mathring{a}_2^{\mathrm{T}}\,\mathring{x}, \quad \ldots, \quad \mathring{a}_1^{\mathrm{T}}\,\mathring{x} \geq \mathring{a}_C^{\mathrm{T}}\,\mathring{x},$$

where $\mathring{a}_C$ contains zeros (see above). Equivalently,

$$(\mathring{a}_1 - \mathring{a}_2)^{\mathrm{T}}\,\mathring{x} \geq 0, \quad \ldots, \quad (\mathring{a}_1 - \mathring{a}_C)^{\mathrm{T}}\,\mathring{x} \geq 0.$$

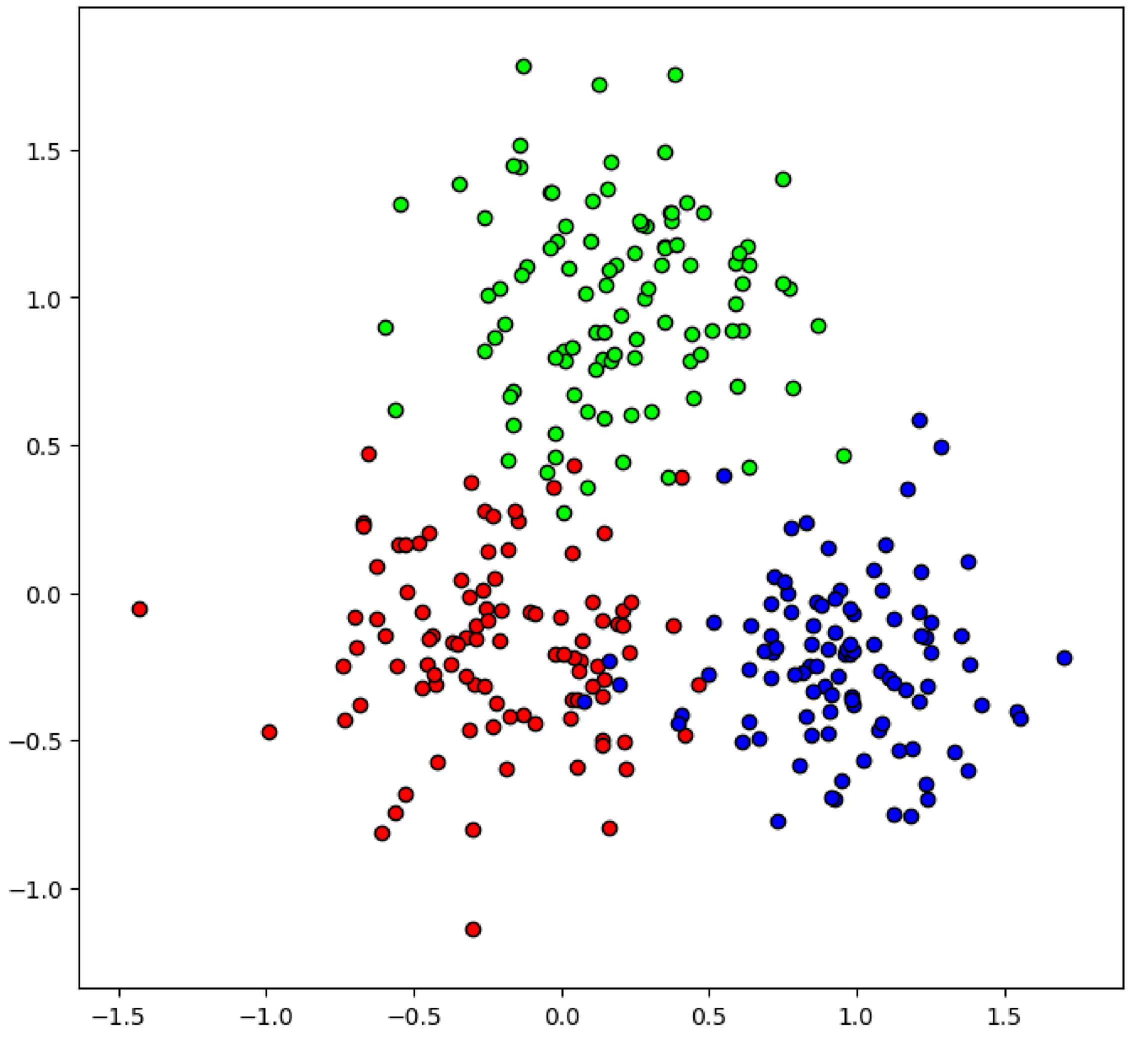
The set of feature vectors x satisfying all these inequalities is the intersection of $C - 1$ halfspaces (also known as C-dimensional polytope).

```
n1 = 100    # samples in class 0
n2 = 100    # samples in class 1
n3 = 100    # samples in class 2

# generate three point clouds
X1 = rng.multivariate_normal([-0.2, -0.2], [[0.1, 0], [0, 0.1]], size=n1)
X2 = rng.multivariate_normal([0.2, 1], [[0.1, 0], [0, 0.1]], size=n2)
X3 = rng.multivariate_normal([1, -0.2], [[0.1, 0], [0, 0.1]], size=n3)
X = np.concatenate((X1, X2, X3))

# set labels
y1 = 1 * np.ones(n1)
y2 = 2 * np.ones(n2)
y3 = 3 * np.ones(n3)
y = np.concatenate((y1, y2, y3))

# set plotting region
x0_min = X[:, 0].min() - 0.2
x0_max = X[:, 0].max() + 0.2
x1_min = X[:, 1].min() - 0.2
x1_max = X[:, 1].max() + 0.2
```

(continues on next page)

(continued from previous page)

```
# plot data set
fig, ax = plt.subplots(figsize=(8,8))
ax.scatter(X[y == 1, 0], X[y == 1, 1], c='#ff0000', edgecolor='black')
ax.scatter(X[y == 2, 0], X[y == 2, 1], c='#00ff00', edgecolor='black')
ax.scatter(X[y == 3, 0], X[y == 3, 1], c='#0000ff', edgecolor='black')
ax.set_xlim(x0_min, x0_max)
ax.set_ylim(x1_min, x1_max)
ax.set_aspect('equal')
plt.show()
```



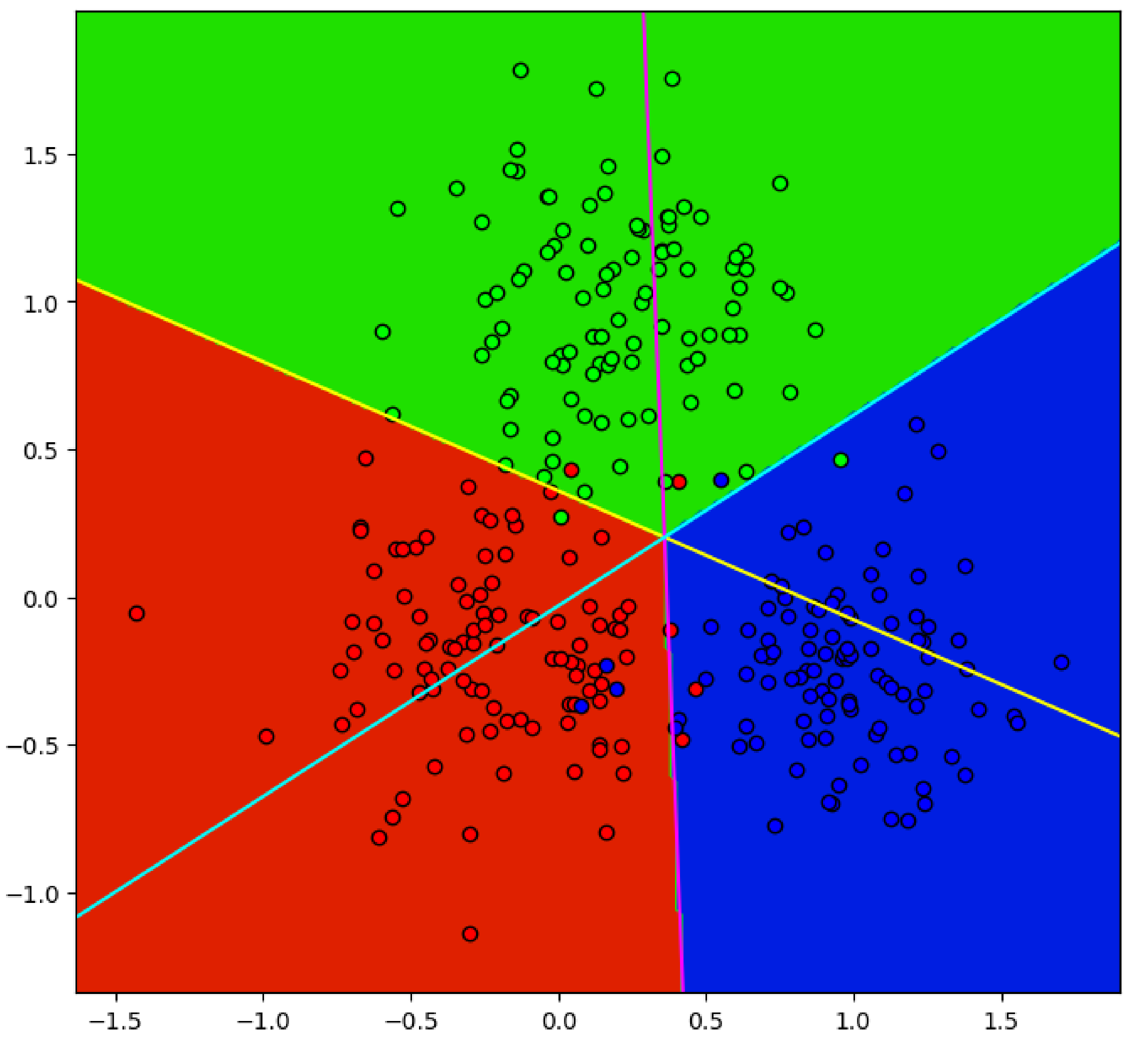
```
def plot_hyperplane(ax, a, b, c, color, style='-'):

    ax.plot([x0_min, x0_max], [-a/c - b/c * x0_min, -a/c - b/c * x0_max], style,␣
↪color=color)
```

```
alpha = 1
logreg = linear_model.LogisticRegression(C=1/alpha)
logreg.fit(X, y)

fig, ax = plt.subplots(figsize=(8, 8))

# regions
x0, x1 = np.meshgrid(np.linspace(x0_min, x0_max, 200), np.linspace(x1_min, x1_max,
↪ 200))
```

(continues on next page)

(continued from previous page)

```
y_grid = logreg.predict(np.stack((x0.reshape(-1), x1.reshape(-1)), axis=1)).
↪reshape(x0.shape)
cm = matplotlib.colors.LinearSegmentedColormap.from_list('rgb', ['#ff0000', '
↪#00ff00', '#0000ff'])
ax.contourf(x0, x1, y_grid, cmap=cm)

# data set
ax.scatter(X[y == 1, 0], X[y == 1, 1], c='#ff0000', edgecolor='black')
ax.scatter(X[y == 2, 0], X[y == 2, 1], c='#00ff00', edgecolor='black')
ax.scatter(X[y == 3, 0], X[y == 3, 1], c='#0000ff', edgecolor='black')

a1, a2, a3 = logreg.intercept_
b1, b2, b3 = logreg.coef_[:, 0]
c1, c2, c3 = logreg.coef_[:, 1]

# decision boundaries (halfspaces)
plot_hyperplane(ax, a1 - a2, b1 - b2, c1 - c2, '#ffff00')
plot_hyperplane(ax, a1 - a3, b1 - b3, c1 - c3, '#ff00ff')
plot_hyperplane(ax, a2 - a3, b2 - b3, c2 - c3, '#00ffff')

# per class decision boundary (threshold = 0.5)
#plot_hyperplane(ax, a1, b1, c1, '#ff0000', '--')
#plot_hyperplane(ax, a2, b2, c2, '#00ff00', '--')
#plot_hyperplane(ax, a3, b3, c3, '#0000ff', '--')

ax.set_xlim(x0_min, x0_max)
ax.set_ylim(x1_min, x1_max)
ax.set_aspect('equal')

plt.show()
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

CHAPTER

TWENTYSIX

ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANNs) are a fundamental technique in modern machine learning and artificial intelligence. The buzzword *deep learning* is used for artificial neural networks with many layers of neurons. In this chapter we consider layered ANNs and the important special case *convolutional ANNs (CNNs)*.

Related projects:

26.1 ANN Basics

Supervised learning aims at approximating a function $f : X \rightarrow Y$ by a function $f_{\text{approx}} : X \rightarrow Y$ based on a finite set of examples $(x_1, y_1), \ldots, (x_n, y_n)$ satisfying $f(x_l) = y_l$ or at least $f(x_l) \approx y_l$ for $l = 1, \ldots, n$. Almost always we have finite dimensional feature spaces $X = \mathbb{R}^m$ and target spaces $Y = \mathbb{R}$.

A good hypothesis f_{approx} has to satisfy $f_{\text{approx}}(x) \approx f(x)$ for all x from the example set (good fit on training set) and for all other x expected to appear in the underlying practical problem (good generalization). To construct good hypotheses we have to pose additional assumptions on f_{approx}.

In linear regression we assume that the hypothesis is a linear combination of several prescribed basis functions. The coefficients are chosen to minimize the fitting error on the training set. Artificial neural networks follow the same idea: take some function containing several parameters and choose parameters such that the fitting error on the training set is small. The only difference to linear regression is the chosen ansatz. Typically one does not write down an explicit formula for f_{approx}, but one provides a graphical scheme containing all information about the hypothesis. In contrast to linear regression, ANNs contain the parameters in a nonlinear fashion, resulting in more difficult minimization procedures. ANNs thus are an example for nonlinear regression. ANNs can be used for classification, too (see below).

26.1.1 Motivation from Biology

ANNs originated from the wish to simulate human brains. A brain consists of many nerve cells (neurons) interconnected to transmit information (electrical pulses). All nerve cells have similar structure. The strength of interconnections between different nerve cells may vary and it is this varying strength which allows humans to learn new things. Learning, as far as we know, is realized by changing the strength of interconnections between nerve cells and, thus, reducing or improving the flow of information between different cells. A neuron takes all the electrical pulses from connected cells (inputs) and generates an output pulse from the inputs.

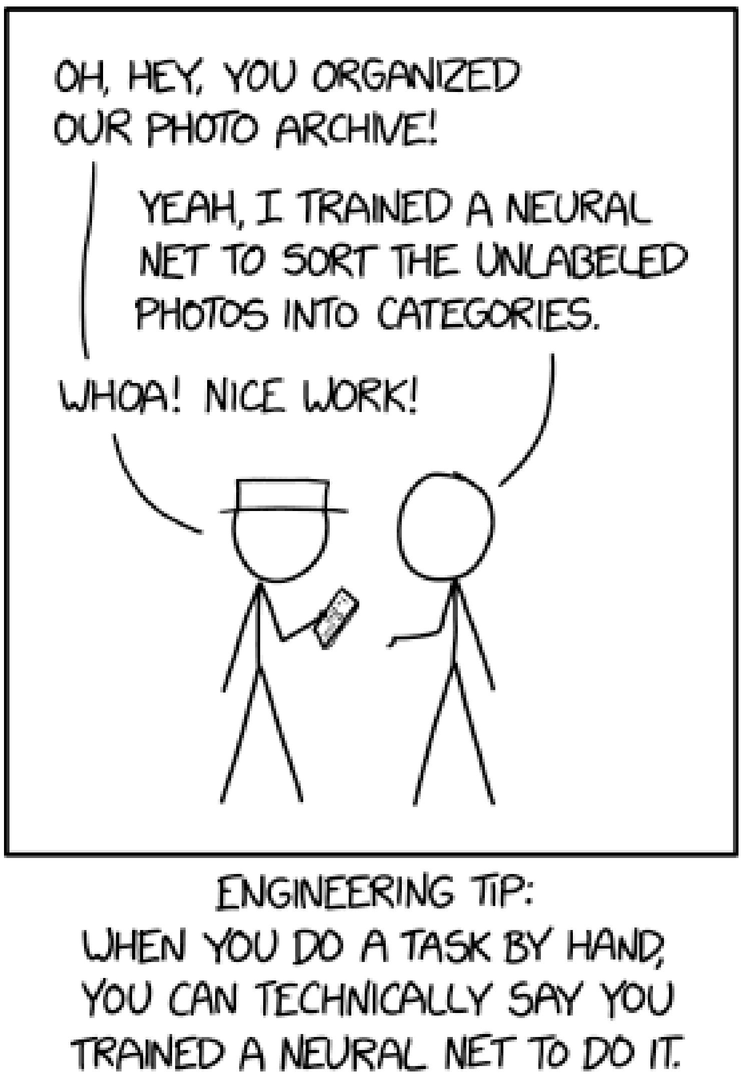

OH, HEY, YOU ORGANIZED OUR PHOTO ARCHIVE!
YEAH, I TRAINED A NEURAL NET TO SORT THE UNLABELED PHOTOS INTO CATEGORIES.
WHOA! NICE WORK!
ENGINEERING TIP:
WHEN YOU DO A TASK BY HAND, YOU CAN TECHNICALLY SAY YOU TRAINED A NEURAL NET TO DO IT.

Fig. 26.1: It also works for anything you teach someone else to do. “Oh yeah, I trained a pair of neural nets, Emily and Kevin, to respond to support tickets.” Source: Randall Munroe, xkcd.com/2173[455]

Of course human brains are much more complicated than described here and many mechanisms are not well understood. But the idea of many interconnected simple units forming a large powerful machine seems to be a key to artificial intelligence. ANNs try to simulate such networks of neurons on a digital computer.

Next to ANNs there exist several other ideas based on the concept of connecting many simple units. If you are interested have a look at cellular automata[456] and collective intelligence[457].

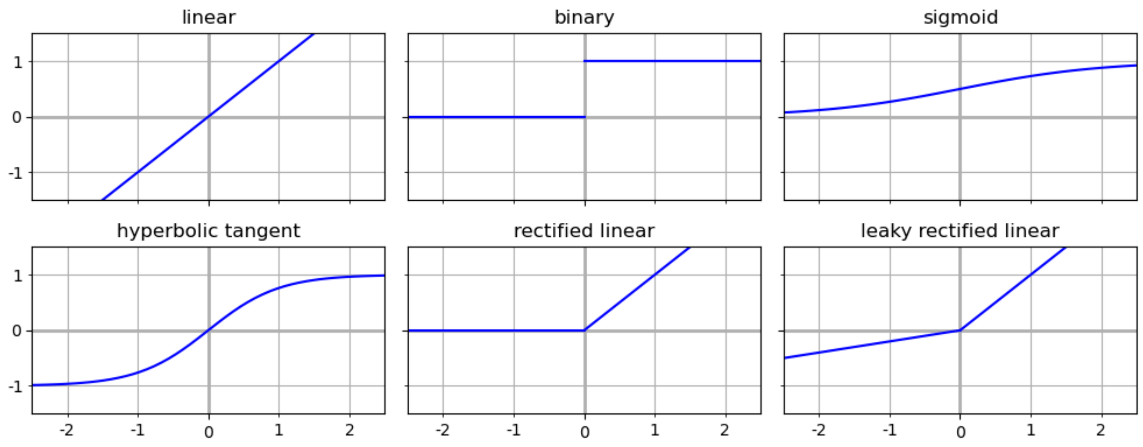
[455] https://xkcd.com/2173
[456] https://en.wikipedia.org/wiki/Cellular_automaton
[457] https://en.wikipedia.org/wiki/Collective_intelligence

26.1.2 Artificial Neurons

ANNs are composed of artificial neurons, mimicking biological neurons. An artificial neuron is a function taking p inputs and yielding one output. Inputs and outputs are real numbers. Each input is multiplied by a *weight*, then all the products are added, and an *activation function* is applied to the sum. The outcome of the activation function is the neuron's output.

Weights correspond to the strength of interconnections between biological neurons. The activation function simulates the fact, that a biological neuron fires (that is, generates an output pulse) only if the level and number of input pulses is high enough.

Activation functions almost always are monotonically increasing. Some examples:



More activation functions are shown in the Wikipedia article on activation functions[458]. Which activation function to choose depends on the underlying practical problem and heavily on experience.

Denoting the inputs by $u_1, \ldots, u_p \in \mathbb{R}$, the weights by $w_1, \ldots, w_p \in \mathbb{R}$, the activation function by $g : \mathbb{R} \to \mathbb{R}$, and the output by $v \in \mathbb{R}$, we have

$$v = g\left(\sum_{\kappa=1}^{p} w_\kappa\, u_\kappa\right).$$

If u is the vector of inputs and w is the vector of weights, we may write

$$v = g(w^{\mathrm{T}}\, u).$$

26.1.3 Networks of Artificial Neurons

The simplest ANN consists of only one neuron. It takes the the feature values $x^{(1)}, \ldots, x^{(m)}$ of a feature vector x as inputs, that is, $p = m$, and the output is interpreted as prediction for the corresponding target $f(x)$.

We could also take more neurons and feed them with all or some of the feature values. Then the outputs of all neurons may be fed to one or more other neurons and so on. This way we obtain a network of neurons similar to biological neural networks (brains). The output of one of the neurons is interpreted as prediction for the targets.

ANNs can be represented graphically. Each neuron is a circle or rect containing information about the activation function used by the neuron. Connections between inputs and outputs are lines and the weights are numbers assigned to the corresponding input's line.

The depicted ANN contains 5 neurons. It's a special case of a fully connected two-layered feedforward network. These terms will be introduced below. We may write down corresponding hypothesis f_{approx} as mathematical formula. Denote the weight vectors by w, $\hat{w}$, $\mathring{w}$, $\tilde{w}$, $\bar{w}$ and the activation functions by g, $\hat{g}$, $\mathring{g}$, $\tilde{g}$, $\bar{g}$. Then we have

$$f_{\text{approx}}(x) = \bar{g}\Big(\bar{w}_1\, g(w^{\mathrm{T}}\, x) + \bar{w}_2\, \hat{g}(\hat{w}^{\mathrm{T}}\, x) + \bar{w}_3\, \mathring{g}(\mathring{w}^{\mathrm{T}}\, x) + \bar{w}_4\, \tilde{g}(\tilde{w}^{\mathrm{T}}\, x)\Big)$$

[458] https://en.wikipedia.org/wiki/Activation_function#Comparison_of_activation_functions

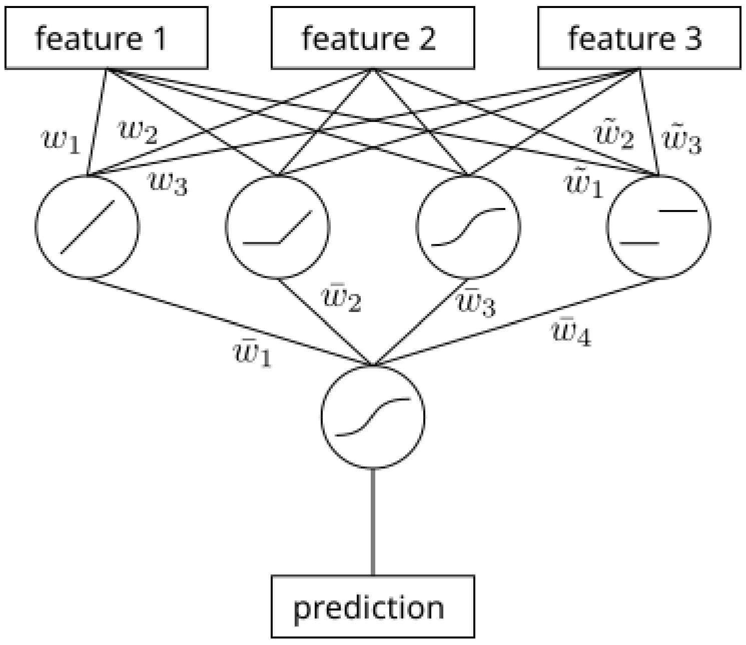


Fig. 26.2: A simple ANN with 5 neurons. Neurons are depicted as circles. Rectangles symbolize input and ouput values.

with 16 parameters (all the weights). Those 16 parameters have to be chosen to solve

$$\frac{1}{n}\sum_{l=1}^{n}\left(f_{\text{approx}}(x_l)-y_l\right)^2\to\min_{\text{weights}}$$

with training samples $(x_1,y_1),\dots,(x_n,y_n)$. Below we will discuss how to solve such nonlinear minimization problems numerically.

26.1.4 Feedforward and Layered Networks

There are many kinds of ANNs and we will meet most of them when going on studying data science. The simplest and most widely used type of ANNs are feedforward networks. Those are networks in which information flows only in one direction. The feature values are fed to a set of neurons. Corresponding outputs are fed to a different set of neurons and so on. The process always ends with a single neuron yielding the prediction. No neuron is used twice. In contrast there are ANNs which feed a neuron's output back to another neuron involved in generating the neuron's input. Such ANNs contain circles and it is not straight forward how to compute the ANN's output. It's a dynamic process which may converge or not. Although such ANNs are more close to biological neural networks, they are rarely used because of their computational complexity. Only very special and well structured non-feedforward ANNs appear in practice.

To allow for more efficient computation, feedforward ANNs often are organized in layers. A layer is a set of neurons with no interconnections. Neurons of a layer only have connections to other layers. Layers are organized sequentially. Inputs of the first layer's neurons are connected to the network inputs (feature values). Outputs are connected to the inputs of the second layer's neurons. Outputs from second layer are connected to inputs of third layer and so on. The last layer has only one neuron yielding the ANN's output.

A layer may be fully connected to previous and next layer or some connections may be missing (corresponding weights are fixed to zero). Networks with all layers fully connected are called *dense networks*.

Computational efficiency of layered feedword networks stems from the fact that the outputs of all neurons in a layer can be computed simultaneously by matrix vector multiplication. Matrix vector multiplication is a very fast operation on modern computers, especially if additional GPU (graphics processing unit) capabilities are available.

If u is the vector of inputs of a layer (that is, the vector of outputs of the previous layer), then to get the output of each neuron we have to compute the inner products $w^{\mathrm{T}}\,u$ with w being different for each neuron. Taking all the weight

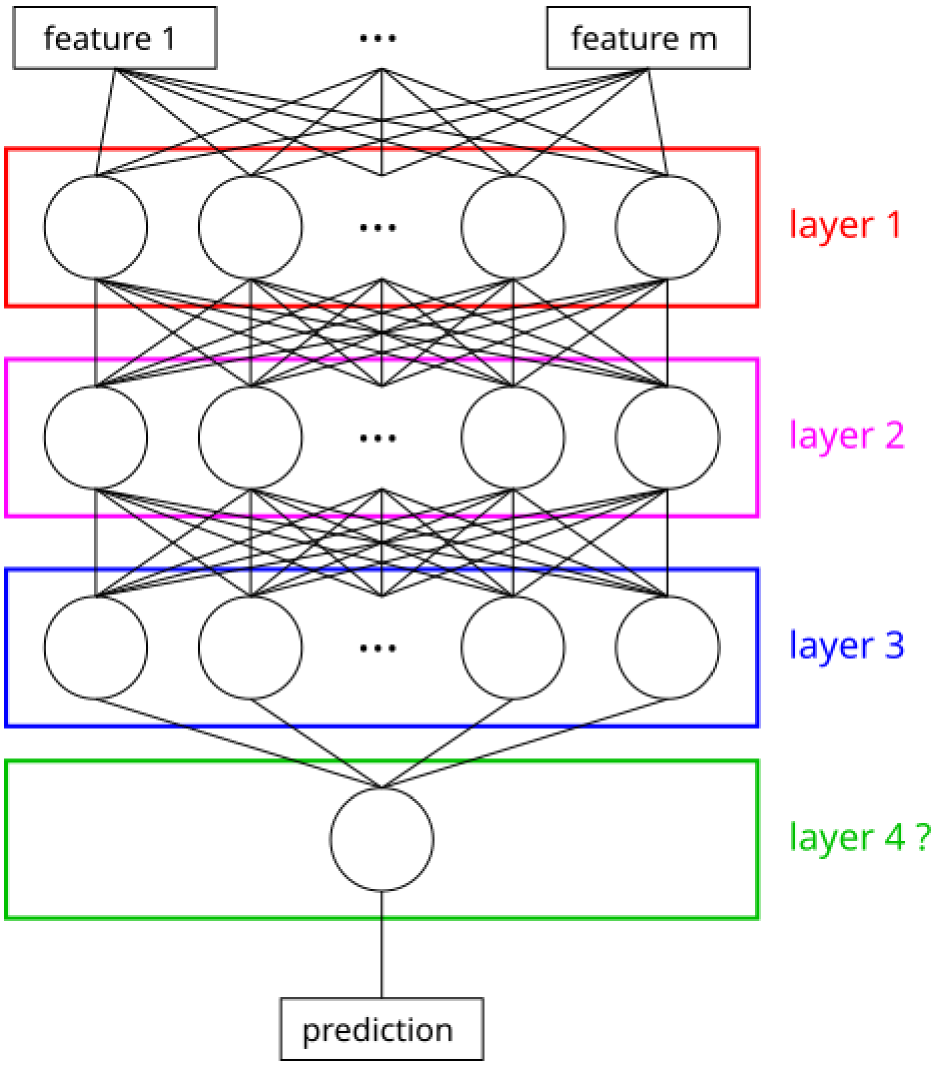


Fig. 26.3: A layered ANN with 3 or 4 layers (depending on the definition of *number of layers*).

vectors of the neurons in the layer as rows of a matrix W the components of $W\,u$ are exactly those inner products. If all neurons in the layer use the same activation function (which typically is the case), then we simply have to apply the activation function to all components of $W\,u$ to get the layer's outputs.

In a three-layered network with weight matrices W_1, W_2, W_3 and (per layer) activation functions g_1, g_2, g_3 we would have

$$f_{\text{approx}}(x) = g_3\Big(W_3\,g_2(W_2\,g_1(W_1\,x))\Big),$$

where the activation functions are applied componentwise.

26.1.5 Training ANNs

Training an ANN means solving the minimization problem

$$\frac{1}{n}\sum_{l=1}^{n}\big(f_{\text{approx}}(x_l) - y_l\big)^2 \to \min_{\text{weights}}.$$

The dependence of f_{approx} on the weights is highly nonlinear. Thus, there is no simple analytical solution. Instead we have to use numerical procedures to find weights which are at least close to minimizing weights.

The basic idea of such numerical algorithms is to start with arbitrary weights and to improve weights iteratively. Next to several more advanced techniques, there is a class of algorithms known as *gradient descent method*. They take the gradient of the objective function to calculate improved weights. The negative gradient is the direction of steepest descent. Thus, it should be a good idea to modify weights by substracting the gradient from the current weights. We stop the iteration, if the gradient is close to zero, that is, if we reached a stationary point.

Gradient descent methods suffer from different problems:

- We need to calculate the gradient of the objective with respect to the weights analytically. Due to the structure of ANNs this involves repeated application of the chain rule for differentiation.
- Usually we end up in an arbitrary stationary point. With some luck it's at least a local minimizer. Finding the global minimizer of a nonlinear function with gradient descent is almost impossible.
- Convergence is slow. We have to do many iterations to find a stationary point.

Due to their simplicity, gradient descent methods are the standard technique for training ANNs. More involved methods only work for special ANNs, whereas gradient descent is almost always applicable.

We will cover the details of gradient descent in a subsequent chapter.

26.1.6 Overfitting and Regularization

Large ANNs tend to overfit the training data. As for linear regression we might add a penalty to the objective function to avoid overfitting. Concerning overfitting and regularization there is no difference between linear regression and ANNs.

For ANNs there also exist regularization techniques not applicable to linear regression. One such technique is kown as *drop out*. In each training step the weights of a randomly selected set of neurons are held fixed, that is, they are excluded from training. This set changes from step to step and the size of the set is a hyperparameter. The idea is to get more redundancies in the ANN and, thus, more reliable predictions. Especially, generalization power can be improved by drop out.

26.1.7 Hyperparameters

ANNs contain two obvious hyperparameters:

- number of layers,
- number of neurons in each layer.

But activation functions may be regarded as hyperparameters, too, since we have to choose them in advance.

There is no essential difference between tuning hyperparameters for ANNs and tuning hyperparameters for linear regression.

26.1.8 Bias Neurons

Artificial neurons suffer from a problem with inputs being all zero. If all inputs are zero, then multiplication with weights yields zero, too. Activation functions again map zero to zero. Thus, artificial neurons are not able to give a nonzero response to all-zero inputs.

One solution would be to use activation functions with nonzero activation for zero input. But this would contradict the idea of an activation function and we would have to add parameters to activation functions to get variable output for all-zero inputs.

A better idea is to add a *bias neuron* to each layer. A bias neuron takes no inputs and always yields the number one as its output. Neurons in the next layer connected to the bias neuron of the previous layer now always have nonzero input. With the corresponding weight we are able to adjust the size of the input. Even if all regular inputs are zero we are able to yield nonzero neuron output this way.

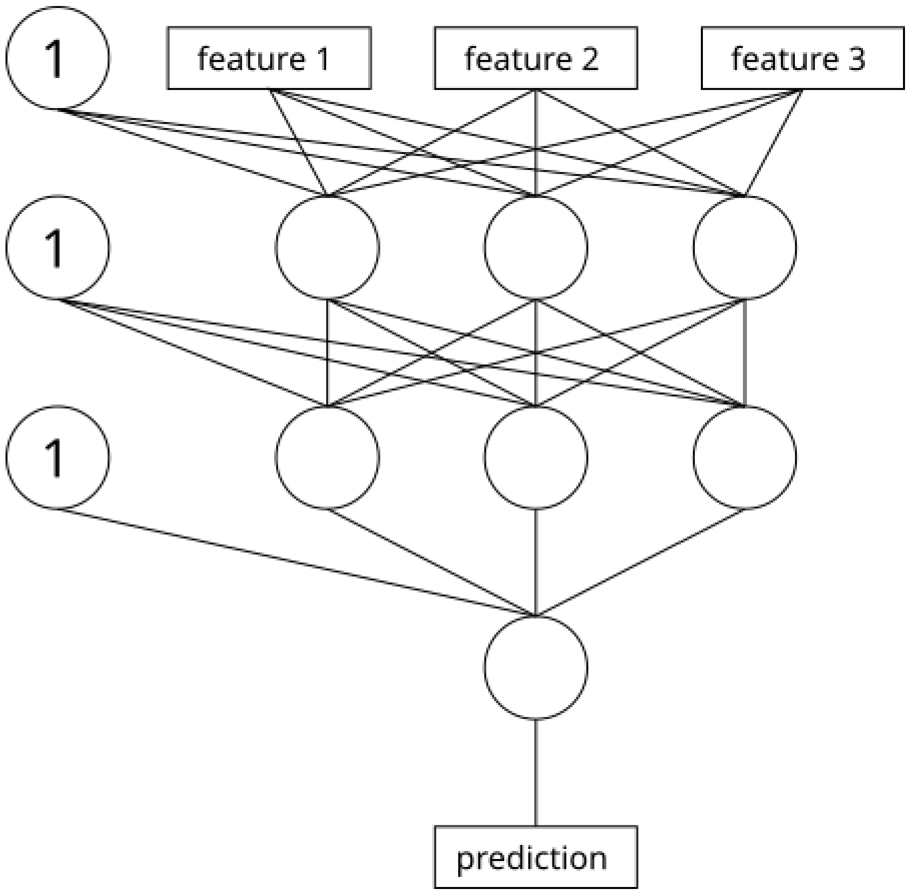


Fig. 26.4: Bias neurons are represented by circles with a one inside.

Denote the activation function of a neuron by g. If w_0 is the weight for the input from the bias neuron and if w and u are the vectors of regular weights and inputs, respectively, then the neuron's output is

$$g(w_0 + w^{\mathrm{T}} u).$$

We see that using bias neurons, the activation function is shifted to the left or to the right, depending on the weight w_0.

For instance, if we use the activation function

$$g(t) = \begin{cases} 1, & \text{if } t > 0, \\ 0, & \text{else}, \end{cases}$$

which fires if and only if the weighted inputs add up to a positive number, then introducing a bias neuron, we obtain

$$g(w_0 + w^{\mathrm{T}} u) = \begin{cases} 1, & \text{if } w^{\mathrm{T}} u > -w_0, \\ 0, & \text{else}. \end{cases}$$

That is, the neuron fires if the sum of weighted regular inputs lies above $-w_0$. In this special case, bias neurons allow for modifying the threshold for acitvation without modifying the activation function.

From the training point of view, bias neurons do not matter, because they have inputs and thus no weights to train.

26.1.9 Approximation Properties

In linear regression it is obvious which types of functions can be represented by the ansatz for f_{approx} (linear functions, polynomials, and so on). For ANNs we have to look more closely. Representable function classes depend on the activation function, on the number of layers, and on the number of neurons in each layer.

For instance, if we have only one layer and we use threshold activation (zero or one), then f_{approx} always is a piecewise constant function. With rectified linear units we always would obtain piecewise linear functions.

Considering more than one layer, things become tricky. But an important result in the theory of ANNs states, that an ANN with at least one layer is able to approximate arbitrary continuous functions. We simply have to use enough neurons. The more neurons the better the approximation.

The number of neurons required for good approximation in a single layer ANN might be very large. Often it is computationally more efficient to have more layers with less neurons. There exist many results on approximation properties of ANNs. The keyword is *universal approximation theorems*.

26.1.10 Vector-valued Regression

Up to now we only considered approximating realvalued functions of several variables (features), that is, the underlying truth has continuous range in $\mathbb{R}$. If we want to approximate functions f taking values in some higher dimensional space $\mathbb{R}^d$, then we could apply linear regression or ANNs to each of the d components of the function independently (multiplying computational cost by factor d).

In constrast to linear regression, ANNs allow for more natural extension to multiple outputs. We simply have to add some neurons to the output layer. This way, the 'knowledge' of the ANN can be used by all outputs without training individual nets for each output component.

Squared error loss for vector-valued regression is

$$\frac{1}{n} \sum_{l=1}^{n} \left| f_{\text{approx}}(x_l) - y_l \right|^2,$$

where $f_{\text{approx}}(x_l) - y_l$ is a vector with d components and $|\cdot|$ denotes the length of a vector.

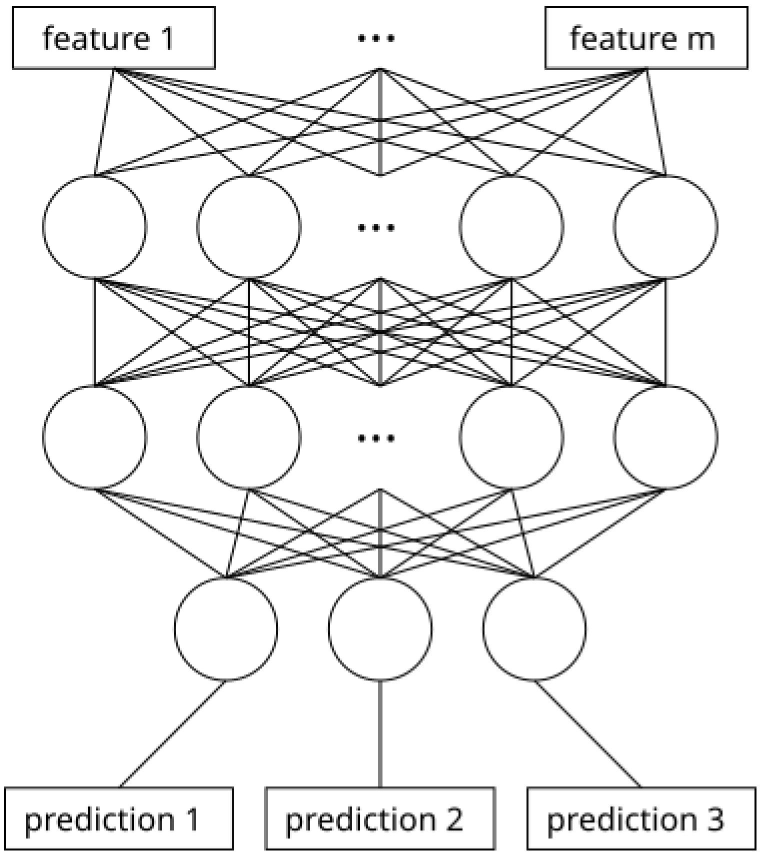


Fig. 26.5: An ANN with three outputs. Number of outputs is independent of the ANN's overall structure.

26.1.11 ANNs for Classification

An import application of multiple output ANNs are classification tasks. Classification differs from regression in that the range of the truth f is discrete with only few different values (classes). If we predict for each class the probability that a feature vector belongs to this class, we have a regression problem with multiple outputs. Thus, ANNs can be used for solving classification problems, too.

ANNs may adapted to classification task in several ways, which we briefly discuss here.

Number of Outputs

Especially for binary classification task we have to decide whether the ANN has one output or two outputs. With one output (and sigmoid activation in the output neuron) predictions close to 1 indicate one class, predictions close to 0 indicate the other class. With two output neurons (sigmoid each) the ANN is able to predict 'both classes' (both outputs close to one) or 'no class' (both outputs close to zero). Which variant to choose depends on the context.

For multiclass classification with C class the analog question is whether to use C or $C-1$ output neurons.

Softmax Activation

Having as many output neurons as classes may result in unclear predictions if sigmoid activation is used in each output neuron (multiple outputs could be close to one). Better for interpretation would be a scoring procedure guaranteeing that all scores add up to 1. Then scores can be interpreted as probability that the input belongs to a certain class.

To implement probability-like scores in ANNs there is the *softmax activation function*. Strictly speaking it's not an activation function because it does not work per neuron but applies jointly to the activations of several neurons. For a classification problem with C classes denote by $a_1, \ldots, a_C$ the output neurons' activations (weighted sums of inputs plus bias) and by $o_1, \ldots, o_C$ the outputs (activation function applied to activations). Then softmax activation computes

$$o_i = \frac{\mathrm{e}^{a_i}}{\sum\limits_{j=1}^{C} \mathrm{e}^{a_j}} \qquad \text{for } i = 1, \ldots, C.$$

The exponential function maps (arbitrary) activations to $(0, \infty)$ and results are weighted to add up to 1.

Log Loss

Mean squared error loss also works for probability-like scores (ANN outputs). But log loss is much more suitable because it interprets ANN outputs as probabilities and computes the overall predicted probability on a test set, that all samples belong to their true class.

26.2 Training ANNs

In this chapter we have a close look at gradient descent methods and implement a complete ANN algorithm. Then we went on to Scikit-Learn's ANN routines.

26.2.1 Gradient Descent for Nonlinear Minimization Problems

Consider a function $h : \mathbb{R}^p \to \mathbb{R}$ and the corresponding minimization problem

$$h(w_1, \ldots, w_p) \to \min_{w \in \mathbb{R}^p}.$$

The gradient

$$\nabla h(w_1, \ldots, w_p) = \begin{bmatrix} \frac{\partial}{\partial w_1} h(w_1, \ldots, w_p) \\ \vdots \\ \frac{\partial}{\partial w_p} h(w_1, \ldots, w_p) \end{bmatrix}$$

is the vector of partial derivatives of h with respect to all variables $w_1, \ldots, w_p$. The gradient is known to be the direction of steepest ascent. In other words, $-\nabla h(w)$ is the direction of steepest descent of h at w.

To find a minimizer of h we might start at some point $w^{(0)}$, substract the gradient $\nabla h(w^{(0)})$ giving a new point $w^{(1)}$, substract $\nabla h(w^{(1)})$, and so on. This way function values should become smaller step by step. The problem is that the negative gradient only provides a direction, but no information about how far we should go in this direction. Thus, we have to introduce a parameters $s_0, s_1, \ldots$ for each step controlling the step length.

For general directions we have the following algorithm:

1. Choose a starting point $w^{(0)}$.
2. Repeat for $i = 0, 1, \ldots$:
    1. Choose a direction r_i.
    2. Choose a step length s_i.
    3. Set $w^{(i+1)} = w^{(i)} + s_i\, r_i$
    4. If $|w^{(i+1)} - w^{(i)}|$ is small enough, then stop iteration.

In the simplest case we would choose

$$r_i = -\nabla h(w^{(i)}) \qquad \text{and} \qquad s_i = s$$

with a constant step length $s > 0$. With this choice we obtain the *steepest descent method*. It is not guaranteed to converge. Convergence means, that the stopping cirterion is satisfied after sufficiently many steps. Small s yields higher chances for convergence, but many iterations are required. Large s decreases the number of iterations, but the method may not converge.

To improve performance directions different from the negative gradient can be used. There is a long list of sensible directions and step lengths, but details are out of this book's scope.

26.2.2 Gradient Descent for ANNs

To apply gradient descent for training ANNs we have to compute the gradient of the objective

$$h(w) = \frac{1}{n}\sum_{l=1}^{n}\bigl(f_{\text{approx}}(x_l) - y_l\bigr)^2$$

with respect to th weight vector w containing all weights of the ANN.

For each component of the gradient $\nabla_w h(w)$ we have

$$\frac{\partial}{\partial w_\kappa} h(w) = \frac{1}{n}\frac{\partial}{\partial w_\kappa}\sum_{l=1}^{n}\bigl(f_{\text{approx}}(x_l) - y_l\bigr)^2 = \frac{1}{n}\sum_{l=1}^{n}\frac{\partial}{\partial w_\kappa}\bigl(f_{\text{approx}}(x_l) - y_l\bigr)^2.$$

Chain rule yields

$$\frac{\partial}{\partial w_\kappa} h(w) = \frac{1}{n}\sum_{l=1}^{n} 2\,\bigl(f_{\text{approx}}(x_l) - y_l\bigr)\,\frac{\partial}{\partial w_\kappa} f_{\text{approx}}(x_l).$$

Thus,

$$\nabla h(w) = \frac{2}{n}\sum_{l=1}^{n}\bigl(f_{\text{approx}}(x_l) - y_l\bigr)\,\nabla f_{\text{approx}}(x_l).$$

If we take $\nabla f_{\text{approx}}(x_1), \ldots, \nabla f_{\text{approx}}(x_n)$ as columns of a matrix $G \in \mathbb{R}^{p \times n}$ with p being the total number of weights and if we set

$$y_{\text{pred}} := \begin{bmatrix} f_{\text{approx}}(x_1) \\ \vdots \\ f_{\text{approx}}(x_n) \end{bmatrix} \qquad \text{and} \qquad y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix},$$

we obtain

$$\nabla h(w) = \frac{2}{n}\, G\,(y_{\text{pred}} - y).$$

It remains to find the gradient of $f_{\text{approx}}(x)$ with respect to the weight vector w for some feature vector x. This gradient heavily depends on the structure of the ANN.

Single Layer ANNs

We compute the gradient $\nabla f_{\text{approx}}(x)$ for an ANN with only one layer. The main difficulty is to find manageable notation. We have m features, q regular neurons, two bias neurons and one output neuron.

We number the regular neurons by $1, \ldots, q$. Weights of the regular neuron with number μ are denoted by $w_0^{\text{in},\mu}, w_1^{\text{in},\mu}, \ldots, w_m^{\text{in},\mu}$, where $w_0^{\text{in},\mu}$ is the weight of the bias input and the others are for the m feature inputes. Weights of the output neuron are denoted by $w_0^{\text{out}}, w_1^{\text{out}}, \ldots, w_q^{\text{out}}$. Again, w_0^{out} is for the bias input and the others are for the inputs from the q regular neurons. Activation functions are g_{in} for all regular neurons and g_{out} for the output neuron.

With this notation we have

$$w = \begin{bmatrix} w_0^{\text{in},1} & \cdots & w_m^{\text{in},1} & \cdots & w_0^{\text{in},q} & \cdots & w_m^{\text{in},q} & w_0^{\text{out}} & \cdots & w_q^{\text{out}} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{(m+1)\,q+q+1}$$

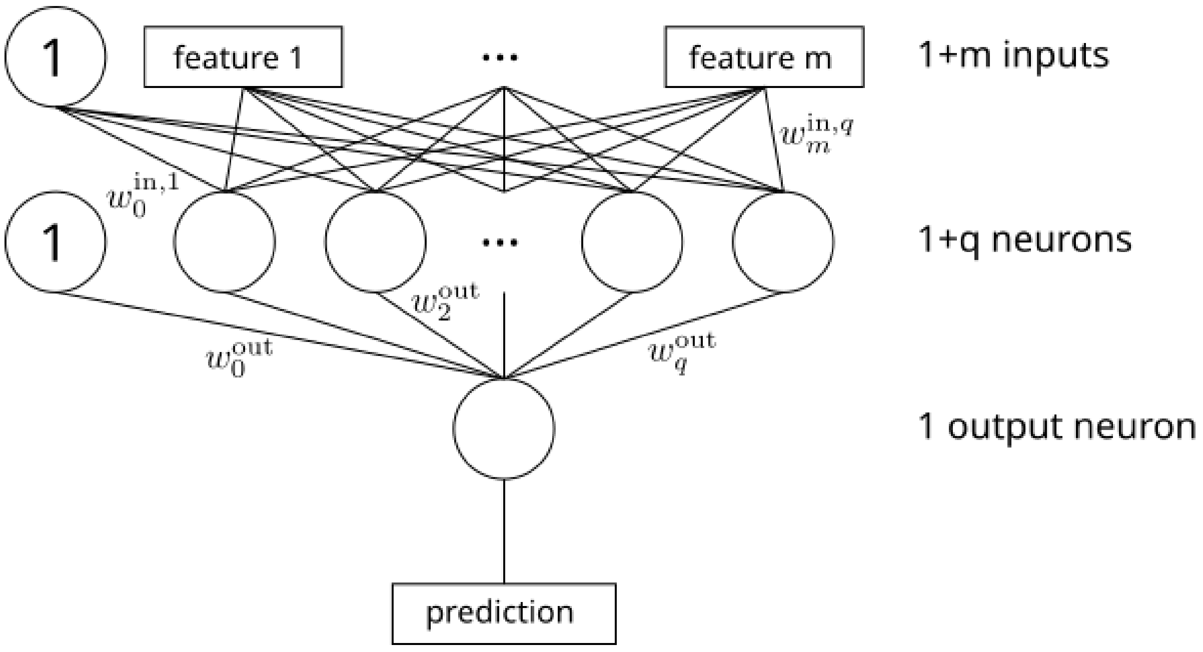


Fig. 26.6: Single layer ANN with mathematical notation for weights.

for the weight vector and

$$f_{\text{approx}}(x) = g_{\text{out}}\left(w_0^{\text{out}} + \sum_{\mu=1}^{q} w_\mu^{\text{out}}\, g_{\text{in}}\left(w_0^{\text{in},\mu} + \sum_{k=1}^{m} w_k^{\text{in},\mu}\, x^{(k)}\right)\right)$$

for the hypothesis. Chain rule yields

$$\frac{\partial}{\partial w_0^{\text{in},\mu}} f_{\text{approx}}(x) = w_\mu^{\text{out}}\, g_{\text{in}}'\left(w_0^{\text{in},\mu} + \sum_{k=1}^{m} w_k^{\text{in},\mu}\, x^{(k)}\right)\, g_{\text{out}}'\left(w_0^{\text{out}} + \sum_{\tilde{\mu}=1}^{q} w_{\tilde{\mu}}^{\text{out}}\, g_{\text{in}}\left(w_0^{\text{in},\tilde{\mu}} + \sum_{k=1}^{m} w_k^{\text{in},\tilde{\mu}}\, x^{(k)}\right)\right),$$

$$\frac{\partial}{\partial w_k^{\text{in},\mu}} f_{\text{approx}}(x) = x^{(k)}\, w_\mu^{\text{out}}\, g_{\text{in}}'\left(w_0^{\text{in},\mu} + \sum_{\tilde{k}=1}^{m} w_{\tilde{k}}^{\text{in},\mu}\, x^{(\tilde{k})}\right)\, g_{\text{out}}'\left(w_0^{\text{out}} + \sum_{\tilde{\mu}=1}^{q} w_{\tilde{\mu}}^{\text{out}}\, g_{\text{in}}\left(w_0^{\text{in},\tilde{\mu}} + \sum_{\tilde{k}=1}^{m} w_{\tilde{k}}^{\text{in},\tilde{\mu}}\, x^{(\tilde{k})}\right)\right)$$

and

$$\frac{\partial}{\partial w_0^{\text{out}}} f_{\text{approx}}(x) = g_{\text{out}}'\left(w_0^{\text{out}} + \sum_{\mu=1}^{q} w_\mu^{\text{out}}\, g_{\text{in}}\left(w_0^{\text{in},\mu} + \sum_{k=1}^{m} w_k^{\text{in},\mu}\, x^{(k)}\right)\right),$$

$$\frac{\partial}{\partial w_\mu^{\text{out}}} f_{\text{approx}}(x) = g_{\text{in}}\left(w_0^{\text{in},\mu} + \sum_{k=1}^{m} w_k^{\text{in},\mu}\, x^{(k)}\right)\, g_{\text{out}}'\left(w_0^{\text{out}} + \sum_{\tilde{\mu}=1}^{q} w_{\tilde{\mu}}^{\text{out}}\, g_{\text{in}}\left(w_0^{\text{in},\tilde{\mu}} + \sum_{k=1}^{m} w_k^{\text{in},\tilde{\mu}}\, x^{(k)}\right)\right).$$

Introducing *activations*

$$a^{\text{in},\mu} := w_0^{\text{in},\mu} + \sum_{k=1}^{m} w_k^{\text{in},\mu}\, x^{(k)}$$

for the regular neurons and

$$a^{\text{out}} := w_0^{\text{out}} + \sum_{\mu=1}^{q} w_\mu^{\text{out}}\, g_{\text{in}}(a^{\text{in},\mu})$$

for the output neuron, we may rewrite those formulas as

$$\frac{\partial}{\partial w_0^{\text{in},\mu}} f_{\text{approx}}(x) = w_\mu^{\text{out}}\, g_{\text{in}}'(a^{\text{in},\mu})\, g_{\text{out}}'(a^{\text{out}}),$$

$$\frac{\partial}{\partial w_k^{\text{in},\mu}} f_{\text{approx}}(x) = x^{(k)}\, w_\mu^{\text{out}}\, g_{\text{in}}'(a^{\text{in},\mu})\, g_{\text{out}}'(a^{\text{out}})$$

and

$$\frac{\partial}{\partial w_0^{\text{out}}} f_{\text{approx}}(x) = g_{\text{out}}'(a^{\text{out}}),$$

$$\frac{\partial}{\partial w_\mu^{\text{out}}} f_{\text{approx}}(x) = g_{\text{in}}(a^{\text{in},\mu})\, g_{\text{out}}'(a^{\text{out}}).$$

To get the gradient $\nabla h(w)$ we have to do the following:

1. Calculate all activations for all feature vectors $x_1, \ldots, x_n$.
2. Calculate predictions for all feature vectors $x_1, \ldots, x_n$ (based on activations from step 1).
3. Built the gradient matrix G (based on activations from step 1).
4. Calculate $\nabla h(w)$ from predictions and gradient matrix (see above).

Multilayer ANNs

We compute the gradient $\nabla f_{\text{approx}}(x)$ for an ANN with $L \geq 2$ layers (excluding the output layer). Above we considered the special case $L = 1$. For general $L \in \mathbb{N}$ notation is slightly more difficult.

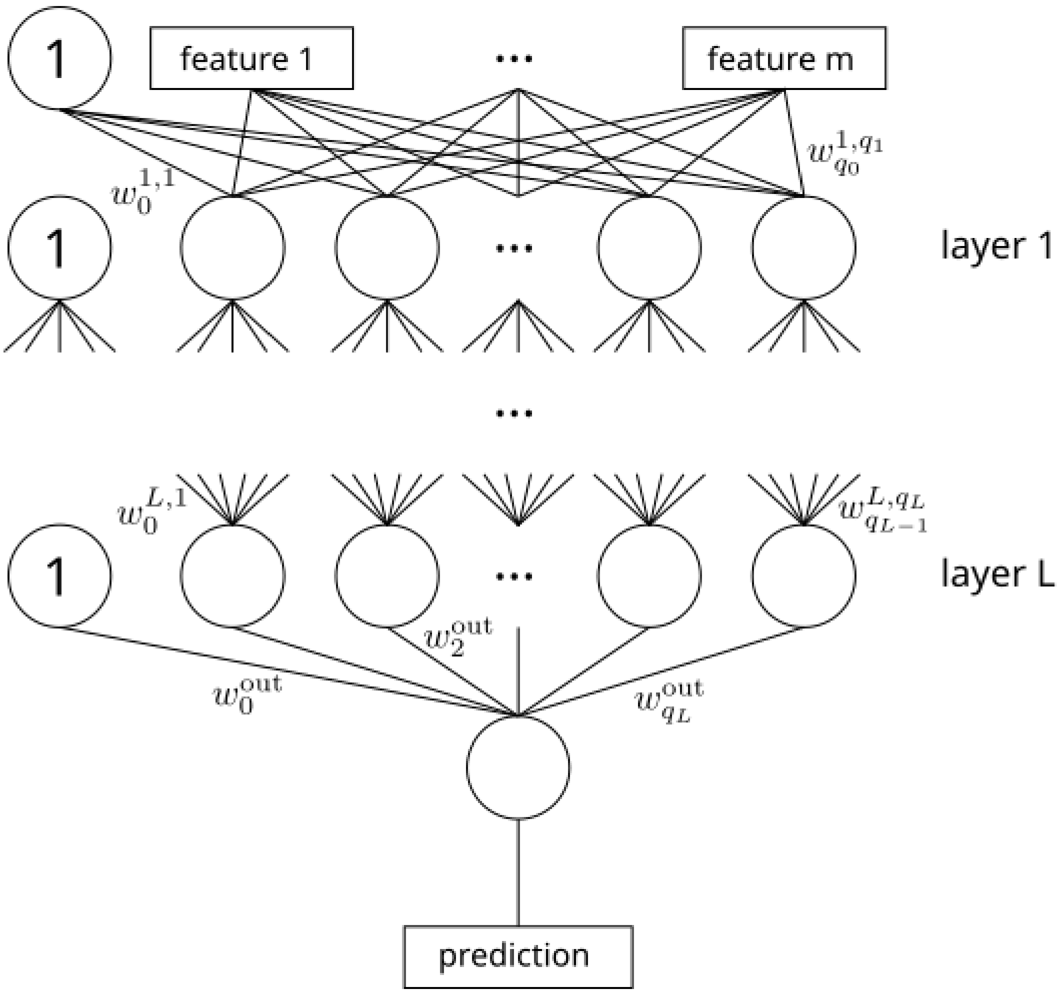


Fig. 26.7: Multilayer ANN with mathematical notation for weights.

We number the layers from 1 (connected to ANN inputs) to L (connected to output neuron). Layer 1 has q_1 neurons, layer 2 has q_2 neurons, and so on. We set $q_0 := m$ to be the number of features (inputs to layer 1). Weights are denoted by $w_\nu^{\lambda,\mu}$ with $\lambda = 1, \ldots, L$ for the layer, $\mu = 1, \ldots, q_\lambda$ for the neuron in the layer, $\nu = 0, 1, \ldots, q_{\lambda-1}$ for the inputs of the neuron (0 for bias neuron plus $q_{\lambda-1}$ regular neurons in previous layer). Weights of the output neuron are $w_0^{\text{out}}, \ldots, w_{q_L}^{\text{out}}$. Activation functions are denoted layerwise by $g_1, \ldots, g_L$ and g_{out}.

We have

$$w=\begin{bmatrix}w_0^{1,1} & \cdots & w_{q_0}^{1,q_1} & \cdots & w_0^{L,1} & \cdots & w_{q_{L-1}}^{L,q_L} & w_0^{\text{out}} & \cdots & w_{q_L}^{\text{out}}\end{bmatrix}^{\mathrm{T}}\in\mathbb{R}^p$$

with

$$p=1+q_L+\sum_{\lambda=1}^{L}q_\lambda\,(1+q_{\lambda-1})$$

for the weight vector.

For each neuron corresponding activations are defined as follows:

$$a^{1,\mu}:=w_0^{1,\mu}+\sum_{\nu=1}^{q_0}w_\nu^{1,\mu}\,x^{(\nu)}\qquad\text{for}\qquad\mu=1,\ldots,q_1,$$

$$a^{\lambda,\mu}:=w_0^{\lambda,\mu}+\sum_{\nu=1}^{q_{\lambda-1}}w_\nu^{\lambda,\mu}\,g_{\lambda-1}(a^{\lambda-1,\nu})\qquad\text{for}\qquad\mu=1,\ldots,q_\lambda\qquad\text{and}\qquad\lambda=2,\ldots,L,$$

$$a^{\text{out}}:=w_0^{\text{out}}+\sum_{\nu=1}^{q_L}w_\nu^{\text{out}}\,g_L(a^{L,\nu}).$$

With this notation the hypothesis is

$$f_{\text{approx}}(x)=g_{\text{out}}(a^{\text{out}}).$$

Partial derivatives with respect to the weights of the output neuron are

$$\frac{\partial}{\partial w_0^{\text{out}}}f_{\text{approx}}(x)=g'_{\text{out}}(a^{\text{out}})\,\frac{\partial}{\partial w_0^{\text{out}}}a^{\text{out}}=g'_{\text{out}}(a^{\text{out}}),$$

$$\frac{\partial}{\partial w_\nu^{\text{out}}}f_{\text{approx}}(x)=g'_{\text{out}}(a^{\text{out}})\,\frac{\partial}{\partial w_\nu^{\text{out}}}a^{\text{out}}=g'_{\text{out}}(a^{\text{out}})\,g_L(a^{L,\nu})\qquad\text{for}\qquad\nu=1,\ldots,q_L.$$

For the other neurons we have

$$\frac{\partial}{\partial w_\nu^{\lambda,\mu}}f_{\text{approx}}(x)=g'_{\text{out}}(a^{\text{out}})\,\frac{\partial}{\partial w_\nu^{\lambda,\mu}}a^{\text{out}}=g'_{\text{out}}(a^{\text{out}})\left(\sum_{\tilde\nu=1}^{q_L}w_{\tilde\nu}^{\text{out}}\,g'_L(a^{L,\tilde\nu})\,\frac{\partial}{\partial w_\nu^{\lambda,\mu}}a^{L,\tilde\nu}\right)\qquad\text{for}\qquad\nu=0,1,\ldots,q_{\lambda-1}.$$

We see that we need the partial derivatives of the activations at layer L. These will depend on the derivatives of the activations at previous layers. Thus, we calculate all partial derivatives of all activations. Consider a weight $w_\nu^{\lambda,\mu}$. This weight has no influence on activations at layers above layer λ and it also has no influence on activations of neurons at layer λ other than neuron μ. So corresponding derivatives are zero. For activations $a^{\lambda,\mu}$ (neuron the weight belongs to) we get an explicit formula. For layers below layer λ we obtain recursive formulas:

$$\begin{aligned}\frac{\partial}{\partial w_\nu^{\lambda,\mu}}a^{\tilde\lambda,\tilde\mu}&=\frac{\partial}{\partial w_\nu^{\lambda,\mu}}\left(w_0^{\tilde\lambda,\tilde\mu}+\sum_{\tilde\nu=1}^{q_{\tilde\lambda-1}}w_{\tilde\nu}^{\tilde\lambda,\tilde\mu}\,g_{\tilde\lambda-1}(a^{\tilde\lambda-1,\tilde\nu})\right)\\&=\begin{cases}0, & \text{if }\tilde\lambda<\lambda,\\ 0, & \text{if }\tilde\lambda=\lambda,\ \tilde\mu\neq\mu,\\ 1, & \text{if }\tilde\lambda=\lambda,\ \tilde\mu=\mu,\ \nu=0,\\ g_{\lambda-1}(a^{\lambda-1,\nu}), & \text{if }\tilde\lambda=\lambda,\ \tilde\mu=\mu,\ \nu>0,\\ \sum\limits_{\tilde\nu=1}^{q_{\tilde\lambda-1}}w_{\tilde\nu}^{\tilde\lambda,\tilde\mu}\,g'_{\tilde\lambda-1}(a^{\tilde\lambda-1,\tilde\nu})\,\frac{\partial}{\partial w_\nu^{\lambda,\mu}}a^{\tilde\lambda-1,\tilde\nu}, & \text{if }\tilde\lambda>\lambda.\end{cases}\end{aligned}$$

To get partial derivatives for activations on layer L with respect to a weight at layer λ we first have to calculate all derivatives for activations at layer λ, then at layer $\lambda+1$ and so on until we reach layer L. Partial derivatives at layer L then yield the desired partial derivative of the ANN's output.

Weight Initialization

For the gradient descent algorithm we need to choose a starting guess. That is we have to choose initial values for all weights. We could set all weights to zero, but then all neurons in a layer would get identical input, leading to identical partial derivatives in the gradient. Thus, each layer would behave like only one single neuron. To brake this symmetry one chooses small random numbers as inital weights. All neurons will have different contributions to the ANN's output and gradient descent will favor some neurons and some neurons will become less influencial.

We have to keep in mind that gradient descent is likely to converge to a local minimum or, even worse, a stationary point close to the starting point. Thus, different sets of initial weights may yield different training results.

Input Standardization

Training data should be standardized to equalize numeric ranges of different features. If there would be a feature with much higher values than the others, then this feature would have much more influence on the initial activations of the neurons. Thus, corresponding weights will have large components in the gradient and will undergo heavy manipulation whereas the other weights change only slightly in each gradient descent step.

Stochastic and Mini-batch Training

In each step of the gradient descent method we need to access the whole data set. Each column of the gradient matrix G corresponds to one sample. For large data sets this approach consumes too many resources. If the data set does not fit into memory, then we cannot use the algorithm.

A much more efficient approach is to use only one sample per step. Then the gradient matrix G has only one column and we save lots of resources. For each gradient descent step we randomly choose a different sample. Thus, all samples will have influence on the training result as before. This approach is known as *stochastic gradient descent*. Another advantage is that if the training data set grows during training, then we can integrate newly arrived data directly into the training process. This is known as *online learning*.

Mini-batch gradient descent is a mixture of both approaches. We split the training data set into a number of disjoint subsets and use a different subset in each gradient descent step. Compared to the full data approach we save resources, but each sample has more influence on the result than for stochastic gradient descent. If the mini-batches are chosen randomly, then this method sometimes is refered to as *stochastic gradient descent*, too.

How to Choose Step Length?

If the step length is small, then there is a good chance to find a minimizer of the objective function, but convergence will be slow. If the step length is large, then we will be relatively close to a minimizer after few descent steps, but the iterates will overshoot the minimizer. Thus, there will be no convergence.

There exist several strategies to choose the step length. In principle, step length for training is a hyperparameter of an ANN. We could try different step lengths and look at the prediction quality of the resulting ANN. Another strategy is to start with large step length and to decrease it during iteration. The idea is to get close to a minimizer within few iterations. Then the minimizer is approached slowly to avoid overshooting. There exist several other proposals for *step length schedules* and there are mathematically justified step length selection rules, too.

Momentum Methods

Steepest descent is only one variant of gradient descent methods. Steepest descent is the simplest and straight forward, but suffers from slow convergence and from getting trapped at saddle points. An imporvement is to add *momentum*. Here, momentum is to be understood in the sense of physics: a ball rolling down a curvy hill does not follow the direction of steepest descent, but, due to its momentum, overshoots curves slightly and will cross small dents without problems. Thus, it will not get trapped by flat local minima or saddle points.

From the mathematical point of view adding momentum means that the step direction not only depends on the current gradient, but also on previous gradients. Usually only the gradient at the previous iterate is used in addition. There are different concrete realizations of the momentum idea out there.

26.2.3 Implementation from Scratch

Later on we will use specialized Python modules for defining and training ANNs. But to gain greater insight into ANNs we implement the code for a multi-layered feedforward net from scratch.

We first define a class for representing ANNs and then we write a script implementing the steepest descent method.

```
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go

rng = np.random.default_rng(0)
```

```
class LayeredFeedforwardANN:

    def __init__(self, inputs, neurons, act_funcs):
        '''
        inputs    ... number of inputs (features) to the ANN
        neurons   ... neurons per layer (list with one item per layer, without␣
↪output layer)
        act_funcs ... activation functions (list with one item per layer,␣
↪including output layer),
                      an activation function has to take two arguments:
                      - 2d NumPy array with activations,
                      - True/False, False for function evaluation, True for␣
↪derivative,
                      has to return NumPy array of same shape as first argument
        '''

        # number of layers (excluding output layer) --> L
        self.layers = len(neurons)

        # number of neurons per layer (layer 0 contains feature values, bias␣
↪neurons excluded)
        # in formulas above: q_0, ..., q_L, 1
        self.neurons = np.array([inputs] + neurons + [1])

        # activation function for each layer (layer 0 has no activation function)
        # in formulas above: -, g_1, ..., g_L, g_out
        self.act_funcs = [None] + act_funcs

        # number of weights per layer
        self.weights = np.empty(self.layers + 2, dtype=np.int32)
        self.weights[0] = 0    # feature values (layer 0)
        self.weights[1:] = self.neurons[1:] * (1 + self.neurons[:-1])

        # weight vector
        self.weight_vector = np.zeros(np.sum(self.weights))
```

(continues on next page)

(continued from previous page)

```
        # activations (one matrix per layer, one row per sample,
        # columns correspond to neurons, column 0 is bias neurons with nan as␣
↪activation
        # to have first regular neuron at column 1)
        self.activations = [None]    # no neurons with activations on layer 0
        for layer in range(1, self.layers + 2):
            self.activations.append(np.zeros((1, 1 + self.neurons[layer])))
            self.activations[layer][0] = np.nan

        # outputs (one matrix per layer, one row per sample,
        # columns correspond to neurons, column 0 is bias neuron)
        self.outputs = []
        for layer in range(0, self.layers + 2):
            self.outputs.append(np.zeros((1, 1 + self.neurons[layer])))

    def update(self, X):
        ''' calculate activations and outputs for all neurons (rows of X are␣
↪feature vectors) '''

        # outputs of layer 0 are feature values
        self.outputs[0] = np.hstack((np.ones((X.shape[0], 1)), X))

        for layer in range(1, self.layers + 2):

            # weight matrix for layer
            W = self.weight_vector[np.sum(self.weights[0:layer]):np.sum(self.
↪weights[0:(layer+1)])]
            W = W.reshape(self.neurons[layer], 1 + self.neurons[layer - 1])

            # activation of neurons in layer
            self.activations[layer] = np.empty((X.shape[0], 1 + self.
↪neurons[layer]))
            self.activations[layer][:, 0] = np.nan    # bias neuron has no␣
↪activation
            self.activations[layer][:, 1:] = np.matmul(W, self.outputs[layer - 1].
↪T).T

            # outputs of neurons in layer
            self.outputs[layer] = np.empty((X.shape[0], 1 + self.neurons[layer]))
            self.outputs[layer][:, 0] = 1
            self.outputs[layer][:, 1:] = self.act_funcs[layer](self.
↪activations[layer][:, 1:])

    def predict(self, X):
        ''' rows of X are feature vectors '''

        self.update(X)

        return self.outputs[-1][:, -1]    # exclude output of bias neuron in␣
↪output layer

    def _flat(self, l, n, i):
        ''' calculate index for self.weight_vector from layer, neuron, input␣
↪indices
        (in formulas above: l --> \lambda, n --> \mu, i --> \nu) '''

        return np.sum(self.weights[0:l]) + (n - 1) * (1 + self.neurons[l - 1]) + i
        # (n - 1) because bias neuron has no weights

    def get_gradient(self, X=None):
```

(continues on next page)

(continued from previous page)

```
        ''' If X is None, then gradient with X from previous call to self.predict
        is returned. Else self.predict(X) is called before calculating the␣
↪gradient.
        Return value is matrix with one column per sample (column is gradient).
        '''

        # update activations and outputs, if necessary
        if not (X is None):
            self.update(X)

        # derivatives of activation functions at current activations
        derivatives = [None]
        for layer in range(1, self.layers + 2):
            derivatives.append(np.empty(self.activations[layer].shape))
            derivatives[layer][:, 0] = np.nan
            derivatives[layer][:, 1:] = self.act_funcs[layer](self.
↪activations[layer][:, 1:],
                                                              derivative=True)

        # partial derivatives of all activations w.r.t. all weights
        # (one 3d-array per layer: dim. 0 is weight, dim. 1 is sample, dim. 2 is␣
↪neuron)
        # Python names versus variables in formulas above:
        # layer --> \tilde{\lambda} ... layer of activation
        # neuron --> \tilde{\mu}    ... neuron of activation
        # l --> \lambda             ... layer neuron weight belongs to
        # n --> \mu                 ... neuron weight belongs to
        # i --> \nu                 ... neuron input comes from
        pd_of_acts = [None]    # no activations on layer 0 (feature values)
        for layer in range(1, self.layers + 2):
            pd_of_acts.append(np.zeros((self.weight_vector.size, *self.
↪activations[layer].shape)))

            for neuron in range(1, self.neurons[layer] + 1):

                # l < layer
                for l in range(1, layer):
                    for n in range(1, self.neurons[l] + 1):
                        for i in range(0, self.neurons[l - 1] + 1):
                            pd_of_acts[layer][self._flat(l, n, i), :, neuron] \
                                = np.sum(self.weight_vector[self._flat(layer,␣
↪neuron, 1):(self._flat(layer, neuron, self.neurons[layer-1])+1)]
                                         * derivatives[layer - 1][:, 1:]
                                         * pd_of_acts[layer - 1][self._flat(l, n,␣
↪i), :, 1:], axis=1)

                # l == layer (only p.d. for n == neuron are nonzero)
                pd_of_acts[layer][self._flat(layer, neuron, 0), :, neuron] = 1
                for i in range(1, self.neurons[layer - 1] + 1):
                    pd_of_acts[layer][self._flat(layer, neuron, i), :, neuron] \
                        = self.outputs[layer - 1][:, i]

        # make gradient
        G = np.empty((self.weight_vector.size, self.outputs[-1].shape[0]))
        for l in range(1, self.layers + 2):
            for n in range(1, self.neurons[l] + 1):
                for i in range(0, self.neurons[l - 1] + 1):
                    G[self._flat(l, n, i), :] = (derivatives[-1][:, 1:]
                                                 * pd_of_acts[-1][self._flat(l, n,␣
↪i), :, 1:]).reshape(-1)
```

(continues on next page)

(continued from previous page)

```
        return G
```

We need some activation functions. Linear activation is used for the output neuron. The others are for the hidden neurons.

```
def linear_activation(a, derivative=False):

    if derivative:
        return np.ones(a.shape)
    else:
        return a


def relu_activation(a, derivative=False):

    if derivative:
        return (a > 0).astype(a.dtype)
    else:
        return np.maximum(np.zeros(a.shape), a)


def tanh_activation(a, derivative=False):

    if derivative:
        return 1 - np.tanh(a) ** 2
    else:
        return np.tanh(a)
```

To test our ANN code we simulate some data with two features. We only use two features for testing, because a function on $\mathbb{R}^2$ can be visualized as 3d plot.

```
def truth(x1, x2):
    return np.sin(np.pi * x1) + x2 ** 2

n_grid = 50    # grid point per axis for plotting

x1 = np.linspace(-1, 1, n_grid)
x2 = np.linspace(-1, 1, n_grid)
[grid_x1, grid_x2] = np.meshgrid(x1, x2)
grid_truth = truth(grid_x1, grid_x2)

fig = go.Figure()
fig.layout.width = 800
fig.layout.height = 600

fig.add_trace(go.Surface(
    x=grid_x1, y=grid_x2, z=grid_truth,
    colorscale=[[0, 'rgb(0,0,255)'], [1, 'rgb(0,0,255)']],
    showscale=False
))

fig.update_scenes(
    xaxis_title_text='feature 1',
    yaxis_title_text='feature 2',
    zaxis_title_text='target'
)
fig.update_layout(title={'text': 'truth', 'x': 0.5, 'xanchor': 'center'})

fig.show()
```

```
<IPython.core.display.HTML object>
```

From the true function (which is unknown in practise) we draw samples for training the ANN. To come closer to real data we add some random noise.

```
n_samples = 100
noise_level = 0.05

X = rng.uniform(-1, 1, (n_samples, 2))
y = truth(X[:, 0], X[:, 1]) + rng.normal(0, noise_level, n_samples)

fig = go.Figure()
fig.layout.width = 800
fig.layout.height = 600

fig.add_trace(go.Surface(
    x=grid_x1, y=grid_x2, z=grid_truth,
    colorscale=[[0, 'rgb(0,0,255)'], [1, 'rgb(0,0,255)']],
    showscale=False
))

fig.add_trace(go.Scatter3d(
    x=X[:, 0], y=X[:, 1], z=y,
    marker={'size': 2, 'color': 'rgba(255,0,0,1)'},
    line={'width': 0, 'color': 'rgba(0,0,0,0)'},
    hoverinfo = 'none'
))

fig.update_scenes(
    xaxis_title_text='feature 1',
    yaxis_title_text='feature 2',
    zaxis_title_text='target'
)
fig.update_layout(title={'text': 'truth and noisy data', 'x': 0.5, 'xanchor':
↪'center'})

fig.show()
```

```
<IPython.core.display.HTML object>
```

We implement gradient descent using all training samples in each gradient step.

```
def gradient_descent_full_batch(net, X, y):

    step_length = 0.1
    max_iter = 10000    # stop after at most so many iterations
    smallest_grad = 1e-10    # abort iteration if gradient size is below this␣
↪value
    show_status_after = 100    # print infos after so many interations

    for i in range(0, max_iter):

        y_pred = net.predict(X)
        G = net.get_gradient()
        grad = 2 / y.size * np.matmul(G, (y_pred - y))

        net.weight_vector = net.weight_vector - step_length * grad

        grad_size = np.max(np.abs(grad))
        if grad_size < smallest_grad:
```

(continues on next page)

(continued from previous page)

```
            print('Stopped by small gradient after {} iterations.'.format(i))
            break

        if i % show_status_after == 0:
            objective = 1 / y.size * np.sum((y - y_pred) ** 2)
            print('iteration {}, grad size {}, objective value {}'.format(i, grad_
↪size, objective))

    else:
        print('Stopped by max_iter with grad_size = {}.'.format(grad_size))
```

Now we define an ANN and start training.

```
net = LayeredFeedforwardANN(2, [5, 4, 3], 3 * [tanh_activation] + [linear_
↪activation])
#net = LayeredFeedforwardANN(2, [5], 1 * [relu_activation] + [linear_activation])

net.weight_vector = 0.1 * rng.normal(size=net.weight_vector.size)
gradient_descent_full_batch(net, X, y)
```

```
iteration 0, grad size 0.5199772941237953, objective value 0.6682282913363523
iteration 100, grad size 0.004188259115251231, objective value 0.
↪6001853239675975
iteration 200, grad size 0.0067462327164474, objective value 0.5994163438040679
iteration 300, grad size 0.01947130728030662, objective value 0.5945976349492498
iteration 400, grad size 0.05696353674497658, objective value 0.
↪27437575660531127
iteration 500, grad size 0.027526464130790823, objective value 0.
↪21478552687825506
iteration 600, grad size 0.009114642672881857, objective value 0.
↪19661760598408443
iteration 700, grad size 0.004846327193297142, objective value 0.193206235258564
iteration 800, grad size 0.004872007975520194, objective value 0.
↪19187781786194577
iteration 900, grad size 0.007417776383947535, objective value 0.
↪19043590964259213
iteration 1000, grad size 0.008885552556249987, objective value 0.
↪18837353720412067
iteration 1100, grad size 0.009568037192828935, objective value 0.
↪18563996550698536
iteration 1200, grad size 0.010012054354481381, objective value 0.
↪18220440430619483
iteration 1300, grad size 0.010418178877651126, objective value 0.
↪17803497660056844
iteration 1400, grad size 0.01075639712936235, objective value 0.
↪17321340025738413
iteration 1500, grad size 0.010947923113323852, objective value 0.
↪16795120443934383
iteration 1600, grad size 0.010934279401701321, objective value 0.
↪16253664459446235
iteration 1700, grad size 0.010698097971309628, objective value 0.
↪1572670641107871
iteration 1800, grad size 0.010262207608537792, objective value 0.
↪1523889646805046
iteration 1900, grad size 0.009676890023421739, objective value 0.
↪14806085026090485
iteration 2000, grad size 0.00900891008384827, objective value 0.14434372346886
iteration 2100, grad size 0.00833508321112536, objective value 0.
↪14120819144319677
iteration 2200, grad size 0.0077223848466977055, objective value 0.
↪13856023629741945
```

(continues on next page)

(continued from previous page)

```
iteration 2300, grad size 0.0072026400917086, objective value 0.
↪13628658605000918
iteration 2400, grad size 0.006774949415083191, objective value 0.
↪1342867512079747
iteration 2500, grad size 0.4381234445624277, objective value 0.
↪15948243205052437
iteration 2600, grad size 0.3040936511232076, objective value 0.1459351832762161
iteration 2700, grad size 0.2935434609394815, objective value 0.
↪14490588894469558
iteration 2800, grad size 0.28580780347654683, objective value 0.
↪14391902900710235
iteration 2900, grad size 0.27746029701818486, objective value 0.
↪14245411381443282
iteration 3000, grad size 0.29328182758454185, objective value 0.
↪1400263806581496
iteration 3100, grad size 0.3103952148237437, objective value 0.1357177431813228
iteration 3200, grad size 0.3089037082357916, objective value 0.
↪12854283894561383
iteration 3300, grad size 0.2906689025696309, objective value 0.
↪12016943141244246
iteration 3400, grad size 0.2703828281297573, objective value 0.
↪11274969322635943
iteration 3500, grad size 0.25129739381599697, objective value 0.106501641601027
iteration 3600, grad size 0.24135077314606818, objective value 0.
↪10141104705971411
iteration 3700, grad size 0.2421217908293113, objective value 0.
↪09778612839471253
iteration 3800, grad size 0.25711890185003916, objective value 0.
↪09568766417502658
iteration 3900, grad size 0.284722246737777, objective value 0.0947076864898635
iteration 4000, grad size 0.3296932782682503, objective value 0.
↪09403499028986839
iteration 4100, grad size 0.3798402640685176, objective value 0.
↪09248875950132941
iteration 4200, grad size 0.4245701332943749, objective value 0.
↪08898185624750109
iteration 4300, grad size 0.45647057405936353, objective value 0.
↪08323629490675982
iteration 4400, grad size 0.4776166938744331, objective value 0.
↪07560503586718813
iteration 4500, grad size 0.4756286013031701, objective value 0.
↪06580074692959394
iteration 4600, grad size 0.4237758050301021, objective value 0.
↪05261848784999673
iteration 4700, grad size 0.32792804776147605, objective value 0.
↪03712512520184369
iteration 4800, grad size 0.23425395310685446, objective value 0.
↪02445699644936167
iteration 4900, grad size 0.18114834289066026, objective value 0.
↪017227636682214578
iteration 5000, grad size 0.16603929903997156, objective value 0.
↪0139897055034033
iteration 5100, grad size 0.1623863909382122, objective value 0.
↪012446982807328397
iteration 5200, grad size 0.1586237216430685, objective value 0.
↪011479020481321035
iteration 5300, grad size 0.15370335463078733, objective value 0.
↪010749620797045056
iteration 5400, grad size 0.1484288710518272, objective value 0.
↪010166610010989872
iteration 5500, grad size 0.14333675997862402, objective value 0.
↪009690543922116402
```

(continues on next page)

(continued from previous page)

```
iteration 5600, grad size 0.13870582731696862, objective value 0.
 ↪009298915283085041
iteration 5700, grad size 0.13458156869585244, objective value 0.
 ↪008972069187505413
iteration 5800, grad size 0.1309219560547808, objective value 0.
 ↪008694452506876942
iteration 5900, grad size 0.12764751905438942, objective value 0.
 ↪008453672061747604
iteration 6000, grad size 0.12467834200270841, objective value 0.
 ↪008240424124501034
iteration 6100, grad size 0.12194550534874606, objective value 0.
 ↪008047859024417934
iteration 6200, grad size 0.11939480580182533, objective value 0.
 ↪007871041114899354
iteration 6300, grad size 0.11698545180964257, objective value 0.
 ↪0077064369028187205
iteration 6400, grad size 0.11468761611433648, objective value 0.
 ↪0075515233404994725
iteration 6500, grad size 0.11247971986898214, objective value 0.
 ↪007404486914100332
iteration 6600, grad size 0.11034623722964698, objective value 0.
 ↪007264013112872696
iteration 6700, grad size 0.10827590974319178, objective value 0.
 ↪007129133998917132
iteration 6800, grad size 0.10626050161504519, objective value 0.
 ↪006999126177998285
iteration 6900, grad size 0.10429386528215825, objective value 0.
 ↪0068734380340564755
iteration 7000, grad size 0.10237132613873698, objective value 0.
 ↪006751641659555333
iteration 7100, grad size 0.10048923678407225, objective value 0.
 ↪006633398560380885
iteration 7200, grad size 0.09864468941543514, objective value 0.
 ↪006518436751616424
iteration 7300, grad size 0.09683531271854097, objective value 0.
 ↪0064065343126327
iteration 7400, grad size 0.09505913998032933, objective value 0.
 ↪0062975080366066164
iteration 7500, grad size 0.0939519377661455, objective value 0.
 ↪006191205164444023
iteration 7600, grad size 0.09285496082438308, objective value 0.
 ↪0060874973754314445
iteration 7700, grad size 0.09176074803899621, objective value 0.
 ↪0059862762779628554
iteration 7800, grad size 0.0906688020220651, objective value 0.
 ↪005887449924631518
iteration 7900, grad size 0.08957875395805977, objective value 0.
 ↪005790940069909926
iteration 8000, grad size 0.08849035491376321, objective value 0.
 ↪005696679930145535
iteration 8100, grad size 0.08740346792360068, objective value 0.
 ↪0056046123270596685
iteration 8200, grad size 0.08631806015976771, objective value 0.
 ↪005514688110645113
iteration 8300, grad size 0.0852341950423185, objective value 0.
 ↪005426864801858907
iteration 8400, grad size 0.0841520242567544, objective value 0.
 ↪005341105413835451
iteration 8500, grad size 0.0830717796635153, objective value 0.
 ↪005257377420520317
iteration 8600, grad size 0.0819937652102004, objective value 0.
 ↪005175651854001437
```

(continues on next page)

(continued from previous page)

```
iteration 8700, grad size 0.08091834891682995, objective value 0.
↪005095902514808226
iteration 8800, grad size 0.07984595502241544, objective value 0.
↪005018105283735866
iteration 8900, grad size 0.07877705637468307, objective value 0.
↪004942237526011821
iteration 9000, grad size 0.0777121671202834, objective value 0.
↪004868277579470936
iteration 9100, grad size 0.07665183574576094, objective value 0.
↪0047962043195361485
iteration 9200, grad size 0.07559663850276738, objective value 0.
↪004725996794204519
iteration 9300, grad size 0.0745471732378859, objective value 0.
↪004657633922577992
iteration 9400, grad size 0.0735040536377324, objective value 0.
↪00459109425080472
iteration 9500, grad size 0.07246790389135159, objective value 0.
↪004526355759560328
iteration 9600, grad size 0.07143935376462186, objective value 0.
↪0044633957174459735
iteration 9700, grad size 0.07041903407678365, objective value 0.
↪00440219057498265
iteration 9800, grad size 0.06940757256578538, objective value 0.
↪004342715894193019
iteration 9900, grad size 0.06840559012591425, objective value 0.
↪0042849463090558135
Stopped by max_iter with grad_size = 0.06732832266043533.
```

Now the net is trained and we may use it for predicting the target variable for arbitrary data.

```
X_grid = np.stack((grid_x1.reshape(-1), grid_x2.reshape(-1)), axis=1)
grid_pred = net.predict(X_grid).reshape(n_grid, n_grid)

fig = go.Figure()
fig.layout.width = 800
fig.layout.height = 600

fig.add_trace(go.Surface(
    x=grid_x1, y=grid_x2, z=grid_truth,
    colorscale=[[0, 'rgb(0,0,255)'], [1, 'rgb(0,0,255)']],
    showscale=False
))

fig.add_trace(go.Scatter3d(
    x=grid_x1.reshape(-1), y=grid_x2.reshape(-1), z=grid_pred.reshape(-1),
    marker={'size': 1, 'color': 'rgba(0,255,0,1)'},
    line={'width': 0, 'color': 'rgba(0,0,0,0)'},
    hoverinfo='none',
    name='predictions'
))

fig.add_trace(go.Scatter3d(
    x=X[:, 0], y=X[:, 1], z=y,
    marker={'size': 2, 'color': 'rgba(255,0,0,1)'},
    line={'width': 0, 'color': 'rgba(0,0,0,0)'},
    hoverinfo = 'none',
    name='training data'
))

fig.update_scenes(
```

(continues on next page)

(continued from previous page)

```
    xaxis_title_text='feature 1',
    yaxis_title_text='feature 2',
    zaxis_title_text='target'
)
fig.update_layout(title={'text': 'truth vs. predictions', 'x': 0.5, 'xanchor':
↪'center'})

fig.show()
```

```
<IPython.core.display.HTML object>
```

26.2.4 ANNs with Scikit-Learn

Scikit-Learn supports layered feedworward ANNs, too. Sometimes, for instance in Scikit-Learn, they are called multi-layer perceptrons or MLPs for short. Corresponding class is `MLPRegressor`[459] in `sklearn.neural_network`. Scikit-Learn's implementation is more efficient than ours above, but not intended for training large scale ANNs. For small and medium sized ANNs it's okay.

Usage is identical to other Scikit-Learn regressors: create an object of type `MLPRegressor`, call `fit`, and then `predict`. Different training algorithms are offered including gradient descent with full batch and mini-batches as well as online learning. For other algorithms have a look at the documentation. Regularization is included, too.

```
# X, y from above

import sklearn.neural_network as neural_network

reg = neural_network.MLPRegressor(hidden_layer_sizes=(5, 4, 3),
                                  activation='relu',
                                  solver='sgd',     # gradient descent
                                  alpha=0,     # no regularization
                                  batch_size=y.size,     # full batch
                                  learning_rate_init=0.1,
                                  max_iter=10000,
                                  momentum=0,     # no momentum
                                  tol=1e-10,     # stop if change of loss is below
                                  verbose=False)     # print status information

reg.fit(X, y)
```

```
MLPRegressor(alpha=0, batch_size=100, hidden_layer_sizes=(5, 4, 3),
             learning_rate_init=0.1, max_iter=10000, momentum=0, solver='sgd',
             tol=1e-10)
```

Now the net is trained and we may use it for prediction.

```
# grid_x1, grid_x2, grid_truth from above

X_grid = np.stack((grid_x1.reshape(-1), grid_x2.reshape(-1)), axis=1)
grid_pred = reg.predict(X_grid).reshape(n_grid, n_grid)

fig = go.Figure()
fig.layout.width = 800
fig.layout.height = 600

fig.add_trace(go.Surface(
```

(continues on next page)

459 https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

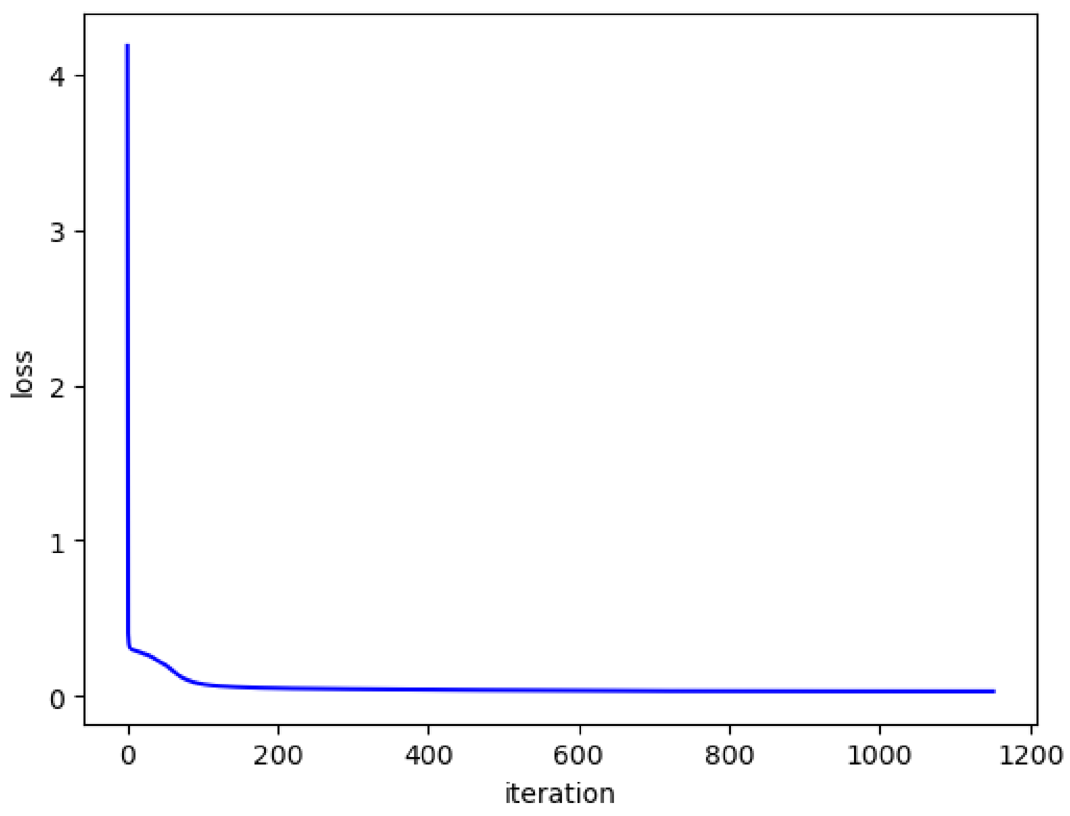
(continued from previous page)

```
    x=grid_x1, y=grid_x2, z=grid_truth,
    colorscale=[[0, 'rgb(0,0,255)'], [1, 'rgb(0,0,255)']],
    showscale=False
))

fig.add_trace(go.Scatter3d(
    x=grid_x1.reshape(-1), y=grid_x2.reshape(-1), z=grid_pred.reshape(-1),
    marker={'size': 1, 'color': 'rgba(0,255,0,1)'},
    line={'width': 0, 'color': 'rgba(0,0,0,0)'},
    hoverinfo='none',
    name='predictions'
))

fig.add_trace(go.Scatter3d(
    x=X[:, 0], y=X[:, 1], z=y,
    marker={'size': 2, 'color': 'rgba(255,0,0,1)'},
    line={'width': 0, 'color': 'rgba(0,0,0,0)'},
    hoverinfo = 'none',
    name='training data'
))

fig.update_scenes(
    xaxis_title_text='feature 1',
    yaxis_title_text='feature 2',
    zaxis_title_text='target'
)
fig.update_layout(title={'text': 'truth vs. predictions', 'x': 0.5, 'xanchor':
 ↪'center'})

fig.show()
```

```
<IPython.core.display.HTML object>
```

Scikit-Learn provides access to several parameters of the ANN and of the training phase. For example, we may inspect the loss curve, that is, the loss (value of objective function) for each iteration. See documentation for more.

```
fig, ax = plt.subplots()
ax.plot(reg.loss_curve_, '-b')
ax.set_xlabel('iteration')
ax.set_ylabel('loss')
plt.show()
```



26.3 ANNs with Keras

To represent complex hypotheses with ANNs we need thousands or even millions of artificial neurons. ANNs with few large layers turned out be less effective than ANNs with many smaller layers. The latter are referred to as *deep networks* and their usage is known as *deep learning*.

Training ANNs requires lots of computation time for large data sets and large networks. Thus, we need efficient implementations of learning procedures and powerful hardware. Scikit-Learn aims at educational projects and offers a wide scale of machine learning methods. Implementation is less optimized for execution speed than for providing insight into the algorithms and providing access to all the parameters and intermediate results. Further, Scikit-Learn does not use all features of modern hardware.

Libraries for high-performance machine learning have to be more specialized on specific tasks to allow for optimizing efficiency. Keras[460] is an open source library for implementing and training ANNs. Like Scikit-Learn it provides preprocessing routines as well as postprocessing (optimizing hyperparameters). But Keras utilizes full computation power of modern computers for training ANNs, leading to much shorter training times.

Modern computers have multi-core CPUs. So they can process several programs in parallel. In addition, almost all computers have a powerful GPU (graphics processing unit). It's like a second CPU specialized at doing floating point computations for rendering 3d graphics. GPUs are much more suited for training ANNs than CPUs, because they are designed to work with many large matrices of floating point numbers in parallel. Nowadays GPUs can be accessed by software developers relatively easily. Thus, we may run programs on the GPU instead of the CPU.

Keras seamlessly integrates GPU power for ANN training into Python. We do not have to care about the details. Keras piggybacks on an open source library called TensorFlow[461] developed by Google. Keras does much of the work for us, but from time to time TensorFlow will show up, too. Keras started independently from TensorFlow, then integrated support for TensorFlow, and now is distributed as a module in the TensorFlow Python package.

[460] https://keras.io
[461] https://www.tensorflow.org/

```
import tensorflow.keras as keras
```

```
2023-04-25 10:26:31.847362: I tensorflow/core/platform/cpu_feature_guard.
 ↪cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network␣
 ↪Library (oneDNN) to use the following CPU instructions in performance-
 ↪critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate␣
 ↪compiler flags.
```

26.3.1 An ANN for Handwritten Digit Recognition

To demonstrate usage of Keras we implement and train a layered feedforward ANN to classify handwritten digits from the QMNIST data set. Inputs to the ANN are images of size 28x28. Thus, the feature space has dimension 784. Outputs are 10 real numbers in $[0, 1]$. Each number represents the probability that the image shows the corresponding digit.

We could also use an ANN with only one output and require that this output is the digit, that is, it has range $[0, 9]$. But how to interpret an output of 3.5? It suggests that the ANN cannot decide between 3 and 4. Or it might waffle on 2 and 5. Using only one output we would introduce artificial order and, thus, wrong similarity assumptions. From the view of similarity of shape (and only that matters in digit recognition), 3 and 8 are more close to each other than 7 and 8 are. Using one output per figure we avoid artificial assumptions and get more precise information on possible missclassifications. Images with high outputs for both 1 and 7 could be marked for subsequent review by a human, for example

Loading Data

For loading QMNIST data we may reuse code from *Load QMNIST* (page 933) project. We have to load training data and test data, both consisting of 60000 images and corresponding labels.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

import qmnist
```

```
train_images, train_labels, _, _ = qmnist.load('../../../../datasets/qmnist/',␣
 ↪subset='train')
test_images, test_labels, _, _ = qmnist.load('../../../../datasets/qmnist/',␣
 ↪subset='test')
```

```
train_images.shape, test_images.shape, train_labels.shape, test_labels.shape
```

```
((60000, 28, 28), (60000, 28, 28), (60000,), (60000,))
```

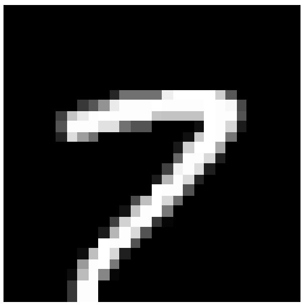
```
train_images[0, :, :].min(), train_images[0, :, :].max()
```

```
(0.0, 1.0)
```

For visualization of data we use a gray scale with black at smallest value and white at highest value.

```
def show_image(img):
    fig, ax = plt.subplots(figsize=(2, 2))
    ax.imshow(img, vmin=0, vmax=1, cmap='gray')
    ax.axis('off')
    plt.show()
```

```
idx = 123

show_image(train_images[idx, :, :])
print('label:', train_labels[idx])
```



```
label: 7
```

Preprocessing

Input to an ANN should be standardized or normalized. QMNIST images have range $[0, 1]$. That's okay.

Optionally, we may center the images with respect to a figure's bounding box. Without this step the center of mass is identical to the image center (we may reuse code from *Image Processing with NumPy* (page 879)). As a by-product of centering bounding boxes each image will have 4 unused pixels at the boundary. Thus, we may crop images to 20x20 pixels without loss of information (resulting in 400 instead of 784 features).

```
def auto_crop(img):

    # binarize image
    mask = img > 0

    # whole image black?
    if not mask.any():
        return np.array([])

    # get top and bottom index of bounding box
    row_mask = mask.any(axis=1)
    top = np.argmax(row_mask)
    bottom = row_mask.size - np.argmax(row_mask[::-1])     # bottom index + 1

    # get left and right index of bounding box
    col_mask = mask[top:bottom, :].any(axis=0)     # [top:bottom, :] for␣
↪efficiency only
    left = np.argmax(col_mask)
    right = col_mask.size - np.argmax(col_mask[::-1])     # right index + 1

    # crop
    return img[top:bottom, left:right].copy()
```

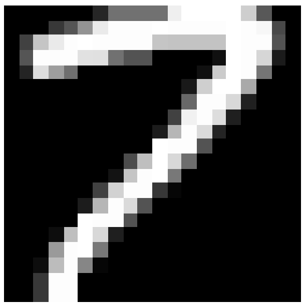
(continues on next page)

(continued from previous page)

```
def center(img, n):

    # check image size
    if np.max(img.shape) > n:
        print('n too small! Cropping image.')
        img = img[0:np.minimum(n, img.shape[0]), 0:np.minimum(n, img.shape[1])]

    # calculate margin width
    top_margin = (n - img.shape[0]) // 2
    left_margin = (n - img.shape[1]) // 2

    # create image
    img_new = np.zeros((n, n), dtype=img.dtype)
    img_new[top_margin:(top_margin + img.shape[0]),
            left_margin:(left_margin + img.shape[1])] = img

    return img_new
```

```
train_images = qmnist.preprocess(train_images, [auto_crop, lambda img: center(img,
 ↪ 20)])
test_images = qmnist.preprocess(test_images, [auto_crop, lambda img: center(img,␣
↪20)])
```

```
idx = 123

show_image(train_images[idx, :, :])
print('label:', train_labels[idx])
```



```
label: 7
```

Training labels have to be one-hot encoded. This can be done manually with NumPy or automatically with Pandas or Scikit-Learn. Also Keras provides a function for one-hot encoding: `to_categorical`[462].

```
train_labels = keras.utils.to_categorical(train_labels)
test_labels = keras.utils.to_categorical(test_labels)

train_labels.shape, test_labels.shape
```

```
((60000, 10), (60000, 10))
```

462 https://keras.io/api/utils/python_utils/#tocategorical-function

Defining the ANN

Keras has a `Model`[463] class representing a directed graph of layers of neurons. At the moment we content ourselves with simple network structures, that is, we have a sequence of layers. For such simple structures Keras has the `Sequential`[464] class. That class represents a stack of layers of neurons. It's a subclass of `Model`.

A layer is represented by one of several layer classes in Keras. For a fully connected feedforward ANN we need an `Input`[465] layer and several `Dense`[466] layers. Layers can be added one by one with `Sequential.add`[467].

`Input` layers accept multi-dimensional inputs. Thus, we do not have to convert the 20x20 images to vectors with 400 components. But `Dense` layers want to have one-dimensional input. Thus, we use a Flatten[468] layer. Like the `Input` layer that's not a layer of neurons. Layers in Keras have to be understood as transformations taking some input and yielding some output. For `Dense` layers we need to specify the number of neurons and the activation function to use. There are several pre-defined activation functions[469] in Keras.

Layers may have a name, which will help accessing single layers for analysis of a trained model. If we do not specify layer names, Keras generates them automatically.

```
model = keras.Sequential()

model.add(keras.Input(shape=(20, 20)))
model.add(keras.layers.Flatten())

model.add(keras.layers.Dense(10, activation='relu', name='dense1'))
model.add(keras.layers.Dense(10, activation='relu', name='dense2'))
```

```
2023-04-25 10:26:40.248000: E tensorflow/compiler/xla/stream_executor/cuda/cuda_
 ↪driver.cc:267] failed call to cuInit: CUDA_ERROR_UNKNOWN: unknown error
2023-04-25 10:26:40.248032: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
 ↪diagnostics.cc:169] retrieving CUDA diagnostic information for host: WHZ-46349
2023-04-25 10:26:40.248039: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
 ↪diagnostics.cc:176] hostname: WHZ-46349
2023-04-25 10:26:40.248184: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
 ↪diagnostics.cc:200] libcuda reported version is: 470.161.3
2023-04-25 10:26:40.248204: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
 ↪diagnostics.cc:204] kernel reported version is: 470.161.3
2023-04-25 10:26:40.248209: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
 ↪diagnostics.cc:310] kernel version seems to match DSO: 470.161.3
2023-04-25 10:26:40.248864: I tensorflow/core/platform/cpu_feature_guard.
 ↪cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network␣
 ↪Library (oneDNN) to use the following CPU instructions in performance-
 ↪critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate␣
 ↪compiler flags.
```

The output layer is a `Dense` layer with 10 neurons. Because all outputs shall have range [0,1] we use the sigmoid function.

```
model.add(keras.layers.Dense(10, activation='sigmoid', name='out'))
```

Output shape can be accessed via corresponding member variable:

```
model.output_shape
```

[463] https://keras.io/api/models/model/
[464] https://keras.io/api/models/sequential/
[465] https://keras.io/api/layers/core_layers/input/
[466] https://keras.io/api/layers/core_layers/dense/
[467] https://keras.io/api/models/sequential/#add-method
[468] https://keras.io/api/layers/reshaping_layers/flatten/
[469] https://keras.io/api/layers/activations/

```
(None, 10)
```

Almost always the first dimension of input or output shapes is the batch size for mini-batch training in Keras. `None` is used, if there is no fixed batch size.

More detailed information about the constructed ANN is provided by `Sequential.summary`:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 400)               0         
                                                                 
 dense1 (Dense)              (None, 10)                4010      
                                                                 
 dense2 (Dense)              (None, 10)                110       
                                                                 
 out (Dense)                 (None, 10)                110       
                                                                 
=================================================================
Total params: 4,230
Trainable params: 4,230
Non-trainable params: 0
_________________________________________________________________
```

Training the ANN

Parameters for training are set with `Model.compile`[470]. Here we may define an optimization routine. Next to gradient descent there are several other optimizers available[471]. The optimizer can be passed by name (as string) or we may create a Python object of the respective optimizer class. The latter allows for custom parameter choice.

Next to the optimizer we have to provide a loss function[472]. Again we may pass a string or an object. Because we have a classification problem we may use log loss.

If we want to validate the model during training, we may pass validation metrics[473] to `compile`. Then output during training includes updated values for the validation metrics on training and validation data. For classification we may use accuracy score. Again, metrics can be passsed by name or as an object. Since we might wish to compute several different metrics, the `metrics` argument expects a list.

```
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=[
↪'categorical_accuracy'])
```

Now the model is ready for training. In Keras training is done by calling `Model.fit`[474]. We may specify the batch size for mini-batch training and the number of *epochs*. An epoch is a sequence of iterations required to cycle through the training data once. Small batch sizes require more iterations per epoch, large batch sizes require fewer iterations. In full-batch training epochs and iterations are equivalent. We may also specify validation data (directly or as fraction of the training data) to get validation metrics after each epoch. Thus, we can see wether the model overfits the data during training and abort training if necessary. The return value of `fit` will be discussed below.

```
history = model.fit(train_images, train_labels, batch_size=100, epochs=5,␣
↪validation_split=0.2)
```

[470] https://keras.io/api/models/model_training_apis/#compile-method
[471] https://keras.io/api/optimizers/#available-optimizers
[472] https://keras.io/api/losses/#available-losses
[473] https://keras.io/api/metrics/
[474] https://keras.io/api/models/model_training_apis/#fit-method

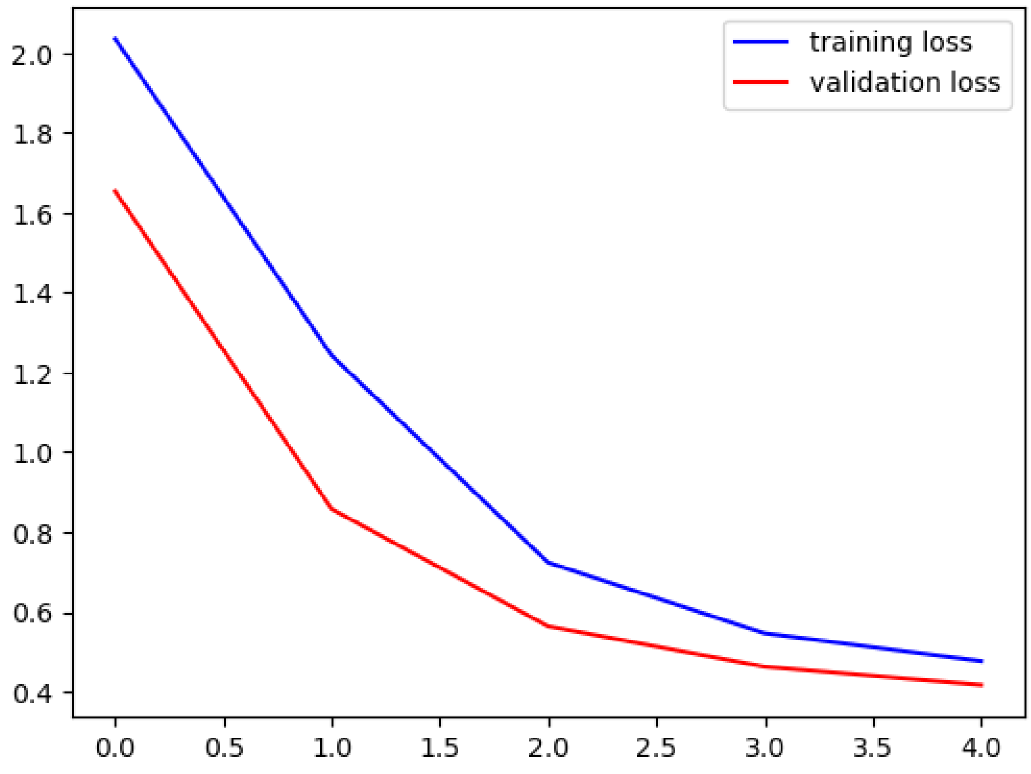
```
Epoch 1/5
480/480 [==============================] - 1s 2ms/step - loss: 2.0348 -␣
 ↪categorical_accuracy: 0.3197 - val_loss: 1.6531 - val_categorical_accuracy: 0.
 ↪4880
Epoch 2/5
480/480 [==============================] - 1s 1ms/step - loss: 1.2421 -␣
 ↪categorical_accuracy: 0.6128 - val_loss: 0.8575 - val_categorical_accuracy: 0.
 ↪7488
Epoch 3/5
480/480 [==============================] - 1s 1ms/step - loss: 0.7235 -␣
 ↪categorical_accuracy: 0.7898 - val_loss: 0.5639 - val_categorical_accuracy: 0.
 ↪8444
Epoch 4/5
480/480 [==============================] - 1s 1ms/step - loss: 0.5463 -␣
 ↪categorical_accuracy: 0.8456 - val_loss: 0.4627 - val_categorical_accuracy: 0.
 ↪8720
Epoch 5/5
480/480 [==============================] - 1s 1ms/step - loss: 0.4769 -␣
 ↪categorical_accuracy: 0.8661 - val_loss: 0.4172 - val_categorical_accuracy: 0.
 ↪8847
```

Hint: Note that validation accuracy displayed by Keras sometimes is higher than training accuracy. The reason is that train accuracy is the mean over all iterations of an epoche, but validation accuracy is calulated only at the end of an epoche. Thus, training accuracy includes poorer accuracy values from beginning of an epoche.

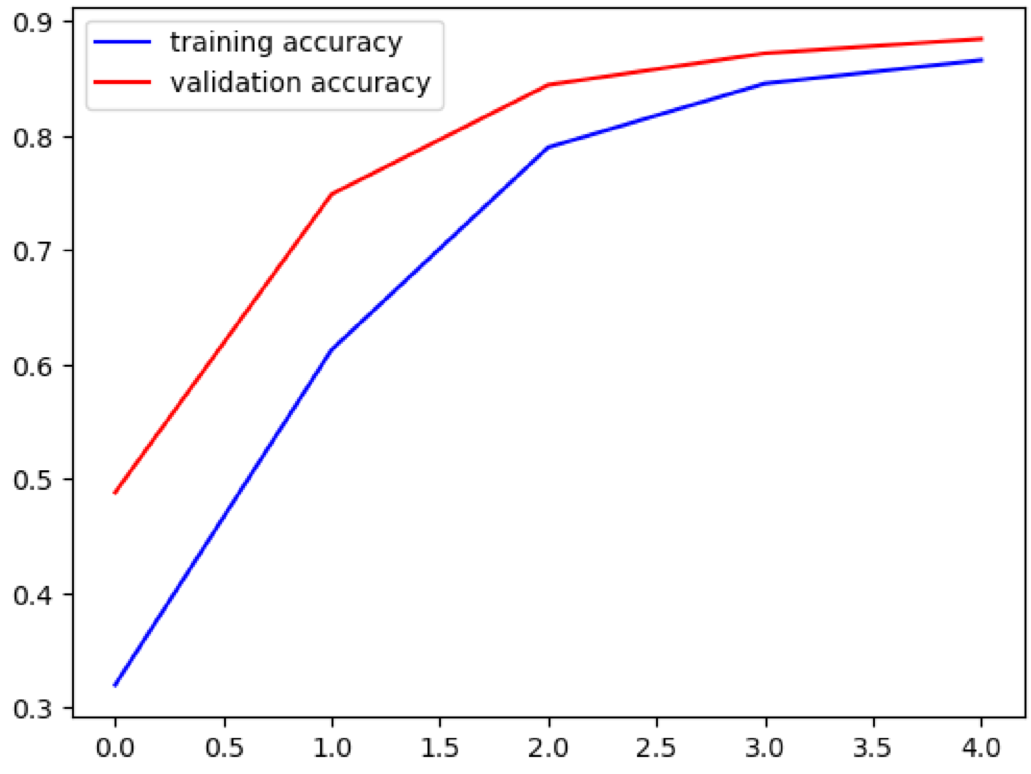
Incremental Training

The `Model.fit` method returns a `History` object containing information about loss and metrics for each training epoch. The object has a dict member `history` containing losses and metrics. Loss keys are `loss` and `val_loss` for training and validation, respectively. Metrics keys depend an the chosen metrics.

```
fig, ax = plt.subplots()
ax.plot(history.history['loss'], '-b', label='training loss')
ax.plot(history.history['val_loss'], '-r', label='validation loss')
ax.legend()
plt.show()
```



```
fig, ax = plt.subplots()
ax.plot(history.history['categorical_accuracy'], '-b', label='training accuracy')
ax.plot(history.history['val_categorical_accuracy'], '-r', label='validation␣
↪accuracy')
ax.legend()
plt.show()
```



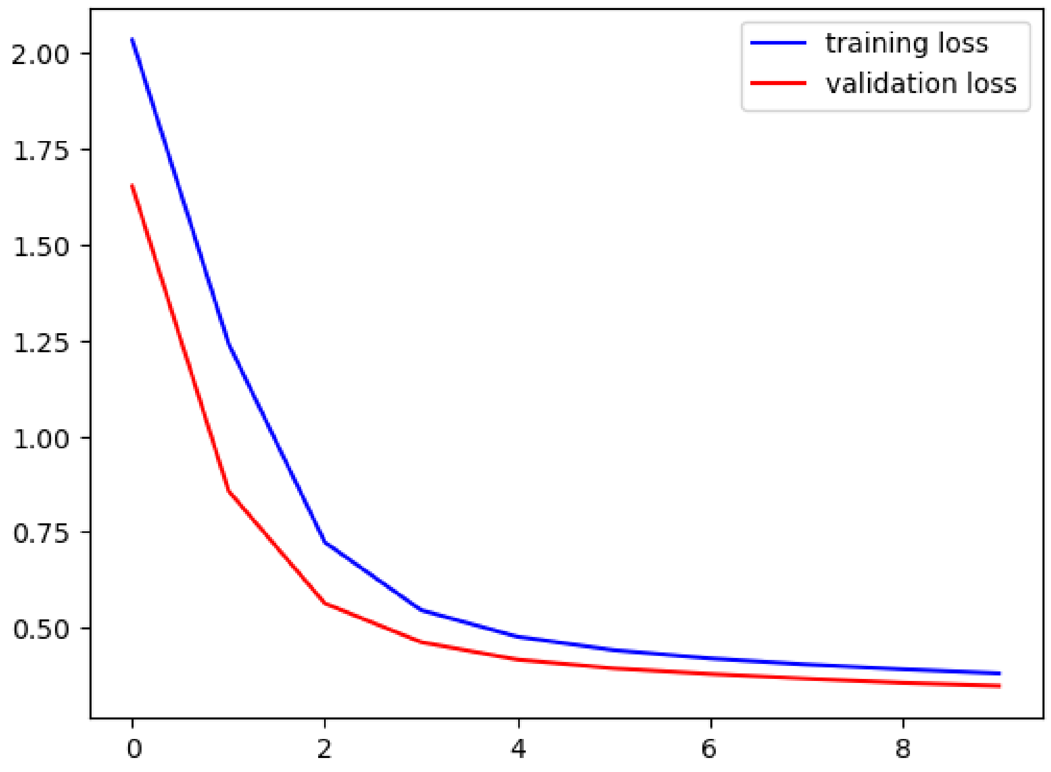
We see that further training could improve the model. Thus, we call `fit` again. Training proceeds from where it has been stopped. We may execute corresponding code cell as often as we like to continue training. To keep the losses and metrics we append them to lists.

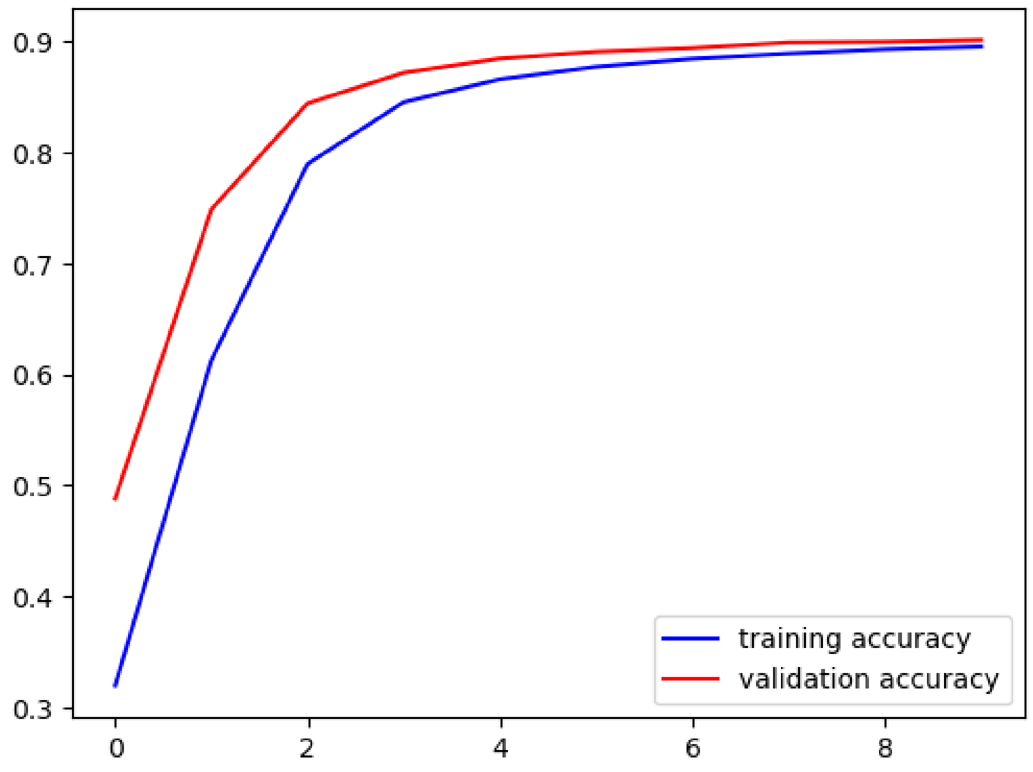
```
loss = history.history['loss']
val_loss = history.history['val_loss']
acc = history.history['categorical_accuracy']
val_acc = history.history['val_categorical_accuracy']
```

```
history = model.fit(train_images, train_labels, batch_size=100, epochs=5,␣
 ↪validation_split=0.2)

loss.extend(history.history['loss'])
val_loss.extend(history.history['val_loss'])
acc.extend(history.history['categorical_accuracy'])
val_acc.extend(history.history['val_categorical_accuracy'])
```

```
Epoch 1/5
480/480 [==============================] - 1s 1ms/step - loss: 0.4419 -␣
 ↪categorical_accuracy: 0.8773 - val_loss: 0.3943 - val_categorical_accuracy: 0.
 ↪8908
Epoch 2/5
480/480 [==============================] - 1s 1ms/step - loss: 0.4203 -␣
 ↪categorical_accuracy: 0.8845 - val_loss: 0.3797 - val_categorical_accuracy: 0.
 ↪8942
Epoch 3/5
480/480 [==============================] - 1s 1ms/step - loss: 0.4045 -␣
 ↪categorical_accuracy: 0.8891 - val_loss: 0.3674 - val_categorical_accuracy: 0.
 ↪8992
Epoch 4/5
480/480 [==============================] - 1s 1ms/step - loss: 0.3920 -␣
 ↪categorical_accuracy: 0.8930 - val_loss: 0.3564 - val_categorical_accuracy: 0.
 ↪8999
Epoch 5/5
480/480 [==============================] - 1s 1ms/step - loss: 0.3813 -␣
 ↪categorical_accuracy: 0.8954 - val_loss: 0.3486 - val_categorical_accuracy: 0.
 ↪9016
```

```
fig, ax = plt.subplots()
ax.plot(loss, '-b', label='training loss')
ax.plot(val_loss, '-r', label='validation loss')
ax.legend()
plt.show()

fig, ax = plt.subplots()
ax.plot(acc, '-b', label='training accuracy')
ax.plot(val_acc, '-r', label='validation accuracy')
ax.legend()
plt.show()
```


training loss
validation loss
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0
2
4
6
8


training accuracy
validation accuracy
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
2
4
6
8

Evaluation and Prediction

To get loss and metrics on the test set call `Model.evaluate`[475].

```
test_loss, test_metric = model.evaluate(test_images, test_labels)

test_loss, test_metric
```

```
1875/1875 [==============================] - 2s 968us/step - loss: 0.3572 -␣
↪categorical_accuracy: 0.9006
```

```
(0.3571886718273163, 0.9005666375160217)
```

For predictions call `Model.predict`[476].

```
test_pred = model.predict(test_images)
```

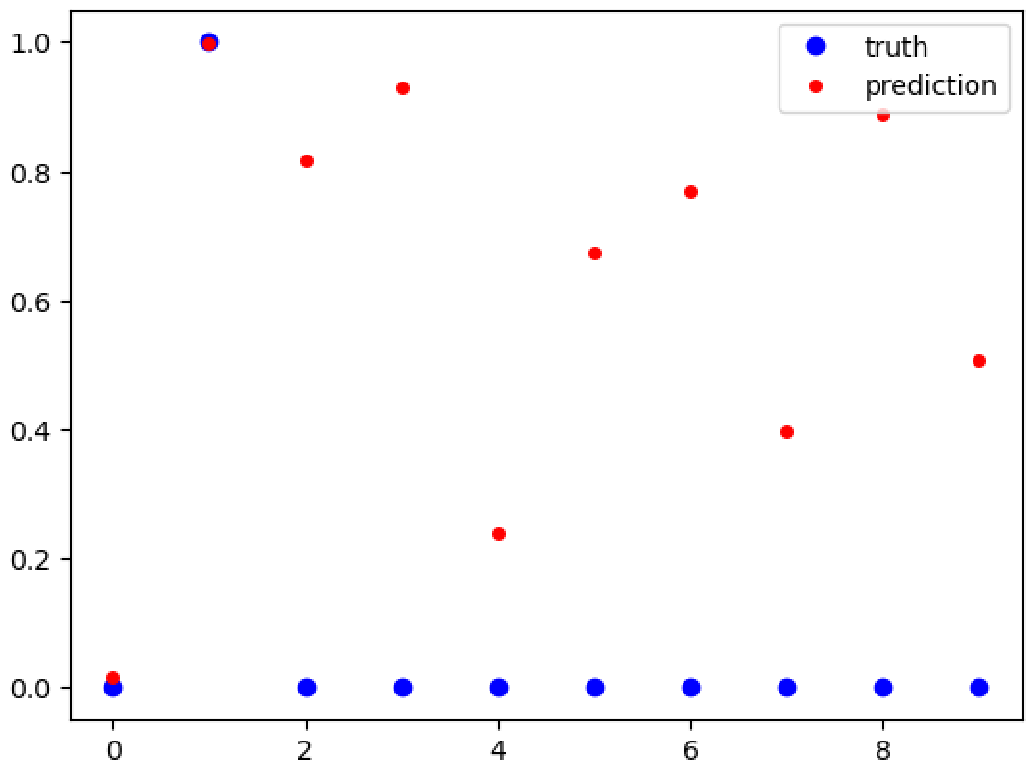
```
1875/1875 [==============================] - 1s 710us/step
```

Predictions are vectors of values from $[0, 1]$. A one indicates that the image shows the corresponding digit, a zero indicates that the digits is not shown in the image.

```
idx = 2

print('truth:      ', test_labels[idx, :])
print('prediction:', test_pred[idx, :])

fig, ax = plt.subplots()
ax.plot(test_labels[idx, :], 'ob', label='truth')
ax.plot(test_pred[idx, :], 'or', markersize=4, label='prediction')
ax.legend()
plt.show()
```

```
truth:      [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
prediction: [0.01495721 0.9989413  0.81571907 0.9306395  0.24000315 0.67344236
 0.7693644  0.39637503 0.88925314 0.50644916]
```

[475] https://keras.io/api/models/model_training_apis/#evaluate-method
[476] https://keras.io/api/models/model_training_apis/#predict-method

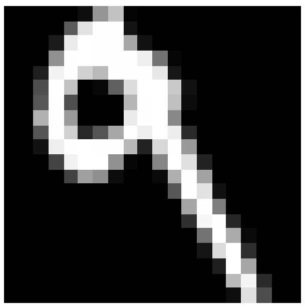


To get more insight into the prediction accuracy we reverse one-hot encoding.

```
true_digits = test_labels.argmax(axis=1)
pred_digits = test_pred.argmax(axis=1)

# indices with wrong predictions
wrong_predictions = np.arange(0, true_digits.size)[true_digits != pred_digits]
print(wrong_predictions.size)
print(wrong_predictions)
```

```
5987
[    8    18    33 ... 59974 59998 59999]
```

```
idx = 7

show_image(test_images[idx])

print('truth: {}, prediction: {}'.format(true_digits[idx], pred_digits[idx]))
```



```
truth: 9, prediction: 9
```

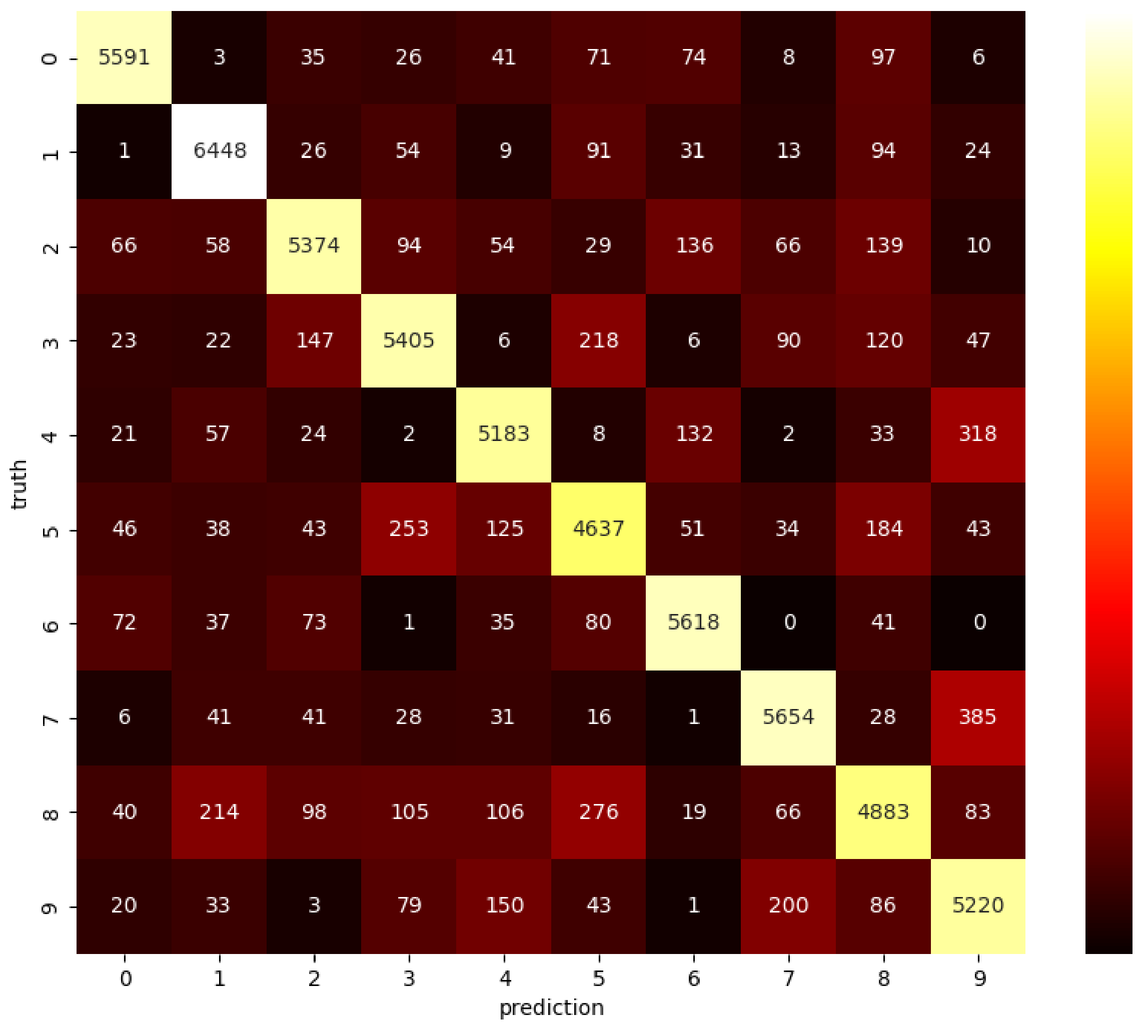
A *confusion matrix* depicts which digits are hard to separate for the ANN. The matrix is 10x10. The entry at row i and column j gives the number of images which show digit i (truth), but corresponding prediction is j. Several Python modules provide functions for building a confusion matrix. Next to Scikit-Learn we may use Pandas for getting the matrix and Seaborn for plotting (`pd.crosstab`[477], `sns.heatmap`[478]).

```
conf_matrix = pd.crosstab(pd.Series(true_digits, name='truth'),
                          pd.Series(pred_digits, name='prediction'))
print(conf_matrix)
```

```
prediction     0     1     2     3     4     5     6     7     8     9
truth
0           5591     3    35    26    41    71    74     8    97     6
1              1  6448    26    54     9    91    31    13    94    24
2             66    58  5374    94    54    29   136    66   139    10
3             23    22   147  5405     6   218     6    90   120    47
4             21    57    24     2  5183     8   132     2    33   318
5             46    38    43   253   125  4637    51    34   184    43
6             72    37    73     1    35    80  5618     0    41     0
7              6    41    41    28    31    16     1  5654    28   385
8             40   214    98   105   106   276    19    66  4883    83
9             20    33     3    79   150    43     1   200    86  5220
```

```
# scale values nonlinearly to get different colors at small values
scaled_conf_matrix = conf_matrix.apply(lambda x: x ** (1/2))

fig = plt.figure(figsize=(10, 8))
sns.heatmap(scaled_conf_matrix,
            annot=conf_matrix,    # use original matrix for labels
            fmt='d',    # format numbers as integers
            cmap='hot',    # color map
            cbar_kws={'ticks': []})    # no ticks for colorbar (would have scaled␣
 ↪labels)
plt.show()
```

[477] https://pandas.pydata.org/docs/reference/api/pandas.crosstab.html
[478] https://seaborn.pydata.org/generated/seaborn.heatmap.html



26.3.2 Hyperparameter Optimization

Keras itself offers no hyperparameter optimization routines. But there is the `keras-tuner`[479] module (import as `keras_tuner`).

```
import keras_tuner
```

We first have to create a function which builds the model and returns a `Model` instance. This function takes a `HyperParameters`[480] object as argument containing information about hyperparameters to optimize. The build function calls methods of the `HyperParameters` object to get values from the current set of hyperparameters.

```
def build_model(hp):

    model = keras.Sequential()
    model.add(keras.Input(shape=(20, 20)))
    model.add(keras.layers.Flatten())

    layers = hp.Int('layers', 1, 3)
    neurons_per_layer = hp.Int('neurons_per_layer', 10, 40, step=10)

    for l in range(0, layers):
```

(continues on next page)

479 https://keras-team.github.io/keras-tuner/

480 https://keras-team.github.io/keras-tuner/documentation/hyperparameters/

(continued from previous page)

```
        model.add(keras.layers.Dense(neurons_per_layer, activation='relu'))

    model.add(keras.layers.Dense(10, activation='sigmoid'))

    model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=[
↪'categorical_accuracy'])

    return model
```

Now we create a `Tuner`[481] object and call its `search`[482] method. Several subclasses are available. `RandomSearch`[483] randomly selects sets of hyperparameters and trains the model for each set of hyperparameters. The constructor takes the model building function, an objective (string with objective name of one of the model's metrics), and the maximum number of parameter sets to test. The `search` function takes training and validation data in full analogy to `fit`.

```
tuner = keras_tuner.tuners.randomsearch.RandomSearch(build_model, 'val_
↪categorical_accuracy', 10)
tuner.search(train_images, train_labels, validation_split=0.2, epochs=10)
```

```
INFO:tensorflow:Reloading Tuner from ./untitled_project/tuner0.json
INFO:tensorflow:Oracle triggered exit
```

Here is a summary of all models considered during hyperparameter optimization:

```
tuner.results_summary()
```

```
Results summary
Results in ./untitled_project
Showing 10 best trials
Objective(name="val_categorical_accuracy", direction="max")

Trial 09 summary
Hyperparameters:
layers: 3
neurons_per_layer: 30
Score: 0.9556666612625122

Trial 02 summary
Hyperparameters:
layers: 2
neurons_per_layer: 30
Score: 0.9505000114440918

Trial 08 summary
Hyperparameters:
layers: 2
neurons_per_layer: 40
Score: 0.9503333568572998

Trial 05 summary
Hyperparameters:
layers: 2
neurons_per_layer: 20
Score: 0.9451666474342346
```

(continues on next page)

481 https://keras-team.github.io/keras-tuner/documentation/tuners/
482 https://keras.io/api/keras_tuner/tuners/base_tuner/#search-method
483 https://keras.io/api/keras_tuner/tuners/random/#randomsearch-class

(continued from previous page)

```
Trial 04 summary
Hyperparameters:
layers: 3
neurons_per_layer: 20
Score: 0.9445000290870667

Trial 01 summary
Hyperparameters:
layers: 1
neurons_per_layer: 40
Score: 0.9440000057220459

Trial 06 summary
Hyperparameters:
layers: 1
neurons_per_layer: 30
Score: 0.9401666522026062

Trial 07 summary
Hyperparameters:
layers: 1
neurons_per_layer: 20
Score: 0.9340833425521851

Trial 03 summary
Hyperparameters:
layers: 1
neurons_per_layer: 10
Score: 0.922249972820282

Trial 00 summary
Hyperparameters:
layers: 3
neurons_per_layer: 10
Score: 0.9195833206176758
```

To get the best model we may call `Tuner.get_best_models`[484], which returns a sorted (best first) list of trained `Model` instances. Alternatively, we may call `Tuner.get_best_hyperparameters`[485] returning a list of `HyperParameter` objects of the best models. Based on the best hyperparameters we may train corresponding model on the full data set to improve results. Both methods take an argument specifying the number of models to return and defaulting to 1.

```
best_hp = tuner.get_best_hyperparameters()[0]
best_model = build_model(best_hp)
best_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten_1 (Flatten)         (None, 400)               0         
                                                                 
 dense (Dense)               (None, 30)                12030     
                                                                 
 dense_1 (Dense)             (None, 30)                930       
                                                                 
 dense_2 (Dense)             (None, 30)                930       
```

(continues on next page)

[484] https://keras.io/api/keras_tuner/tuners/base_tuner/#get_best_models-method
[485] https://keras.io/api/keras_tuner/tuners/base_tuner/#get_best_hyperparameters-method

(continued from previous page)

```
 dense_3 (Dense)             (None, 10)                310

=================================================================
Total params: 14,200
Trainable params: 14,200
Non-trainable params: 0
_________________________________________________________________
```

```
best_model.fit(train_images, train_labels, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.7758 -␣
 ↪categorical_accuracy: 0.7655
Epoch 2/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.3384 -␣
 ↪categorical_accuracy: 0.9027
Epoch 3/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2765 -␣
 ↪categorical_accuracy: 0.9202
Epoch 4/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2381 -␣
 ↪categorical_accuracy: 0.9314
Epoch 5/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.2127 -␣
 ↪categorical_accuracy: 0.9384
Epoch 6/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1937 -␣
 ↪categorical_accuracy: 0.9430
Epoch 7/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1786 -␣
 ↪categorical_accuracy: 0.9478
Epoch 8/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1668 -␣
 ↪categorical_accuracy: 0.9514
Epoch 9/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1571 -␣
 ↪categorical_accuracy: 0.9545
Epoch 10/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1478 -␣
 ↪categorical_accuracy: 0.9569
```

```
<keras.callbacks.History at 0x7f86bc2a9960>
```

```
test_loss, test_metric = best_model.evaluate(test_images, test_labels)

test_loss, test_metric
```

```
1875/1875 [==============================] - 2s 1000us/step - loss: 0.1729 -␣
 ↪categorical_accuracy: 0.9482
```

```
(0.17285792529582977, 0.948199987411499)
```

26.3.3 Stopping Criteria

So far we stopped training after a fixed number of epochs. But Keras also implements a mechanism for stopping training if loss or metrics stop improving. That mechanism is denoted as `callbacks`[486]. We simply have to create a suitable `Callback` object and pass it to the fit method. Stopping criteria can be implemented with `EarlyStopping`[487] objects (it's a subclass of `Callback`). If we want to stop training if the validation loss starts to increase for at least 3 consecutive epochs, we have to pass `monitor='val_loss', mode='min', patience=3, restore_best_weights=True`. The last argument tells the `fit` method to not return the final model, but the best model.

```
es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=3,␣
↪restore_best_weights=True)
best_model.fit(train_images, train_labels, validation_split=0.2, epochs=1000,␣
↪callbacks=[es])
```

```
Epoch 1/1000
1500/1500 [==============================] - 2s 2ms/step - loss: 0.1421 -␣
↪categorical_accuracy: 0.9586 - val_loss: 0.1316 - val_categorical_accuracy: 0.
↪9625
Epoch 2/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.1361 -␣
↪categorical_accuracy: 0.9603 - val_loss: 0.1337 - val_categorical_accuracy: 0.
↪9603
Epoch 3/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.1307 -␣
↪categorical_accuracy: 0.9615 - val_loss: 0.1365 - val_categorical_accuracy: 0.
↪9598
Epoch 4/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.1250 -␣
↪categorical_accuracy: 0.9633 - val_loss: 0.1283 - val_categorical_accuracy: 0.
↪9613
Epoch 5/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.1196 -␣
↪categorical_accuracy: 0.9649 - val_loss: 0.1277 - val_categorical_accuracy: 0.
↪9620
Epoch 6/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.1153 -␣
↪categorical_accuracy: 0.9659 - val_loss: 0.1260 - val_categorical_accuracy: 0.
↪9629
Epoch 7/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.1112 -␣
↪categorical_accuracy: 0.9673 - val_loss: 0.1286 - val_categorical_accuracy: 0.
↪9628
Epoch 8/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.1070 -␣
↪categorical_accuracy: 0.9683 - val_loss: 0.1286 - val_categorical_accuracy: 0.
↪9614
Epoch 9/1000
1500/1500 [==============================] - 2s 2ms/step - loss: 0.1032 -␣
↪categorical_accuracy: 0.9697 - val_loss: 0.1299 - val_categorical_accuracy: 0.
↪9618
```

```
<keras.callbacks.History at 0x7f86bc0224a0>
```

Resulting accuracy on test set:

[486] https://keras.io/api/callbacks/
[487] https://keras.io/api/callbacks/early_stopping/

```
test_loss, test_metric = best_model.evaluate(test_images, test_labels)
test_loss, test_metric
```

```
1875/1875 [==============================] - 2s 1ms/step - loss: 0.1479 -␣
↪categorical_accuracy: 0.9562
```

```
(0.14791476726531982, 0.9561833143234253)
```

26.3.4 Saving and Loading Models

Keras models provide a `save`[488] to a model to a file. To load a model use `load_model`[489].

```
best_model.save('keras_save_best_model')
```

```
WARNING:absl:Found untraced functions such as _update_step_xla while saving␣
↪(showing 1 of 1). These functions will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: keras_save_best_model/assets
```

```
INFO:tensorflow:Assets written to: keras_save_best_model/assets
```

```
model = keras.models.load_model('keras_save_best_model')
```

26.3.5 Visualization of Training Progress

TensorFlow comes with a visualization tool called TensorBoard. It uses a web interface for visualizing training dynamics and it can be integrated into Jupyter notebooks.

To use TensorBoard we have to pass a `TensorBoard`[490] callback to `fit`. Corresponding constructor takes a path to a directory for storing temporary training data. Running

```
tensorboard --logdir=path/to/directory
```

in the terminal will show an URL to access the TensorBoard interface within a web browser.

To use TensorBoard inside a Jupyter Notebook, execute the magic commands

```
%load_ext tensorboard
%tensorboard --logdir path/to/directory
```

```
es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min', patience=3,␣
↪restore_best_weights=True)
tb = keras.callbacks.TensorBoard('tensorboard_data')
best_model.fit(train_images, train_labels, validation_split=0.2, epochs=1000,␣
↪callbacks=[es, tb])
```

[488] https://keras.io/api/models/model_saving_apis/model_saving_and_loading/#save-method
[489] https://keras.io/api/models/model_saving_apis/model_saving_and_loading/#load_model-function
[490] https://keras.io/api/callbacks/tensorboard/

```
Epoch 1/1000
1500/1500 [==============================] - 2s 2ms/step - loss: 0.1106 -␣
↪categorical_accuracy: 0.9669 - val_loss: 0.1313 - val_categorical_accuracy: 0.
↪9596
Epoch 2/1000
1500/1500 [==============================] - 2s 2ms/step - loss: 0.1069 -␣
↪categorical_accuracy: 0.9683 - val_loss: 0.1273 - val_categorical_accuracy: 0.
↪9618
Epoch 3/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.1036 -␣
↪categorical_accuracy: 0.9689 - val_loss: 0.1255 - val_categorical_accuracy: 0.
↪9626
Epoch 4/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.0994 -␣
↪categorical_accuracy: 0.9700 - val_loss: 0.1300 - val_categorical_accuracy: 0.
↪9615
Epoch 5/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.0970 -␣
↪categorical_accuracy: 0.9715 - val_loss: 0.1274 - val_categorical_accuracy: 0.
↪9628
Epoch 6/1000
1500/1500 [==============================] - 2s 1ms/step - loss: 0.0940 -␣
↪categorical_accuracy: 0.9726 - val_loss: 0.1297 - val_categorical_accuracy: 0.
↪9611
```

```
<keras.callbacks.History at 0x7f869ae5ae60>
```

```
%load_ext tensorboard
%tensorboard --logdir tensorboard_data
```

26.4 Convolutional Neural Networks

Convolutional ANNs (CNNs or ConvNets for short) are specially structured layered feedforward ANNs. Their main field of application are computer vision tasks, but also predictions based on time series data. CNNs have the following special properties:

- **Spacial relationship between neurons in a layer:** A layer's neurons are arranged in a grid (1d or 2d or 3d or higher). Thus, we may talk about neighboring neurons or about distance between two neurons.
- **Local connectivity:** Neurons aren't connected to all neurons of the previous layer. Instead, each neuron has only few connections to the previous layer and all those connected neurons are located close to each other.
- **Weight sharing:** Different connections between neurons may share the same weight. Thus, the number of trainable weights is much lower than the number of connections between neurons.

Before we present the details of CNNs we have to understand *convolutions*, an important mathematical tool for signal and image processing. We only consider convolutions for finite discrete signals (vectors, images), not for continuous signals or signals of infinite length.

26.4.1 Convolutions in 1d

By 1d-convolution we refer to a mathematical operation taking two input vectors and yielding one output vector. One of the input vectors is the signal, the other the *convolution kernel* or *filter*. The filter is much shorter than the signal. The length of the output vector is more or less the same as the input vector.

Denote the input vector by $u\in\mathbb{R}^m$ and the filter by $w\in\mathbb{R}^p$ with $p\leq m$ (typically p is odd). We denote the convolution of u and w by $u*w$. It's a vector in $\mathbb{R}^{m-p+1}$ with components

$$[u*w]_\kappa:=\sum_{\nu=1}^{p}u_{\kappa+\nu-1}\,w_\nu.$$

In machine learning contexts convolutions are defined as we did here. In mathematics w_ν is replaced by $w_{p+1-\nu}$. If w is symmetric, that is, $w_\nu=w_{p+1-\nu}$ for $\nu=1,\ldots,p$, then both variants coincide.

To see the principle we consider an example with $m=9$ and $p=3$. Thus, the convolution will have 7 components.

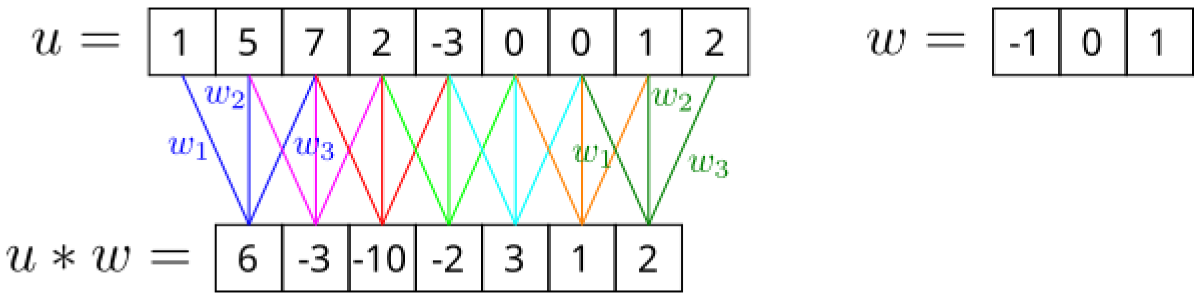


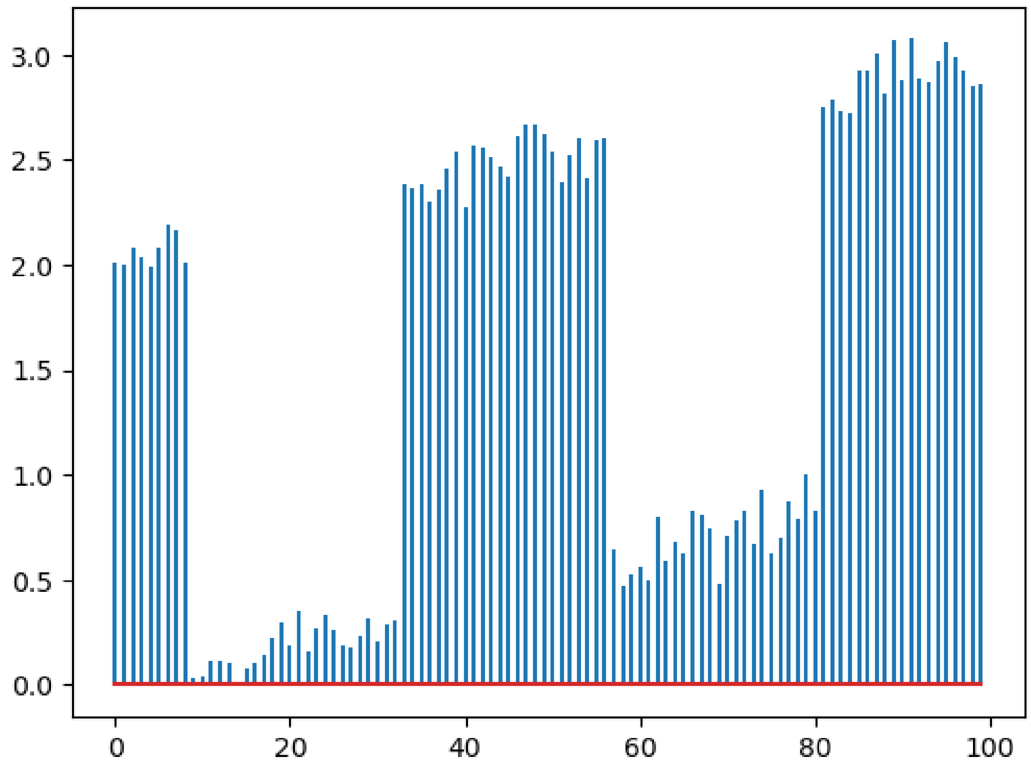
Fig. 26.8: For computing 1d convolutions place the filter at the signal's start, compute the inner product of signal and filter, then move the filter one step to the right, compute the inner product, and so on.

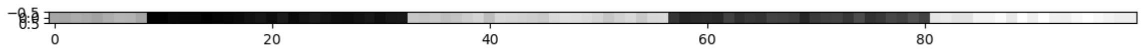
With convolutions we may filter information out of a signal. To see this effect we consider longer signals.

```
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.default_rng(0)

def plot_signal(x):

    fig, ax = plt.subplots()
    ax.stem(x, markerfmt=' ')
    plt.show()

    fig = plt.figure(figsize=(14, 1))
    plt.imshow(x.reshape(1, -1), cmap='gray')
    plt.show()
```

Consider the following signal:

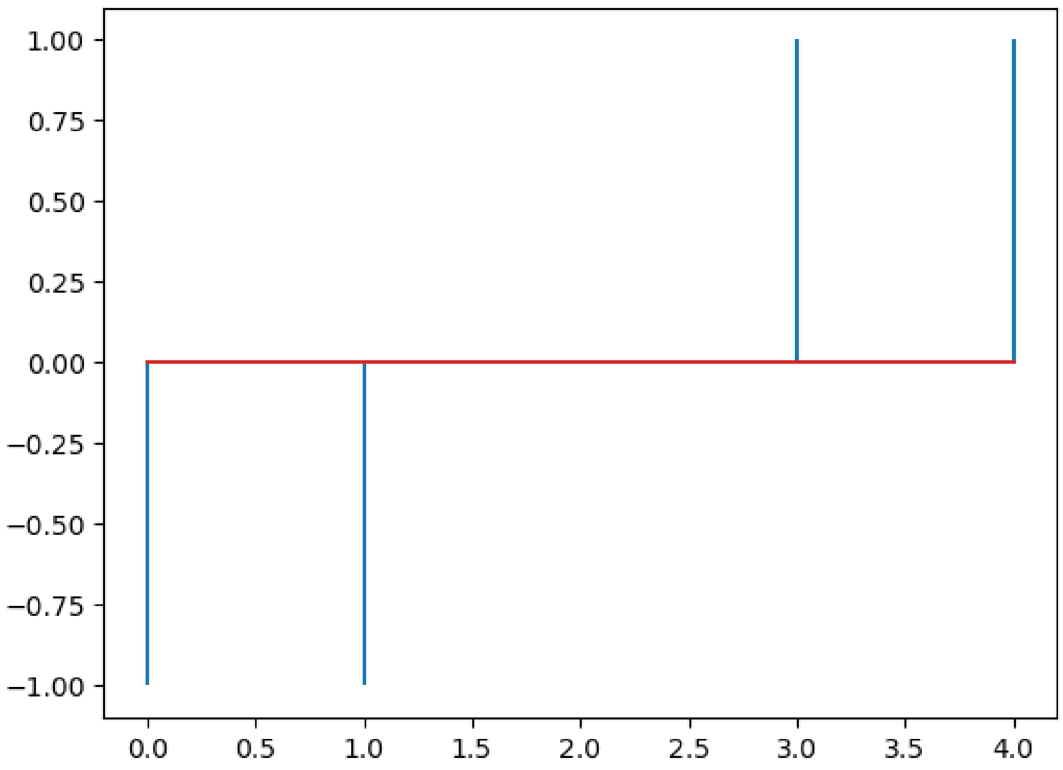
```
u = np.abs(1 + np.sign(np.sin(np.linspace(2, 15, 100)))
             + np.linspace(0, 1, 100)
             + rng.normal(0, 0.1, size=100))

plot_signal(u)
```

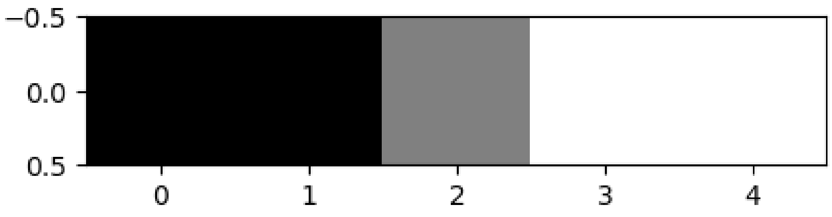




To detect jumps in the signal we may use a suitable filter, looking itself like a jump in a signal.

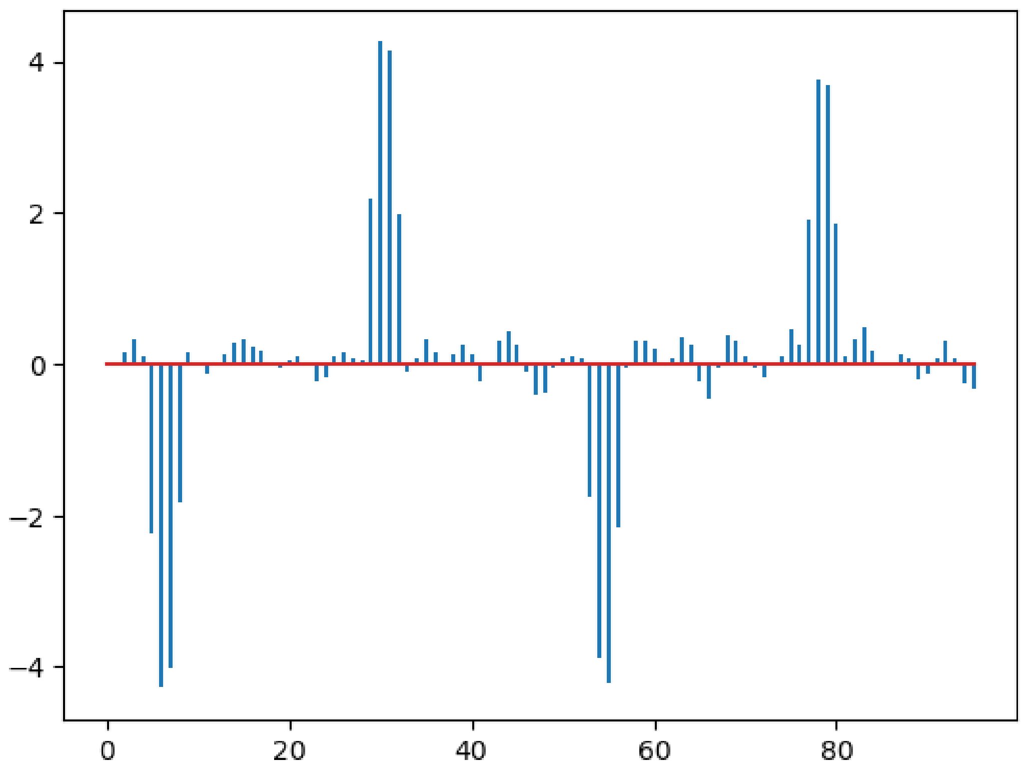
```
w = np.array([-1, -1, 0, 1, 1])

plot_signal(w)
```

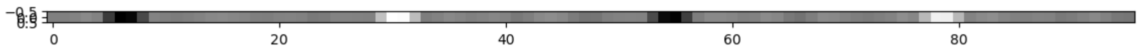




The filtered signal (that is, the convolution of signal and filter) has peaks at the jump positions and is almost zero else.

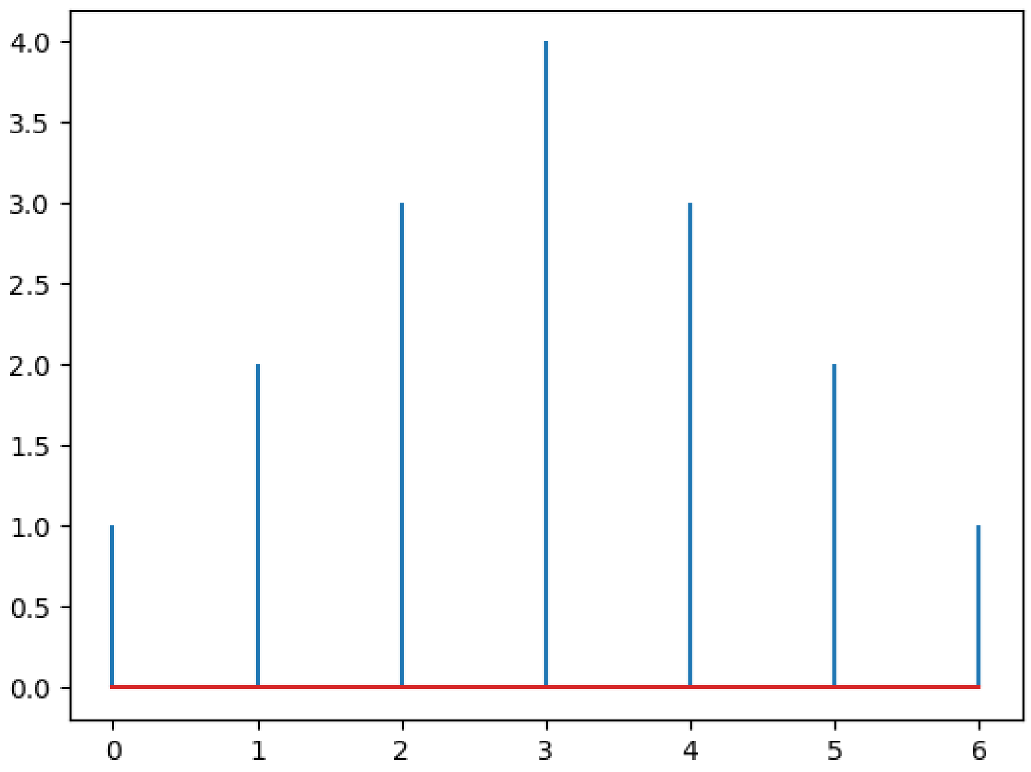
```
conv = np.empty(u.size - w.size + 1)
for k in range(0, conv.size):
    conv[k] = np.sum(u[k:(k + w.size)] * w)

plot_signal(conv)
```

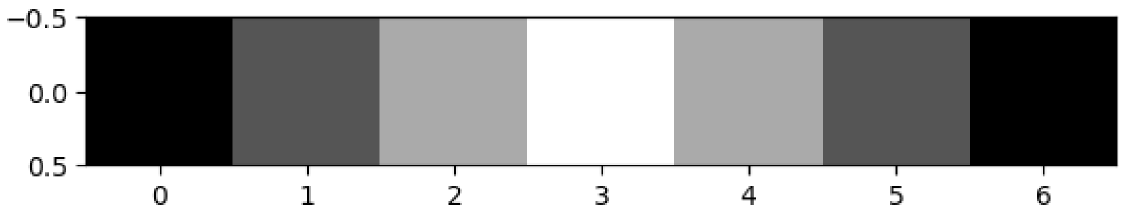




Another application of filters is blurring for noise reduction.

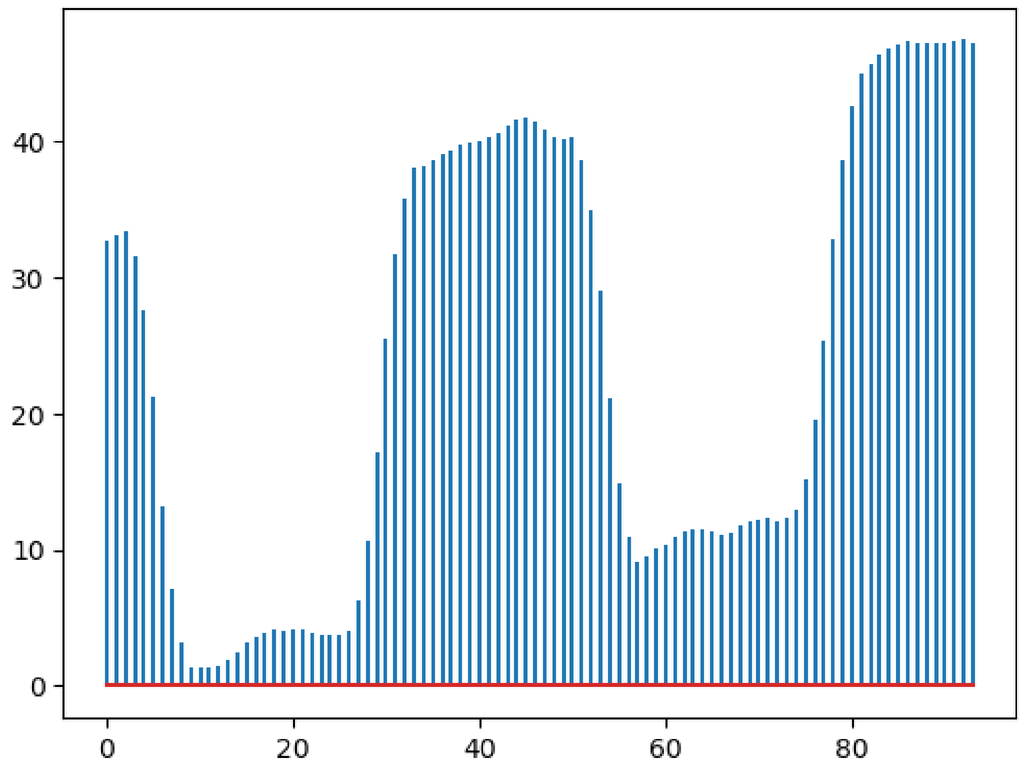
```
w = np.array([1, 2, 3, 4, 3, 2, 1])

plot_signal(w)
```

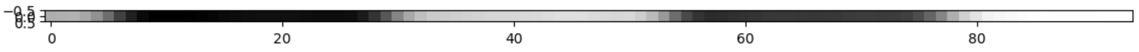


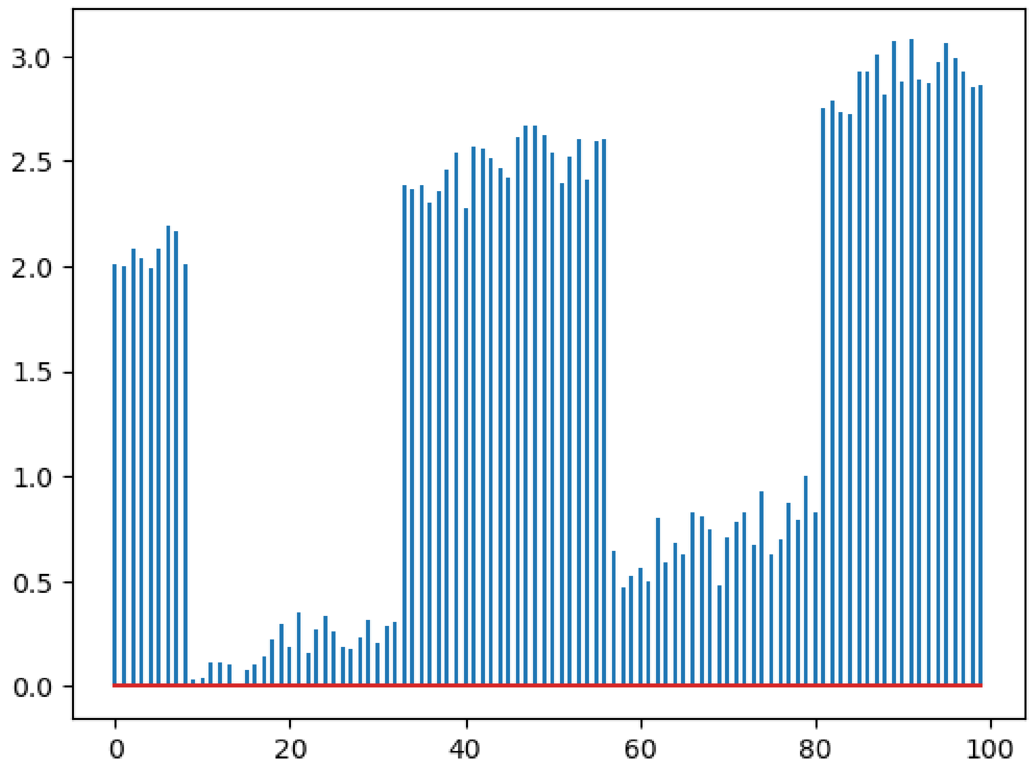


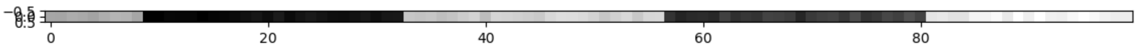
The filtered signal shows less oscillations.

```
conv = np.empty(u.size - w.size + 1)
for k in range(0, conv.size):
    conv[k] = np.sum(u[k:(k + w.size)] * w)

plot_signal(conv)
plot_signal(u)
```









Filtering decreases signal length slightly. To avoid this effect one can add sufficiently many zeros at both sides of the signel ($\frac{p-1}{2}$ zeros at each side if p is the filter length). Then filtering can start with the filter centered at the first regular signal value.

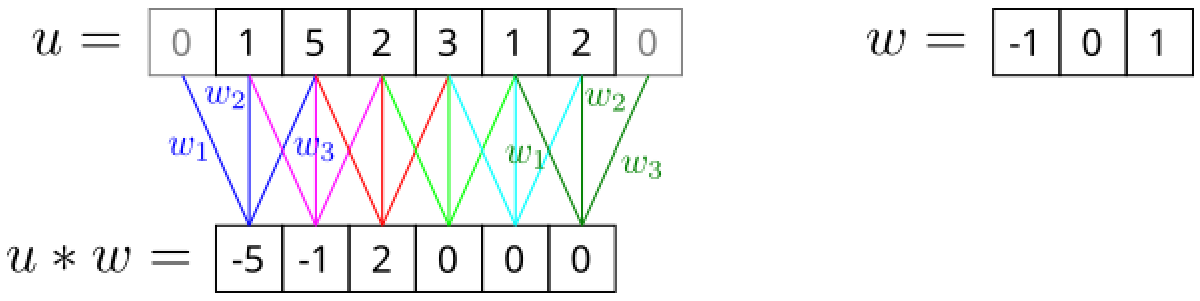


Fig. 26.9: With zero padding original and filtered signal are of same length.

Whether zero padding is necessary and appropriate has to be decided from case to case. It's important to keep in mind that zero padding adds artificial information to the signal. Maybe the signal is a slice of a longer signal. Then zero padding adds the information that the longer signal is zero before and after the sliced signal. That might be wrong.

26.4.2 Convolutions in 2d

Let u be a large matrix (a gray scale image for instance) and w be a small matrix. The convolution of u and w is defined analogously to the 1d case, yielding a matrix of almost the same size as u. Number of rows is decreased by the number of rows in w minus 1. Same for the columns. We skip mathematical formulas and content ourselves with an example.

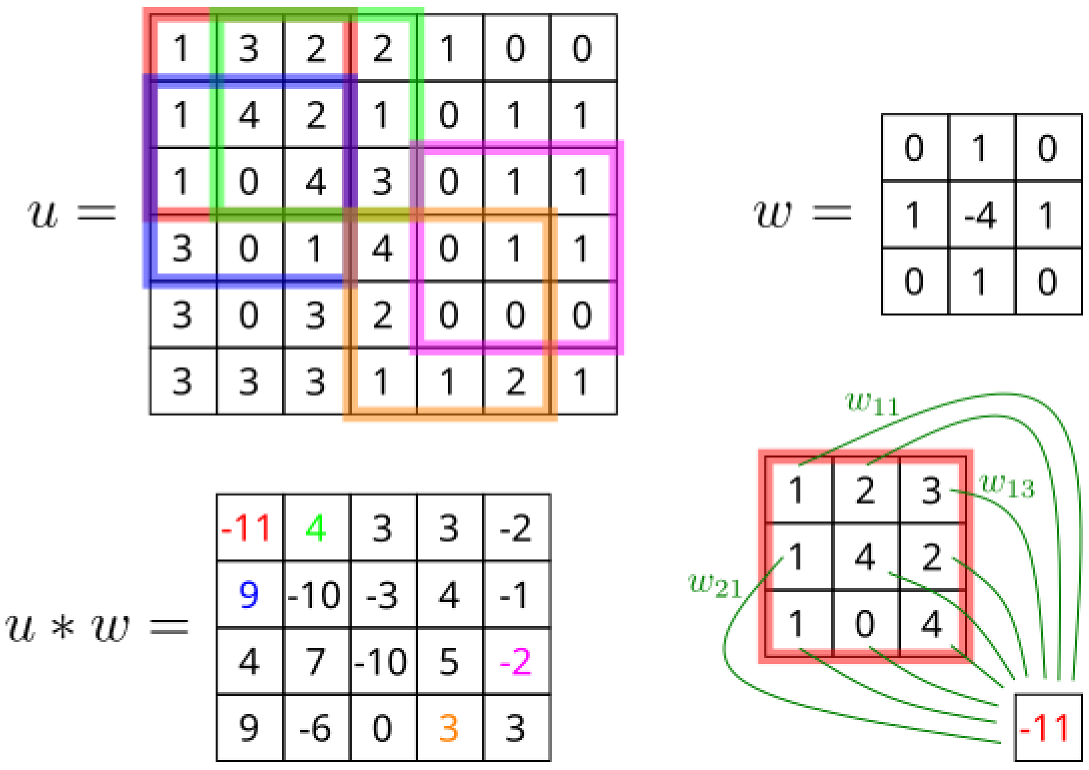
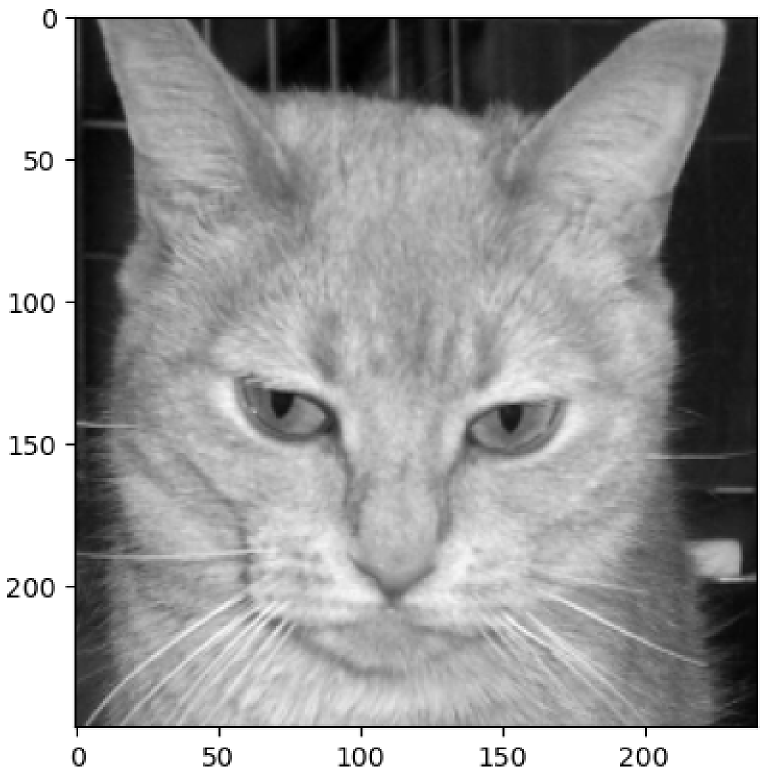


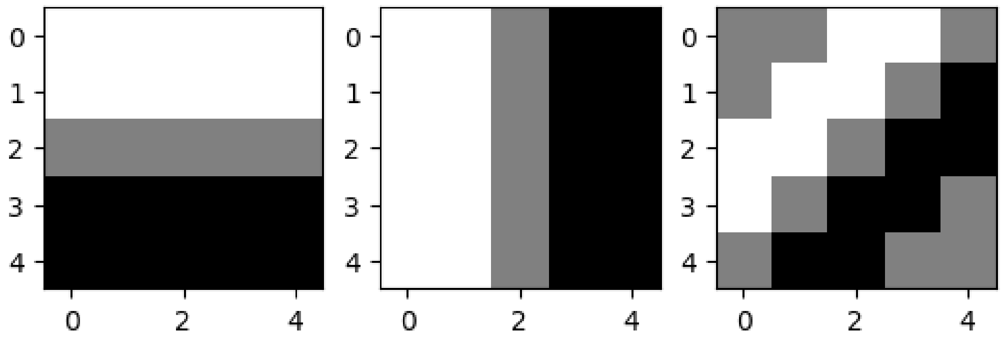
Fig. 26.10: Computations for 2d convolutions follow the same scheme as for 1d convolutions.

Like in 1d we may use convolutions to detect certain features in images or to remove noise. Here is an edge detection example:

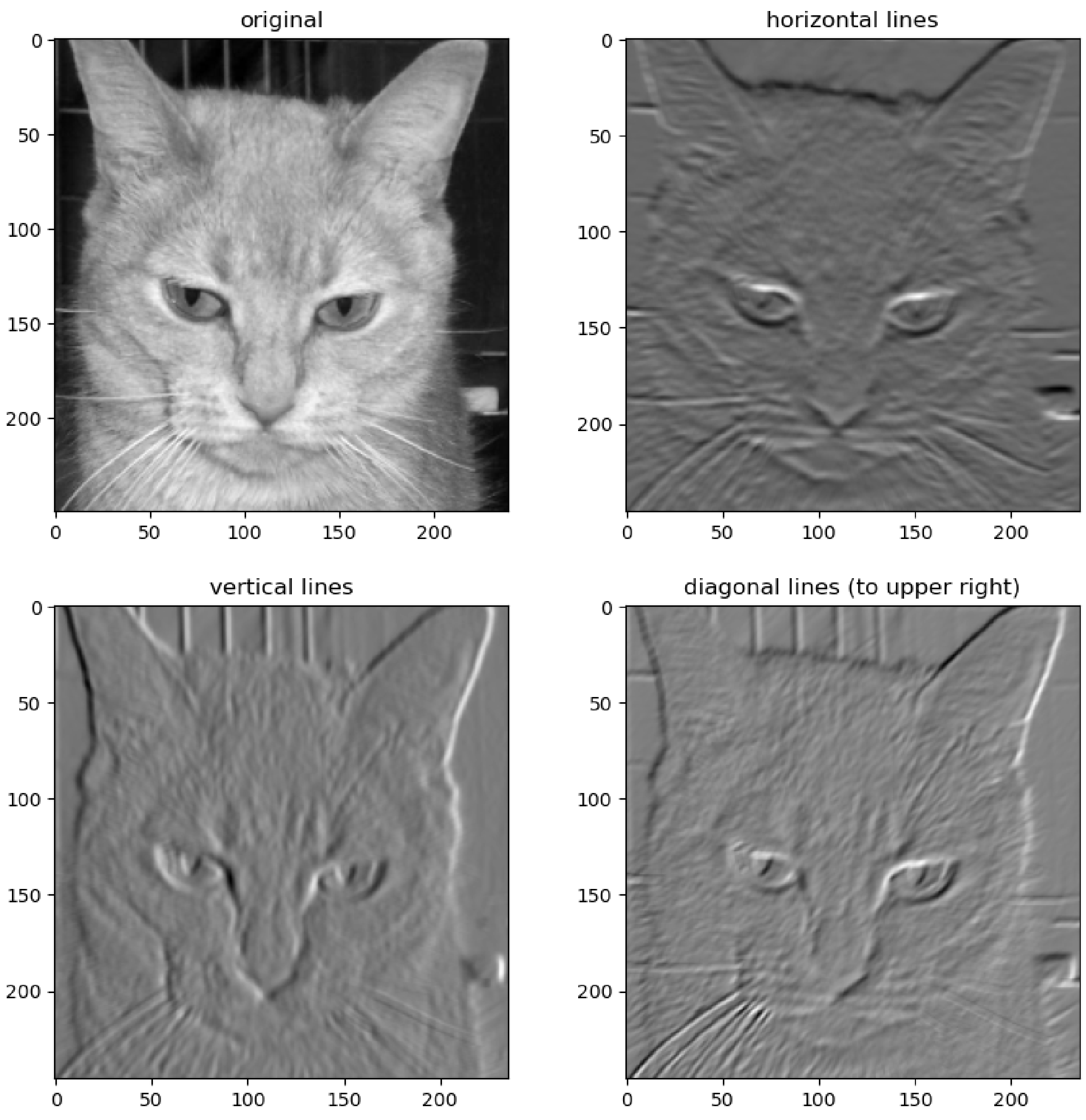
```
import imageio.v3 as imageio
```

```
u = imageio.imread('cat.png')[:, :, 0]
plt.imshow(u, cmap='gray')
plt.show()

wh = np.array([[1, 1, 1, 1, 1],
               [1, 1, 1, 1, 1],
               [0, 0, 0, 0, 0],
               [-1, -1, -1, -1, -1],
               [-1, -1, -1, -1, -1]])
wv = wh.T
wd = np.array([[0, 0, 1, 1, 0],
               [0, 1, 1, 0, -1],
               [1, 1, 0, -1, -1],
               [1, 0, -1, -1, 0],
               [0, -1, -1, 0, 0]])

fig, [ax1, ax2, ax3] = plt.subplots(1, 3)
ax1.imshow(wh, cmap='gray')
ax2.imshow(wv, cmap='gray')
ax3.imshow(wd, cmap='gray')
plt.show()
```





```
def get_conv2d(u, w):
    conv = np.empty((u.shape[0] - w.shape[0] + 1, u.shape[1] - w.shape[1] + 1))
    for k in range(0, conv.shape[0]):
        for l in range(0, conv.shape[1]):
            conv[k, l] = np.sum(u[k:(k + w.shape[0]), l:(l + w.shape[1])] * w)
    return conv

convh = get_conv2d(u, wh)
convv = get_conv2d(u, wv)
convd = get_conv2d(u, wd)

fig, [[ax1, ax2], [ax3, ax4]] = plt.subplots(2, 2, figsize=(10, 10))
ax1.imshow(u, cmap='gray')
ax1.set_title('original')
ax2.imshow(convh, cmap='gray')
ax2.set_title('horizontal lines')
ax3.imshow(convv, cmap='gray')
ax3.set_title('vertical lines')
ax4.imshow(convd, cmap='gray')
ax4.set_title('diagonal lines (to upper right)')
plt.show()
```

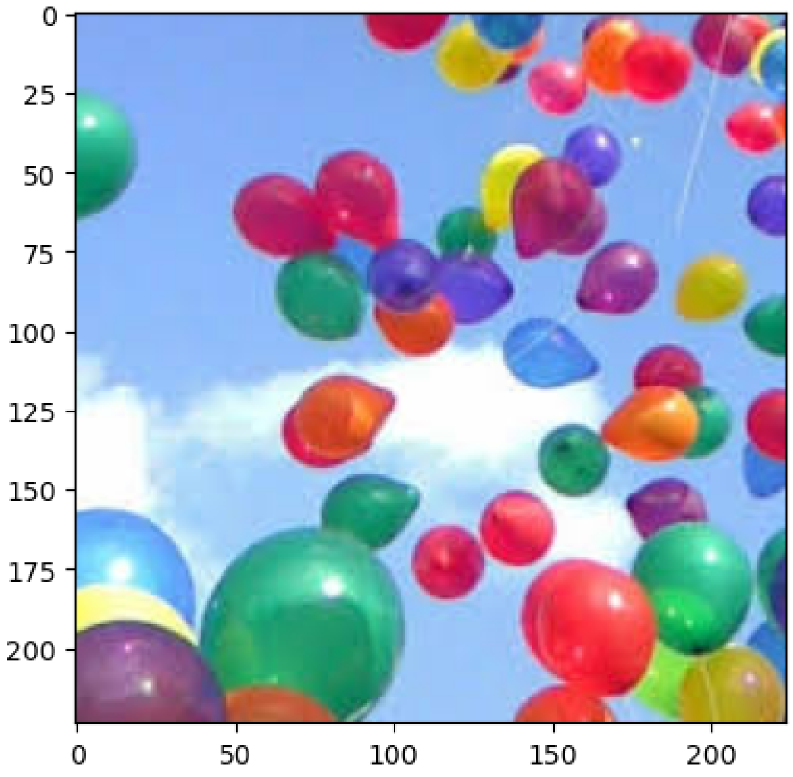


26.4.3 Convolutions in 3d

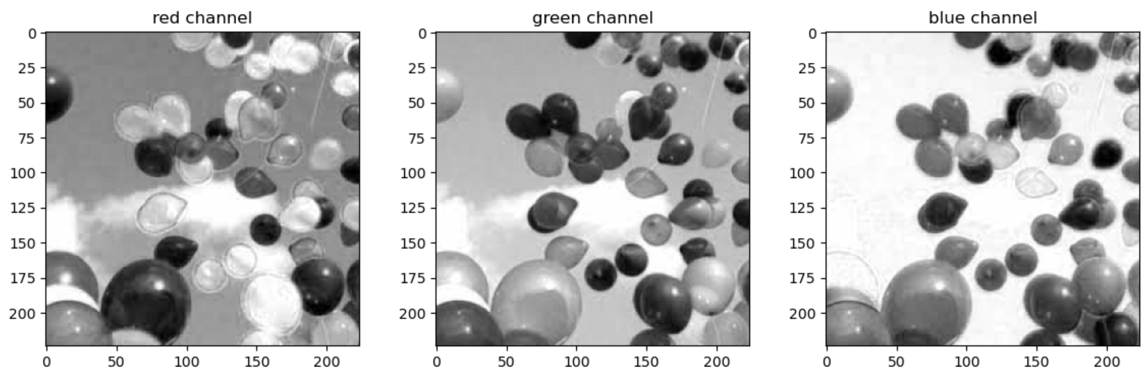
Digital color images are represented as a set of three matrices in computer science. Each pixel's color is composed of certain amounts of red, green and blue (additive color mixing, see Wikipedia on color mixing[491]). One says that the image has three color channels, the red channel, the green channel and the blue channel. Each single channel can be considered a gray scale image. Sometimes there is also a fourth channel, the alpha channel. The alpha channel contains information on transparency (small values for opaque pixels, high values for transparent pixels).

```
u = imageio.imread('balloons.png')[:, :, 0:3]
print(u.shape)
plt.imshow(u)
plt.show()

fig, [ax1, ax2, ax3] = plt.subplots(1, 3, figsize=(14,4))
ax1.imshow(u[:, :, 0], cmap='gray')
ax1.set_title('red channel')
ax2.imshow(u[:, :, 1], cmap='gray')
ax2.set_title('green channel')
ax3.imshow(u[:, :, 2], cmap='gray')
ax3.set_title('blue channel')
plt.show()
```

```
(224, 224, 3)
```



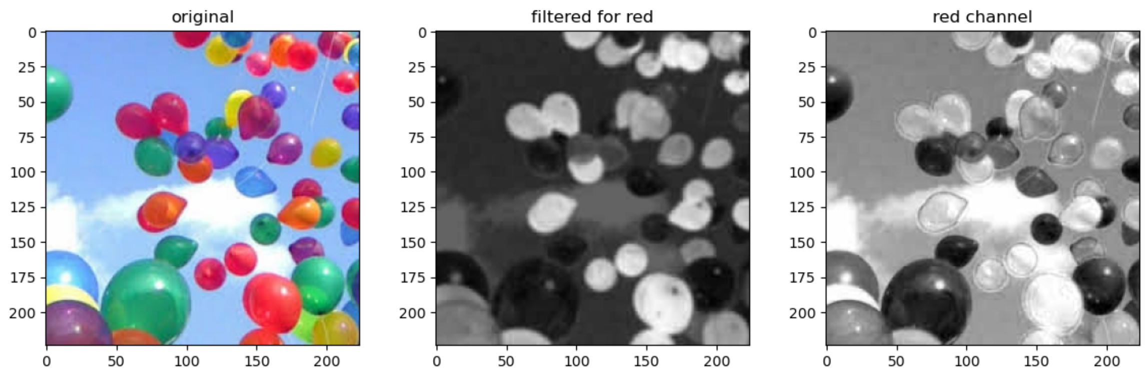
[491] https://en.wikipedia.org/wiki/Color_mixing



Filters in 3d are cuboids of numbers (or stacks of matrices or tensors of rank 3). Convolution takes a subcuboid of the color image, multiplies componentwise by the filter, and then sums up all the products.

We may use 3d convolution for color extraction: If we want to mark regions with red colored objects, we could look at the red channel. But white or yellow or pink objects have maximum values at the red channel, too. Filtering allows to look for differences between channels. We may think of color extraction as looking for jumps or edges in the depth direction.

```
w = np.array([1, -0.5, -0.5]).reshape(1, 1, 3)

conv = np.empty((u.shape[0] - w.shape[0] + 1, u.shape[1] - w.shape[1] + 1, u.
↪shape[2] - w.shape[2] + 1))
for k in range(0, conv.shape[0]):
    for l in range(0, conv.shape[1]):
        for m in range(0, conv.shape[2]):
            conv[k, l, m] = np.sum(u[k:(k + w.shape[0]), l:(l + w.shape[1]), m:(m␣
↪+ w.shape[2])] * w)

fig, [ax1, ax2, ax3] = plt.subplots(1, 3, figsize=(14,4))
ax1.imshow(u)
ax1.set_title('original')
ax2.imshow(conv, cmap='gray')
ax2.set_title('filtered for red')
ax3.imshow(u[:, :, 0], cmap='gray')
ax3.set_title('red channel')
plt.show()
```



26.4.4 Convolutional Layers

CNNs mainly contain so called convolutional layers. A 1d convolutional layer takes a multi-channel signal as input, a 2d convolutional layer takes a multi-channel image as input. The output is a list of filtered signals or images, which again can be considered as a multi-channel signal or image. Filter size in the depth dimension equals the number of input channels. Thus, there is exactly one output channel per filter. The output of a convolutional layer is sometimes referred to as *feature map*, because it contains information about certain features of the input.

Filters correspond to the ANN's weights as follows:

- A convolutional layer has as many neurons as the filtered signal or image has components and neurons are (mentally) arranged in the same layout as the signal's or image's components.
- Each neuron gets input only from the signal or image components used for calculating corresponding component of the convolution (*local connectivity*).
- Weights play the role of the filter's components. Thus, all neurons share the same set of weights (*weight sharing*).

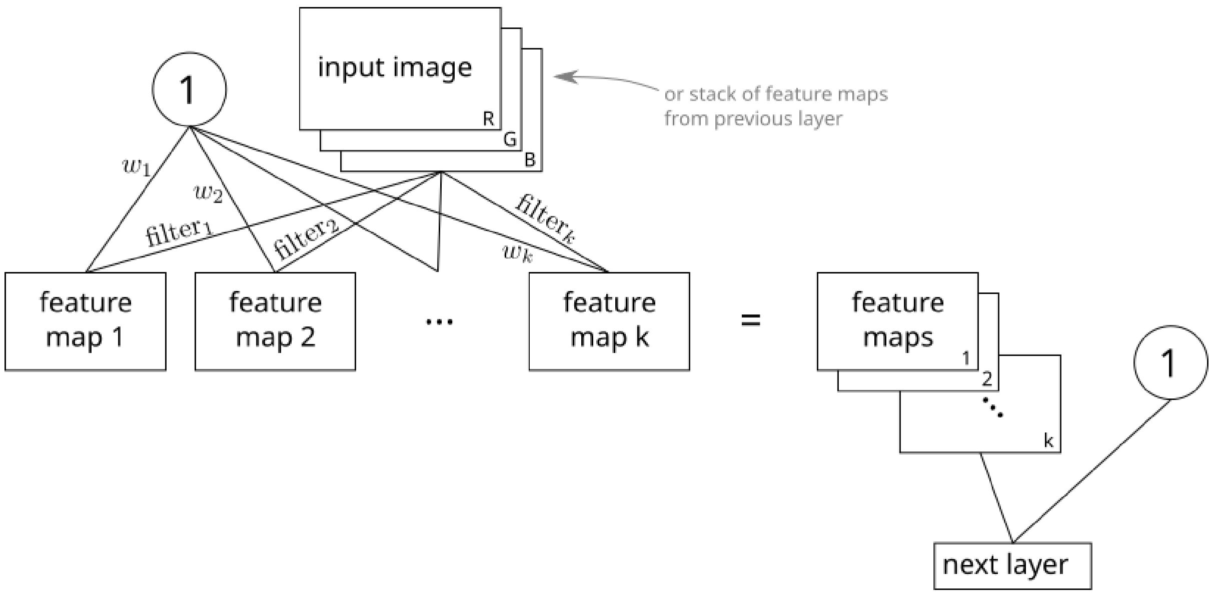


Fig. 26.11: Convolutional layers take a stack of signals/images and yield a stack of filtered signals/images.

Filters for convolution layers are not prescribed as in classical image processing. Instead the CNN has to learn useful filters from training data. As for usual ANNs each layer may get input from an additional bias neuron.

Convolutional layers may have a further parameter: the *stride*. A stride of 1 means that we consider every component of the filtered signal or image. With a stride of 2 we only keep every second component, and so on. Strides greater than one should be used only in combination with large filters. Else we would miss too much information.

Often rectified linear units are used as neurons in convolutional layers. Negative activations indicate presence of features that contradict the filter. If we only want to know if the feature corresponding to a filter is in the image, then the interpretation of negative and zero activation is identical (feature not present in both cases). Thus, it seems reasonable to cut off negative acitvations. That is exactly what rectified linear units do.

The principles of local connectivity and weight sharing lead to much fewer weights to be trained, thus allowing for deeper networks.

26.4.5 Overall CNN structure

Typical CNNs contain a stack of convolutional layers followed by a stack of dense layers. The idea is that the convolutional layers extract features from the inputs and the dense layers combine extracted features to predictions.

The deeper a convolutional layer in the stack the larger the area of the input image having influence on the layer activations. Thus, deeper layers extract less localized features, whereas the first layers only have access to very small regions of the input image.

The stack of convolutional layers may contain so called *pooling layers*. A pooling layer samples feature maps down to smaller feature maps. Usually each feature map is split into disjoint pieces of size 2x2 and each piece is replaced by its average value (*average pooling*) or the maximum value (*max pooling*). The idea behind pooling is that features will not vary much locally or that only most relevant features are of interest locally. Max pooling seems to yield better results than average pooling, but the need for pooling at all is controversial. From the computational point of view pooling reduces feature map sizes and thus the number of weights to train.

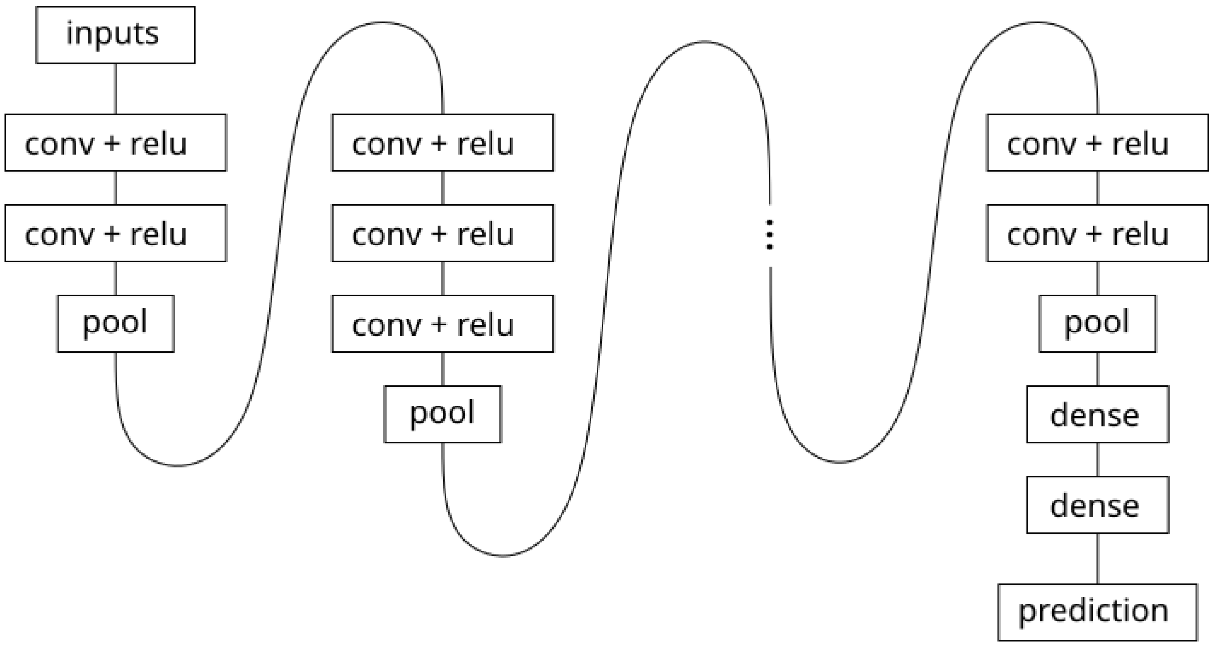


Fig. 26.12: CNNs typically consist of serveral convolution stacks and a small number of dense layers.

26.5 CNNs with Keras

We aim to construct and train a CNN for object detection in images using Keras. The CNN shall decide whether there is a cat or a dog in the image presented to the net. Training data for such tasks can be scraped from the web. But nowadays there are lots of image data bases holding additional information like labels for the images. We use a set of 25000 cat/dog images published under CC0[492] for competition on www.kaggle.com[493]. Next to 25000 labeled (cat or dog) images for training the data set contains 12500 unlabeled images of cats and dogs.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow.keras as keras

data_path = '/home/jef19jdw/myfiles/datasets_teaching/ds2/catsdogs/data/'
```

[492] https://creativecommons.org/publicdomain/zero/1.0/
[493] https://www.kaggle.com/tongpython/cat-and-dog
[494] https://xkcd.com/1425

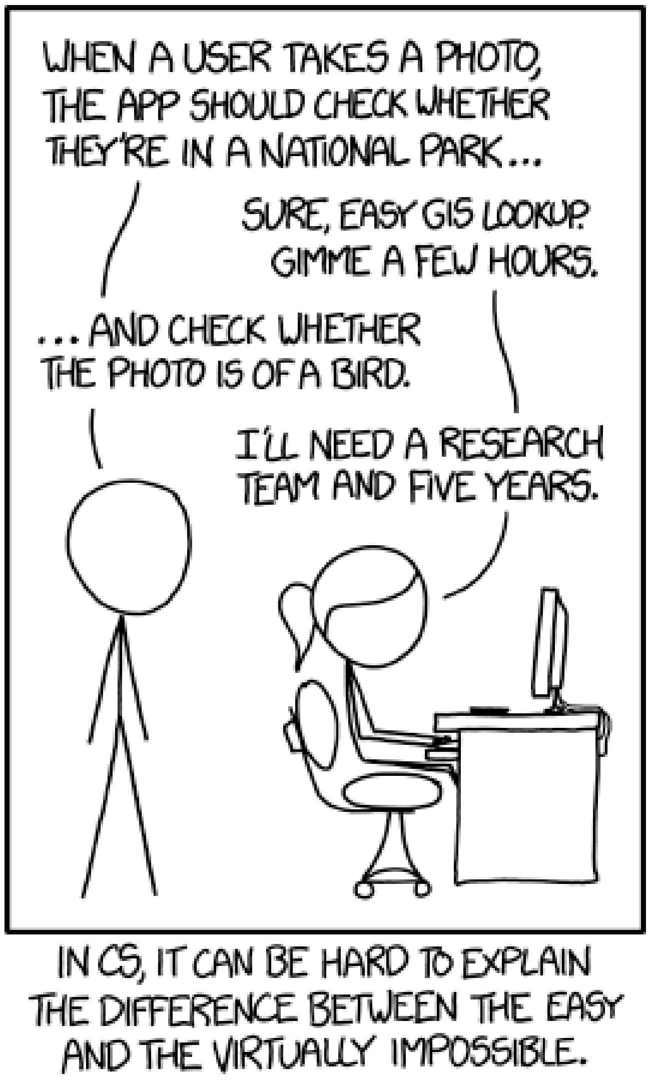

WHEN A USER TAKES A PHOTO, THE APP SHOULD CHECK WHETHER THEY'RE IN A NATIONAL PARK...
SURE, EASY GIS LOOKUP. GIMME A FEW HOURS.
...AND CHECK WHETHER THE PHOTO IS OF A BIRD.
I'LL NEED A RESEARCH TEAM AND FIVE YEARS.
IN CS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALLY IMPOSSIBLE.

Fig. 26.13: In the 60s, Marvin Minsky assigned a couple of undergrads to spend the summer programming a computer to use a camera to identify objects in a scene. He figured they'd have the problem solved by the end of the summer. Half a century later, we're still working on it. Source: Randall Munroe, xkcd.com/1425[494]

```
2023-07-03 05:33:58.382411: I tensorflow/core/platform/cpu_feature_guard.
 ↪cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network␣
 ↪Library (oneDNN) to use the following CPU instructions in performance-
 ↪critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate␣
 ↪compiler flags.
```

```
# workarounds for some problems with Tensorflow (only use if neccessary)

import tensorflow as tf
import os

physical_devices = tf.config.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(physical_devices[0], True)

os.environ['XLA_FLAGS']='--xla_gpu_cuda_data_dir=/usr/lib/cuda'
```

```
2023-07-03 05:34:00.580105: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
 ↪gpu_executor.cc:981] successful NUMA node read from SysFS had negative value␣
 ↪(-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-07-03 05:34:00.626591: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
 ↪gpu_executor.cc:981] successful NUMA node read from SysFS had negative value␣
 ↪(-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-07-03 05:34:00.627646: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
 ↪gpu_executor.cc:981] successful NUMA node read from SysFS had negative value␣
 ↪(-1), but there must be at least one NUMA node, so returning NUMA node zero
```

26.5.1 Loading and Preprocessing Images

Keras supports loading images and labels from directories step by step during training without holding the whole data set in memory. Each class (cat or dog) has to have its own subdirectory. To use this feature we have to call `keras.preprocessing.image_dataset_from_directory`[495], which returns a Tensorflow `Dataset`[496] object. That object is an iterator yielding batches of images and corresponding labels.

We use 15000 images for training, 5000 for validation, and 5000 for testing.

Next to loading images from disk `image_dataset_from_directory` may apply simple preprocessing steps. Since all the images have different sizes we have to resize them.

```
img_size = 128    # width and height of images

train_data = keras.preprocessing.image_dataset_from_directory(
    data_path + 'labeled/train',
    label_mode = 'categorical',    # one-hot encoding with two columns
    image_size=(img_size, img_size),
    validation_split=0.25,
    subset='training',
    seed=0
)
val_data = keras.preprocessing.image_dataset_from_directory(
    data_path + 'labeled/train',
    label_mode = 'categorical',
    image_size=(img_size, img_size),
    validation_split=0.25,
    subset='validation',
    seed=0    # same seed as for training
```

(continues on next page)

[495] https://keras.io/api/preprocessing/image/
[496] https://www.tensorflow.org/api_docs/python/tf/data/Dataset

(continued from previous page)

```
)
test_data = keras.preprocessing.image_dataset_from_directory(
    data_path + 'labeled/test',
    label_mode = 'categorical',
    image_size=(img_size, img_size),
)
```

```
Found 20000 files belonging to 2 classes.
Using 15000 files for training.
```

```
2023-07-03 05:34:01.436610: I tensorflow/core/platform/cpu_feature_guard.
↪cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network␣
↪Library (oneDNN) to use the following CPU instructions in performance-
↪critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate␣
↪compiler flags.
2023-07-03 05:34:01.437382: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪gpu_executor.cc:981] successful NUMA node read from SysFS had negative value␣
↪(-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-07-03 05:34:01.438385: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪gpu_executor.cc:981] successful NUMA node read from SysFS had negative value␣
↪(-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-07-03 05:34:01.438842: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪gpu_executor.cc:981] successful NUMA node read from SysFS had negative value␣
↪(-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-07-03 05:34:02.164888: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪gpu_executor.cc:981] successful NUMA node read from SysFS had negative value␣
↪(-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-07-03 05:34:02.165241: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪gpu_executor.cc:981] successful NUMA node read from SysFS had negative value␣
↪(-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-07-03 05:34:02.165544: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪gpu_executor.cc:981] successful NUMA node read from SysFS had negative value␣
↪(-1), but there must be at least one NUMA node, so returning NUMA node zero
2023-07-03 05:34:02.165814: I tensorflow/core/common_runtime/gpu/gpu_device.
↪cc:1613] Created device /job:localhost/replica:0/task:0/device:GPU:0 with␣
↪1649 MB memory:  -> device: 0, name: NVIDIA GeForce MX130, pci bus id:␣
↪0000:01:00.0, compute capability: 5.0
```

```
Found 20000 files belonging to 2 classes.
Using 5000 files for validation.
Found 5000 files belonging to 2 classes.
```

Note that resizing a non-square image to a square image distorts its content. Alternatively we could crop the longer side or fill the shorter with some color. Experiments have shown that distortions from resizing have no substantial influence on the prediction accuracy of the ANN trained with distorted images.

Each iterate generated by one of the data iterators is a tuple containing two NumPy arrays. The first array is a batch of images, the second contains the one-hot encoded labels.

To get an iterate we call Python's built-in functions `iter` and `next` (a TensorFlow `Dataset` object is an iterable object, which becomes an iterator via `iter`).

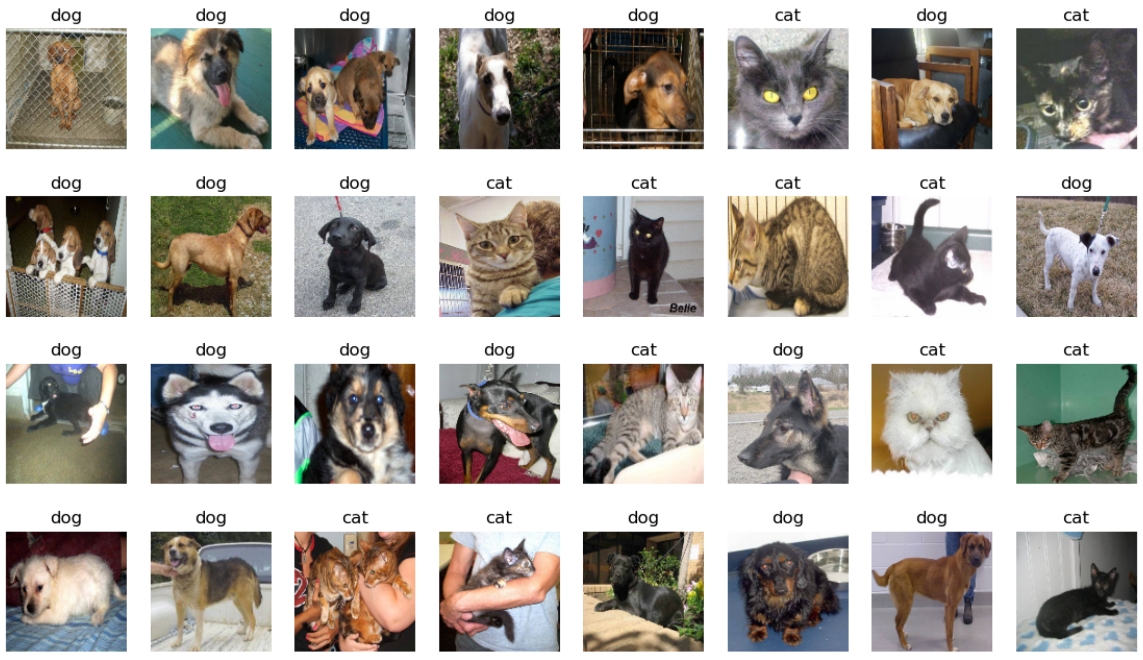
```
images, labels = next(iter(train_data))
images.shape, labels.shape
```

```
(TensorShape([32, 128, 128, 3]), TensorShape([32, 2]))
```

Default batch size is 32. Other batch sizes can be set via `batch_size` argument of `image_dataset_from_directory`. Images are 128x128 and have 3 color channels. We have two classes (cats and dogs), thus two columns for one-hot encoded labels.

Let's have a look at the images:

```
images = 1/255 * images    # scale to range [0, 1]

rows = 4
cols = 8

fig, axs = plt.subplots(4, 8, figsize=(14,8))

for r in range(0, rows):
    for c in range(0, cols):
        idx = r * cols + c
        axs[r, c].imshow(images[idx, :, :, :])
        if labels[idx, 0] == 1:
            axs[r, c].set_title('cat')
        else:
            axs[r, c].set_title('dog')
        axs[r, c].axis('off')

plt.show()
```



26.5.2 Defining the CNN

There is nothing special in defining a CNN with Keras. We simply have to use the correct layer types. We need Conv2D[497], MaxPooling2D[498] and Dense[499].

To scale the image's range to $[0, 1]$ we may use a Rescaling layer[500].

Important: Due to a bug in TensorFlow 2.9 and above training of Keras models with preprocessing layers is

[497] https://keras.io/api/layers/convolution_layers/convolution2d/
[498] https://keras.io/api/layers/pooling_layers/max_pooling2d/
[499] https://keras.io/api/layers/core_layers/dense/
[500] https://keras.io/api/layers/preprocessing_layers/image_preprocessing/rescaling/

extremely slow. See TensorFlow issue[501] for current discussion.

```
model = keras.models.Sequential()

model.add(keras.Input(shape=(img_size, img_size, 3)))
model.add(keras.layers.Rescaling(1/255))

model.add(keras.layers.Conv2D(16, 3, activation='relu', name='conv1'))
model.add(keras.layers.Conv2D(16, 3, activation='relu', name='conv2'))
model.add(keras.layers.MaxPooling2D(name='pool1'))
model.add(keras.layers.Conv2D(32, 3, activation='relu', name='conv3'))
model.add(keras.layers.Conv2D(32, 3, activation='relu', name='conv4'))
model.add(keras.layers.MaxPooling2D(name='pool2'))

model.add(keras.layers.Flatten( name='flatten'))

#model.add(keras.layers.Dropout(0.5, name='dropout'))
#model.add(keras.layers.Dense(10, activation='relu', kernel_regularizer='l2'))

model.add(keras.layers.Dense(10, activation='relu', name='dense1'))
model.add(keras.layers.Dense(10, activation='relu', name='dense2'))

model.add(keras.layers.Dense(2, activation='sigmoid', name='out'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling (Rescaling)       (None, 128, 128, 3)       0

 conv1 (Conv2D)              (None, 126, 126, 16)      448

 conv2 (Conv2D)              (None, 124, 124, 16)      2320

 pool1 (MaxPooling2D)        (None, 62, 62, 16)        0

 conv3 (Conv2D)              (None, 60, 60, 32)        4640

 conv4 (Conv2D)              (None, 58, 58, 32)        9248

 pool2 (MaxPooling2D)        (None, 29, 29, 32)        0

 flatten (Flatten)           (None, 26912)             0

 dense1 (Dense)              (None, 10)                269130

 dense2 (Dense)              (None, 10)                110

 out (Dense)                 (None, 2)                 22

=================================================================
Total params: 285,918
Trainable params: 285,918
Non-trainable params: 0
_________________________________________________________________
```

Number of filters per convolutional layer is chosen from experience. As rule of thumb many small (3x3 or 5x5) filters and several convolutional layers are better than few large filters. From layer to layer number of filters may be

[501] https://github.com/tensorflow/tensorflow/issues/55639

increased as feature map sizes get smaller. Without increasing the number of filters we lose information. This could be good (if lost information is useless for the net's task) or bad (if too much is lost).

By default, convolutional layers in Keras do not zero pad inputs. Padding is controlled by the `padding` argument of `Conv2D`.

26.5.3 Training

For classifications tasks log loss is a good choice. For numerical minimization we may use stochastic gradient descent or Kera's default minimizer RMSProp (a momentum method).

```
model.compile(loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```

Note that `Model.fit` does not support automatic train-validation splits if data is provided as iterator object. Thus, we have to provide a separate validation set.

```
loss = []
val_loss = []
acc = []
val_acc = []
```

```
history = model.fit(train_data, epochs=10, validation_data=val_data)

loss.extend(history.history['loss'])
val_loss.extend(history.history['val_loss'])
acc.extend(history.history['categorical_accuracy'])
val_acc.extend(history.history['val_categorical_accuracy'])
```

```
Epoch 1/10
```

```
2023-07-03 05:34:07.590706: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪dnn.cc:428] Loaded cuDNN version 8401
2023-07-03 05:34:08.843518: I tensorflow/compiler/xla/service/service.cc:173]␣
↪XLA service 0x7f7e59b32570 initialized for platform CUDA (this does not␣
↪guarantee that XLA will be used). Devices:
2023-07-03 05:34:08.843578: I tensorflow/compiler/xla/service/service.cc:181]   ␣
↪StreamExecutor device (0): NVIDIA GeForce MX130, Compute Capability 5.0
2023-07-03 05:34:08.849771: I tensorflow/compiler/mlir/tensorflow/utils/dump_
↪mlir_util.cc:268] disabling MLIR crash reproducer, set env var `MLIR_CRASH_
↪REPRODUCER_DIRECTORY` to enable.
2023-07-03 05:34:09.009400: I tensorflow/compiler/jit/xla_compilation_cache.
↪cc:477] Compiled cluster using XLA!  This line is logged at most once for the␣
↪lifetime of the process.
```

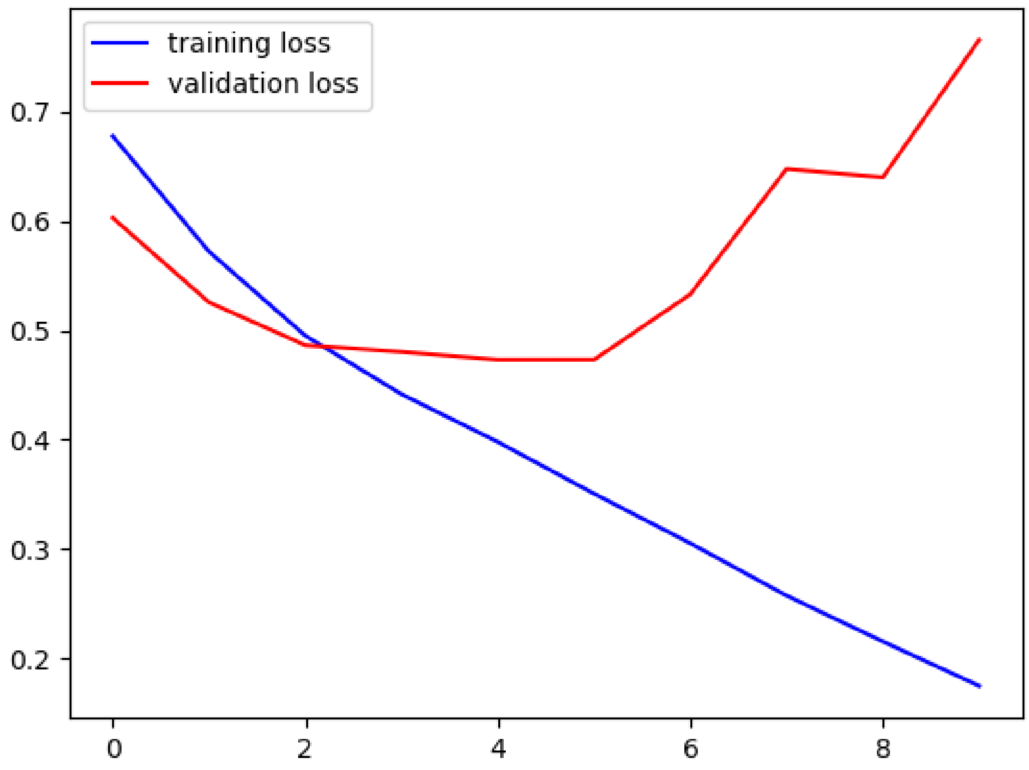
```
469/469 [==============================] - 68s 131ms/step - loss: 0.6774 -␣
↪categorical_accuracy: 0.5473 - val_loss: 0.6027 - val_categorical_accuracy: 0.
↪6616
Epoch 2/10
469/469 [==============================] - 60s 126ms/step - loss: 0.5719 -␣
↪categorical_accuracy: 0.6987 - val_loss: 0.5256 - val_categorical_accuracy: 0.
↪7410
Epoch 3/10
469/469 [==============================] - 60s 126ms/step - loss: 0.4951 -␣
↪categorical_accuracy: 0.7602 - val_loss: 0.4864 - val_categorical_accuracy: 0.
↪7690
Epoch 4/10
469/469 [==============================] - 60s 127ms/step - loss: 0.4416 -␣
↪categorical_accuracy: 0.7971 - val_loss: 0.4803 - val_categorical_accuracy: 0.
↪7660
```

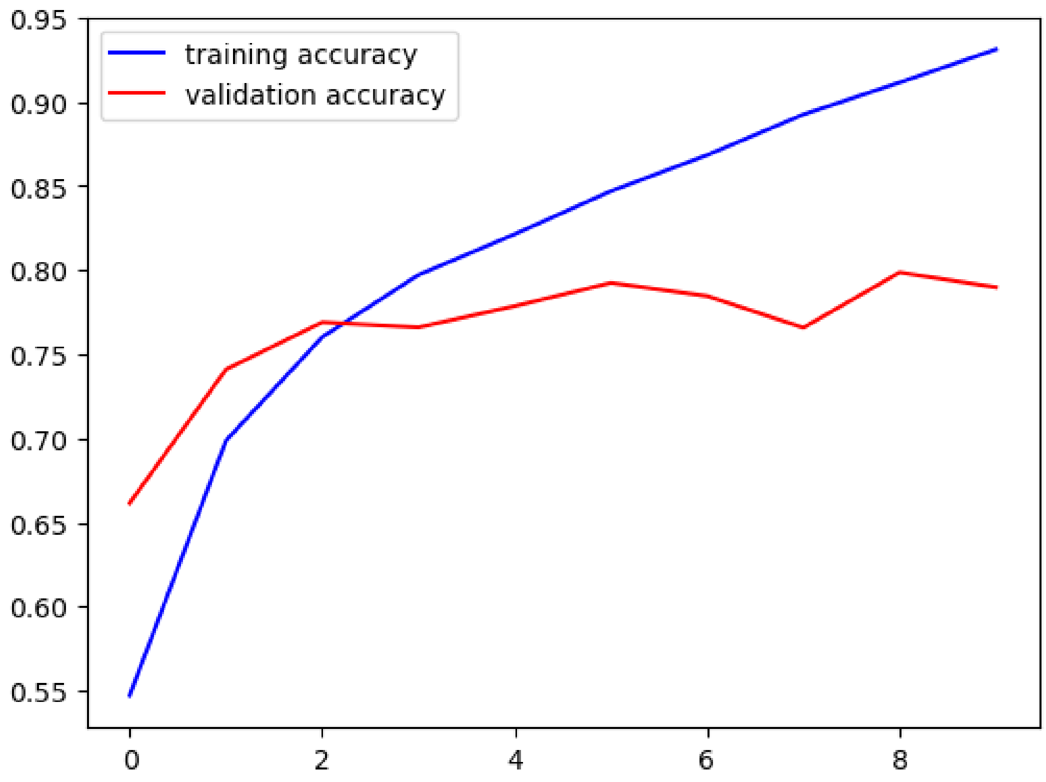
(continues on next page)

(continued from previous page)

```
Epoch 5/10
469/469 [==============================] - 60s 128ms/step - loss: 0.3980 -␣
↪categorical_accuracy: 0.8213 - val_loss: 0.4730 - val_categorical_accuracy: 0.
↪7786
Epoch 6/10
469/469 [==============================] - 60s 128ms/step - loss: 0.3507 -␣
↪categorical_accuracy: 0.8469 - val_loss: 0.4731 - val_categorical_accuracy: 0.
↪7924
Epoch 7/10
469/469 [==============================] - 60s 128ms/step - loss: 0.3054 -␣
↪categorical_accuracy: 0.8683 - val_loss: 0.5329 - val_categorical_accuracy: 0.
↪7846
Epoch 8/10
469/469 [==============================] - 60s 128ms/step - loss: 0.2576 -␣
↪categorical_accuracy: 0.8925 - val_loss: 0.6474 - val_categorical_accuracy: 0.
↪7658
Epoch 9/10
469/469 [==============================] - 60s 128ms/step - loss: 0.2156 -␣
↪categorical_accuracy: 0.9115 - val_loss: 0.6397 - val_categorical_accuracy: 0.
↪7986
Epoch 10/10
469/469 [==============================] - 60s 128ms/step - loss: 0.1751 -␣
↪categorical_accuracy: 0.9311 - val_loss: 0.7653 - val_categorical_accuracy: 0.
↪7898
```

```python
fig, ax = plt.subplots()
ax.plot(loss, '-b', label='training loss')
ax.plot(val_loss, '-r', label='validation loss')
ax.legend()
plt.show()

fig, ax = plt.subplots()
ax.plot(acc, '-b', label='training accuracy')
ax.plot(val_acc, '-r', label='validation accuracy')
ax.legend()
plt.show()
```





After 5 epochs we observe overfitting. Thus, we should add some regularization. We could use one or more Dropout[502] layers. Placement in the layer stack is somewhat arbitrary. But placing them before a dense layer will have more effect because dense layers have many input weights. With dropout we deactivate a random selection of these weights in each training step.

If dropout does not prevent overfitting, more training data should be aquired or penalty based regularization techniques

[502] https://keras.io/api/layers/regularization_layers/dropout/

should be tested. How to obtain more data will be discussed later. Penalty based regularization on a layer's weights can be activatet by passing `kernel_regularizer='l2'` to the layer's constructor. See Keras' documentation for details on penalty based regularization[503].

```
model.save('cnnmodel')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _
↪jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_
↪convolution_op while saving (showing 4 of 4). These functions will not be␣
↪directly callable after loading.
```

```
INFO:tensorflow:Assets written to: cnnmodel/assets
```

```
INFO:tensorflow:Assets written to: cnnmodel/assets
```

```
model = keras.models.load_model('cnnmodel')
```

26.5.4 Evaluation

To evaluate model performance we look at the metrics on the test set, which was not involved in training.

```
test_loss, test_metric = model.evaluate(test_data)
```

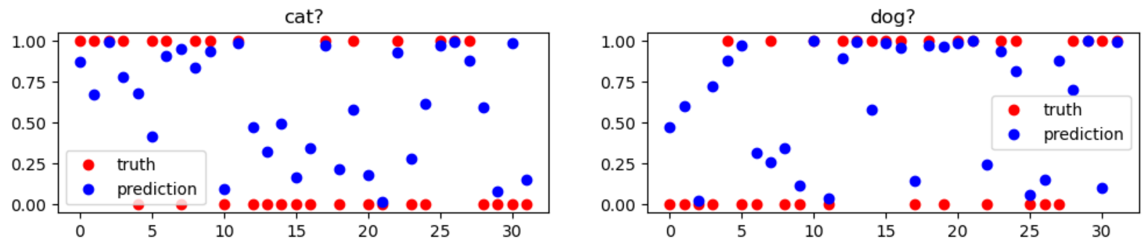
```
157/157 [==============================] - 7s 45ms/step - loss: 0.7523 -␣
↪categorical_accuracy: 0.7894
```

We should have a look at missclassified images to get an idea of what features might cause missclassification.

```
test_images, test_labels = next(iter(test_data))
test_pred = model.predict(test_images)
```

```
1/1 [==============================] - 0s 134ms/step
```

```
fig, [ax1, ax2] = plt.subplots(1, 2, figsize=(12, 2))
ax1.plot(test_labels[:, 0], 'or', label='truth')
ax1.plot(test_pred[:, 0], 'ob', label='prediction')
ax1.legend()
ax1.set_title('cat?')
ax2.plot(test_labels[:, 1], 'or', label='truth')
ax2.plot(test_pred[:, 1], 'ob', label='prediction')
ax2.legend()
ax2.set_title('dog?')
plt.show()
```



[503] https://keras.io/api/layers/regularizers/

```
idx = 3

fig, ax = plt.subplots()
ax.imshow(1/255 * test_images[idx, :, :, :])
ax.set_title('cat: {:.2f}, dog: {:.2f}'.format(test_pred[idx, 0], test_pred[idx,␣
↪1]))
ax.axis('off')
plt.show()
```

cat: 0.78, dog: 0.72

26.6 What did the CNN learn?

We want to obtain some insight into the internal workings of CNNs. The techniques presented below are mainly used for CNNs, but same principles apply to all types of feedforward ANNs.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow.keras as keras
import tensorflow as tf

data_path = '/home/jef19jdw/myfiles/datasets_teaching/ds2/catsdogs/data/'
```

```
2023-07-03 06:38:04.012731: I tensorflow/core/platform/cpu_feature_guard.
 ↪cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network␣
 ↪Library (oneDNN) to use the following CPU instructions in performance-
 ↪critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate␣
 ↪compiler flags.
```

```
# workarounds for some problems with Tensorflow (only use if neccessary)

#import os

#physical_devices = tf.config.list_physical_devices('GPU')
#tf.config.experimental.set_memory_growth(physical_devices[0], True)

#os.environ['XLA_FLAGS']='--xla_gpu_cuda_data_dir=/usr/lib/cuda'
```

```
model = keras.models.load_model('cnnmodel')
model.summary()
```

```
2023-07-03 06:38:05.643827: E tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪driver.cc:267] failed call to cuInit: CUDA_ERROR_UNKNOWN: unknown error
2023-07-03 06:38:05.643893: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:169] retrieving CUDA diagnostic information for host: WHZ-46349
2023-07-03 06:38:05.643911: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:176] hostname: WHZ-46349
2023-07-03 06:38:05.644122: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:200] libcuda reported version is: 470.161.3
2023-07-03 06:38:05.644175: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:204] kernel reported version is: 470.161.3
2023-07-03 06:38:05.644191: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:310] kernel version seems to match DSO: 470.161.3
2023-07-03 06:38:05.645067: I tensorflow/core/platform/cpu_feature_guard.
↪cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network␣
↪Library (oneDNN) to use the following CPU instructions in performance-
↪critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate␣
↪compiler flags.
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 rescaling (Rescaling)       (None, 128, 128, 3)       0         
                                                                 
 conv1 (Conv2D)              (None, 126, 126, 16)      448       
                                                                 
 conv2 (Conv2D)              (None, 124, 124, 16)      2320      
                                                                 
 pool1 (MaxPooling2D)        (None, 62, 62, 16)        0         
                                                                 
 conv3 (Conv2D)              (None, 60, 60, 32)        4640      
                                                                 
 conv4 (Conv2D)              (None, 58, 58, 32)        9248      
                                                                 
 pool2 (MaxPooling2D)        (None, 29, 29, 32)        0         
                                                                 
 flatten (Flatten)           (None, 26912)             0         
                                                                 
 dense1 (Dense)              (None, 10)                269130    
                                                                 
 dense2 (Dense)              (None, 10)                110       
                                                                 
 out (Dense)                 (None, 2)                 22        
                                                                 
=================================================================
Total params: 285,918
```

(continues on next page)

(continued from previous page)

```
Trainable params: 285,918
Non-trainable params: 0
_________________________________________________________________
```

26.6.1 Visualizing Feature Maps

Each convolutional layer outputs a stack of feature maps. In the language of CNNs feature maps are filtered versions of the input image. In the language of ANNs a feature map contains neuron activations. Given an input image we may look at the feature maps to get an idea of what features the learned filters extract.

To get activations of intermediate layers for a given input image we define a new Keras model, which reuses parts of the existing model. When creating a model Keras builds a TensorFlow data structure (the *graph*) representing the flow of data and operations on data. This graph starts with an input node (a `Tensor`[504] object) and ends with the output node (again a `Tensor` object). When calling `Model.predict` Keras takes the data and hands it over to TensorFlow. TensorFlow executes the graph with the provided data and returns the output to Keras. Each layer's output is represented by an intermediate `Tensor` object in the graph, too. So we may fool Keras by creating a new model providing existing `Tensor` objects as inputs and outputs of the model. This feature is not well documented. What is missing in the documentation is the fact, that keyword arguments `inputs` and `outputs` of the `Model` constructor also accept TensorFlows `Tensor` objects instead of Keras' `Input` and `Layer` objects. `Tensor` objects of existing models or layers are accessible through `inputs` and `outputs` member variables. From this knowledge we are able to create a new `Model` instance using an existing TensorFlow graph or parts of it.

```
layer_name = 'conv1'

submodel = keras.models.Model(inputs=model.inputs, outputs=model.get_layer(layer_
↪name).output)
submodel.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0

 rescaling (Rescaling)       (None, 128, 128, 3)       0

 conv1 (Conv2D)              (None, 126, 126, 16)      448

=================================================================
Total params: 448
Trainable params: 448
Non-trainable params: 0
_________________________________________________________________
```

No we load an image and get corresponding predictions from the submodel. Predictions of the submodel are the feature maps (after applying activation function) of the chosen layer in the original model. The image has to be resized to fit the model's input size. We use Kera's `load_img`[505]. This function returns a `PIL image object`[506] which is understood by NumPy.

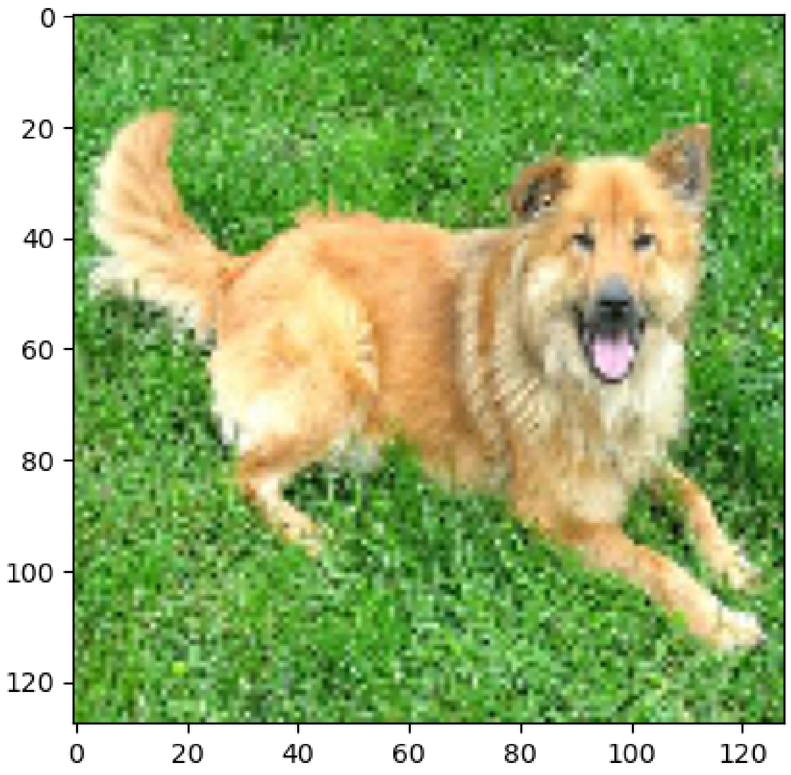
```
img_size = 128
img = keras.preprocessing.image.load_img(data_path + 'unlabeled/4.jpg',
                                         target_size=(img_size, img_size))
img = np.asarray(img, dtype=np.float32)
```

(continues on next page)

[504] https://www.tensorflow.org/api_docs/python/tf/Tensor
[505] https://keras.io/api/preprocessing/image/#loadimg-function
[506] https://pillow.readthedocs.io/en/stable/reference/Image.html#PIL.Image.Image

(continued from previous page)

```
fig, ax = plt.subplots()
ax.imshow(img / 255)
plt.show()

fmaps = submodel.predict(img.reshape(1, img_size, img_size, 3))
fmaps = fmaps.reshape(fmaps.shape[1:])
print(fmaps.shape)
```



```
1/1 [==============================] - 0s 203ms/step
(126, 126, 16)
```

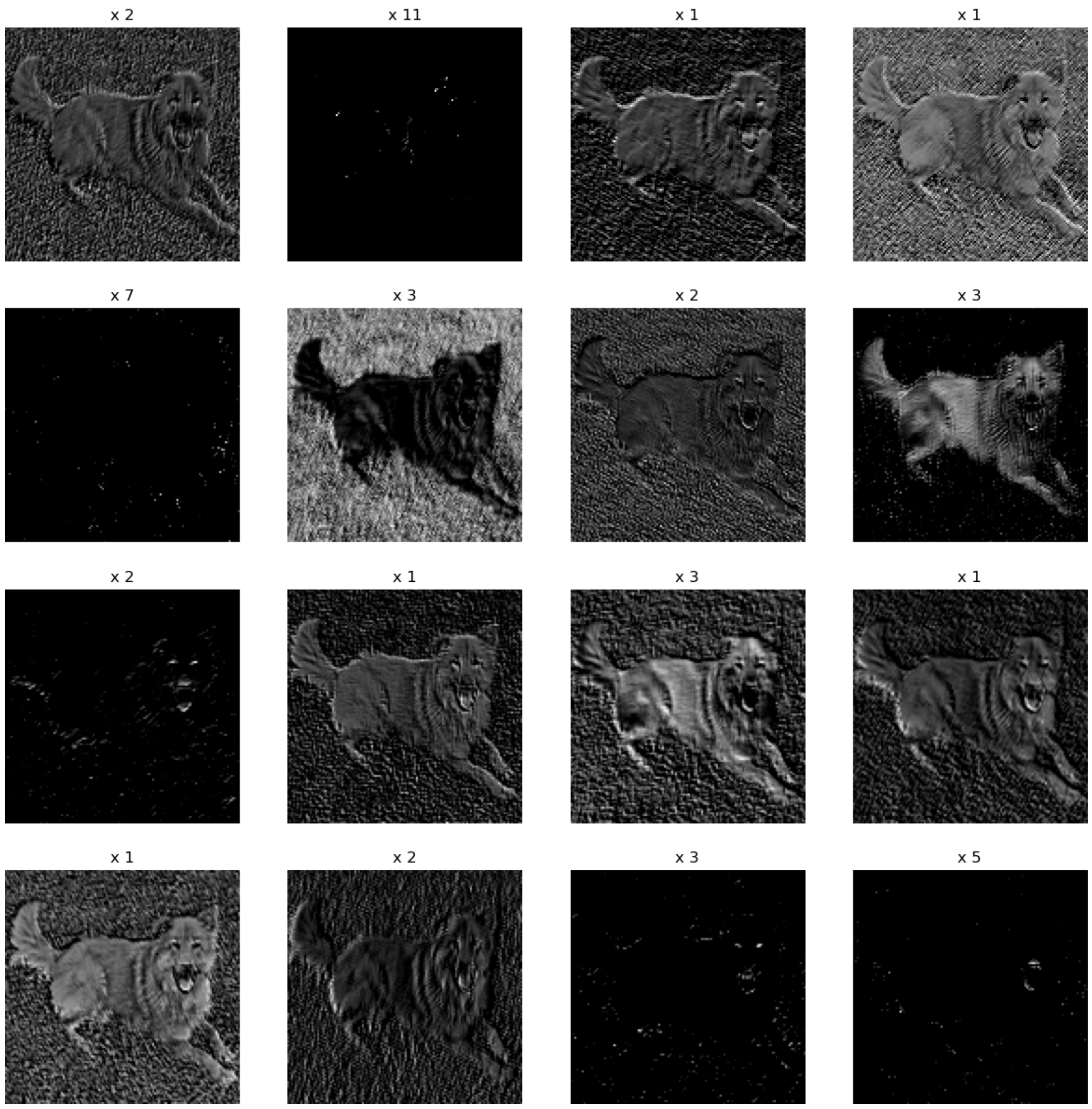
It remains to rescale and plot all the feature maps. We first rescale all feature maps at once to have range $[0, 1]$. Then we rescale each map individually to increase contrast for low intensity images. The individual scaling factor will be shown in the plots. A high factor indicates low intensities.

```
cols = 4
rows = fmaps.shape[2] // cols

fmaps = 1 / (fmaps.max() - fmaps.min()) * (fmaps - fmaps.min())

fig, axs = plt.subplots(rows, cols, figsize=(15, 15))

for r in range(0, rows):
    for c in range(0, cols):
        fmap = fmaps[:, :, r * cols + c]
        if fmap.max() > 0:
            fac = 1 / fmap.max()
            fmap = fac * fmap
        else:
            fac = 1
```

(continues on next page)

(continued from previous page)

```
        axs[r, c].imshow(fmap, cmap='gray')
        axs[r, c].axis('off')
        axs[r, c].set_title('x {:.0f}'.format(fac))

plt.show()
```



26.6.2 Visualizing Filters

Each convolutional layer is defined by a list of filters. Filters are a set of shared weights. We may obtain weights of a layer by calling `Layer.get_weights`[507]. For layers with input from a bias neuron the method returns a list with two items. First item is a NumPy array of regular weights, second is a NumPy array of bias weights.

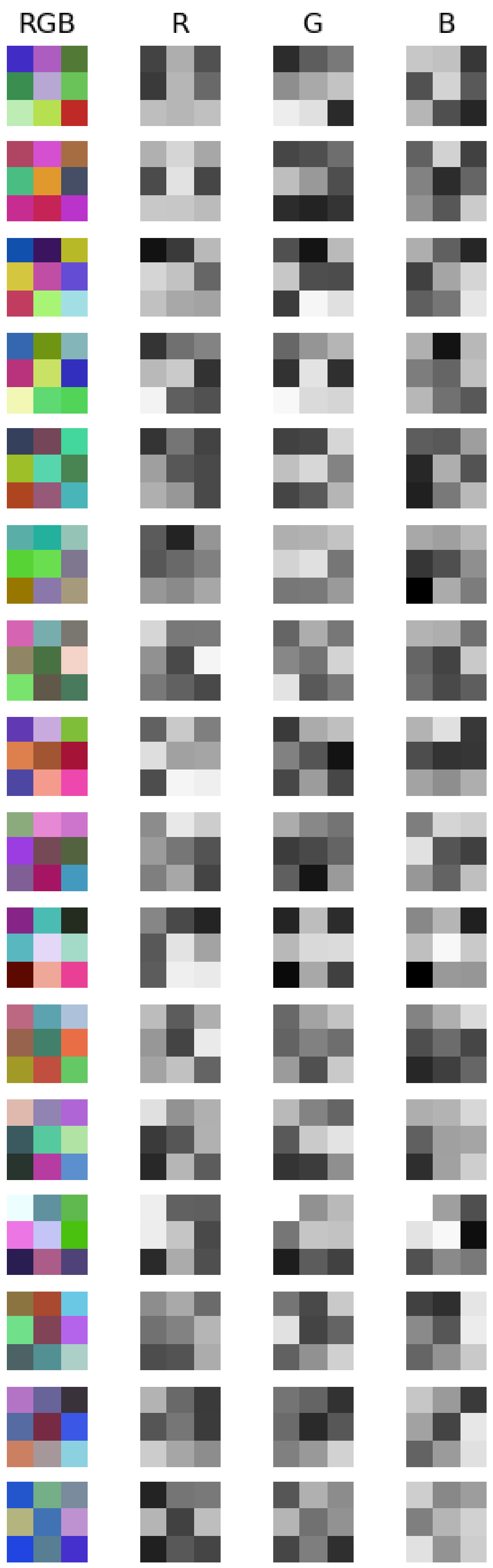
```
layer = model.get_layer('conv1')

filters, bias_weights = layer.get_weights()
print(filters.shape, bias_weights.shape)
```

[507] https://keras.io/api/layers/base_layer/#getweights-method

```
(3, 3, 3, 16) (16,)
```

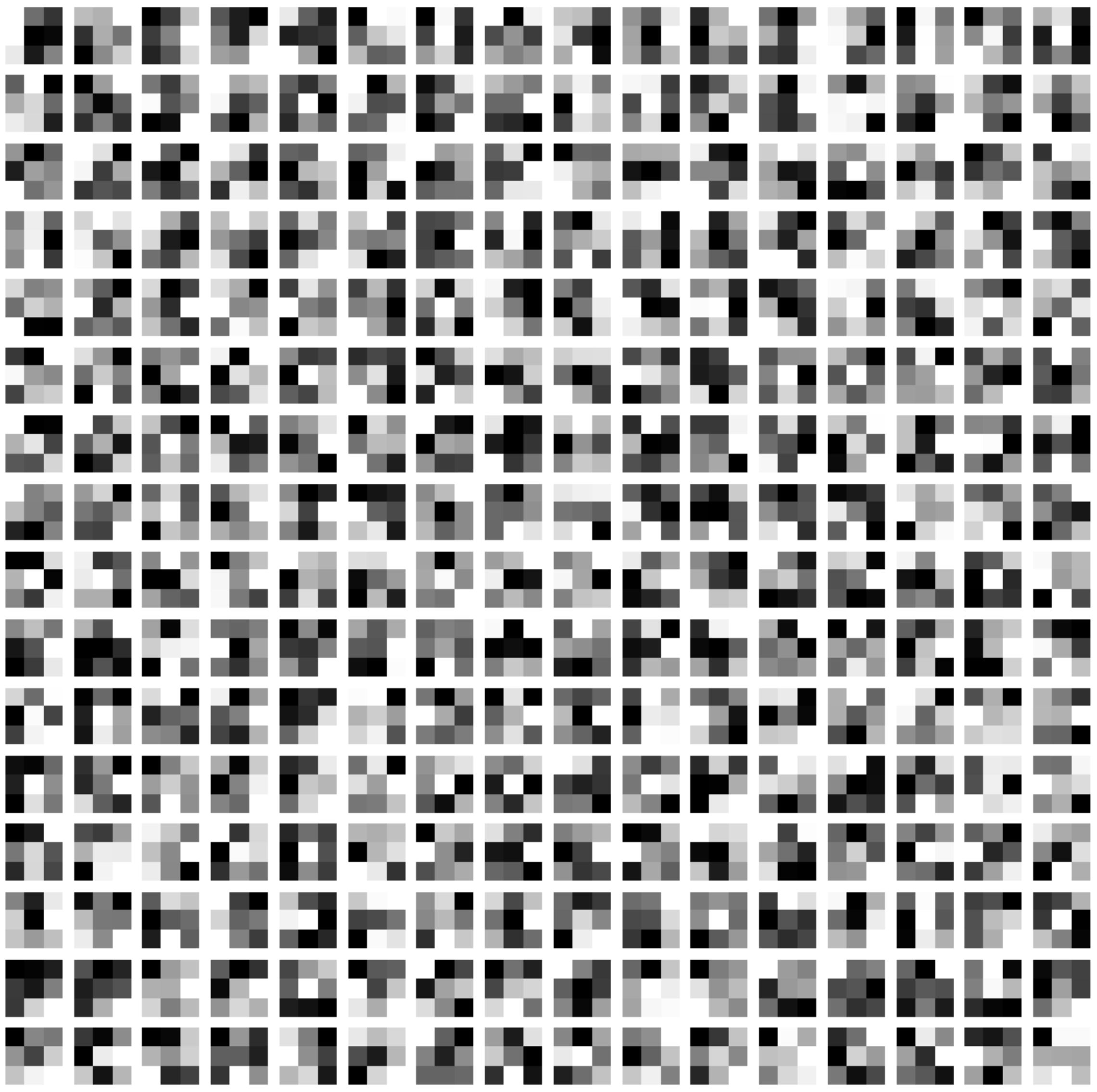
In the first layer we have three input channels (red, green, blue). Thus, filter depth is 3 and we may visualize each filter as color image. Filter pixels may have range different from [0, 1]. Thus, we linearly scale all filters.

```
filters = 1 / (filters.max() - filters.min()) * (filters - filters.min())

fig, axs = plt.subplots(filters.shape[3], 4, figsize=(4, 12))

for row in range(0, filters.shape[3]):
    axs[row, 0].imshow(filters[:, :, :, row], vmin=0, vmax=1)
    axs[row, 0].axis('off')
    axs[row, 1].imshow(filters[:, :, 0, row], cmap='gray', vmin=0, vmax=1)
    axs[row, 1].axis('off')
    axs[row, 2].imshow(filters[:, :, 1, row], cmap='gray', vmin=0, vmax=1)
    axs[row, 2].axis('off')
    axs[row, 3].imshow(filters[:, :, 2, row], cmap='gray', vmin=0, vmax=1)
    axs[row, 3].axis('off')
    if row == 0:
        axs[row, 0].set_title('RGB')
        axs[row, 1].set_title('R')
        axs[row, 2].set_title('G')
        axs[row, 3].set_title('B')

plt.show()
```


RGB
R
G
B

For deeper layers there is no color interpretation, because filters have more than 3 depth levels. So we may visualize a filter as a list of sections perpendicular to the depth axis. In the following plot each row contains the sections of one filter.

```
layer = model.get_layer('conv2')

filters, bias_weights = layer.get_weights()
filters = 1 / (filters.max() - filters.min()) * (filters - filters.min())

fig, axs = plt.subplots(filters.shape[3], filters.shape[2], figsize=(12, 12))

for row in range(0, filters.shape[3]):
    for col in range(0, filters.shape[2]):
        axs[row, col].imshow(filters[:, :, col, row], cmap='gray')
        axs[row, col].axis('off')

plt.show()
```



26.6.3 Maximizing Neuron Activation

To get a better idea of what causes neurons to fire, we may seek for images with high activation of a fixed neuron. This is an optimization problem. The objective is a neuron's activation. The search space is the set of all images fitting the model's input size.

We apply gradient descent to the negative objective (that is, gradient ascent to the objective) and use some Keras features simplifying implementation.

The objective is a neuron's output and we handle the objective as a Keras model. This will allow for using Keras to compute gradients.

```
layer = model.get_layer('conv3')
neuron = (5, 5, 0)
#layer = model.get_layer('dense2')
#neuron = (0, )
#layer = model.get_layer('out')
#neuron = (0, )

submodel = keras.models.Model(inputs=model.inputs, outputs=layer.output[(0, ) +␣
↪neuron])
submodel.summary()
```

```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 128, 128, 3)]     0

 rescaling (Rescaling)       (None, 128, 128, 3)       0

 conv1 (Conv2D)              (None, 126, 126, 16)      448

 conv2 (Conv2D)              (None, 124, 124, 16)      2320

 pool1 (MaxPooling2D)        (None, 62, 62, 16)        0

 conv3 (Conv2D)              (None, 60, 60, 32)        4640

 tf.__operators__.getitem_4   ()                       0
 (SlicingOpLambda)

=================================================================
Total params: 7,408
Trainable params: 7,408
Non-trainable params: 0
_________________________________________________________________
```

Now we define a function which computes objective value and gradient for a given input image. First we call `convert_to_tensor`[508] to convert the image into a `Tensor` object, which fits the model's input dimensions. Then we tell TensorFlow to watch the operations performed on the image while calculating the objective function. From the collected information TensorFlow then can calculate the gradient of the objective function. To watch the flow of the image through the TensorFlow graph we have to create a context manager of type `GradientTape`[509]. The flow of all variables marked for watching with `GradientTape.watch`[510] is recorded for all graph executions inside the `with` block. After executing the graph we get the gradient from `GradientTape.gradient`[511]. Note that calling `Model.predict` does not support watching the variables flow. Instead we have to use a different API variant of Keras: `Model` objects are callable, that is, they can be used as a function, and yield a prediction if called

[508] https://www.tensorflow.org/api_docs/python/tf/convert_to_tensor
[509] https://www.tensorflow.org/api_docs/python/tf/GradientTape
[510] https://www.tensorflow.org/api_docs/python/tf/GradientTape#watch
[511] https://www.tensorflow.org/api_docs/python/tf/GradientTape#gradient

with some input as argument.

```
def get_grad(submodel, img):

    img_tensor = tf.convert_to_tensor(img.reshape(1, img_size, img_size, 3))

    with tf.GradientTape() as tape:
        tape.watch(img_tensor)
        objective_value = submodel(img_tensor)
        grad = tape.gradient(objective_value, img_tensor)

    return objective_value.numpy(), grad.numpy().reshape(img.shape)
```

We are ready for gradient ascent. We are free to choose an arbitrary initial guess, but we have to keep in mind that on the one hand we may end up in a local maximum and on the other hand there might be many global maxima. Thus, the initial guess will have influence on the result. We put everything in a function. So we can reuse it below.

```
def gradient_ascent(submodel, init_img, max_iter, step_length):

    img = init_img

    for i in range(0, max_iter):
        obj, grad = get_grad(submodel, img)

        img = img + step_length * grad

        print(i, obj, np.max(np.abs(grad)))

    return img
```

```
# constant image
img = 128 * np.ones((img_size, img_size, 3), dtype=float)

# photo
#img = keras.preprocessing.image.load_img(data_path + 'unlabeled/365.jpg',
#                                          target_size=(img_size, img_size))
#img = np.asarray(img, dtype=np.float32)
```

```
# parameters for gradient ascent
img = gradient_ascent(submodel, img, 1000, 100) # for conv3/dense2 with constant
#img = gradient_ascent(submodel, img, 100, 100) # for output neuron with photo

# show result
img_to_show = 1 / (img.max() - img.min()) * (img - img.min())
fig, ax = plt.subplots()
ax.imshow(img_to_show)
plt.show()
```

```
0 0.07344329 0.0003321536350995302
1 0.073513456 0.00028220663079991937
2 0.07358889 0.00032136144000010556
3 0.07369768 0.00030644104117527604
4 0.07383166 0.00031520603806711733
5 0.07399155 0.00030644104117527604
6 0.074129954 0.00031513432622887194
7 0.0742565 0.00030454201623797417
8 0.07441732 0.0003239291545469314
9 0.07457861 0.00030454201623797417
10 0.07470061 0.00030454201623797417
```

(continues on next page)

(continued from previous page)

```
11 0.07487879 0.00030454201623797417
12 0.07501966 0.00030454201623797417
13 0.07516515 0.0003058453439734876
14 0.075305745 0.00030454201623797417
15 0.0754803 0.0003058453439734876
16 0.07562959 0.00030454201623797417
17 0.07579061 0.00030454201623797417
18 0.07593748 0.0003058453439734876
19 0.07606852 0.00030454201623797417
20 0.076212205 0.00030454201623797417
21 0.07638178 0.0003369719488546252
22 0.076535545 0.00030454201623797417
23 0.07666714 0.0003113079583272338
24 0.07682478 0.00030454201623797417
25 0.07697314 0.00030454201623797417
26 0.07714298 0.00030454201623797417
27 0.077249855 0.0003259675286244601
28 0.07743089 0.00030454201623797417
29 0.07758624 0.0003369719488546252
30 0.07774404 0.00030454201623797417
31 0.0778713 0.00031524914084002376
32 0.07803682 0.0003058453439734876
33 0.078191474 0.0003058453439734876
34 0.07833725 0.00036290791467763484
35 0.07847955 0.00030454201623797417
36 0.07863803 0.00030454201623797417
37 0.07877792 0.0003113079583272338
38 0.07892162 0.00030454201623797417
39 0.07909869 0.00030454201623797417
40 0.07921913 0.0003113079583272338
41 0.07938554 0.00030454201623797417
42 0.07953825 0.00030454201623797417
43 0.07970761 0.00030454201623797417
44 0.07983406 0.00031524914084002376
45 0.07996544 0.0003058453439734876
46 0.08013553 0.00030454201623797417
47 0.0802946 0.00030454201623797417
48 0.08043927 0.0003058453439734876
49 0.08059702 0.00030454201623797417
50 0.08072535 0.00030454201623797417
51 0.08089805 0.00030454201623797417
52 0.08103597 0.0003369719488546252
53 0.08119964 0.0003058453439734876
54 0.08135997 0.00030454201623797417
55 0.08145313 0.0003259675286244601
56 0.08164934 0.00030454201623797417
57 0.081796795 0.00030454201623797417
58 0.08194393 0.00030454201623797417
59 0.08208322 0.00033827530569396913
60 0.08226341 0.00030454201623797417
61 0.08238993 0.00031524914084002376
62 0.082546435 0.00030454201623797417
63 0.08268981 0.00030454201623797417
64 0.082855605 0.0003113079583272338
65 0.08297971 0.00030454201623797417
66 0.08315368 0.00030454201623797417
67 0.08331067 0.00030454201623797417
68 0.08345787 0.0003048637881875038
69 0.083590515 0.0003035604313481599
70 0.0837581 0.0003232989402022213
71 0.08390607 0.0003113079583272338
```

(continues on next page)

(continued from previous page)

```
72 0.084045365 0.0003247601562179625
73 0.08419961 0.0003035604313481599
74 0.08436007 0.0003035604313481599
75 0.08451763 0.0003048637881875038
76 0.08465027 0.0003060454619117081
77 0.08482045 0.0003035604313481599
78 0.084977776 0.0003035604313481599
79 0.08512163 0.0003035604313481599
80 0.08527139 0.0003035604313481599
81 0.08544078 0.0003048637881875038
82 0.085562915 0.0003035604313481599
83 0.085736886 0.0003048637881875038
84 0.085881226 0.0003035604313481599
85 0.086030394 0.00033729374990798533
86 0.086165085 0.0003035604313481599
87 0.086344495 0.0003197602345608175
88 0.08646924 0.0003035604313481599
89 0.08664642 0.0003035604313481599
90 0.086805105 0.0003035604313481599
91 0.086939305 0.0003035604313481599
92 0.08710596 0.00031557094189338386
93 0.08726247 0.0003035604313481599
94 0.08738602 0.0003035604313481599
95 0.08757089 0.0003035604313481599
96 0.087709084 0.0003048637881875038
97 0.08785117 0.0003172186843585223
98 0.08802833 0.0003197602345608175
99 0.08816219 0.0003035604313481599
100 0.08833459 0.0003359904221724719
101 0.0884886 0.0003035604313481599
102 0.08864398 0.0003048637881875038
103 0.08880227 0.0003035604313481599
104 0.08896902 0.0003048637881875038
105 0.08910107 0.0003035604313481599
106 0.08926667 0.0003359904221724719
107 0.0894378 0.0003035604313481599
108 0.089568794 0.0003048637881875038
109 0.08974953 0.0003172186843585223
110 0.08987595 0.0003048637881875038
111 0.09004665 0.0003035604313481599
112 0.090225294 0.0003035604313481599
113 0.090371795 0.00031557094189338386
114 0.09052053 0.0003035604313481599
115 0.09068331 0.0003035604313481599
116 0.0908356 0.0003259675286244601
117 0.09098149 0.0003172186843585223
118 0.091161326 0.0003035604313481599
119 0.09132437 0.0003035604313481599
120 0.09145064 0.00031426758505403996
121 0.09163647 0.0003049639635719359
122 0.09175947 0.000303660606732592
123 0.09195082 0.000303660606732592
124 0.09209989 0.00037339873961173
125 0.09226224 0.00033609167439863086
126 0.092403606 0.000303660606732592
127 0.092575684 0.0003232989402022213
128 0.092724964 0.0003113079583272338
129 0.0928749 0.000303660606732592
130 0.093041636 0.0003049639635719359
131 0.0932492 0.000548863725271076
132 0.093575545 0.000548863725271076
```

(continues on next page)

(continued from previous page)

```
133 0.09393424 0.000548863725271076
134 0.09426286 0.0005501671112142503
135 0.09457503 0.000548863725271076
136 0.0949109 0.0006987230153754354
137 0.095343046 0.0006967210792936385
138 0.095765874 0.0006967210792936385
139 0.09618598 0.0006980244070291519
140 0.09660856 0.0006967210792936385
141 0.09702786 0.0006980244070291519
142 0.09744506 0.0006967210792936385
143 0.09787216 0.0006964638596400619
144 0.098283805 0.0006977671873755753
145 0.09869732 0.0006964638596400619
146 0.09912459 0.0006964638596400619
147 0.09951783 0.0006964638596400619
148 0.09995559 0.0006964638596400619
149 0.10036949 0.0006977671873755753
150 0.10079682 0.0006964638596400619
151 0.10120723 0.0006964638596400619
152 0.10162692 0.0006964638596400619
153 0.102024704 0.0006964638596400619
154 0.10247612 0.0006964638596400619
155 0.10286892 0.0006964638596400619
156 0.10330826 0.0006964638596400619
157 0.10371728 0.0006977671873755753
158 0.10411946 0.0006964638596400619
159 0.104556054 0.0006964638596400619
160 0.10494343 0.0006964638596400619
161 0.105392754 0.0006964638596400619
162 0.1058107 0.0006977671873755753
163 0.10622219 0.0006964638596400619
164 0.10665828 0.0006964638596400619
165 0.107054524 0.0006964638596400619
166 0.10749701 0.0006964638596400619
167 0.107880875 0.0006977671873755753
168 0.108363695 0.0009242984815500677
169 0.108925685 0.0009242984815500677
170 0.10950871 0.0009242984815500677
171 0.11008029 0.0009256018092855811
172 0.110664986 0.0009242984815500677
173 0.11124636 0.0009242984815500677
174 0.1118204 0.0009242984815500677
175 0.11237122 0.0009242984815500677
176 0.112955004 0.0009256018092855811
177 0.11352999 0.0009242984815500677
178 0.11410865 0.0009242984815500677
179 0.11469558 0.0009242984815500677
180 0.115223646 0.0009256018092855811
181 0.11583601 0.0009242984815500677
182 0.116399184 0.0009242984815500677
183 0.11696331 0.0009242984815500677
184 0.11755805 0.0009256018092855811
185 0.11812425 0.0009242984815500677
186 0.11870559 0.0009242984815500677
187 0.11927338 0.0009256018092855811
188 0.11986539 0.0009256018092855811
189 0.12042022 0.0009242984815500677
190 0.12099801 0.0009152829879894853
191 0.121533066 0.0009165863157249987
192 0.12208147 0.0009152829879894853
193 0.12265298 0.0009165863157249987
```

(continues on next page)

(continued from previous page)

```
194 0.123206116 0.0009152829879894853
195 0.12377263 0.0009152829879894853
196 0.12432422 0.0009152829879894853
197 0.1248651 0.0009165863157249987
198 0.12543273 0.0009152829879894853
199 0.12598741 0.0009165863157249987
200 0.12649682 0.0009165863157249987
201 0.12707181 0.0009152829879894853
202 0.12763208 0.0009152829879894853
203 0.12818763 0.0009152829879894853
204 0.12873638 0.0009152829879894853
205 0.12930945 0.001039297436363995
206 0.13005458 0.0010406007058918476
207 0.13081264 0.001039297436363995
208 0.13159339 0.001039297436363995
209 0.13235143 0.001039297436363995
210 0.13312067 0.0010406007058918476
211 0.13388121 0.001039297436363995
212 0.13465029 0.001039297436363995
213 0.13540941 0.001039297436363995
214 0.13616672 0.001039297436363995
215 0.13694865 0.001039297436363995
216 0.13772929 0.001039297436363995
217 0.13847396 0.001039297436363995
218 0.13922313 0.0010406007058918476
219 0.14000468 0.001039297436363995
220 0.1407558 0.0010269630001857877
221 0.1414311 0.0010269630001857877
222 0.1421513 0.0010269630001857877
223 0.14284176 0.0010282662697136402
224 0.14353868 0.0010282662697136402
225 0.14421485 0.0010269630001857877
226 0.14493802 0.0010269630001857877
227 0.14563352 0.0010269630001857877
228 0.14630292 0.0010269630001857877
229 0.14701118 0.0010282662697136402
230 0.14767927 0.0010269630001857877
231 0.14838807 0.0010282662697136402
232 0.14909413 0.0010269630001857877
233 0.14977466 0.0010269630001857877
234 0.15044625 0.0010269630001857877
235 0.15116742 0.0010282662697136402
236 0.15185341 0.0010453721042722464
237 0.15255716 0.0010453721042722464
238 0.15326595 0.0010466754902154207
239 0.15398586 0.0010466754902154207
240 0.15467824 0.0010453721042722464
241 0.15538774 0.0010453721042722464
242 0.15609169 0.0010466754902154207
243 0.15681161 0.0010334550170227885
244 0.15749998 0.0010334550170227885
245 0.15817234 0.001034758286550641
246 0.1588495 0.0010334550170227885
247 0.15955731 0.001034758286550641
248 0.1602395 0.0010334550170227885
249 0.16091433 0.001034758286550641
250 0.16161926 0.0010334550170227885
251 0.16230682 0.0010334550170227885
252 0.16297057 0.0010334550170227885
253 0.16366643 0.001034758286550641
254 0.16435352 0.0010334550170227885
```

(continues on next page)

(continued from previous page)

```
255 0.16506052 0.0010422063060104847
256 0.1657235 0.0010422063060104847
257 0.16641217 0.0010409029200673103
258 0.16713175 0.0011633450631052256
259 0.16799636 0.0011633450631052256
260 0.16887681 0.0011633450631052256
261 0.16975336 0.0011633450631052256
262 0.17060798 0.0011633450631052256
263 0.1714916 0.0011633450631052256
264 0.1723558 0.0011633450631052256
265 0.17321947 0.0011633450631052256
266 0.17410064 0.0011633450631052256
267 0.17496765 0.0011646484490484
268 0.17584488 0.0011633450631052256
269 0.17672223 0.0011633450631052256
270 0.17758405 0.0011633450631052256
271 0.17844957 0.0011633450631052256
272 0.17929256 0.0011633450631052256
273 0.1801504 0.0011646484490484
274 0.1810208 0.0011633450631052256
275 0.18185535 0.0011633450631052256
276 0.18274412 0.0011633450631052256
277 0.1835914 0.0011633450631052256
278 0.18444327 0.0011633450631052256
279 0.18529923 0.0011646484490484
280 0.18615171 0.0011633450631052256
281 0.18702075 0.0011633450631052256
282 0.18789604 0.0011633450631052256
283 0.18873143 0.0011633450631052256
284 0.18964577 0.0011633450631052256
285 0.19048303 0.0011633450631052256
286 0.19135761 0.0011646484490484
287 0.1922316 0.0011633450631052256
288 0.19306631 0.0011633450631052256
289 0.19395226 0.0011633450631052256
290 0.19479892 0.0011646484490484
291 0.1956524 0.0011633450631052256
292 0.19652039 0.0011633450631052256
293 0.19735864 0.0011646484490484
294 0.1982168 0.0011633450631052256
295 0.19905323 0.0011633450631052256
296 0.19994366 0.0011633450631052256
297 0.20076564 0.0011646484490484
298 0.20162287 0.0011633450631052256
299 0.20249972 0.0011633450631052256
300 0.20334432 0.0011646484490484
301 0.20421043 0.0011633450631052256
302 0.20506933 0.0011633450631052256
303 0.20591384 0.0011633450631052256
304 0.2067788 0.0011633450631052256
305 0.2076214 0.0011630564695224166
306 0.20847076 0.001161753199994564
307 0.20930934 0.0011547609465196729
308 0.21017495 0.0011547609465196729
309 0.2110126 0.0011560643324628472
310 0.21185476 0.0011547609465196729
311 0.21270594 0.0011547609465196729
312 0.21352428 0.0011547609465196729
313 0.21437383 0.0011547609465196729
314 0.21522999 0.0011547609465196729
315 0.21606508 0.0011547609465196729
```

(continues on next page)

(continued from previous page)

```
316 0.21693018 0.0011547609465196729
317 0.21776438 0.0011547609465196729
318 0.21860445 0.0011547609465196729
319 0.21946377 0.0011547609465196729
320 0.22030687 0.0011547609465196729
321 0.22114545 0.0011547609465196729
322 0.22197145 0.0011547609465196729
323 0.22282624 0.0011547609465196729
324 0.22366986 0.0011547609465196729
325 0.22451457 0.0011547609465196729
326 0.22537518 0.0011547609465196729
327 0.22621363 0.0011547609465196729
328 0.22706088 0.0011547609465196729
329 0.22789448 0.0011547609465196729
330 0.22871679 0.0011547609465196729
331 0.2295802 0.0011547609465196729
332 0.23042527 0.0011547609465196729
333 0.23126474 0.0011547609465196729
334 0.23213267 0.0011547609465196729
335 0.23293126 0.0011547609465196729
336 0.23380387 0.00119047611951828
337 0.23467505 0.00119047611951828
338 0.23550901 0.00119047611951828
339 0.23637694 0.0011949328472837806
340 0.23720443 0.0011949328472837806
341 0.23803565 0.0011949328472837806
342 0.23889732 0.0011949328472837806
343 0.23972046 0.0011859633959829807
344 0.24053669 0.0011859633959829807
345 0.24140269 0.0011859633959829807
346 0.2422227 0.0011859633959829807
347 0.24304807 0.0011769563425332308
348 0.24385178 0.0011859633959829807
349 0.24469233 0.0011769563425332308
350 0.2455194 0.0011859633959829807
351 0.24634546 0.0011859633959829807
352 0.2471638 0.0011769563425332308
353 0.24799958 0.0011769563425332308
354 0.24882391 0.0011859633959829807
355 0.24963304 0.0011859633959829807
356 0.25048518 0.0011801046784967184
357 0.25129056 0.0011693075066432357
358 0.25210515 0.0011693075066432357
359 0.25294277 0.0011801046784967184
360 0.25376484 0.0011801046784967184
361 0.2546005 0.0011801046784967184
362 0.2554038 0.0011693075066432357
363 0.2562459 0.0011801046784967184
364 0.2570611 0.0011693075066432357
365 0.2578666 0.0011801046784967184
366 0.25872436 0.0011693075066432357
367 0.25952893 0.0011801046784967184
368 0.2603687 0.0011801046784967184
369 0.26117754 0.0011693075066432357
370 0.26200697 0.0011693075066432357
371 0.26280937 0.0011801046784967184
372 0.2636469 0.0011801046784967184
373 0.2644574 0.0011801046784967184
374 0.26530296 0.0011775860330089927
375 0.26613316 0.0011883832048624754
376 0.26697263 0.0011775860330089927
```

(continues on next page)

(continued from previous page)

```
377 0.26780468 0.0011883832048624754
378 0.26862416 0.0011775860330089927
379 0.26945022 0.0011883832048624754
380 0.27029875 0.0011775860330089927
381 0.27112618 0.0011883832048624754
382 0.2719626 0.0011883832048624754
383 0.27275893 0.0011775860330089927
384 0.27363223 0.0011883832048624754
385 0.27446464 0.0011883832048624754
386 0.27529076 0.0011775860330089927
387 0.2761158 0.0011775860330089927
388 0.27696207 0.0011883832048624754
389 0.2778019 0.0011730468831956387
390 0.27860922 0.001174796256236732
391 0.27945238 0.001174796256236732
392 0.28024793 0.001174796256236732
393 0.28105906 0.001174796256236732
394 0.2819235 0.001174796256236732
395 0.2827406 0.001174796256236732
396 0.2835723 0.001174796256236732
397 0.284405 0.001174796256236732
398 0.28518394 0.0011692551197484136
399 0.2860337 0.0011756159365177155
400 0.286859 0.0011756159365177155
401 0.2876567 0.0011756159365177155
402 0.2884748 0.0011756159365177155
403 0.28927705 0.0011756159365177155
404 0.2900658 0.0011756159365177155
405 0.29090843 0.0011756159365177155
406 0.29168615 0.0011756159365177155
407 0.29249954 0.0011756159365177155
408 0.29331246 0.0011756159365177155
409 0.29412633 0.0011756159365177155
410 0.29493684 0.0011756159365177155
411 0.29575172 0.0011756159365177155
412 0.29655385 0.0011670526582747698
413 0.2974703 0.0011670526582747698
414 0.29838023 0.0011670526582747698
415 0.29926324 0.0011670526582747698
416 0.30021134 0.0011670526582747698
417 0.30111364 0.0011670526582747698
418 0.3020156 0.0011670526582747698
419 0.30295184 0.0011670526582747698
420 0.3038516 0.0011670526582747698
421 0.30474448 0.0011670526582747698
422 0.3056714 0.0011670526582747698
423 0.30658302 0.0011670526582747698
424 0.307517 0.0011670526582747698
425 0.30842844 0.0011670526582747698
426 0.30931112 0.0011670526582747698
427 0.310214 0.0011670526582747698
428 0.31112623 0.0011670526582747698
429 0.3120542 0.0011670526582747698
430 0.31292132 0.0011670526582747698
431 0.31384367 0.0011670526582747698
432 0.31476337 0.0011670526582747698
433 0.31565458 0.0011670526582747698
434 0.3165592 0.0011670526582747698
435 0.3174939 0.0011670526582747698
436 0.31838903 0.0011670526582747698
437 0.31929648 0.0011670526582747698
```

(continues on next page)

(continued from previous page)

```
438 0.32019052 0.0011670526582747698
439 0.321099 0.0011670526582747698
440 0.32200316 0.0011670526582747698
441 0.32291114 0.0011670526582747698
442 0.32379955 0.0011670526582747698
443 0.32469973 0.0011670526582747698
444 0.32562268 0.0011670526582747698
445 0.32650757 0.0011670526582747698
446 0.32741365 0.0011670526582747698
447 0.3283232 0.0011670526582747698
448 0.3292402 0.0011670526582747698
449 0.330129 0.0011670526582747698
450 0.33103305 0.0011670526582747698
451 0.3319336 0.0011670526582747698
452 0.3328155 0.0011670526582747698
453 0.33374733 0.0011670526582747698
454 0.33464786 0.0011670526582747698
455 0.33553892 0.0011670526582747698
456 0.3364692 0.0011670526582747698
457 0.33737844 0.0011670526582747698
458 0.3382727 0.0011670526582747698
459 0.3391565 0.0011670526582747698
460 0.34008282 0.0011670526582747698
461 0.3409896 0.0011670526582747698
462 0.34186038 0.0011670526582747698
463 0.34278762 0.0011670526582747698
464 0.3436863 0.0011670526582747698
465 0.34458283 0.0011670526582747698
466 0.34549478 0.0010790652595460415
467 0.34631377 0.0010790652595460415
468 0.34719032 0.0010790652595460415
469 0.3480214 0.0010790652595460415
470 0.34890258 0.0010790652595460415
471 0.3497575 0.0010790652595460415
472 0.3505807 0.0010790652595460415
473 0.3514599 0.0010790652595460415
474 0.3522837 0.0010790652595460415
475 0.35314745 0.0010790652595460415
476 0.35401186 0.0010790652595460415
477 0.35486692 0.0010790652595460415
478 0.35568154 0.0010790652595460415
479 0.35657242 0.0010790652595460415
480 0.35737127 0.0010154407937079668
481 0.3581802 0.0010154407937079668
482 0.35896868 0.0010154407937079668
483 0.35978597 0.0010154407937079668
484 0.36058876 0.0010154407937079668
485 0.36137655 0.0010154407937079668
486 0.36217743 0.0010154407937079668
487 0.36296797 0.0010154407937079668
488 0.3637755 0.0010154407937079668
489 0.36456716 0.0010154407937079668
490 0.36538386 0.0010154407937079668
491 0.36614984 0.0010154407937079668
492 0.36697516 0.0010154407937079668
493 0.36775056 0.0010154407937079668
494 0.36854511 0.0010154407937079668
495 0.3693443 0.0010154407937079668
496 0.37014896 0.0010154407937079668
497 0.37095195 0.0010154407937079668
498 0.37172404 0.0010154407937079668
```

(continues on next page)

(continued from previous page)

```
499 0.37254268 0.0010154407937079668
500 0.37333837 0.0010154407937079668
501 0.37415114 0.0010154407937079668
502 0.37494037 0.0010154407937079668
503 0.3757527 0.0010154407937079668
504 0.3765402 0.0010154407937079668
505 0.37734315 0.0010154407937079668
506 0.37814957 0.0010154407937079668
507 0.3789367 0.0010154407937079668
508 0.3797198 0.0010154407937079668
509 0.38052315 0.0010154407937079668
510 0.3813415 0.0010154407937079668
511 0.38213024 0.0010154407937079668
512 0.38292485 0.0010154407937079668
513 0.38370124 0.0010154407937079668
514 0.3845234 0.0010154407937079668
515 0.38530743 0.0010154407937079668
516 0.38611794 0.0010154407937079668
517 0.3869216 0.0010154407937079668
518 0.3876673 0.0010154407937079668
519 0.38849553 0.0010154407937079668
520 0.3892975 0.0010154407937079668
521 0.39006588 0.0010154407937079668
522 0.39090362 0.0010154407937079668
523 0.39167362 0.0010154407937079668
524 0.39247113 0.0010154407937079668
525 0.3932494 0.0010154407937079668
526 0.3940714 0.0010154407937079668
527 0.394863 0.0010154407937079668
528 0.3956417 0.0010154407937079668
529 0.3964413 0.0010154407937079668
530 0.39725867 0.0010154407937079668
531 0.398041 0.0010154407937079668
532 0.39885044 0.0010154407937079668
533 0.3996294 0.0010154407937079668
534 0.40043512 0.0010154407937079668
535 0.40123594 0.0010154407937079668
536 0.40202937 0.0010154407937079668
537 0.402824 0.0010154407937079668
538 0.40364966 0.0010154407937079668
539 0.40445825 0.0010154407937079668
540 0.40526012 0.0010154407937079668
541 0.40606946 0.0010154407937079668
542 0.40684387 0.0010154407937079668
543 0.4076617 0.0010154407937079668
544 0.40847754 0.0010154407937079668
545 0.40928942 0.0010154407937079668
546 0.41008174 0.0010154407937079668
547 0.41090408 0.0010154407937079668
548 0.41169664 0.0010154407937079668
549 0.41249254 0.0010154407937079668
550 0.41331044 0.0010154407937079668
551 0.41412392 0.00101051339879632
552 0.41491395 0.00101051339879632
553 0.41570634 0.00101051339879632
554 0.4165171 0.00101051339879632
555 0.4173471 0.00101051339879632
556 0.41813064 0.00101051339879632
557 0.4189338 0.00101051339879632
558 0.41971493 0.00101051339879632
559 0.42053598 0.00101051339879632
```

(continues on next page)

(continued from previous page)

```
560 0.4213166 0.00101051339879632
561 0.42214185 0.00101051339879632
562 0.42296207 0.00101051339879632
563 0.42373446 0.00101051339879632
564 0.4245199 0.00101051339879632
565 0.42533642 0.00101051339879632
566 0.4261579 0.00101051339879632
567 0.42693517 0.00101051339879632
568 0.42773804 0.0010139307705685496
569 0.42855272 0.0010139307705685496
570 0.42936152 0.0010139307705685496
571 0.43015447 0.0010139307705685496
572 0.4309811 0.0010139307705685496
573 0.43175057 0.0010139307705685496
574 0.43258232 0.0010139307705685496
575 0.43339017 0.0010139307705685496
576 0.4341939 0.0010139307705685496
577 0.43496978 0.0010139307705685496
578 0.4357804 0.0010139307705685496
579 0.4365905 0.0009832626674324274
580 0.4373628 0.0009832626674324274
581 0.43810424 0.0009832626674324274
582 0.4388808 0.0009832626674324274
583 0.4396517 0.0009832626674324274
584 0.44040602 0.0009832626674324274
585 0.441188 0.0009832626674324274
586 0.44193944 0.0009832626674324274
587 0.44268337 0.0009832626674324274
588 0.44346732 0.0009832626674324274
589 0.4442556 0.0009832626674324274
590 0.4449988 0.0009832626674324274
591 0.44578514 0.0009832626674324274
592 0.44653383 0.0009832626674324274
593 0.44730932 0.0009832626674324274
594 0.44806883 0.0009832626674324274
595 0.44879472 0.0009832626674324274
596 0.4495962 0.0009832626674324274
597 0.45036265 0.0009832626674324274
598 0.4511082 0.0009832626674324274
599 0.45191392 0.0009832626674324274
600 0.45263118 0.0009832626674324274
601 0.45340177 0.0009832626674324274
602 0.45418617 0.0009832626674324274
603 0.45495698 0.0009832626674324274
604 0.4556701 0.0009832626674324274
605 0.45644075 0.0009832626674324274
606 0.45722923 0.0009832626674324274
607 0.4580336 0.0010870908154174685
608 0.4589291 0.0010870908154174685
609 0.45981702 0.001059028902091086
610 0.46067777 0.001059028902091086
611 0.461596 0.001059028902091086
612 0.46244848 0.001059028902091086
613 0.46335196 0.001059028902091086
614 0.4642234 0.001059028902091086
615 0.4651056 0.001059028902091086
616 0.46600258 0.001059028902091086
617 0.46684855 0.001059028902091086
618 0.46774 0.001059028902091086
619 0.4686332 0.001059028902091086
620 0.46951178 0.001059028902091086
```

(continues on next page)

(continued from previous page)

```
621 0.47038 0.000985809019766748
622 0.47121063 0.0010329419746994972
623 0.47214574 0.0010329419746994972
624 0.47310114 0.0010329419746994972
625 0.47406873 0.0010329419746994972
626 0.47502652 0.0010329419746994972
627 0.47594658 0.0010326988995075226
628 0.47688997 0.0010326988995075226
629 0.47786474 0.0010326988995075226
630 0.4788167 0.0010326988995075226
631 0.4797461 0.0010326988995075226
632 0.48066002 0.0010326988995075226
633 0.4816415 0.0010326988995075226
634 0.4825965 0.0010326988995075226
635 0.4835614 0.0010326988995075226
636 0.4844857 0.0010326988995075226
637 0.48542342 0.0010326988995075226
638 0.48639002 0.0010326988995075226
639 0.4872793 0.0010326988995075226
640 0.48816916 0.0010326988995075226
641 0.4891415 0.0010326988995075226
642 0.4900629 0.0010326988995075226
643 0.4910013 0.0010326988995075226
644 0.49189064 0.0010326988995075226
645 0.49282485 0.0010326988995075226
646 0.49373966 0.0010326988995075226
647 0.49464673 0.0010326988995075226
648 0.49556008 0.0010326988995075226
649 0.49648935 0.0010326988995075226
650 0.49737707 0.0010326988995075226
651 0.49832752 0.0010326988995075226
652 0.49926782 0.0010326988995075226
653 0.5001807 0.0010326988995075226
654 0.5010698 0.0010326988995075226
655 0.5020299 0.0010326988995075226
656 0.5029193 0.0010326988995075226
657 0.5038766 0.0010326988995075226
658 0.50479966 0.0010326988995075226
659 0.5056684 0.0010326988995075226
660 0.50657624 0.0010326988995075226
661 0.5075078 0.0010326988995075226
662 0.50838625 0.0010326988995075226
663 0.5092764 0.0010326988995075226
664 0.51016474 0.0010326988995075226
665 0.51108813 0.0010326988995075226
666 0.51197284 0.0010326988995075226
667 0.51285535 0.0010326988995075226
668 0.51377314 0.0010326988995075226
669 0.5146683 0.0010326988995075226
670 0.51558805 0.0010326988995075226
671 0.5164205 0.0010326988995075226
672 0.51737213 0.0010326988995075226
673 0.5182538 0.0010326988995075226
674 0.51913476 0.0010326988995075226
675 0.52007437 0.0010326988995075226
676 0.5209829 0.0010326988995075226
677 0.52187335 0.0010326988995075226
678 0.5227336 0.0010326988995075226
679 0.5236664 0.0010326988995075226
680 0.52454185 0.0010326988995075226
681 0.52543604 0.0010326988995075226
```

(continues on next page)

(continued from previous page)

```
682 0.5263518 0.0010326988995075226
683 0.5271962 0.0010326988995075226
684 0.52804476 0.0010326988995075226
685 0.5289265 0.0010326988995075226
686 0.52977735 0.0010326988995075226
687 0.53062344 0.0010326988995075226
688 0.5314872 0.0010326988995075226
689 0.53233325 0.0010326988995075226
690 0.5331717 0.0010326988995075226
691 0.5340625 0.0010326988995075226
692 0.5349046 0.0010326988995075226
693 0.5357088 0.0010326988995075226
694 0.5365881 0.0010326988995075226
695 0.53747183 0.0010326988995075226
696 0.5382971 0.0010326988995075226
697 0.5391491 0.0010307517368346453
698 0.5400001 0.0010307517368346453
699 0.54090506 0.0010307517368346453
700 0.5417581 0.0010307517368346453
701 0.5425821 0.0010307517368346453
702 0.5434768 0.0010307517368346453
703 0.5442994 0.0010307517368346453
704 0.545164 0.0010307517368346453
705 0.5460554 0.0010307517368346453
706 0.54692215 0.0010307517368346453
707 0.5477654 0.0010307517368346453
708 0.54859924 0.0010307517368346453
709 0.5495042 0.0010307517368346453
710 0.5503597 0.0010307517368346453
711 0.55117035 0.0010307517368346453
712 0.55207765 0.0010307517368346453
713 0.5529067 0.0010238096583634615
714 0.5537985 0.0010238096583634615
715 0.5546446 0.0009919860167428851
716 0.55550605 0.0009919860167428851
717 0.5563586 0.0009919860167428851
718 0.55718243 0.0009919860167428851
719 0.55800784 0.0010421713814139366
720 0.5589189 0.0010421713814139366
721 0.559784 0.001022367156110704
722 0.560643 0.001022367156110704
723 0.5615605 0.001022367156110704
724 0.5624156 0.001022367156110704
725 0.5632752 0.001022367156110704
726 0.5641829 0.001022367156110704
727 0.56504005 0.001022367156110704
728 0.5659065 0.0010457722237333655
729 0.5667304 0.000956058909650892
730 0.5675022 0.000956058909650892
731 0.5683356 0.000956058909650892
732 0.56912684 0.000956058909650892
733 0.5699113 0.0009668168495409191
734 0.5706983 0.0009668168495409191
735 0.5714856 0.0009668168495409191
736 0.57226485 0.0009668168495409191
737 0.5730624 0.0009668168495409191
738 0.57386065 0.0009668168495409191
739 0.57462347 0.0009668168495409191
740 0.57542187 0.0009668168495409191
741 0.57620674 0.0009668168495409191
742 0.5769996 0.0009668168495409191
```

(continues on next page)

(continued from previous page)

```
743 0.57778823 0.0009668168495409191
744 0.5785791 0.0009668168495409191
745 0.5793726 0.0009668168495409191
746 0.5801276 0.0009668168495409191
747 0.58092684 0.0009668168495409191
748 0.5817179 0.0009668168495409191
749 0.5825163 0.0009668168495409191
750 0.5832929 0.0009668168495409191
751 0.5840812 0.0009668168495409191
752 0.5848595 0.0009668168495409191
753 0.58565956 0.0009668168495409191
754 0.5864584 0.0009668168495409191
755 0.58722854 0.0009668168495409191
756 0.5880059 0.0009668168495409191
757 0.58879775 0.0009668168495409191
758 0.5895954 0.0009668168495409191
759 0.59037656 0.0009668168495409191
760 0.5911552 0.0009668168495409191
761 0.59193975 0.0009668168495409191
762 0.5927428 0.0009668168495409191
763 0.59353584 0.0009668168495409191
764 0.5942976 0.0009668168495409191
765 0.5951029 0.0009668168495409191
766 0.5958876 0.0009668168495409191
767 0.59666055 0.0009668168495409191
768 0.5974607 0.0009668168495409191
769 0.5982454 0.0009668168495409191
770 0.599039 0.0009668168495409191
771 0.59982723 0.0009668168495409191
772 0.6006173 0.0009668168495409191
773 0.60140604 0.0009668168495409191
774 0.60219723 0.0009668168495409191
775 0.6029881 0.0009668168495409191
776 0.6037439 0.0009668168495409191
777 0.60455 0.0009668168495409191
778 0.6053583 0.0009668168495409191
779 0.60611725 0.0009668168495409191
780 0.6069193 0.0009668168495409191
781 0.6077134 0.0009668168495409191
782 0.6084981 0.0009668168495409191
783 0.6092804 0.0009668168495409191
784 0.61007494 0.0009668168495409191
785 0.61085117 0.0009668168495409191
786 0.6116433 0.0009668168495409191
787 0.61245275 0.0009668168495409191
788 0.6132225 0.0009668168495409191
789 0.61400676 0.0009886898333206773
790 0.61478317 0.0009886898333206773
791 0.6155784 0.0009886898333206773
792 0.6163532 0.0009886898333206773
793 0.6171217 0.0009886898333206773
794 0.6179187 0.0009886898333206773
795 0.61868626 0.0009886898333206773
796 0.6194822 0.0009886898333206773
797 0.6202694 0.0009886898333206773
798 0.62104636 0.0009886898333206773
799 0.6218326 0.0009886898333206773
800 0.6226178 0.0009886898333206773
801 0.62339514 0.0009886898333206773
802 0.6241851 0.0009886898333206773
803 0.62495697 0.0009886898333206773
```

(continues on next page)

(continued from previous page)

```
804 0.6257293 0.0009886898333206773
805 0.6264954 0.0009886898333206773
806 0.62730217 0.0009886898333206773
807 0.62808347 0.0009886898333206773
808 0.628863 0.0009886898333206773
809 0.6296295 0.0009886898333206773
810 0.6304254 0.0009886898333206773
811 0.6312075 0.0009886898333206773
812 0.6320016 0.0009886898333206773
813 0.63276213 0.0009979268070310354
814 0.633546 0.0009979268070310354
815 0.6343409 0.0009979268070310354
816 0.63511616 0.0009979268070310354
817 0.63590056 0.0009979268070310354
818 0.63669586 0.0009979268070310354
819 0.6374912 0.001023309538140893
820 0.6383158 0.001023309538140893
821 0.6391846 0.001023309538140893
822 0.6400445 0.001023309538140893
823 0.6408873 0.001023309538140893
824 0.6417466 0.001023309538140893
825 0.6425912 0.001023309538140893
826 0.643449 0.001023309538140893
827 0.6442834 0.0009945675265043974
828 0.6451267 0.0009945675265043974
829 0.64594156 0.0009945675265043974
830 0.6467531 0.0009945675265043974
831 0.6476129 0.0009945675265043974
832 0.6484228 0.0009945675265043974
833 0.6492394 0.0009945675265043974
834 0.65007967 0.0009945675265043974
835 0.6508905 0.0009945675265043974
836 0.6517256 0.0009945675265043974
837 0.6525289 0.0009945675265043974
838 0.65336 0.0009945675265043974
839 0.65417266 0.0009945675265043974
840 0.6549925 0.0009945675265043974
841 0.6558189 0.0009945675265043974
842 0.6566242 0.0009945675265043974
843 0.6574596 0.0009945675265043974
844 0.6582777 0.0009945675265043974
845 0.6590987 0.0009945675265043974
846 0.65990216 0.0009945675265043974
847 0.66071415 0.0009945675265043974
848 0.6615644 0.0009945675265043974
849 0.6623605 0.0009945675265043974
850 0.6631765 0.0009945675265043974
851 0.66401565 0.0009945675265043974
852 0.6648319 0.0009945675265043974
853 0.665637 0.0009945675265043974
854 0.66644573 0.0009945675265043974
855 0.6672871 0.0009945675265043974
856 0.66809356 0.0009945675265043974
857 0.66891545 0.0009945675265043974
858 0.6697405 0.0009945675265043974
859 0.6705429 0.0009945675265043974
860 0.6713725 0.0009945675265043974
861 0.6721709 0.0009945675265043974
862 0.67298466 0.0009945675265043974
863 0.67382187 0.0009945675265043974
864 0.6746325 0.0009945675265043974
```

(continues on next page)

(continued from previous page)

```
865 0.6754544 0.0009945675265043974
866 0.67626846 0.0009945675265043974
867 0.67705846 0.0009945675265043974
868 0.67789626 0.0009945675265043974
869 0.67871684 0.0009945675265043974
870 0.6795344 0.0009945675265043974
871 0.6803413 0.0009945675265043974
872 0.6811339 0.0009945675265043974
873 0.6819754 0.0009945675265043974
874 0.6827903 0.0009903388563543558
875 0.6835876 0.0009903388563543558
876 0.6843899 0.0009995902655646205
877 0.68519706 0.0009995902655646205
878 0.6860052 0.0009995902655646205
879 0.68679684 0.0009995902655646205
880 0.68760914 0.0009995902655646205
881 0.6884222 0.0009995902655646205
882 0.6892284 0.0009995902655646205
883 0.6900231 0.0009995902655646205
884 0.6908194 0.0009995902655646205
885 0.6916016 0.0009995902655646205
886 0.6924194 0.0009995902655646205
887 0.693211 0.0009995902655646205
888 0.6940029 0.0009995902655646205
889 0.6947848 0.0009995902655646205
890 0.6955754 0.0009995902655646205
891 0.69638306 0.0009995902655646205
892 0.69717735 0.0009995902655646205
893 0.697969 0.0009995902655646205
894 0.69876575 0.0009995902655646205
895 0.6995406 0.0009995902655646205
896 0.7003432 0.0009995902655646205
897 0.70114625 0.0009995902655646205
898 0.70194095 0.0009995902655646205
899 0.70272666 0.0009995902655646205
900 0.7035292 0.0009995902655646205
901 0.7043215 0.0009995902655646205
902 0.7050826 0.0009995902655646205
903 0.70590913 0.0009995902655646205
904 0.7066984 0.0009995902655646205
905 0.7074973 0.0009995902655646205
906 0.7082788 0.0009995902655646205
907 0.7090724 0.0009995902655646205
908 0.7098745 0.0009995902655646205
909 0.710666 0.0009995902655646205
910 0.7114511 0.0009995902655646205
911 0.7122621 0.0009995902655646205
912 0.7130417 0.0009995902655646205
913 0.71384054 0.0009995902655646205
914 0.7146361 0.0009995902655646205
915 0.7154125 0.0009995902655646205
916 0.7162201 0.0009995902655646205
917 0.7170155 0.001052268315106303
918 0.71787304 0.001052268315106303
919 0.7187354 0.001052268315106303
920 0.7195928 0.001052268315106303
921 0.72045165 0.001052268315106303
922 0.7212997 0.001052268315106303
923 0.7221617 0.001052268315106303
924 0.72301835 0.001052268315106303
925 0.72386307 0.001052268315106303
```

(continues on next page)

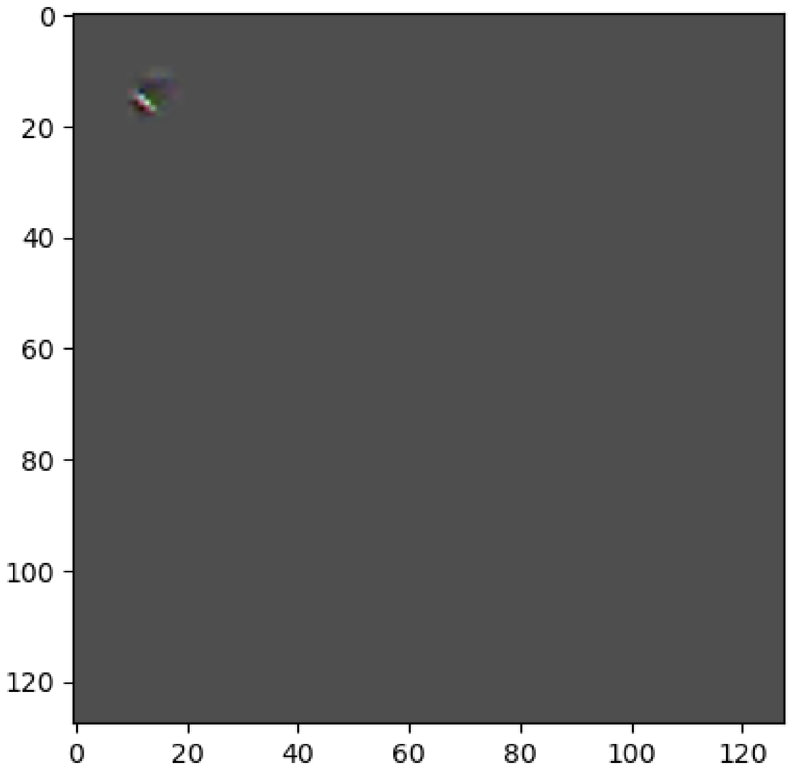
(continued from previous page)

```
926 0.7247229 0.0010522683151066303
927 0.7255807 0.0010522683151066303
928 0.7264381 0.0010522683151066303
929 0.7272903 0.0010522683151066303
930 0.7281472 0.0010522683151066303
931 0.7290011 0.0010522683151066303
932 0.72984874 0.0010522683151066303
933 0.7307101 0.0010522683151066303
934 0.7315547 0.0010522683151066303
935 0.73242086 0.0010522683151066303
936 0.73327273 0.0010522683151066303
937 0.7341252 0.0010522683151066303
938 0.7349861 0.0010522683151066303
939 0.7358517 0.0010522683151066303
940 0.73670053 0.0010522683151066303
941 0.73756105 0.0010522683151066303
942 0.7384129 0.0010522683151066303
943 0.73925865 0.0010522683151066303
944 0.74012375 0.0010522683151066303
945 0.74095243 0.0009738617809489369
946 0.74173504 0.0009738617809489369
947 0.7424955 0.0009738617809489369
948 0.7432958 0.0009738617809489369
949 0.7440823 0.0009738617809489369
950 0.74485135 0.0009738617809489369
951 0.7456361 0.0009738617809489369
952 0.7464003 0.0009509164374321699
953 0.74717706 0.0009738617809489369
954 0.74796754 0.0009738617809489369
955 0.7487558 0.0009738617809489369
956 0.7495361 0.0009738617809489369
957 0.7503084 0.0009738617809489369
958 0.75110865 0.0009738617809489369
959 0.7518831 0.0009738617809489369
960 0.7526496 0.0009509164374321699
961 0.7534354 0.0009738617809489369
962 0.7542203 0.0009738617809489369
963 0.7550154 0.0009738617809489369
964 0.7557916 0.0009738617809489369
965 0.7565614 0.0009738617809489369
966 0.7573651 0.0009738617809489369
967 0.75814277 0.0009738617809489369
968 0.75891125 0.0009509164374321699
969 0.7597001 0.0009738617809489369
970 0.76046485 0.0009738617809489369
971 0.761245 0.0009738617809489369
972 0.76204234 0.0009672498563304543
973 0.7628137 0.0009672498563304543
974 0.7635646 0.0009460232686251402
975 0.7643449 0.0009460232686251402
976 0.76511234 0.0009460232686251402
977 0.7658671 0.0009460232686251402
978 0.7666363 0.0009460232686251402
979 0.76740277 0.0009460232686251402
980 0.7681641 0.0009460232686251402
981 0.768928 0.0009455836843699217
982 0.76968527 0.0009460232686251402
983 0.77044827 0.0009460232686251402
984 0.77121806 0.0009460232686251402
985 0.771972 0.0009067181963473558
986 0.77267444 0.0009067181963473558
```

(continues on next page)

(continued from previous page)

```
987 0.7733834 0.0009067181963473558
988 0.7740967 0.0009067181963473558
989 0.7748062 0.0009067181963473558
990 0.7755114 0.0009067181963473558
991 0.77621996 0.0009067181963473558
992 0.7769272 0.0009067181963473558
993 0.77763164 0.0009067181963473558
994 0.7783504 0.0009067181963473558
995 0.7790618 0.0009067181963473558
996 0.77976304 0.0009067181963473558
997 0.78046834 0.0009067181963473558
998 0.7811791 0.0009067181963473558
999 0.7818845 0.0009067181963473558
```



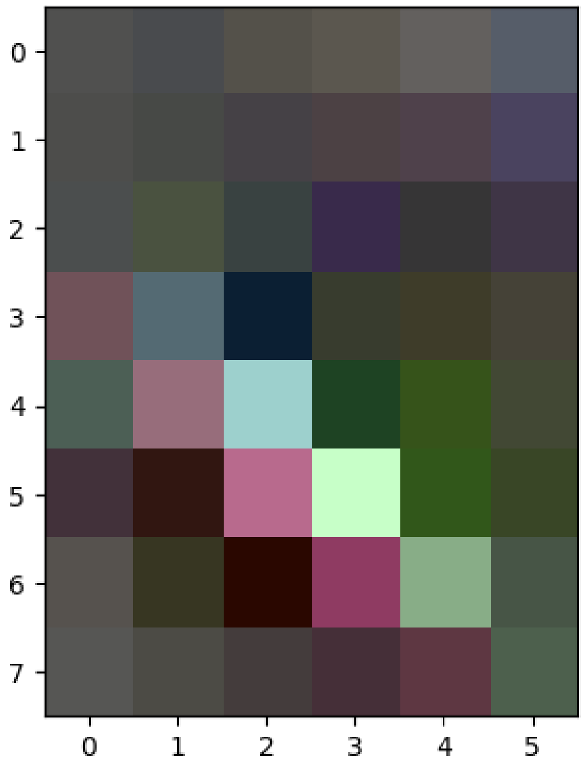
If we maximize the output of a neuron in a convolutional layer, then the result will differ from the initial guess only in the region the neuron is connected to. All other pixels have no influence on the neuron's output. Thus, corresponding components of the gradient are zero in each iteration. To see the details we crop the image. For neurons in the first convolution layer, the maximizing input is the corresponding filter.

```
# mask pixels to keep when cropping
mask_r = np.abs(img_to_show[:, :, 0] - img_to_show[-1, -1, 0]) > 0.09
mask_g = np.abs(img_to_show[:, :, 1] - img_to_show[-1, -1, 1]) > 0.09
mask_b = np.abs(img_to_show[:, :, 2] - img_to_show[-1, -1, 2]) > 0.09
mask = np.logical_or(mask_r, np.logical_or(mask_g, mask_b))

# get active columns
col_mask = mask.any(0)
bb_col_start = col_mask.argmax()
bb_col_end = img_to_show.shape[1] - 1 - col_mask[::-1].argmax()

# get active rows
row_mask = mask.any(1)
```

(continues on next page)

(continued from previous page)

```
bb_row_start = row_mask.argmax()
bb_row_end = img_to_show.shape[0] - 1 - row_mask[::-1].argmax()

# crop image to bounding box
bb_img = img_to_show[bb_row_start:(bb_row_end + 1), bb_col_start:(bb_col_end + 1)]

# show cropped image
fig, ax = plt.subplots()
ax.imshow(bb_img, cmap='gray')
plt.show()
```



Maximizing the output of the first output neuron modifies the initial guess to yield output 1 (the maximum value of sigmoid activation function). That is, we obtain an image the net regards as a cat. Starting with a plain image we get some artistic images. Starting with a photo of a dog we get a slightly blurred dog, which the net labels as cat. By modifying images that way CNNs can be fooled. The CNN ‘sees’ a very different thing than a human.

```
pred = model.predict(img.reshape(1, *img.shape))[0]
print('cat: {:.4f}, dog: {:.4f}'.format(pred[0], pred[1]))
```

```
1/1 [==============================] - 0s 24ms/step
cat: 0.9065, dog: 0.2174
```

The idea of searching for output maximizing inputs is known as *dreaming*. Google’s DeepDream[512] from 2015 uses the techniques discussed above. A similar application of dreaming CNNs is neural style transfer[513], also appearing in 2015.

[512] https://en.wikipedia.org/wiki/DeepDream
[513] https://en.wikipedia.org/wiki/Neural_Style_Transfer

26.6.4 Maximizing Feature Maps

Instead of maximizing single neuron outputs we could look for feature maps having high values in all components or at least high mean (the latter is easier to differentiate). An input image that maximizes a feature map would show a pattern that is tightly connected to the corresponding filter.

```
layer = model.get_layer('conv4')
fmap_index = 2

submodel = keras.models.Model(inputs=model.inputs,
                              outputs=tf.math.reduce_mean(layer.output[0, :, :,␣
↪fmap_index]))


img = 255 * np.random.default_rng(0).normal(0.5, 0.2, size=(img_size, img_size,␣
↪3))

# parameters for gradient ascent
img = gradient_ascent(submodel, img, 1000, 1000000)

# show result
img_to_show = 1 / (img.max() - img.min()) * (img - img.min())
fig, ax = plt.subplots()
ax.imshow(img_to_show)
plt.show()
```

```
0 0.010571016 2.397079242655309e-06
1 0.012987887 2.784569460345665e-06
2 0.01555704 2.6928230454359436e-06
3 0.018290414 2.2864280708745355e-06
4 0.021174435 2.192614147134009e-06
5 0.024220329 2.4407656837865943e-06
6 0.027513845 2.4178386865969514e-06
7 0.03095166 2.223908268206287e-06
8 0.03454724 2.0893653527309652e-06
9 0.038174365 2.028772314588423e-06
10 0.041862242 2.097629021591274e-06
11 0.045696992 2.3129020974010928e-06
12 0.049698126 2.385100970059284e-06
13 0.05387558 2.3095196866051992e-06
14 0.058383714 2.396672925897292e-06
15 0.063163 2.3852505819377257e-06
16 0.068157524 2.510322474336135e-06
17 0.07321679 2.4331638996955007e-06
18 0.07840866 2.3761479042150313e-06
19 0.08379156 2.492263092790381e-06
20 0.089350894 2.4447754185530357e-06
21 0.09505746 2.397075149929151e-06
22 0.10108615 2.545427378208842e-06
23 0.10727019 2.3443969894287875e-06
24 0.11355108 2.2837014057586202e-06
25 0.11987263 2.431382426948403e-06
26 0.12621516 2.6055895432364196e-06
27 0.13275892 2.548260454204865e-06
28 0.13940799 2.536550937293214e-06
29 0.14614432 2.5595429633540334e-06
30 0.15306608 2.4765708985796664e-06
31 0.1601255 2.4590774501120904e-06
32 0.16713977 2.3455713744624518e-06
33 0.17422041 2.3978557237569476e-06
34 0.18140693 2.474193479429232e-06
```

(continues on next page)

(continued from previous page)

```
35 0.1886376 2.5417300548724597e-06
36 0.19593534 2.533454107833677e-06
37 0.20329633 2.704197186176316e-06
38 0.21070997 2.6751181394502055e-06
39 0.21831936 2.7035023322241614e-06
40 0.2259554 2.7090159164799843e-06
41 0.23365016 2.4669375306984875e-06
42 0.24140108 2.3419049739459297e-06
43 0.24929094 2.5573299353709444e-06
44 0.25731435 2.5564454517734703e-06
45 0.26534614 2.4323860543518094e-06
46 0.2732757 2.296659886269481e-06
47 0.28120157 2.4815644792397507e-06
48 0.28913957 2.3013376448943745e-06
49 0.29706028 2.140596961908159e-06
50 0.30498374 1.9981957848358434e-06
51 0.3129182 2.2039150735508883e-06
52 0.32090297 2.156832579203183e-06
53 0.32904193 2.1195842236920726e-06
54 0.33714753 2.443357743686647e-06
55 0.34530818 2.8032877708028536e-06
56 0.3534469 2.4665589535288746e-06
57 0.36151606 2.3440138647856656e-06
58 0.3696367 2.518916517146863e-06
59 0.37779084 2.2818105662736343e-06
60 0.38606593 2.8909905722684925e-06
61 0.39434585 2.2529472971655196e-06
62 0.4026392 2.309225692442851e-06
63 0.4110172 2.1857999854546506e-06
64 0.41925484 2.1461057713167975e-06
65 0.42749077 2.2192639335116837e-06
66 0.4357061 2.193360160163138e-06
67 0.44399276 2.109282604578766e-06
68 0.45219928 2.2869414806336863e-06
69 0.46037853 2.239397417724831e-06
70 0.4686118 2.8001759346807376e-06
71 0.47689474 2.122434580087429e-06
72 0.48513776 2.1441801436594687e-06
73 0.4933584 2.334981900276034e-06
74 0.5015519 2.120641738656559e-06
75 0.50966257 2.1056046080047963e-06
76 0.5177873 2.0936017790518235e-06
77 0.525847 2.438975798213505e-06
78 0.53396064 2.3880647859186865e-06
79 0.5421345 2.4256362394226016e-06
80 0.5503483 2.482177706042421e-06
81 0.55855685 2.787142193483305e-06
82 0.56671786 3.04928016703343e-06
83 0.5748916 2.9708369311265415e-06
84 0.5830835 2.8886945528938668e-06
85 0.5912361 2.8094241315557156e-06
86 0.59929144 2.7368155315343756e-06
87 0.60734797 2.5096640001720516e-06
88 0.6153681 2.514457037250395e-06
89 0.6233643 2.5291581096098525e-06
90 0.6313867 2.600254674689495e-06
91 0.6394314 2.527786591599579e-06
92 0.64746135 2.4455698621750344e-06
93 0.6554675 2.67075483861845e-06
94 0.6634189 2.3975730982783716e-06
95 0.67137367 2.2373308183887275e-06
```

(continues on next page)

(continued from previous page)

```
96 0.6793155 2.5580250166967744e-06
97 0.68721086 2.279660520798643e-06
98 0.69512206 2.402846348559251e-06
99 0.7030043 2.3763377612340264e-06
100 0.7108531 2.5845868094620528e-06
101 0.71870077 2.334294322281494e-06
102 0.7265177 2.280145736222039e-06
103 0.73432523 2.3041966414893977e-06
104 0.7420898 2.303242581547238e-06
105 0.74978316 2.1794035092170816e-06
106 0.7574304 2.0365685031720204e-06
107 0.76501596 2.0377319742692634e-06
108 0.77257127 2.194588660131558e-06
109 0.78012925 2.131105247826781e-06
110 0.7877227 1.955324023583671e-06
111 0.7952557 2.091439228024683e-06
112 0.80278605 2.1004384507250506e-06
113 0.810318 2.441367769279168e-06
114 0.8178436 2.139418256774661e-06
115 0.825362 2.3440695713361492e-06
116 0.8328828 2.318167616977007e-06
117 0.84033525 2.320716703252401e-06
118 0.8478031 2.3263285129360156e-06
119 0.85527384 2.0725783542729914e-06
120 0.8627522 2.3086456621967955e-06
121 0.8702377 2.137223191311932e-06
122 0.87767583 2.0074051008123206e-06
123 0.88507175 2.003750751100597e-06
124 0.89243096 2.084359493892407e-06
125 0.8997858 2.1222529085207498e-06
126 0.9070997 2.1319103780115256e-06
127 0.9144078 2.0067941477464046e-06
128 0.92168254 2.066610932160984e-06
129 0.9289937 2.0197871890559327e-06
130 0.93623686 2.020472265940043e-06
131 0.9434584 2.035428451563348e-06
132 0.9507076 2.061598024738487e-06
133 0.95796466 2.0523373223113595e-06
134 0.96518636 2.0212710296618752e-06
135 0.97236973 2.0048980786668835e-06
136 0.9794895 2.0497568584687542e-06
137 0.986583 2.0535619569272967e-06
138 0.9936203 2.0658633275161264e-06
139 1.000673 2.0821651105507044e-06
140 1.007712 2.084585275952122e-06
141 1.014711 2.0824995772272814e-06
142 1.0216967 2.0984755337849492e-06
143 1.0286314 1.8881830783357145e-06
144 1.0355525 1.9895196601282805e-06
145 1.0424492 1.8702750139709678e-06
146 1.0493333 2.0059178496012464e-06
147 1.0561991 1.966285481103114e-06
148 1.0630286 1.821309638216917e-06
149 1.0698861 1.9129713564325357e-06
150 1.076735 1.7878787730296608e-06
151 1.0835289 1.8801962369252578e-06
152 1.0903178 1.8124344478565035e-06
153 1.0970829 1.948431417986285e-06
154 1.1038616 1.7839453221313306e-06
155 1.1105878 2.1282348825479858e-06
156 1.1173223 1.8473035652277758e-06
```

(continues on next page)

(continued from previous page)

```
157 1.124048 1.9240574147261214e-06
158 1.1307495 1.90238677078014e-06
159 1.1374056 1.964408056664979e-06
160 1.144039 1.807362195904716e-06
161 1.1506593 1.8224728819404845e-06
162 1.1572381 1.8059943158732494e-06
163 1.1638066 1.8310357745576766e-06
164 1.1703814 1.8234918570669834e-06
165 1.176913 1.7533444633954787e-06
166 1.183428 1.8026402130999486e-06
167 1.1899256 1.757058157636493e-06
168 1.1964225 1.7007130281854188e-06
169 1.2028683 1.8384580471320078e-06
170 1.2093241 1.7406479173587286e-06
171 1.2157739 1.8171124338550726e-06
172 1.2222054 1.7633992683840916e-06
173 1.2286144 1.758349867486686e-06
174 1.2350003 1.7779620975488797e-06
175 1.2413809 1.8715239775701775e-06
176 1.2477415 1.7879774532048032e-06
177 1.2541293 1.7896188637678279e-06
178 1.2604756 1.7246613879251527e-06
179 1.266821 1.7677963342066505e-06
180 1.2731405 1.768632500898093e-06
181 1.2794458 1.7839678321251995e-06
182 1.2857305 1.773675194272073e-06
183 1.2919948 1.7608682583158952e-06
184 1.2982318 1.7746033336152323e-06
185 1.3044622 1.7512152226117905e-06
186 1.3106936 1.7233068092536996e-06
187 1.316934 1.7201651871801005e-06
188 1.3231684 1.742623908285168e-06
189 1.3293878 2.024584318860434e-06
190 1.3355787 1.7420566109649371e-06
191 1.3417751 1.9152628283336526e-06
192 1.3479532 1.7532282754473272e-06
193 1.3541085 1.9044773580390029e-06
194 1.3602808 1.7041833189068711e-06
195 1.366442 1.7418759625797975e-06
196 1.3725847 1.6856592992553487e-06
197 1.3787143 1.696750814517145e-06
198 1.3848201 1.7385125374858035e-06
199 1.3909066 1.6674932794558117e-06
200 1.3969783 1.6927473325267783e-06
201 1.4030663 1.6610508737358032e-06
202 1.4091588 1.7703822550174664e-06
203 1.4152539 1.631240934329805e-06
204 1.4213343 1.8443898852638085e-06
205 1.4274035 1.7495285646873526e-06
206 1.433469 1.8178127447754377e-06
207 1.4395151 1.7495285646873526e-06
208 1.4455491 1.7731418893163209e-06
209 1.4515649 1.8737998743745266e-06
210 1.4575803 1.6517809626748203e-06
211 1.4635949 1.8592620563140372e-06
212 1.4695919 1.6185745153052267e-06
213 1.4755827 1.761304929459584e-06
214 1.4815725 1.738357582325989e-06
215 1.487561 1.6811599152788403e-06
216 1.4935583 1.715796543066972e-06
217 1.4995418 1.7122054032370215e-06
```

(continues on next page)

(continued from previous page)

```
218 1.5055261 1.7736508652888006e-06
219 1.5115044 1.7891200059239054e-06
220 1.5174745 1.746884549902461e-06
221 1.5234437 1.7592467429494718e-06
222 1.5294116 1.6930725905694999e-06
223 1.5353682 1.746011434988759e-06
224 1.5413431 1.7534750895720208e-06
225 1.547309 1.7895562223202433e-06
226 1.5532707 1.6966677094387705e-06
227 1.5592226 1.7026038676704047e-06
228 1.5651724 1.7099637261708267e-06
229 1.571117 1.7223369468410965e-06
230 1.5770563 1.7227149555765209e-06
231 1.5829679 1.7147704056696966e-06
232 1.5889034 1.7099637261708267e-06
233 1.594825 1.7354402643832145e-06
234 1.6007354 1.7176122355522239e-06
235 1.6066363 1.7144874391306075e-06
236 1.6125311 1.7581785414222395e-06
237 1.6184242 1.7484341015006066e-06
238 1.6243143 1.7495514157417347e-06
239 1.6301879 1.7149026234619669e-06
240 1.6360868 1.6524779766768916e-06
241 1.6419692 1.7078756400223938e-06
242 1.6478531 1.6397100353060523e-06
243 1.6537348 1.7243577303815982e-06
244 1.6596344 1.6890739971131552e-06
245 1.6655271 1.7073930393962655e-06
246 1.6714175 1.7070633475668728e-06
247 1.6773007 1.645634938540752e-06
248 1.6831809 1.619958425180812e-06
249 1.6890433 1.728237066345173e-06
250 1.6949303 1.6089852579170838e-06
251 1.7008134 1.7228853721462656e-06
252 1.7066984 1.6063656858023023e-06
253 1.7125757 1.6878447013368714e-06
254 1.7184291 1.6039915635701618e-06
255 1.7243074 1.6913855915845488e-06
256 1.7301677 1.6213464277825551e-06
257 1.7360325 1.6034778127504978e-06
258 1.7418728 1.6607069710516953e-06
259 1.747716 1.6025653621909441e-06
260 1.7535695 1.586839061928913e-06
261 1.759432 1.7798387261791504e-06
262 1.7653024 1.654791276450851e-06
263 1.771161 1.6225779972955934e-06
264 1.7770302 1.7413586874681641e-06
265 1.782881 1.7419246205463423e-06
266 1.7887278 1.7413586874681641e-06
267 1.7945843 1.719369265629214e-06
268 1.8004186 1.6063656858023023e-06
269 1.8062667 1.6225779972955934e-06
270 1.8121052 1.7510262750874972e-06
271 1.8179452 1.664269689172153e-06
272 1.8237971 1.623943830963981e-06
273 1.8296481 1.664269689172153e-06
274 1.8354741 1.6678801557645784e-06
275 1.8413198 1.620687726244796e-06
276 1.8471508 1.6626680690023932e-06
277 1.8529727 1.5862756299611647e-06
278 1.8588016 1.5581568959532888e-06
```

(continues on next page)

(continued from previous page)

```
279 1.8646201 1.6088408756331773e-06
280 1.8704267 1.588329268997768e-06
281 1.8762422 1.547453166494961e-06
282 1.8820443 1.6431011999884504e-06
283 1.8878525 1.5395630725834053e-06
284 1.8936607 1.7182350120492629e-06
285 1.8994606 1.5625425930920755e-06
286 1.9052788 1.6618981817373424e-06
287 1.9110817 1.6484990510434727e-06
288 1.9168767 1.5724858712928835e-06
289 1.922693 1.7047034361894475e-06
290 1.9284917 1.714845780043106e-06
291 1.9343079 1.7263245126741822e-06
292 1.940104 1.5685726566516678e-06
293 1.9459344 1.5873513348196866e-06
294 1.9517435 1.6013855201890692e-06
295 1.9575598 1.6020694602048025e-06
296 1.9633615 1.7158174614451127e-06
297 1.9691645 1.7601339550310513e-06
298 1.9749548 1.6021722331061028e-06
299 1.9807627 1.6033654901548289e-06
300 1.986553 1.6478585393997491e-06
301 1.9923509 1.6066078387666494e-06
302 1.9981462 1.5969324067555135e-06
303 2.003934 1.716640213089704e-06
304 2.0097232 1.6126494983836892e-06
305 2.0155315 1.6188232621061616e-06
306 2.0213351 1.6428897424702882e-06
307 2.0271363 1.6122189663292374e-06
308 2.0329351 1.6568342289247084e-06
309 2.0387275 1.6917849734454649e-06
310 2.044552 1.6539599982934305e-06
311 2.0503466 1.6400422282458749e-06
312 2.0561657 1.6703975234122481e-06
313 2.0619702 1.642959659875487e-06
314 2.06779 1.6539599982934305e-06
315 2.0735984 1.6016140307328897e-06
316 2.0794022 1.5999531797206146e-06
317 2.085221 1.6428463140982785e-06
318 2.0910168 1.709260345705843e-06
319 2.096826 1.6943026821536478e-06
320 2.1026254 1.8266432562086266e-06
321 2.1084294 1.7157017282443121e-06
322 2.1142511 1.709772050162428e-06
323 2.1200469 1.614865482224559e-06
324 2.125849 1.6451700730613084e-06
325 2.131642 1.7358287323077093e-06
326 2.1374488 1.7870889905680087e-06
327 2.14325 1.789030761756294e-06
328 2.1490548 1.789030761756294e-06
329 2.154855 1.8155537873099092e-06
330 2.1606586 1.8311978919882677e-06
331 2.166473 1.8155537873099092e-06
332 2.172281 1.8155537873099092e-06
333 2.178079 1.8155537873099092e-06
334 2.183883 1.812017671909416e-06
335 2.1896703 1.8203036233899184e-06
336 2.1954832 1.7114438151111244e-06
337 2.2012768 1.7114438151111244e-06
338 2.2070882 1.748037789184309e-06
339 2.212884 1.7485351690993411e-06
```

(continues on next page)

(continued from previous page)

```
340 2.218683 1.7728050352161517e-06
341 2.2244797 1.7535410279378993e-06
342 2.230282 1.7556739067003946e-06
343 2.2360845 1.7250074506591773e-06
344 2.241873 1.7556739067003946e-06
345 2.2476814 1.7623950725464965e-06
346 2.2534804 1.73460819041793e-06
347 2.2592962 1.737128286549705e-06
348 2.2651088 1.73460819041793e-06
349 2.2709193 1.7476054381404538e-06
350 2.2767446 1.7422609062123229e-06
351 2.2825718 1.7539147165734903e-06
352 2.2883985 1.7437025690014707e-06
353 2.294224 1.7143889863291406e-06
354 2.3000553 1.7928193756233668e-06
355 2.305869 1.7338508087050286e-06
356 2.311704 1.7166065617857384e-06
357 2.3175263 1.7014000377457705e-06
358 2.3233387 1.7084613546103355e-06
359 2.3291824 1.6873108279469307e-06
360 2.3350034 1.7359732282784535e-06
361 2.34083 1.7815876844906597e-06
362 2.3466506 1.6766136923251906e-06
363 2.3524632 1.6829019386932487e-06
364 2.3582797 1.721587750580511e-06
365 2.3641138 1.684452399786096e-06
366 2.369927 1.8488865407562116e-06
367 2.375744 1.789150815056928e-06
368 2.381565 1.813656581362011e-06
369 2.3874013 1.8092271147907013e-06
370 2.3932395 1.8679544382393942e-06
371 2.399063 1.8159023511543637e-06
372 2.404899 1.8193944697486586e-06
373 2.4107351 1.8106647985405289e-06
374 2.4165576 1.8255827853863593e-06
375 2.4223917 1.8103193042406929e-06
376 2.4282436 1.8615978660818655e-06
377 2.4340796 1.7979589301830856e-06
378 2.4399252 1.8405683022137964e-06
379 2.4457798 1.8159023511543637e-06
380 2.4516253 1.8072145167025155e-06
381 2.457479 1.837138029259222e-06
382 2.4633346 1.8013893168244977e-06
383 2.4691906 1.8369254348726827e-06
384 2.4750426 1.8137498045689426e-06
385 2.4808903 1.7914175032274215e-06
386 2.4867392 1.6816944707898074e-06
387 2.4925807 1.8200786371380673e-06
388 2.4984374 1.6695898921170738e-06
389 2.5042825 1.8162896822104813e-06
390 2.510125 1.6782643115220708e-06
391 2.515974 1.8200786371380673e-06
392 2.52181 1.7966257246371242e-06
393 2.527664 1.8048295942207915e-06
394 2.533502 1.8313425016458496e-06
395 2.5393505 1.7238245391126839e-06
396 2.5451915 1.7825421991801704e-06
397 2.5510437 1.8361664615440532e-06
398 2.556896 1.7741521105563152e-06
399 2.562738 1.7606042774787056e-06
400 2.5685985 1.7003198991005775e-06
```

(continues on next page)

(continued from previous page)

```
401 2.5744686 1.7891816241899505e-06
402 2.5803156 1.8211824226455064e-06
403 2.5861924 1.703745624581643e-06
404 2.5920415 1.8082954511555727e-06
405 2.5979273 1.6503026927239262e-06
406 2.6037896 1.8172761429013917e-06
407 2.6096485 1.7681799135971232e-06
408 2.6155293 1.6457153151350212e-06
409 2.6214068 1.777228476385062e-06
410 2.62728 1.6804528968350496e-06
411 2.6331532 1.7856614249467384e-06
412 2.6390297 1.7129739262600197e-06
413 2.644907 1.6397831359427073e-06
414 2.6507964 1.7460699837101856e-06
415 2.6566744 1.73190471741691e-06
416 2.6625571 1.7189885284096817e-06
417 2.6684518 1.725313836686837e-06
418 2.6743395 1.6665145494698663e-06
419 2.6802287 1.7390160564900725e-06
420 2.686115 1.8027202486337046e-06
421 2.6920135 1.7086223351725494e-06
422 2.6979165 1.7450466884838534e-06
423 2.7038124 1.681463118075044e-06
424 2.7097096 1.588900545357319e-06
425 2.7156024 1.7194050769830937e-06
426 2.7214992 1.689453029030119e-06
427 2.72739 1.7903746538650012e-06
428 2.7332873 1.7283655324717984e-06
429 2.739193 1.6751293969718972e-06
430 2.7450848 1.6778164990682853e-06
431 2.7509806 1.6357305412384449e-06
432 2.7568893 1.6976675851765322e-06
433 2.7627888 1.6230202390943305e-06
434 2.7686825 1.6719586710678414e-06
435 2.774582 1.6784836134320358e-06
436 2.7804813 1.6762437553552445e-06
437 2.7863734 1.7071191678041941e-06
438 2.792264 1.6391867347920197e-06
439 2.798154 1.7005889958454645e-06
440 2.804056 1.6307027408402064e-06
441 2.809946 1.7031997003869037e-06
442 2.8158476 1.6785926391094108e-06
443 2.8217416 1.6883423086255789e-06
444 2.8276358 1.6752669580455404e-06
445 2.8335352 1.6647729808028089e-06
446 2.839438 1.7077062466341886e-06
447 2.8453295 1.7174426147903432e-06
448 2.8512273 1.6801551510070567e-06
449 2.8571374 1.8201924376626266e-06
450 2.8630302 1.672326789048384e-06
451 2.868934 1.6624579757262836e-06
452 2.8748403 1.7368059843647643e-06
453 2.8807316 1.7102281617553672e-06
454 2.8866398 1.7216825654031709e-06
455 2.8925488 1.7064105577446753e-06
456 2.8984416 1.6882175941645983e-06
457 2.9043498 1.6590736322541488e-06
458 2.9102578 1.6737456007831497e-06
459 2.9161575 1.6651905525577604e-06
460 2.9220626 1.6969248690656968e-06
461 2.9279609 1.6696956208761549e-06
```

(continues on next page)

(continued from previous page)

```
462 2.933869 1.7771179727787967e-06
463 2.9397695 1.674076656854595e-06
464 2.945677 1.7163570191769395e-06
465 2.951589 1.6731877394704497e-06
466 2.9575086 1.7139578858405002e-06
467 2.963423 1.661029273236636e-06
468 2.9693468 1.7322533949482022e-06
469 2.975268 1.6756764580350136e-06
470 2.9812174 1.6987077060548472e-06
471 2.9871242 1.7214939589393907e-06
472 2.9930735 1.7048627114490955e-06
473 2.9990044 1.669786229285819e-06
474 3.0049403 1.7561002323418506e-06
475 3.010886 1.6786905234766891e-06
476 3.0168402 1.7134439076471608e-06
477 3.0227835 1.8284858924744185e-06
478 3.0287352 1.6514883327545249e-06
479 3.0346859 1.6616103266642313e-06
480 3.0406318 1.7071628235498792e-06
481 3.0465846 1.7420715039406787e-06
482 3.0525298 1.64506320743385e-06
483 3.058484 1.6777397604528232e-06
484 3.064436 1.6965221902864869e-06
485 3.0704076 1.6948403072092333e-06
486 3.0763555 1.7142388060165104e-06
487 3.0823278 1.6574963410675991e-06
488 3.0882726 1.7402866205884493e-06
489 3.0942423 1.6537692317797337e-06
490 3.1002085 1.7189370282721939e-06
491 3.106177 1.703623070170579e-06
492 3.112145 1.641988546907669e-06
493 3.1181276 1.6608723854005802e-06
494 3.1241035 1.6886179992070538e-06
495 3.1300848 1.80542645011883e-06
496 3.1360772 1.6668717535139876e-06
497 3.1420631 1.7484957197666517e-06
498 3.1480522 1.667902779445285e-06
499 3.1540456 1.6968116369753261e-06
500 3.1600385 1.6713271406842978e-06
501 3.166042 1.6710120007701335e-06
502 3.1720457 1.8380768551651272e-06
503 3.1780443 1.6790278323242092e-06
504 3.184061 1.6918279470701236e-06
505 3.1900814 1.6624138652332476e-06
506 3.1960917 1.6688819641785813e-06
507 3.2021148 1.676875285738788e-06
508 3.2081254 1.6905082702578511e-06
509 3.2141535 1.6573727634749957e-06
510 3.220169 1.698192818366806e-06
511 3.2261899 1.7021875464706682e-06
512 3.232219 1.668042045821494e-06
513 3.2382464 1.6760531025283854e-06
514 3.2442691 1.6632249071335536e-06
515 3.250313 1.7215471643794444e-06
516 3.256336 1.6828076923047774e-06
517 3.2623727 1.6920608914006152e-06
518 3.2684093 1.7298826833211933e-06
519 3.274449 1.7009809880619287e-06
520 3.280478 1.6754269154262147e-06
521 3.286532 1.6854327213877696e-06
522 3.2925699 1.708270474409801e-06
```

(continues on next page)

(continued from previous page)

```
523 3.2986016 1.6272686025331495e-06
524 3.3046494 1.6486686718053534e-06
525 3.310684 1.7615728893360938e-06
526 3.3167267 1.6852378621479147e-06
527 3.3227637 1.7030619119395851e-06
528 3.328809 1.6097968682515784e-06
529 3.3348544 1.670290544097952e-06
530 3.340892 1.6916332015171065e-06
531 3.3469334 1.6749883116062847e-06
532 3.352978 1.721573880786309e-06
533 3.3590178 1.7256733144677128e-06
534 3.3650823 1.6826103319544927e-06
535 3.3711236 1.6640146895952057e-06
536 3.3771732 1.6377612155338284e-06
537 3.3832316 1.6585408957325853e-06
538 3.3892872 1.7635908307056525e-06
539 3.3953278 1.5998950857465388e-06
540 3.4013922 1.6487526863784296e-06
541 3.4074502 1.6773368542999378e-06
542 3.4135115 1.8254601172884577e-06
543 3.4195817 1.7112763543991605e-06
544 3.4256518 1.6451551800855668e-06
545 3.4317117 1.7143267996289069e-06
546 3.437784 1.694193542789435e-06
547 3.443861 1.6784805438874173e-06
548 3.449934 1.7635908307056525e-06
549 3.4560049 1.7452287011110457e-06
550 3.4620864 1.5816675613677944e-06
551 3.4681492 1.6864200915733818e-06
552 3.474223 1.7116740309575107e-06
553 3.480306 1.6506276097061345e-06
554 3.4863844 1.6905745496842428e-06
555 3.4924586 1.7918794128490845e-06
556 3.498539 1.7131752656496246e-06
557 3.5046186 1.6392875750170788e-06
558 3.510706 1.6861249605426565e-06
559 3.516794 1.7711198552206042e-06
560 3.5228853 1.6755483329689014e-06
561 3.5289695 1.6604211623416631e-06
562 3.5350573 1.5914338291622698e-06
563 3.541166 1.7293054952460807e-06
564 3.5472608 1.7280317479162477e-06
565 3.5533526 1.71656506608997e-06
566 3.5594554 1.6032978464863845e-06
567 3.5655522 1.6928891000134172e-06
568 3.5716484 1.627438336981868e-06
569 3.577751 1.8018662331087398e-06
570 3.5838428 1.755718699314457e-06
571 3.5899494 1.6494269630129565e-06
572 3.596051 1.7617962839722168e-06
573 3.6021695 1.7485892840340966e-06
574 3.6082754 1.6284676576105994e-06
575 3.6143827 1.6298188256769208e-06
576 3.6204987 1.6975056951196166e-06
577 3.626606 1.7488928278908134e-06
578 3.6327133 1.6309453485519043e-06
579 3.638844 1.7182813962790533e-06
580 3.6449485 1.6578959503021906e-06
581 3.6510751 1.6714745925128227e-06
582 3.6571927 1.6831987750265398e-06
583 3.6633105 1.6187074152185232e-06
```

(continues on next page)

(continued from previous page)

```
584 3.6694348 1.7194745396409417e-06
585 3.6755514 1.7071603224394494e-06
586 3.681671 1.8047642242891015e-06
587 3.687788 1.7210505802722764e-06
588 3.6939116 1.6983469777187565e-06
589 3.7000415 1.636445517760876e-06
590 3.706157 1.6957121715677204e-06
591 3.7122867 1.7668204463916481e-06
592 3.7184098 1.734637976369413e-06
593 3.724531 1.6203589439101052e-06
594 3.7306647 1.7357579054078087e-06
595 3.7367845 1.7006499319904833e-06
596 3.7429194 1.9755648281716276e-06
597 3.749042 1.6935819076024927e-06
598 3.7551737 1.618860665075772e-06
599 3.7613108 1.7006159396260045e-06
600 3.767444 1.7253556734431186e-06
601 3.773571 1.6727385627746116e-06
602 3.779703 1.7998248722506105e-06
603 3.7858403 1.6769959074736107e-06
604 3.7919753 1.631628720133449e-06
605 3.7981172 1.6779396219135378e-06
606 3.804259 1.675853582128184e-06
607 3.810389 1.6736714769649552e-06
608 3.816542 1.8809955690812785e-06
609 3.8226871 1.6862687743923743e-06
610 3.8288317 1.6732579979361617e-06
611 3.8349714 1.7302587593803764e-06
612 3.841117 1.6666998590153526e-06
613 3.8472602 1.6034911141105113e-06
614 3.8534007 1.740515017445432e-06
615 3.8595514 1.764596504472138e-06
616 3.865693 1.6709847159290803e-06
617 3.8718433 1.6161435496542254e-06
618 3.877986 1.8330010789213702e-06
619 3.884143 1.6563634517297032e-06
620 3.890289 1.7164430801130948e-06
621 3.8964326 1.8467457039150759e-06
622 3.90259 1.9118274394713808e-06
623 3.9087365 1.806626642064657e-06
624 3.9148822 1.6197426475628163e-06
625 3.9210389 1.7004207393256365e-06
626 3.92719 1.6171619563465356e-06
627 3.9333296 1.7012167745633633e-06
628 3.939485 1.6737743635530933e-06
629 3.9456491 1.6502413018315565e-06
630 3.9518042 1.7604070308152586e-06
631 3.957956 1.6484343632328091e-06
632 3.9641092 1.842885239966563e-06
633 3.9702735 1.6041195749494364e-06
634 3.9764354 1.7223239865415962e-06
635 3.9825883 1.6795503370303777e-06
636 3.9887533 1.7361360278300708e-06
637 3.994913 1.695530613687879e-06
638 4.001069 1.651663524171454e-06
639 4.0072265 1.624738047212304e-06
640 4.0134077 1.6882809177332092e-06
641 4.0195637 1.7647918184593436e-06
642 4.02574 1.5934554085106356e-06
643 4.0319204 1.7163232541861362e-06
644 4.038089 1.628685936339025e-06
```

(continues on next page)

(continued from previous page)

```
645 4.0442743 1.6988828974717762e-06
646 4.050464 1.6111703189380933e-06
647 4.056641 1.8734641571427346e-06
648 4.0628304 1.6050108797571738e-06
649 4.069009 1.7011698218993843e-06
650 4.0751905 1.6797209809737979e-06
651 4.081372 1.632559360587038e-06
652 4.087567 1.7163202983283554e-06
653 4.09374 1.6684649608578184e-06
654 4.099928 1.6832823348522652e-06
655 4.1061044 1.6589945062150946e-06
656 4.1122856 1.6490472489749664e-06
657 4.118484 1.725449465084239e-06
658 4.124662 1.6525050341442693e-06
659 4.1308465 1.7104114249377744e-06
660 4.1370444 1.5741793504275847e-06
661 4.143232 1.7449405049774214e-06
662 4.149425 1.550896627122711e-06
663 4.1556153 1.8236502228319296e-06
664 4.1618085 1.7070160538423806e-06
665 4.1679993 1.7178124380734516e-06
666 4.174194 1.6629330730211223e-06
667 4.1803927 1.8105811250279658e-06
668 4.186591 1.7740534303811728e-06
669 4.1927905 1.7804288745537633e-06
670 4.198987 1.6653951888656593e-06
671 4.2051773 1.88624676411564e-06
672 4.211372 1.7927103499459918e-06
673 4.21757 1.741463279358868e-06
674 4.2237678 1.7786034050004673e-06
675 4.229956 1.8284563338966109e-06
676 4.236158 1.6502409607710433e-06
677 4.242359 1.7456526393289096e-06
678 4.248543 1.7897558564072824e-06
679 4.2547474 1.7170327737403568e-06
680 4.2609386 1.60837669227476e-06
681 4.2671447 1.7002779486574582e-06
682 4.273341 1.7003785615088418e-06
683 4.2795444 1.6132710243255133e-06
684 4.28574 1.6683542298778775e-06
685 4.2919436 1.6401078255512402e-06
686 4.2981515 1.7814900274970569e-06
687 4.3043685 1.6978412986645708e-06
688 4.310569 1.6302261656164774e-06
689 4.316775 1.766163791216968e-06
690 4.3229656 1.772887003426149e-06
691 4.3291874 1.805934857657121e-06
692 4.3354044 1.6795447663753293e-06
693 4.3416104 1.6885235254449071e-06
694 4.3478255 1.7699204590826412e-06
695 4.354029 1.7617926459934097e-06
696 4.360256 1.7122670215030666e-06
697 4.3664637 1.7800831528802519e-06
698 4.3726835 1.6175806649698643e-06
699 4.378895 1.645407905925822e-06
700 4.385107 1.97333974938374e-06
701 4.39134 1.7292912843913655e-06
702 4.397546 1.6468771946165361e-06
703 4.4037657 1.6599730088273645e-06
704 4.409991 1.7150166513602016e-06
705 4.4162073 1.7266902432311326e-06
```

(continues on next page)

(continued from previous page)

```
706 4.4224424 1.8328626083530253e-06
707 4.4286623 1.6524694501640624e-06
708 4.4348845 1.6990726408039336e-06
709 4.4411087 1.6110396927615511e-06
710 4.447324 1.777778606992797e-06
711 4.4535575 1.980507704502088e-06
712 4.4597797 1.675831072134315e-06
713 4.4660063 1.7191332517541014e-06
714 4.4722314 1.73876469489187e-06
715 4.478453 1.8104384480466251e-06
716 4.4846807 1.8196443534179707e-06
717 4.490916 1.6171507013496011e-06
718 4.4971414 1.8607137235449045e-06
719 4.503376 1.7222489532287e-06
720 4.509608 1.7009537032208755e-06
721 4.515831 1.6891222003323492e-06
722 4.522068 1.7695875840217923e-06
723 4.528312 1.808035563044541e-06
724 4.5345435 1.7160188008347177e-06
725 4.540775 1.6616450011497363e-06
726 4.547013 1.6634945723126293e-06
727 4.5532475 1.655758751439862e-06
728 4.559491 1.6850452766448143e-06
729 4.5657253 1.572742917232972e-06
730 4.5719643 1.7430047591915354e-06
731 4.5782022 1.7566829910720116e-06
732 4.584439 1.7316807543465984e-06
733 4.590691 1.6235982229773072e-06
734 4.596923 1.823119191612932e-06
735 4.603169 1.6479848454764578e-06
736 4.6094174 1.7051301028914168e-06
737 4.61565 1.805746592253854e-06
738 4.621894 1.7050143696906161e-06
739 4.6281376 1.6618547533653327e-06
740 4.634396 1.7940695897777914e-06
741 4.640645 1.822428544073773e-06
742 4.6468854 1.644363237573998e-06
743 4.6531396 1.6407803968832013e-06
744 4.659387 1.7317855736109777e-06
745 4.6656394 1.7253956912099966e-06
746 4.6718917 1.7928567785929772e-06
747 4.6781487 1.732252258079825e-06
748 4.6844163 1.7266588656639215e-06
749 4.690659 1.7194546444443404e-06
750 4.6969137 1.7269716181544936e-06
751 4.7031803 1.7441326463085716e-06
752 4.7094336 1.7580592839294695e-06
753 4.7156887 1.7437578208046034e-06
754 4.721974 1.6934529867285164e-06
755 4.728212 1.764227704370569e-06
756 4.7344856 1.770920221133565e-06
757 4.74074 1.7886527530208696e-06
758 4.747015 1.645152678975137e-06
759 4.7532816 1.686193741079478e-06
760 4.759547 1.8361829461355228e-06
761 4.7658143 1.7834132677307935e-06
762 4.772084 1.8147169384974404e-06
763 4.778359 1.6451288047392154e-06
764 4.7846365 1.7195129657920916e-06
765 4.7909126 1.7350421330775134e-06
766 4.7971773 1.7551217297295807e-06
```

(continues on next page)

(continued from previous page)

```
767 4.803445 1.7297710428465507e-06
768 4.809731 1.720574573482736e-06
769 4.816003 1.7493496216047788e-06
770 4.8222837 1.727334961287852e-06
771 4.8285623 1.6459425751236267e-06
772 4.8348346 1.7159388789877994e-06
773 4.8411174 1.8683374491956783e-06
774 4.847401 1.6251543684120406e-06
775 4.853673 1.6988705056064646e-06
776 4.859957 1.6846032622197527e-06
777 4.866243 1.7799635543269687e-06
778 4.8725348 1.7368907947457046e-06
779 4.8788166 1.573215854477894e-06
780 4.885112 1.7900595139508368e-06
781 4.891404 1.6394740214309422e-06
782 4.8976965 1.6818598851386923e-06
783 4.9039927 1.6216988569794921e-06
784 4.9102807 1.842824758568895e-06
785 4.916579 1.664713295213005e-06
786 4.9228773 1.6767688748586806e-06
787 4.9291778 1.7131501408584882e-06
788 4.935464 1.6453772104796371e-06
789 4.941762 1.6528427977391402e-06
790 4.948056 1.6644481775074382e-06
791 4.954363 1.8117095805791905e-06
792 4.9606543 1.640500386201893e-06
793 4.9669495 1.6357402046196512e-06
794 4.9732513 1.7153088265331462e-06
795 4.9795537 1.693157855697791e-06
796 4.985849 1.6430649338872172e-06
797 4.992156 1.6320084341714391e-06
798 4.9984574 1.697559468993859e-06
799 5.0047445 1.6592917972957366e-06
800 5.0110483 1.6623159808659693e-06
801 5.017361 1.7724245253702975e-06
802 5.023653 1.6340566162398318e-06
803 5.029963 1.7970033923120354e-06
804 5.036264 1.7759978163667256e-06
805 5.042574 1.7013944670907222e-06
806 5.0488806 1.6543017409276217e-06
807 5.05519 1.8300238480151165e-06
808 5.0615 1.6380139413740835e-06
809 5.0678115 1.6407725524913985e-06
810 5.074121 1.656165864005743e-06
811 5.0804253 1.7543821968502016e-06
812 5.086738 1.642948177504877e-06
813 5.093045 1.651580191719404e-06
814 5.0993586 1.7746024241205305e-06
815 5.1056824 1.7024489125105902e-06
816 5.111981 1.639034280742635e-06
817 5.1182985 1.6255378341156757e-06
818 5.12461 1.761962835189479e-06
819 5.130929 1.6022416957639507e-06
820 5.1372495 1.8234849221698823e-06
821 5.1435556 1.7080334373531514e-06
822 5.1498723 1.8549034166426281e-06
823 5.1561894 1.6324186162819387e-06
824 5.162512 1.7546200297147152e-06
825 5.168831 1.763626528372697e-06
826 5.175143 1.8991477190866135e-06
827 5.1814575 1.7334813264824334e-06
```

(continues on next page)

(continued from previous page)

```
828 5.1877775 1.6132496512000216e-06
829 5.1941004 1.6302150243063807e-06
830 5.200423 1.7489427364125731e-06
831 5.2067413 1.636852744013595e-06
832 5.2130594 1.724720505080768e-06
833 5.219387 1.6810116676424514e-06
834 5.2257137 1.6008250440791016e-06
835 5.2320414 1.7502334230812266e-06
836 5.238367 1.6377963447666843e-06
837 5.244691 1.6773300330896745e-06
838 5.251017 1.6209320392590598e-06
839 5.2573433 1.7680218888926902e-06
840 5.263673 1.637284412936424e-06
841 5.270005 1.8438696542943944e-06
842 5.27634 1.6766979342719424e-06
843 5.282678 1.736919671202486e-06
844 5.289015 1.6591635585427866e-06
845 5.2953405 1.7078185692298575e-06
846 5.3016806 1.655257278798672e-06
847 5.3080063 1.7163685015475494e-06
848 5.3143487 1.7763649111657287e-06
849 5.3206854 1.6711087482690345e-06
850 5.3270345 1.6502870039403206e-06
851 5.333374 1.7075091136575793e-06
852 5.3397136 1.6468693502247334e-06
853 5.34606 1.7454465250921203e-06
854 5.3524003 1.579881086399837e-06
855 5.358747 1.77857532435155e-06
856 5.3650904 1.6221921441683662e-06
857 5.371433 1.6678801557645784e-06
858 5.3777833 1.7296210899075959e-06
859 5.3841195 1.8131132719645393e-06
860 5.390481 1.633234433029429e-06
861 5.396835 1.7384630837113946e-06
862 5.403186 1.6590410041317227e-06
863 5.4095407 1.635640614949807e-06
864 5.415895 1.7211859812960029e-06
865 5.4222507 1.5508245496675954e-06
866 5.4286013 1.9029386066904408e-06
867 5.4349575 1.7718562048685271e-06
868 5.441319 1.7395070699421922e-06
869 5.4476905 1.6992609062072006e-06
870 5.454049 1.7201507489517098e-06
871 5.460402 1.7075091136575793e-06
872 5.46676 1.6576551615798962e-06
873 5.473131 1.760751956680906e-06
874 5.479487 1.6266823195110192e-06
875 5.485844 1.6503728375028004e-06
876 5.492205 1.7671486602921505e-06
877 5.498563 1.7708632640278665e-06
878 5.504928 1.7089939774450613e-06
879 5.5112777 1.6250041880994104e-06
880 5.517655 1.6548065104871057e-06
881 5.5240183 1.7017023310472723e-06
882 5.5303836 1.6224463479375117e-06
883 5.5367413 1.7235940958926221e-06
884 5.543112 1.62195044595137e-06
885 5.5494895 1.6322679812219576e-06
886 5.555859 1.7500346984888893e-06
887 5.562226 1.7096535884775221e-06
888 5.568587 1.604781687092327e-06
```

(continues on next page)

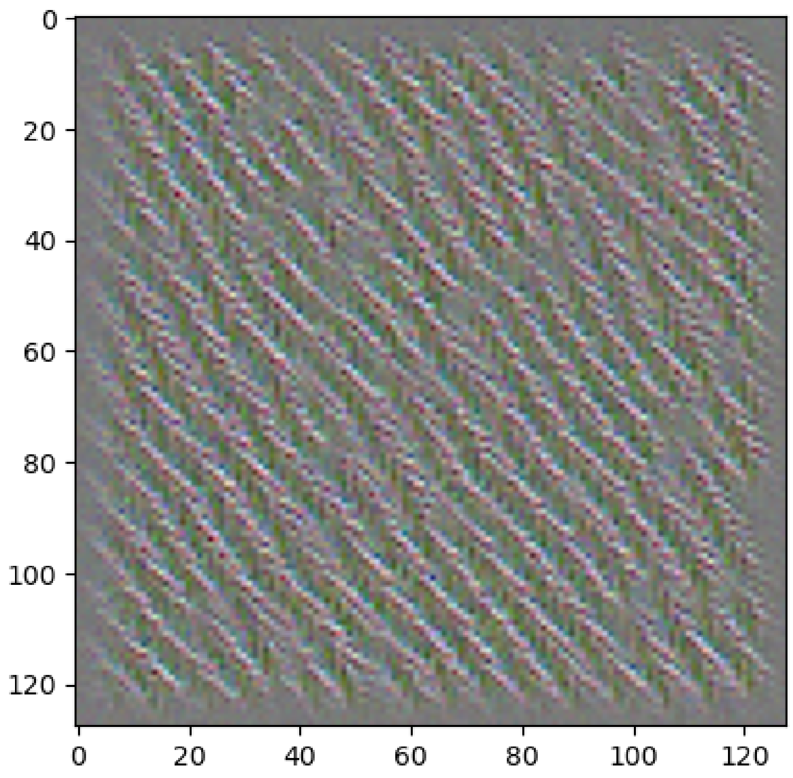
(continued from previous page)

```
889 5.574962 1.6294660554194706e-06
890 5.581338 1.780065531420405e-06
891 5.587704 1.8804458932208945e-06
892 5.594076 1.7026867453751038e-06
893 5.6004515 1.6619214875390753e-06
894 5.6068215 1.6469680303998757e-06
895 5.613177 1.6184942523977952e-06
896 5.619562 1.6599205991951749e-06
897 5.625937 1.7797709688238683e-06
898 5.632298 1.891331521619577e-06
899 5.638677 1.7176524806927773e-06
900 5.6450505 1.723349100757332e-06
901 5.6514096 1.6731536334191333e-06
902 5.6577888 1.6086880805232795e-06
903 5.664159 1.8455691588314949e-06
904 5.670536 1.6383276033593575e-06
905 5.6769075 1.7095630937546957e-06
906 5.6832795 1.6461182212879066e-06
907 5.689653 1.654504558246117e-06
908 5.6960125 1.7337039253106923e-06
909 5.7024026 1.778723230927426e-06
910 5.70878 1.8618028434502776e-06
911 5.715159 1.7076870335586136e-06
912 5.7215366 1.754103891471459e-06
913 5.7279105 1.7281404325331096e-06
914 5.7342978 1.8664819663172239e-06
915 5.7406793 1.7003835637297016e-06
916 5.747051 1.7714909290589276e-06
917 5.753442 1.694864067758317e-06
918 5.759825 1.764663466019556e-06
919 5.76621 1.718317435006611e-06
920 5.7725987 1.6749115729908226e-06
921 5.7789946 1.6792032511148136e-06
922 5.7853804 1.6587167692705407e-06
923 5.7917705 1.671293148319819e-06
924 5.7981577 1.6653403918098775e-06
925 5.8045535 1.7507753682366456e-06
926 5.8109474 1.6899822412597132e-06
927 5.817342 1.7267226439798833e-06
928 5.823733 1.7247328969460796e-06
929 5.830119 1.71208273513912e-06
930 5.836513 1.5968960269674426e-06
931 5.842914 1.8610041934152832e-06
932 5.8493123 1.7507753682366456e-06
933 5.8557 1.7530488776174025e-06
934 5.8621154 1.7467413044869318e-06
935 5.868517 1.8935936623165617e-06
936 5.8749228 1.6666591591274482e-06
937 5.881329 1.7613148202144657e-06
938 5.8877196 1.7071403135560104e-06
939 5.8941364 1.708106651676644e-06
940 5.9005475 1.6474068615934812e-06
941 5.906949 1.7909155758388806e-06
942 5.913348 1.8553491827333346e-06
943 5.919756 1.6858411981957033e-06
944 5.9261665 1.7536426639708225e-06
945 5.932576 1.6733692973502912e-06
946 5.938983 1.7633853985898895e-06
947 5.9453936 1.7010621604640619e-06
948 5.9518075 1.670299525358132e-06
949 5.958211 1.6554276953684166e-06
```

(continues on next page)

(continued from previous page)

```
950 5.964619 1.7164958308057976e-06
951 5.9710383 1.719412352940708e-06
952 5.9774466 1.6592796328041004e-06
953 5.9838567 1.8132978993889992e-06
954 5.990267 1.7923658788276953e-06
955 5.9966774 1.7282524140682654e-06
956 6.0030828 1.7527827367302962e-06
957 6.0094995 1.848863917075505e-06
958 6.015902 1.6764336123742396e-06
959 6.022327 1.6923114571909537e-06
960 6.028728 1.723529635455634e-06
961 6.0351386 1.6817120922496542e-06
962 6.0415545 1.7283176703131176e-06
963 6.0479617 1.7984257283387706e-06
964 6.054376 1.7307843336311635e-06
965 6.0607986 1.6838952205944224e-06
966 6.0672064 1.6693428506187047e-06
967 6.0736256 1.7929770592672867e-06
968 6.080048 1.7184443095175084e-06
969 6.086478 1.7510076304461109e-06
970 6.092886 1.686983182480617e-06
971 6.09931 1.775976784301747e-06
972 6.1057267 1.7042441413650522e-06
973 6.112159 1.8057138504445902e-06
974 6.118568 1.749461034705746e-06
975 6.124989 1.737494244480331e-06
976 6.131411 1.697133484412916e-06
977 6.1378384 1.6054913203333854e-06
978 6.144261 1.7779842664822354e-06
979 6.1506705 1.7613322143006371e-06
980 6.1570954 1.735858290885517e-06
981 6.1635146 1.7273670209760894e-06
982 6.1699443 1.8016651210928103e-06
983 6.176364 1.6577887436142191e-06
984 6.182785 1.7000637626551907e-06
985 6.189211 1.6098349533422152e-06
986 6.1956363 1.6708280554666999e-06
987 6.202062 1.6812678040878382e-06
988 6.20849 1.675260136835277e-06
989 6.2149096 1.6073533970484277e-06
990 6.221336 1.7308377664448926e-06
991 6.2277627 1.7490103800810175e-06
992 6.2341957 1.73156490745896e-06
993 6.2406282 1.7092773987315013e-06
994 6.247062 1.7839250858742162e-06
995 6.253493 1.768219590303488e-06
996 6.2599254 1.6105500435514841e-06
997 6.2663507 1.867201490313164e-06
998 6.2727757 1.5852081105549587e-06
999 6.2792096 1.7314188198724878e-06
```



26.6.5 Class Activation Maps

We may ask what regions of an image make the CNN 'think' that there is a cat or a dog. A simple approach is to pass an image through the CNN and then look at the gradient of the last convolution layer's output with respect to an output neuron (cat or dog). By the principle of local connectivity spatial regions of a feature map are strongly related to the same spatial regions of the input image. High positive components in the gradient tell us that increasing the presence of the corresponding feature in the corresponding region would increase the chosen output neuron's output. Very negative components tell us that the feature in this region lowers output.

To get the gradient of an output neuron with respect to the outputs of a hidden layer we have to remember what TensorFlow's automatic differentiation routines can do and what they cannot do. What TensorFlow can do is calculating the gradient of some function with respect to a concrete tensor flowing through the graph. But derivatives with respect to some abstract tensor (a kind of placeholder) are not accessible. So we may formulate more precisely: we want to have the gradient of a neuron's output with respect to the tensor flowing out of a hidden layer when some tensor is pushed through the CNN. The problem is that Keras does not implement accessing interim results. The solution is to create a new model with two outpus. One output is the usual output layer, the other is the hidden convolution layer of interest. This does not change the CNN's structure, but forces Keras to provide access to the concrete tensor object coming out of the hidden layer and moving on to the next layer.

```
layer = model.get_layer('conv4')

submodel = keras.models.Model(inputs=model.inputs,
                              outputs=[layer.output, model.output])
```

Now we load an image and preprocess it as usual.

```
img = keras.preprocessing.image.load_img(data_path + 'unlabeled/1696.jpg', # 318,␣
↪786, 907, 1696
                                         target_size=(img_size, img_size))
img = np.asarray(img, dtype=np.float32)
```

We want to have two gradients: the gradient of the cat output neuron and the gradient of the dog output neuron. Since we have two outputs in our model, predictions yield a list of two tensors.

```
img_tensor = tf.convert_to_tensor(img.reshape(1, img_size, img_size, 3))

with tf.GradientTape() as tape:
    tape.watch(img_tensor)
    pred = submodel(img_tensor)
    cat_grad = tape.gradient(pred[1][0, 0], pred[0])

with tf.GradientTape() as tape:
    tape.watch(img_tensor)
    pred = submodel(img_tensor)
    dog_grad = tape.gradient(pred[1][0, 1], pred[0])

fmaps = pred[0].numpy()[0, :, :, :]
cat_grad = cat_grad.numpy()[0, :, :, :]
dog_grad = dog_grad.numpy()[0, :, :, :]

print(fmaps.shape, cat_grad.shape, dog_grad.shape)
```

```
(58, 58, 32) (58, 58, 32) (58, 58, 32)
```

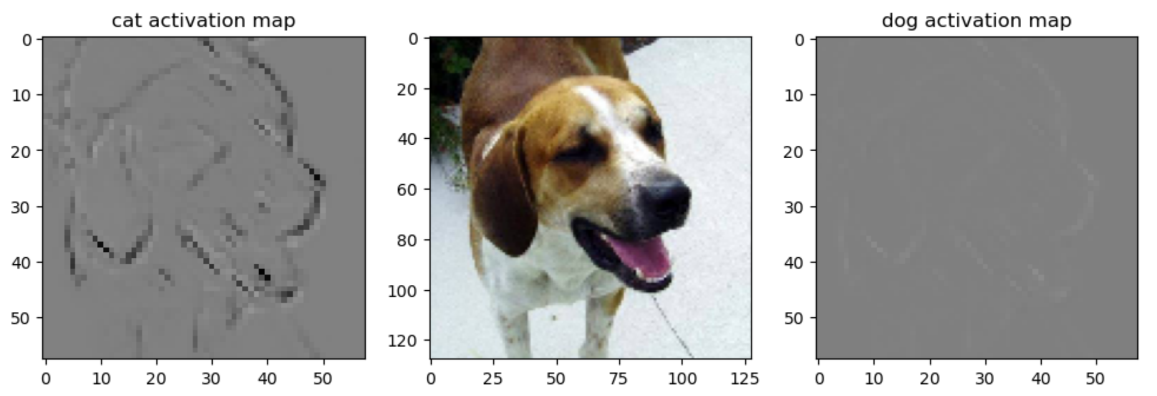
Now we are ready to compute the class activation map (CAM). The CAM has same shape as a feature map in the last convolutional layer (same width and height, depth is 1). The CAM is a weighted sum of all feature maps of the last convolutional layer. The weights are calculated from the gradient by spacial averaging. Thus, for each feature map the weight is something like a mean partial derivative. If the weight is positive, then the feature represented by the corresponding feature map potentially increases class activation. If the weight is negative, then class activation is decreased the more nonzero values in the feature map.

Multiplying mean gradients by the feature map values yields high positive numbers in regions where a class activation increasing feature is present in the input image, but negative values in regions where features are present which potentially decrease class activation.

We scale the CAM to $[0, 1]$ such that 0.5 corresponds to 0 in the original CAM.

```
cat_weights = np.mean(cat_grad, axis=(0, 1)).reshape(1, 1, -1)
cat_cam = np.sum(fmaps * cat_weights, axis=2)
dog_weights = np.mean(dog_grad, axis=(0, 1)).reshape(1, 1, -1)
dog_cam = np.sum(fmaps * dog_weights, axis=2)

fac = np.maximum(np.max(np.abs(cat_cam)), np.max(np.abs(dog_cam)))
cat_cam = 0.5 * (1 + cat_cam / fac)
dog_cam = 0.5 * (1 + dog_cam / fac)

print('cat: {:.2f}, dog: {:.2f}'.format(pred[1][0, 0], pred[1][0, 1]))

fig, [ax1, ax2, ax3] = plt.subplots(1, 3, figsize=(12, 6))
ax1.imshow(cat_cam, cmap='gray', vmin=0, vmax=1)
ax2.imshow(img / 255)
ax3.imshow(dog_cam, cmap='gray', vmin=0, vmax=1)
ax1.set_title('cat activation map')
ax3.set_title('dog activation map')
plt.show()
```

```
cat: 0.12, dog: 1.00
```



For better visual interpretation we overlay the original image with the CAM. Many people do this in a very sloppy way by simply resizing the CAM to image size. But we take the hard and correct one. The difficult part is to find the region associated with a value in the CAM. Going backwards through the CNN's layers we have to calculate size and position of the *region of interest* (ROI) for each component of the CAM.

A pixel in the feature map results from a convolution with a 3x3 filter. Thus a 3x3 region is the preimage of the pixel. One layer up we have a 5x5 region (convolution with 3x3 filter again). Then there is a pooling layer. So the ROI's size before pooling is 10x10. Then again two 3x3 convolutions, yielding a 14x14 ROI.

The CAM is 58x58. The original image is 128x128. Centers of all ROIs have to be placed equally spaced in the 128x128 image such that there is a 7 pixel boundary. Else some ROIs would partially lie outside the image. Distance between ROI centers is $(128 - 14)/57 = 2$ pixels.

With this knowledge we create a stack of images. One image per CAM component. Each containing the CAM component's value in all pixels belonging to the component's ROI. Then we merge all images in the stack by taking the pixelwise mean. Here we have to take into account that pixels near the boundary belong to fewer ROIs than pixels in the image center.

To overlay CAM image and original image we use a color map with blue for negative CAM values, gray for zero and red for positive CAM values.

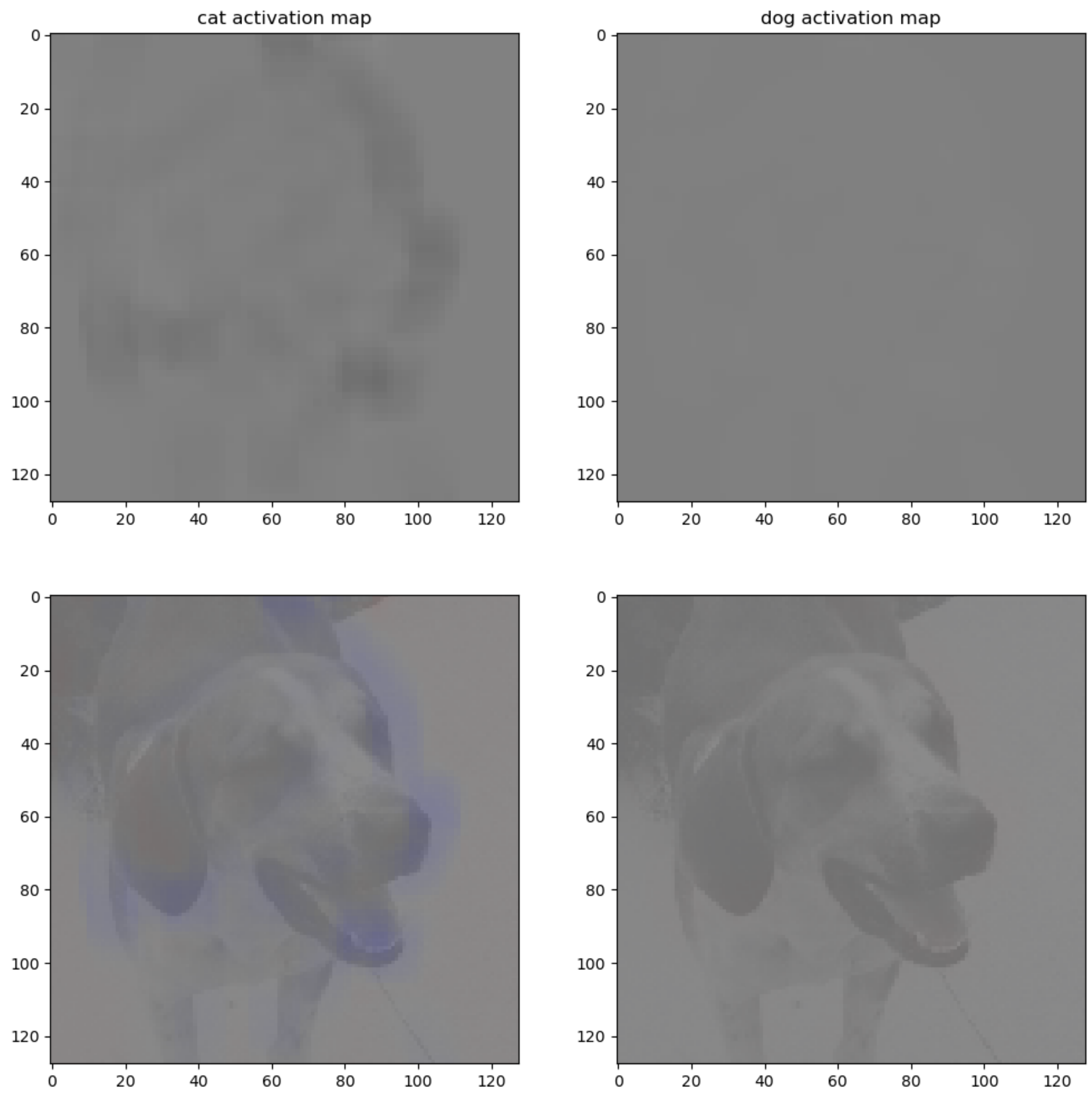
```
def cam_to_img(cam):

    cam_size = cam.shape[0]
    roi_size = 14
    roi_gap = 2
    roi = np.zeros((img_size, img_size, cam_size * cam_size))
    mask = np.full(roi.shape, 0)
    for i in range(0, cam_size):
        for j in range(0, cam_size):
            first_i = roi_gap * i
            last_i = first_i + roi_size
            first_j = roi_gap * j
            last_j = first_j + roi_size
            roi[first_i:last_i, first_j:last_j, i * cam_size + j] = cam[i, j]
            mask[first_i:last_i, first_j:last_j, i * cam_size + j] = 1

    return roi.sum(axis=2) / mask.sum(axis=2)

def mix_images(gray, color):

    result = np.empty((img_size, img_size, 3))
    result[:, :, 0] = 0.1 * color.mean(axis=2)
    result[:, :, 1] = result[:, :, 0]
    result[:, :, 2] = result[:, :, 0]
    result[:, :, 0] = result[:, :, 0] + 0.89 * gray
    result[:, :, 1] = result[:, :, 1] + 0.89 * (0.5 - np.abs(gray - 0.5))
    result[:, :, 2] = result[:, :, 2] + 0.89 * (1 - gray)
```

(continues on next page)

(continued from previous page)

```
    return result
```

```
cat_img = cam_to_img(cat_cam)
dog_img = cam_to_img(dog_cam)

cat_mix = mix_images(cat_img, img / 255)
dog_mix = mix_images(dog_img, img / 255)

fig, [[ax1, ax2], [ax3, ax4]] = plt.subplots(2, 2, figsize=(12, 12))
ax1.imshow(cat_img, cmap='gray', vmin=0, vmax=1)
ax2.imshow(dog_img, cmap='gray', vmin=0, vmax=1)
ax3.imshow(cat_mix)
ax4.imshow(dog_mix)
ax1.set_title('cat activation map')
ax2.set_title('dog activation map')
plt.show()
```



26.7 Improving CNN performance

So far we only considered the basics of CNNs. Now we discuss techniques for improving prediction quality and for decreasing training times. First we introduce the ideas, then we implement all techniques for better cats and dogs classification.

26.7.1 Data Augmentation

Prediction accuracy heavily depends on amount and variety of data available for training. Collecting more data is expensive. Thus, we could generate synthetic data from existing data. In case of image data we may rotate, scale, translate or distort the images to get new images showing identical objects in slightly different ways. This idea is known as *data augmentation* and increases the amount of data as well as the variety.

Kera provides several types of data augmentation (rotation, zoom, pan, brightness, flip, and some more). Activating this feature yields a stream of augmented images. Augmentation steps have to be incorporated to the model via preprocessing layers, see Image augmentation layers[514].

26.7.2 Pre-trained CNNs

CNNs have two major components: the feature extraction stack (convolutional and pooling layers) and the decision stack (dense layers for classification or regression). The task of the feature extraction stack is to automatically preprocess images resulting in a set of feature maps containing higher level information than just colored pixels. Based on this higher level information the decision stack predicts the targets.

With this two-step approach in mind we may use more powerful feature extraction. The feature extraction part is more or less the same for all object classification problems in image processing. Thus, we might use a feature extraction stack trained on much larger training data and with much more computational resources. Such pre-trained CNNs are available in the internet and Keras ships with some, too. See Keras Applications[515] for a list of pre-trained CNNs in Keras.

In Keras' documentation the feature extraction stack is called *convolutional base* and the decision stack is the *head* of the CNN. When loading a pre-trained model we have to decide wether to load the full model or only the convolutional base. If we do not use the pre-trained head, we have to specify the input shape for the network. This sounds a bit strange, but the convolutional base works for arbitrary input shapes and specifing a concrete shape fixes the output shape of the convolutional base. If we use the pre-trained head, then the output shape of the convolutional base has to fit the input shape of the head. Thus, the head determines the input shape of the CNN.

26.7.3 Other Minimization Algorithms

Up to now we only considered simple gradient descent. But there are much better algorithms for minimizing loss functions. Keras implements some of them and we should use them although at the moment we do not know what those algorithms do in detail. Advanced minimization techniques are not covered in this book.

[514] https://keras.io/api/layers/preprocessing_layers/image_augmentation/
[515] https://keras.io/api/applications/

26.7.4 Faster Preprocessing

Loading images from the disk and preprocessing them during training might slow down training. One solution is to load all images (including augmentation) to memory before training, but large memory is required. Another solution is to asynchronously load and preprocess data. That is, while the GPU does some calculations the CPU loads and preprocesses images. Keras and TensorFlow support such advanced techniques, but we will not cover them here.

26.7.5 Example

We consider object detection with cats and dogs again.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import tensorflow.keras as keras

data_path = '/home/jef19jdw/myfiles/datasets_teaching/ds2/catsdogs/data/'
```

```
2023-05-08 07:12:32.560037: I tensorflow/core/platform/cpu_feature_guard.
 ↪cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network␣
 ↪Library (oneDNN) to use the following CPU instructions in performance-
 ↪critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate␣
 ↪compiler flags.
```

To speed up training we would like to have all data in memory. Images have $128^2 = 16384$ pixels, each taking 3 bytes for the colors (one byte per channel) if color values are integers. For colors scaled to $[0, 1]$ we need 4 bytes per channel with `np.float32` as data type. Thus, we need 196608 bytes per image, say 200 kB. These are 5 images per MB or 5000 images per GB. Our data set has 25000 images and we could increase it to arbitrary size by data augmentation. Note that data augmentation is only useful for training data. Validation and test data should not be augmented. To save memory we do augmentation in real-time, that is, we only keep original training images in memory and generate batches of augmented images as needed.

To load all images we use Keras' methods for loading images from directories and inferring labels. Then we extract images and labels from the resulting data structure to ensure they are in memory (TensorFlow `Dataset` objects do not necessarily read all data to memory).

```
img_size = 128

train_data = keras.preprocessing.image_dataset_from_directory(
    data_path + 'labeled/train',
    label_mode = 'categorical',     # one-hot encoding with two columns
    batch_size=15000,     # load all images in one batch
    image_size=(img_size, img_size),
    validation_split=0.25,
    subset='training',
    seed=0
)
val_data = keras.preprocessing.image_dataset_from_directory(
    data_path + 'labeled/train',
    label_mode = 'categorical',
    batch_size=5000,
    image_size=(img_size, img_size),
    validation_split=0.25,
    subset='validation',
    seed=0     # same seed as for training
)
test_data = keras.preprocessing.image_dataset_from_directory(
```

(continues on next page)

(continued from previous page)

```
    data_path + 'labeled/test',
    label_mode = 'categorical',
    batch_size=5000,
    image_size=(img_size, img_size),
)

# extract images and labels from TensorFlow Dataset
train_images, train_labels = next(iter(train_data))
val_images, val_labels = next(iter(val_data))
test_images, test_labels = next(iter(test_data))
```

```
Found 20000 files belonging to 2 classes.
Using 15000 files for training.
```

```
2023-05-08 07:12:34.818592: E tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪driver.cc:267] failed call to cuInit: CUDA_ERROR_UNKNOWN: unknown error
2023-05-08 07:12:34.818623: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:169] retrieving CUDA diagnostic information for host: WHZ-46349
2023-05-08 07:12:34.818631: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:176] hostname: WHZ-46349
2023-05-08 07:12:34.818725: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:200] libcuda reported version is: 470.161.3
2023-05-08 07:12:34.818746: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:204] kernel reported version is: 470.161.3
2023-05-08 07:12:34.818752: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:310] kernel version seems to match DSO: 470.161.3
2023-05-08 07:12:34.818982: I tensorflow/core/platform/cpu_feature_guard.
↪cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network␣
↪Library (oneDNN) to use the following CPU instructions in performance-
↪critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate␣
↪compiler flags.
```

```
Found 20000 files belonging to 2 classes.
Using 5000 files for validation.
Found 5000 files belonging to 2 classes.
```

```
2023-05-08 07:12:45.792236: I tensorflow/core/kernels/data/shuffle_dataset_op.
↪cc:392] Filling up shuffle buffer (this may take a while): 14408 of 120000
2023-05-08 07:12:46.293722: I tensorflow/core/kernels/data/shuffle_dataset_op.
↪cc:417] Shuffle buffer filled.
2023-05-08 07:12:46.304312: W tensorflow/tsl/framework/cpu_allocator_impl.
↪cc:82] Allocation of 2949120000 exceeds 10% of free system memory.
2023-05-08 07:12:51.302941: W tensorflow/tsl/framework/cpu_allocator_impl.
↪cc:82] Allocation of 983040000 exceeds 10% of free system memory.
2023-05-08 07:12:56.156447: W tensorflow/tsl/framework/cpu_allocator_impl.
↪cc:82] Allocation of 983040000 exceeds 10% of free system memory.
```

Now we start to build the model. As mentioned above, data augmentation has to be implemented via preprocessing layers in the model.

Important: Due to a bug in TensorFlow 2.9 and above training of Keras models with preprocessing layers is extremely slow. See TensorFlow issue[516] for current discussion.

[516] https://github.com/tensorflow/tensorflow/issues/55639

```
model = keras.models.Sequential()
model.add(keras.Input(shape=(img_size, img_size, 3)))

#model.add(keras.layers.RandomFlip())
#model.add(keras.layers.RandomRotation(0.5))
#model.add(keras.layers.RandomZoom(0.2))
#model.add(keras.layers.RandomContrast(0.2))
#model.add(keras.layers.RandomBrightness(0.2, value_range=(0, 1)))
```

Note that preprocessing layers in Keras only are active during training. In evaluation and prediction phases they are skipped.

Next, we load a pre-trained convolutional base.

```
conv_base = keras.applications.Xception(
    include_top=False,
    input_shape=(img_size, img_size, 3)
)

conv_base.summary()
```

```
Model: "xception"
________________________________________________________________________________
 ↪____________________
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================
 input_2 (InputLayer)           [(None, 128, 128, 3  0           []
                                )]

 block1_conv1 (Conv2D)          (None, 63, 63, 32)   864         ['input_2[0][0]
 ↪']

 block1_conv1_bn (BatchNormaliz  (None, 63, 63, 32)  128         ['block1_
 ↪conv1[0][0]']
 ation)

 block1_conv1_act (Activation)  (None, 63, 63, 32)   0           ['block1_conv1_
 ↪bn[0][0]']

 block1_conv2 (Conv2D)          (None, 61, 61, 64)   18432       ['block1_conv1_
 ↪act[0][0]']

 block1_conv2_bn (BatchNormaliz  (None, 61, 61, 64)  256         ['block1_
 ↪conv2[0][0]']
 ation)

 block1_conv2_act (Activation)  (None, 61, 61, 64)   0           ['block1_conv2_
 ↪bn[0][0]']

 block2_sepconv1 (SeparableConv  (None, 61, 61, 128)  8768       ['block1_conv2_
 ↪act[0][0]']
 2D)

 block2_sepconv1_bn (BatchNorma  (None, 61, 61, 128)  512        ['block2_
 ↪sepconv1[0][0]']
 lization)

 block2_sepconv2_act (Activatio  (None, 61, 61, 128)  0          ['block2_
 ↪sepconv1_bn[0][0]']
 n)
```

(continues on next page)

(continued from previous page)

```
block2_sepconv2 (SeparableConv  (None, 61, 61, 128)  17536      ['block2_
↪sepconv2_act[0][0]']
2D)

block2_sepconv2_bn (BatchNorma  (None, 61, 61, 128)  512        ['block2_
↪sepconv2[0][0]']
lization)

conv2d (Conv2D)                (None, 31, 31, 128)  8192       ['block1_conv2_
↪act[0][0]']

block2_pool (MaxPooling2D)     (None, 31, 31, 128)  0          ['block2_
↪sepconv2_bn[0][0]']

batch_normalization (BatchNorm  (None, 31, 31, 128)  512        ['conv2d[0][0]
↪']
alization)

add (Add)                      (None, 31, 31, 128)  0          ['block2_
↪pool[0][0]',

                                                                 'batch_
↪normalization[0][0]']

block3_sepconv1_act (Activatio  (None, 31, 31, 128)  0          ['add[0][0]']
n)

block3_sepconv1 (SeparableConv  (None, 31, 31, 256)  33920      ['block3_
↪sepconv1_act[0][0]']
2D)

block3_sepconv1_bn (BatchNorma  (None, 31, 31, 256)  1024       ['block3_
↪sepconv1[0][0]']
lization)

block3_sepconv2_act (Activatio  (None, 31, 31, 256)  0          ['block3_
↪sepconv1_bn[0][0]']
n)

block3_sepconv2 (SeparableConv  (None, 31, 31, 256)  67840      ['block3_
↪sepconv2_act[0][0]']
2D)

block3_sepconv2_bn (BatchNorma  (None, 31, 31, 256)  1024       ['block3_
↪sepconv2[0][0]']
lization)

conv2d_1 (Conv2D)              (None, 16, 16, 256)  32768      ['add[0][0]']

block3_pool (MaxPooling2D)     (None, 16, 16, 256)  0          ['block3_
↪sepconv2_bn[0][0]']

batch_normalization_1 (BatchNo  (None, 16, 16, 256)  1024       ['conv2d_
↪1[0][0]']
rmalization)

add_1 (Add)                    (None, 16, 16, 256)  0          ['block3_
↪pool[0][0]',

                                                                 'batch_
↪normalization_1[0][0]']

block4_sepconv1_act (Activatio  (None, 16, 16, 256)  0          ['add_1[0][0]']
```

(continues on next page)

(continued from previous page)

```
n)

block4_sepconv1 (SeparableConv  (None, 16, 16, 728)  188672     ['block4_
↪sepconv1_act[0][0]']
2D)

block4_sepconv1_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block4_
↪sepconv1[0][0]']
lization)

block4_sepconv2_act (Activatio  (None, 16, 16, 728)  0          ['block4_
↪sepconv1_bn[0][0]']
n)

block4_sepconv2 (SeparableConv  (None, 16, 16, 728)  536536     ['block4_
↪sepconv2_act[0][0]']
2D)

block4_sepconv2_bn (BatchNorma  (None, 16, 16, 728)  2912       ['block4_
↪sepconv2[0][0]']
lization)

conv2d_2 (Conv2D)              (None, 8, 8, 728)    186368     ['add_1[0][0]']

block4_pool (MaxPooling2D)     (None, 8, 8, 728)    0          ['block4_
↪sepconv2_bn[0][0]']

batch_normalization_2 (BatchNo  (None, 8, 8, 728)   2912       ['conv2d_
↪2[0][0]']
rmalization)

add_2 (Add)                    (None, 8, 8, 728)    0          ['block4_
↪pool[0][0]',
                                                                'batch_
↪normalization_2[0][0]']

block5_sepconv1_act (Activatio  (None, 8, 8, 728)   0          ['add_2[0][0]']
n)

block5_sepconv1 (SeparableConv  (None, 8, 8, 728)   536536     ['block5_
↪sepconv1_act[0][0]']
2D)

block5_sepconv1_bn (BatchNorma  (None, 8, 8, 728)   2912       ['block5_
↪sepconv1[0][0]']
lization)

block5_sepconv2_act (Activatio  (None, 8, 8, 728)   0          ['block5_
↪sepconv1_bn[0][0]']
n)

block5_sepconv2 (SeparableConv  (None, 8, 8, 728)   536536     ['block5_
↪sepconv2_act[0][0]']
2D)

block5_sepconv2_bn (BatchNorma  (None, 8, 8, 728)   2912       ['block5_
↪sepconv2[0][0]']
lization)

block5_sepconv3_act (Activatio  (None, 8, 8, 728)   0          ['block5_
↪sepconv2_bn[0][0]']
```

(continues on next page)

(continued from previous page)

```
n)

block5_sepconv3 (SeparableConv  (None, 8, 8, 728)   536536      ['block5_
↪sepconv3_act[0][0]']
2D)

block5_sepconv3_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block5_
↪sepconv3[0][0]']
lization)

add_3 (Add)                    (None, 8, 8, 728)    0           ['block5_
↪sepconv3_bn[0][0]',

                                                                 'add_2[0][0]']

block6_sepconv1_act (Activatio  (None, 8, 8, 728)   0           ['add_3[0][0]']
n)

block6_sepconv1 (SeparableConv  (None, 8, 8, 728)   536536      ['block6_
↪sepconv1_act[0][0]']
2D)

block6_sepconv1_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block6_
↪sepconv1[0][0]']
lization)

block6_sepconv2_act (Activatio  (None, 8, 8, 728)   0           ['block6_
↪sepconv1_bn[0][0]']
n)

block6_sepconv2 (SeparableConv  (None, 8, 8, 728)   536536      ['block6_
↪sepconv2_act[0][0]']
2D)

block6_sepconv2_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block6_
↪sepconv2[0][0]']
lization)

block6_sepconv3_act (Activatio  (None, 8, 8, 728)   0           ['block6_
↪sepconv2_bn[0][0]']
n)

block6_sepconv3 (SeparableConv  (None, 8, 8, 728)   536536      ['block6_
↪sepconv3_act[0][0]']
2D)

block6_sepconv3_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block6_
↪sepconv3[0][0]']
lization)

add_4 (Add)                    (None, 8, 8, 728)    0           ['block6_
↪sepconv3_bn[0][0]',

                                                                 'add_3[0][0]']

block7_sepconv1_act (Activatio  (None, 8, 8, 728)   0           ['add_4[0][0]']
n)

block7_sepconv1 (SeparableConv  (None, 8, 8, 728)   536536      ['block7_
↪sepconv1_act[0][0]']
2D)

block7_sepconv1_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block7_
↪sepconv1[0][0]']
```

(continues on next page)

(continued from previous page)

```
lization)

block7_sepconv2_act (Activatio  (None, 8, 8, 728)   0           ['block7_
↪sepconv1_bn[0][0]']
n)

block7_sepconv2 (SeparableConv  (None, 8, 8, 728)   536536      ['block7_
↪sepconv2_act[0][0]']
2D)

block7_sepconv2_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block7_
↪sepconv2[0][0]']
lization)

block7_sepconv3_act (Activatio  (None, 8, 8, 728)   0           ['block7_
↪sepconv2_bn[0][0]']
n)

block7_sepconv3 (SeparableConv  (None, 8, 8, 728)   536536      ['block7_
↪sepconv3_act[0][0]']
2D)

block7_sepconv3_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block7_
↪sepconv3[0][0]']
lization)

add_5 (Add)                    (None, 8, 8, 728)    0           ['block7_
↪sepconv3_bn[0][0]',

                                                                 'add_4[0][0]']

block8_sepconv1_act (Activatio  (None, 8, 8, 728)   0           ['add_5[0][0]']
n)

block8_sepconv1 (SeparableConv  (None, 8, 8, 728)   536536      ['block8_
↪sepconv1_act[0][0]']
2D)

block8_sepconv1_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block8_
↪sepconv1[0][0]']
lization)

block8_sepconv2_act (Activatio  (None, 8, 8, 728)   0           ['block8_
↪sepconv1_bn[0][0]']
n)

block8_sepconv2 (SeparableConv  (None, 8, 8, 728)   536536      ['block8_
↪sepconv2_act[0][0]']
2D)

block8_sepconv2_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block8_
↪sepconv2[0][0]']
lization)

block8_sepconv3_act (Activatio  (None, 8, 8, 728)   0           ['block8_
↪sepconv2_bn[0][0]']
n)

block8_sepconv3 (SeparableConv  (None, 8, 8, 728)   536536      ['block8_
↪sepconv3_act[0][0]']
2D)
```

(continues on next page)

(continued from previous page)

```
block8_sepconv3_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block8_
↪sepconv3[0][0]']
lization)

add_6 (Add)                    (None, 8, 8, 728)    0           ['block8_
↪sepconv3_bn[0][0]',
                                                                 'add_5[0][0]']

block9_sepconv1_act (Activatio  (None, 8, 8, 728)   0           ['add_6[0][0]']
n)

block9_sepconv1 (SeparableConv  (None, 8, 8, 728)   536536      ['block9_
↪sepconv1_act[0][0]']
2D)

block9_sepconv1_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block9_
↪sepconv1[0][0]']
lization)

block9_sepconv2_act (Activatio  (None, 8, 8, 728)   0           ['block9_
↪sepconv1_bn[0][0]']
n)

block9_sepconv2 (SeparableConv  (None, 8, 8, 728)   536536      ['block9_
↪sepconv2_act[0][0]']
2D)

block9_sepconv2_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block9_
↪sepconv2[0][0]']
lization)

block9_sepconv3_act (Activatio  (None, 8, 8, 728)   0           ['block9_
↪sepconv2_bn[0][0]']
n)

block9_sepconv3 (SeparableConv  (None, 8, 8, 728)   536536      ['block9_
↪sepconv3_act[0][0]']
2D)

block9_sepconv3_bn (BatchNorma  (None, 8, 8, 728)   2912        ['block9_
↪sepconv3[0][0]']
lization)

add_7 (Add)                    (None, 8, 8, 728)    0           ['block9_
↪sepconv3_bn[0][0]',
                                                                 'add_6[0][0]']

block10_sepconv1_act (Activati  (None, 8, 8, 728)   0           ['add_7[0][0]']
on)

block10_sepconv1 (SeparableCon  (None, 8, 8, 728)   536536      ['block10_
↪sepconv1_act[0][0]']
v2D)

block10_sepconv1_bn (BatchNorm  (None, 8, 8, 728)   2912        ['block10_
↪sepconv1[0][0]']
alization)

block10_sepconv2_act (Activati  (None, 8, 8, 728)   0           ['block10_
↪sepconv1_bn[0][0]']
on)
```

(continues on next page)

(continued from previous page)

```
block10_sepconv2 (SeparableCon  (None, 8, 8, 728)   536536      ['block10_
↪sepconv2_act[0][0]']
v2D)

block10_sepconv2_bn (BatchNorm  (None, 8, 8, 728)   2912        ['block10_
↪sepconv2[0][0]']
alization)

block10_sepconv3_act (Activati  (None, 8, 8, 728)   0           ['block10_
↪sepconv2_bn[0][0]']
on)

block10_sepconv3 (SeparableCon  (None, 8, 8, 728)   536536      ['block10_
↪sepconv3_act[0][0]']
v2D)

block10_sepconv3_bn (BatchNorm  (None, 8, 8, 728)   2912        ['block10_
↪sepconv3[0][0]']
alization)

add_8 (Add)                    (None, 8, 8, 728)    0           ['block10_
↪sepconv3_bn[0][0]',

                                                                 'add_7[0][0]']

block11_sepconv1_act (Activati  (None, 8, 8, 728)   0           ['add_8[0][0]']
on)

block11_sepconv1 (SeparableCon  (None, 8, 8, 728)   536536      ['block11_
↪sepconv1_act[0][0]']
v2D)

block11_sepconv1_bn (BatchNorm  (None, 8, 8, 728)   2912        ['block11_
↪sepconv1[0][0]']
alization)

block11_sepconv2_act (Activati  (None, 8, 8, 728)   0           ['block11_
↪sepconv1_bn[0][0]']
on)

block11_sepconv2 (SeparableCon  (None, 8, 8, 728)   536536      ['block11_
↪sepconv2_act[0][0]']
v2D)

block11_sepconv2_bn (BatchNorm  (None, 8, 8, 728)   2912        ['block11_
↪sepconv2[0][0]']
alization)

block11_sepconv3_act (Activati  (None, 8, 8, 728)   0           ['block11_
↪sepconv2_bn[0][0]']
on)

block11_sepconv3 (SeparableCon  (None, 8, 8, 728)   536536      ['block11_
↪sepconv3_act[0][0]']
v2D)

block11_sepconv3_bn (BatchNorm  (None, 8, 8, 728)   2912        ['block11_
↪sepconv3[0][0]']
alization)

add_9 (Add)                    (None, 8, 8, 728)    0           ['block11_
↪sepconv3_bn[0][0]',
```

(continues on next page)

(continued from previous page)

```
                                                                      'add_8[0][0]']

 block12_sepconv1_act (Activati  (None, 8, 8, 728)   0           ['add_9[0][0]']
 on)

 block12_sepconv1 (SeparableCon  (None, 8, 8, 728)   536536      ['block12_
↪sepconv1_act[0][0]']
 v2D)

 block12_sepconv1_bn (BatchNorm  (None, 8, 8, 728)   2912        ['block12_
↪sepconv1[0][0]']
 alization)

 block12_sepconv2_act (Activati  (None, 8, 8, 728)   0           ['block12_
↪sepconv1_bn[0][0]']
 on)

 block12_sepconv2 (SeparableCon  (None, 8, 8, 728)   536536      ['block12_
↪sepconv2_act[0][0]']
 v2D)

 block12_sepconv2_bn (BatchNorm  (None, 8, 8, 728)   2912        ['block12_
↪sepconv2[0][0]']
 alization)

 block12_sepconv3_act (Activati  (None, 8, 8, 728)   0           ['block12_
↪sepconv2_bn[0][0]']
 on)

 block12_sepconv3 (SeparableCon  (None, 8, 8, 728)   536536      ['block12_
↪sepconv3_act[0][0]']
 v2D)

 block12_sepconv3_bn (BatchNorm  (None, 8, 8, 728)   2912        ['block12_
↪sepconv3[0][0]']
 alization)

 add_10 (Add)                   (None, 8, 8, 728)    0           ['block12_
↪sepconv3_bn[0][0]',
                                                                  'add_9[0][0]']

 block13_sepconv1_act (Activati  (None, 8, 8, 728)   0           ['add_10[0][0]
↪']
 on)

 block13_sepconv1 (SeparableCon  (None, 8, 8, 728)   536536      ['block13_
↪sepconv1_act[0][0]']
 v2D)

 block13_sepconv1_bn (BatchNorm  (None, 8, 8, 728)   2912        ['block13_
↪sepconv1[0][0]']
 alization)

 block13_sepconv2_act (Activati  (None, 8, 8, 728)   0           ['block13_
↪sepconv1_bn[0][0]']
 on)

 block13_sepconv2 (SeparableCon  (None, 8, 8, 1024)  752024      ['block13_
↪sepconv2_act[0][0]']
 v2D)
```

(continues on next page)

(continued from previous page)

```
 block13_sepconv2_bn (BatchNorm  (None, 8, 8, 1024)  4096        ['block13_
↪sepconv2[0][0]']
 alization)

 conv2d_3 (Conv2D)              (None, 4, 4, 1024)  745472      ['add_10[0][0]
↪']

 block13_pool (MaxPooling2D)    (None, 4, 4, 1024)  0           ['block13_
↪sepconv2_bn[0][0]']

 batch_normalization_3 (BatchNo  (None, 4, 4, 1024)  4096        ['conv2d_
↪3[0][0]']
 rmalization)

 add_11 (Add)                   (None, 4, 4, 1024)  0           ['block13_
↪pool[0][0]',
                                                                  'batch_
↪normalization_3[0][0]']

 block14_sepconv1 (SeparableCon  (None, 4, 4, 1536)  1582080     ['add_11[0][0]
↪']
 v2D)

 block14_sepconv1_bn (BatchNorm  (None, 4, 4, 1536)  6144        ['block14_
↪sepconv1[0][0]']
 alization)

 block14_sepconv1_act (Activati  (None, 4, 4, 1536)  0           ['block14_
↪sepconv1_bn[0][0]']
 on)

 block14_sepconv2 (SeparableCon  (None, 4, 4, 2048)  3159552     ['block14_
↪sepconv1_act[0][0]']
 v2D)

 block14_sepconv2_bn (BatchNorm  (None, 4, 4, 2048)  8192        ['block14_
↪sepconv2[0][0]']
 alization)

 block14_sepconv2_act (Activati  (None, 4, 4, 2048)  0           ['block14_
↪sepconv2_bn[0][0]']
 on)

================================================================================================
Total params: 20,861,480
Trainable params: 20,806,952
Non-trainable params: 54,528
________________________________________________________________________________
↪__________________
```

We see that there are new layer types: separable convolutions and batch normalization. Separable convolutions are a special case of usual convolution allowing for more efficient computation by restricting to specially structured filters. Batch normalization is a kind of rescaling layer outputs. The more important observation is the output shape: 4x4x2048. That is, we obtain 2048 feature maps each of size 4x4. This is where we connect our decision stack.

Models in Keras behave like layers (the `Model` class inherits from `Layer`). Thus, we may add the pre-trained convolutional base as layer to our model.

When using pre-trained models, data preprocessing has to be done in exactly the same way as has been done in training. For each pre-trained model in Keras there is a `preprocess_input`[517] function doing necessary preprocessing.

[517] https://www.tensorflow.org/api_docs/python/tf/keras/applications/xception/preprocess_input

To apply this function to all data flowing through our model we use a Lambda layer[518].

```
model.add(keras.layers.Lambda(keras.applications.xception.preprocess_input))
model.add(conv_base)
```

To complete the model we add dense layers.

```
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation='relu', name='dense1'))
model.add(keras.layers.Dense(10, activation='relu', name='dense2'))
model.add(keras.layers.Dense(2, activation='sigmoid', name='out'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lambda (Lambda)             (None, 128, 128, 3)       0

 xception (Functional)       (None, 4, 4, 2048)        20861480

 flatten (Flatten)           (None, 32768)             0

 dense1 (Dense)              (None, 10)                327690

 dense2 (Dense)              (None, 10)                110

 out (Dense)                 (None, 2)                 22

=================================================================
Total params: 21,189,302
Trainable params: 21,134,774
Non-trainable params: 54,528
_________________________________________________________________
```

Before we start training we have to tell Keras to keep the weights of the convolutional base constant. We simply have to set the layer's `trainable` attribute to `False`:

```
model.get_layer('xception').trainable = False
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lambda (Lambda)             (None, 128, 128, 3)       0

 xception (Functional)       (None, 4, 4, 2048)        20861480

 flatten (Flatten)           (None, 32768)             0

 dense1 (Dense)              (None, 10)                327690

 dense2 (Dense)              (None, 10)                110

 out (Dense)                 (None, 2)                 22

```

(continues on next page)

[518] https://keras.io/api/layers/core_layers/lambda/

(continued from previous page)

```
=================================================================
Total params: 21,189,302
Trainable params: 327,822
Non-trainable params: 20,861,480
_________________________________________________________________
```

```
model.compile(loss='categorical_crossentropy', metrics=['categorical_accuracy'])
```

Now training can be started.

```
loss = []
val_loss = []
acc = []
val_acc = []
```

```
history = model.fit(
    train_images, train_labels,
    epochs=5,
    validation_data=(val_images, val_labels),
    batch_size=100
)

loss.extend(history.history['loss'])
val_loss.extend(history.history['val_loss'])
acc.extend(history.history['categorical_accuracy'])
val_acc.extend(history.history['val_categorical_accuracy'])
```

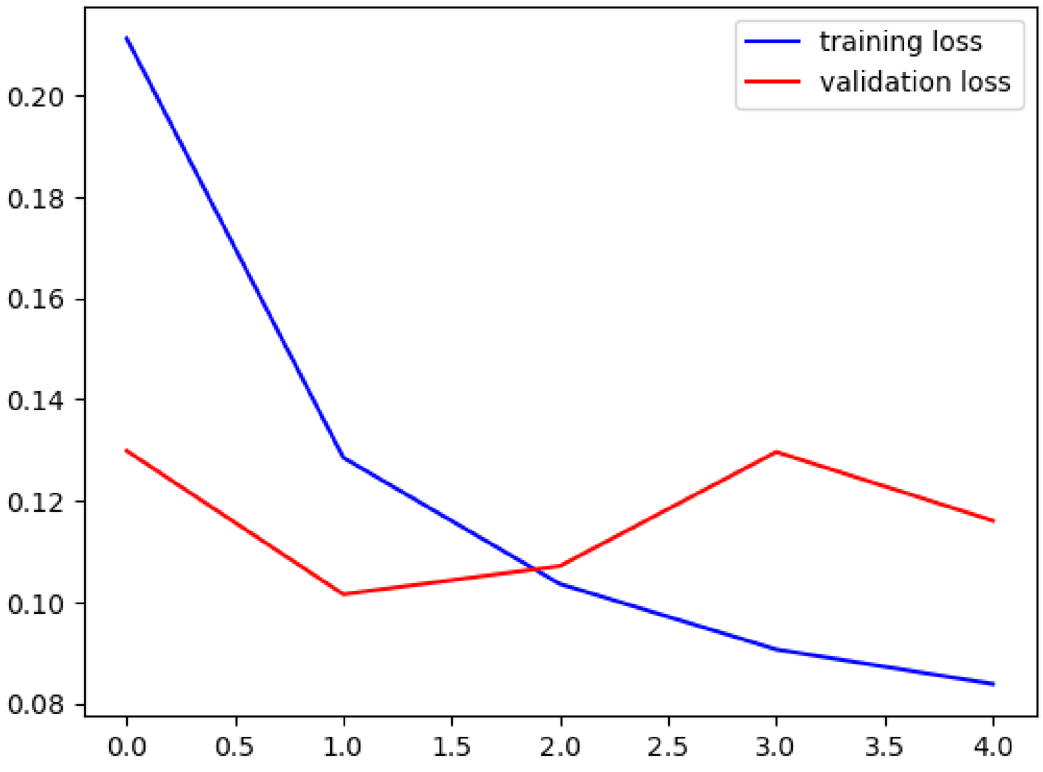
```
Epoch 1/5
150/150 [==============================] - 594s 4s/step - loss: 0.2113 -␣
↪categorical_accuracy: 0.9325 - val_loss: 0.1298 - val_categorical_accuracy: 0.
↪9496
Epoch 2/5
150/150 [==============================] - 577s 4s/step - loss: 0.1285 -␣
↪categorical_accuracy: 0.9529 - val_loss: 0.1015 - val_categorical_accuracy: 0.
↪9592
Epoch 3/5
150/150 [==============================] - 571s 4s/step - loss: 0.1036 -␣
↪categorical_accuracy: 0.9629 - val_loss: 0.1071 - val_categorical_accuracy: 0.
↪9580
Epoch 4/5
150/150 [==============================] - 587s 4s/step - loss: 0.0906 -␣
↪categorical_accuracy: 0.9661 - val_loss: 0.1296 - val_categorical_accuracy: 0.
↪9604
Epoch 5/5
150/150 [==============================] - 556s 4s/step - loss: 0.0838 -␣
↪categorical_accuracy: 0.9673 - val_loss: 0.1161 - val_categorical_accuracy: 0.
↪9592
```

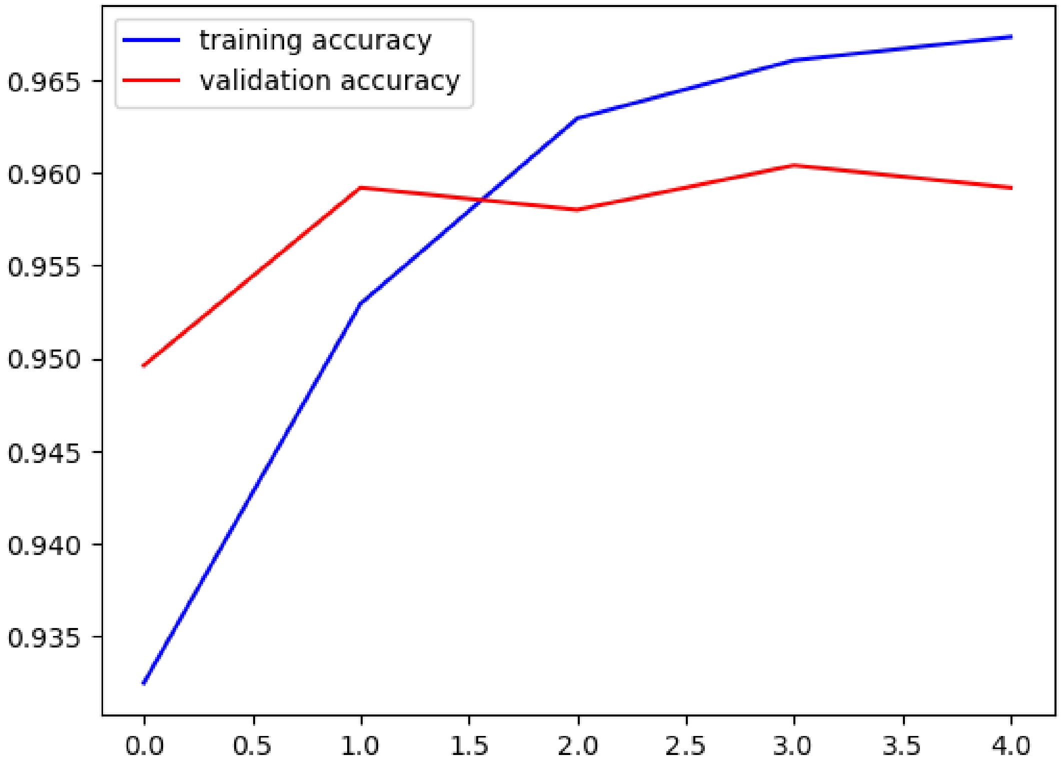
```
fig, ax = plt.subplots()
ax.plot(loss, '-b', label='training loss')
ax.plot(val_loss, '-r', label='validation loss')
ax.legend()
plt.show()

fig, ax = plt.subplots()
ax.plot(acc, '-b', label='training accuracy')
ax.plot(val_acc, '-r', label='validation accuracy')
```

(continues on next page)

(continued from previous page)

```
ax.legend()
plt.show()
```





```
model.save('cnnmodelimproved')
```

```
WARNING:absl:Found untraced functions such as _update_step_xla, _jit_compiled_
↪convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _
↪jit_compiled_convolution_op while saving (showing 5 of 41). These functions␣
↪will not be directly callable after loading.
```

```
INFO:tensorflow:Assets written to: cnnmodelimproved/assets
```

```
INFO:tensorflow:Assets written to: cnnmodelimproved/assets
```

```
test_loss, test_metric = model.evaluate(x=test_images, y=test_labels)
print(test_metric)
```

```
157/157 [==============================] - 136s 860ms/step - loss: 0.1460 -␣
↪categorical_accuracy: 0.9516
0.9516000151634216
```

CHAPTER

TWENTYSEVEN

DECISION TREES

Decision trees are a class of relatively simple yet powerful machine learning methods suited for both regression and classification.

Related projects:

27.1 Basics

Decision trees, also known as *classification and regression trees (CART)*, are a class of machine learning techniques based on tree data structures.

27.1.1 Decision Tree Structure

A tree is an abstract structure made of *nodes* and *edges*. Nodes are connected by edges. Each node has exactly one parent node and may have several child nodes. There is one node without parent node, called the *root node*. Nodes without children are *leaves*. A subtree is a node together with all its descendants (children, grandchildren and so on).

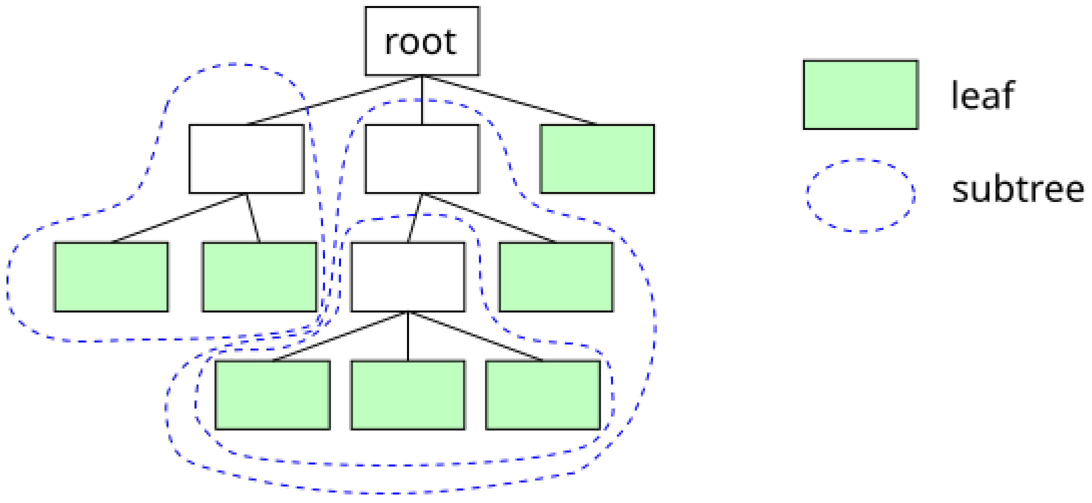


Fig. 27.1: A tree consists of nodes and edges. Some nodes are special, like the root node and the leaves.

In a decision tree each node represents a condition on a feature in a learning task. A feature vector is passed through the tree starting at the root and finally arriving at one of the tree's leaves, representing the predictions. At each node corresponding condition is evaluated for the concrete feature vector. Based on the result the feature vector is passed on to one of the node's child. Evaluating a node may have several possible outcomes, but often conditions are either satisfied or not, yielding a binary tree (two children per node).

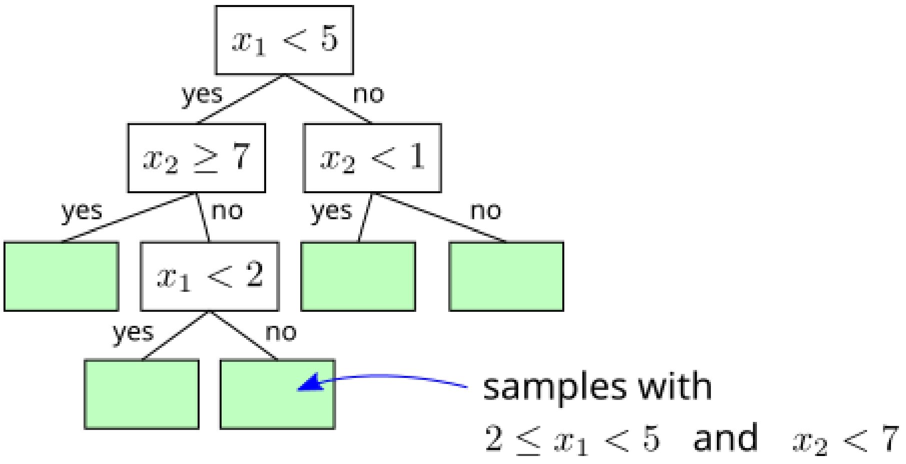


Fig. 27.2: Each node in a decision tree, which is not a leaf, represents a condition on the samples passed down the tree.

The training phase consists of building a decision tree and the prediction phase consists of passing feature vectors through the tree. Prediction is fast and simple, but for training we have to answer difficult questions:

- Which features should we consider in the nodes? Which one first?
- Which conditions should we check on the features?
- How large should the tree be?

Major advantages of decision trees:

- They can be applied for arbitrary data types including categorical data.
- They not only yield predictions but also a list of human readable decisions leading to that prediction.

27.1.2 Training

Training a decision tree is a relatively complex task. We start general remarks and then provide concrete algorithms in subsequent sections.

Growing a Tree

There exist many techniques to grow decision trees. The overall procedure is as follows:

1. Start with a tree containing only the root node.
2. Select one of the features and a condition involving only the selected feature.
3. Split the training data set according to the condition into disjoint subsets.
4. For each subset create a child node.
5. Process each child node in the same way as the root node (that is, go to 2), but with the full data set replaced by the subset corresponding to the child node.

This splitting procedure is repeated until all leaves satisfy some stopping criterion. Common stopping criteria are:

- variance in the leaf is small,
- only few samples correspond to leaf,

- predefined depth of tree reached,
- maximum number of leaves reached.

After stopping the growth process, each leave corresponds to a small set of training samples (the ones satisfiying all conditions on the path to the leaf). The prediction corresponding to a leave is

- the mean of all targets of the samples in that set in case of regression,
- the class most samples in that st belong to in case of classification.

For numerical features conditions are formulated as a single inequality, so the feature's range is splitted into two disjoint intervals. Since we only have finitely many samples, there are at most as many sensible splitting points as we have samples.

For categorical features with few categories splitting into as many child nodes as there are categories is feasible. Else some condition with binary result should be considered. There are at most $2^{\text{number of categories}}$ different conditions with binary result for a categorical feature.

Choosing features and conditions is the hard part. There exist many techniques to do this. Some prominent ones will be considered below.

Pruning

Small trees aren't able to represent complex hypotheses. Large trees tend to overfit training data. Thus, growth of trees has to be stopped at the right moment by some stopping criterion (see above). A more complex regularization technique is *pruning*. Here we grow a very complex tree, which overfits training data, and then remove some nodes together with all descendants. Removing a node means that we replace it by a leaf as if splitting had never happend. We try to remove nodes which can be removed without effecting prediction accuracy on a validation data set too much. Conrete pruning algorithms will be considered below.

27.2 Regression Trees

Here we consider a concrete splitting (*variance reduction*) for growing regression trees as well as a concrete pruning algorithm (*cost-complexity pruning*) for regression trees.

27.2.1 Variance reduction

Variance reduction aims at splitting nodes in a way yielding children with small variance. Nodes with small variance yield more precise predictions than nodes with large variance (remember: prediction = mean of targets of all training samples corresponding to the node).

Consider one node and the corresponding subset S of the training data set $\{(x_1, y_1, \ldots, (x_n, y_n)\}$. Let

$$I_S := \{k \in \{1, \ldots, n\} : (x_k, y_k) \in S\}$$

be the index set holding all indices of samples in S and denote mean and variance of targets in S by

$$m(S) := \frac{1}{|S|} \sum_{k \in I_S} y_k$$

and

$$v(S) := \frac{1}{|S|} \sum_{k \in I_S} \bigl(y_k - m(S)\bigr)^2,$$

respectively.

Given a split $S = L \cup R$ into disjoint subsets L and R we look at the variances $v(L)$ and $v(R)$. For an arbitrary split it's not clear whether $v(L)$ and $v(R)$ are lower or higher than the original $v(S)$.

Example

Let $y_1 = 1$, $y_2 = 5$, $y_3 = 2$, $y_4 = 4$. Calculate $v(S)$ for $S = \{1, 2, 3, 4\}$ and then $v(L)$ and $v(R)$ for following splits:

- $L = \{1, 3\}, \quad R = \{2, 4\}$,
- $L = \{1, 2\}, \quad R = \{3, 4\}$.

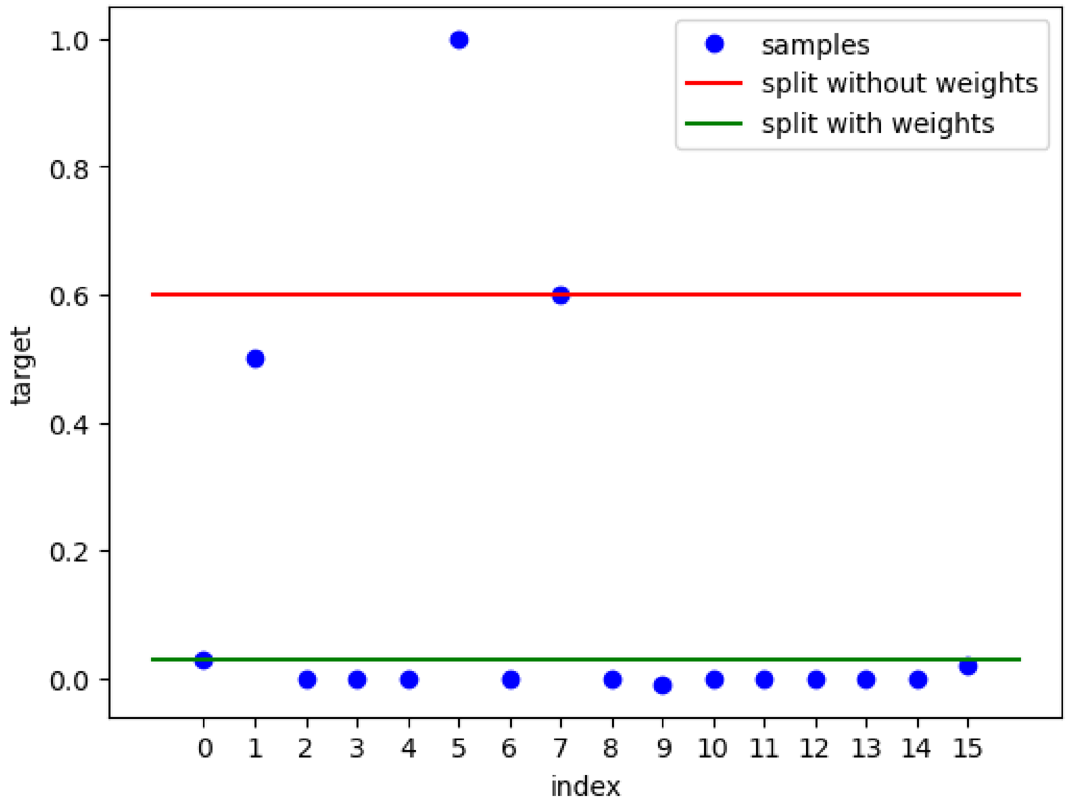
Since we aim at low variance in the tree's leaves we would like to choose a split which minimizes both variances $v(L)$ and $v(R)$ at once. Optimization problems with multiple objective functions are hard to handle. So we look for an objective combining both variances. We simply could use the sum $v(L) + v(R)$, but then small subsets with low variance have the same weight as large subsets with low variance. Leafs corresponding to large subsets with low variance are good because they yield good predictions on many training samples, whereas leaves corresponding to small subsets do not matter so much. A better idea for a joint objective is a weighted sum of variances with weights representing subset sizes:

$$\frac{|L|}{|S|}\,v(L) + \frac{|R|}{|S|}\,v(R).$$

Here is a script for comparing splits resulting from joint variance without and with weights:

```python
import numpy as np
import matplotlib.pyplot as plt
```

```python
# targets of samples in set S
y = np.array([0.03, 0.5, 0, 0, 0, 1, 0, 0.6, 0, -0.01, 0, 0, 0, 0, 0, 0.02])

# all splitting points
splits = np.sort(np.unique(y))

# empty arrays to be filled with v(L), v(R), joint variance without/with weights
vL = np.empty(splits.size - 1)
vR = np.empty(splits.size - 1)
jv = np.empty(splits.size - 1)
jvw = np.empty(splits.size - 1)

# calculate joint variances
for k in range(0, splits.size - 1):
    yL = y[y <= splits[k]]    # targets in subset L
    yR = y[y > splits[k]]     # targets in subset R
    vL[k] = np.var(yL)
    vR[k] = np.var(yR)
    jv[k] = vL[k] + vR[k]
    jvw[k] = yL.size / y.size * vL[k] + yR.size / y.size * vR[k]

# splitting points for both variants
jv_split = splits[jv.argmin()]
jvw_split = splits[jvw.argmin()]

# plot targets and splitting points
fig, ax = plt.subplots()
ax.plot(y, 'ob', label='samples')
ax.plot([-1, y.size], [jv_split, jv_split], '-r', label='split without weights')
ax.plot([-1, y.size], [jvw_split, jvw_split], '-g', label='split with weights')
ax.legend()
ax.set_xticks(range(y.size))
ax.set_xlabel('index')
ax.set_ylabel('target')
plt.show()
```



Note, that the formula for joint variance can be reduced to

$$\frac{|L|}{|S|}\,v(L)+\frac{|R|}{|S|}\,v(R)=\frac{1}{|S|}\left(\sum_{k\in I_L}\big(y_k-m(L)\big)^2+\sum_{k\in I_R}\big(y_k-m(R)\big)^2\right).$$

If we take into account that the mean is the best constant approximation to the targets of a set of samples, we immediately see

$$\sum_{k\in I_L}\big(y_k-m(L)\big)^2\leq\sum_{k\in I_L}\big(y_k-m(S)\big)^2\qquad\text{and}\qquad\sum_{k\in I_R}\big(y_k-m(R)\big)^2\leq\sum_{k\in I_R}\big(y_k-m(S)\big)^2.$$

Thus,

$$\frac{|L|}{|S|}\,v(L)+\frac{|R|}{|S|}\,v(R)\leq v(S),$$

that is, the joint variance of the child nodes is smaller than the parent node's variance. This justifies calling the splitting rule *variance reduction*.

27.2.2 Cost-Complexity Pruning

Hint: There exist several variants of *cost-complexity* pruning. Here we describe the variant implemented in Scikit-Learn.

In a trained tree each leaf corresponds to a subset S of training samples. For each leaf (or corresponding subset S) we may compute the variance $v(\text{leaf})$ in the same way as $v(S)$ above. For non-leaf nodes we may compute the joint variance of all leafs of the nodes subtree:

$$v(\text{subtree}):=\sum_{\substack{\text{leaves of}\\\text{subtree}}}\frac{|\text{leaf}|}{|\text{subtree}|}\,v(\text{leaf}),$$

where $|\cdots|$ denotes the number of training samples corresponding to a leaf or to all leafs of subtree, respectively.

Joint variance of a tree is closely related to prediction quality. More precisely, joint variance is the tree's prediction error (MSE) on the training data.

Above we already met the important inequality

$$v(\text{subtree}) \leq v(\text{subtree replaced by corresponding leaf}).$$

If we remove a node (and all its descendants) from the tree, joint variance of the whole tree increases or remains unchanged. If it remains unchanged, the now smaller tree has same prediction quality as the larger one. If joint variance increases, we have to decide if increase is not too large compared to decrease of complexity. Complexity of a tree can be expressed as the total number of leaves. The trade-off between joint variance and complexity can be expressed by the *cost-complexity measure (CCM)*:

$$CCM(\text{subtree}) := v(\text{subtree}) + \alpha \cdot \text{leaves in subtree}.$$

The first summand expresses the prediction error, the second the complexity of the subtree. The regularization parameter α controls the trade-off between error and complexity. CCM of a leaf (a subtree containing only one node) is

$$CCM(\text{leaf}) = v(\text{leaf}) + \alpha.$$

If we replace a subtree by a leaf the first summand (error) in CCM increases and the second (complexity) decreases to one. For $\alpha = 0$ we have $CCM(\text{subtree}) \leq CMM(\text{subtree replaced by leaf})$, because prediction error dominates CCM. For very large α we have $CCM(\text{subtree}) > CCM(\text{subtree replaced by leaf})$, because complexity dominates CCM.

Cost-complexity pruning for each non-leaf node replaces the corresponding subtree by a leaf if $CCM(\text{subtree}) > CCM(\text{subtree replaced by leaf})$, that is, if the leaf will have smaller cost-complexity measure than the subtree. For $\alpha = 0$ nothing will be pruned. The larger α the smaller the resulting tree will be. The hyperparameter α has to be chosen carefully like every other hyperparameter, for instance by comparing prediction quality on training and validation data sets.

Cost-complexity pruning sometimes is described by the following equivalent formulation:

For each non-leaf node look for α such that

$$CCM(\text{subtree}) = CCM(\text{subtree replaced by leaf}).$$

Such an α is called *effective* α. The formula is

$$\alpha = \frac{v(\text{subtree replaced by leaf}) - v(\text{subtree})}{\text{leaves in subtree} - 1}$$

For the effective α CCM does not change if we replace a subtree by a leaf. Small effective α indicates that prediction error changes only slightly while complexity is changed much more when replacing the subtree. Based on that observation we compute effective α for all subtrees and remove subtrees with effective α below some predefined upper bound. That upper bound is identical to the hyperparameter α above.

27.2.3 Pruning versus Penalization

Formula for CCM are very similar to formula for regularizing loss function based learning methods (linear regression, ANNs). For loss function based methods we looked for minimizers of

$$\text{loss}\big(\text{model(inputs), targets}\big) + \alpha \cdot \text{penalty(inputs)} \to \min_{\text{model}},$$

over a certain class of models. Following this idea, in regression tree learning we could ask for a tree minimizing

$$\text{loss}\big(\text{model(inputs), targets}\big) + \alpha \cdot \text{number of leaves} \to \min_{\text{model}}$$

over the set of all regression trees. Formulating such an optimization problem is not hard, but how to solve it? The objective is not differentiable and the search space (set of all trees) is extremely large.

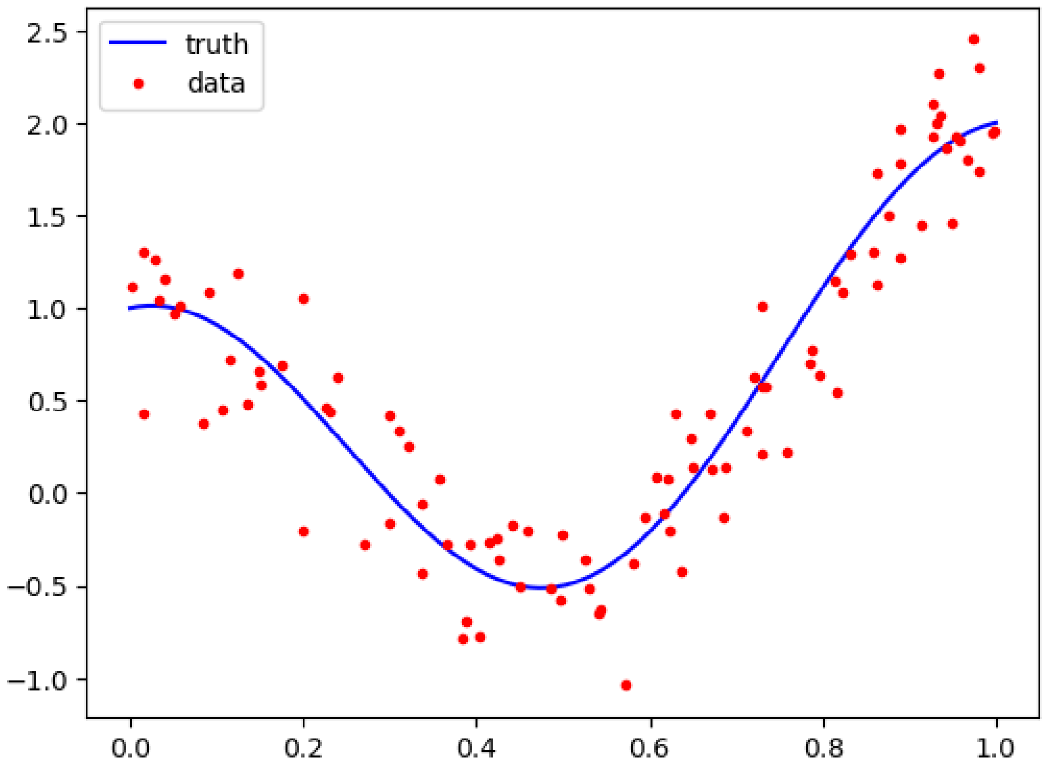
Cost-complexity pruning solves this optimization problem for a much smaller search space. The pruned tree minimizes the objective over the set of all trees which can be obtained from the unpruned tree by removing nodes. The regularization parameter α in the objective is the lower bound for effective α values to keep. In this sense cost-complexity pruning fits well into the usual regularization framework.

27.2.4 Regression Trees with Scikit-Learn

Scikit-Learn implements regression trees in `DecisionTreeRegressor`[519].

```python
import numpy as np
import matplotlib.pyplot as plt

import sklearn.tree as tree

rng = np.random.default_rng(0)
```

```python
def truth(x):
    return x + np.cos(2 * np.pi * x)

xmin = 0
xmax = 1
x = np.linspace(xmin, xmax, 200)

n = 100     # number of data points to generate
noise_level = 0.3     # standard deviation of artificial noise

# simulate data
X = (xmax - xmin) * rng.random((n, 1)) + xmin
y = truth(X).reshape(-1) + noise_level * rng.standard_normal(n)

# plot truth and data
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='truth')
ax.plot(X.reshape(-1), y, 'or', markersize=3, label='data')
ax.legend()
plt.show()
```

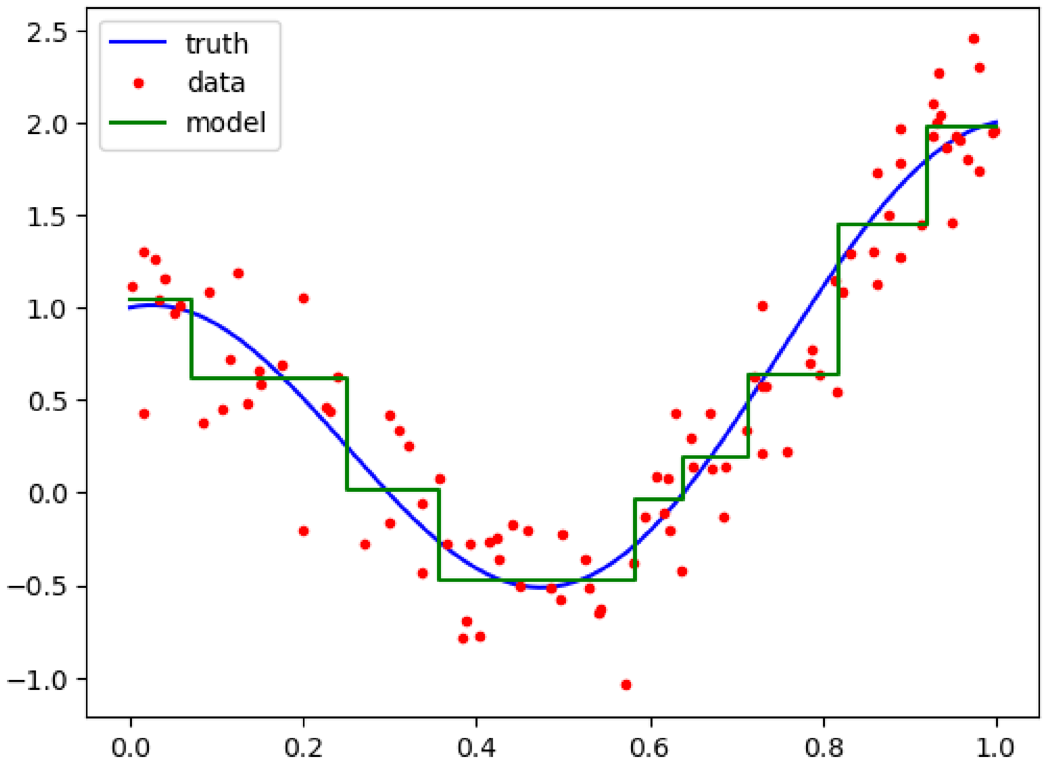


[519] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

`DecisionTreeRegressor` takes several parameters for stopping growth of the tree and also supports cost-complexity pruning. For the latter the `ccp_alpha` parameter has to be specified. Nodes with smaller effective α will be removed. Scikit-Learn's default values for stopping criteria lead to trees with one training sample per leaf, that is, to maximum complexity.

To find good splits, Scikit-Learn uses variance reduction by default, but other techniques are available (parameter `criterion`). Further, instead of considering all possible splits, we may reduce computation time by considering fewer features or fewer splitting points (parameters `splitter` and `max_features`).

```
# regression
reg = tree.DecisionTreeRegressor(ccp_alpha=0.004)
reg.fit(X, y)

# get hypothesis for plotting
y_reg = reg.predict(x.reshape(-1, 1))

# plot truth, data, hypothesis
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='truth')
ax.plot(X.reshape(-1), y, 'or', markersize=3, label='data')
ax.step(x, y_reg, '-g', label='model')
ax.legend()
plt.show()
```

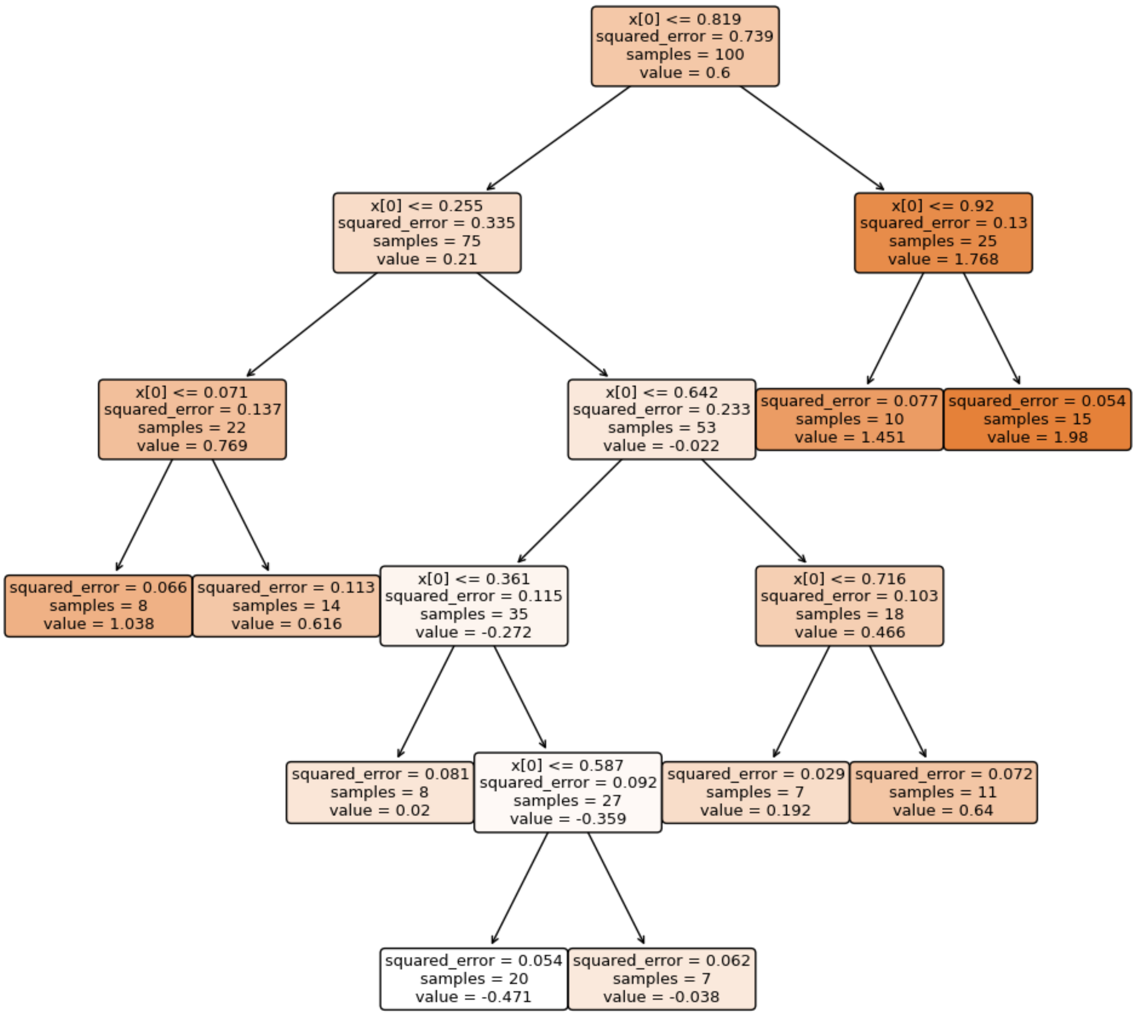


Regression trees always yield piecewise constant hypotheses. The number of plateaus corresponds to the number of leaves.

Scikit-Learn provides the `plot_tree`[520] function to visualize decision trees.

```
fig, ax = plt.subplots(figsize=(12, 12))
tree.plot_tree(reg, ax=ax, filled=True, rounded=True)
plt.show()
```

[520] https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html



27.3 Classification Trees

Classification trees work much the same like regression trees, but there are some more standard splitting rules for growing a tree.

27.3.1 Splitting Strategies

All splitting rules described here define some error measure and choose the split which minimizes the error.

Splitting by Missclassification Rate

For each possible split we determine the number of missclassified samples in each resulting leaf. That is, we count all samples not belonging to their leaf's majority class (the leaf's prediction). The lower the resulting number the better the split.

Note that sometimes different splits have identical missclassification rate, but from manual inspection we would clearly favor one of them.

Example

Consider 200 samples, 100 belonging to class A, 100 belonging to class B, and two splits

$$\text{25 A / 75 B} \quad \text{and} \quad \text{75 A / 25 B}, \qquad\qquad \text{50 A / 100 B} \quad \text{and} \quad \text{50 A / 0 B}.$$

Both splits missclassify 50 samples, but the second one has a pure leaf and, thus, should be preferred. For the first split we would need at least two further splitting steps to make all leaves pure. For the second split one further split might suffice.

Another issue splitting by missclassifications is that there might be no split that decreases missclassifiction, suggesting to stop the growth process to avoid overfitting. But other measures (see below) justify further splitting.

Example

Consider three samples with feature values $x_1=1$, $x_2=2$, $x_3=3$ belonging to classes A, B, A, respectively. Then the original node and the two possible splits

$$\text{A} \quad \text{and} \quad \text{BA}, \qquad\qquad \text{AB} \quad \text{and} \quad \text{A}$$

have exactly one missclassified sample. Thus, splitting does not reduce missclassification.

Splitting by Gini Impurity

Gini impurity of a leaf is the probability that two samples randomly chosen from the subset corresponding to the leaf belong to different classes. Gini impurity of a split is the weighted sum of all new leaves' Gini impurities with weights relative to leaf size (samples in leaf divided by samples in parent).

Given C classes consider a leaf with n samples, n_1 samples from class 1, n_2 samples from class 2, and so on. Then Gini impurity of the leaf is

$$\sum_{i=1}^{C}\frac{n_i}{n}\left(1-\frac{n_i}{n}\right)=\sum_{i=1}^{C}\left(\frac{n_i}{n}-\left(\frac{n_i}{n}\right)^2\right)=\sum_{i=1}^{C}\frac{n_i}{n}-\sum_{i=1}^{C}\left(\frac{n_i}{n}\right)^2=1-\sum_{i=1}^{C}\left(\frac{n_i}{n}\right)^2.$$

Gini impurity of a pure leaf is 0. Gini impurity of a leaf with same number of samples from all classes is $1-\frac{1}{C}$. In particular, Gini impurity is always below one. The lower the Gini impurity the better the split.

Example

Consider the 200 samples from above again, 100 belonging to class A, 100 belonging to class B, and two splits

$$\text{25 A / 75 B} \quad \text{and} \quad \text{75 A / 25 B} \qquad\qquad \text{50 A / 100 B} \quad \text{and} \quad \text{50 A / 0 B}.$$

Gini impurity of first split is

$$\frac{100}{200}\left(1-\left(\frac{25}{100}\right)^2-\left(\frac{75}{100}\right)^2\right)+\frac{100}{200}\left(1-\left(\frac{75}{100}\right)^2-\left(\frac{25}{100}\right)^2\right)=0.3750.$$

For the second split we have

$$\frac{150}{200}\left(1-\left(\frac{50}{150}\right)^2-\left(\frac{100}{150}\right)^2\right)+\frac{50}{200}\left(1-\left(\frac{50}{50}\right)^2-\left(\frac{0}{50}\right)^2\right)=0.3333.$$

Looking at Gini impurity we would choose the second split.

Example

Consider the three samples from above again belonging to classes A, B, A. Then the original node has Gini impurity

$$1-\left(\frac{1}{3}\right)^2-\left(\frac{2}{3}\right)^2=\frac{4}{9}$$

and the two possible splits

$$\text{A}\quad\text{and}\quad\text{BA},\qquad\qquad \text{AB}\quad\text{and}\quad\text{A}$$

each have Gini impurity

$$\frac{1}{3}\cdot 0+\frac{2}{3}\left(1-\left(\frac{1}{2}\right)^2-\left(\frac{1}{2}\right)^2\right)=\frac{1}{3}.$$

Thus, both splits reduce Gini impurity whereas number of missclassifications remains the same.

Splitting by entropy

Entropy is an alternative to and very similar to Gini impurity. The formula for entropy of a leaf is

$$-\sum_{i=1}^{C}\frac{n_i}{n}\log\frac{n_i}{n}\qquad\text{with}\quad 0\cdot\log 0:=0.$$

Entropy is a concept from information theory motivated by entropy in physics. Entropy is a way to quantify information. Most data scientists use the term, but only few understand it. So we spend some time to explain the ideas behind.

Probabilities versus information: Consider a bag of colored balls. We do not know the exact number of balls of each color, but we know that on average (of many bags with colored balls) 50 per cent of the balls are red, 30 percent are green, 15 per cent are blue and 5 per cent are yellow. If we randomly take one ball out of the bag we may ask: What do we learn from this one ball about the contents of the bag? If the ball is red, then we know that there are red balls in the bag. That's not surprising since we knew that on average half the balls are red. We did not learn something really new about the contents of the bag. But if the ball is yellow, then we know, that there was at least one yellow ball in the bag. Probability for yellow was 5 per cent. Thus, it's not unlikely that there is no yellow ball in the bag. So from finding a yellow ball we obtain much more new information about the bag's contents than from finding a red ball. We may state our observation as follows: The less likely an event we observe is the more information it contains.

Measuring information: To express the information content of an event we may transform the events' probabilities to satisfy the following criteria.

- Information is nonnegative.
- Information is 0 if and only if probability is 1.
- The higher the probability, the lower the information obtained from observing the event (monotonicity).

- Information obtained from observing two independent events is the sum of information obtained from each of the two events.

The only function satisfying all four requirements is the negative logarithm (to an arbitrary base). So we define the amount of information obtained from observing an event as the logarithm of the event's probability. If p is a probability, then corresponding information is

$$-\log p.$$

Choosing a concrete base just scales the measure, because

$$\log_a p = (\log_a b)\,\log_b p.$$

For base 2 the unit for information is *bits*, for base e it's *nats*, for base 10 it's *dits* or *bans*.

In the above example finding a red ball has information 0.69 nats, finding a yellow ball has information 3.00 nats.

Entropy: Entropy is defined as mean information content. It's the weighted sum of information for all events with probabilities as weights. So more likely events have heigher weight. In the above example the bag's entropy is

$$-0.5\,\log 0.5 - 0.3\,\log 0.3 - 0.15\,\log 0.15 - 0.05\,\log 0.05 = 1.14.$$

Entropy measures disorder. The highest form of order is that only one event can occur (with probability 1). Then entropy is 0. The highest form of disorder is that all events are equally likely. With n events, entropy then is $\log n$. The more equally likely events, the higher the disorder.

Entropy for classification: In classification contexts we have one event per class. Analogously to the colored balls example above we randomly pick one sample out of a leaf and look at its label. Probabilities are the relative class counts. We choose the split with the lowest entropy. Or the other way round, we choose the split with the highest decrease in information obtainable from looking at concrete samples. Lowest information is reached if all samples in a leaf have the same label. So we cannot learn something from looking at a specific example.

27.3.2 Pruning

Fully grown trees should be pruned to avoid overfitting. *Reduced error pruning* is a simple and fast pruning technique for classification trees. One by one, beginning from the leaves, subtrees are replaced by leaves and resulting missclassification rate on validation data is calculated. If missclassification rate does not increase, the change is kept. This way we obtain a tree which cannot be improved by removing further subtrees.

An alternative is *cost-complexity pruning* based on some classification error measure (missclassification rate, Gini impurity, entropy,...). See *Regression Trees* (page 589) for details.

27.3.3 Classification Trees with Scikit-Learn

Scikit-Learn provides the `DecisionTreeClassifier`[521] class.

[521] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

CHAPTER

TWENTYEIGHT

ENSEMBLE METHODS

To increase overall prediction quality we could train several different models and somehow aggregate their prediction results. There are many different ways to realize this idea. Machine learning methods exploiting more than one trained model are called *ensemble methods*. Here we consider three classes of ensemble methods: *stacking*, *bootstrap aggregation (bagging)*, *boosting*.

Related projects:

28.1 The Idea

Imagine you have a difficult problem and ask an expert for advice. Why not ask several experts? One expert could tell you something wrong, but you do not realize that he or she is wrong, because you aren't an expert. If we have a list of experts at our disposal, what can we do with their advice?

- We could ask all experts and then find a 'meta-expert' who knows how to combine all the answers to a final answer. This is known as stacking.
- We could ask them independently and apply some simple aggregation function to their answers to obtain a final answer. If we ask for numerical values, we could take the mean. If we ask for categories, we could take the one appearing most often in the list of answers. That's the basic idea of bagging.
- We could ask one expert, think about his answer and identify possible weak points in her or his answer. Then we go to the second expert, ask the same question but tell her or him to look at certain aspects more closely. Then we go to the third expert and add information about weak points in the second expert's answer. We do this as long as we feel uncomfortable with the answers or as long as our list of experts isn't exhausted. This strategy is known as boosting. Each expert boosts the answer of the previous one.

All three strategies have in common that each expert could be rather weak (not really an expert), but the final answer will be quite accurate.

28.2 Stacking

Given a list of models trained on the same task we train a 'meta-model' to combine predictions of all models. The meta-model usually is an ANN.

Training data should be split:

- either one subset for training the weak models and one for training the meta-model
- or individual subsets for all models.

Stacking is typically used with heterogenous weak models. For instance we could combine results from an ANN, from a decision tree, and from k-NN.

Stacking is not widely used, but may become more important in future, because stacking allows to combine small specialized models trained an very different tasks (e.g., speech recognition and object detection in images) to one large model (e.g., detection of humans in combined video/audio signals). See, for instance, Google Pathways[522]

Scikit-Learn supports stacking with StackingRegressor[523] from the `ensemble` module.

28.3 Bagging

Bagging (short for bootstrap aggregation) averages predictions of many simple models to obtain a more accurate prediction than each single simple model can provide. The aim of bagging is to reduce variance (that is, prediction error due to overfitting) by averaging results from many high variance models.

Although bagging in principle can be applied to a set of very different machine learning models, usually it is used with a set of identical models.

28.3.1 Bootstrapping

If we train identical models on identical training data, models will yield more or less identical predictions. Thus, we have to train each model on a different data set. We could divide the data set into as many subsets as we have models, but then each subset would be rather small. Instead we use a method known as *bootstrapping* in statistics. We sample new data sets from the original data set with replacement. Thus, samples may occur several times in the new sets. The advantage of replacement is that distributions of samples in the new sets are independent from each other making the trained models independent from each other. Bootstrapping yields a list of data sets which on the one hand follow more or less the same distribution as the original data set and on the other hand can be (at least in principle) arbitrarily large.

28.3.2 Bagging with Scikit-Learn

Scikit-Learn supports bagging for regression tasks with BaggingRegressor[524] and BaggingClassifier[525] from Scikit-Learn's `ensemble` module. Corresponding estimator objects have the usual `fit` and `predict` interface. When creating the estimator we may pass the following arguments:

- `estimator`: a Scikit-Learn estimator object (linear regression, ANN, decision tree aso.) to be trained several times,
- `n_estimators`: how many models to train,
- `max_samples`: size of training subsets.

[522] https://blog.google/technology/ai/introducing-pathways-next-generation-ai-architecture/
[523] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingRegressor.html
[524] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html
[525] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html

There is also a `max_features` argument to restrict the number of features to consider in each model. Instead of training each model on a different data set we might train models on different sets of features (*random subspace method*).

Note that `BaggingRegressor` and `BaggingClassifier` also supports some bagging-like techniques we do not introduce here.

28.3.3 Random Forests

If bagging is used with decision trees as base model, then we have a *random forest* (a forest is a collection of trees). Scikit-Learn has some specialized routines for training random forests: RandomForestRegressor[526] and RandomForestClassifier[527].

Standard behavior is to grow trees to their maximum size. For complex data sets growing a forest of maximum size trees may result in memory exhaustion.

28.3.4 Random Forests for Feature Selection

Random forests can be exploited for feature selection. Having a trained random forest at hand, to calculate the importance of a feature do the following:

1. For all trees find all nodes splitting with respect to the feature.
2. For all nodes from 1. calculate the decrease in the impurity measure (variance, missclassification rate,…) caused by the split.
3. Calculate the weighted sum of all decreases. Weights are the number of samples in each node.

This procedure ensures that

- features decreasing impurity more than others have higher importance.
- features corresponding to nodes close to a root (more samples in node) have higher importance.

In Scikit-Learn we have access to random forest based feature importances via the `feature_importances_`[528] attribute of the `RandomForestRegressor` or `RandomForestClassifier` object after training the forest.

28.4 Boosting

The fundamental idea of boosting is to use information about prediction quality of a trained model to improve training of new model. Information about prediction quality of the second model then is used to train a third model, and so on. This process yields a sequence of models, each making up for weaknesses of the previous one. Finally, either predictions of the last model in the sequence are used or some averaging is done.

Although there is no restriction of the chosen models, one typically uses identical models for boosting. Boosting tends to reduce bias (prediction error due to lacking model complexity). Thus, we may use very weak models like decision stumps (trees with only two leaves). Note that this is in contrast to bagging, which tends to reduce variance.

There exist many different implementations of boosting. Here we consider three boosting algorithms in more detail: AdaBoost, gradient boosting, and XGBoost.

[526] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[527] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[528] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html?highlight=randomforest#sklearn.ensemble.RandomForestRegressor.feature_importances_

28.4.1 Adaptive Boosting for Regression (AdaBoost.R2)

AdaBoost originally has been developed for binary classification. Later an adaption to regression appeared (AdaBoost.R) and someday a modified regression version has been published (AdaBoost.R2). Scikit-Learn implements AdaBoost.R2. Thus, we restrict attention to that version. AdaBoost.R2 was introduced in Improving Regressors using Boosting Techniques[529] by Harris Drucker.

In adaptive boosting we associate a weight to each training sample. A small weight marks the sample as unimportant, high weight means 'very important'. We may use such weights in two ways:

- train a model only on samples with high weight or
- include weights into the training procedure (weighted mean squared error as loss function).

The first variant is always applicable. The second variant works for models trained by minimizing some loss function. AdaBoost.R2 only uses the first variant. But a weighted loss function is used for updating the weights.

Step 1 (initialization): Assign weights $w_1, \ldots, w_n$ to the training samples $(x_1, y_1,), \ldots, (x_n, y_n)$ and initialize all weights to $\frac{1}{n}$.

Step 2 (subset selection): Calculate probabilities

$$p_l := \frac{w_l}{\sum\limits_{\lambda=1}^{n} w_\lambda}, \qquad l = 1, \ldots, n,$$

and choose $N < n$ training samples according to the probabilities $p_1, \ldots, p_n$ with replacement.

Step 3 (training): Train a model on the N samples.

Step 4 (stopping criteria): Get predictions $y_{\text{pred},1}, \ldots, y_{\text{pred},n}$ for all (!) samples and calculate normalized squared errors

$$e_l := \frac{\left(y_{\text{pred},l} - y_l\right)^2}{\sum\limits_{\lambda=1}^{n} \left(y_{\text{pred},\lambda} - y_\lambda\right)^2} \in [0, 1].$$

Normalization simplyfies formulas below. If the denominator is zero we have perfect fitting (overfitting) and should stop the procedure, because there is no more need for improvement. But that's rarely seen in practice. Else calulate the weighted loss

$$L := \sum_{l=1}^{n} p_l\, e_l \in [0, 1].$$

Stop if $L \geq \frac{1}{2}$. Without weighting we would have $L = 1$. With weighting L expresses the average loss on the more important samples. Thus, L close to 1 (for instance $L \geq \frac{1}{2}$) indicates that the current model is not able to fit important samples much better than less important ones. So we stop the boosting procedure because no more improvement can be expected.

Step 5 (weight update): Multiply weight w_l by

$$\left(\frac{L}{1 - L}\right)^{1-e_l}$$

for $l = 1, \ldots, n$ and go to step 2. The function $L \mapsto \frac{L}{1-L}$ is monotonically increasing and maps $[0, \frac{1}{2})$ to $[0, 1)$. Small L decreases weights more than L close to $\frac{1}{2}$. With $L = \frac{1}{2}$ weights would remain unchanged. The closer an individual error e_l is to zero the closer the corresponding update factor is to $\frac{L}{1-L}$. The larger e_l the closer the update factor is to 1 (weight remains almost unchanged). Thus, weights of well fitted samples are decreased whereas weights of less well fitted ones remain almost unchanged.

After stopping (aggregation): After stopping the iteration we have a list of trained hypotheses $f_{\text{approx},1}, \ldots, f_{\text{approx},q}$, where we exclude the last one, which satisfied the stopping criterion. Based on this list we define the final hypothesis f_{approx} as follows:

[529] https://www.researchgate.net/publication/2424244_Improving_Regressors_Using_Boosting_Techniques/link/0deec51ae736538cec000000/download

- For a feature vector x get predictions $f_{\text{approx},1}(x),\ldots,f_{\text{approx},q}(x)$ from all models.
- For each model calculate
$$\ln\left(\frac{1}{L}-1\right)$$
with modeldependent L as introduced in step 4. Denote these numbers by $a_1,\ldots,a_q$. They will be used as weights for the models. The function $L\mapsto\frac{1}{L}-1$ is monotonically decreasing and maps 0 and $\frac{1}{2}$ to $+\infty$ and 1, respectively. The logarithm maps $[1,\infty]$ monotonically to $[0,\infty]$. Thus, L close to zero yields a high weight for the model, L close to $\frac{1}{2}$ yields a small weight.
- Calculate the weighted median of $f_{\text{approx},1}(x),\ldots,f_{\text{approx},q}(x)$ with weights $a_1,\ldots,a_q$:
$$f_{\text{approx}}(x):=\inf\left\{y\in\mathbb{R}:\sum_{\substack{\mu=1\\f_{\text{approx},\mu}(x)\leq y}}^{q}a_\mu\geq\frac{1}{2}\sum_{\mu=1}^{q}a_\mu\right\}.$$

Rules for weight update, stopping and aggregation are somewhat arbitrary, but yield good results in practice.

Scikit-Learn implements adaptive boosting in AdaBoostRegressor[530] in the `ensemble` module.

28.4.2 Adaptive Boosting for Classification (AdaBoost-SAMME)

Above we considered AdaBoost for regression tasks. For classification problems AdaBoost originated in 1995 focussing on binary classification. In 2009 the now standard AdaBoost algorithm for multiclass problems has been proposed (see Multi-class AdaBoost[531] by Zhu, Zou, Rosset, Hastie), known as AdaBoost-SAMME. An AdaBoost model outputs class labels (not probabilities) and base models are required to yield class labels, too.

Like for regression each training sample is assigned a weight. Depending on the prediction qualitiy of the model weights are modified to control fitting of the next model. The better the previous fit the smaller the weight. Thus, fitting concentrates on difficult samples. The detailed procedure for C classes is as follows:

Step 1 (initialization): Set all weights $w_1,\ldots,w_n$ to $\frac{1}{n}$.

Step 2 (training): Train a model on the weighted samples.

Step 3 (weighted classification error): Calculate
$$e:=\frac{\text{sum of weights of missclassified samples}}{\text{sum of all weights}}.$$

Step 4 (stopping criteria): Typically AdaBoost is stopped after a fixed number of iterations (100, for instance) or if $e=0$ (*early stopping*). Alternatively one may stop AdaBoost if $e\geq\frac{C-1}{C}$, that is, if the error is not better than the error of random guessing.

Step 5 (weight update): Multiply all weights of *missclassified* samples by
$$(C-1)\,\frac{1-e}{e}$$
and go to step 2. The update factor is greater than 1 if and only if the error e is smaller than the error for randomly assigned classes $\frac{C-1}{C}$.

After stoppping (aggregation): After stopping the iteration we have a list of trained hypotheses $f_{\text{approx},1},\ldots,f_{\text{approx},q}$ (step 2) and a list of weighted errors $e_1,\ldots,e_q$ (step 3). We define the final hypothesis f_{approx} as follows:

- Given a sample x for each model $f_{\text{approx},\mu}$ define a vector $a_\mu\in\{0,1\}^C$ with components
$$a_\mu^{(i)}:=\begin{cases}0, & \text{if } f_{\text{approx},\mu}(x)\neq i,\\ 1, & \text{if } f_{\text{approx},\mu}(x)=i,\end{cases}\qquad i=1,\ldots,C.$$

  Vector $a_1,\ldots,a_q$ can be regarded as one-hot encoded predictions of corresponding models.

[530] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html
[531] http://ww.web.stanford.edu/~hastie/Papers/SII-2-3-A8-Zhu.pdf

- Calculate the weighted sum

$$a := \sum_{\mu=1}^{q} \log\left((C-1)\,\frac{1-e_\mu}{e_\mu}\right)\,a_\mu.$$

  Components of the vector a can be regarded as scores for each class. Coefficients are 0 if the error e_μ equals the error from random guessing. The smaller the error the larger the coefficient.

- Predict class i for x, where i is the index of the largest component of a.

28.4.3 Gradient Boosting

Gradient boosting is an approximate gradient descent for a loss function. Approximations of the gradients are chosen to be predictions of (simple) models on the training set. Thus, we minimize a loss by improving an initial model. In each step a new model is added, yielding a weighted sum of models. Prediction performance of the overall model will be much better than for each single model. The procedure is repeated until some stopping criterion (performance on validation set, for instance) is satisfied. That's the basic idea. Now we have to fill the details.

Denote training samples by $(x_1, y_1), \ldots, (x_n, y_n)$ and let $L : \mathbb{R}^n \to \mathbb{R}$ be the loss with respect to the true labels $y_1, \ldots, y_n$. For mean squared error loss we have

$$L(z_1, \ldots, z_n) = \frac{1}{n}\sum_{l=1}^{n}(z_l - y_l)^2,$$

but we stick to the general case here.

Denote the initial hypothesis by $f_{\text{approx},1}$ (some trained model). Given a hypothesis $f_{\text{approx},j}$ we denote the vector of predictions by

$$y_{\text{pred},j} := (f_{\text{approx},j}(x_1), \ldots, f_{\text{approx},j}(x_n)).$$

We want to improve the training loss $L(y_{\text{pred},j})$ by doing a gradient step. In other words, how to modify predictions $y_{\text{pred},j}$ to make L smaller? Step direction is $-\nabla L(y_{\text{pred},j})$ and with step length s we obtain the updated predictions

$$y_{\text{pred},j} - s\,\nabla L(y_{\text{pred},j}).$$

The problem is that in general this is not a prediction of some model of interest. Thus, we fit a new model to samples

$$\left(x_1, \frac{\partial L}{\partial z_1}(y_{\text{pred},j})\right), \ldots, \left(x_n, \frac{\partial L}{\partial z_n}(y_{\text{pred},j})\right)$$

and set

$$f_{\text{approx},j+1} := f_{\text{approx},j} - s\,f_{\text{grad}},$$

where f_{grad} denotes the hypothesis fitted to the gradient.

The described iterative procedure finally yields a weighted sum of many models. Each model approximates a gradient. Samples very well fitted by a model will have small gradient. Thus, the next model will not modify corresponding prediction too much. The other way round, the next model will concentrate on samples with large prediction error with the previous model.

For mean squared error loss we have

$$\nabla L(y_{\text{pred},j}) = \frac{2}{n}(y_{\text{pred},j} - y),$$

the difference between predicted and true labels, also known as *residual vector* (neglecting the factor $\frac{2}{n}$). Consequently, the gradient approximation model is fit to the residual vector of the previous model. For a perfect fit model f_{grad} and step length $s = \frac{n}{2}$ we would have

$$f_{\text{approx},j+1}(x_l) = f_{\text{approx},j}(x_l) - s\,f_{\text{grad}}(x_l) = f_{\text{approx},j}(x_l) - (f_{\text{approx},j}(x_l) - y_l) = y_l$$

for $l = 1, \ldots, n$. In other words, the next model would perfectly fit the training data.

Scikit-Learn implements gradient boosting in GradientBoostingRegressor[532] and GradientBoostingClassifier[533] in the `ensemble` module.

Note that the the output of a gradient boosted model is some real number due to its additive nature. To solve binary classification tasks with gradient boosted models sigmoid or some similare function has to be applied to the outputs.

Multiclass classification task usually are solved in a one-versus-rest manner if boosting shall be applied. If we have C classes, then gradient boosting for binary classification is done C times in parallel. If the final model's output shall be probabilities softmax is applied to the predictions.

In principle, gradient boosting can be applied directly to a multiclass problem (no one-versus-all). But this is less efficient because a more complex model with C outputs has to be trained in each step. Moreover each gradient step yields smaller decrease of the loss function than with a one-versus-all approach. The reason for the latter observation is the structure of most loss functions. Loss is calculated for each class individually based on class probabilities and then summed up. Summands are mutually independent. Minimizing each summand individually typically yields faster descent than minimizing the whole sum as one function.

28.4.4 AdaBoost versus Gradient Boosting

AdaBoost-SAMME is a special case of gradient boosting. This relation has been discovered in 2000, five years after invention of the original AdaBoost algorithm. With C being the number of classes, class labels y have to be encoded in a one-hot-like manner

$$\tilde{y} := (\tilde{y}^{(1)}, \ldots, \tilde{y}^{(C)}), \qquad \tilde{y}^{(i)} := \begin{cases} 1, & \text{if } y = i, \\ -\frac{1}{C-1}, & \text{if } y \neq i, \end{cases}$$

and the loss function for gradient boosting is

$$L(z_1, \ldots, z_n) := \frac{1}{n} \sum_{l=1}^{n} \mathrm{e}^{-\frac{1}{C}\left(\tilde{y}_l^{(1)}\,\tilde{z}_l^{(1)} + \cdots + \tilde{y}_l^{(C)}\,\tilde{z}_l^{(C)}\right)}.$$

The full proof that AdaBoost-SAMME is equivalent to gradient boosting is provided by the authors of AdaBoost-SAMME in the above mentioned article.

28.4.5 XGBoost

XGBoost orignated in 2016 and became very popular due its success in several machine learning contests. XGBoost uses decision trees to boost an intial model and aims at large-scale high-performance computing.

The basic boosting idea is similar to gradient boosting, but instead of gradient descent for the loss function XGBoost uses second order minimization methods involving second partial derivatives (Newton method).

See XGBoost: A Scalable Tree Boosting System[534] for details on the algorithm and XGBoost Python Package[535] for a Python package.

[532] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
[533] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
[534] https://arxiv.org/abs/1603.02754
[535] https://xgboost.readthedocs.io/en/latest/python/index.html

CHAPTER

TWENTYNINE

SUPPORT-VECTOR MACHINES

The term support-vector machine (SVM) is used for a combination of certain machine learning models and corresponding training algorithms. SVMs are restricted to binary classification tasks, but can be applied to multiclass tasks via one-versus-rest or one-versus-one approach.

SVMs yield a separating hyperplane with maximum distance to both classes. Extensions to nonlinear separation will be discussed, too (kernel SVMs). Linear SVMs come in two flavors: hard margin SVMs and soft margin SVMs.

SVMs are state-of-art techniques for classification. Necessary computations can be implemented very efficiently and resulting models are very robust to changes in the training data. SVMs are well suited for very high dimensional data sets. A trained SVM model requires only very few memory (in constrast to ANNs) and predictions can be computed very efficiently.

Related projects:

29.1 Hard Margin SVMs

Given training samples $(x_1, y_1), \ldots, (x_n, y_n)$ with inputs in $\mathbb{R}^m$ and labels in $\{-1, 1\}$ (binary classification) we want to find a separating hyperplane with maximum distance to both classes. The distance betwenn both classes with respect to a hyperplane is called *margin*.

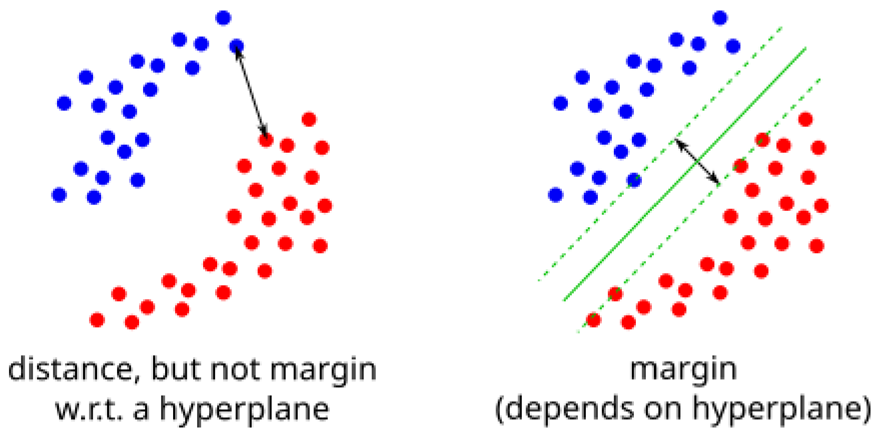


Fig. 29.1: Typically, the margin w.r.t. a hyperplane is smaller than the distance between two sets.

29.1.1 Hyperplanes

A hyperplane is the set of all x in $\mathbb{R}^m$ satisfying

$$a^{\mathrm{T}}\,x+b=0.$$

The vector $a\in\mathbb{R}^m\quad\{0\}$ controls the direction of the hyperplane (normal vector) and $b\in\mathbb{R}$ controls the distance to the origin (the distance is $\frac{|b|}{|a|}$). If two normal vectors only differ in length, not in direction, then corresponding hyperplanes are in parallel. The hyperplane equation can be multiplied by any nonzero real number without effecting the hyperplane. Thus, many different pairs (a,b) yield the same hyperplane.

A hyperplane is the level set for level 0 of a function

$$h_{a,b}:\mathbb{R}^m\rightarrow\mathbb{R},\quad h_{a,b}(x):=a^{\mathrm{T}}\,x+b.$$

On one side of the hyperplane we have $h_{a,b}(x)>0$. On the other side we have $h_{a,b}(x)<0$. The absolute value $|h_{a,b}(x)|$ grows linearly with the distance of x to the hyperplane. All level sets of $h_{a,b}$ are hyperplanes parallel to the hyperplane $h_{a,b}(x)=0$.

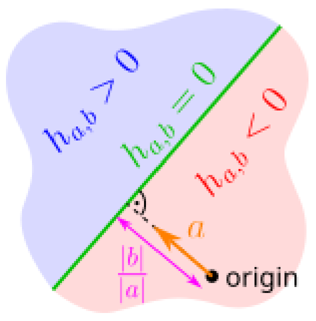


Fig. 29.2: A hyperplane can be regarded as the zero level set of a linear function.

Hint: For brevity we'll write 'the hyperplane $h_{a,b}(x)=0$' instead of the more correct 'the hyperplane $\{x\in\mathbb{R}^m:h_{a,b}(x)=0\}$'.

29.1.2 Separating Hyperplanes

Let L^+ and L^- be the index sets of positive and negative samples, respectively. That is,

$$L^+ := \{l \in \{1,\dots,n\} : y_l = 1\} \quad \text{and} \quad L^- := \{l \in \{1,\dots,n\} : y_l = -1\}.$$

A hyperplane $h_{a,b}(x) = 0$ is called *separating* (with respect to the given data set) if

$$h_{a,b}(x_l) > 0 \quad \text{for} \quad l \in L^+ \qquad \text{and} \qquad h_{a,b}(x_l) < 0 \quad \text{for} \quad l \in L^-.$$

We may rewrite this condition as

$$y_l\, h_{a,b}(x_l) > 0 \quad \text{for all } l.$$

Hint: In our definition of 'separating' we not only require separation of both classes but also that the negative class is on the negative side of the hyperplane and the positive class is on the positive side. If classes are on the wrong side, that is, if $y_l\, h_{a,b}(x_l) < 0$ for all l, then $h_{-a,-b}(x) = 0$ is a separating hyperplane in terms of our definition. But note that $h_{a,b}(x) = 0$ and $h_{-a,-b}(x) = 0$ in fact are two descriptions of one and the same hyperplane.

Given a separating hyperplane $h_{a,b}(x) = 0$ the margin is

$$\begin{aligned}
\text{margin} &= \min_{l\in L^+} \frac{|a^{\mathrm{T}} x_l + b|}{|a|} + \min_{l\in L^-} \frac{|a^{\mathrm{T}} x_l + b|}{|a|}\\
&= \min_{l\in L^+} \frac{a^{\mathrm{T}} x_l + b}{|a|} + \min_{l\in L^-} \frac{-(a^{\mathrm{T}} x_l + b)}{|a|}\\
&= \min_{l\in L^+} \frac{a^{\mathrm{T}} x_l + b}{|a|} - \max_{l\in L^-} \frac{a^{\mathrm{T}} x_l + b}{|a|}\\
&= \min_{l\in L^+} \frac{a^{\mathrm{T}} x_l}{|a|} - \max_{l\in L^-} \frac{a^{\mathrm{T}} x_l}{|a|}\\
&= \min_{l\in L^+} \left(\frac{a}{|a|}\right)^{\mathrm{T}} x_l - \max_{l\in L^-} \left(\frac{a}{|a|}\right)^{\mathrm{T}} x_l.
\end{aligned}$$

Obviously, the margin does not depend on b and it does not depend on the length of a (because a gets normalized in the formula above). Solely the direction of a matters. Note that $\left(\frac{a}{|a|}\right)^{\mathrm{T}} x_l$ is the (signed) distance between the origin and the projection of x_l onto the subspace spanned by a. Set

$$d_a^+ := \min_{l\in L^+} \left(\frac{a}{|a|}\right)^{\mathrm{T}} x_l \qquad \text{and} \qquad d_a^- := \max_{l\in L^-} \left(\frac{a}{|a|}\right)^{\mathrm{T}} x_l.$$

Then

$$\text{margin} = d_a^+ - d_a^-.$$

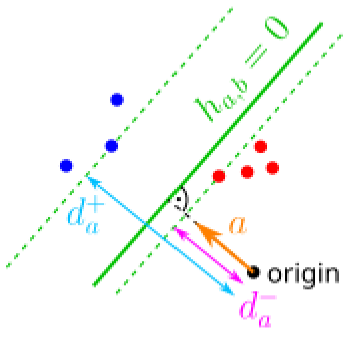


Fig. 29.3: Margin with respect to a separating hyperplane with corresponding notation.

29.1.3 A Centered Separating Hyperplane

Given a vector a with $d_a^+ > d_a^-$ each b with

$$-|a|\,d_a^+ < b < -|a|\,d_a^-$$

yields a separating hyperplane $h_{a,b}(x)=0$ (prove this!).

For fixed a there is only one separating hyperplane with equal distances to both classes. Corresponding b is

$$b = -|a|\,\frac{d_a^+ + d_a^-}{2}.$$

To see this simply calculate the distances:

$$\begin{aligned}\text{distance to positive class} &= \min_{l\in L^+}\frac{|a^\mathrm{T}\,x_l+b|}{|a|} = \min_{l\in L^+}\frac{a^\mathrm{T}\,x_l-|a|\,\frac{d_a^+ + d_a^-}{2}}{|a|} = d_a^+ - \frac{d_a^+ + d_a^-}{2} = \frac{d_a^+ - d_a^-}{2}\\ \text{distance to negative class} &= \min_{l\in L^-}\frac{|a^\mathrm{T}\,x_l+b|}{|a|} = \min_{l\in L^-}\frac{-\left(a^\mathrm{T}\,x_l-|a|\,\frac{d_a^+ + d_a^-}{2}\right)}{|a|} = -\max_{l\in L^-}\frac{a^\mathrm{T}\,x_l-|a|\,\frac{d_a^+ + d_a^-}{2}}{|a|}\\ &= -\left(d_a^- - \frac{d_a^+ + d_a^-}{2}\right) = -\frac{d_a^- - d_a^+}{2} = \frac{d_a^+ - d_a^-}{2}\end{aligned}$$

29.1.4 Maximum Margin

So far we know how to find a centered separating hyperplane given a fixed direction a and we also know how to calculate the margin. To find a (centered) separating hyperplane with maximum margin we have to solve

$$\min_{l\in L^+}\left(\frac{a}{|a|}\right)^\mathrm{T} x_l - \max_{l\in L^-}\left(\frac{a}{|a|}\right)^\mathrm{T} x_l \to \max_{a\in\mathbb{R}^m}$$

for a and then calculate b. This minimization problem is non-differentiable and lacks any other useful structure for analytical or numerical minimization.

Although the idea of a separating hyperplane with maximum margin is simple and straight forward, the major contribution of the inventors of SVMs (Vapnik[536] and Chervonenkis[537]) is a reformulation of the margin maximization problem as a quadratic minimization problem. A minimization problem is called quadratic if the objective function is quadratic and all contraints are linear (the set of feasible points is an intersection of half spaces). There exist several very efficient algorithms for solving quadratic minimization problems, making margin maximization a computationally tractable task.

Parameters a and b for the centered separating hyperplane with maximum margin are the solution to

$$\boxed{|a|^2 \to \min_{a\in\mathbb{R}^m}\qquad \text{with constraints}\qquad \begin{cases} a^\mathrm{T}\,x_l+b\geq 1, & \text{for } l\in L^+,\\ a^\mathrm{T}\,x_l+b\leq -1 & \text{for } l\in L^-.\end{cases}}$$

We now derive the quadratic minimization problem from our considerations above.

We start with fixed a and consider the corresponding centered separating hyperplane (assuming there is a separating hyperplane with normal vector a):

$$a^\mathrm{T}\,x - |a|\,\frac{d_a^+ + d_a^-}{2} = 0.$$

Dividing the equation by $|a|\,\frac{d_a^+ - d_a^-}{2}$ (second factor is half the margin) does not change the hyperplane. Resulting parameters

$$a^* := \frac{2}{d_a^+ - d_a^-}\,\frac{a}{|a|}\qquad\text{and}\qquad b^* := -\frac{d_a^+ + d_a^-}{d_a^+ - d_a^-}$$

536 https://en.wikipedia.org/wiki/Vladimir_Vapnik

537 https://en.wikipedia.org/wiki/Alexey_Chervonenkis

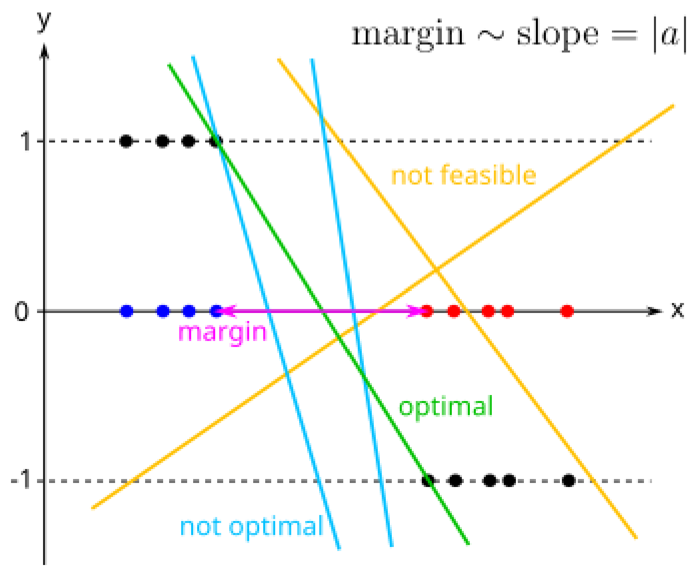


Fig. 29.4: A hyperplane has to be feasible and has to have maximum slope to solve the optimization problem.

do not depend on the length of a but solely on its direction. We now have

$$h_{a^*,b^*}(x_l) \geq \frac{2}{d_a^+ - d_a^-}\, d_a^+ - \frac{d_a^+ + d_a^-}{d_a^+ - d_a^-} = \frac{d_a^+ - d_a^-}{d_a^+ - d_a^-} = 1 \qquad \text{for} \quad l \in L^+$$

and

$$h_{a^*,b^*}(x_l) \leq \frac{2}{d_a^+ - d_a^-}\, d_a^- - \frac{d_a^+ + d_a^-}{d_a^+ - d_a^-} = \frac{d_a^- - d_a^+}{d_a^+ - d_a^-} = -1 \qquad \text{for} \quad l \in L^-,$$

that is a^* and b^* satisfy the constraints of the quadratic minimization problem.

Next we show that whenever we have a hyperplane $h_{a,b}(x) = 0$ satisfying the constraints and with a having the same fixed direction as a^*, then $|a| \geq |a^*|$. That is, a^* and b^* solve the quadratic minimization problem if we only consider one direction. Remember that as long as all considered normal vectors a have the same direction (but different length) all such a yield identical a^*. With

$$|a|\, d_a^+ = \min_{l \in L^+} a^{\mathrm{T}} x_l \geq 1 - b$$

and

$$|a|\, d_a^- = \max_{l \in L^-} a^{\mathrm{T}} x_l \leq -1 - b$$

we see

$$|a^*| = \frac{2}{d_a^+ - d_a^-} \leq \frac{2}{\frac{1-b}{|a|} - \frac{-1-b}{|a|}} = |a|.$$

The final step is to show that $|a^*|$ is the smaller the larger the margin is. But this follows immediately from

$$|a^*| = \frac{2}{d_a^+ - d_a^-}$$

because $d_a^+ - d_a^-$ is the margin.

29.1.5 Support Vectors

Given the centered separating hyperplane $h_{a,b}(x) = 0$ with maximum margin each sample x_l satisfying $h_{a,b}(x_l) = \pm 1$ is called *support vector*. If we remove all samples from the data set but the support vectors, the SVM solution does not change. The solution *is supported* by the support vectors.

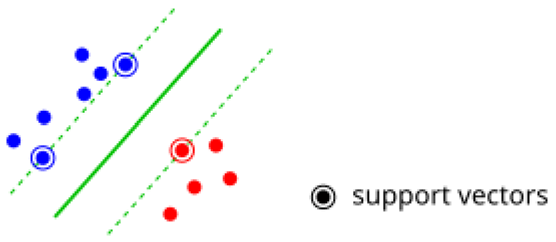


Fig. 29.5: Support vectors are data points on the margin's boundary.

Additional training samples alter the trained model only if they lie inside the margin! Thus, SVM classifiers are very robust to changes in the training set.

29.2 Soft Margin SVMs

Hard margin SVMs only work if the two classes are linearly separable. This is rarely seen in practise because some samples might be misslabeled or we do not have enough information to obtain clearly separated classes. In such cases the quadratic optimization problem has no feasible point and, thus, no solution.

29.2.1 From Constraints to Loss Functions

To overcome non-existence of solutions we may relax the constraints. Instead of requiring all constraints to be fully satisfied, we could measure the violation of constraints. Then we have two objectives: minimize constraint violation and maximize margin. Remember that maximizing the margin is equivalent to minimizing the length of the separating hyperplane's normal vector. So we have to solve two minimization problems at once. The standard approach is to minimize a weighted sum of both objective functions:

$$\text{measure for contraint violation} + \alpha\,|a|^2 \to \min_{a,b}.$$

The parameter α controls the trade-off between satisfaction of constraints and size of the margin. Small α yields well satisfied constraints (almost all samples on the correct side of the margin) but small margin (large $|a|$). Large α leads to a wide margin but violated constraints (incorrect predictions on training set).

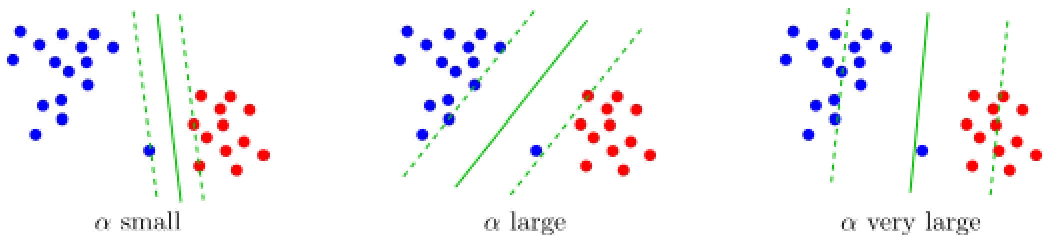


Fig. 29.6: Margin width and separation quality depend on the parameter α.

The measure for constraint violation is a typical loss functions, because for SVMs constraint violation is equivalent to missclassification. A loss function measures the distance between a model's predictions and true labels. An SVM model's prediction for input x is the sign of $a^{\mathrm{T}}\,x + b$, but the value $a^{\mathrm{T}}\,x + b$ carries more information than just the sign (that is, the predicted class): if $|a^{\mathrm{T}}\,x + b|$ is large the prediction is very reliable, if it is small the sample is very close to the decision boundary. We may interpret $a^{\mathrm{T}}\,x + b$ as a score and, thus, as the model's prediction.

For a given loss function $L:\mathbb{R}\times\{-1,1\}\to\mathbb{R}$ we want to solve

$$\frac{1}{n}\sum_{l=1}^{n} L(a^{\mathrm{T}}x_l+b,y_l)+\alpha\,|a|^2\to\min_{a,b}.$$

The loss function L should be zero if and only if x_l is on the correct side of the margin, that is, if and only if $y_l\,(a^{\mathrm{T}}x_l+b)\geq 1$. If x_l is not on the correct side (on the wrong side or inside margin), then L should be the larger the farther away x_l is from the correct side. In this case $1-y_l\,(a^{\mathrm{T}}x_l+b)$ is a reasonable choice. Both cases can be expressed in one formula:

$$L(z,y):=\max\{0,1-y\,z\}.$$

This loss function is known as *hinge loss*.

The minimization problem of soft margin SVMs now reads

$$\boxed{\frac{1}{n}\sum_{l=1}^{n}\max\{0,1-y_l\,(a^{\mathrm{T}}x_l+b)\}+\alpha\,|a|^2\to\min_{a,b}.}$$

Soft margin SVMs still try to maximize the margin between both classes, but some samples are allowed to lie inside the margin or even on the wrong side. So the margin is not a hard one, but in some sense soft.

29.2.2 Quadratic Optimization

The minimization problem above is not differentiable, but convex. There are several efficient algorithms for approximating the minimizer (subgradient descent). Alternatively we may rewrite it as a quadratic optimization problem. For this purpose we start with the quadratic hard margin problem

$$|a|^2\to\min_{a\in\mathbb{R}^m}\qquad\text{with constraints}\quad y_l\,(a^{\mathrm{T}}x_l+b)\geq 1$$

and introduce n additional variables $s_1,\ldots,s_n$ (sometimes called *slack variables*) expressing the violation of the hard margin constraints. Instead of the hard margin constraints we require

$$y_l\,(a^{\mathrm{T}}x_l+b)\geq 1-s_l\qquad\text{and}\qquad s_l\geq 0.$$

Additional nonnegativity constraints ensure that satisfied hard margin constraints always yield a violation of zero (instead of negative violation). Minimal constraint violation can be reached by minimizing the sum of all s_l. Because we want to minimize $|a|$, too, we minimize a weighted sum

$$\boxed{\frac{1}{n}\sum_{l=1}^{n}s_l+\alpha\,|a|^2\to\min_{s,a,b}\qquad\text{with constraints}\quad y_l\,(a^{\mathrm{T}}x_l+b)\geq 1-s_l,\quad s_l\geq 0.}$$

For each a and b constraints can be satisfied by choosing $s_1,\ldots,s_n$ large enough. The smallest feasible s_l is

$$\max\{0,1-y_l\,(a^{\mathrm{T}}x_l+b)\}.$$

Thus, solving the minimization problem with respect to $s_1,\ldots,s_n$ (with fixed a and b) yields the optimal value

$$\min_{a,b}\frac{1}{n}\sum_{l=1}^{n}\max\{0,1-y_l\,(a^{\mathrm{T}}x_l+b)\}+\alpha\,|a|^2.$$

This shows that the quadratic problem with slack variables is equivalent to the original non-differentiable problem.

29.2.3 Another Reformulation

Applying some mathematical standard techniques for transforming optimization problems (Lagrange duality[538]) we may derive another reformulation of the soft margin SVM minimization problem:

$$\sum_{l=1}^{n} c_l - \frac{1}{2}\sum_{l=1}^{n}\sum_{\lambda=1}^{n} y_l\,y_\lambda\,(x_l^{\mathrm{T}}\,x_\lambda)\,c_l\,c_\lambda \to \max_{c_1,\ldots,c_n}$$

$$\text{with constraints}\quad \sum_{l=1}^{n} c_l\,y_l = 0 \quad\text{and}\quad 0 \leq c_l \leq \frac{1}{2\,n\,\alpha},\ l = 1,\ldots,n.$$

This again is a quadratic optimization problem with linear constraints. From $c_1,\ldots,c_n$ we obtain the centered separating hyperplane with maximum margin by (without proof):

$$a = \sum_{l=1}^{n} c_l\,y_l\,x_l, \qquad b = y_\lambda - \sum_{l=1}^{n} c_l\,y_l\,x_l^{\mathrm{T}}\,x_\lambda \quad\text{for some } \lambda \text{ with } 0 < c_\lambda < \frac{1}{2\,n\,\alpha}.$$

From duality theory one obtains the following interpretation of the c_l:

- If $c_l = 0$, then x_l is on the correct side of the margin.
- If $c_l = \frac{1}{2\,n\,\alpha}$, then x_l is inside the margin or on the wrong side.
- If $0 < c_l < \frac{1}{2\,n\,\alpha}$, then x_l is on the boundary between margin and correct side.

29.2.4 Support Vectors

In the context of soft margin SVMs *support vectors* are samples x_l which are not classified correctly or lie on the margin's boundary. With the above reformulation of the soft margin minimization problem support vectors are characterized by $c_l > 0$.

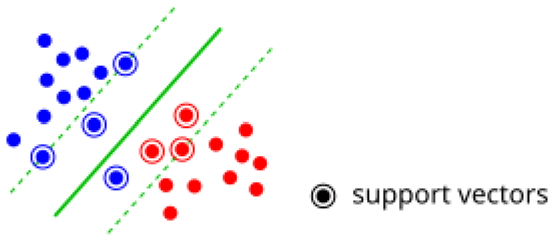


Fig. 29.7: In contrast to hard margin SVMs support vectors may lie inside the margin.

From the above reformulation we immediately see that the separating hyperplane (that is, a and b) can be calculated from the support vectors. Thus, all other training samples do not influence classification.

Given some input x prediction is the sign of

$$\sum_{l=1}^{n} c_l\,y_l\,x_l^{\mathrm{T}}\,x - \sum_{l=1}^{n} c_l\,y_l\,x_l^{\mathrm{T}}\,x_\lambda + y_\lambda \quad\text{for some } \lambda \text{ with } 0 < c_\lambda < \frac{1}{2\,n\,\alpha}.$$

Most of the c_l are zero. Only the (few) support vectors are required for calculating predictions. Thus, predictions from SVMs are very fast.

Another remarkable feature of the reformulated minimization problem is that the minimization problem as well as corresponding predictions only depend on inner products of (training) inputs, not on the x_l themselves.

[538] https://en.wikipedia.org/wiki/Duality_(optimization)

29.3 Kernel SVMs

SVMs as introduced above only yield linear classifiers. With some simple modification, known as the *kernel trick*, we may extend soft margin SVMs to nonlinear classification, where the decision boundary is defined by some nonlinear function instead of a hyperplane.

29.3.1 Feature Transforms

When considering linear regression we applied functions $\varphi_1, \ldots, \varphi_\mu : \mathbb{R}^m \rightarrow \mathbb{R}$ to the feature values and then applied linear regression with linear functions to the transformed features to obtain nonlinear models. Exactly the same idea applies to SVMs. Given a vector-valued function $\varphi : \mathbb{R}^m \rightarrow \mathbb{R}^\mu$ with $\mu > m$ we transform all inputs x to $\varphi(x)$ and train a linear SVM classifier in $\mathbb{R}^\mu$.

Example

For $m = 3$ polynomial features of degree 2 are given by

$$\varphi(x) = \left(\tfrac{1}{\sqrt{2}},\, x^{(1)},\, x^{(2)},\, x^{(3)},\, \tfrac{1}{\sqrt{2}}\left(x^{(1)}\right)^2,\, \tfrac{1}{\sqrt{2}}\left(x^{(2)}\right)^2,\, \tfrac{1}{\sqrt{2}}\left(x^{(3)}\right)^2,\, x^{(1)}\,x^{(2)},\, x^{(1)}\,x^{(3)},\, x^{(2)}\,x^{(3)}\right).$$

Why we use $\sqrt{2}$ here will become clear below.

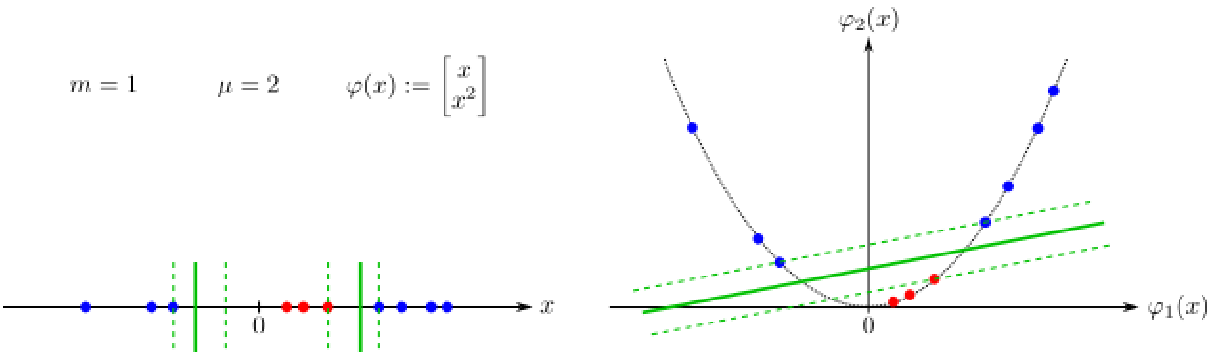


Fig. 29.8: Data set and margin in original and in transformed space.

29.3.2 Kernels

Training and prediction with soft margin SVMs do not use the feature values directly but only inner products of the inputs. Thus, the transform φ only appears in expressions

$$K(x, \tilde{x}) := \varphi(x)^{\mathrm{T}}\,\varphi(\tilde{x}).$$

Given some feature transform φ corresponding function $K : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$ is called a *kernel*.

Kernels can be interpreted as similarity measures because K attains its maximum for $x = \tilde{x}$ and K is zero if $\varphi(x)$ is orthogonal to $\varphi(\tilde{x})$.

Example

For $m = 3$ polynomial features of degree 2 as above yield the kernel

$$K(x, \tilde{x}) = \tfrac{1}{2}\left(x^{\mathrm{T}}\,\tilde{x} + 1\right)^2.$$

The $\sqrt{2}$ in the transform φ ensures that we get such a simple expression for the inner product of two transformed feature vectors.

Working with kernels instead of feature transforms is much more efficient. In the above example computing $\varphi(x)^{\mathrm{T}}\,\varphi(\tilde{x})$ requires two feature transforms and an inner product in $\mathbb{R}^{10}$. Computing $K(x,\tilde{x})$ only requires an inner product in $\mathbb{R}^3$ plus one addition and one multiplication in $\mathbb{R}$.

For general m and polynomial features of degree 2 we would have $\mu = \frac{m\,(m+1)}{2} + m + 1$. For $m = 1000$ this yields $\mu = 501501$. Thus, computing inner products in $\mathbb{R}^\mu$ is much more expensive than computing inner products in $\mathbb{R}^m$. The kernel trick allows for working with feature transforms without additional computational efforts.

29.3.3 More Kernels

Next to *inhomogeneous* polynomial kernels

$$(x^{\mathrm{T}}\,\tilde{x} + 1)^p \qquad \text{with some } p \in \mathbb{N}.$$

and *homogeneous* polynomial kernels

$$(x^{\mathrm{T}}\,\tilde{x})^p \qquad \text{with some } p \in \mathbb{N}.$$

there are several other kernels used in practise. The most important one is the *Gaussian kernel*

$$\mathrm{e}^{-\gamma\,|x-\tilde{x}|^2} \qquad \text{with some } \gamma > 0,$$

also known as *radial basis function (RBF) kernel*. Deriving corresponding feature transform requires some advanced math, because the feature transform maps inputs into an infinite dimensional space. Gaussian kernel can be interpreted as an inhomogeneous polynomial kernel of infinite degree.

For fixed $\tilde{x}$ and $m = 2$ the RBF kernel has a bell shaped graph with the bell centered at $\tilde{x}$. Predictions of an SVM for inputs x are the signs of

$$\sum_{l=1}^{n} c_l\, y_l\, K(x_l, x) + \text{constant}.$$

This function is a weighted sum of bells centered at $x_1, \ldots, x_n$ with weights zero for non-support vectors x_l.

29.4 SVMs with Scikit-Learn

With `LinearSVC`[539] Scikit-Learn offers a fast and well scaling implementation of linear SVMs for classification. For kernel SVMs there is `SVC`[540].

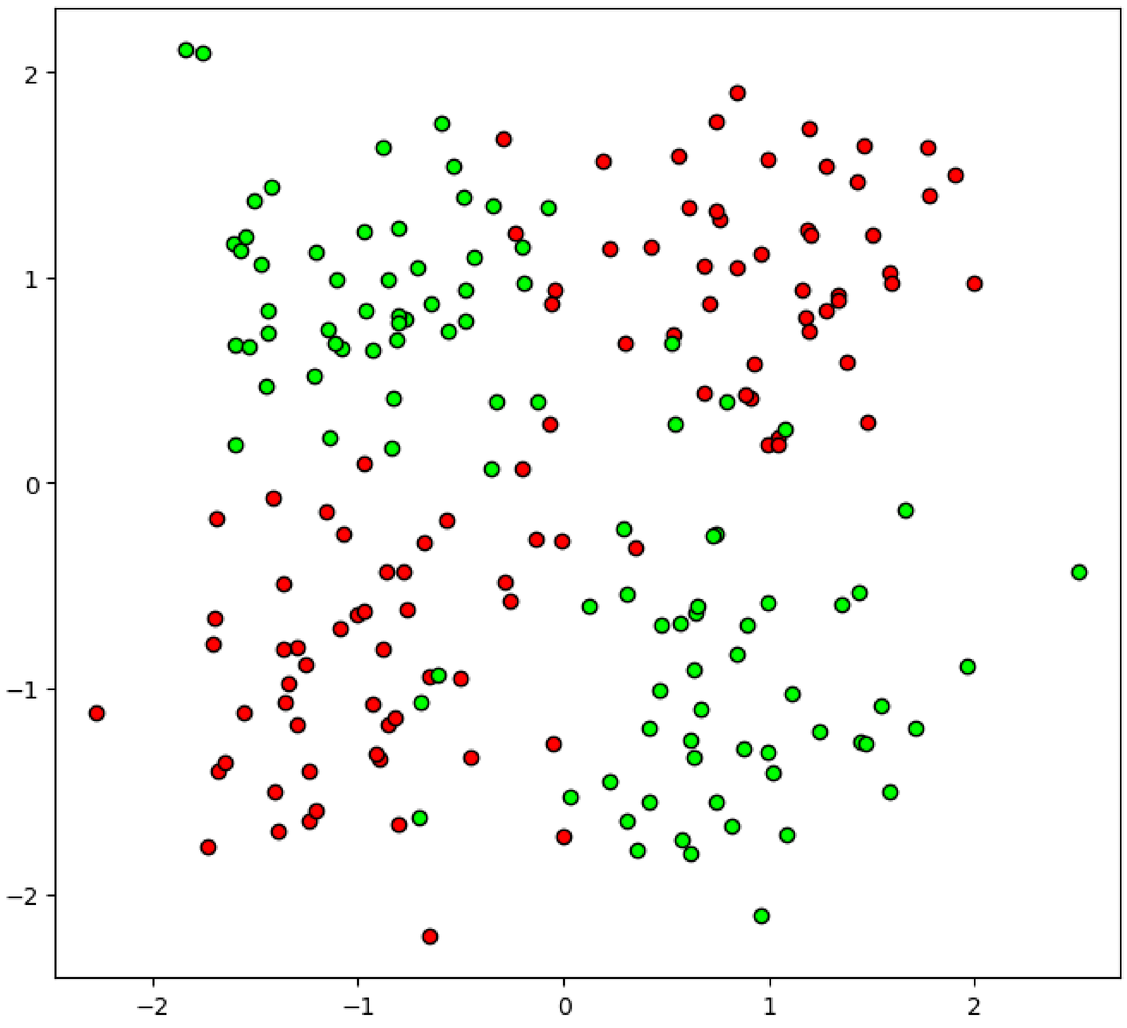
`SVC` by default uses the RBF kernel with γ adapted to the variance of the training feature values. Weighting between margin width and correct classification is controlled by the parameter `C` which is $\frac{1}{\alpha}$. After fitting, the classifier object provides access to the support vectors and to the decision function.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors
import sklearn.svm as svm

rng = np.random.default_rng(0)
```

We generate synthetic data with two classes, which are not linearly separable.

539 https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
540 https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

```
n0 = 100    # samples in class 0
n1 = 100    # samples in class 1

# generate two point clouds
X0a = rng.multivariate_normal([-1, -1], [[0.3, 0], [0, 0.3]], size=n0 // 2)
X0b = rng.multivariate_normal([1, 1], [[0.3, 0], [0, 0.3]], size=n0 // 2)
X1a = rng.multivariate_normal([1, -1], [[0.3, 0], [0, 0.3]], size=n1 // 2)
X1b = rng.multivariate_normal([-1, 1], [[0.3, 0], [0, 0.3]], size=n1 // 2)
X = np.concatenate((X0a, X0b, X1a, X1b))

# set labels
y0 = -np.ones(n0)
y1 = np.ones(n1)
y = np.concatenate((y0, y1))

# set plotting region
x0_min = X[:, 0].min() - 0.2
x0_max = X[:, 0].max() + 0.2
x1_min = X[:, 1].min() - 0.2
x1_max = X[:, 1].max() + 0.2

# plot data set
fig, ax = plt.subplots(figsize=(8,8))
ax.scatter(X[y == -1, 0], X[y == -1, 1], c='#ff0000', edgecolor='black')
ax.scatter(X[y == 1, 0], X[y == 1, 1], c='#00ff00', edgecolor='black')
ax.set_xlim(x0_min, x0_max)
ax.set_ylim(x1_min, x1_max)
ax.set_aspect('equal')
plt.show()
```



Classification accuracy can be controlled via α and γ. The higher α the wider the margin. The lower α the more accurate the classifications. Small γ yields smooth but possibly imprecise decision boundaries. For large γ decision boundaries are fit more tightly to the training data, resulting in more accurate predictions on training data.

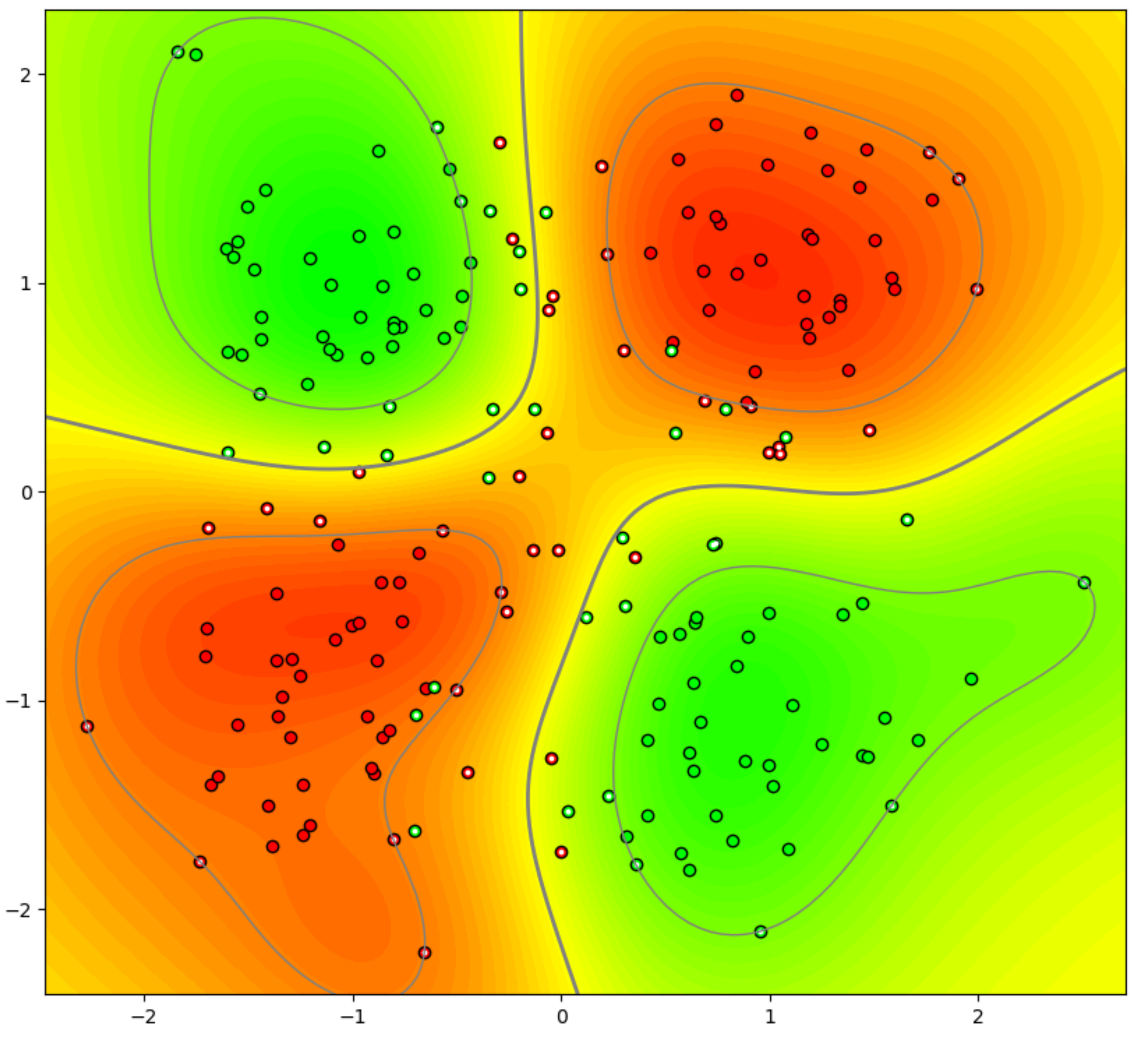
Small α and/or large γ may result in overfitting.

```
alpha = 1
gamma = 1

svc = svm.SVC(C=1/alpha, gamma=gamma)
svc.fit(X, y)

fig, ax = plt.subplots(figsize=(10, 10))

# plot model (function values color-coded)
x0, x1 = np.meshgrid(np.linspace(x0_min, x0_max, 100), np.linspace(x1_min, x1_max,
 ↪ 100))
y_grid = svc.decision_function(np.stack((x0.reshape(-1), x1.reshape(-1)),␣
 ↪axis=1)).reshape(100, 100)
max_y = np.max(np.abs(y_grid))
cm = matplotlib.colors.LinearSegmentedColormap.from_list('ryg', ['#ff0000', '
 ↪#ffff00', '#00ff00'])
ax.contourf(x0, x1, y_grid, cmap=cm, levels=np.linspace(-max_y, max_y, 50))

# plot decision boundary and margin
ax.contour(x0, x1, y_grid, levels=[-1, 0, 1], linewidths=[1, 2, 1], colors=3*['
 ↪#808080'])
```

(continues on next page)

(continued from previous page)

```
# plot data set
ax.scatter(X[y == -1, 0], X[y == -1, 1], c='#ff0000', edgecolor='black')
ax.scatter(X[y == 1, 0], X[y == 1, 1], c='#00ff00', edgecolor='black')

# plot support vectors
X_supp = X[svc.support_]
ax.scatter(X_supp[:, 0], X_supp[:, 1], c='#ffffff', marker='o', s=3)

ax.set_xlim(x0_min, x0_max)
ax.set_ylim(x1_min, x1_max)
ax.set_aspect('equal')

plt.show()
```



29.5 Support-Vector Regression (SVR)

Support vector regression is the little brother of support vector machines used for classification.

Self-study task

Get an overview of SVR from A tutorial on support vector regression[541]. Read section 1 (skip formulas in 1.3 and 1.4), subsections 2.1 and 2.2 (skipping formulas again), subsections 3.1 and 3.2, section 4. Then try to answer the following questions:

- Which loss function is used for SVR formulated as a minimization problem?
- What type of minimization problem has to be solved (linear, quadratic, cubic, general nonlinear)?
- Is the term *feature space* used to denote the same thing in the paper and in our book?
- Can SVR cope with high-dimensional feature spaces (in our terminology) similar to SVM?

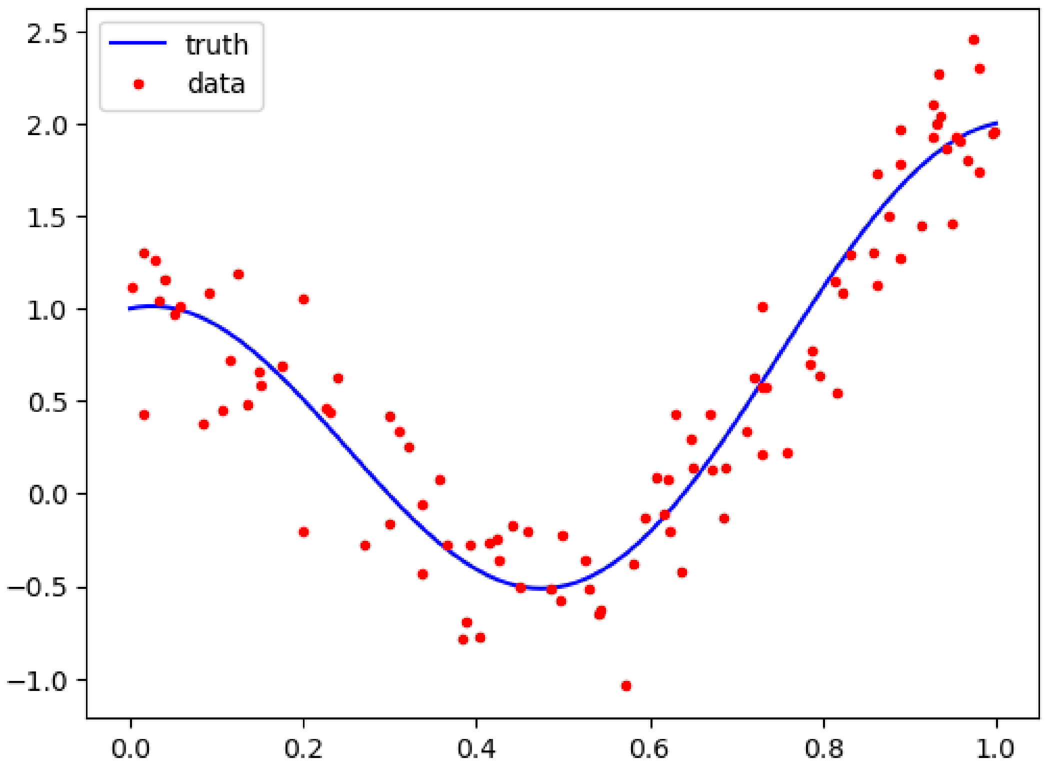
29.5.1 SVR with Scikit-Learn

Scikit-Learn implements SVR in `SVR`[542].

```
import numpy as np
import matplotlib.pyplot as plt

import sklearn.svm as svm

rng = np.random.default_rng(0)
```

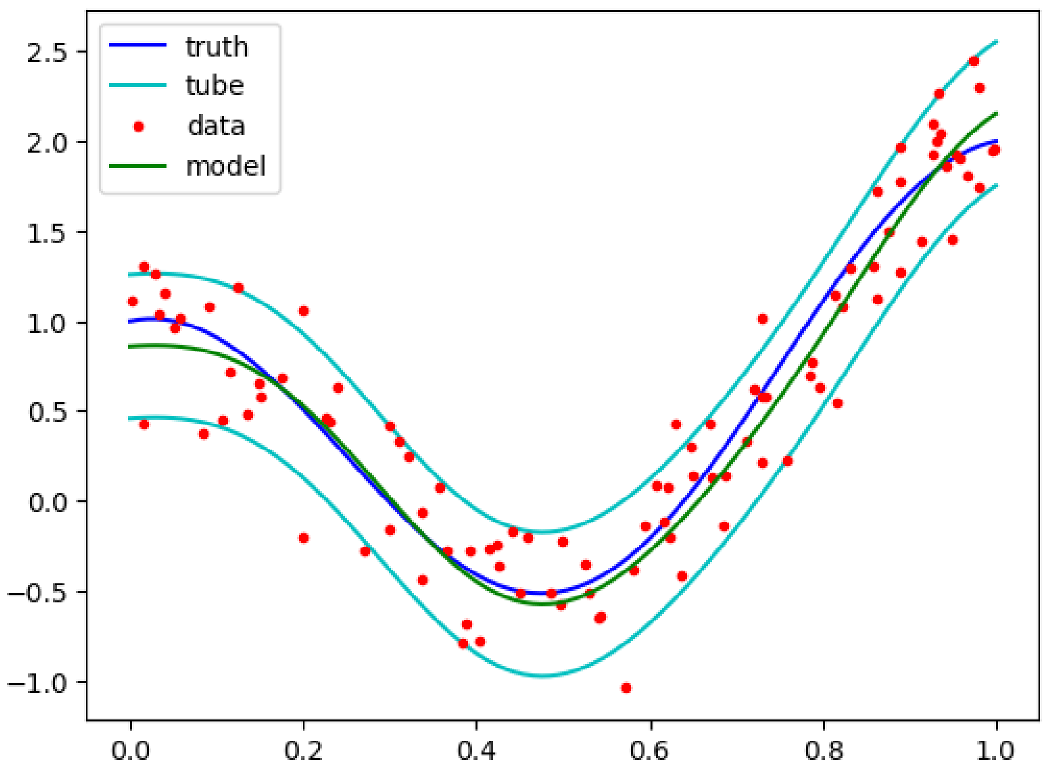
```
def truth(x):
    return x + np.cos(2 * np.pi * x)

xmin = 0
xmax = 1
x = np.linspace(xmin, xmax, 100)

n = 100     # number of data points to generate
noise_level = 0.3     # standard deviation of artificial noise

# simulate data
X = (xmax - xmin) * rng.random((n, 1)) + xmin
y = truth(X).reshape(-1) + noise_level * rng.standard_normal(n)

# plot truth and data
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='truth')
ax.plot(X.reshape(-1), y, 'or', markersize=3, label='data')
ax.legend()
plt.show()
```

[541] https://alex.smola.org/papers/2004/SmoSch04.pdf
[542] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html



When creating the SVR object we may provide the width of the ε-tube containing data with no influence on the loss function and we may select one of several predefined kernels. The regularization parameter can be specified, too. Here we have to take care, because the higher the parameter the less regularization is applied.

```
epsilon = 0.4
alpha = 1e-1

# regression
svr = svm.SVR(epsilon=epsilon, kernel='rbf', C=1/alpha)
svr.fit(X, y)

# get hypothesis for plotting
y_svr = svr.predict(x.reshape(-1, 1))

# plot truth, data, hypothesis
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='truth')
ax.plot(x, y_svr+epsilon, '-c', label='tube')
ax.plot(x, y_svr-epsilon, '-c')
ax.plot(X.reshape(-1), y, 'or', markersize=3, label='data')
ax.plot(x, y_svr, '-g', label='model')
ax.legend()
plt.show()
```



If regularization is very weak, then the tube contains almost all data points. For higher regularization the fitted hypothesis is smoother, but the tube does not contain all data points.

CHAPTER

THIRTY

NAIVE BAYES CLASSIFICATION

Naive Bayes classifiers are a class of relatively simple and computationally efficient classifiers for multiclass classification tasks. They are based on Bayes' theorem[543] for conditional probabilities and on the (naive) assumption that features are mutually independent[544] if interpreted as random variables.

Given training samples $(x_1, y_1), \ldots, (x_n, y_n)$ with m-dimensional feature vectors x_l and labels $y_l \in \{1, \ldots, C\}$ we want to train a model which assigns a label y_0 to non-training input x_0.

30.1 Principal Approach

The training samples can be regarded as n realizations of a pair (X, Y) of random variables. Then a natural way to select a label for the unlabeled feature vector x_0 is to postulate

$$P(Y = i | X = x_0) \to \max_{i \in \{1, \ldots, C\}}.$$

That is, we choose y_0 to be the maximizer of $P(Y = i | X = x_0)$ with respect to i.

The probabilities $P(Y = i | X = x_0)$ are not accessible. But Bayes' theorem states

$$P(Y = i | X = x_0) = \frac{p_{X|Y=i}(x_0)\, P(Y = i)}{p_X(x_0)}$$

for all $i \in \{1, \ldots, C\}$. Here, $p_{X|Y=i}$ is a density of the conditional probability measure $P(\,\cdot\,| Y = i)$ and p_X is the marginal density with respect to X of a density for P. If all components of X are discrete random variables, then $p_{X|Y=i}(x_0)$ and $p_X(x_0)$ can be replaced by the probabilities $P(X = x_0 | Y = i)$ and $P(X = x_0)$, respectively. If X contains continuous components, then we have to work with densities.

The denominator $p_X(x_0)$ in Bayes' theorem does not depend on i and, thus, does not influence maximization. Knowing $p_{X|Y=i}(x_0)$ and $P(Y = i)$ for all i we may compute

$$p_X(x_0) = \sum_{i=1}^{C} p_{X|Y=i}(x_0)\, P(Y = i),$$

because the sum of all $P(Y = i | X = x_0)$ has to be 1. The conditional density $p_{X|Y=i}(x_0)$ and the probably $P(Y = i)$ can be estimated from the training samples.

[543] https://en.wikipedia.org/wiki/Bayes%27_theorem

[544] https://en.wikipedia.org/wiki/Independence_(probability_theory)

30.2 Estimating Label Probabilities

For estimating the probabilities $P(Y=i)$ with $i\in\{1,\ldots,C\}$ there exist two standard approaches:

- If it is known from the context of the learning task, that all labels are equally likely, then
$$P(Y=i)=\frac{1}{C}$$
for all $i\in\{1,\ldots,C\}$.
- If there is no additional information about label distribution and if the training samples reflect the unknown underlying distribution sufficiently accurate, then
$$P(Y=i)=\frac{\bigl|\{l\in\{1,\ldots,n\}\,:\,y_l=i\}\bigr|}{n}$$
for all $i\in\{1,\ldots,C\}$ is a reasonable joice.

For some learning tasks there might be additional information about label distribution available. Then a tailored estimate for $P(Y=i)$ should be used.

30.3 Estimating Classes' Feature Probabilities

To estimate the densities $p_{X|Y=i}(x_0)$ in Bayes' theorem above we assume that all components $X^{(1)},\ldots,X^{(m)}$ of the random variable X are mutually independent. Usually this assumption is not satisfied, but it simplifies formulas and makes computations more efficient. Thus, the classification approach discussed here is called naive.

Mutual independency yields
$$p_{X|Y=i}(x_0)=p_{X^{(1)}|Y=i}\left(x_0^{(1)}\right)\cdots p_{X^{(m)}|Y=i}\left(x_0^{(m)}\right).$$

So we only have to estimate one-dimensional densities. To simplify notation we write $p_{U|Y=i}\,(u_0)$ as placeholder for one of the m densities.

30.3.1 Gaussian Probabilities

If U is a continuous random variable we may assume that $p_{U|Y=i}$ is a Gaussian density with mean μ_i and standard deviation σ_i:
$$p_{U|Y=i}(u_0)=\frac{1}{\sigma_i\,\sqrt{2\,\pi}}\,\exp\left(-\frac{1}{2}\left(\frac{u_0-\mu_i}{\sigma_i}\right)^2\right)$$
Parameters μ_i and σ_i can be estimated from the training samples in class i with standard techniques (see statistics lecture).

30.3.2 Bernoulli Probabilities

If U takes values in $\{0,1\}$ then $p_{U|Y=i}(u_0)$ in Bayes' theorem has to be replaced by $P(U=u_0|Y=i)$ with
$$P(U=0|Y=i)=1-p_i\qquad\text{and}\qquad P(U=1|Y=i)=p_i.$$

The parameter p_i can be estimated from the training samples in class i by counting the samples with value 1 for the feature under consideration.

30.3.3 Multinomial Probabilities

If $X^{(1)}, \dots, X^{(m)}$ count the occurrences of m possible events for several independent trials of an experiment, then the corresponding random vector follows a multinomial distribution[545] with parameters $p_1, \dots, p_m$ (probabilities of events) and ν (number of trials):

$$P(X = x_0) = \frac{\nu!}{x_0^{(1)}! \cdots x_0^{(m)}!}\, p_1^{x_0^{(1)}} \cdots p_m^{x_0^{(m)}},$$

where $p_1 + \cdots + p_m = 1$.

A typical machine learning application with multinomially distributed inputs is language processing. Each feature counts the number of occurrences of some word in a text. Based on such word counts models then can be trained to solve classification tasks.

Note that components of a multinomially distributed random vector are not independent. Thus, multinomial probabilities do not fit into the naive Bayes framework. Nonetheless *multinomial naive Bayes* is a fixed term, because non-independence does not prevent us from writing probabilities as products with each factor depending only on one component of the random vector. To see this we introduce probabilities $p_{i,1}, \dots, p_{i,m}$ for each class i. Then

$$P(X = x_0 | Y = i) = \frac{\nu!}{x_0^{(1)}! \cdots x_0^{(m)}!}\, p_{i,1}^{x_0^{(1)}} \cdots p_{i,m}^{x_0^{(m)}}$$

with $\nu = x_0^{(1)} + \cdots + x_0^{(m)}$. The factor

$$c(x_0) := \frac{\left(x_0^{(1)} + \cdots + x_0^{(m)}\right)!}{x_0^{(1)}! \cdots x_0^{(m)}!}$$

only depends on the sample x_0, but neither on the probabilities $p_{i,k}$ nor on the class i. Thus, we have the typical product structure

$$P(X = x_0 | Y = i) = \prod_{k=1}^{m} c(x_0)^{\frac{1}{m}}\, p_{i,k}^{x_0^{(k)}}$$

of the naive Bayes approach, although components of X are not independent. Moreover, we do not have to compute $c(x_0)$ explicitely. It's a scaling factor which can be computed subsequently from the fact that all probabilities have to sum to 1 (cf. $p_X(x_0)$ above).

The $p_{i,k}$ can be estimated from training data by counting the occurences of event k in class i. Denoting the number of occurrences by $\nu_{i,k}$ and the number of samples in class i by n_i the estimate reads

$$p_{i,k} = \frac{\nu_{i,k}}{n_i}.$$

If $\nu_{i,k}$ is zero for some event, which might by the case for small training set size, then $p_{i,k}$ is estimated to be zero. But if $p_{i,k}$ is zero for some event k, then due to the product structure the whole probability $P(X = x_0 | Y = i)$ will become zero. Thus, classes for which some event never occurres in the training data are never used as labels by the model.

Note: The problem with zero probabilities stems from estimation. The exact probabilities always are strictly positive, else we would count occurrences of events having zero probability. Due to estimating probabilites from incomplete data we may estimate probabilities to be zero although they aren't.

To avoid such failure either sufficiently large training sets have to be used or event counts have to be set to at least one for each event and each class. An typical estimate for $p_{i,k}$ which prevents problems with missing events is

$$p_{i,k} = \frac{\nu_{i,k} + 1}{n_i + m},$$

known as *Laplace smoothing*. The total number of samples per class is increased by m. The idea here is that per feature we add one training sample showing count 1 for this feature. To prevent zero counts for all features we have to add m samples.

[545] https://en.wikipedia.org/wiki/Multinomial_distribution

30.3.4 Kernel Density Estimates

If the underlying probability model is not know in advance, the densities $p_{X^{(k)}|Y=i}$ can be estimated from training data using kernel density estimation.

Self-study task

Read about kernel density estimation at Wikipedia[546] (introduction, definition, example).

30.4 Naive Bayes Classification with Scikit-Learn

Different variants of naive Bayes classification are implemented in Scikit-Learns's `naive_bayes` module[547].

30.5 Related Projects

- *Forged Banknotes* (page 963)
  - *Naive Bayes Classification* (page 969) (project)

[546] https://en.wikipedia.org/wiki/Kernel_density_estimation
[547] https://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive_bayes

CHAPTER

THIRTYONE

TEXT CLASSIFICATION

Up to now we only considered classification tasks with numerical features. But the very important field of text analysis and classification lacks such numerical features. Here we discuss relevant points to consider when using texts as model inputs. To avoid too much theory we immediately apply everything to a real-world example.

Related projects:

31.1 Preprocessing Text Data

We want to classify blog posts by age and gender of the post's author. Training data is available from The Blog Authorship Corpus[548], containing 650000 posts from 19000 blogs. Data may be freely used for non-commercial research purposes. Data was collected for research published in Effects of Age and Gender on Blogging[549] (J. Schler, M. Koppel, S. Argamon, J. Pennebaker, Proceedings of 2006 AAAI Spring Symposium on Computational Approaches for Analyzing Weblogs).

In addition to usual preprocessing and model training we will discuss how to convert text data to numerical features and how to cope with very high dimensional feature spaces. Models will be based on word counts. This first chapter contains everything we need to do before counting words. The second chapter discusses how to count words and how to use word counts for training machine learning models.

```
data_path = '/data/datasets/blogs/'
```

31.1.1 Getting and Restructuring the Data Set

Data comes as ZIP file from the above mentioned website (313 MB). The ZIP file contains one XML[550] file per blog (uncompressed size is 808 MB). An XML file is a text file containing some markup code (similar to HTML). Information about a blog's author is provided in the file name.

[548] https://u.cs.biu.ac.il/~koppel/BlogCorpus.htm
[549] https://u.cs.biu.ac.il/~schlerj/schler_springsymp06.pdf
[550] https://en.wikipedia.org/wiki/XML

Extracting Blog Information

File names have the format `blog_id.gender.age.industry.astronomical_sign.xml`. We create a data frame containing all the information but astronomical signs and save it to a CSV file.

```
import pandas as pd
import numpy as np
import zipfile
import re
import langdetect
```

```
with zipfile.ZipFile(data_path + 'blogs.zip') as zf:
    file_names = zf.namelist()

print(file_names[:5])
```

```
['blogs/', 'blogs/1000331.female.37.indUnk.Leo.xml', 'blogs/1000866.female.17.
↪Student.Libra.xml', 'blogs/1004904.male.23.Arts.Capricorn.xml', 'blogs/
↪1005076.female.25.Arts.Cancer.xml']
```

XML files are in a subdirectory and the subdirectory is listed by `zf.namelist()`, too.

```
blog_ids = []
genders = []    # 'm' if male, 'f' if female
ages = []
industries = []

for file_name in file_names:

    if file_name.split('.')[-1] != 'xml':
        print('skipping', file_name)
        continue

    blog_id, gender, age, industry, astro = file_name.split('/')[-1].split('.
↪')[0:-1]

    blog_ids.append(int(blog_id))

    if gender == 'male':
        genders.append('m')
    elif gender == 'female':
        genders.append('f')
    else:
        print('unknown gender:', gender)

    ages.append(int(age))

    industries.append(industry)

blogs = pd.DataFrame({'gender': genders, 'age': ages, 'industry': industries},␣
↪index=blog_ids)
```

```
skipping blogs/
```

```
blogs
```

```
        gender  age            industry
1000331      f   37              indUnk
```

(continues on next page)

(continued from previous page)

```
1000866      f   17               Student
1004904      m   23                  Arts
1005076      f   25                  Arts
1005545      m   25           Engineering
...        ...  ...                   ...
996147       f   36    Telecommunications
997488       m   25                indUnk
998237       f   16                indUnk
998966       m   27                indUnk
999503       m   25              Internet

[19320 rows x 3 columns]
```

```
blogs['industry'] = blogs['industry'].str.replace('indUnk', 'unknown')
```

```
blogs
```

```
        gender  age              industry
1000331      f   37               unknown
1000866      f   17               Student
1004904      m   23                  Arts
1005076      f   25                  Arts
1005545      m   25           Engineering
...        ...  ...                   ...
996147       f   36    Telecommunications
997488       m   25               unknown
998237       f   16               unknown
998966       m   27               unknown
999503       m   25              Internet

[19320 rows x 3 columns]
```

```
blogs.to_csv(data_path + 'blogs.csv')
```

Converting XML Files to one CSV File

We would like to have a data frame containing all blog posts. This data frame can easily be modified (data preprocessing!) and saved to a CSV file for future use.

Reading XML files can be done with the module `xml.etree.ElementTree`[551] from the standard python library. Usage is relatively simple but the parser gets stuck at almost all files. Although file extension is XML the files do not contain valid XML. Most files contain characters not allowed in XML files and some files even contain HTML fragments, which make the parser fail. Thus, we have to parse files manually.

The structure of the files is as follows:

```
<Blog>
<date>DAY,MONTH_NAME,YEAR</date>
<post>TEXT_OF_POST</post>
...
<date>DAY,MONTH_NAME,YEAR</date>
<post>TEXT_OF_POST</post>
</Blog>
```

When parsing the files we have to take into account the following observations:

[551] https://docs.python.org/3/library/xml.etree.elementtree.html

- Files are littered with random white space characters.
- Enconding is unknown and may vary from file to file.
- Files contain non-printable characters (00h-1Fh) other than line breaks (could be an encoding issue).
- Links in original posts are marked by `urlLink` (not `urllink` as indicated in the data set description).

White space can be removed with `str.strip()`. At least some files are not Unicode encoded. Interpreting non-Unicode files as Unicode may lead to errors. Using a 1-byte-encoding like ASCII or ISO 8859-1 also works for UTF-8 files, because each byte is interpreted as some character. Using a 1-byte-encoding for UTF-8 files may result in non-printable characters, which have to be removed before further processing of the text data. Removing all non-printable characters also removes line breaks. But line breaks do not matter for our classification task. So that's not a problem. The link marker `urlLink` can be savely removed. Possible URLs following the marker are hard to remove. For the moment we keep them.

Months in the date field are given by name, mostly in English but also in some other languages. To translate month names to numbers we use a dictionary. To get the dictionary we may start with English month names and then add more names one by one, if program stops with key error. Here is a list of languages and one XML file per language:

language	file
French	1022086.female.17.Student.Cancer.xml
Spanish	1162265.male.17.Student.Aries.xml
Portuguese	1253219.female.27.indUnk.Sagittarius.xml
German	1366984.female.25.Technology.Aries.xml
Estonian	1405766.male.24.HumanResources.Scorpio.xml
Italian	1847277.female.24.Student.Gemini.xml
Finnish	2042296.female.25.Student.Sagittarius.xml
Dutch	3032299.female.33.Non-Profit.Scorpio.xml
Polish	3340219.male.45.Technology.Virgo.xml
Romanian	3559973.female.36.Manufacturing.Aries.xml
Swedish	4145017.male.23.BusinessServices.Libra.xml
Russian	4230660.female.13.Student.Virgo.xml
Croatian	817097.female.26.Student.Taurus.xml
Norwegian	887044.female.23.indUnk.Pisces.xml

Some dates are missing with `,,` in the date field. We set such dates to `0/0/0`.

Looking at some of the XML files with non-English month names we see that there are some post written in languages other than English. The data set providers only checked whether a blog (not a post) contains at least 200 common English words. Thus, we have to remove some posts. Language detection can be done with the `langdetect` module[552].

```
# cell execution may take several hours due to language detection

month_num = {'january': 1, 'february': 2, 'march': 3, 'april': 4, 'may': 5, 'june
↪': 6,
             'july': 7, 'august': 8, 'september': 9, 'october': 10, 'november':␣
↪11, 'december': 12,
             # French
             'janvier': 1, 'mars': 3, 'avril': 4, 'mai': 5, 'juin': 6,
             'juillet': 7, 'septembre': 9, 'octobre': 10, 'novembre': 11,
             # Spanish
             'enero': 1, 'febrero': 2, 'marzo': 3, 'abril': 4, 'mayo': 5, 'junio
↪': 6,
             'julio': 7, 'agosto': 8, 'septiembre': 9, 'octubre': 10, 'noviembre
↪': 11, 'diciembre': 12,
             # Portuguese
             'janeiro': 1, 'fevereiro': 2, 'maio': 5, 'junho': 6,
```

(continues on next page)

[552] https://github.com/Mimino666/langdetect

(continued from previous page)

```
              'julho': 7, 'agosto': 8, 'setembro': 9, 'outubro': 10, 'novembro':␣
↪11, 'dezembro': 12,
              # German
              'januar': 1, 'februar': 2, 'märz': 3, 'april': 4, 'mai': 5, 'juni':␣
↪6,
              'juli': 7, 'august': 8, 'september': 9, 'oktober': 10, 'november':␣
↪11, 'dezember': 12,
              # Estonian
              'jaanuar': 1, 'aprill': 4, 'juuni': 6, 'juuli': 7,
              # Italian
              'giugno': 6, 'luglio': 7, 'ottobre': 10,
              # Finnish
              'toukokuu': 5, 'elokuu': 8,
              # Dutch
              'maart': 3, 'mei': 5, 'augustus': 8,
              # Polish
              'maj': 5, 'czerwiec': 6, 'lipiec': 7,
              # Romanian
              'ianuarie': 1, 'februarie': 2, 'iulie': 7, 'septembrie': 9,
↪'noiembrie': 11,
              # Swedish
              'augusti': 8,
              # Russian
              'avgust': 8,
              # Croatian
              'lipanj': 6, 'kolovoz': 8,
              # Norwegian
              'mars': 3, 'desember': 12,
              # unknown
              'unknown': 0}


blog_ids = []
days = []
months = []
years = []
texts = []
langs = []

with zipfile.ZipFile(data_path + 'blogs.zip') as zf:

    for file_name in zf.namelist():

        if file_name.split('.')[-1] != 'xml':
            print('skipping', file_name)
            continue

        #print(file_name)

        blog_id = int(file_name.split('/')[-1].split('.')[0])

        with zf.open(file_name) as f:
            xml = f.read().decode(encoding='iso-8859-1')

        xml_posts = xml.split('<date>')[1:]

        for xml_post in xml_posts:

            day, month, year = xml_post[:(xml_post.find('</date>'))].split(',')
            if len(day) == 0:
                day = '0'
```

(continues on next page)

(continued from previous page)

```
            if len(month) == 0:
                month = 'unknown'
            if len(year) == 0:
                year = '0'

            text = xml_post[(xml_post.find('<post>') + 6):(xml_post.find('</post>
↪'))]
            text = re.sub(r'[\x00-\x1F]+', ' ', text)     # non-printable␣
↪characters
            text = text.replace('&nbsp;', ' ')     # HTML entity for protected␣
↪spaces
            text = text.replace('urlLink', '')     # link marker
            text = text.strip()

            try:
                lang = langdetect.detect(text)
            except langdetect.LangDetectException:
                lang = ''

            if len(text) > 0:
                blog_ids.append(blog_id)
                days.append(int(day))
                months.append(month_num[month.lower()])
                years.append(int(year))
                texts.append(text)
                langs.append(lang)

posts = pd.DataFrame(data={'blog_id': blog_ids, 'day': days, 'month': months,
↪'year': years,
                           'text': texts, 'lang': langs})
posts
```

```
skipping blogs/
```

```
        blog_id  day  month  year  \
0       1000331   31      5  2004
1       1000331   29      5  2004
2       1000331   28      5  2004
3       1000331   28      5  2004
4       1000331   28      5  2004
...         ...  ...    ...   ...
676023   999503    4      7  2004
676024   999503    3      7  2004
676025   999503    2      7  2004
676026   999503    1      7  2004
676027   999503    1      7  2004

                                                     text lang
0       Well, everyone got up and going this morning. ...   en
1       My four-year old never stops talking.  She'll ...   en
2       Actually it's not raining yet, but I bought 15...   en
3       Ha! Just set up my RSS feed - that is so easy!...   en
4       Oh, which just reminded me, we were talking ab...   en
...                                                   ...  ...
676023  Today we celebrate our independence day.    In...   en
676024  Ugh, I think I have allergies...  My nose has ...   en
676025  "Science is like sex; occasionally something p...   en
676026  Dog toy or marital aid   I managed 10/14 on th...   en
676027  I had a dream last night about a fight when I ...   en
```

(continues on next page)

(continued from previous page)

```
[676028 rows x 6 columns]
```

```
print(len(posts))

posts = posts.loc[posts['lang'] == 'en', :]
posts = posts.drop(columns=['lang'])

print(len(posts))
```

```
676028
653764
```

```
posts.to_csv(data_path + 'posts.csv')
```

```
posts = pd.read_csv(data_path + 'posts.csv', index_col=0)

posts
```

```
        blog_id  day  month  year  \
0       1000331   31      5  2004
1       1000331   29      5  2004
2       1000331   28      5  2004
3       1000331   28      5  2004
4       1000331   28      5  2004
...         ...  ...    ...   ...
676023   999503    4      7  2004
676024   999503    3      7  2004
676025   999503    2      7  2004
676026   999503    1      7  2004
676027   999503    1      7  2004

                                                     text
0       Well, everyone got up and going this morning. ...
1       My four-year old never stops talking.  She'll ...
2       Actually it's not raining yet, but I bought 15...
3       Ha! Just set up my RSS feed - that is so easy!...
4       Oh, which just reminded me, we were talking ab...
...                                                   ...
676023  Today we celebrate our independence day.    In...
676024  Ugh, I think I have allergies...  My nose has ...
676025  "Science is like sex; occasionally something p...
676026  Dog toy or marital aid   I managed 10/14 on th...
676027  I had a dream last night about a fight when I ...

[653702 rows x 5 columns]
```

31.1.2 Exploring the Data Set

Now we have two data frames: `blogs` and `posts`. We should have a look at the data before tackling the learning task.

```
print('blogs:', len(blogs))
print('posts:', len(posts))
```

```
blogs: 19320
posts: 653702
```

Exploring Blog Authors

```
blogs.groupby('gender').count()
```

```
         age  industry
gender
f       9660      9660
m       9660      9660
```

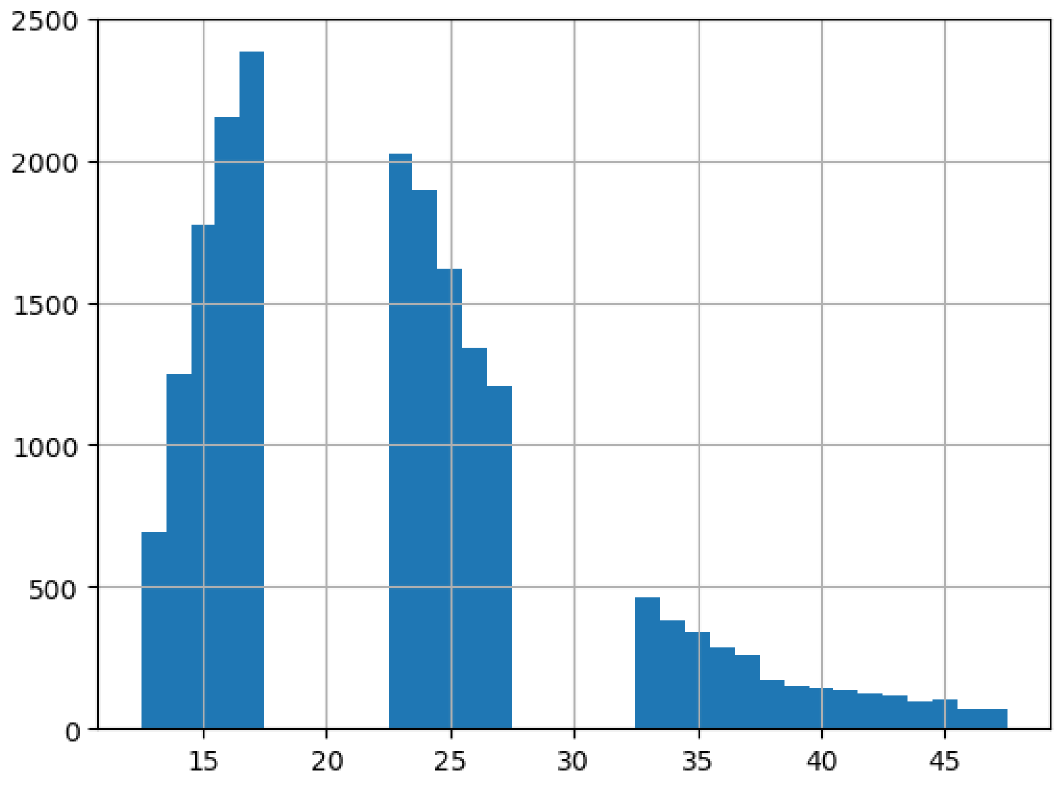
The data set is well balanced with respect to blog author's gender.

```
blogs.groupby('age').count()
```

```
     gender  industry
age
13      690       690
14     1246      1246
15     1771      1771
16     2152      2152
17     2381      2381
23     2026      2026
24     1895      1895
25     1620      1620
26     1340      1340
27     1205      1205
33      464       464
34      378       378
35      338       338
36      288       288
37      259       259
38      171       171
39      152       152
40      145       145
41      139       139
42      127       127
43      116       116
44       94        94
45      103       103
46       72        72
47       71        71
48       77        77
```

```
blogs['age'].hist(bins=np.arange(13, 49)-0.5)
```

```
<Axes: >
```



We have three age groups of different size:

```
print(np.count_nonzero((blogs['age'] < 20).to_numpy()))
print(np.count_nonzero(((blogs['age'] > 20) & (blogs['age'] < 30)).to_numpy()))
print(np.count_nonzero((blogs['age'] > 30).to_numpy()))
```

```
8240
8086
2994
```

According to the data set description gender should be balanced in each age group.

```
blogs['age_group'] = pd.cut(blogs['age'], bins=[0, 20, 30, 100])

blogs.groupby(['age_group', 'gender']).count()
```

```
/tmp/ipykernel_19434/2441672544.py:3: FutureWarning: The default of␣
↪observed=False is deprecated and will be changed to True in a future version␣
↪of pandas. Pass observed=False to retain current behavior or observed=True to␣
↪adopt the future default and silence this warning.
  blogs.groupby(['age_group', 'gender']).count()
```

```
                    age  industry
age_group gender
(0, 20]   f        4120      4120
          m        4120      4120
(20, 30]  f        4043      4043
          m        4043      4043
```

(continues on next page)

(continued from previous page)

```
(30, 100] f       1497      1497
          m       1497      1497
```

```
blogs.groupby('industry').count()['age'].sort_values(ascending=False)
```

```
industry
unknown                    6827
Student                    5120
Education                   980
Technology                  943
Arts                        721
Communications-Media        479
Internet                    397
Non-Profit                  372
Engineering                 312
Government                  236
Law                         197
Consulting                  191
Science                     184
Marketing                   180
BusinessServices            163
Publishing                  150
Advertising                 145
Religion                    139
Telecommunications          119
Military                    116
Banking                     112
Accounting                  105
Fashion                      98
Tourism                      94
HumanResources               94
Transportation               91
Sports-Recreation            90
Manufacturing                87
Architecture                 69
Chemicals                    62
Biotech                      57
LawEnforcement-Security      57
RealEstate                   55
Museums-Libraries            55
Construction                 55
Automotive                   54
Agriculture                  36
InvestmentBanking            33
Environment                  28
Maritime                     17
Name: age, dtype: int64
```

Industry is available for about 60 per cent of the blog authors.

Exploring Blog Posts

Although for our classification task dates of posts are irrelevant, we have a look at them. Looking at irrelevant columns yields a better feeling for the data set and its reliability.

```
posts.groupby('year').count()
```

```
      blog_id     day   month    text
year                                 
0          23      23      23      23
1999       68      68      68      68
2000      866     866     866     866
2001     5427    5427    5427    5427
2002    21772   21772   21772   21772
2003   100199  100199  100199  100199
2004   525318  525318  525318  525318
2005        6       6       6       6
2006       23      23      23      23
```

The data set providers scraped the data in August 2004. Thus, there should be no newer posts. Since we only are interested in post texts, we do not care about this inconsistency here.

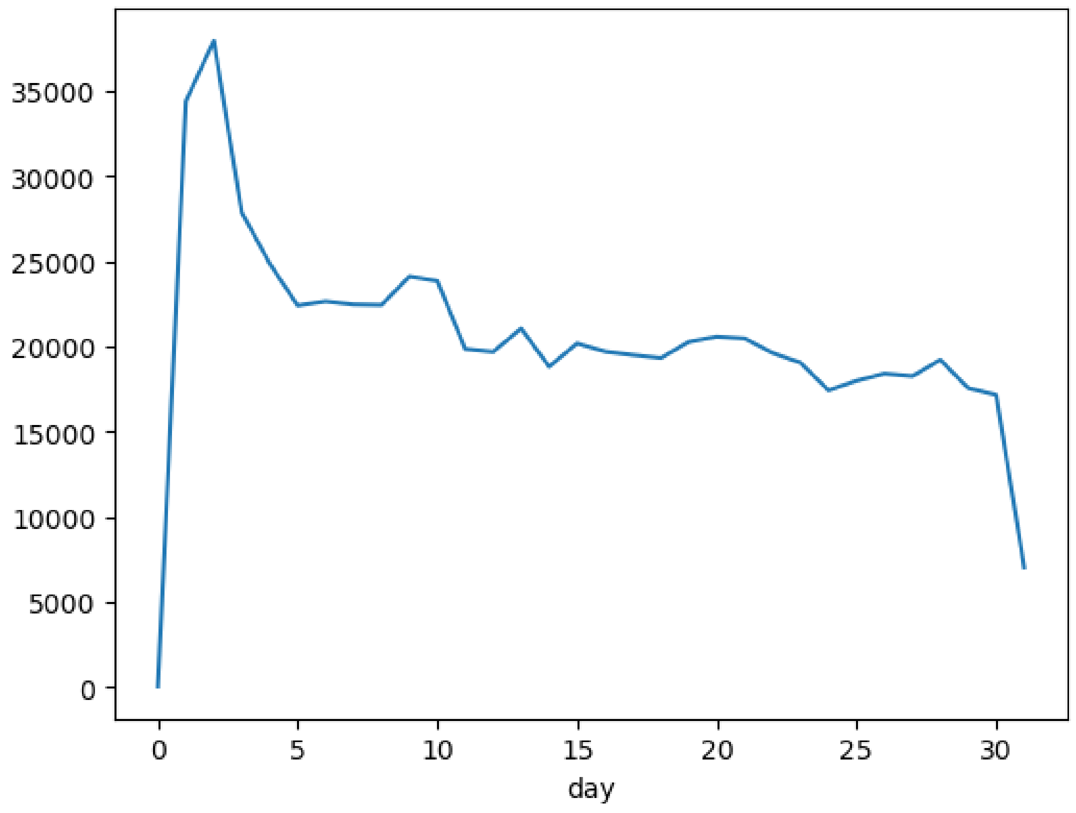
```
posts.groupby('month').count()
```

```
       blog_id     day    year    text
month                                 
0           23      23      23      23
1        21812   21812   21812   21812
2        24663   24663   24663   24663
3        29099   29099   29099   29099
4        33166   33166   33166   33166
5        75275   75275   75275   75275
6       125797  125797  125797  125797
7       154842  154842  154842  154842
8       125919  125919  125919  125919
9        13051   13051   13051   13051
10       16296   16296   16296   16296
11       16525   16525   16525   16525
12       17234   17234   17234   17234
```

The maximum in summer months is not because people write more blog posts in summer. Blogging became more and more popular from month to month and posts had been collected till August 2004. Including posts from September 2004 we (presumable) would get September counts higher than August counts. Slight drop from July to August could be caused by incomplete data for August 2004.

```
posts.groupby('day').count()['blog_id'].plot()
```

```
<Axes: xlabel='day'>
```



There are more posts at the beginning of a month than at a month's end. Counts for 31st are much lower because not every month has a 31st.

We should have a look at class balancing. We already know that gender is well balanced and age is not if we count on a per-blog basis. But since we want to classify blog posts (not complete blogs) by gender and age of the author we have to consider class sizes on a per post-basis.

```
posts_per_blog = posts.groupby('blog_id')['day'].count()

blogs['posts'] = 0
blogs.loc[posts_per_blog.index, 'posts'] = posts_per_blog

blogs.groupby(['gender', 'age_group'])['posts'].sum()
```

```
/tmp/ipykernel_19434/1016594775.py:6: FutureWarning: The default of␣
↪observed=False is deprecated and will be changed to True in a future version␣
↪of pandas. Pass observed=False to retain current behavior or observed=True to␣
↪adopt the future default and silence this warning.
  blogs.groupby(['gender', 'age_group'])['posts'].sum()
```

```
gender  age_group
f       (0, 20]      110387
        (20, 30]     155425
        (30, 100]     56786
m       (0, 20]      115255
        (20, 30]     152739
        (30, 100]     63110
Name: posts, dtype: int64
```

In the highest age group gender is somewhat unbalanced, but not much. An even more accurate measure of class size (data per class) is the cummulated text length per class.

```
posts['length'] = posts['text'].str.len()
chars_per_blog = posts.groupby('blog_id')['length'].sum()

blogs['chars'] = 0
blogs.loc[chars_per_blog.index, 'chars'] = chars_per_blog

blogs.groupby(['gender', 'age_group'])['chars'].sum()
```

```
/tmp/ipykernel_19434/1431541158.py:7: FutureWarning: The default of␣
↪observed=False is deprecated and will be changed to True in a future version␣
↪of pandas. Pass observed=False to retain current behavior or observed=True to␣
↪adopt the future default and silence this warning.
 blogs.groupby(['gender', 'age_group'])['chars'].sum()
```

```
gender  age_group
f       (0, 20]      117123221
        (20, 30]     179750611
        (30, 100]     74878617
m       (0, 20]      119793206
        (20, 30]     175919412
        (30, 100]     72894990
Name: chars, dtype: int64
```

Here balancing of gender looks better, but again the highest age group is much smaller than the other two age groups.

31.1.3 Preprocessing Text for Counting Words

Machine learning algorithms expect numbers as inputs. So we have to convert strings to vectors of numbers. There exist different conversion techniques, some advanced ones like Word2vec[553] and some simpler ones like the *bag of words* approach. The latter assigns each word in a corpus a position in a vector and represents a string by counting the occurrences of each word. The vector representation of a string is the vector containing all word counts.

Input features are word counts and length of feature vectors equals the number of different words in a dictionary. Thus, feature vectors are extremely long and contain zeros almost everywhere. Vectors containing zeros almost everywhere are called *sparse vectors*. A sparse vectors is not stored as array, but as list of index-value pairs for non-zero components only. Memory consumption is not given by vector length but by the number of non-zero components. Scikit-Learn and NumPy support sparse vectors (and matrices) and automatically choose a suitable data type where appropriate.

The dictionary (and, thus, the feature space dimension) is determined from the training set. All words contained in the training set form the dictionary. Usually one leaves out words occurring only in very few training samples or words occurring in almost all training samples. From the former a model cannot learn something useful due to lack of samples. The latter do not contain useful information to discriminate between different classes.

Before converting strings to vectors some preprocessing is necessary. At least punctuation and other special characters should be removed. Other preprocessing steps may include:

- *Stop word removal:* Remove common words like ‘and’, ‘or’, ‘have’. There exist list of stop words for most languages. Stop word removal has to be used with care, because some common words may contain important information, like ‘not’ for instance.
- *Stemming:* Remove word endings like plural ‘s’ or ‘ing’ to get word stems. There exist many different stemming algorithms. Results are sometimes incorrect. For instance, ‘thus’ is usually stemmed to ‘thu’.
- *Lemmatization:* Get the base form of a word. It’s a more intelligent form of stemming, but requires lots of computation time. Again, there exist many different algorithms.

[553] https://en.wikipedia.org/wiki/Word2vec

Stop words, stemming, and lemmatization are, for instance, implemented in the `nltk` Python package (Natural Language Toolkit)[554]. The subject is known as *natural language processing*.

Removing Punctuation and other Special Characters

We remove all characters but A-Z, numbers, single spaces, and basic punctuation (!, ?, dot, comma, apostrophe). Regular expressions[555] allow for efficient removal.

```
posts['text'] = posts['text'].str.replace(r"[^\w ,\.\?!']", '', regex=True)

posts['text'] = posts['text'].str.replace(r'\s+', ' ', regex=True)
```

```
posts['text']
```

```
0         Well, everyone got up and going this morning. ...
1         My fouryear old never stops talking. She'll sa...
2         Actually it's not raining yet, but I bought 15...
3         Ha! Just set up my RSS feed that is so easy! W...
4         Oh, which just reminded me, we were talking ab...
                                ...
676023    Today we celebrate our independence day. In ho...
676024    Ugh, I think I have allergies... My nose has b...
676025    Science is like sex occasionally something pra...
676026    Dog toy or marital aid I managed 1014 on this ...
676027    I had a dream last night about a fight when I ...
Name: text, Length: 653702, dtype: object
```

Maybe some texts are empty now. We should remove them.

```
print(len(posts))

posts = posts.loc[posts.loc[:, 'text'] != '', :]

print(len(posts))
```

```
653702
653702
```

Lemmatization

To reduce dictionary size and increase chances for good classification results we use lemmatization. For instance we want to count 'child' and 'children' as one and the same word. We choose the `WordNetLemmatizer`[556] of NLTK. WordNet[557] is a database provided by Princeton University which contains relations between English words.

The `WordNetLemmatizer` takes a word and looks it up in the data base. If it is found there, it returns the base form, else the original word is returned. WordNet data base can be searched online[558], too. Searching WordNet database with NLTK or online includes some stemming-like preprocessing steps[559].

```
import nltk
```

Before first use of `WordNetLemmatizer` we have to download the database.

[554] https://www.nltk.org
[555] https://docs.python.org/3/library/re.html#regular-expression-syntax
[556] https://www.nltk.org/api/nltk.stem.wordnet.html
[557] https://wordnet.princeton.edu
[558] http://wordnetweb.princeton.edu/perl/webwn
[559] https://wordnet.princeton.edu/documentation/morphy7wn

```
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /var/lib/u21302575108/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

```
True
```

We have to create a `WordNetLemmatizer` object and then call its `lemmatize` method.

```
lemmatizer = nltk.stem.WordNetLemmatizer()
```

```
lemmatizer.lemmatize('child')
```

```
'child'
```

```
lemmatizer.lemmatize('children')
```

```
'child'
```

Note that the WordNet lemmatizer only works for lower case words (a not well documented fact).

```
lemmatizer.lemmatize('Children')
```

```
'Children'
```

Simply calling `WordNetLemmatizer` with some word may yield unexpected results. Given the sentence ‘He is killing him.’ we would expect ‘killing’ to be lemmatized to ‘kill’.

```
lemmatizer.lemmatize('killing')
```

```
'killing'
```

The problem here is that ‘killing’ is the base form of a noun (‘That resulted in a killing.’) and `WordNetLemmatizer` by default looks for nouns. A second argument to `lemmatize` modifies the default behavior.

```
lemmatizer.lemmatize('killing', pos=nltk.corpus.reader.wordnet.VERB)
```

```
'kill'
```

The abbreviation ‘pos’ stands for ‘part of speech’. The module `nltk.corpus.reader.wordnet` contains some WordNet related functionality. It defines some constants, for instance. Passing `nltk.corpus.reader.wordnet.NOUN` to `pos` (the default) tells the lemmatizer that the word is a noun. Passing `nltk.corpus.reader.wordnet.VERB` tells it that the word is a verb. Further options are `nltk.corpus.reader.wordnet.ADJ` (adjectives) and `nltk.corpus.reader.wordnet.ADV` (adverbs).

```
print(nltk.corpus.reader.wordnet.NOUN)
print(nltk.corpus.reader.wordnet.VERB)
print(nltk.corpus.reader.wordnet.ADJ)
print(nltk.corpus.reader.wordnet.ADV)
```

```
n
v
a
r
```

Although these are simple strings, we should use the constants. If implementation of NLTK oder WordNet changes, our code is more likely to remain working then.

The question now is: How to obtain POS information? NLTK implements several *POS taggers*. If you do not want to decide which one to choose, use the one recommended by NLTK by simply calling `pos_tag()`. This function takes a list of words and punctuation symbols as argument. Such a list can be generated by calling `word_tokenize()`. Again several tokenization algorithms are available and `word_tokenize()` uses the recommended one.

To use tokenization and tagging we have to download some NLTK data. The data to download may change if NLTK recommends other algorithms in future. But corresponding methods will show a warning if required data is not available and the warning message contains the code for downloading.

```
nltk.download('punkt')     # for tokenization
nltk.download('averaged_perceptron_tagger')     # for POS tagging
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /var/lib/u21302575108/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /var/lib/u21302575108/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
True
```

Tokenization and tagging (in theory) could be implemented as follows.

```
posts['tokenized'] = None
posts['tagged'] = None

for idx in posts.index:
    posts.loc[idx, 'tokenized'] = nltk.tokenize.word_tokenize(posts.loc[idx, 'text
↪'])
    posts.loc[idx, 'tagged'] = nltk.tag.pos_tag(posts.loc[idx, 'tokenized'])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[39], line 5
      2 posts['tagged'] = None
      4 for idx in posts.index:
----> 5     posts.loc[idx, 'tokenized'] = nltk.tokenize.word_tokenize(posts.
↪loc[idx, 'text'])
      6     posts.loc[idx, 'tagged'] = nltk.tag.pos_tag(posts.loc[idx,
↪'tokenized'])

File /opt/conda/envs/python3/lib/python3.11/site-packages/pandas/core/indexing.
↪py:849, in _LocationIndexer.__setitem__(self, key, value)
    846 self._has_valid_setitem_indexer(key)
    848 iloc = self if self.name == "iloc" else self.obj.iloc
--> 849 iloc._setitem_with_indexer(indexer, value, self.name)

File /opt/conda/envs/python3/lib/python3.11/site-packages/pandas/core/indexing.
↪py:1835, in _iLocIndexer._setitem_with_indexer(self, indexer, value, name)
   1832 # align and set the values
```

(continues on next page)

(continued from previous page)

```
   1833 if take_split_path:
   1834     # We have to operate column-wise
-> 1835     self._setitem_with_indexer_split_path(indexer, value, name)
   1836 else:
   1837     self._setitem_single_block(indexer, value, name)

File /opt/conda/envs/python3/lib/python3.11/site-packages/pandas/core/indexing.
↪py:1891, in _iLocIndexer._setitem_with_indexer_split_path(self, indexer,␣
↪value, name)
   1886     if len(value) == 1 and not is_integer(info_axis):
   1887         # This is a case like df.iloc[:3, [1]] = [0]
   1888         #  where we treat as df.iloc[:3, 1] = 0
   1889         return self._setitem_with_indexer((pi, info_axis[0]), value[0])
-> 1891     raise ValueError(
   1892         "Must have equal len keys and value "
   1893         "when setting with an iterable"
   1894     )
   1896 elif lplane_indexer == 0 and len(value) == len(self.obj.index):
   1897     # We get here in one case via .loc with a all-False mask
   1898     pass

ValueError: Must have equal len keys and value when setting with an iterable
```

This code results in an error which is somewhat hard to figure out. If on the right-hand side of an assignment to some Pandas object is an iterable, then Pandas expects a same-sized iterable on the left-hand side. Thus, it is not possible to assign, for instance, a list to a cell of a data frame. There exist several more or less complicated workarounds, which all are rather inefficient. Thus, we use the following code, which requires two for loops (list comprehensions) instead of one.

```
# cell execution takes several minutes

posts['tokenized'] = [nltk.tokenize.word_tokenize(text) for text in posts['text']]
```

```
posts['tokenized']
```

```
0         [Well, ,, everyone, got, up, and, going, this,...
1         [My, fouryear, old, never, stops, talking, ., ...
2         [Actually, it, 's, not, raining, yet, ,, but, ...
3         [Ha, !, Just, set, up, my, RSS, feed, that, is...
4         [Oh, ,, which, just, reminded, me, ,, we, were...
                                ...
676023    [Today, we, celebrate, our, independence, day,...
676024    [Ugh, ,, I, think, I, have, allergies, ..., My...
676025    [Science, is, like, sex, occasionally, somethi...
676026    [Dog, toy, or, marital, aid, I, managed, 1014,...
676027    [I, had, a, dream, last, night, about, a, figh...
Name: tokenized, Length: 653702, dtype: object
```

```
# cell execution takes an hour and requires 25 GB of memory

posts['tagged'] = [nltk.tag.pos_tag(tokens) for tokens in posts['tokenized']]
```

```
posts['tagged']
```

```
0         [(Well, RB), (,, ,), (everyone, NN), (got, VBD...
1         [(My, PRP$), (fouryear, JJ), (old, JJ), (never...
```

(continues on next page)

(continued from previous page)

```
2         [(Actually, RB), (it, PRP), ('s, VBZ), (not, R...
3         [(Ha, NNP), (!, .), (Just, RB), (set, VBN), (u...
4         [(Oh, UH), (,, ,), (which, WDT), (just, RB), (...
                                ...
676023    [(Today, NN), (we, PRP), (celebrate, VBP), (ou...
676024    [(Ugh, NNP), (,, ,), (I, PRP), (think, VBP), (...
676025    [(Science, NN), (is, VBZ), (like, IN), (sex, N...
676026    [(Dog, NNP), (toy, NN), (or, CC), (marital, JJ...
676027    [(I, PRP), (had, VBD), (a, DT), (dream, NN), (...
Name: tagged, Length: 653702, dtype: object
```

The problem now is to translate NLTK POS tags to WordNet POS tags. Have a look at the list of NLTK POS tags[560]. There we see the following relations:

NLTK POS tag	WordNet POS tag
JJ…	ADJ
RB…	ADV
NN…	NOUN
VB…	VERB

All NLTK POS tags have at least two characters. So we may use the following conversion function.

```python
def NLTKPOS_to_WordNetPOS(tag):

    if tag[0:2] == 'JJ':
        return nltk.corpus.reader.wordnet.ADJ
    elif tag[0:2] == 'RB':
        return nltk.corpus.reader.wordnet.ADV
    elif tag[0:2] == 'NN':
        return nltk.corpus.reader.wordnet.NOUN
    elif tag[0:2] == 'VB':
        return nltk.corpus.reader.wordnet.VERB
    else:
        return None
```

Tokens with NLTK POS tags not present in WordNet can be removed, because they do not carry much information about the text. Here we have to keep in mind that our model will be based on word counts. So no relations between words are considered. For our model the sentence 'John is in the house.' will be the same as 'The house is in John'. For more advanced models, relations between words, and thus word classes other than adjectives, adverbs, verbs, nouns, may be of importance.

Note that the `lemmatize` function always returns some word. If a word is not found in the WordNet data base, then the orignal word is returned. If we want to sort out words not contained in WordNet we have use a trick. Looking at the source code of `lemmatize`[561] we see that the function calls `nltk.corpus.wordnet._morphy`. The `_morphy` function returns a (possibly empty) list of lemmas found in WordNet. If `_morphy` returns an empty list, we know that the word under consideration is something unusual (contains typos, for instance) and should be ignored. Else we use the first lemma in the list.

In principle, this approach is good, but there's a snag to it: If we pass the wrong POS tag to `_morphy` we won't get a result.

```python
nltk.corpus.wordnet._morphy('children', nltk.corpus.reader.wordnet.VERB)
```

```
[]
```

[560] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
[561] https://www.nltk.org/_modules/nltk/stem/wordnet.html

So we have to refine our strategy. If `_morphy` returns an empty list, we call `_morphy` with all possible POS tags. If all returned lists are empty, then we can be relatively sure that the word is something unsual and, thus, irelevant for our classification task.

Another issue is that WordNet does not lemmatize words containing apostrophes. For *she's* or *havn't* that's not a real problem, because such words are of little importance for our classification tasks. But what about *mama's*, for instance? We should remove all occurences of *'s*.

```
# cell execution may take several hours

print_max = 1000
printed = 0

posts['lemmatized'] = None

for idx in posts.index:
    lemmas = []     # list of all lemmatized words of current post
    for token, tag in posts.loc[idx, 'tagged']:
        modified_token = token.lower().replace("'s", '')

        wordnet_tag = NLTKPOS_to_WordNetPOS(tag)
        if not (wordnet_tag is None):

            morphy_result = nltk.corpus.wordnet._morphy(modified_token, wordnet_
 ↪tag)
            if len(morphy_result) > 0:
                lemmas.append(morphy_result[0])
            else:
                morphy_result_all = nltk.corpus.wordnet._morphy(modified_token,␣
 ↪nltk.corpus.reader.wordnet.NOUN) \
                                    + nltk.corpus.wordnet._morphy(modified_token,␣
 ↪nltk.corpus.reader.wordnet.VERB) \
                                    + nltk.corpus.wordnet._morphy(modified_token,␣
 ↪nltk.corpus.reader.wordnet.ADJ) \
                                    + nltk.corpus.wordnet._morphy(modified_token,␣
 ↪nltk.corpus.reader.wordnet.ADV)
                if len(morphy_result_all) > 0:
                    lemmas.append(morphy_result_all[0])
                else:
                    if printed < print_max:
                        print(modified_token, end=', ')
                        printed += 1

    posts.loc[idx, 'lemmatized'] = ' '.join(lemmas)

posts['lemmatized']
```

```
everyone, , , everything, .., fouryear, ummm, ...., ummm, anything, , goldeyes,␣
 ↪n't, skydome, occassionally, goldeyes, n't, 'm, , n't, everyone, gameboy, n't,
 ↪ n't, n't, everything, 've, hmm, something, else, mcnally, something, anyone,
 ↪'ve, 've, breadmaker, n't, n't, 've, imdb, everything, something, something,
 ↪'m, ineveitable, n't, , n't, 'm, 've, kel, 'm, ya, di, amore, 'm, .., n't, n
 ↪'t, heh, , everyone, sleepyland, n't, 'm, ya, di, amore, whoopie, 've, umm, n
 ↪'t, epvm, n't, poopeyhead, nathan, 're, n't, nathan, nathan, alstadt, we, we,␣
 ↪dan, dave, we, dan, , , .., n't, 'm, thnk, 'm, epvm, umm, heh, n't, arg, , 'm,
 ↪ 'm, madsen, eek, umm, .., brett, alex, joanne, now.they, 're, alex, , gosh,␣
 ↪darnit, improv, 'm, 'm, 're, n't, ...., ya, di, amore, 'm, 'm, n't, ya, di,␣
 ↪amore, , something, grrr, , 'm, enjoyig, hey, catie, momly, n't, duper, n't,
 ↪'m, anyone, 'm, , thingamabobber, 're, 're, 're, 've, di, amore, n't, 'm, 'm,␣
 ↪, everyone, brandon, jake, n't, meh, , ok., my, n't, a., 'm, kane, ummm,␣
 ↪something, else, psh, n't, n't, heh, anything, 'm, ya, di, amore, 'm,␣
 ↪superduper, ...., , 're, contraversial, blahdeblahdeblah, nequa, naperville,
 ↪'re, 're, 're, n't, heck, kathryn, how, poopy, 'm, sleepyland, goodnight,␣
 ↪everyone, ya, di, amore, 'm, superduper, ...., , 're, contraversial,␣
 ↪blahdeblahdeblah, nequa, naperville, 're, 're, 're, n't, heck, kathryn, how,␣
 ↪poopy, 'm, sleepyland, goodnight, everyone, ya, di, amore, hey, alex, 'm,␣
 ↪baaack, 'm, something, everyone, hink, genevieve, n't, grr, pooey, , tootsie,␣
 ↪, catie, , n't, steve, chem, , , ah, passon, .., kast, hmmm, , anyone,␣
 ↪everything, 've, heh, spontanious, catie, rar, ah, n't, something, everything,
```

(continues on next page)

(continued from previous page)

```
0         well get up go morning still raining okay sort...
1         old never stop talk say mom say say oh yeah do...
2         actually not raining yet buy ticket game mom b...
3         ha just set r feed be so easy do do enough tod...
4         just remind be talk can food coffee break morn...
                                ...
676023    today celebrate independence day honor event g...
676024    think have allergy nose have be stuff week mak...
676025    science be sex occasionally practical come not...
676026    dog toy marital aid manage little quiz see wel...
676027    have dream last night fight be younger dad hea...
Name: lemmatized, Length: 653702, dtype: object
```

```
posts = posts[['blog_id', 'lemmatized']]

posts.to_csv(data_path + 'posts_lemmatized.csv')
```

```
posts
```

```
        blog_id                                        lemmatized
0       1000331  well get up go morning still raining okay sort...
1       1000331  old never stop talk say mom say say oh yeah do...
2       1000331  actually not raining yet buy ticket game mom b...
3       1000331  ha just set r feed be so easy do do enough tod...
4       1000331  just remind be talk can food coffee break morn...
...         ...                                               ...
676023   999503  today celebrate independence day honor event g...
676024   999503  think have allergy nose have be stuff week mak...
676025   999503  science be sex occasionally practical come not...
676026   999503  dog toy marital aid manage little quiz see wel...
676027   999503  have dream last night fight be younger dad hea...

[653702 rows x 2 columns]
```

```
posts_lemmatized = pd.read_csv(data_path + 'posts_lemmatized.csv', index_col=0,␣
 ↪nrows=100)

posts['lemmatized'] = posts_lemmatized['lemmatized']
```

```
idx = 0

print(posts.loc[0, 'lemmatized'])
```

```
well get up go morning still raining okay sort suit mood easily have stay home␣
 ↪bed book cat have be lot rain people have wet basement be lake be golf course␣
 ↪fields be green green green be suppose be degree friday be deal mosquito next␣
 ↪week hear winnipeg describe old testament city cbc radio one last week sort␣
 ↪rings true flood infestation
```

We could improve preprocessing by tagging geographical locations, names of persons, and so on. But somewhere one has to stop. Let's see what a machine learning model can learn from our (not perfectly) preprocessed data…

31.2 Training

In this second chapter on text classification we train several machine learning models on the preprocessed data from the first chapter.

31.2.1 Loading Data

```
import pandas as pd
import numpy as np

data_path = '/data/datasets/blogs/'
```

```
blogs = pd.read_csv(data_path + 'blogs.csv', index_col=0)

blogs
```

```
        gender  age            industry
1000331      f   37             unknown
1000866      f   17             Student
1004904      m   23                Arts
1005076      f   25                Arts
1005545      m   25         Engineering
...        ...  ...                 ...
996147       f   36  Telecommunications
997488       m   25             unknown
998237       f   16             unknown
998966       m   27             unknown
999503       m   25            Internet

[19320 rows x 3 columns]
```

```
posts = pd.read_csv(data_path + 'posts.csv', index_col=0)
posts_lemmatized = pd.read_csv(data_path + 'posts_lemmatized.csv', index_col=0)

posts['lemmatized'] = posts_lemmatized['lemmatized']

posts
```

```
        blog_id  day  month  year  \
0       1000331   31      5  2004
1       1000331   29      5  2004
2       1000331   28      5  2004
3       1000331   28      5  2004
4       1000331   28      5  2004
...         ...  ...    ...   ...
676023   999503    4      7  2004
676024   999503    3      7  2004
676025   999503    2      7  2004
676026   999503    1      7  2004
676027   999503    1      7  2004

                                                     text  \
0       Well, everyone got up and going this morning. ...
1       My four-year old never stops talking.  She'll ...
2       Actually it's not raining yet, but I bought 15...
3       Ha! Just set up my RSS feed - that is so easy!...
```

(continues on next page)

(continued from previous page)

```
4       Oh, which just reminded me, we were talking ab...
...                                                   ...
676023  Today we celebrate our independence day.    In...
676024  Ugh, I think I have allergies...  My nose has ...
676025  "Science is like sex; occasionally something p...
676026  Dog toy or marital aid   I managed 10/14 on th...
676027  I had a dream last night about a fight when I ...

                                               lemmatized
0       well get up go morning still raining okay sort...
1       old never stop talk say mom say say oh yeah do...
2       actually not raining yet buy ticket game mom b...
3       ha just set r feed be so easy do do enough tod...
4       just remind be talk can food coffee break morn...
...                                                   ...
676023  today celebrate independence day honor event g...
676024  think have allergy nose have be stuff week mak...
676025  science be sex occasionally practical come not...
676026  dog toy marital aid manage little quiz see wel...
676027  have dream last night fight be younger dad hea...

[653702 rows x 6 columns]
```

Some posts do not contain any lemmatized text. Note that Pandas interprets empty fields in a CSV file as `nan`. So we may run into troubles if we do not replace `nan` by empty strings or remove the rows from the data frame.

```
posts.loc[posts['lemmatized'].isna(), :]
```

```
        blog_id  day  month  year  \
830     1004904   19      6  2004
1166    1008329   10      4  2004
1756    1011311   25      5  2004
3882    1022086    6      7  2004
3885    1022086    6      7  2004
...         ...  ...    ...   ...
673231   988941    3      2  2003
674144   988941   19      1  2004
674253   988941   25      3  2004
674723   992078   17     11  2003
675851   998237   28      8  2003

                                                   text lemmatized
830                                           J.A.M.C.        NaN
1166                  www.teenopendiary.com ----> elysicidal        NaN
1756                                       artie and me        NaN
3882                                     The three of us        NaN
3885                                        Amalia and I        NaN
...                                                  ...        ...
673231                                     I will, I will!        NaN
674144                                             What?        NaN
674253                                          YAYness!        NaN
674723  Ugh.     http://www.cnn.com/2003/US/Southwest/...        NaN
675851                     whoa im tlaking to kevin daly whoa        NaN

[968 rows x 6 columns]
```

```
posts.dropna(how='any', inplace=True)

posts
```

```
        blog_id  day  month  year  \
0       1000331   31      5  2004
1       1000331   29      5  2004
2       1000331   28      5  2004
3       1000331   28      5  2004
4       1000331   28      5  2004
...         ...  ...    ...   ...
676023   999503    4      7  2004
676024   999503    3      7  2004
676025   999503    2      7  2004
676026   999503    1      7  2004
676027   999503    1      7  2004

                                                     text  \
0       Well, everyone got up and going this morning. ...
1       My four-year old never stops talking.  She'll ...
2       Actually it's not raining yet, but I bought 15...
3       Ha! Just set up my RSS feed - that is so easy!...
4       Oh, which just reminded me, we were talking ab...
...                                                   ...
676023  Today we celebrate our independence day.    In...
676024  Ugh, I think I have allergies...  My nose has ...
676025  "Science is like sex; occasionally something p...
676026  Dog toy or marital aid   I managed 10/14 on th...
676027  I had a dream last night about a fight when I ...

                                               lemmatized
0       well get up go morning still raining okay sort...
1       old never stop talk say mom say say oh yeah do...
2       actually not raining yet buy ticket game mom b...
3       ha just set r feed be so easy do do enough tod...
4       just remind be talk can food coffee break morn...
...                                                   ...
676023  today celebrate independence day honor event g...
676024  think have allergy nose have be stuff week mak...
676025  science be sex occasionally practical come not...
676026  dog toy marital aid manage little quiz see wel...
676027  have dream last night fight be younger dad hea...

[652734 rows x 6 columns]
```

31.2.2 Counting Words

As discussed in the previous chapter we use the *bag of words* model: features are word counts and feature space dimension equals the size of the dictionary. Say we have m different words in our dictionary. Then the components $x^{(1)}, \ldots, x^{(m)}$ of a feature vector x count the occurrences of words $1, \ldots, m$ of the dictionary in a blog post.

The dictionary is created from the training data. Important: words not contained in the training data will be ignored by the model. So training data has to be sufficiently rich. To generate the dictionary we have to split data into training and test sets. But up to now we did not compose concrete training samples.

Model Inputs and Outputs

Model inputs will be vectors of word counts (or of numbers derived from word counts, see below). For the moment we only have strings. We arrange all the strings in a 1d NumPy array:

```
S = posts['lemmatized'].to_numpy()

print(S.shape)
S
```

```
(652734,)
```

```
array(['well get up go morning still raining okay sort suit mood easily have␣
 ↪stay home bed book cat have be lot rain people have wet basement be lake be␣
 ↪golf course fields be green green green be suppose be degree friday be deal␣
 ↪mosquito next week hear winnipeg describe old testament city cbc radio one␣
 ↪last week sort rings true flood infestation',
       'old never stop talk say mom say say oh yeah do lady bug hide rain hear␣
 ↪own voice very very exhaust now remember be go work sigh',
       'actually not raining yet buy ticket game mom birthday tonight be␣
 ↪suppose rain do cancel baseball game rain ballpark be beautiful ai use go jay␣
 ↪game live toronto really take kid game now do know blue jay ticket cost now␣
 ↪sure cheap here winnipeg oh just check definitely be',
       ..., 'science be sex occasionally practical come not reason do',
       'dog toy marital aid manage little quiz see well do',
       'have dream last night fight be younger dad heavy wrench brother hit␣
 ↪head be bleed bad hope be just dream'],
      dtype=object)
```

Model outputs will be class labels. We have 6 classes: all combinations of 2 genders and 3 age groups.

```
# add one column per class
posts['label'] = ''

# fill columns blogwise
for blog_id in blogs.index:

    # get class for blog
    label = blogs.loc[blog_id, 'gender']
    if blogs.loc[blog_id, 'age'] < 20:
        label = label + '1'
    elif blogs.loc[blog_id, 'age'] < 30:
        label = label + '2'
    else:
        label = label + '3'

    # set class for all posts of the blog
    posts.loc[posts['blog_id'] == blog_id, 'label'] = label

posts
```

```
        blog_id  day  month  year  \
0       1000331   31      5  2004
1       1000331   29      5  2004
2       1000331   28      5  2004
3       1000331   28      5  2004
4       1000331   28      5  2004
...         ...  ...    ...   ...
676023   999503    4      7  2004
```

(continues on next page)

(continued from previous page)

```
676024   999503    3      7  2004
676025   999503    2      7  2004
676026   999503    1      7  2004
676027   999503    1      7  2004

                                                      text  \
0       Well, everyone got up and going this morning. ...
1       My four-year old never stops talking.  She'll ...
2       Actually it's not raining yet, but I bought 15...
3       Ha! Just set up my RSS feed - that is so easy!...
4       Oh, which just reminded me, we were talking ab...
...                                                   ...
676023  Today we celebrate our independence day.    In...
676024  Ugh, I think I have allergies...  My nose has ...
676025  "Science is like sex; occasionally something p...
676026  Dog toy or marital aid   I managed 10/14 on th...
676027  I had a dream last night about a fight when I ...

                                                lemmatized label
0       well get up go morning still raining okay sort...    f3
1       old never stop talk say mom say say oh yeah do...    f3
2       actually not raining yet buy ticket game mom b...    f3
3       ha just set r feed be so easy do do enough tod...    f3
4       just remind be talk can food coffee break morn...    f3
...                                                   ...   ...
676023  today celebrate independence day honor event g...    m2
676024  think have allergy nose have be stuff week mak...    m2
676025  science be sex occasionally practical come not...    m2
676026  dog toy marital aid manage little quiz see wel...    m2
676027  have dream last night fight be younger dad hea...    m2

[652734 rows x 7 columns]
```

```
posts['label'] = posts['label'].astype('category')
posts['label'] = posts['label'].cat.reorder_categories(['f1', 'f2', 'f3', 'm1',
 ↪'m2', 'm3'])
print(posts['label'].cat.categories)
```

```
Index(['f1', 'f2', 'f3', 'm1', 'm2', 'm3'], dtype='object')
```

```
y = posts['label'].cat.codes.to_numpy()

y.shape
```

```
(652734,)
```

Train-Test Split

```
posts.loc[posts['lemmatized'].isna(), :]
```

```
Empty DataFrame
Columns: [blog_id, day, month, year, text, lemmatized, label]
Index: []
```

```
import sklearn.model_selection as model_selection
```

```
S_train, S_test, y_train, y_test = model_selection.train_test_split(S, y, test_
↪size=0.2)

S_train.shape, S_test.shape
```

```
((522187,), (130547,))
```

Simple Word Counting

Scikit-Learn implements dictionary generation and word counting in `sklearn.feature_extraction.text.CountVectorizer`[562]. All strings by default are converted to lower case. Scikit-Learn's `CountVectorizer` counts occurrences of words or group of words (*n-grams*). Counting words is the default. Stop words can be removed automatically based on a built-in list of English stop words.

Calling `fit` builds the dictionary. Calling `transform` counts words based on the dictionary. As usual `fit_transform` does both steps at once. But for `CountVectorizer` calling `fit_transform` is more efficient than calling `fit` and `transform` separately.

`CountVectorizer` provides some options for excluding rare or very frequent words. We will come back to those options below.

```
import sklearn.feature_extraction.text as fe_text
```

```
count_vect = fe_text.CountVectorizer()
X_train = count_vect.fit_transform(S_train)
```

```
X_train.shape
```

```
(522187, 50985)
```

We have more than 50000 different words in our vocabulary. The vocabulary is accessible through `Count_vect.vocabulary_`. It's a dict with strings as keys and corresponding indices as values.

```
count_vect.vocabulary_['sunshine']
```

```
43601
```

The feature with this index contains counts of 'sunshine'.

We should have a closer look on the word counts. There will be some words occurring only in very few post. From such rare words a machine learning model cannot learn something useful. On the other hand, words appearing in almost all posts do not contain information to differentiate between classes.

[562] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

The number of samples containing a fixed word is denoted as *document frequency (DF)* of the word. The word count within one sample is denoted as *term frequency (TF)* of the word (and the sample).

Let's have a look at document frequencies:

```
dfs = np.array(X_train.sign().sum(axis=0)).reshape(-1)
words = count_vect.vocabulary_.keys()
idxs = count_vect.vocabulary_.values()

vocabulary = pd.DataFrame({'word': words, 'idx': idxs})
vocabulary['df'] = dfs[vocabulary['idx'].to_numpy()]

vocabulary = vocabulary.sort_values('df', ascending=False)

vocabulary
```

```
                word    idx      df
5                 be   4005  439217
81              have  20248  332156
11                do  13220  297286
125              get  18599  244616
1                 go  18935  241553
...              ...    ...     ...
50948         taipeh  44256       1
50949      razorbill  36475       1
50950       porifera  34339       1
50951     reformable  36939       1
50952  periodontitis  33028       1

[50985 rows x 3 columns]
```

Note that X_train is not a usual NumPy array, but a sparse array. The sum method works as usual, but reshaping is not supported by sparse arrays. Thus, we convert the result of sum (which is much smaller than X_train) into a NumPy array before reshaping it.

Here is a list of words occuring only in very few posts:

```
vocabulary.loc[vocabulary['df'] < 5,'word']
```

```
39234          dashiki
39075          peerage
39076       pictograph
39079         diamante
39288               3d
              ...
50948           taipeh
50949        razorbill
50950         porifera
50951       reformable
50952    periodontitis
Name: word, Length: 13572, dtype: object
```

More than 13000 words appear in only 1 to 4 posts. Such rare words should be removed from the vocabulary. In other words, we may remove more than 13000 features, because they do not contain useful information.

Here are the words appearing most often together with the cut of posts containing them:

```
for idx in range(100):
    word = vocabulary['word'].iloc[idx]
    df = vocabulary['df'].iloc[idx]
    print(word, '({:.2f})'.format(df / X_train.shape[0]), end=' ')
```

```
be (0.84) have (0.64) do (0.57) get (0.47) go (0.46) not (0.44) so (0.43) just␣
↪(0.40) think (0.34) know (0.33) now (0.32) make (0.31) time (0.30) say (0.30)␣
↪more (0.27) see (0.27) really (0.27) well (0.26) good (0.26) come (0.25) then␣
↪(0.25) take (0.25) want (0.25) day (0.22) people (0.22) much (0.22) back (0.
↪22) work (0.21) today (0.21) here (0.21) only (0.21) look (0.21) even (0.20)␣
↪too (0.19) other (0.19) last (0.19) way (0.19) still (0.18) right (0.18) need␣
↪(0.17) new (0.17) love (0.17) tell (0.17) feel (0.17) things (0.17) try (0.
↪17) first (0.17) very (0.16) life (0.16) start (0.16) give (0.16) thing (0.
↪16) there (0.16) little (0.16) also (0.15) night (0.15) again (0.15) friend␣
↪(0.15) call (0.15) never (0.14) talk (0.14) leave (0.14) home (0.13) use (0.
↪13) most (0.13) long (0.13) week (0.13) let (0.13) mean (0.13) like (0.12)␣
↪keep (0.12) as (0.12) read (0.12) end (0.12) better (0.12) find (0.12) great␣
↪(0.12) guy (0.11) ever (0.11) always (0.11) next (0.11) few (0.11) write (0.
↪11) watch (0.11) post (0.11) hope (0.11) seem (0.11) actually (0.11) play (0.
↪11) put (0.11) sure (0.11) up (0.11) ask (0.11) bad (0.11) many (0.11) maybe␣
↪(0.11) one (0.11) happen (0.11) days (0.10) place (0.10)
```

Here we see that there are no words that appear in almost all posts. In principle, every word might be useful to differentiate between classes. Even if we would throw away words appearing in, for instance, more than 50 per cent of posts, we could remove only 3 features.

Scikit-Learn's `CountVectorizer()` takes parameters `min_df` and `max_df` to exclude rare words and words appearing very often, respectively.

```
count_vect = fe_text.CountVectorizer(min_df=5, max_df=1.0)
X_train = count_vect.fit_transform(S_train)

X_train.shape
```

```
(522187, 37413)
```

Weighted Counting

For short posts word counts will be lower than for longer posts. Thus, we should normalize word counts. Then feature values contain counts relative to the length of a post. Here we use `sklearn.preprocessing.normalize`[563].

```
import sklearn.preprocessing as preprocessing
```

```
X_train = preprocessing.normalize(X_train, norm='l1')
```

Another issue is the importance of words. The more posts contain a word, the less important for classifying posts the word is. So we could apply a weighting with weights the higher the lower the document frequency is. There exist different concrete weighting rules. The default one of Scikit-Learn is

$$\text{weight of word} = 1 + \log\frac{1 + \text{number of documents}}{1 + \text{document frequency of word}}.$$

If a word appears in all documents, the weight is 1. Else the weight is greater than 1. Sometimes the logarithm is used without adding 1. Then words appearing in all documents have weight 0, that is, they are ignored.

The vector of weighted counts usually is normalized as above.

Scikit-Learn implements this so called TF-IDF-weighting (TF: term frequency, IDF: inverse document frequency) in `sklearn.feature_extraction.text.TfidfVectorizer`[564]. Usage is identical to `CountVectorizer`.

[563] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html
[564] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

```
tfidf_vect = fe_text.TfidfVectorizer(min_df=5, max_df=1.0)
X_train = tfidf_vect.fit_transform(S_train)

X_train.shape
```

```
(522187, 37413)
```

Do not forget to transform test samples in the same way (no second call to `fit`!):

```
X_test = tfidf_vect.transform(S_test)

X_test.shape
```

```
(130547, 37413)
```

31.2.3 Naive Bayes Classifier

Feature vectors containing word counts follow a multinomial distribution, which is well suited for naive Bayes classification. Naive Bayes classifiers are very fast in training and prediction because only some emperical probabilities have to be calculated from sample counts. Thus, naive Bayes classifiers can cope with high dimensional feature spaces and large training data sets.

Strictly speaking, naive Bayes classifiers expect integer inputs (counts). But formulas allow for real-valued features, too. There is no theory backing multinomial naive Bayes for real-valued features. But experience shows that it works quite well. In context of bag of words language processing multinomial naive Bayes with TF-IDF values works much better than with simple word counts.

```
import sklearn.naive_bayes as naive_bayes
```

```
mnb = naive_bayes.MultinomialNB()
mnb.fit(X_train, y_train)
```

```
MultinomialNB()
```

Note that `MultinomialNB` has only very few parameters with default values suitable for our learning task. By default it uses Laplace smoothing and label probabilities are estimated from the training data (instead of uniform label distribution).

```
y_train_pred = mnb.predict(X_train)
y_test_pred = mnb.predict(X_test)
```

For visualizing classification accuracy we use the following function.

```
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn.metrics as metrics
```

```
def show_accuracy(y, y_pred):

    print('correct classification rate:          {:.4f} (random guessing: 0.1667)
↪'.format(metrics.accuracy_score(y, y_pred)))
    print('correct classification rate (gender): {:.4f} (random guessing: 0.5000)
↪'.format(metrics.accuracy_score(y // 3, y_pred // 3)))
    print('correct classification rate (age):    {:.4f} (random guessing: 0.3333)
↪'.format(metrics.accuracy_score(y - 3 * (y // 3), y_pred - 3 * (y_pred // 3))))
```

(continues on next page)

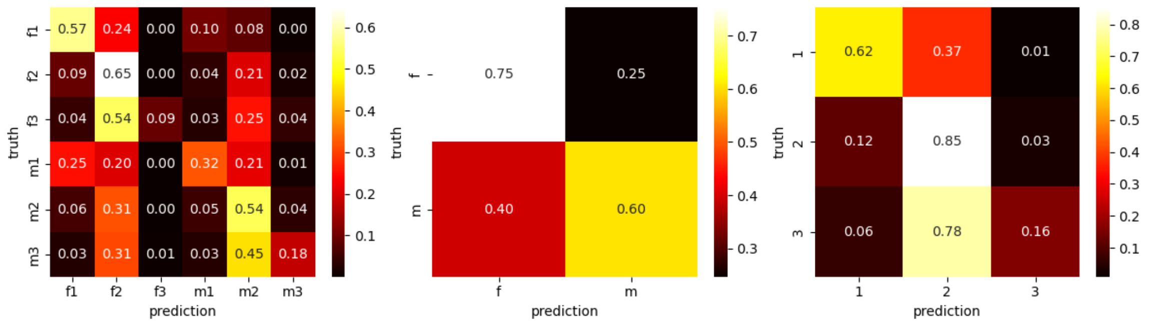
(continued from previous page)

```
    # calculate confusion matrices
    cm = pd.crosstab(pd.Series(y, name='truth'), pd.Series(y_pred, name=
↪'prediction'))
    cm = cm.reindex(index=[0, 1, 2, 3, 4, 5], columns=[0, 1, 2, 3, 4, 5], fill_
↪value=0)
    cm_gender = pd.crosstab(pd.Series(y // 3, name='truth'), pd.Series(y_pred //␣
↪3, name='prediction'))
    cm_gender = cm_gender.reindex(index=[0, 1], columns=[0, 1], fill_value=0)
    cm_age = pd.crosstab(pd.Series(y - 3 * (y // 3), name='truth'), pd.Series(y_
↪pred - 3 * (y_pred // 3), name='prediction'))
    cm_age = cm_age.reindex(index=[0, 1, 2], columns=[0, 1, 2], fill_value=0)

    # normalize rows of confusion matrices (we have unbalanced classes)
    cm = cm.astype(float)
    cm_gender = cm_gender.astype(float)
    cm_age = cm_age.astype(float)
    for i in range(0, 6):
        cm.loc[i, :] = cm.loc[i, :] / cm.loc[i, :].sum()
    for i in range(0, 2):
        cm_gender.loc[i, :] = cm_gender.loc[i, :] / cm_gender.loc[i, :].sum()
    for i in range(0, 3):
        cm_age.loc[i, :] = cm_age.loc[i, :] / cm_age.loc[i, :].sum()

    # plot confusion matrices
    fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12,3.5), tight_layout=True)
    sns.heatmap(cm, annot=True, fmt='.2f', cmap='hot', ax=ax1,
                xticklabels=posts['label'].cat.categories, yticklabels=posts[
↪'label'].cat.categories)
    sns.heatmap(cm_gender, annot=cm_gender, fmt='.2f', cmap='hot', ax=ax2,
                xticklabels=['f', 'm'], yticklabels=['f', 'm'])
    sns.heatmap(cm_age, annot=cm_age, fmt='.2f', cmap='hot', ax=ax3,
                xticklabels=['1', '2', '3'], yticklabels=['1', '2', '3'])
    plt.show()
```

Prediction accuracy on training data:

```
show_accuracy(y_train, y_train_pred)
```

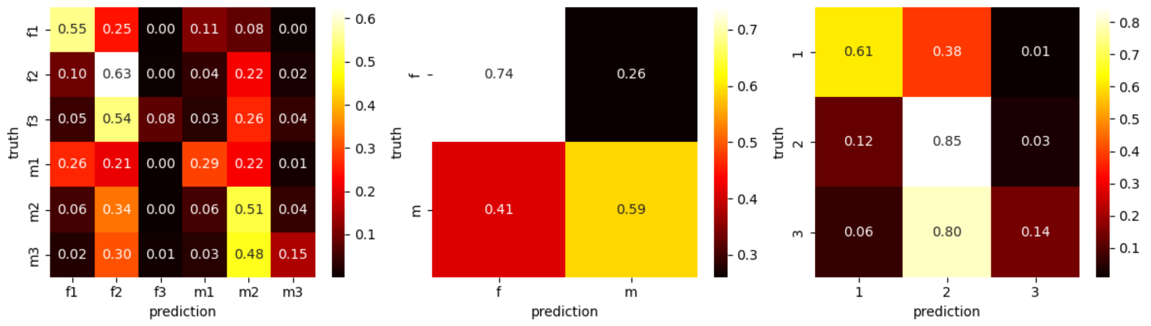
```
correct classification rate:          0.4577 (random guessing: 0.1667)
correct classification rate (gender): 0.6777 (random guessing: 0.5000)
correct classification rate (age):    0.6457 (random guessing: 0.3333)
```



Prediction accuracy on test data:

```
show_accuracy(y_test, y_test_pred)
```

```
correct classification rate:          0.4338 (random guessing: 0.1667)
correct classification rate (gender): 0.6634 (random guessing: 0.5000)
correct classification rate (age):    0.6351 (random guessing: 0.3333)
```



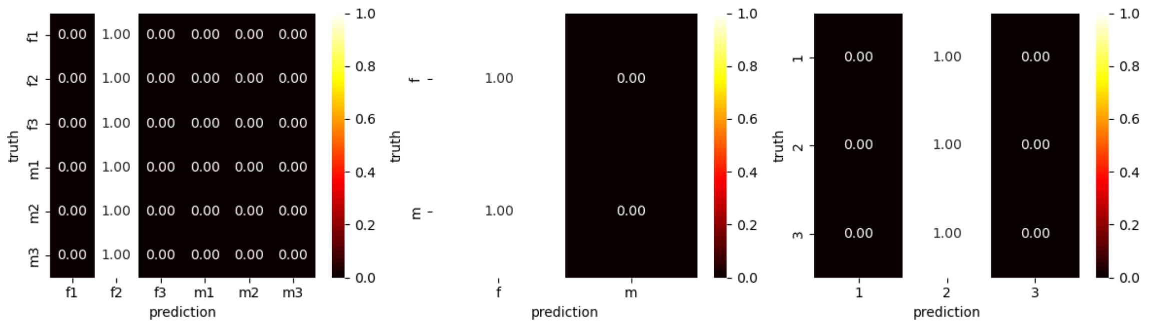
Prediction accuracy of the naive Bayes model is much better than random guessing. But we have to take into account class imbalance. Next to random guessing we may consider another trivial model for comparison of prediction accuracy: What happens if a model always predicts the largest class?

```
samples = np.histogram(y_train, bins=6, range=(-0.5, 5.5))[0]
print(samples)
```

```
[ 88173 123839  45420  92169 122165  50421]
```

```
show_accuracy(y_test, np.argmax(samples) * np.ones(y_test.shape))
```

```
correct classification rate:          0.2402 (random guessing: 0.1667)
correct classification rate (gender): 0.4958 (random guessing: 0.5000)
correct classification rate (age):    0.4727 (random guessing: 0.3333)
```



Always predicting the largest class works better than random guessing, but not as good as naive Bayes.

We should have a closer look at our model to get a better understanding of it's decision rules and also of the data set. A multinomial Bayes model is completely determined by the probabilities $p_{k,i}$. The values $p_{k,i}$ express the probability to observe feature k (the kth word in our vocabulary) in class i. The `MultinomialNB` objects provides the logarithm of these probabilities as a 2d NumPy array `MultinomialNB.feature_log_prob_`. First index is the class, second index the feature.

```
p = np.exp(mnb.feature_log_prob_)

p
```

```
array([[2.82486314e-06, 1.06657203e-05, 1.50133781e-05, ...,
        1.94223374e-06, 1.74543529e-06, 2.20958134e-06],
       [2.42658465e-06, 6.50200990e-06, 1.61185337e-05, ...,
```

(continues on next page)

(continued from previous page)

```
         5.40897601e-06, 1.52174765e-06, 1.86291326e-06],
        [3.16922035e-06, 8.23294289e-06, 1.46964824e-05, ...,
         5.55744801e-06, 3.46527245e-06, 2.94293686e-06],
        [4.33136042e-06, 1.01679660e-05, 2.14931631e-05, ...,
         4.23157086e-06, 1.69074347e-06, 2.65137775e-06],
        [2.44322058e-06, 7.15660286e-06, 1.61798918e-05, ...,
         1.32512575e-05, 1.94623715e-06, 2.01389749e-06],
        [3.11215272e-06, 7.82176886e-06, 1.72747277e-05, ...,
         7.67104041e-06, 5.87706081e-06, 3.72104924e-06]])
```

We may ask: Which words are most likely contained in a post written by a woman in age group 3?

```
i = 2    # class

# make dict for mapping indices to words
i2w = dict(zip(tfidf_vect.vocabulary_.values(), tfidf_vect.vocabulary_.keys()))

# get 1d array of indices sorted descending by probability
s = p[i, :].argsort()[::-1]

# print most likely words
for idx in s[0:20]:
    print(i2w[idx], p[i, idx])
```

```
be 0.01807095671226715
have 0.007949892249550804
do 0.006283225758642027
get 0.004215487511561934
not 0.003949829251723054
go 0.0038146191475615997
so 0.0032094272219337326
just 0.003171984731898014
know 0.0027767618498825836
think 0.002707606061709398
say 0.002463912137631007
make 0.0024453729582870366
time 0.00244219262808092
work 0.0023443172188228716
now 0.0022574399274471963
see 0.002239260855123209
day 0.002179888002166481
more 0.0021602842162711912
take 0.002147657978906127
want 0.002085134066190077
```

Obviously, we have to formulate our question more precisely: Which words are more likely contained in a post written by a woman in age group 3 than in any other class?

```
i = 2    # class

# get mask for selecting relevant words
likeliest_classes = p.argsort(axis=0)[-1, :]
mask = likeliest_classes == i

# get 1d array of indices sorted descending by probability
s = p[i, :].argsort()[::-1]

# print most likely words
for idx in s[mask[s]][0:20]:
    print(i2w[idx], p[i, idx])
```

```
book 0.0010989358770037991
old 0.0010590729846730097
woman 0.0010209207510928407
kid 0.0010198160240573167
child 0.0009682983100168208
comment 0.0009273079531121888
family 0.0008822763541782803
photo 0.0007498600268252408
rick 0.0007213990173024888
send 0.000693886210236962
husband 0.0006279847725706796
mother 0.0005971536115977698
dog 0.000572935663428436
favorite 0.0005620350678289687
light 0.0005582600807629513
water 0.0005342255476616197
visit 0.0005333974878505576
son 0.0005249505300652883
begin 0.0005129303011784495
body 0.0005077890351387825
```

This contains some more information about important words in a class. But ultimately we want to know which words dichotomize between classes. So we have to go one step further. Instead of looking for the most likely class given a word, we have to find words for which probabilities of the two most likely classes are as far away from each other as possible. We may calculate the ratio of both probabilities. If it is close to 1, then both probabilities are almost identical. If it is much greater than one, then the one probability is much higher than the other.

```
best_two_classes = p.argsort(axis=0)[:-3:-1, :]

best_probs_0 = p[best_two_classes[0, :], np.arange(p.shape[1])]
best_probs_1 = p[best_two_classes[1, :], np.arange(p.shape[1])]
prob_ratios =  best_probs_0 / best_probs_1

s = prob_ratios.argsort()[::-1]

for idx in s[0:20]:
    print(i2w[idx], prob_ratios[idx], best_two_classes[0, idx], best_two_
↪classes[1, idx])
```

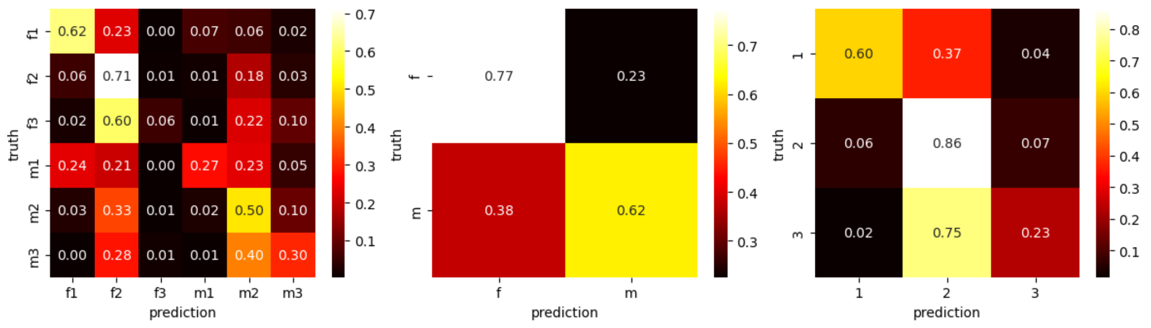
```
guam 57.690526155910824 5 2
corsair 47.53816228311916 5 2
uygur 34.368973906082616 5 2
tyke 31.360347041069716 3 2
battalion 28.488640702798268 2 5
improvised 19.254284653553338 2 3
regiment 18.911414974965442 2 3
xinjiang 18.30088754992347 5 2
expeditionary 18.082096818654847 2 5
infantry 15.621887983747696 2 5
pfc 15.156355170252311 2 5
spanner 15.112521737020673 2 1
dayton 14.519504223779798 5 4
40th 14.329376817750704 2 5
venerable 14.172037806976384 2 5
sherry 13.689534679813065 2 0
feast 13.499351300023987 2 1
cavalry 13.273764590605861 2 5
oratory 11.670985186018493 2 3
recessional 11.48808974663984 2 5
```

This list contains several words related to military, war, politics. Presumable there is a blog in our data set discussing such topics in many posts. Even if there is only one blog containing some word, which has many posts, the word is not removed from the vocabulary because it is contained in many documents. All those posts then have identical labels. Consequently, the word in question is tightly connected with one class. To avoid such one-blog influences we could classify whole blogs, that is, consider all posts of a blog as one document.

Correct classification rates are not as high as they should be. So let's further investigate prediction quality. It seems obvious that longer posts should be easier to classify than short posts, because long post contain more words and, thus, more information about the author.

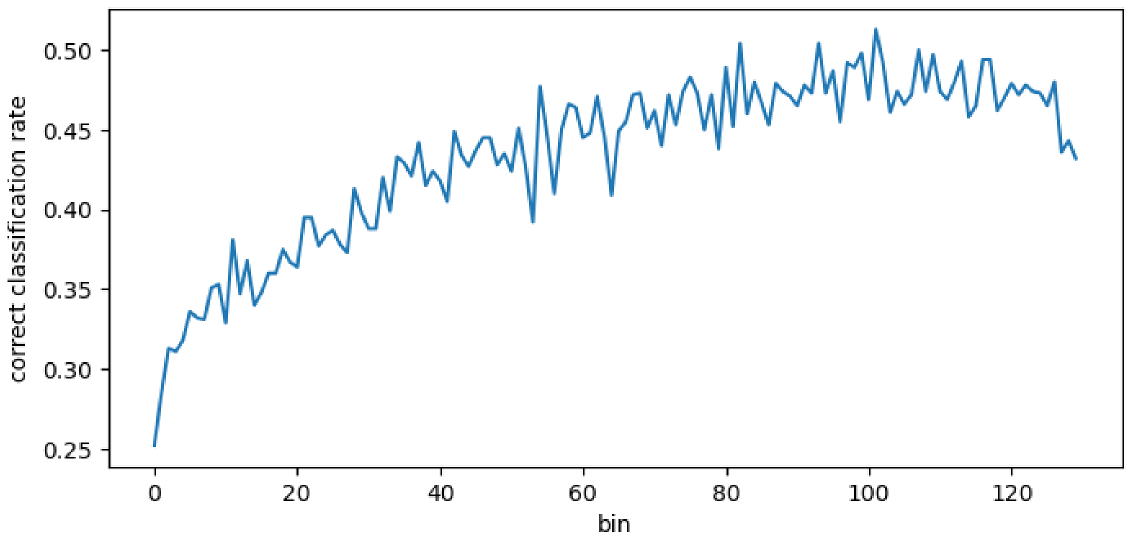
```
lengths = np.array([len(s) for s in S_test])
mask = lengths > 1000

print('considering {} posts'.format(mask.sum()))

show_accuracy(y_test[mask], y_test_pred[mask])
```

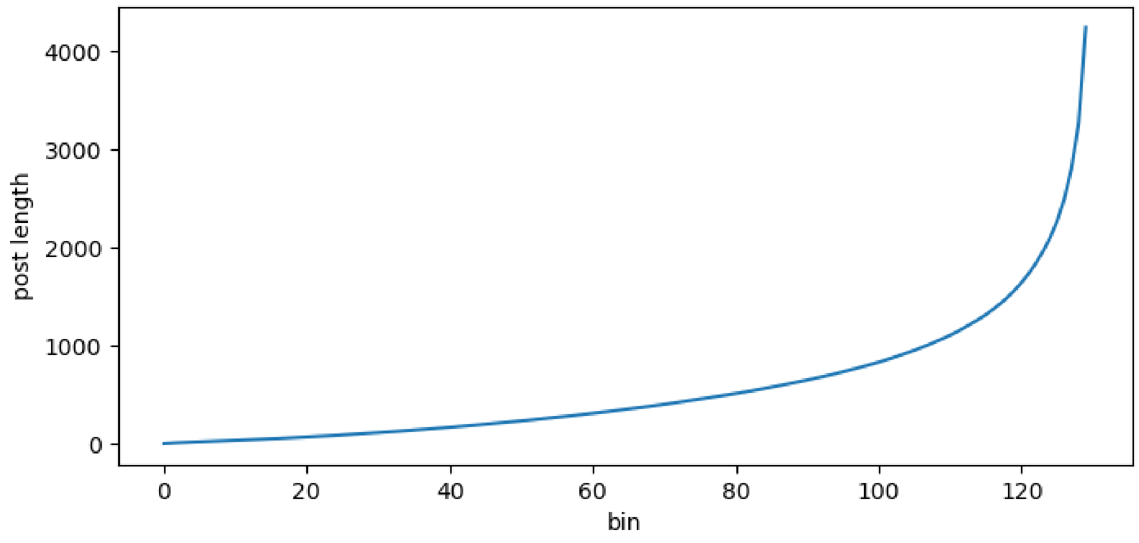
```
considering 23419 posts
correct classification rate:          0.4712 (random guessing: 0.1667)
correct classification rate (gender): 0.7006 (random guessing: 0.5000)
correct classification rate (age):    0.6542 (random guessing: 0.3333)
```



Considering only posts with at least 1000 characters indeed yields slightly better correct classification rates. Here comes some more detail: sort all posts by length, bin sorted posts into bins of equal size, calculate correct classification rate for each bin.

```
bin_size = 1000

lengths = np.array([len(s) for s in S_test])
s = lengths.argsort()

acc = []
mean_length = []
for k in range(bin_size, s.size, bin_size):
    acc.append(metrics.accuracy_score(y_test[s[(k-bin_size):k]], y_test_pred[s[(k-
↪bin_size):k]]))
    mean_length.append(lengths[s[(k-bin_size):k]].mean())

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8,8))
ax1.plot(acc)
ax2.plot(mean_length)
ax1.set_xlabel('bin')
ax1.set_ylabel('correct classification rate')
ax2.set_xlabel('bin')
ax2.set_ylabel('post length')
plt.show()
```





31.2.4 Support Vector Machine

Scikit-Learn provides different implementations of support vector machines for classification:

- `SVC`[565] is the most general one. It supports linear and kernel SVMs, but is relatively slow.
- `LinearSVC`[566] is a more efficient implementation supporting only linear SVMs.
- `SGDClassifier`[567] uses gradient descent to minimize some loss function, the hinge loss for instance. It supports penalty terms for regularization and, thus, SVMs.

[565] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
[566] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[567] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

Linear SVM

We start with `SGDClassifier`. For multi-class problems the one-versus-all approach is used. The 6 linear models can be trained in parallel.

```
import sklearn.linear_model as linear_model
```

```
sgdsvm = linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=1,
                                    n_jobs=1, tol=1e-3, max_iter=10)

param_grid = {'alpha': [0.1 ** k for k in range(0, 10)]}
gs = model_selection.GridSearchCV(sgdsvm, param_grid, scoring='accuracy', cv=5, n_
 ↪jobs=-1)
gs.fit(X_train, y_train)

print(gs.best_params_)
```

```
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
 ↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
 ↪reached before convergence. Consider increasing max_iter to improve the fit.
```

(continues on next page)

(continued from previous page)

```
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
```

(continues on next page)

(continued from previous page)

```
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
```

```
{'alpha': 1.0000000000000004e-06}
```

```
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↪stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration␣
↪reached before convergence. Consider increasing max_iter to improve the fit.
  warnings.warn(
```

```
sgdsvm = linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=1e-6,
                                    verbose=1, n_jobs=-1, tol=1e-3, max_iter=1000)
sgdsvm.fit(X_train, y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 20 concurrent␣
↪workers.
```

```
-- Epoch 1
-- Epoch 1
-- Epoch 1
-- Epoch 1
-- Epoch 1
-- Epoch 1
```

(continues on next page)

(continued from previous page)

```
Norm: 219.73, NNZs: 33064, Bias: -2.337957, T: 522187, Avg. loss: 0.779561
Total training time: 0.24 seconds.
-- Epoch 2
Norm: 268.65, NNZs: 37108, Bias: -1.501407, T: 522187, Avg. loss: 1.184225
Total training time: 0.26 seconds.
-- Epoch 2
Norm: 199.09, NNZs: 35332, Bias: -2.960665, T: 522187, Avg. loss: 0.466104
Total training time: 0.31 seconds.
-- Epoch 2
Norm: 209.23, NNZs: 35929, Bias: -2.149786, T: 522187, Avg. loss: 0.534687
Total training time: 0.33 seconds.
-- Epoch 2
Norm: 237.08, NNZs: 35350, Bias: -2.486370, T: 522187, Avg. loss: 0.908144
Total training time: 0.32 seconds.
-- Epoch 2
Norm: 262.72, NNZs: 36823, Bias: -1.829896, T: 522187, Avg. loss: 1.211338
Total training time: 0.31 seconds.
-- Epoch 2
Norm: 168.64, NNZs: 34434, Bias: -1.778373, T: 1044374, Avg. loss: 0.419178
Total training time: 0.48 seconds.
-- Epoch 3
Norm: 210.25, NNZs: 37329, Bias: -1.387185, T: 1044374, Avg. loss: 0.635662
Total training time: 0.54 seconds.
-- Epoch 3
Norm: 181.81, NNZs: 36309, Bias: -1.833561, T: 1044374, Avg. loss: 0.482218
Total training time: 0.59 seconds.
-- Epoch 3
Norm: 150.58, NNZs: 36359, Bias: -2.077375, T: 1044374, Avg. loss: 0.241740
Total training time: 0.60 seconds.
-- Epoch 3
Norm: 202.72, NNZs: 37215, Bias: -1.345307, T: 1044374, Avg. loss: 0.650129
Total training time: 0.61 seconds.
-- Epoch 3
Norm: 157.23, NNZs: 36641, Bias: -1.687595, T: 1044374, Avg. loss: 0.278794
Total training time: 0.65 seconds.
-- Epoch 3
Norm: 150.96, NNZs: 34985, Bias: -1.671367, T: 1566561, Avg. loss: 0.371611
Total training time: 0.77 seconds.
-- Epoch 4
Norm: 189.24, NNZs: 37382, Bias: -1.152830, T: 1566561, Avg. loss: 0.560032
Total training time: 0.86 seconds.
-- Epoch 4
Norm: 160.93, NNZs: 36699, Bias: -1.524977, T: 1566561, Avg. loss: 0.425225
Total training time: 0.89 seconds.
-- Epoch 4
Norm: 131.42, NNZs: 36678, Bias: -1.912441, T: 1566561, Avg. loss: 0.210238
Total training time: 0.92 seconds.
-- Epoch 4
Norm: 137.20, NNZs: 36898, Bias: -1.463867, T: 1566561, Avg. loss: 0.241759
Total training time: 0.96 seconds.
-- Epoch 4
Norm: 180.91, NNZs: 37305, Bias: -1.462382, T: 1566561, Avg. loss: 0.574577
Total training time: 0.96 seconds.
-- Epoch 4
Norm: 142.23, NNZs: 35270, Bias: -1.565278, T: 2088748, Avg. loss: 0.351199
Total training time: 1.04 seconds.
-- Epoch 5
Norm: 178.76, NNZs: 37395, Bias: -1.191582, T: 2088748, Avg. loss: 0.528111
Total training time: 1.16 seconds.
-- Epoch 5
Norm: 149.78, NNZs: 36852, Bias: -1.514599, T: 2088748, Avg. loss: 0.400745
```

(continues on next page)

(continued from previous page)

```
Total training time: 1.16 seconds.
-- Epoch 5
Norm: 125.77, NNZs: 37022, Bias: -1.384963, T: 2088748, Avg. loss: 0.225969
Total training time: 1.23 seconds.
-- Epoch 5
Norm: 120.35, NNZs: 36855, Bias: -1.777925, T: 2088748, Avg. loss: 0.196958
Total training time: 1.23 seconds.
-- Epoch 5
Norm: 136.46, NNZs: 35440, Bias: -1.536337, T: 2610935, Avg. loss: 0.340258
Total training time: 1.31 seconds.
-- Epoch 6
Norm: 169.97, NNZs: 37337, Bias: -1.335996, T: 2088748, Avg. loss: 0.541578
Total training time: 1.28 seconds.
-- Epoch 5
Norm: 142.67, NNZs: 36940, Bias: -1.575590, T: 2610935, Avg. loss: 0.386590
Total training time: 1.46 seconds.
-- Epoch 6
Norm: 171.57, NNZs: 37399, Bias: -1.137722, T: 2610935, Avg. loss: 0.509895
Total training time: 1.49 seconds.
-- Epoch 6
Norm: 118.16, NNZs: 37077, Bias: -1.374983, T: 2610935, Avg. loss: 0.217068
Total training time: 1.52 seconds.
-- Epoch 6
Norm: 113.18, NNZs: 36939, Bias: -1.660853, T: 2610935, Avg. loss: 0.189184
Total training time: 1.55 seconds.
-- Epoch 6
Norm: 133.26, NNZs: 35543, Bias: -1.505106, T: 3133122, Avg. loss: 0.333129
Total training time: 1.59 seconds.
-- Epoch 7
Norm: 163.21, NNZs: 37352, Bias: -1.249988, T: 2610935, Avg. loss: 0.523599
Total training time: 1.64 seconds.
-- Epoch 6
Norm: 112.62, NNZs: 37122, Bias: -1.349333, T: 3133122, Avg. loss: 0.211330
Total training time: 1.81 seconds.
-- Epoch 7
Norm: 167.07, NNZs: 37400, Bias: -1.152601, T: 3133122, Avg. loss: 0.497132
Total training time: 1.85 seconds.
-- Epoch 7
Norm: 137.27, NNZs: 37003, Bias: -1.440040, T: 3133122, Avg. loss: 0.378094
Total training time: 1.83 seconds.
-- Epoch 7
Norm: 130.37, NNZs: 35606, Bias: -1.430404, T: 3655309, Avg. loss: 0.328107
Total training time: 1.87 seconds.
-- Epoch 8
Norm: 107.79, NNZs: 37000, Bias: -1.613265, T: 3133122, Avg. loss: 0.184543
Total training time: 1.87 seconds.
-- Epoch 7
Norm: 158.85, NNZs: 37359, Bias: -1.239013, T: 3133122, Avg. loss: 0.511492
Total training time: 2.00 seconds.
-- Epoch 7
Norm: 108.23, NNZs: 37153, Bias: -1.293614, T: 3655309, Avg. loss: 0.207319
Total training time: 2.10 seconds.
-- Epoch 8
Norm: 128.56, NNZs: 35664, Bias: -1.465531, T: 4177496, Avg. loss: 0.324141
Total training time: 2.16 seconds.
-- Epoch 9
Norm: 103.84, NNZs: 37030, Bias: -1.560485, T: 3655309, Avg. loss: 0.181205
Total training time: 2.20 seconds.
-- Epoch 8
Norm: 163.42, NNZs: 37402, Bias: -1.144784, T: 3655309, Avg. loss: 0.489235
Total training time: 2.22 seconds.
```

(continues on next page)

(continued from previous page)

```
-- Epoch 8
Norm: 133.91, NNZs: 37031, Bias: -1.430053, T: 3655309, Avg. loss: 0.371696
Total training time: 2.20 seconds.
-- Epoch 8
Norm: 155.12, NNZs: 37363, Bias: -1.212708, T: 3655309, Avg. loss: 0.503037
Total training time: 2.37 seconds.
-- Epoch 8
Norm: 104.81, NNZs: 37172, Bias: -1.243639, T: 4177496, Avg. loss: 0.204386
Total training time: 2.40 seconds.
-- Epoch 9
Norm: 127.11, NNZs: 35696, Bias: -1.463909, T: 4699683, Avg. loss: 0.321378
Total training time: 2.44 seconds.
-- Epoch 10
Norm: 100.69, NNZs: 37050, Bias: -1.526634, T: 4177496, Avg. loss: 0.178778
Total training time: 2.52 seconds.
-- Epoch 9
Norm: 160.40, NNZs: 37405, Bias: -1.104108, T: 4177496, Avg. loss: 0.482781
Total training time: 2.58 seconds.
-- Epoch 9
Norm: 130.57, NNZs: 37059, Bias: -1.383276, T: 4177496, Avg. loss: 0.367050
Total training time: 2.57 seconds.
-- Epoch 9
Norm: 101.85, NNZs: 37188, Bias: -1.259478, T: 4699683, Avg. loss: 0.202182
Total training time: 2.69 seconds.
-- Epoch 10
Norm: 126.07, NNZs: 35738, Bias: -1.464982, T: 5221870, Avg. loss: 0.319141
Total training time: 2.73 seconds.
-- Epoch 11
Norm: 152.42, NNZs: 37364, Bias: -1.225642, T: 4177496, Avg. loss: 0.496826
Total training time: 2.73 seconds.
-- Epoch 9
Norm: 98.12, NNZs: 37073, Bias: -1.470048, T: 4699683, Avg. loss: 0.176941
Total training time: 2.85 seconds.
-- Epoch 10
Norm: 158.30, NNZs: 37405, Bias: -1.051860, T: 4699683, Avg. loss: 0.478211
Total training time: 2.95 seconds.
-- Epoch 10
Norm: 128.14, NNZs: 37075, Bias: -1.369116, T: 4699683, Avg. loss: 0.363634
Total training time: 2.94 seconds.
-- Epoch 10
Norm: 99.42, NNZs: 37195, Bias: -1.205442, T: 5221870, Avg. loss: 0.200388
Total training time: 2.98 seconds.
-- Epoch 11
Norm: 125.14, NNZs: 35757, Bias: -1.439000, T: 5744057, Avg. loss: 0.317268
Total training time: 3.01 seconds.
-- Epoch 12
Norm: 149.96, NNZs: 37370, Bias: -1.267810, T: 4699683, Avg. loss: 0.491971
Total training time: 3.10 seconds.
-- Epoch 10
Norm: 96.06, NNZs: 37095, Bias: -1.451079, T: 5221870, Avg. loss: 0.175365
Total training time: 3.18 seconds.
-- Epoch 11
Norm: 97.22, NNZs: 37205, Bias: -1.209968, T: 5744057, Avg. loss: 0.199035
Total training time: 3.27 seconds.
-- Epoch 12
Norm: 124.42, NNZs: 35780, Bias: -1.437333, T: 6266244, Avg. loss: 0.315803
Total training time: 3.29 seconds.
-- Epoch 13
Norm: 156.49, NNZs: 37405, Bias: -1.104196, T: 5221870, Avg. loss: 0.474072
Total training time: 3.31 seconds.
-- Epoch 11
```

(continues on next page)

(continued from previous page)

```
Norm: 125.91, NNZs: 37095, Bias: -1.350864, T: 5221870, Avg. loss: 0.360769
Total training time: 3.32 seconds.
-- Epoch 11
Norm: 147.85, NNZs: 37374, Bias: -1.248811, T: 5221870, Avg. loss: 0.488246
Total training time: 3.46 seconds.
-- Epoch 11
Norm: 94.17, NNZs: 37108, Bias: -1.408897, T: 5744057, Avg. loss: 0.174297
Total training time: 3.51 seconds.
-- Epoch 12
Norm: 95.36, NNZs: 37218, Bias: -1.171307, T: 6266244, Avg. loss: 0.197882
Total training time: 3.56 seconds.
-- Epoch 13
Norm: 123.68, NNZs: 35804, Bias: -1.484681, T: 6788431, Avg. loss: 0.314356
Total training time: 3.57 seconds.
-- Epoch 14
Norm: 154.48, NNZs: 37405, Bias: -1.084382, T: 5744057, Avg. loss: 0.471331
Total training time: 3.67 seconds.
-- Epoch 12
Norm: 124.08, NNZs: 37103, Bias: -1.325807, T: 5744057, Avg. loss: 0.358538
Total training time: 3.69 seconds.
-- Epoch 12
Norm: 93.75, NNZs: 37225, Bias: -1.164110, T: 6788431, Avg. loss: 0.196876
Total training time: 3.84 seconds.
-- Epoch 14
Norm: 123.22, NNZs: 35817, Bias: -1.440790, T: 7310618, Avg. loss: 0.313391
Total training time: 3.85 seconds.
-- Epoch 15
Norm: 92.55, NNZs: 37119, Bias: -1.389629, T: 6266244, Avg. loss: 0.173379
Total training time: 3.84 seconds.
-- Epoch 13
Norm: 146.14, NNZs: 37375, Bias: -1.153201, T: 5744057, Avg. loss: 0.485277
Total training time: 3.83 seconds.
-- Epoch 12
Norm: 153.22, NNZs: 37405, Bias: -1.094359, T: 6266244, Avg. loss: 0.468621
Total training time: 4.03 seconds.
-- Epoch 13
Norm: 122.38, NNZs: 37112, Bias: -1.275066, T: 6266244, Avg. loss: 0.356526
Total training time: 4.06 seconds.
-- Epoch 13
Norm: 122.59, NNZs: 35829, Bias: -1.438006, T: 7832805, Avg. loss: 0.312608
Total training time: 4.13 seconds.
-- Epoch 16
Norm: 92.30, NNZs: 37230, Bias: -1.161619, T: 7310618, Avg. loss: 0.196101
Total training time: 4.13 seconds.
-- Epoch 15
Norm: 91.10, NNZs: 37126, Bias: -1.376473, T: 6788431, Avg. loss: 0.172595
Total training time: 4.16 seconds.
-- Epoch 14
Norm: 144.55, NNZs: 37376, Bias: -1.218568, T: 6266244, Avg. loss: 0.482629
Total training time: 4.19 seconds.
-- Epoch 13
Norm: 151.83, NNZs: 37405, Bias: -1.086403, T: 6788431, Avg. loss: 0.466587
Total training time: 4.39 seconds.
-- Epoch 14
Norm: 122.16, NNZs: 35835, Bias: -1.404379, T: 8354992, Avg. loss: 0.311554
Total training time: 4.41 seconds.
-- Epoch 17
Norm: 91.03, NNZs: 37232, Bias: -1.152392, T: 7832805, Avg. loss: 0.195366
Total training time: 4.42 seconds.
-- Epoch 16
Norm: 121.01, NNZs: 37120, Bias: -1.323271, T: 6788431, Avg. loss: 0.354857
```

(continues on next page)

(continued from previous page)

```
Total training time: 4.44 seconds.
-- Epoch 14
Norm: 89.78, NNZs: 37132, Bias: -1.393862, T: 7310618, Avg. loss: 0.171843
Total training time: 4.51 seconds.
-- Epoch 15
Norm: 143.38, NNZs: 37376, Bias: -1.217295, T: 6788431, Avg. loss: 0.480415
Total training time: 4.57 seconds.
-- Epoch 14
Norm: 121.82, NNZs: 35842, Bias: -1.377545, T: 8877179, Avg. loss: 0.310999
Total training time: 4.68 seconds.
-- Epoch 18
Norm: 89.83, NNZs: 37236, Bias: -1.175119, T: 8354992, Avg. loss: 0.194731
Total training time: 4.70 seconds.
-- Epoch 17
Norm: 150.75, NNZs: 37405, Bias: -1.079330, T: 7310618, Avg. loss: 0.464582
Total training time: 4.75 seconds.
-- Epoch 15
Norm: 119.71, NNZs: 37126, Bias: -1.244118, T: 7310618, Avg. loss: 0.353777
Total training time: 4.82 seconds.
-- Epoch 15
Norm: 88.64, NNZs: 37134, Bias: -1.367492, T: 7832805, Avg. loss: 0.171274
Total training time: 4.83 seconds.
-- Epoch 16
Norm: 142.04, NNZs: 37377, Bias: -1.192788, T: 7310618, Avg. loss: 0.478759
Total training time: 4.87 seconds.
-- Epoch 15
Norm: 121.53, NNZs: 35850, Bias: -1.410848, T: 9399366, Avg. loss: 0.310291
Total training time: 4.95 seconds.
-- Epoch 19
Norm: 88.73, NNZs: 37240, Bias: -1.146500, T: 8877179, Avg. loss: 0.194189
Total training time: 4.98 seconds.
-- Epoch 18
Norm: 149.66, NNZs: 37405, Bias: -1.025816, T: 7832805, Avg. loss: 0.463129
Total training time: 5.09 seconds.
-- Epoch 16
Norm: 87.59, NNZs: 37137, Bias: -1.338993, T: 8354992, Avg. loss: 0.170799
Total training time: 5.16 seconds.
Convergence after 16 epochs took 5.16 seconds
Norm: 140.98, NNZs: 37378, Bias: -1.191497, T: 7832805, Avg. loss: 0.477089
Total training time: 5.17 seconds.
-- Epoch 16
Norm: 118.56, NNZs: 37133, Bias: -1.275177, T: 7832805, Avg. loss: 0.352552
Total training time: 5.19 seconds.
-- Epoch 16
Norm: 121.13, NNZs: 35852, Bias: -1.386067, T: 9921553, Avg. loss: 0.309652
Total training time: 5.22 seconds.
-- Epoch 20
Norm: 87.80, NNZs: 37241, Bias: -1.116436, T: 9399366, Avg. loss: 0.193698
Total training time: 5.25 seconds.
Convergence after 18 epochs took 5.25 seconds
```

```
[Parallel(n_jobs=-1)]: Done   2 out of   6 | elapsed:    5.3s remaining:   10.6s
```

```
Norm: 148.73, NNZs: 37405, Bias: -1.046856, T: 8354992, Avg. loss: 0.461690
Total training time: 5.44 seconds.
-- Epoch 17
Norm: 121.05, NNZs: 35857, Bias: -1.384460, T: 10443740, Avg. loss: 0.309075
Total training time: 5.50 seconds.
-- Epoch 21
```

(continues on next page)

(continued from previous page)

```
Norm: 140.07, NNZs: 37379, Bias: -1.177948, T: 8354992, Avg. loss: 0.475676
Total training time: 5.54 seconds.
-- Epoch 17
Norm: 117.56, NNZs: 37136, Bias: -1.294708, T: 8354992, Avg. loss: 0.351373
Total training time: 5.56 seconds.
-- Epoch 17
Norm: 120.81, NNZs: 35869, Bias: -1.375922, T: 10965927, Avg. loss: 0.308701
Total training time: 5.77 seconds.
Convergence after 21 epochs took 5.77 seconds
Norm: 147.81, NNZs: 37405, Bias: -1.039440, T: 8877179, Avg. loss: 0.460558
Total training time: 5.79 seconds.
-- Epoch 18
Norm: 139.18, NNZs: 37380, Bias: -1.162537, T: 8877179, Avg. loss: 0.474609
Total training time: 5.89 seconds.
-- Epoch 18
Norm: 116.50, NNZs: 37140, Bias: -1.292449, T: 8877179, Avg. loss: 0.350666
Total training time: 5.91 seconds.
-- Epoch 18
Norm: 147.02, NNZs: 37405, Bias: -1.056075, T: 9399366, Avg. loss: 0.459475
Total training time: 6.16 seconds.
-- Epoch 19
Norm: 138.34, NNZs: 37380, Bias: -1.124137, T: 9399366, Avg. loss: 0.473367
Total training time: 6.25 seconds.
-- Epoch 19
Norm: 115.65, NNZs: 37142, Bias: -1.300770, T: 9399366, Avg. loss: 0.349788
Total training time: 6.27 seconds.
-- Epoch 19
Norm: 146.34, NNZs: 37405, Bias: -1.042664, T: 9921553, Avg. loss: 0.458375
Total training time: 6.53 seconds.
-- Epoch 20
Norm: 137.66, NNZs: 37383, Bias: -1.129650, T: 9921553, Avg. loss: 0.472506
Total training time: 6.60 seconds.
-- Epoch 20
Norm: 114.86, NNZs: 37143, Bias: -1.250730, T: 9921553, Avg. loss: 0.349037
Total training time: 6.61 seconds.
-- Epoch 20
Norm: 145.62, NNZs: 37405, Bias: -1.032389, T: 10443740, Avg. loss: 0.457693
Total training time: 6.88 seconds.
-- Epoch 21
Norm: 137.01, NNZs: 37383, Bias: -1.119291, T: 10443740, Avg. loss: 0.471555
Total training time: 6.89 seconds.
-- Epoch 21
Norm: 113.98, NNZs: 37146, Bias: -1.237537, T: 10443740, Avg. loss: 0.348467
Total training time: 6.95 seconds.
-- Epoch 21
Norm: 136.37, NNZs: 37383, Bias: -1.136997, T: 10965927, Avg. loss: 0.470758
Total training time: 7.18 seconds.
-- Epoch 22
Norm: 145.04, NNZs: 37406, Bias: -1.056571, T: 10965927, Avg. loss: 0.456832
Total training time: 7.21 seconds.
-- Epoch 22
Norm: 113.30, NNZs: 37149, Bias: -1.265502, T: 10965927, Avg. loss: 0.347908
Total training time: 7.30 seconds.
Convergence after 21 epochs took 7.30 seconds
Norm: 135.71, NNZs: 37383, Bias: -1.137318, T: 11488114, Avg. loss: 0.470119
Total training time: 7.46 seconds.
-- Epoch 23
Norm: 144.41, NNZs: 37406, Bias: -1.044690, T: 11488114, Avg. loss: 0.456202
Total training time: 7.51 seconds.
-- Epoch 23
Norm: 135.18, NNZs: 37383, Bias: -1.126682, T: 12010301, Avg. loss: 0.469384
```

(continues on next page)

(continued from previous page)

```
Total training time: 7.74 seconds.
Convergence after 23 epochs took 7.74 seconds
Norm: 143.94, NNZs: 37406, Bias: -1.031312, T: 12010301, Avg. loss: 0.455551
Total training time: 7.79 seconds.
-- Epoch 24
Norm: 143.41, NNZs: 37406, Bias: -1.047508, T: 12532488, Avg. loss: 0.454907
Total training time: 8.06 seconds.
Convergence after 24 epochs took 8.06 seconds
```

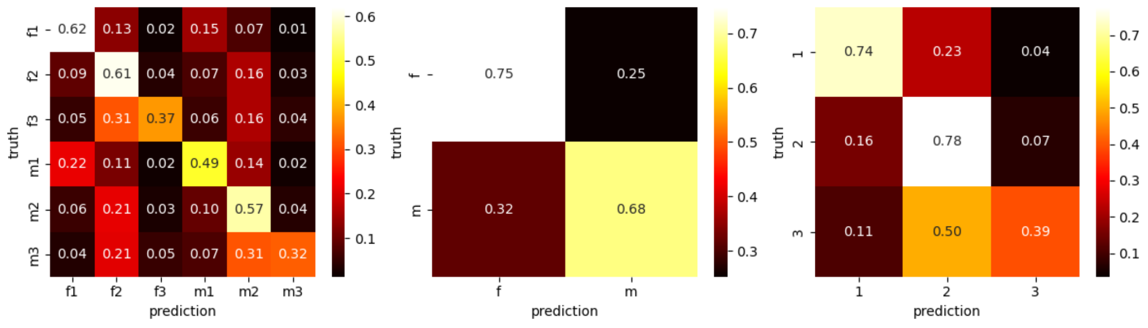
```
[Parallel(n_jobs=-1)]: Done   6 out of   6 | elapsed:    8.1s finished
```

```
SGDClassifier(alpha=1e-06, n_jobs=-1, verbose=1)
```

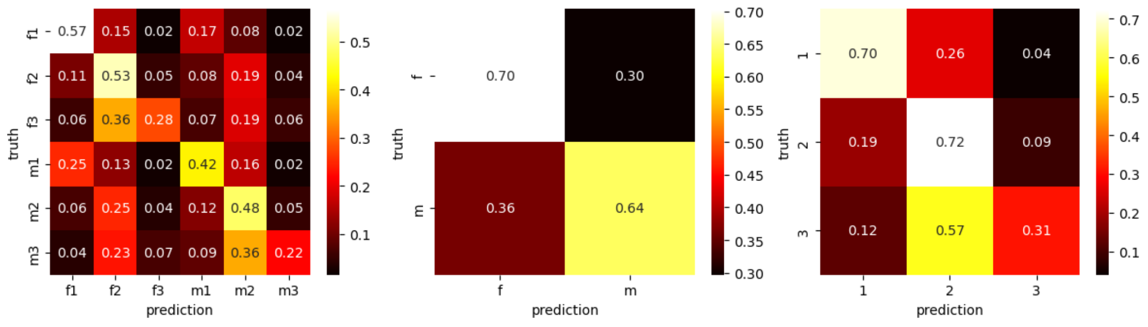
```
y_train_pred = sgdsvm.predict(X_train)
y_test_pred = sgdsvm.predict(X_test)
```

```
show_accuracy(y_train, y_train_pred)
show_accuracy(y_test, y_test_pred)
```

```
correct classification rate:          0.5322 (random guessing: 0.1667)
correct classification rate (gender): 0.7157 (random guessing: 0.5000)
correct classification rate (age):    0.6922 (random guessing: 0.3333)
```



```
correct classification rate:          0.4530 (random guessing: 0.1667)
correct classification rate (gender): 0.6695 (random guessing: 0.5000)
correct classification rate (age):    0.6397 (random guessing: 0.3333)
```



Results are slightly better than with naive Bayes. `LinearSVC` should yield similar results because the same optimization problem is solved with a different algorithm.

```
import sklearn.svm as svm
```

```
linearsvm = svm.LinearSVC(C=1e6, tol=1e-3, max_iter=100, dual='auto')
linearsvm.fit(X_train, y_train)
```

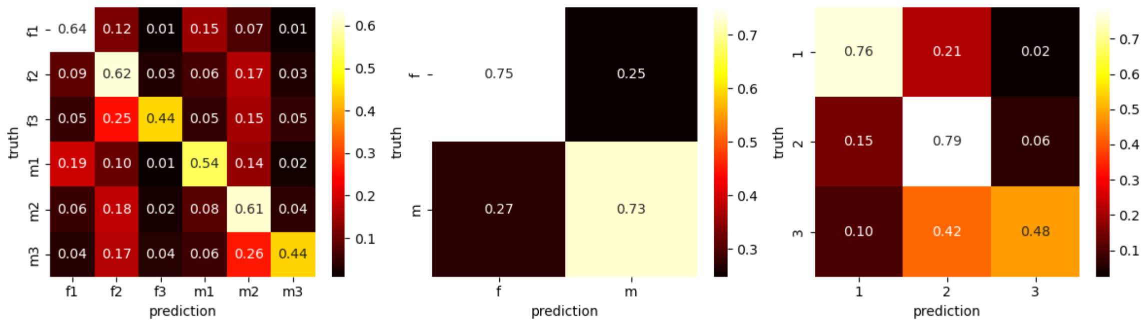
```
LinearSVC(C=1000000.0, dual='auto', max_iter=100, tol=0.001)
```

The minimization algorithm shows very slow convergence. Stopping before convergence usually yields low prediction quality.

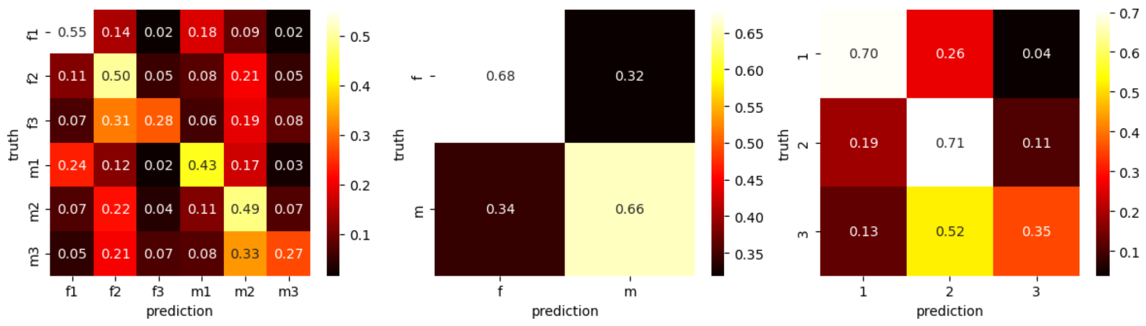
```
y_train_pred = linearsvm.predict(X_train)
y_test_pred = linearsvm.predict(X_test)
```

```
show_accuracy(y_train, y_train_pred)
show_accuracy(y_test, y_test_pred)
```

```
correct classification rate:          0.5743 (random guessing: 0.1667)
correct classification rate (gender): 0.7393 (random guessing: 0.5000)
correct classification rate (age):    0.7233 (random guessing: 0.3333)
```



```
correct classification rate:          0.4520 (random guessing: 0.1667)
correct classification rate (gender): 0.6696 (random guessing: 0.5000)
correct classification rate (age):    0.6392 (random guessing: 0.3333)
```



Kernel SVM

To improve prediction quality we could use a nonlinear kernel. This is supported by `SVC`, but requires much more computation time for training as well as for prediction.

Alternatively, we could first transform features and then train a linear classifier. But we have almost 40000 features. Using polynomial features of degree 2 would result in approximately $\frac{40000^2}{2}$, that is, almost 1 billion features.

```
kernelsvm = svm.SVC(kernel='rbf', C=1e6, tol=1e-5, max_iter=100)
kernelsvm.fit(X_train, y_train)
```

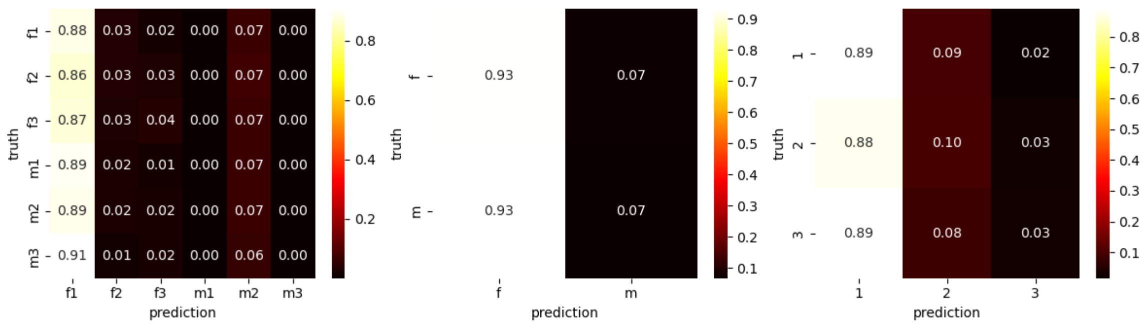
```
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/svm/_base.py:297:␣
↪ConvergenceWarning: Solver terminated early (max_iter=100).  Consider pre-
↪processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
```

```
SVC(C=1000000.0, max_iter=100, tol=1e-05)
```

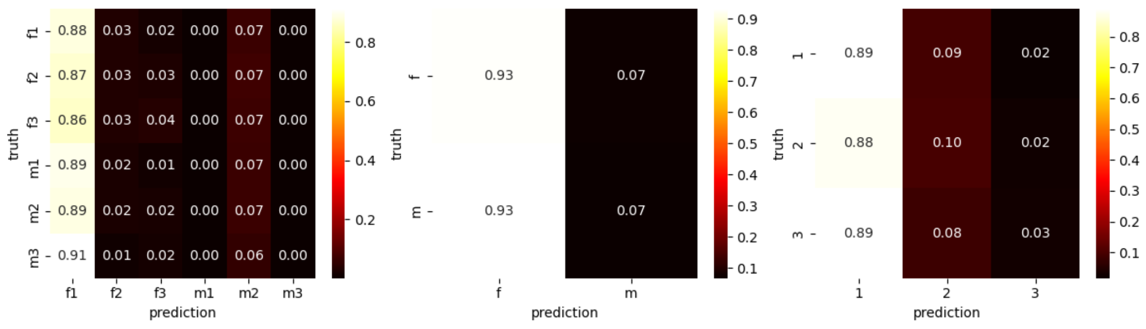
```
y_train_pred = kernelsvm.predict(X_train)
y_test_pred = kernelsvm.predict(X_test)
```

```
show_accuracy(y_train, y_train_pred)
show_accuracy(y_test, y_test_pred)
```

```
correct classification rate:          0.1762 (random guessing: 0.1667)
correct classification rate (gender): 0.4912 (random guessing: 0.5000)
correct classification rate (age):    0.3589 (random guessing: 0.3333)
```



```
correct classification rate:          0.1754 (random guessing: 0.1667)
correct classification rate (gender): 0.4933 (random guessing: 0.5000)
correct classification rate (age):    0.3569 (random guessing: 0.3333)
```



Again minimization did not converge in acceptable time and predictions are not satisfying.

Scikit-Learn provides a technique known as *kernel approximation*. The idea is to mimic the behavior of a kernel by an inner product in a relatively low dimensional space. Using kernel approximation we should get similar results like from a kernel SVM, but from a linear SVM on a low dimensional space. See Random Features for Large-Scale Kernel Machines[568], especially Section 1, for some more background.

```
import sklearn.kernel_approximation as kernel_approximation
```

Kernel approximation requires very much memory. Avoid parallelization (parameter `n_jobs`) to save memory. The values for α and γ below were obtained via hyperparameter optimization (took several days).

[568] https://people.eecs.berkeley.edu/~brecht/papers/07.rah.rec.nips.pdf

```
# cell execution takes several minutes and requires 70 GB of memory

rbf = kernel_approximation.RBFSampler(n_components=10000, gamma=0.13)
X_train_rbf = rbf.fit_transform(X_train)
X_test_rbf = rbf.transform(X_test)

rbfsvm = linear_model.SGDClassifier(loss='hinge', penalty='l2', alpha=0.00390625,
                                    verbose=1, n_jobs=-1, tol=1e-3, max_iter=100)
rbfsvm.fit(X_train_rbf, y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 20 concurrent␣
↪workers.
```

```
-- Epoch 1-- Epoch 1

-- Epoch 1
-- Epoch 1
-- Epoch 1
-- Epoch 1
Norm: 0.09, NNZs: 10000, Bias: -1.005309, T: 522187, Avg. loss: 0.174627
Total training time: 11.02 seconds.
-- Epoch 2
Norm: 0.07, NNZs: 10000, Bias: -1.004696, T: 522187, Avg. loss: 0.193802
Total training time: 11.12 seconds.
-- Epoch 2
Norm: 0.19, NNZs: 10000, Bias: -1.005802, T: 522187, Avg. loss: 0.338775
Total training time: 12.17 seconds.
-- Epoch 2
Norm: 0.12, NNZs: 10000, Bias: -1.002819, T: 522187, Avg. loss: 0.354316
Total training time: 12.33 seconds.
-- Epoch 2
Norm: 0.12, NNZs: 10000, Bias: -1.000431, T: 522187, Avg. loss: 0.469682
Total training time: 13.28 seconds.
-- Epoch 2
Norm: 0.12, NNZs: 10000, Bias: -1.002134, T: 522187, Avg. loss: 0.476062
Total training time: 13.32 seconds.
-- Epoch 2
Norm: 0.07, NNZs: 10000, Bias: -1.002831, T: 1044374, Avg. loss: 0.174006
Total training time: 23.26 seconds.
-- Epoch 3
Norm: 0.05, NNZs: 10000, Bias: -1.002826, T: 1044374, Avg. loss: 0.193173
Total training time: 23.29 seconds.
-- Epoch 3
Norm: 0.15, NNZs: 10000, Bias: -1.005076, T: 1044374, Avg. loss: 0.337748
Total training time: 26.07 seconds.
-- Epoch 3
Norm: 0.09, NNZs: 10000, Bias: -1.001710, T: 1044374, Avg. loss: 0.353110
Total training time: 26.12 seconds.
-- Epoch 3
Norm: 0.09, NNZs: 10000, Bias: -0.999915, T: 1044374, Avg. loss: 0.468034
Total training time: 28.09 seconds.
-- Epoch 3
Norm: 0.09, NNZs: 10000, Bias: -1.000301, T: 1044374, Avg. loss: 0.474449
Total training time: 28.51 seconds.
-- Epoch 3
Norm: 0.04, NNZs: 10000, Bias: -1.002124, T: 1566561, Avg. loss: 0.193149
Total training time: 35.85 seconds.
-- Epoch 4
Norm: 0.06, NNZs: 10000, Bias: -1.002027, T: 1566561, Avg. loss: 0.173986
Total training time: 38.03 seconds.
```

(continues on next page)

(continued from previous page)

```
-- Epoch 4
Norm: 0.13, NNZs: 10000, Bias: -1.004452, T: 1566561, Avg. loss: 0.337721
Total training time: 39.54 seconds.
-- Epoch 4
Norm: 0.07, NNZs: 10000, Bias: -1.001182, T: 1566561, Avg. loss: 0.353068
Total training time: 39.75 seconds.
-- Epoch 4
Norm: 0.07, NNZs: 10000, Bias: -0.999728, T: 1566561, Avg. loss: 0.467977
Total training time: 46.13 seconds.
-- Epoch 4
Norm: 0.07, NNZs: 10000, Bias: -1.000222, T: 1566561, Avg. loss: 0.474392
Total training time: 46.56 seconds.
-- Epoch 4
Norm: 0.04, NNZs: 10000, Bias: -1.001369, T: 2088748, Avg. loss: 0.193139
Total training time: 48.33 seconds.
-- Epoch 5
Norm: 0.12, NNZs: 10000, Bias: -1.003747, T: 2088748, Avg. loss: 0.337711
Total training time: 53.12 seconds.
-- Epoch 5
Norm: 0.06, NNZs: 10000, Bias: -1.001167, T: 2088748, Avg. loss: 0.353051
Total training time: 53.53 seconds.
-- Epoch 5
Norm: 0.05, NNZs: 10000, Bias: -1.001559, T: 2088748, Avg. loss: 0.173979
Total training time: 54.16 seconds.
-- Epoch 5
Norm: 0.03, NNZs: 10000, Bias: -1.001132, T: 2610935, Avg. loss: 0.193134
Total training time: 60.81 seconds.
-- Epoch 6
Norm: 0.06, NNZs: 10000, Bias: -0.999880, T: 2088748, Avg. loss: 0.467954
Total training time: 63.45 seconds.
-- Epoch 5
Norm: 0.06, NNZs: 10000, Bias: -1.000137, T: 2088748, Avg. loss: 0.474368
Total training time: 63.81 seconds.
-- Epoch 5
Norm: 0.11, NNZs: 10000, Bias: -1.002949, T: 2610935, Avg. loss: 0.337706
Total training time: 66.07 seconds.
-- Epoch 6
Norm: 0.06, NNZs: 10000, Bias: -1.000917, T: 2610935, Avg. loss: 0.353042
Total training time: 66.61 seconds.
-- Epoch 6
Norm: 0.05, NNZs: 10000, Bias: -1.000998, T: 2610935, Avg. loss: 0.173974
Total training time: 67.14 seconds.
-- Epoch 6
Norm: 0.03, NNZs: 10000, Bias: -1.000767, T: 3133122, Avg. loss: 0.193130
Total training time: 71.86 seconds.
Convergence after 6 epochs took 71.86 seconds
Norm: 0.06, NNZs: 10000, Bias: -0.999961, T: 2610935, Avg. loss: 0.467942
Total training time: 78.22 seconds.
-- Epoch 6
Norm: 0.05, NNZs: 10000, Bias: -1.000163, T: 2610935, Avg. loss: 0.474355
Total training time: 78.77 seconds.
-- Epoch 6
Norm: 0.11, NNZs: 10000, Bias: -1.002777, T: 3133122, Avg. loss: 0.337703
Total training time: 79.11 seconds.
-- Epoch 7
Norm: 0.05, NNZs: 10000, Bias: -1.000635, T: 3133122, Avg. loss: 0.353037
Total training time: 79.93 seconds.
-- Epoch 7
Norm: 0.04, NNZs: 10000, Bias: -1.000990, T: 3133122, Avg. loss: 0.173972
Total training time: 80.23 seconds.
Convergence after 6 epochs took 80.23 seconds
```

```
[Parallel(n_jobs=-1)]: Done   2 out of   6 | elapsed:  1.3min remaining:  2.7min
```

```
Norm: 0.10, NNZs: 10000, Bias: -1.002764, T: 3655309, Avg. loss: 0.337700
Total training time: 93.07 seconds.
Convergence after 7 epochs took 93.07 seconds
Norm: 0.05, NNZs: 10000, Bias: -0.999862, T: 3133122, Avg. loss: 0.467934
Total training time: 93.08 seconds.
-- Epoch 7
Norm: 0.05, NNZs: 10000, Bias: -1.000170, T: 3133122, Avg. loss: 0.474347
Total training time: 93.70 seconds.
-- Epoch 7
Norm: 0.05, NNZs: 10000, Bias: -1.000561, T: 3655309, Avg. loss: 0.353033
Total training time: 93.97 seconds.
Convergence after 7 epochs took 93.97 seconds
Norm: 0.05, NNZs: 10000, Bias: -1.000004, T: 3655309, Avg. loss: 0.467928
Total training time: 107.42 seconds.
Convergence after 7 epochs took 107.42 seconds
Norm: 0.04, NNZs: 10000, Bias: -1.000248, T: 3655309, Avg. loss: 0.474341
Total training time: 108.28 seconds.
Convergence after 7 epochs took 108.28 seconds
```

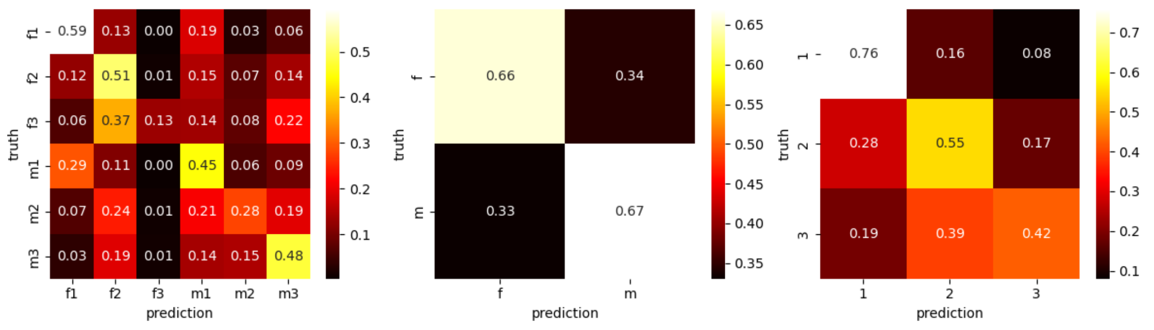
```
[Parallel(n_jobs=-1)]: Done   6 out of   6 | elapsed:  1.8min finished
```

```
SGDClassifier(alpha=0.00390625, max_iter=100, n_jobs=-1, verbose=1)
```

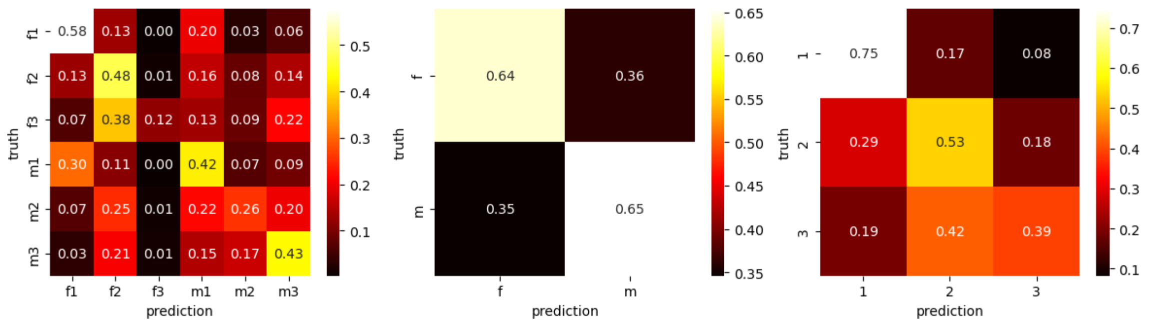
```
y_train_pred = rbfsvm.predict(X_train_rbf)
y_test_pred = rbfsvm.predict(X_test_rbf)
```

```
show_accuracy(y_train, y_train_pred)
show_accuracy(y_test, y_test_pred)
```

```
correct classification rate:          0.4220 (random guessing: 0.1667)
correct classification rate (gender): 0.6626 (random guessing: 0.5000)
correct classification rate (age):    0.5989 (random guessing: 0.3333)
```



```
correct classification rate:          0.3964 (random guessing: 0.1667)
correct classification rate (gender): 0.6465 (random guessing: 0.5000)
correct classification rate (age):    0.5807 (random guessing: 0.3333)
```



Prediction quality on training set:

```
correct classification rate:          0.4348 (random guessing: 0.1667)
correct classification rate (gender): 0.6607 (random guessing: 0.5000)
correct classification rate (age):    0.6243 (random guessing: 0.3333)
```

Prediction quality on test set:

```
correct classification rate:          0.4105 (random guessing: 0.1667)
correct classification rate (gender): 0.6466 (random guessing: 0.5000)
correct classification rate (age):    0.6095 (random guessing: 0.3333)
```

We see that prediction quality is not as good as for fast and simple naive Bayes classification or linear SVC. A reason might be lack of data. Although we have more than 500000 training samples the data set is very sparse in a 37000 dimensional space. If we wanted to have at least one sample per vertex of a 37000 dimensional cube, we would require $2^{37000} = (2^{37})^{1000} > (10^{11})^{1000} = 10^{11000}$ samples. It's very likely that our classes can be seprated by hyperplanes without need for nonlinear separation.

31.2.5 Decision Tree

Due to their simplicity decision trees are well suited for large scale classification problems, too. As we will see below, prediction quality is not as good as with naive Bayes. But for sake of completeness we provide some variants here.

Simple Decision Tree

Scikit-Learn's `DecisionTreeClassifier`[569] by default yields a fully grown tree. With more than 37000 features and more than 500000 samples the tree would be too large to fit into memory. Thus, we cannot use pruning, but have to limit the tree's size in advance.

```
import sklearn.tree as tree
```

```
dtc = tree.DecisionTreeClassifier(max_depth=10)
dtc.fit(X_train, y_train)
```

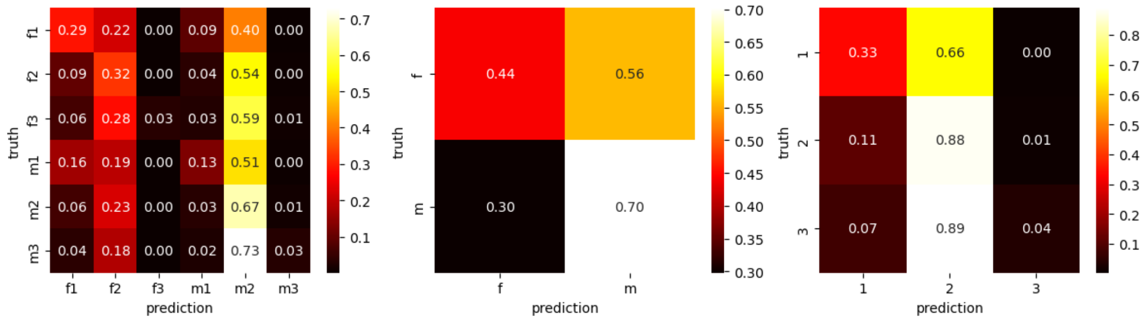
```
DecisionTreeClassifier(max_depth=10)
```

```
y_train_pred = dtc.predict(X_train)
y_test_pred = dtc.predict(X_test)
```

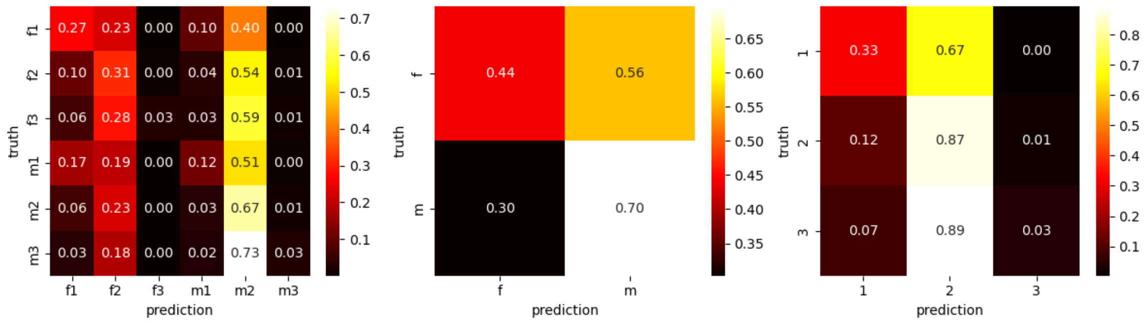
```
show_accuracy(y_train, y_train_pred)
show_accuracy(y_test, y_test_pred)
```

[569] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

```
correct classification rate:          0.3101 (random guessing: 0.1667)
correct classification rate (gender): 0.5736 (random guessing: 0.5000)
correct classification rate (age):    0.5356 (random guessing: 0.3333)
```



```
correct classification rate:          0.3013 (random guessing: 0.1667)
correct classification rate (gender): 0.5691 (random guessing: 0.5000)
correct classification rate (age):    0.5300 (random guessing: 0.3333)
```



Random Forest

Applying bagging to decision trees is known as random forests. Scikit-Learn's `RandomForestClassifier`[570] trains a number of decision trees and then calculates average class probabilites. The class with the highest averaged probability is used as prediction. Again we have to limit the size of the trees.

```
import sklearn.ensemble as ensemble
```

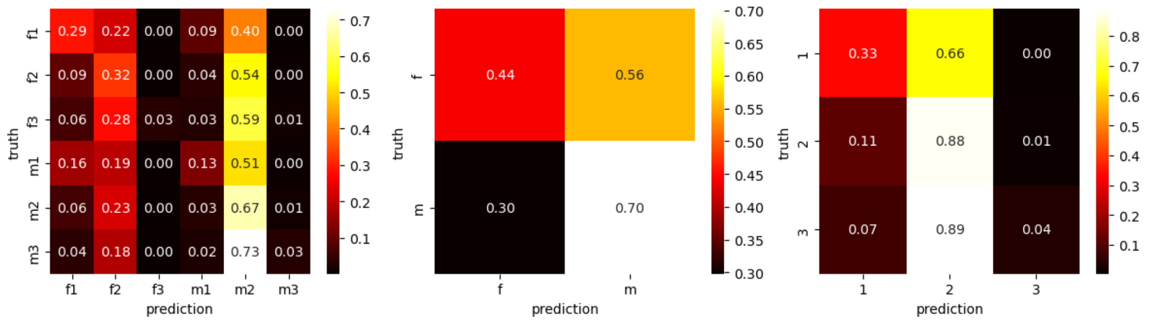
```
rf = ensemble.RandomForestClassifier(n_estimators=100, max_depth=10)
rf.fit(X_train, y_train)
```

```
RandomForestClassifier(max_depth=10)
```

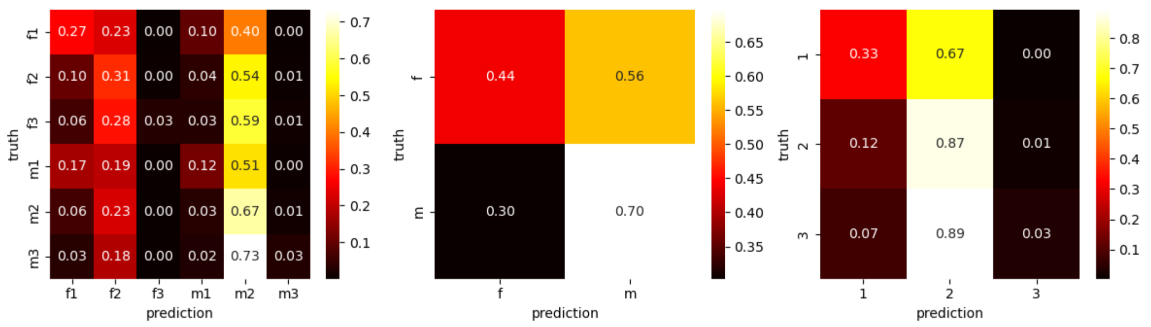
```
show_accuracy(y_train, y_train_pred)
show_accuracy(y_test, y_test_pred)
```

```
correct classification rate:          0.3101 (random guessing: 0.1667)
correct classification rate (gender): 0.5736 (random guessing: 0.5000)
correct classification rate (age):    0.5356 (random guessing: 0.3333)
```

[570] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html



```
correct classification rate:          0.3013 (random guessing: 0.1667)
correct classification rate (gender): 0.5691 (random guessing: 0.5000)
correct classification rate (age):    0.5300 (random guessing: 0.3333)
```



Boosted Trees

We may use AdaBoost in conjuction with decision stumps (trees with depth 1). This is the default behavior of Scikit-Learn's `AdaBoostClassifier`[571].

```
ada = ensemble.AdaBoostClassifier(n_estimators=100)
ada.fit(X_train, y_train)
```

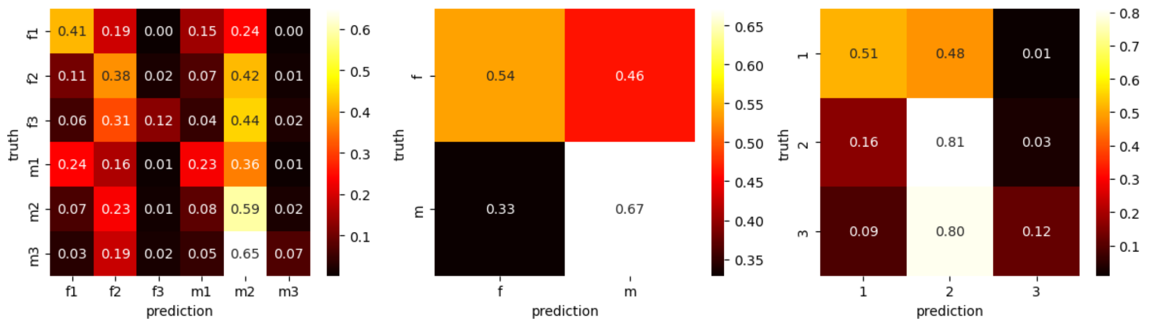
```
AdaBoostClassifier(n_estimators=100)
```

```
y_train_pred = ada.predict(X_train)
y_test_pred = ada.predict(X_test)
```

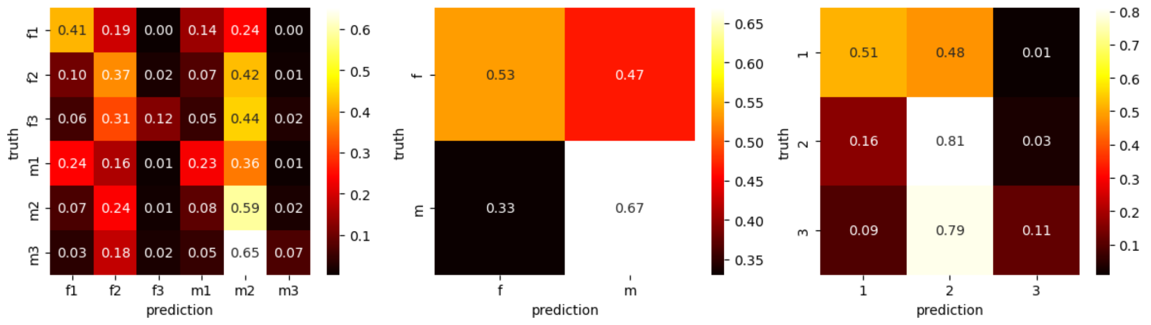
```
show_accuracy(y_train, y_train_pred)
show_accuracy(y_test, y_test_pred)
```

```
correct classification rate:          0.3560 (random guessing: 0.1667)
correct classification rate (gender): 0.6050 (random guessing: 0.5000)
correct classification rate (age):    0.5800 (random guessing: 0.3333)
```

[571] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html



```
correct classification rate:          0.3540 (random guessing: 0.1667)
correct classification rate (gender): 0.6020 (random guessing: 0.5000)
correct classification rate (age):    0.5805 (random guessing: 0.3333)
```



Part VII

Unsupervised Learning

CHAPTER

THIRTYTWO

TYPICAL TASKS

Unsupervised machine learning algorithms extract information from data sets without the need for examples of the information to extract. The algorithm determines what to extract and humans have to interpret the extracted information to obtain insights into the data. Thus, it's very easy to collect data sets suitable for unsupervised learning, but interpretation of results may be difficult.

Unsupervised learning is an extremely wide field. Here we focus on *clustering* and also consider *dimensionality reduction*. Further, we'll have a first glance at *generative models*. But there exist other subfields, too, *anomaly detection* and *association analysis* for instance. Most techniques can be applied to different tasks.

Like for supervised learning we denote the data space by X and the items of the data set under consideration by $x_1, \ldots, x_n$. Almost always we will have $X = \mathbb{R}^m$. Again we need a training data set for fitting a model. Validation and test sets are used for hyperparameter optimization and model evaluation as before.

32.1 Clustering

Clustering aims at finding subsets of similar items in large data sets.

Details depend on the application in mind.

- Do we want to find *hard clusters* (either a sample belongs to a cluster or not) or *soft clusters* (score/probability for each combination of samples and clusters)?
- Shall all samples belong to some cluster or do we allow for *outliers*?
- May clusters overlap (some samples belong to more than one cluster)?

There exist many different approaches for clustering algorithms. Main classes are:

- *Centroid-based algorithms* represent each cluster by one point (midpoint or centroid). Which samples belong to which clusters is determined by some rule involving those midpoints.
- *Density-based algorithms* look at the distances between samples and define clusters to be subsets of closely spaced samples.
- *Distribution-based algorithms* represent each cluster be a probability distribution. A sample belongs to the cluster for which the sample's probability is highest.
- *Hierarchical algorithms* generate a sequence of clusterings. Either starting with as many clusters as there are samples (and then coarsening the clustering) or starting with one cluster (and then refining).

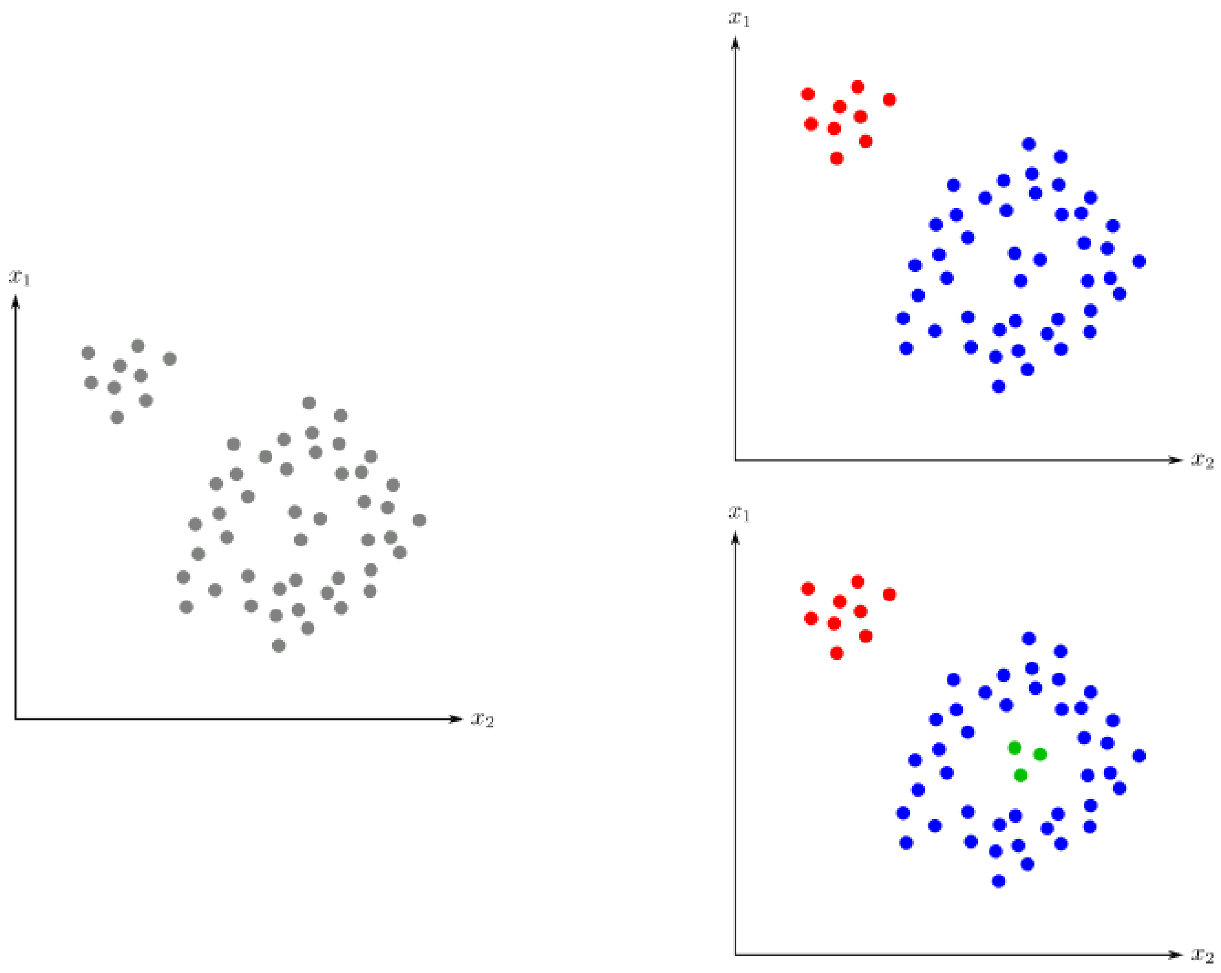


Fig. 32.1: How to define the term *cluster* is not as straight-forward as it seems.

32.2 Dimensionality Reduction

Dimensionality reduction tries to reduce the number of features without loosing too much information. We already know principal component analysis as a linear dimensionality reduction technique. Here, 'linear' means that the mapping from the high dimensional data space to the lower dimensional space is linear (a matrix).

Nonlinear dimensionality reduction techniques are more powerful, but also much more computationally expensive.

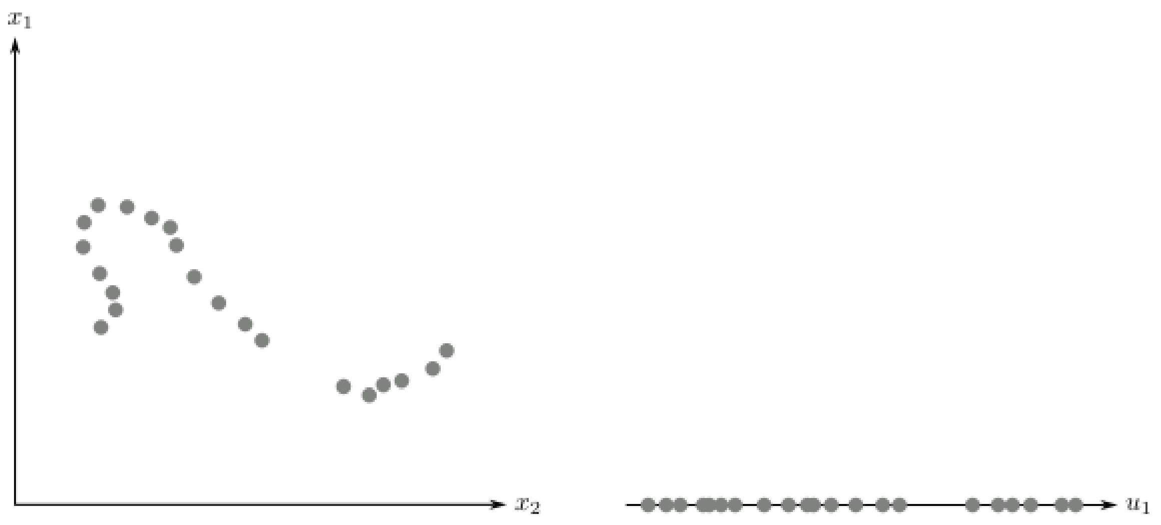


Fig. 32.2: Often data is not scattered over the whole space but lives close to a lower dimensional nonlinear manifold.

32.3 Generative Models

Unsupervised learning algorithms may not only learn to distinguish between similar and dissimilar samples. Some algorithms yield so called *generative models*. Generative models contain all information necessary to automatically create new samples similar to samples in the training data set.

Generative models can be used for generating natural looking artifical images[572] or for generating art[573], for instance.

[572] https://en.wikipedia.org/wiki/StyleGAN
[573] https://en.wikipedia.org/wiki/Edmond_de_Belamy

CHAPTER

THIRTYTHREE

QUALITY MEASURES FOR CLUSTERING

Evaluating the quality of a clustering is not as simple as it looks at first glance. Almost always we do not have a ground truth at hand (external evaluation). Instead we can only look at size and shape of clusters themselves (internal evaluation). There exist lots of internal evaluation metrics. Below we only consider two examples. There is no best metric, because cluster evaluation heavily depends on the intended application. The only reliable evaluation metric is human inspection. But human inspection is restricted to visualizations, which are not available for high dimensional data sets. Dimensionality reduction techniques may help.

In the following we represent clusters as subsets of our data set. In other words, a cluster C is a subset of $\{x_1, \ldots, x_n\}$.

33.1 Silhouette Score

The silhouette score relates intra-cluster distances to inter-cluster distances. It is defined for each sample of a data set. The silhouette score of a whole data set then is the mean score of all samples.

Given some distance measure $d : X \times X \to [0, \infty)$ (usually Euclidean distance) the intra-cluster distance of a sample x and the cluster C the sample belongs to is the mean distance of the sample to all other samples in the cluster:

$$\text{intra}(x, C) := \frac{1}{|C| - 1} \sum_{\tilde{x} \in C} d(x, \tilde{x}).$$

The inter-cluster distance of a sample x to a cluster $\tilde{C}$ the sample does not belong to is the mean distance of the sample to all samples in the cluster under consideration:

$$\text{inter}(x, \tilde{C}) := \frac{1}{|\tilde{C}|} \sum_{\tilde{x} \in \tilde{C}} d(x, \tilde{x}).$$

Now the silhouette score of a sample x is the ratio of the intra-cluster distance and the smallest inter-cluster-distance:

$$a := \text{intra}(x, C), \quad b := \min_{\tilde{C} \neq C} \text{inter}(x, \tilde{C}), \quad \text{silhouette}(x) := \frac{b - a}{\max\{a, b\}}.$$

If x is the only element in C, then this formula does not work and one sets the silhouette score to zero.

Silhouette score always lie in $[-1, 1]$. It is the higher the lower the intra-cluster distance is and the higher the inter-cluster distance is. Thus, high silhouette score for a sample indicates that it belongs to a cluster well separated from all other clusters. Score close to 0 indicates that the sample belongs to overlapping clusters. A score close to -1 indicates a missclustering (sample is closer to other clusters than to its own cluster).

Silhouette score of a data set represents the average clustering quality. Many missclustered samples result in negative silhouette score and so on. Note, that a silhouette score close to zero may indicate that half the samples have been missclustered as well as that there is no clustering (all clusters heavily overlap).

Important: Silhouette score depends on the chosen distance and, thus, on scaling of each feature. If some feature has much higher numerical values than other features, then that feature will dominate the distances.

Another noteworthy point is that silhouette scores are more reliable if clusters are convex. Else inter-cluster distances might be smaller than intra-cluster distances although clusters were correctly identified.

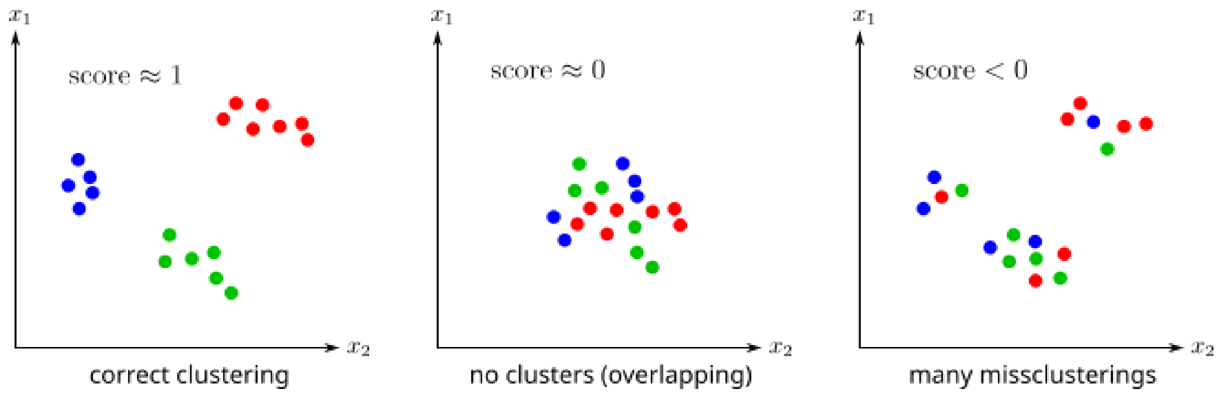


Fig. 33.1: Silhouette scores for different clustering results.

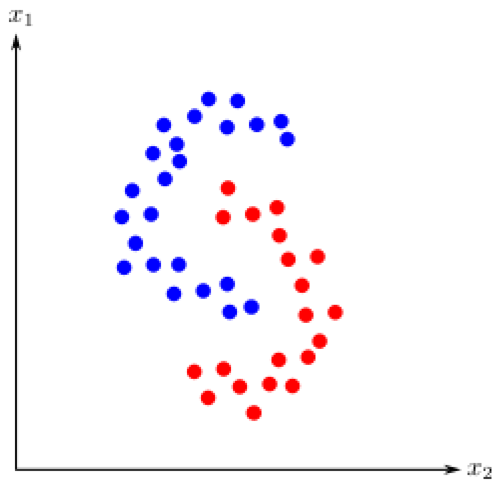


Fig. 33.2: For non-convex clusters silhouette score may indicate bad clustering results although clusters have been identified correctly.

33.2 Davies-Bouldin Index

The Davies-Bouldin index relates cluster diameters to distances between clusters. It is defined for each pair of clusters. The Davies-Bouldin index of a single cluster is the worst Davies-Bouldin index of each pair containing the cluster under consideration. The Davies-Bouldin index of a whole clustering is the mean Davies-Bouldin index of all clusters.

To define the Davies-Bouldin index we need the *centroid* of a cluster C. It's the coordinatewise arithmetic mean of all samples in the cluster:

$$\text{cent}(C) := \frac{1}{|C|}\sum_{x\in C} x.$$

The cluster radius can be defined as the mean distance of samples to the cluster's centroid:

$$r(C) := \frac{1}{|C|},\sum_{x\in C} d(x,\text{cent}(C))$$

with some distance measure d. Usually d is the Euclidean distance, because the notion of centroid is based on considerations involving Euclidean distances. For other distance measures introducing a sensible notion of centroids is difficult. Given two clusters C_1 and C_2 we may define the cluster distance as the distance of their centroids:

$$\text{dist}(C_1,C_2) := d(\text{cent}(C_1),\text{cent}(C_2)).$$

The Davies-Bouldin index of two clusters C_1 and C_2 is

$$\text{DB}(C_1,C_2) := \frac{r(C_1)+r(C_2)}{\text{dist}(C_1,C_2)}.$$

It takes values in $[0,\infty)$ and is the closer to zero the smaller the clusters are and the higher the distance between clusters is.

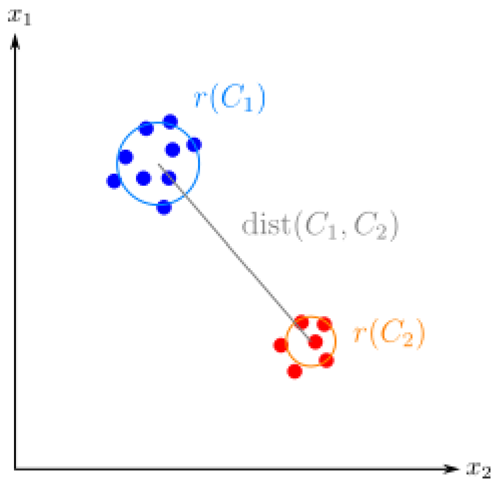


Fig. 33.3: Davies-Bouldin index relates distance between clusters to cluster diameters.

A Davies-Bouldin index above 1 indicates overlapping clusters (at least if the clusters' shapes are close to spheres). If clusters are not sphere shaped the Davies-Boulding index does not yield useful information.

Note that the Davies-Bouldin index of two clusters is symmetric, that is, does not depend on the ordering of the clusters. If there are more than two clusters, the Davies-Bouldin index of each cluster is

$$\text{DB}(C) := \max_{\tilde{C}\neq C} \text{DB}(C,\tilde{C})$$

and the Davies-Bouldin index of the whole clustering is mean Davies-Bouldin index of all clusters.

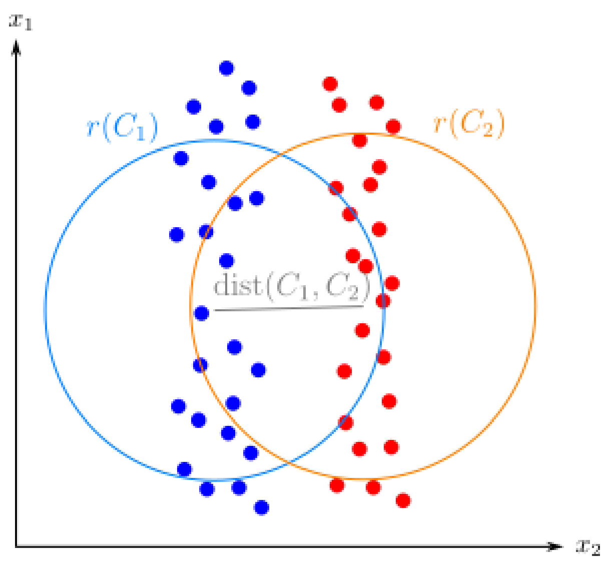


Fig. 33.4: If clusters are not sphere shaped Davies-Bouldin index may indicate bad clustering although clustering is correct.

CHAPTER

THIRTYFOUR

CENTROID-BASED CLUSTERING (K-MEANS)

Centroid-based clustering algorithms represent clusters by single points in the data space (center points, mostly the clusters' centroids). Corresponding clusters then are given by some distance condition. All points closer to some center point than to all others belong to one cluster, for instance. Consequently, centroid-based methods yield sphere-shaped clusters with concrete shape depending on the chosen distance measure.

The major centroid-based clustering method is k-means. Others, not discussed here, are k-medians[574] and k-medoids[575].

Related projects:

- *MNIST Character Recognition* (page 931)
  - *Semisupervised Classification* (page 940)
- *Supermarket Customers* (page 983)

34.1 k-Means Idea

Clusters obtained from k-means method are determined by center points. Given k points $u_1, \ldots, u_k$ in $\mathbb{R}^m$ (the centers) corresponding clusters are defined by

$$C(u_l) := \{x \in \{x_1, \ldots, x_n\} : d(x, u_l) \leq d(x, u_\lambda) \text{ for all } \lambda \neq l\}.$$

A cluster contains all samples which are closer to the cluster's center than to other clusters' centers. In case of equality a sample formally belongs to two or more clusters. In practice, a cluster is chosen by chance to obtain mutually disjoint clusters.

A prescribed clustering (collection of subsets of our data set) may or may not have a set of center points, that is, a set such that for each point in the set all points in the corresponding cluster are closer to that point than to any other cluster's center (simple example: one cluster belongs to the convex hull of another cluster).

Although k-means is closely related to centroids (see below), a cluster's centroid not necessarily is a center point.

The k-means method aims at minimizing the average distance of samples to the closest cluster center. The number k of clusters has to be provided in advance. The outcome are the cluster centers.

In formulas, we want to minimize the function

$$(u_1, \ldots, u_k) \mapsto \frac{1}{n} \sum_{l=1}^{k} \sum_{x \in C(u_l)} d(x, u_l)$$

with respect to all possible cluster centers $\{u_1, \ldots, u_k\}$. The distance measure d almost always is the squared Euclidean distance.

Finding optimal cluster centers is closely related to finding optimal locations for warehouses, hospitals, fire stations, and so on (see Voronoi diagram[576]).

[574] https://en.wikipedia.org/wiki/K-medians_clustering
[575] https://en.wikipedia.org/wiki/K-medoids
[576] https://en.wikipedia.org/wiki/Voronoi_diagram

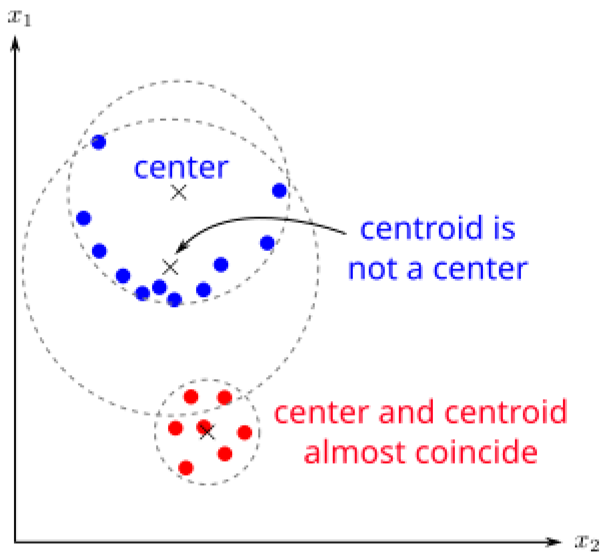


Fig. 34.1: Centroids not necessarily are center points.

34.2 Naive k-Means Algorithm

Naive k-means algorithm (also known as Lloyd's algorithm) starts with a set of initial center points $u_1, \ldots, u_k$ and repeats the following two steps:

- Build clusters $C(u_1), \ldots, C(u_k)$ for current centers $u_1, \ldots, u_k$.
- Update $u_1, \ldots, u_k$ to be the centroids of the current clusters.

Iteration is stopped if clusters do not change anymore. One can show that this method always converges (that is, stops) and that the resulting center points are a local (not a global!) minimizer of the above objective function. From the stopping criterion we immediately see, that centers computed by naive k-means always are centroids of their clusters.

The described algorithm takes two parameters:

- the number k of clusters to find,
- initial center points $u_1, \ldots, u_k$ defining the initial clustering.

There exist several methods to choose these parameters. In some cases k can be obtained from domain knowledge. Else k has to be obtained from typical hyperparameter optimization techniques. Center points may be initialized randomly or by some specialized initialization routines (see below).

34.3 Implementation from Scratch

Before we have a closer look at different methods for choosing k and initial centers we implement naive k-means from scratch. Later we will use Scikit-Learn's implementation.

First we create some synthetic data to be clustered.

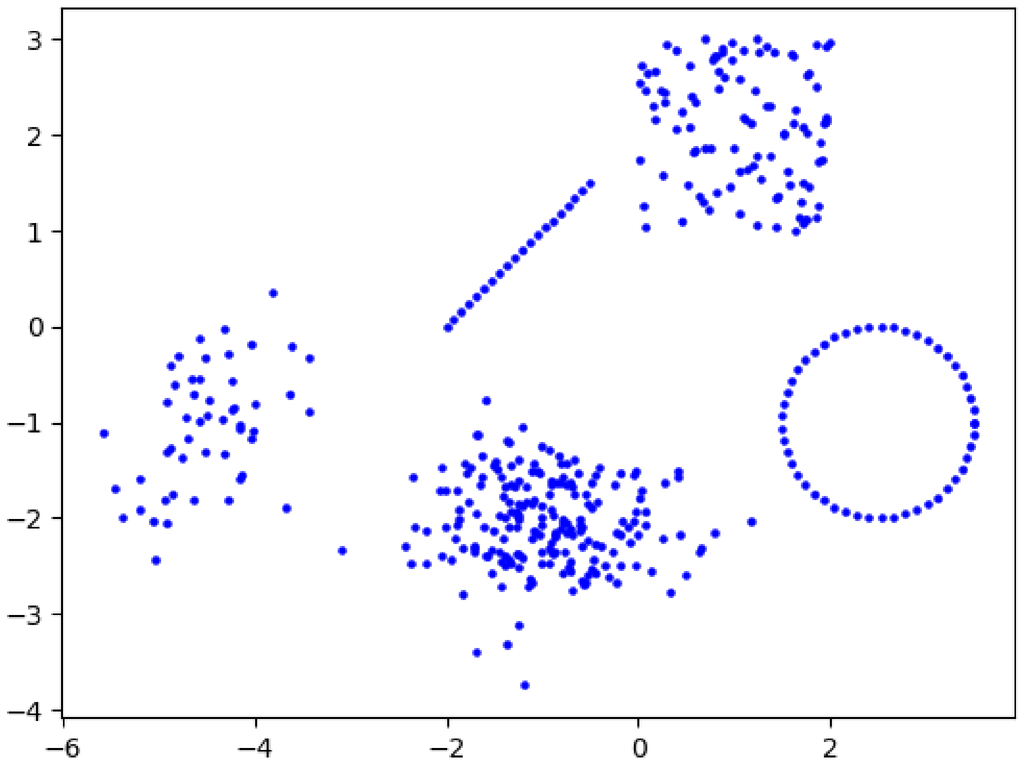
```
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.default_rng(0)
```

```
# 5 clusters in 2 dimensions

n1, n2, n3, n4, n5 = 100, 200, 20, 50, 50
```

(continues on next page)

(continued from previous page)

```
n = n1 + n2 + n3 + n4 + n5

X1 = rng.uniform((0, 1), (2, 3), (n1, 2))
X2 = rng.multivariate_normal((-1, -2), ((0.5, 0), (0, 0.2)), n2)
X3 = np.linspace(-1, 0.5, n3).reshape(-1, 1) * np.ones((1, 2)) + np.array([[-1,␣
↪1]])
X4 = rng.multivariate_normal((-4.5, -1), ((0.2, 0.1), (0.1, 0.4)), n4)
phi = np.linspace(0, 2 * np.pi, n5).reshape(-1, 1)
X5 = np.array([[2.5, -1]]) + np.concatenate((np.cos(phi), np.sin(phi)), axis=1)

X = np.concatenate((X1, X2, X3, X4, X5))

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c='b')
ax.axis('equal')
plt.show()
```



k-means takes the number of clusters `k` and initial centers `init_centers` as parameters. Iteration is stopped if current and previous clustering coincide. To prevent too many iteration we also set a maximum number of iterations `max_iter`.

Center points will be stored rowwise in NumPy arrays (like for the samples in `X`). In each iteration we have to calculate all distances between samples and center points. Those distances will be stored in an `(n, k)` NumPy array `dists`. A clustering will be represented by a 1d NumPy array `clusters` of length `n` containing for each sample the index of the closest center point. When updating the centers we have to keep the old centers to check the stopping criterion. After each k-means step we plot the intermediate result.

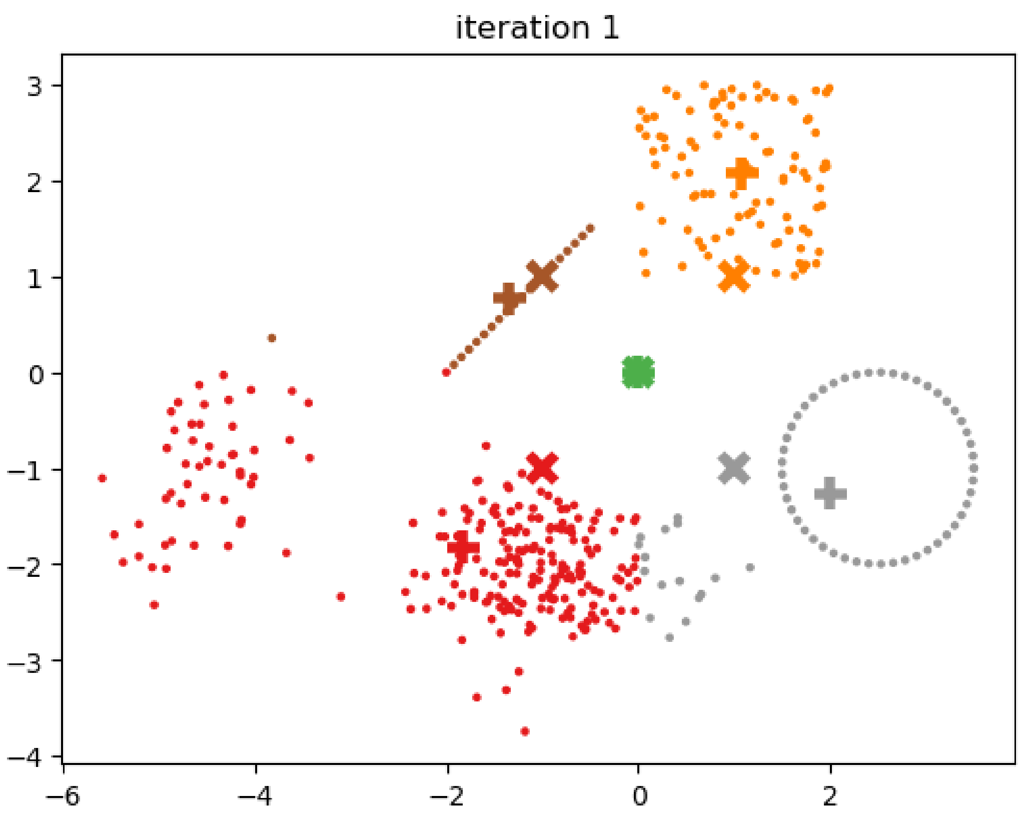
```
k = 5
init_centers = np.array([[-1, -1], [0, 0], [1, 1], [-1, 1], [1, -1]], dtype=float)
#k = 4
#init_centers = np.array([[-1, -1], [1, 1], [-1, 1], [1, -1]], dtype=float)
#k = 3
```

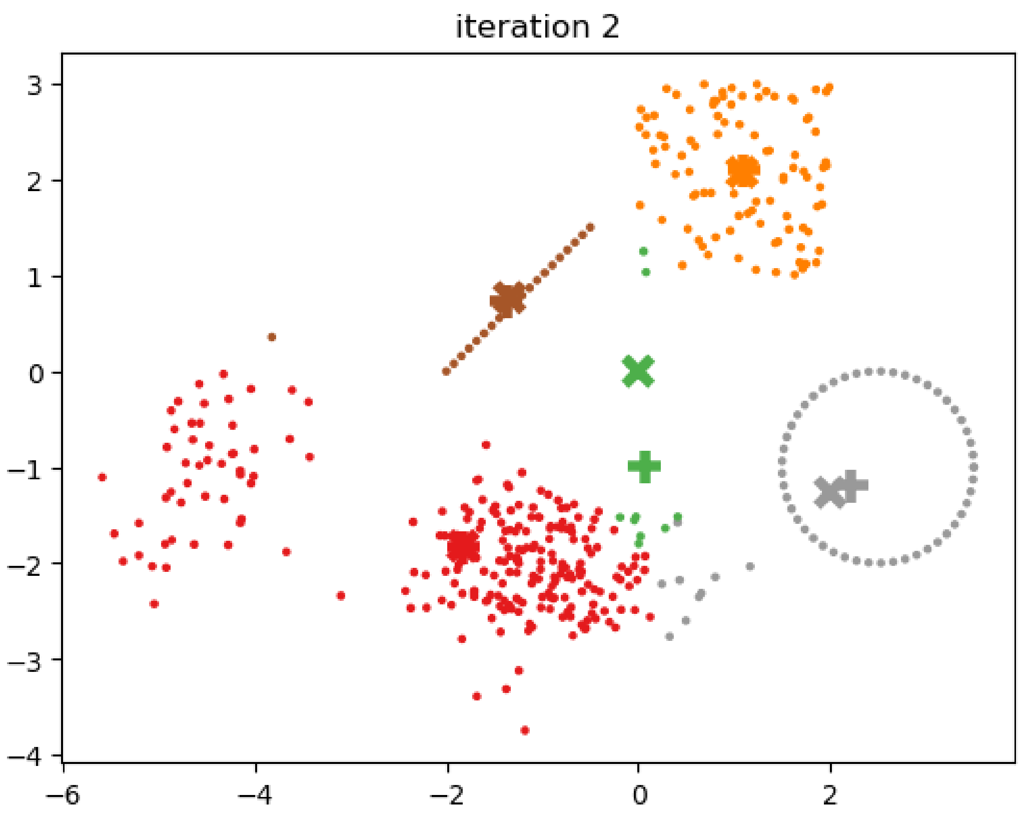
(continues on next page)

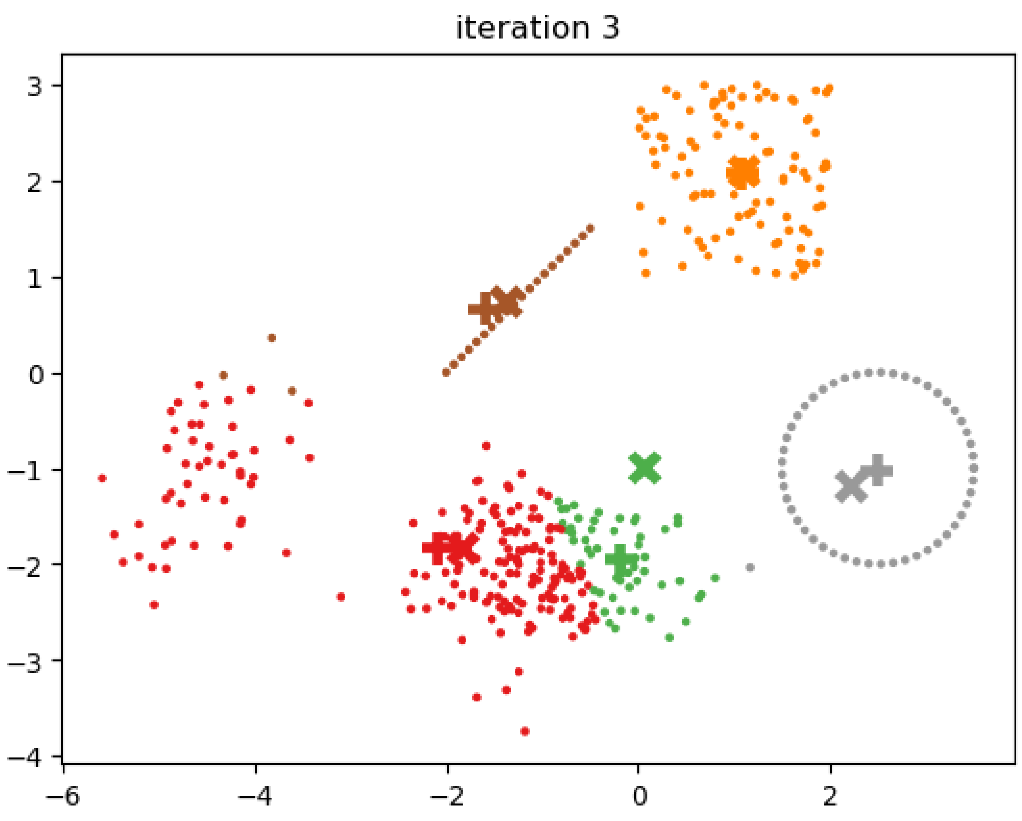
(continued from previous page)

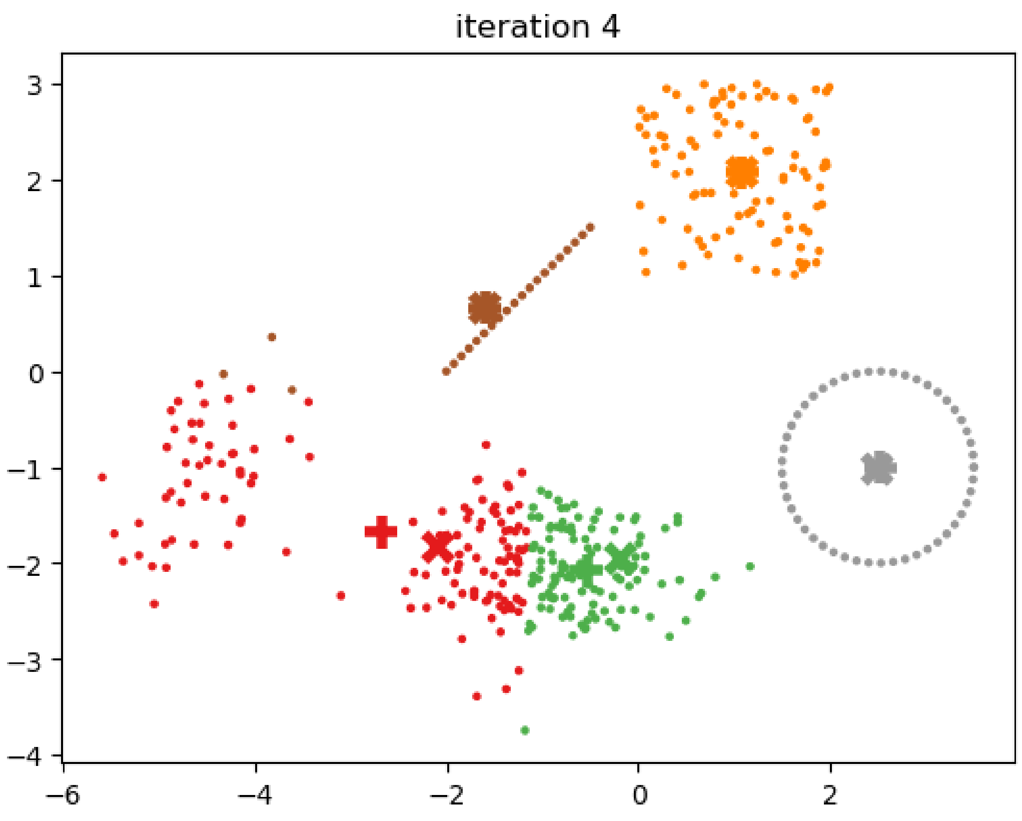
```
#init_centers = np.array([[-1, -1], [-1, 1], [1, -1]], dtype=float)
#k = 6
#init_centers = np.array([[-1, -1], [0, 0], [1, 1], [-1, 1], [1, -1], [0.5, 0.5]],
↪ dtype=float)
#k = 7
#init_centers = np.array([[-1, -1], [0, 0], [1, 1], [-1, 1], [1, -1], [0.5, 0.5],␣
↪[-0.5, -0.5]], dtype=float)

max_iter = 20

print('+ is current centroid')
print('x is previous centroid')

centers = init_centers
for i in range(0, max_iter):

    fig, ax = plt.subplots()

    # get clusters
    if i > 0:
        old_clusters = clusters
    dists = np.empty((n, k))
    for l in range(0, k):
        dists[:, l] = np.sum((X - centers[l, :]) ** 2, axis=1)
    clusters = dists.argmin(axis=1)

    # plot clusters and centers
    ax.scatter(X[:, 0], X[:, 1], s=5, c=clusters, cmap='Set1')
    ax.scatter(centers[:, 0], centers[:, 1], s=100, c=range(0, k), cmap='Set1',␣
↪marker='x', linewidth=4)

    # update centers
    old_centers = centers.copy()
    for l in range(0, k):
        if np.any(clusters == l):    # update only if cluster is not empty
            centers[l, :] = X[clusters == l, :].mean(axis=0)
        else:
            print('empty cluster')

    # stopping criterion
    if i > 0 and (old_clusters == clusters).all():
        print('stopping criterion satisfied after {} iterations'.format(i))
        break

    # plot new centers
    ax.scatter(centers[:, 0], centers[:, 1], s=150, c=range(0, k), cmap='Set1',␣
↪marker='+', linewidth=4)

    ax.set_title('iteration ' + str(i + 1))
    ax.axis('equal')

    plt.show()

else:
    print('max_iter reached')
```

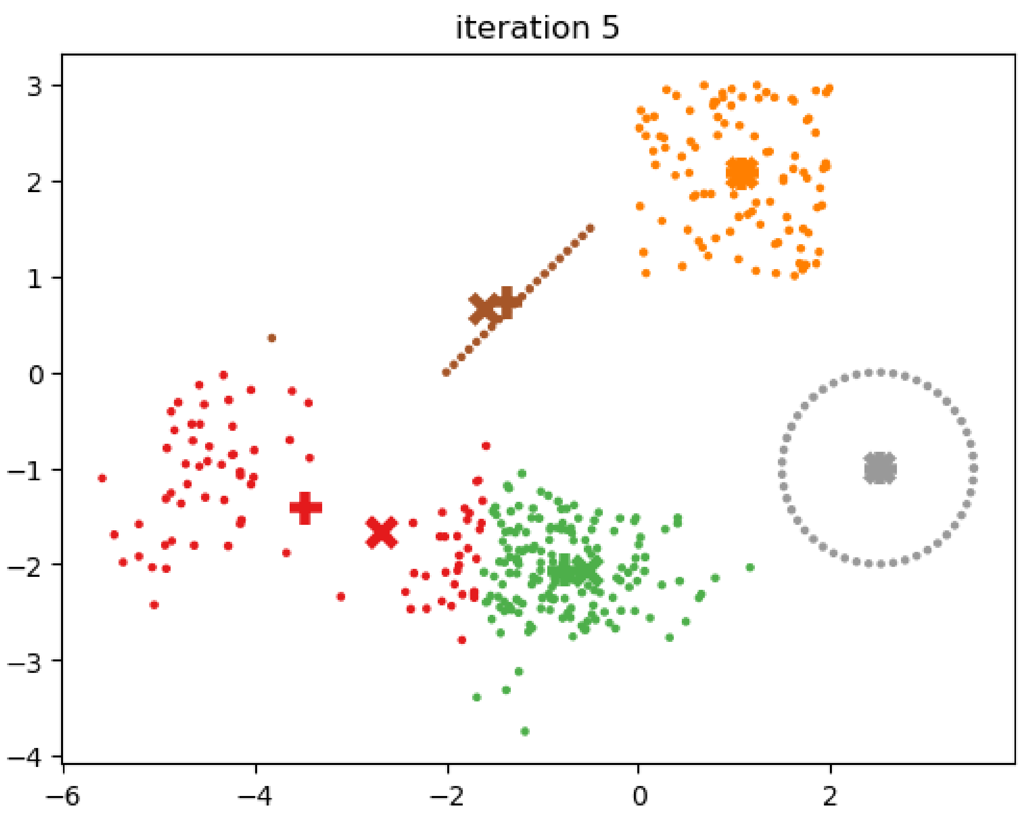
```
+ is current centroid
x is previous centroid
empty cluster
```

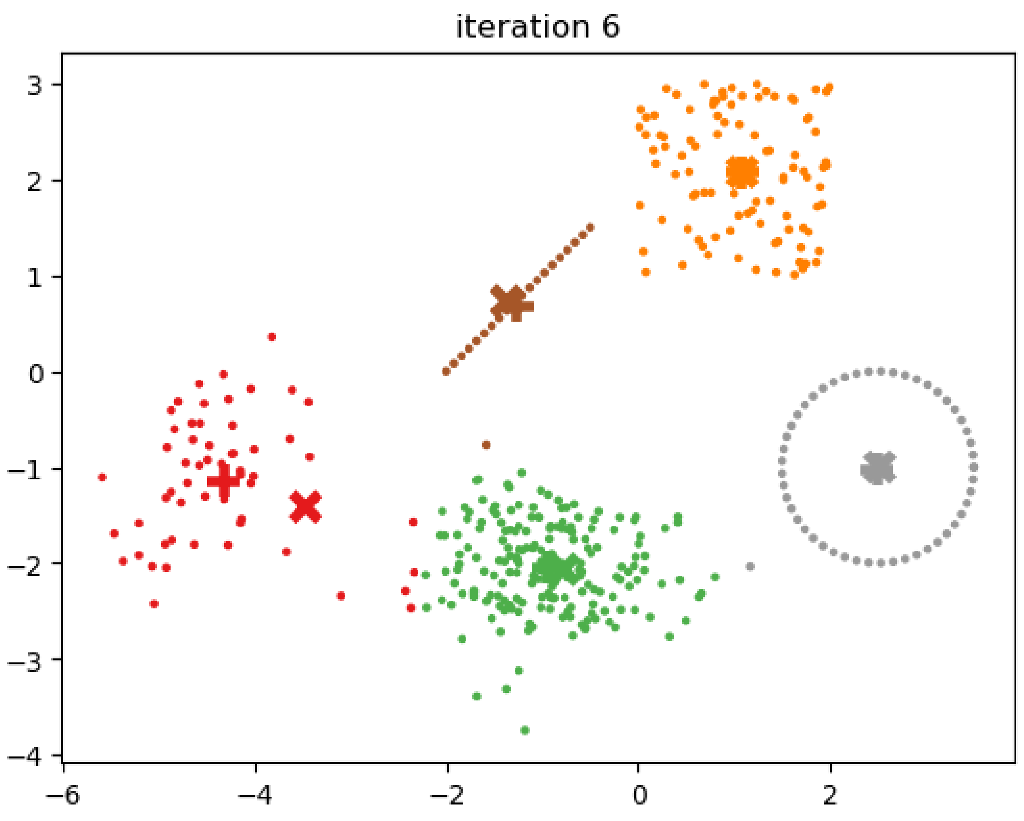

iteration 1
3
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2

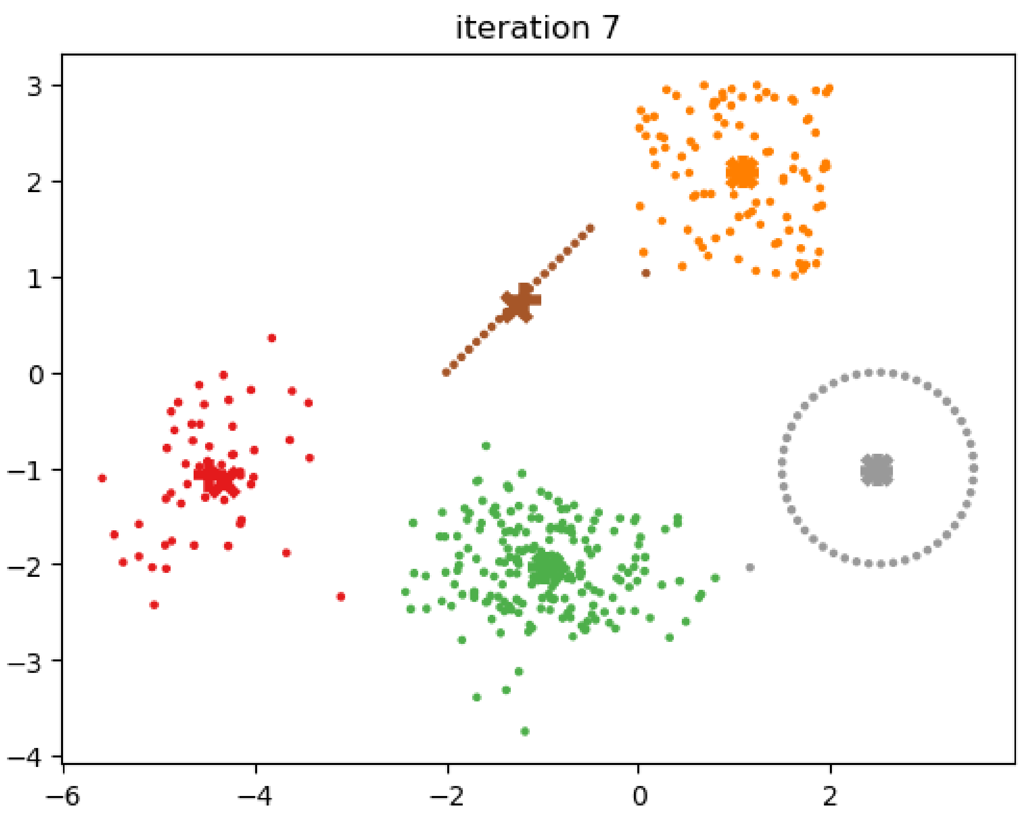

iteration 2
3
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2


iteration 3
3
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2

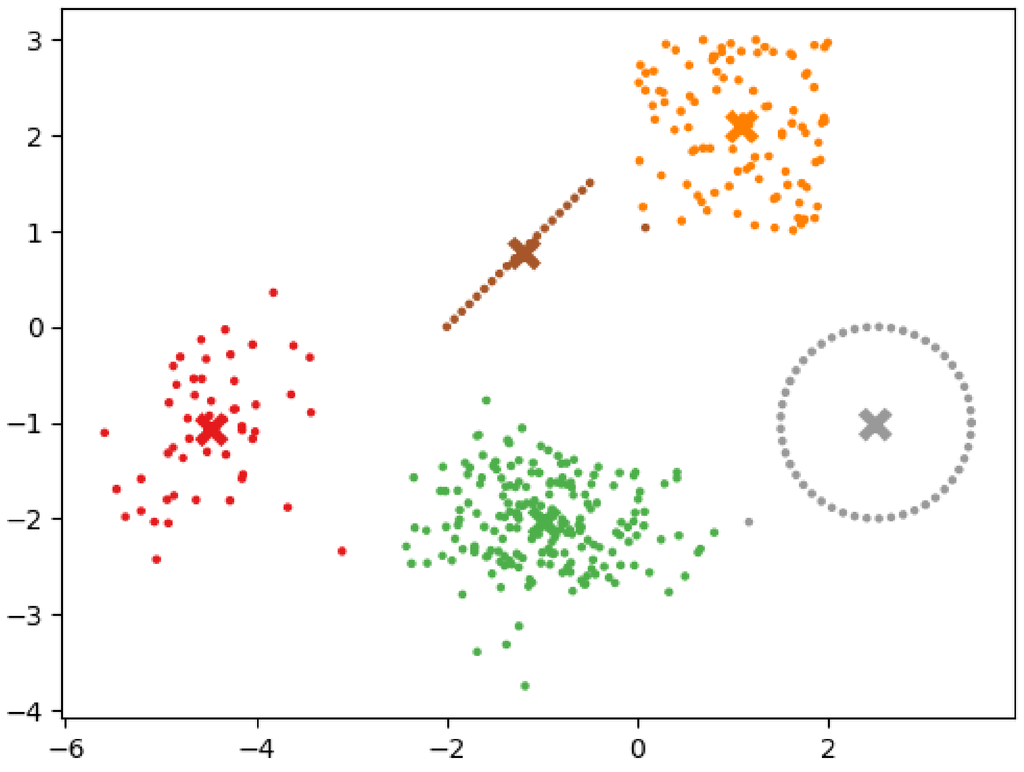

iteration 4
3
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2


iteration 5
3
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2


iteration 6
3
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2



```
stopping criterion satisfied after 7 iterations
```



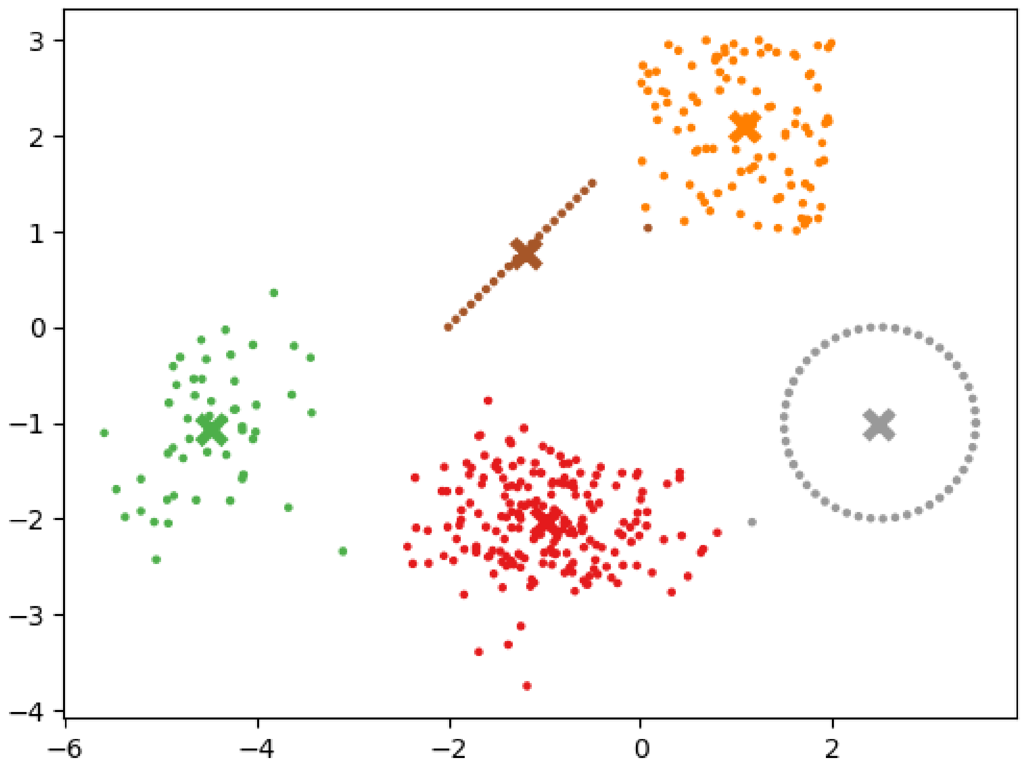
34.4 Implementation with Scikit-Learn

Scikit-Learn implements a `KMeans`[577] class in its `cluster` module. The `fit` method generates the clustering, resulting in a list of cluster centers. The `predict` method assigns cluster labels to samples. Note that `fit` already computes labels for training data. So we do not have to call `predict` on training data.

The `KMeans` constructor takes several arguments described in the documentation. Values for the `init` parameter will be described below. In case of random initialization `n_init` is the number of k-means runs. From multiple runs Scikit-Learn chooses the result with lowest inertia, that is, with smallest sum of squared distances of samples to their closest cluster center. Next to naive k-means Scikit-Learn supports the (often) more efficient Elkan method, which uses the triangle inequality to avoid unnecessary distance calculations.

```
import sklearn.cluster as cluster
```

```
k = 5
init_centers = np.array([[-1, -1], [0, 0], [1, 1], [-1, 1], [1, -1]], dtype=float)

km = cluster.KMeans(n_clusters=k, init=init_centers, n_init=1)
km.fit(X)
centers = km.cluster_centers_
clusters = km.labels_

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c=clusters, cmap='Set1')
ax.scatter(centers[:, 0], centers[:, 1], s=100, c=range(0, k), cmap='Set1',␣
↪marker='x', linewidth=4)
ax.axis('equal')
plt.show()
```



[577] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

34.5 Random Initialization

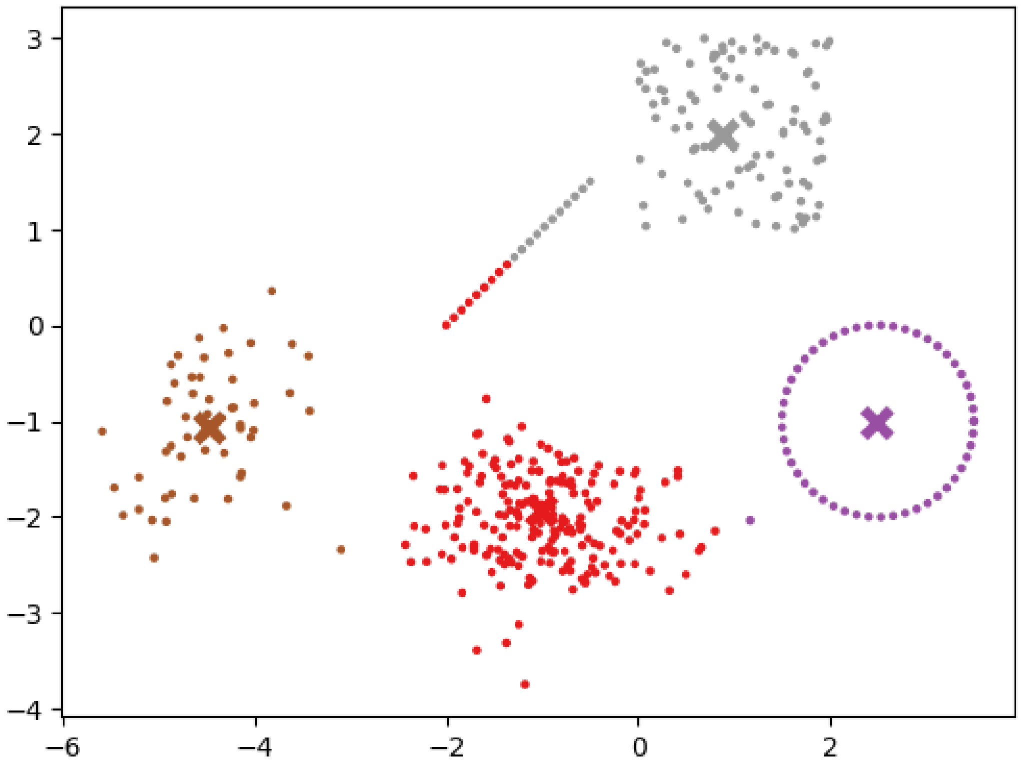
There exist (at least) two approaches for randomly choosing initial cluster centers:

- Choosing k random samples from the training data.
- Randomly assign samples to k clusters and take the centroids as initial centers.

The second approach results in closely spaced initial centers, whereas the first yields initial centers scattered over the data set. Scikit-Learn only implements the first approach, which can be activated with `init='random'`.

If we use random initialization the `n_init` parameter should be greater than one (defaults to 10). Scikit-Learn will run the algorithm `n_init` times and choose the best result (lowest inertia).

```
k = 4

km = cluster.KMeans(n_clusters=k, init='random', n_init=10)
km.fit(X)
centers = km.cluster_centers_
clusters = km.labels_

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c=clusters, cmap='Set1')
ax.scatter(centers[:, 0], centers[:, 1], s=100, c=range(0, k), cmap='Set1',␣
↪marker='x', linewidth=4)
ax.axis('equal')
plt.show()
```



34.6 k-Means++ for Initializing Centers

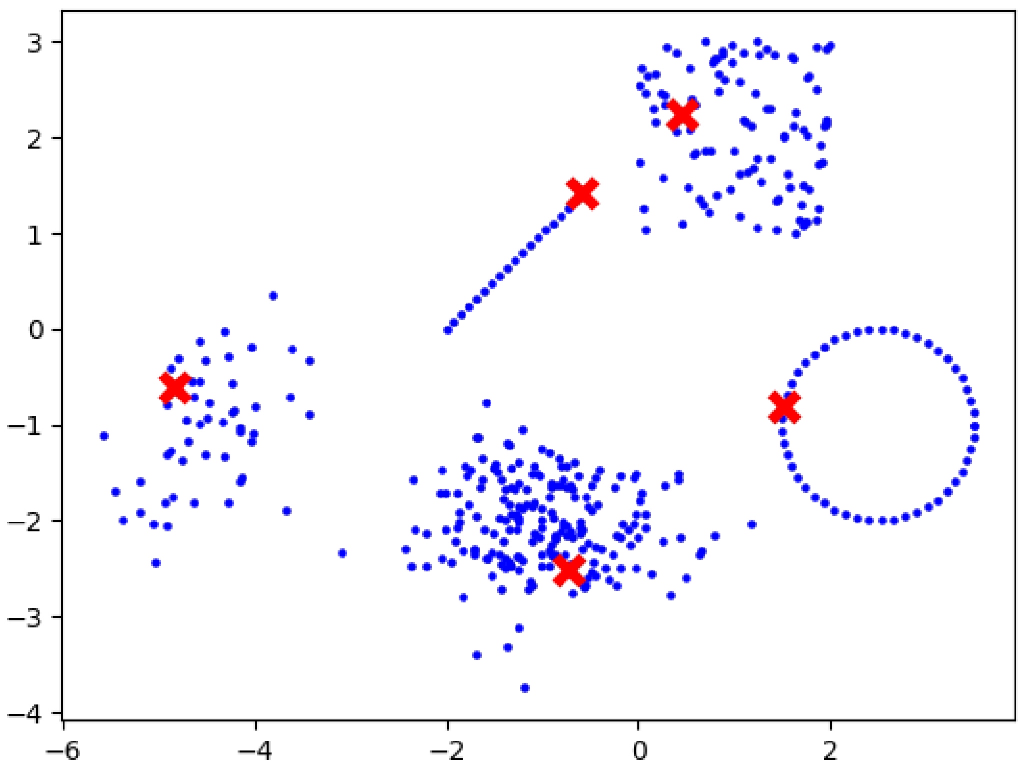
k-means++ is not an improved variant of k-means, but an algorithm for choosing good initial cluster centers for k-means. Here 'good' means fast convergence of k-means based on the initial centers.

k-means++ chooses all initial centers randomly from the training data set (like random initialization), but only the first center is chosen uniformly at random. After the first center has been chosen the following procedure is repeated until k centers have been found:

- For each sample (except the ones already chosen as center point) calculate the distance to the closest center.
- Choose a sample as next center point at random with probabilities proportional to the squared distances.

Samples far away from already existing center points are more likely to become the next center. Thus, initial centers chosen by k-means++ will be scattered over the whole data set and closely spaced centers are very unlikely.

```
k = 5

init_centers = np.empty((k, X.shape[1]))

# first center uniformly at random
init_centers[0, :] = X[rng.integers(0, X.shape[0]), :]

for l in range(1, k):

    # calculate distances to closest center (l centers already exist)
    dists = np.empty((n, l))
    for j in range(0, l):
        dists[:, j] = np.sum((X - init_centers[j, :]) ** 2, axis=1)
    min_dists = dists.min(axis=1)

    # next center with probability proportional to distance to closest center
    init_centers[l, :] = rng.choice(X, 1, p=min_dists/min_dists.sum())

# plot
fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c='b')
ax.scatter(init_centers[:, 0], init_centers[:, 1], s=100, c='r', marker='x',␣
↪linewidth=4)
ax.axis('equal')
plt.show()
```

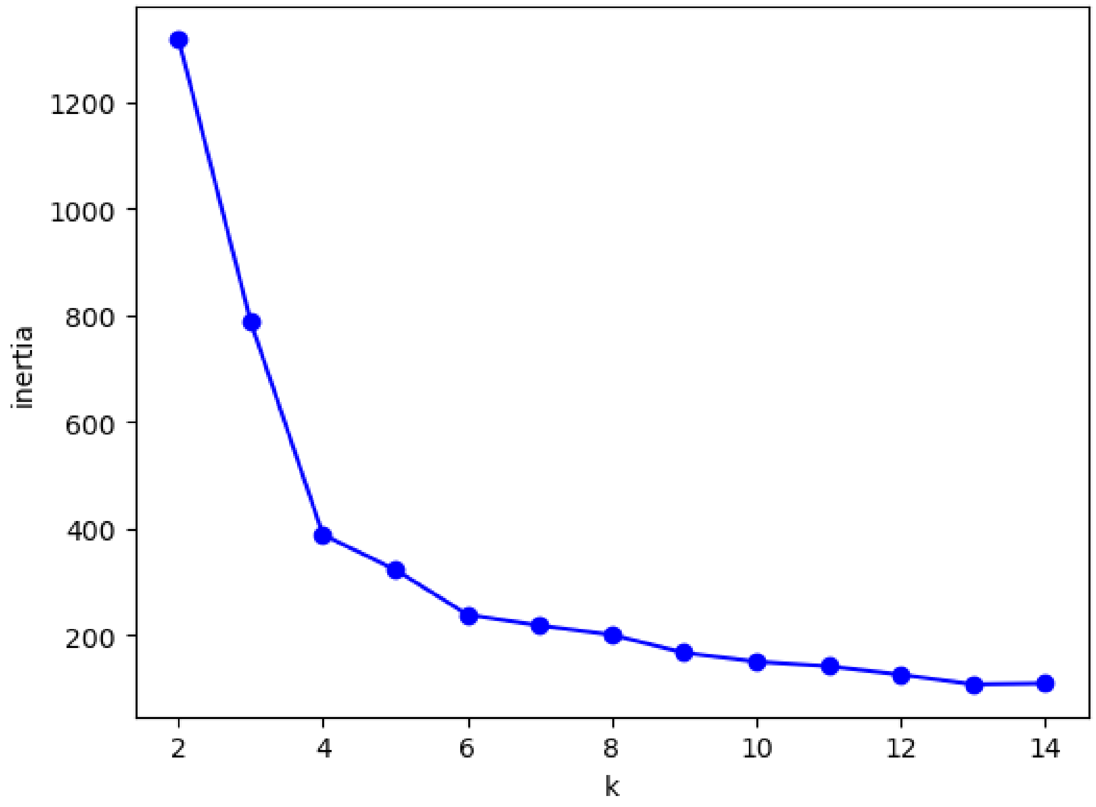


34.7 Choosing k

k-means aims to minimize the sum of distances of all samples to the closest cluster center. Thus, a suitable k should yield small objective function. Obviously $k = n$ with each sample being a cluster center would be the best choice, but this is not our intention when clustering data. Instead we want to have a sensible number of clusters.

Starting with $k = 2$ we may run k-means for an increasing sequence of values for k. The more clusters we allow, the smaller the objective in the above minimization problem. If k becomes larger than the number of clusters in the data, then the drop in the objective will be much smaller than for smaller k. If we plot the objective values against the number of clusters we should see an elbow-like curve. The k at the elbow should be chosen. This heuristic approach is known as *Elbow method*. Often it works quite well, but especially in case of clusters not well separated the elbow may be hard to identify. Another drawback is, that reliable automatic detection of the elbow's position is difficult.

```
ks = range(2, 15)

obj_values = []
for k in ks:
    km = cluster.KMeans(n_clusters=k, n_init='auto')
    km.fit(X)
    obj_values.append(km.inertia_)

fig, ax = plt.subplots()
ax.plot(ks, obj_values, '-ob')
ax.set_xlabel('k')
ax.set_ylabel('inertia')
plt.show()
```



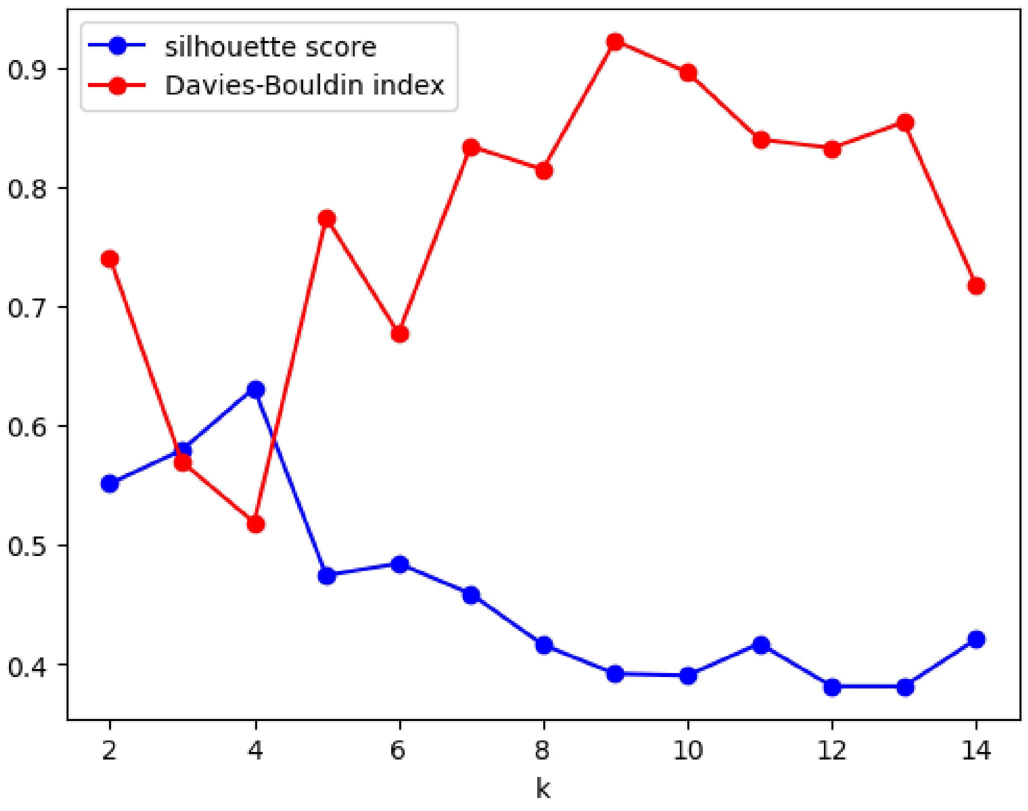
The elbow method here suggests $k = 4$.

An alternative to the elbow method is to consider quality measures for clusterings for different k. Scikit-Learn provides implementations for the silhouette score (`silhouette_score`[578]) and for the Davies-Bouldin index (`davies_bouldin_score`[579]) and for many other clustering metrics[580].

```
import sklearn.metrics as metrics
```

```
ks = range(2, 15)

sil = []
db = []
for k in ks:
    km = cluster.KMeans(n_clusters=k, n_init='auto')
    km.fit(X)
    sil.append(metrics.silhouette_score(X, km.labels_))
    db.append(metrics.davies_bouldin_score(X, km.labels_))

fig, ax = plt.subplots()
ax.plot(ks, sil, '-ob', label='silhouette score')
ax.plot(ks, db, '-or', label='Davies-Bouldin index')
ax.set_xlabel('k')
ax.legend()
plt.show()
```

[578] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html
[579] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.davies_bouldin_score.html
[580] https://scikit-learn.org/stable/modules/classes.html#clustering-metrics



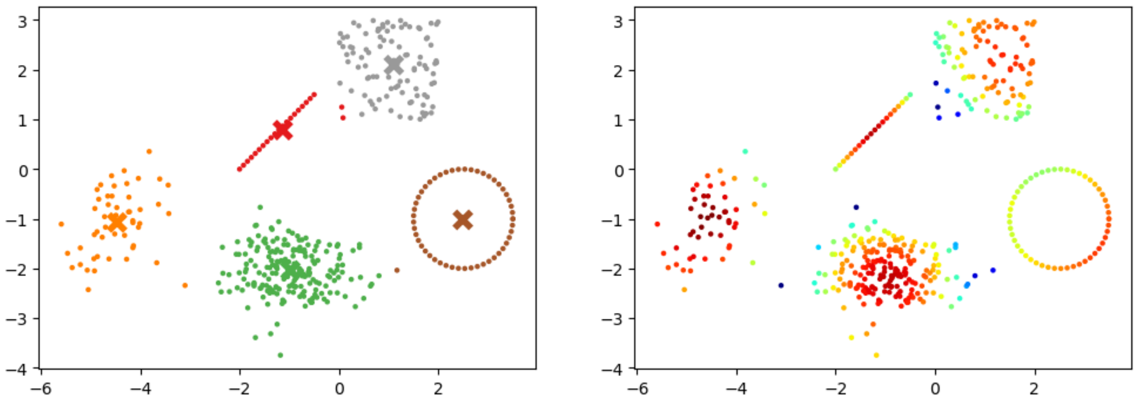
Both scores suggest $k = 4$.

34.8 Per Sample Silhouette Score

To judge on the quality of a clustering we may also look at each sample's silhouette score. This way we can identify missclusterings or samples with uncertain cluster assignment. Scikit-Learn provides `silhouette_samples`[581] for this purpose.

```
k = 5

km = cluster.KMeans(n_clusters=k, n_init='auto')
km.fit(X)
centers = km.cluster_centers_
clusters = km.labels_
sil = metrics.silhouette_samples(X, km.labels_)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))
ax1.scatter(X[:, 0], X[:, 1], s=5, c=clusters, cmap='Set1')
ax1.scatter(centers[:, 0], centers[:, 1], s=100, c=range(0, k), cmap='Set1',␣
↪marker='x', linewidth=4)
ax1.axis('equal')
ax2.scatter(X[:, 0], X[:, 1], s=5, c=sil, cmap='jet')
ax2.axis('equal')
plt.show()
```

[581] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_samples.html



Red points (high silhouette score) clearly belong to their cluster, blue points (low silhouette score) also could belong to a different or to no cluster.

34.9 k-Means for Very Large Data Sets

k-means requires as many distance computations per cluster and per iteration as there are samples in the training data set. To reduce computation time for large data sets we may

- use only a subset of the training data in each iteration (subsets change from iteration to iteration) and
- update center points sample by sample instead of cluster by cluster.

Working with subsets obviously reduces the amount of distance calculations. Depending on the randomly chosen subsets, cluster centers may change very much from iteration to iteration. To prevent such hopping, updates are calculated sample by sample, resulting in slightly different update results. A side effect of the sample-by-sample approach is a more efficient implementation of the update step. Details are given in Web-Scale K-Means Clustering[582].

Scikit-Learn implements this approach as `MiniBatchKMeans`[583].

34.10 When to Use k-Means?

k-means is very fast (at least faster than most other clustering algorithms). But it has some drawbacks one has to be aware of.

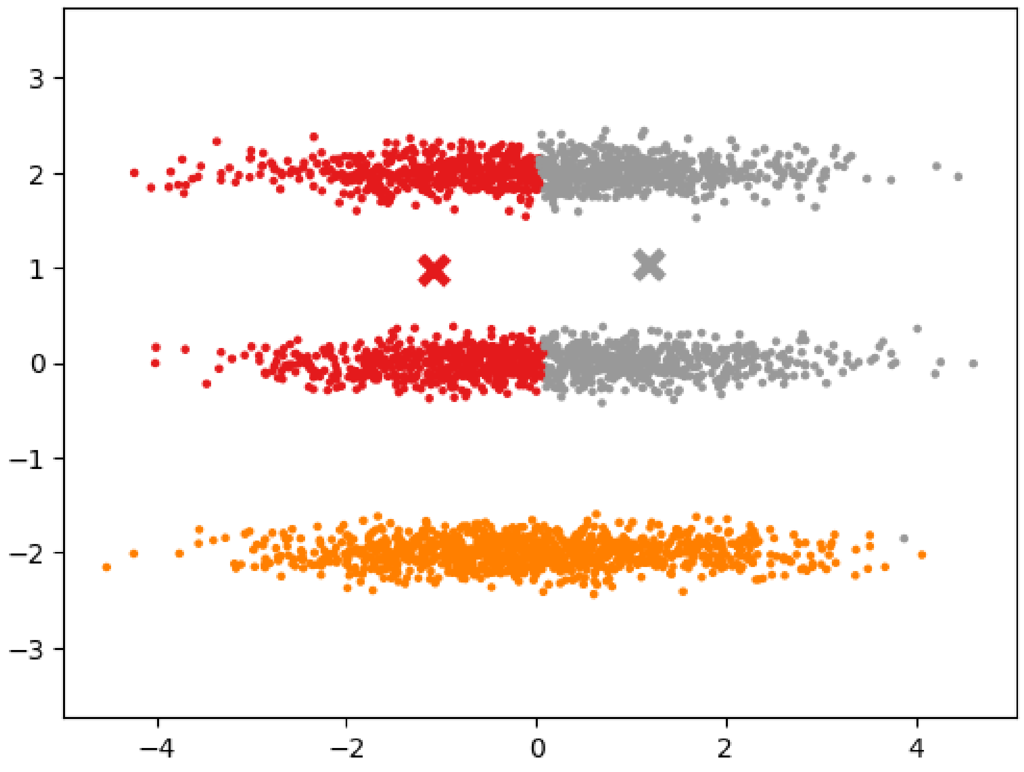
For very high dimensional data (thousands of features, images for instance) distances between samples do not carry useful information because all distances are almost identical (cf. discussion of curse of dimensionality in *Introductory Example (k-NN)* (page 317)). Thus, k-means wont work. Resulting clusters will look like samples were assigned to clusters uniformly at random. Dimension reduction techniques (e.g. PCA) should be applied before trying k-means.

k-means prefers convex and sphere shaped clusters. Proper scaling of the data may improve results drastically, especially if features have very different numerical ranges. Alternatively the used distance measure has to be adapted to the data under consideration (weighted Euclidean distance, for instance).

```
n1, n2, n3 = 1000, 1000, 1000
n = n1 + n2 + n3

X1 = rng.multivariate_normal((0, -2), ((2, 0), (0, 0.02)), n1)
X2 = rng.multivariate_normal((0, 0), ((2, 0), (0, 0.02)), n2)
X3 = rng.multivariate_normal((0, 2), ((2, 0), (0, 0.02)), n3)

X = np.concatenate((X1, X2, X3))
```

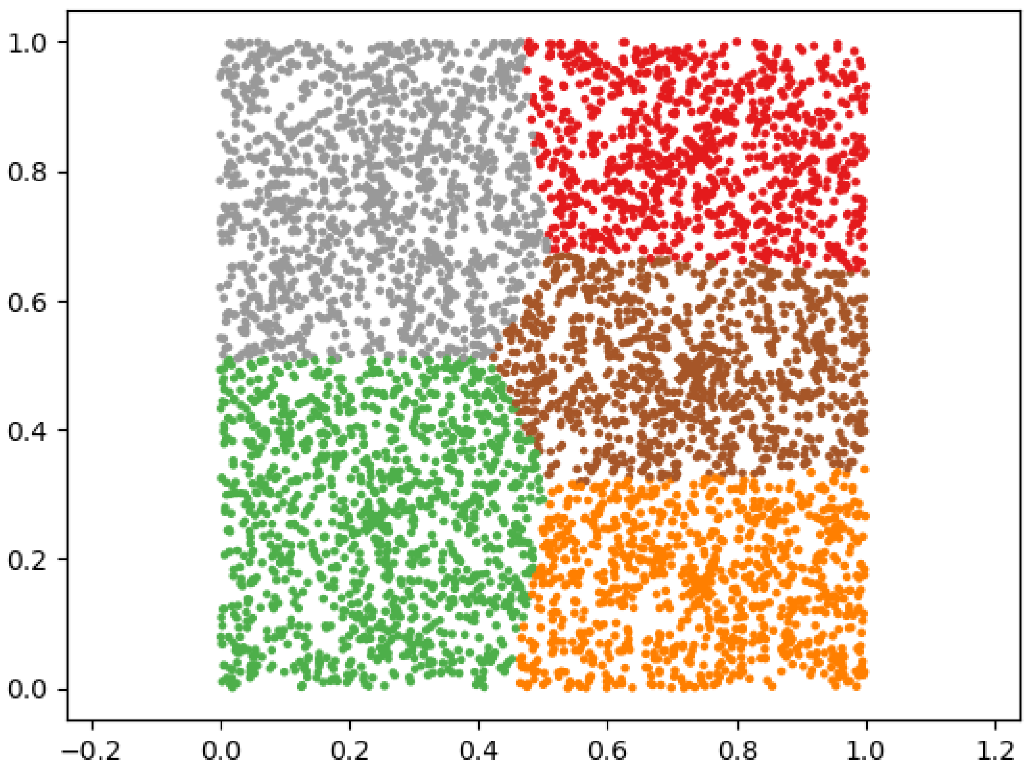
[582] https://web.archive.org/web/20230210073106/https://www.eecs.tufts.edu/~dsculley/papers/fastkmeans.pdf
[583] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MiniBatchKMeans.html

```
k = 3

km = cluster.KMeans(n_clusters=k, n_init='auto')
km.fit(X)
centers = km.cluster_centers_
clusters = km.labels_

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c=clusters, cmap='Set1')
ax.scatter(centers[:, 0], centers[:, 1], s=100, c=range(0, k), cmap='Set1',␣
↪marker='x', linewidth=4)
ax.axis('equal')
plt.show()
```



Due to its distance-to-center based nature k-means prefers clusters with similar spatial extent.

```
n = 5000
X = rng.uniform((0, 0), (1, 1), (n, 2))

k = 5

km = cluster.KMeans(n_clusters=k, n_init='auto')
km.fit(X)
centers = km.cluster_centers_
clusters = km.labels_

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c=clusters, cmap='Set1')
ax.scatter(centers[:, 0], centers[:, 1], s=100, c=range(0, k), cmap='Set1',␣
↪marker='x', linewidth=4)
ax.axis('equal')
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2

CHAPTER

THIRTYFIVE

HIERARCHICAL CLUSTERING

Hierarchical clustering algorithms not only yield one clustering but a series of clusterings. There are two approaches for generating a series of clusters:

- *coarse to fine* or *divisive* (the first clustering consists of only one cluster (the whole training data set) and the last contains as many clusters as there are samples),
- *fine to coarse* or *agglomerative* (the first clustering contains as many clusters as samples and the last clustering has only one large cluster).

In the first case the next clustering results from splitting a cluster in the previous clustering by some rule. In the second case two clusters are selected by some rule and than joined to form only one cluster. Both variants result in a nested sequence of clusters, usually (not always) with increasing intercluster dissimilarity from fine to coarse.

Hierarchical clustering methods do not imply canonical prediction routines (in contrast to k-means). They only assign labels to the training data. To add new data points to existing clusters kNN or some other supervised learning technique has to be used.

Related projects:

- *Chinese Celadons* (page 987)
  - *Hierarchical Clustering* (page 987)

35.1 Dendrograms

The aim of hierarchical clustering is not only to find a good clustering, but to better understand structures in the data set. A *dendrogram* is a graphical representation of a sequence of nested clusterings. On the one hand, a dendrogram shows relations between clusters of different clusterings. On the other hand, it provides information about dissimilarity of clusters within one clustering.

A dendrogram is a binary (that is, two children per parent node) tree. Each node represents a subset (cluster) of the data set. The whole data set is the root node and there are as many leaves as there are samples in the data set. Each intermediate node represents the union of its two children. The height of a node in the tree is related to the value of a dissimilarity measure between clusters which lead to the split or join operation.

Dendrograms can be used to determine the number of clusters in a data set. The wider the height gap between two nodes, the better the clustering (with respect to the chosen distance). Here 'better' means that the clustering corresponding to a wide gap is very stable with respect to the merging criterion, that is, contains no subclusters with low dissimilarity. See below for an example.

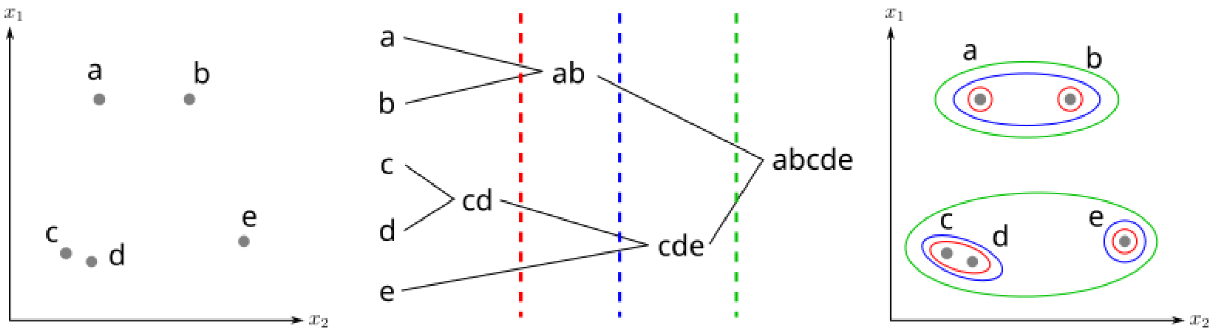


Fig. 35.1: A dendrogram and corresponding clusters.

35.2 Divisive Clustering

Divisive clustering is rarely used in practice, because good splitting stategies are computationally expensive. One possible approach is to find the cluster with highest intracluster dissimilarity and then use k-means with $k = 2$ for splitting the cluster. Depending on the chosen dissimilarity measure dissimilarity values at subsequent splits may be nondecreasing. Thus, height of nodes in the dendrogram cannot encode dissimilarity.

35.3 Agglomerative Clustering

For agglomerative clustering we have to choose a point-to-point distance d (Euclidean distance, for instance) and a distance D between two disjoint sets. Next to some relatively uncommon variants there are three major notions for distances between disjoint sets:

- minimum distance of points (*single linkage clustering*):

$$D(C,\tilde{C}) := \min_{x\in C,\tilde{x}\in\tilde{C}} d(x,\tilde{x}),$$

- maximum distance of points (*complete linkage clustering*):

$$D(C,\tilde{C}) := \max_{x\in C,\tilde{x}\in\tilde{C}} d(x,\tilde{x}),$$

- average distance of points (*average linkage clustering*):

$$D(C,\tilde{C}) := \frac{1}{|C|\,|\tilde{C}|}\sum_{x\in C}\sum_{\tilde{x}\in\tilde{C}} d(x,\tilde{x}).$$

Starting with the finest clustering the following steps are repeated until there is only one large cluster containing the whole data set:

- Calculate the distance D for each pair of clusters.
- Join the two clusters with smallest distance.

Clustering results for different distance measures may differ significantly. Single linkage clustering suffers from *chaining*. That is, several (for human eyes) clearly separated clusters are joined into one because they touch at one point. An advantage of single linkage clustering is that it finds clusters of arbitrary shape and does not prefer convex clusters. Complete linkage prefers compact sphere shaped clusters. Average linkage is a compromise between both extremes. Have a look at the plots of the linkage example in Scikit-Learn's documentaion[584].

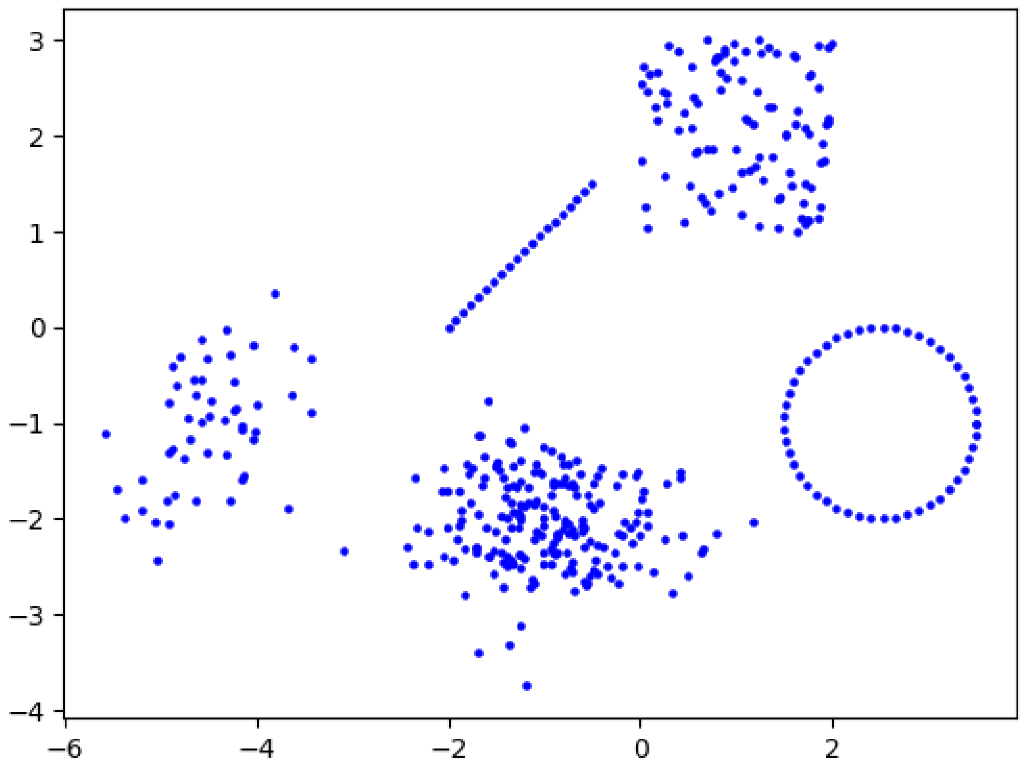
[584] https://scikit-learn.org/stable/auto_examples/cluster/plot_linkage_comparison.html

35.4 Hierarchical Clustering with Scikit-Learn

Scikit-Learn only supports agglomerative clustering via `AgglomerativeClustering`[585] class. We either have to provide the number of clusters `n_clusters` or the distance `distance_threshold` at which to cut the dendrogram.

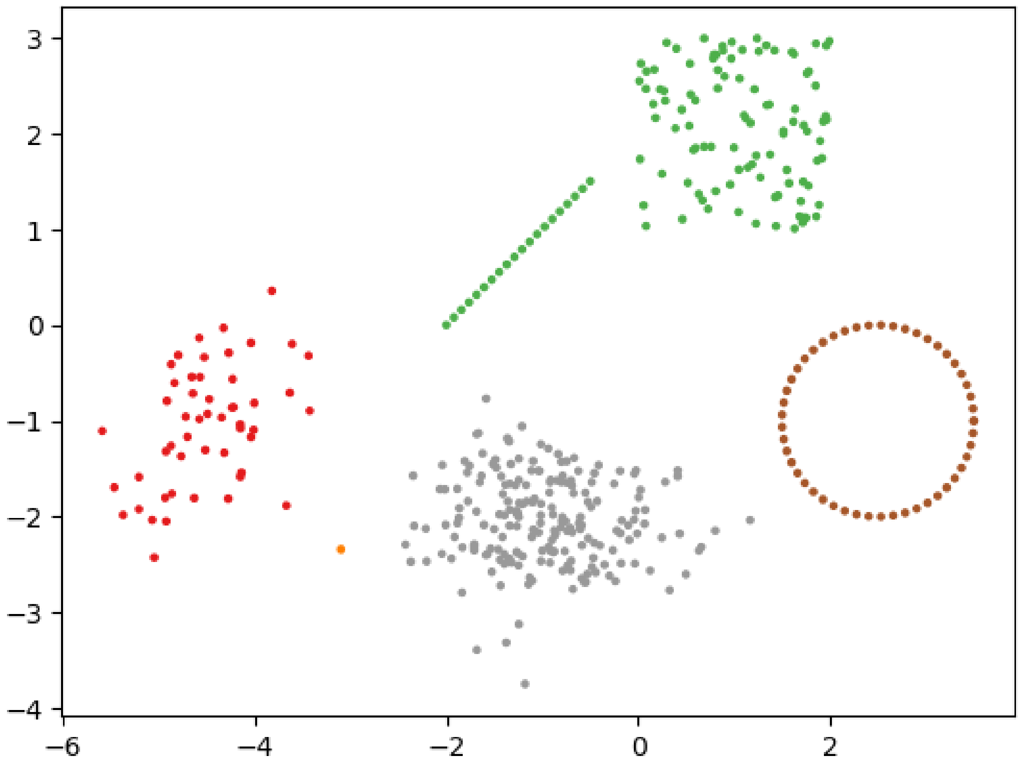
```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.cluster as cluster

rng = np.random.default_rng(0)
```

```
# 5 clusters in 2 dimensions

n1, n2, n3, n4, n5 = 100, 200, 20, 50, 50
n = n1 + n2 + n3 + n4 + n5

X1 = rng.uniform((0, 1), (2, 3), (n1, 2))
X2 = rng.multivariate_normal((-1, -2), ((0.5, 0), (0, 0.2)), n2)
X3 = np.linspace(-1, 0.5, n3).reshape(-1, 1) * np.ones((1, 2)) + np.array([[-1,␣
↪1]])
X4 = rng.multivariate_normal((-4.5, -1), ((0.2, 0.1), (0.1, 0.4)), n4)
phi = np.linspace(0, 2 * np.pi, n5).reshape(-1, 1)
X5 = np.array([[2.5, -1]]) + np.concatenate((np.cos(phi), np.sin(phi)), axis=1)

X = np.concatenate((X1, X2, X3, X4, X5))

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c='b')
ax.axis('equal')
plt.show()
```



[585] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

```
ac = cluster.AgglomerativeClustering(5, linkage='single')
ac.fit(X)

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c=ac.labels_, cmap='Set1')
ax.axis('equal')
plt.show()
```



Neither Scikit-Learn, nor Matplotlib, nor Seaborn support plotting dendrograms directly. Only SciPy[586] has a `dendrogram`[587] function for plotting. SciPy also has the `linkage`[588] function for agglomerative clustering. But we would like to use Scikit-Learn for clustering although SciPy shall plot the dendrogram. Thus, we have to adapt Scikit-Learn's output to the needs of SciPy.

```
import scipy.cluster.hierarchy as scipy_ch
```

```
def plot_dendrogram(ac, ax, max_nodes=10, color_threshold=0.5):

    n = ac.labels_.shape[0]     # number of samples

    data = np.empty((n - 1, 4))
    data[:, 0:2] = ac.children_
    data[:, 2] = ac.distances_

    # get number of samples per node
    for i in range(0, n - 1):     # visit each node
        c = 0

        # add samples from left child
        if ac.children_[i, 0] < n:     # child is leaf
```

(continues on next page)

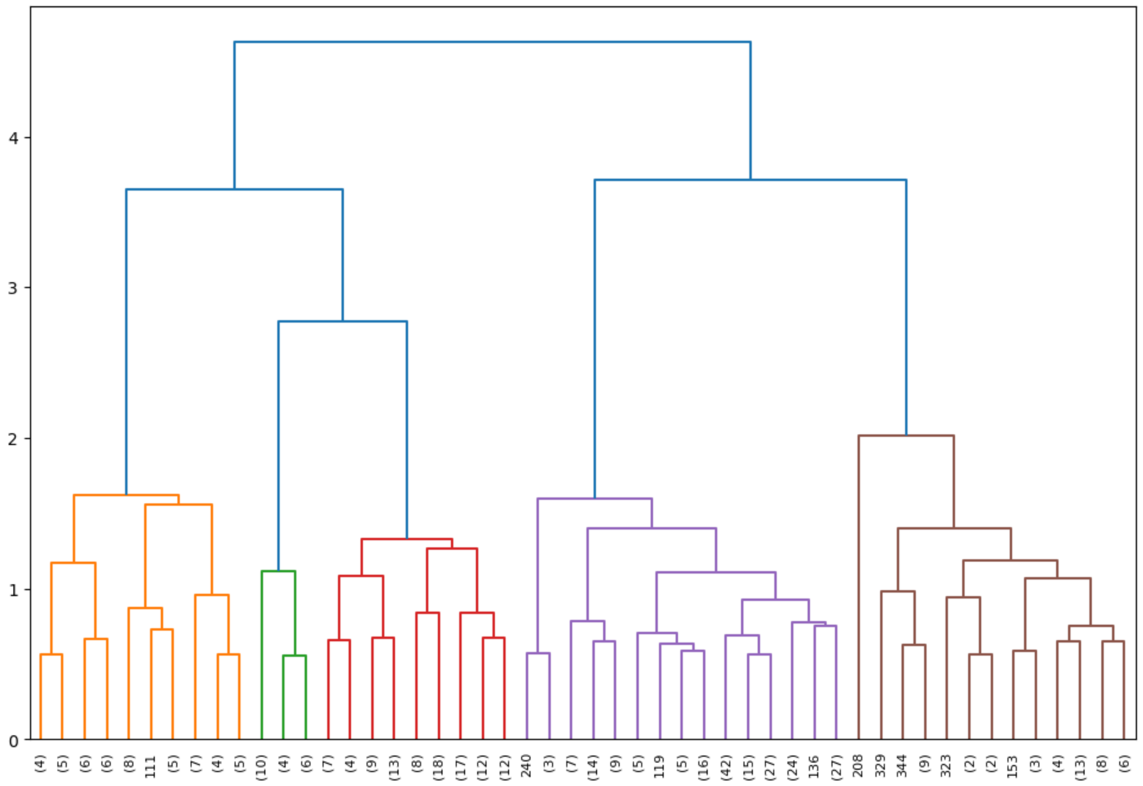
586 https://docs.scipy.org/doc/scipy/index.html
587 https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html
588 https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html

(continued from previous page)

```
            c = c + 1
        else:
            c = c + data[ac.children_[i, 0] - n, 3]

        # add samples from right child
        if ac.children_[i, 1] < n:     # child is leaf
            c = c + 1
        else:
            c = c + data[ac.children_[i, 1] - n, 3]
        data[i, 3] = c

    scipy_ch.dendrogram(data, ax=ax, truncate_mode='lastp', p=max_nodes, color_
↪threshold=color_threshold)
```

Note that the `n_clusters` parameter of `AgglomerativeClustering` does not matter, because we are interested in the dendrogram, not in a particular clustering.

```
ac = cluster.AgglomerativeClustering(linkage='average', compute_distances=True)
ac.fit(X)

fig, ax = plt.subplots(figsize=(12, 8))
plot_dendrogram(ac, ax, 50, 2.5)
plt.show()
```



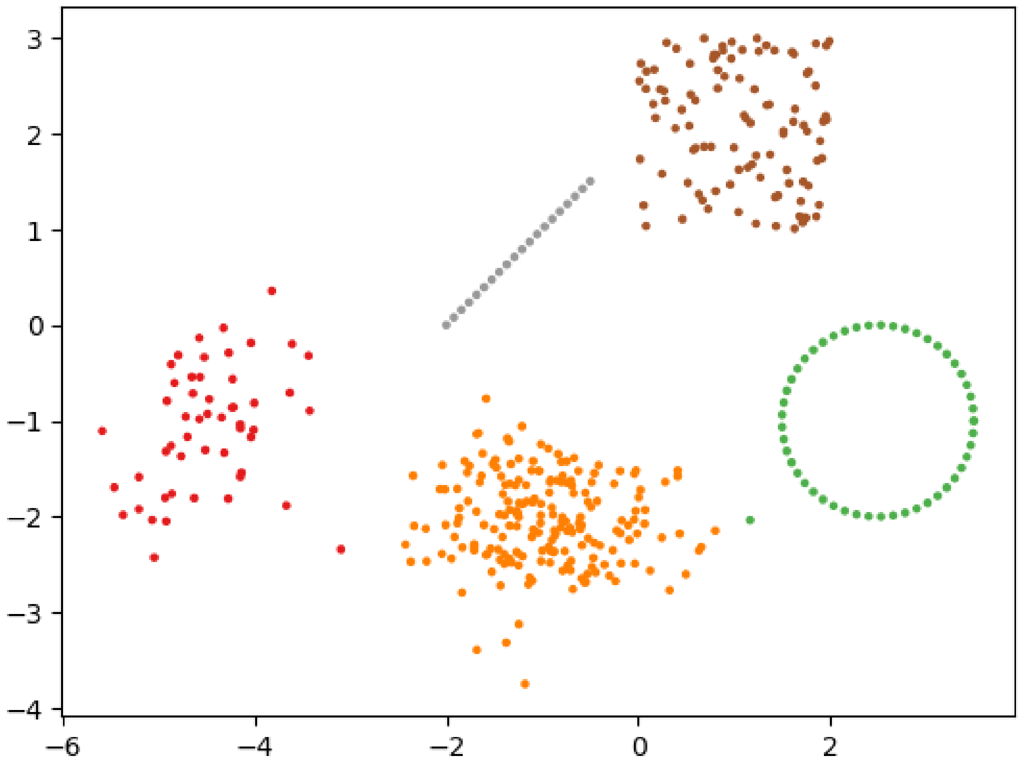
From the dendrogram we see that the number of clusters in the data is 2, 4 or 5.

```
ac = cluster.AgglomerativeClustering(n_clusters=5, linkage='average')
ac.fit(X)

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c=ac.labels_, cmap='Set1')
ax.axis('equal')
```

(continues on next page)

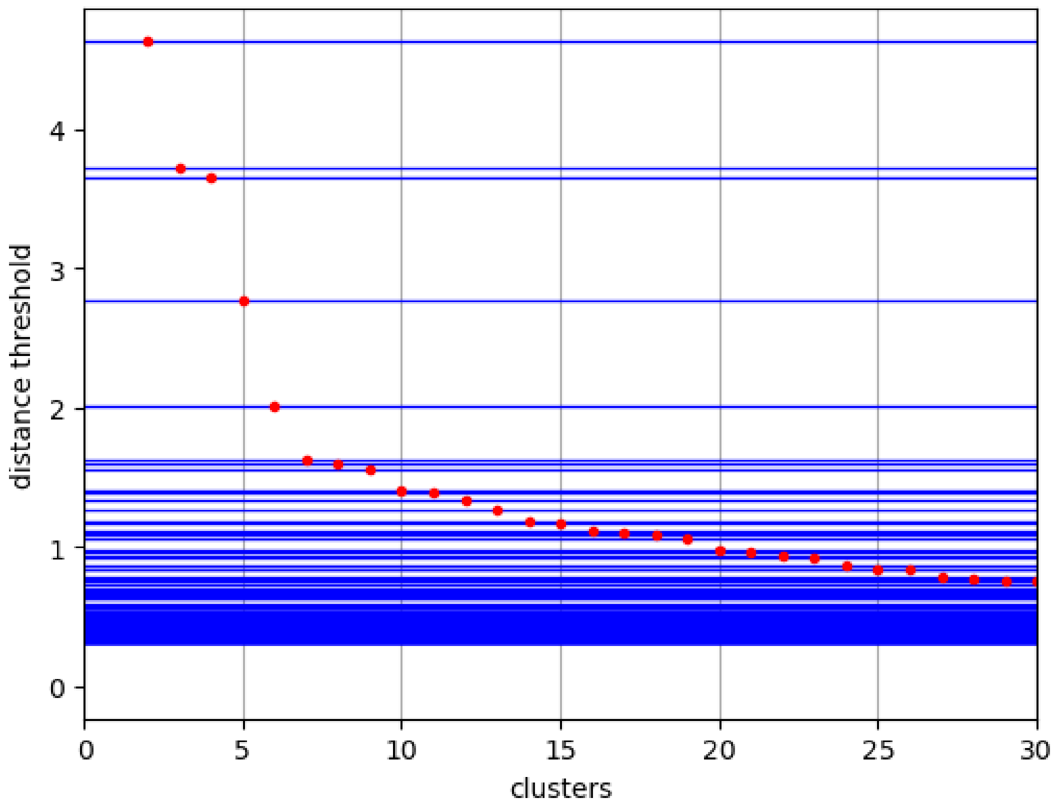
(continued from previous page)

```
plt.show()
```



If we only are interested in the number of clusters we may plot the number of clusters versus the distance threshold and look for wide vertical gaps.

```
ac = cluster.AgglomerativeClustering(linkage='average', compute_distances=True)
ac.fit(X)

fig, ax = plt.subplots()

dists = ac.distances_
for i in range(dists.size - 100, dists.size):
    ax.plot([0, dists.size], [dists[i], dists[i]], '-b', linewidth=1)
ax.plot(range(dists.size + 1, 1, -1), dists, 'or', markersize=3)

ax.set_xlim(0, 30)
ax.set_xlabel('clusters')
ax.set_ylabel('distance threshold')
ax.grid(axis='x')

plt.show()
```


distance threshold
clusters
0
1
2
3
4
0
5
10
15
20
25
30

CHAPTER

THIRTYSIX

DENSITY-BASED CLUSTERING

Density-based clustering aims at finding connected regions of closely spaced points in a data set. The basic idea is to mark *core points* (points with many surrounding points) and look for clusters in the set of core points. A distance measure is only used to determine a neighborhood for each point. Thus, the choice of a concrete distance measure is not as important as for centroid-based methods or hierarchical clustering.

There exist many ways to fill in the details of the approach. The most widely used density-based clustering algorithms are DBSCAN and OPTICS. Both only assign cluster labels to the training data but do not imply a canonical prediction routine. Both algorithms may yield clusters of arbitrary shape.

Another density-based clustering method, not discussed here, is mean shift[589].

Related projects:

36.1 DBSCAN

The DBSCAN (density-based spatial clustering of applications with noise) algorithm takes two parameters $\varepsilon > 0$ and $N \in \mathbb{N}$, which together specify what a *core point* is: a point x from the training data set is a core point if the ball of radius ε centered at x contains at least N points of the data set (including x). We may say that core points are points with dense neighborhood.

In the context of DBSCAN a cluster is a subset S of the data set with the following properties:

- S contains at least one core point $\bar{x}_0$.
- For each point x in S there is a finite sequence of core points $\bar{x}_1, \ldots, \bar{x}_k$ such that $\bar{x}_l$ belongs to the ε-neighborhood of $\bar{x}_{l-1}$ for $l = 1, \ldots, k$ and x belongs to the ε-neighborhood of $\bar{x}_k$.

Each cluster contains core points and may contain non-core points (points with few neighbors). Non-core points can be regarded as the cluster's edge. From the sequence property above one immediately sees that each core point of a cluster may take the role of $\bar{x}_0$.

There may exist points which do not belong to any cluster. Such points are regarded as noise or outliers.

DBSCAN starts with some point of the data set. If it's a non-core point, it is marked as visited and a new point is chosen. If it is a core point, a new cluster is started. All points from the ε-neighborhood are added to the cluster. Then all neighborhoods of core points in the starting point's neighborhood are added, and so on until there are no more reachable core points. Then a new unvisited point is chosen and the cluster discovery starts again from this new point (if it is a core point). If all points have been visited, the points not assigned to a cluster are interpreted as outliers.

[589] https://en.wikipedia.org/wiki/Mean-shift

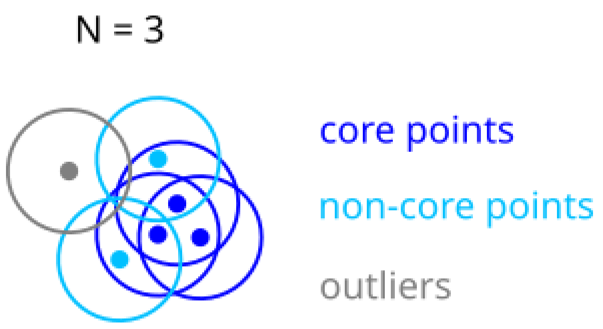


Fig. 36.1: DBSCAN classifies points as core points, non-core points, and outliers.

The computationally expensive part is to find all points in the ε-neighborhood of a given point. There exist efficient implementations for such *range queries* for data sets stored in well structured data bases.

A major advantage of DBSCAN is that we do not have to estimate the number of clusters in advance. Also results do not depend on (random) initializations. Although assignment of non-core points to clusters depends on the processing order of the points, clustering of core points is always the same.

Parameters ε and N may be hard to choose. But they have a clear interpretation. The smaller ε and the higher N the closer points have to be to form a cluster. Domain knowledge may help to choose these parameters. Another drawback is that there is only one set of parameters for all clusters. Consequently all clusters should show comparable density.

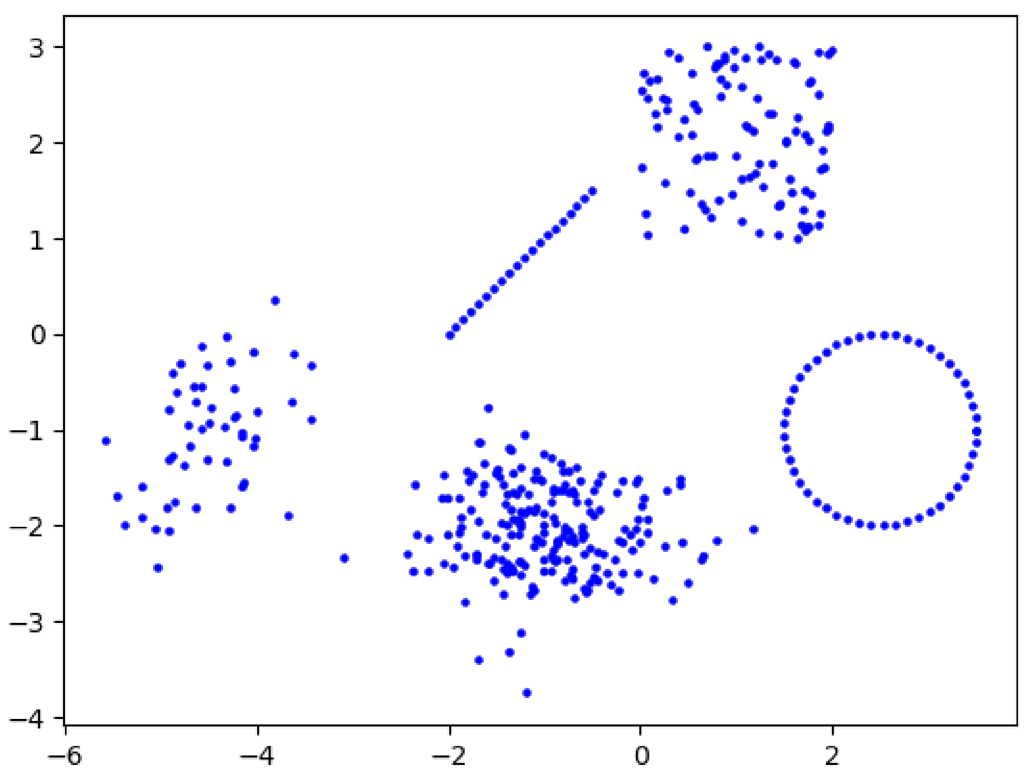
36.1.1 DBSCAN with Scikit-Learn

Scikit-Learn provides a DBSCAN[590] class in its `cluster` module. Relevant parameters are `eps` (our ε) and `min_samples` (our N).

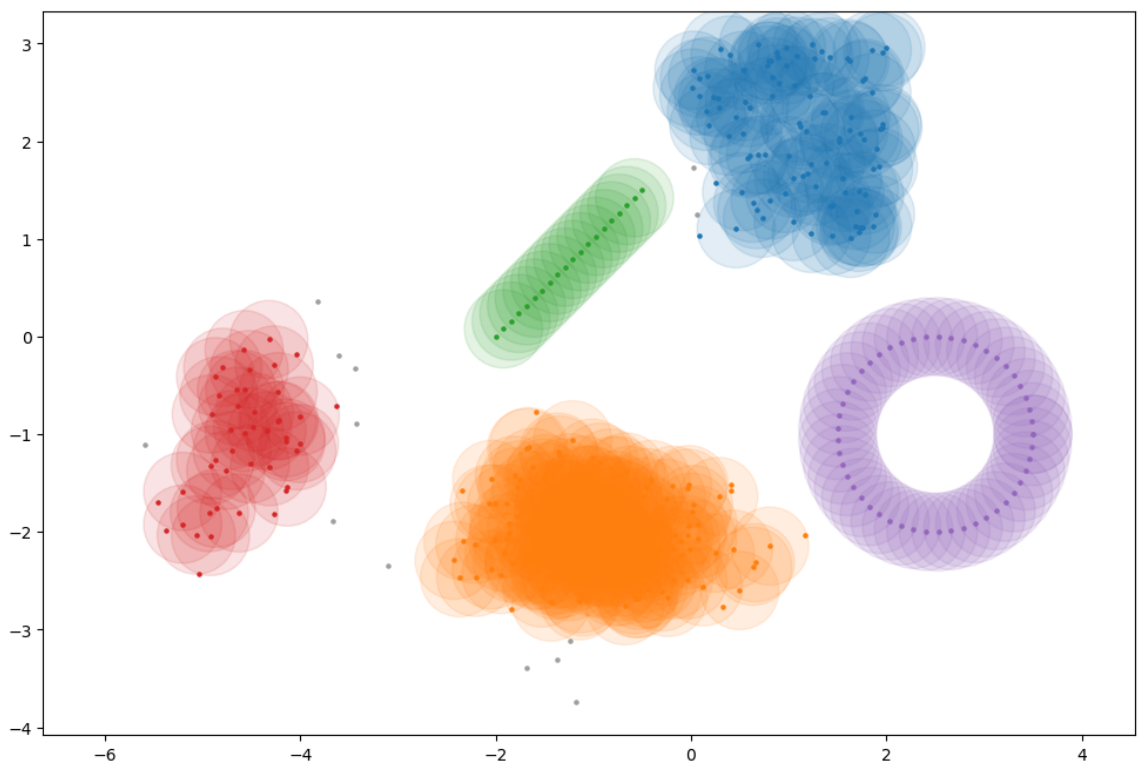
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import sklearn.cluster as cluster

rng = np.random.default_rng(0)
```

```
# 5 clusters in 2 dimensions

n1, n2, n3, n4, n5 = 100, 200, 20, 50, 50
n = n1 + n2 + n3 + n4 + n5

X1 = rng.uniform((0, 1), (2, 3), (n1, 2))
X2 = rng.multivariate_normal((-1, -2), ((0.5, 0), (0, 0.2)), n2)
X3 = np.linspace(-1, 0.5, n3).reshape(-1, 1) * np.ones((1, 2)) + np.array([[-1,␣
↪1]])
X4 = rng.multivariate_normal((-4.5, -1), ((0.2, 0.1), (0.1, 0.4)), n4)
phi = np.linspace(0, 2 * np.pi, n5).reshape(-1, 1)
X5 = np.array([[2.5, -1]]) + np.concatenate((np.cos(phi), np.sin(phi)), axis=1)

X = np.concatenate((X1, X2, X3, X4, X5))

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c='b')
ax.axis('equal')
plt.show()
```

[590] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html



```python
eps = 0.4
N = 5

dbs = cluster.DBSCAN(eps=eps, min_samples=N)
dbs.fit(X)

colors = list(mpl.colors.TABLEAU_COLORS.values())

fig, ax = plt.subplots(figsize=(12, 8))

# plot neighborhoods of core points
for i in dbs.core_sample_indices_:
    color = colors[dbs.labels_[i] % len(colors)] + '20'
    ax.add_artist(mpl.patches.Circle(X[i, :], eps, color=color))

# plot points
for label in np.unique(dbs.labels_):
    if label == -1:
        color = '#A0A0A0'
    else:
        color = colors[label % len(colors)]
    mask = dbs.labels_ == label
    ax.scatter(X[mask, 0], X[mask, 1], c=color, s=5)

ax.axis('equal')
plt.show()
```



36.2 OPTICS

OPTICS (ordering points to identify the clustering structure) is an extension of DBSCAN, which automatically chooses ε and allows for cluster-dependent ε. Instead of ε OPTICS requires a parameter $\varepsilon_{\max}$ describing the maximum value to consider when choosing ε. This parameter has to be chosen large enough to not miss a cluster but there is no upper bound. The larger $\varepsilon_{\max}$ the more computation time is required while clustering results remain the same. Thus, choosing a good $\varepsilon_{\max}$ is only important for large scale data sets.

OPTICS differs from DBSCAN as follows:

- Neighborhood radius ε depends on the point x under consideration. It is chosen as small as possible but large enough to have N points (including x) in the $\varepsilon(x)$-neighborhood. In this sense every point is a core point, at least if $\varepsilon_{\max}$ is large enough (which we assume below).
- The direct output of the OPTICS algorithm is not a clustering. Instead, to each processed point a numerical value, the *reachability distance*, is assigned. These values imply a linear ordering of all processed points. From this ordering a clustering can be deduced by several different methods (see below).

36.2.1 Basic Algorithm

During runtime of OPTICS algorithm each point is in one of three states: untouched or discovered or processed. At the beginning all points are untouched, at the end all points will be processed (if $\varepsilon_{\max}$ is large enough).

OPTICS, like DBSCAN, starts at an arbitrary point x. For each point $\tilde{x}$ in the $\varepsilon_{\max}$-neighborhood of x then the reachability distance is computed as

$$\max\{\varepsilon(x), |x-\tilde{x}|\},$$

all points but x are appended to the list of discovered (but unprocessed) points and x is marked as processed. Now the following steps are repeated until no discovered but unprocessed points remain:

- The point with lowest reachability distance is chosen from the list of discovered points and marked as processed.
- For all unprocessed points in its $\varepsilon_{\max}$-neighborhood reachability distances with respect to the point are computed.

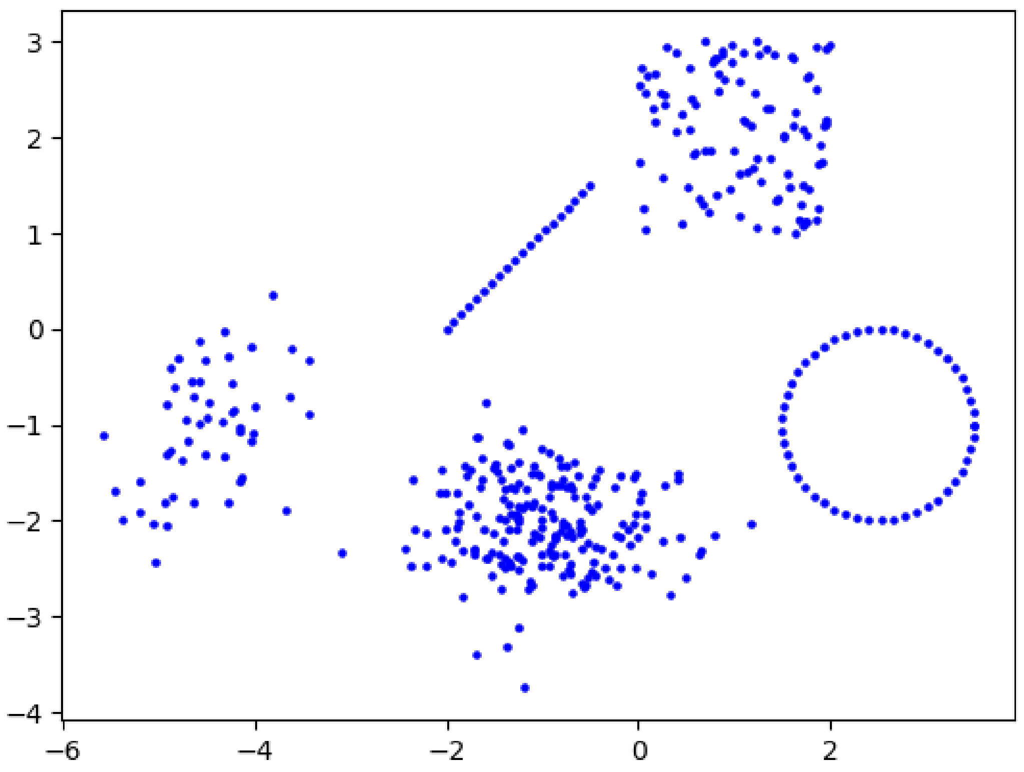
- If such an unprocessed neighboring point already has been discovered before, the smaller one of old and new reachability distance is chosen. If a neighboring point has been discovered for the first time, it is added to the list of discovered points.

36.2.2 Implementation from Scratch

Note that efficient implementation of OPTICS algorithm requires knowledge of advanced data structures. For understanding the basic principles here we only provide an inefficient implementation avoiding any use of advanced data structures. Further we set $\varepsilon_{\text{max}} = \infty$, that is, instead of (large) neighborhoods we always discover the whole data set. Consequently, there will be no untouched points after processing the starting point.

```python
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.default_rng(0)
```

```python
# 5 clusters in 2 dimensions

n1, n2, n3, n4, n5 = 100, 200, 20, 50, 50
n = n1 + n2 + n3 + n4 + n5

X1 = rng.uniform((0, 1), (2, 3), (n1, 2))
X2 = rng.multivariate_normal((-1, -2), ((0.5, 0), (0, 0.2)), n2)
X3 = np.linspace(-1, 0.5, n3).reshape(-1, 1) * np.ones((1, 2)) + np.array([[-1,␣
↪1]])
X4 = rng.multivariate_normal((-4.5, -1), ((0.2, 0.1), (0.1, 0.4)), n4)
phi = np.linspace(0, 2 * np.pi, n5).reshape(-1, 1)
X5 = np.array([[2.5, -1]]) + np.concatenate((np.cos(phi), np.sin(phi)), axis=1)

X = np.concatenate((X1, X2, X3, X4, X5))

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c='b')
ax.axis('equal')
plt.show()
```



```
N = 10
start_idx = 0

n = X.shape[0]

dists = np.empty(n)     # reachability distances
epss = np.empty(n)     # epsilons
processed = np.full(n, False)     # point already processed?
order = np.empty(n, dtype=int)     # indices in the order of processing (for␣
↪identifying clusters)
previous = np.empty(n, dtype=int)     # index of core point with shortest distance␣
↪(for visualization only)

# calculate pairwise distances
D = np.empty((n, n))
for l in range(0, n):
    D[l, l] = 0
    for ll in range(0, l):
        D[l, ll] = np.sqrt(np.sum((X[l, :] - X[ll, :]) ** 2))
        D[ll, l] = D[l, ll]

# calculate epsilon for each point
for l in range(0, n):
    epss[l] = np.sort(D[l, :])[N]

# calculate reachability distances with respect to starting point
x = X[start_idx, :]
for l in range(0, n):
    dists[l] = np.maximum(epss[start_idx], D[start_idx, l])
    previous[l] = start_idx
processed[start_idx] = True
order[0] = start_idx
previous[start_idx] = -1     # no predecessor
```

(continues on next page)

(continued from previous page)

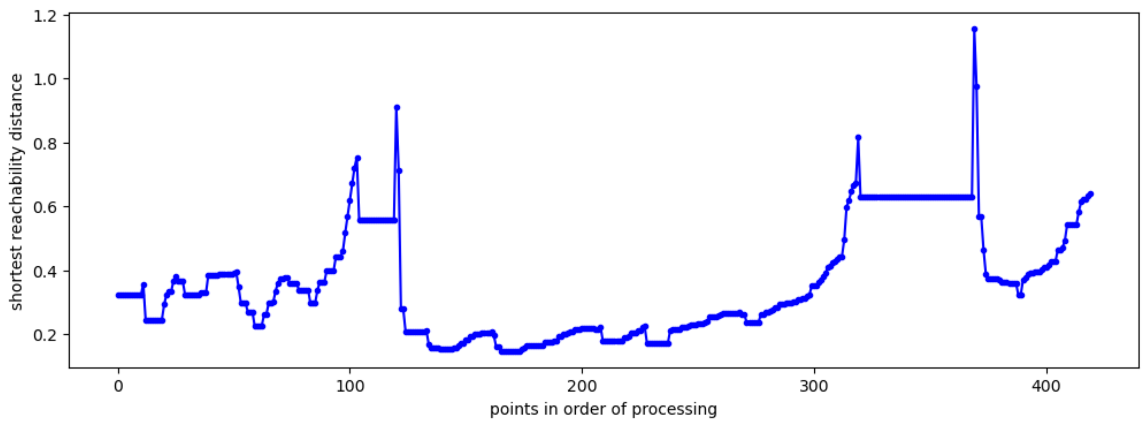
```
# process all points
for i in range(1, n):

    # get next core point (the one closest to set of processed points)
    dists_sort = np.argsort(dists)
    idx_sorted = np.flatnonzero(processed[dists_sort] == False)[0]
    idx = dists_sort[idx_sorted]
    processed[idx] = True
    order[i] = idx

    # update reachability distances
    for l in np.flatnonzero(processed == False):
        new_dist = np.maximum(epss[idx], D[idx, l])
        if new_dist < dists[l]:
            dists[l] = new_dist
            previous[l] = idx
```

36.2.3 Identifying Clusters

When OPTICS algorithm has finished we have the processing order and shortest reachability distances (SRD) of all points. The SRD of a point tells us how far away the point was from the already processed points before it got processed itself. As long as SRDs remain small while processing point by point, new points are close to previous ones. If SRD suddenly increases, then there are no more close points left. Thus, all points of a cluster have been processed and the processing of a new cluster starts.

If we plot SRD versus position in the processed chain of points, we may identify valleys as clusters. To extract a clustering from OPTICS results we have to identify valleys in the plot. Points with high SRD (mountains between valleys) can be interpreted as outliers.

```
fig, ax = plt.subplots(figsize=(12, 4))
ax.plot(dists[order] , '-ob', markersize=3)
ax.set_xlabel('points in order of processing')
ax.set_ylabel('shortest reachability distance')
plt.show()
```

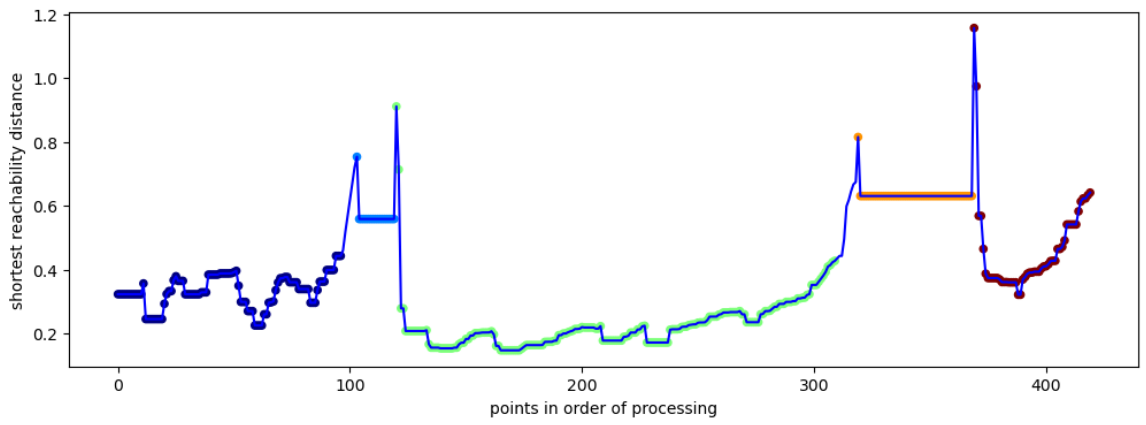


There exist several algorithms for finding valleys in the plot. One could look for steep decents or ascents or one could look for almost constant regions. Depending on the sensitivity of this valley search more or less clusters will be found. With different levels of sensitivity we would obtain a sequence of nested clusterings. In this sense, OPTICS algorithm is sometimes classified as hierarchical clustering method.

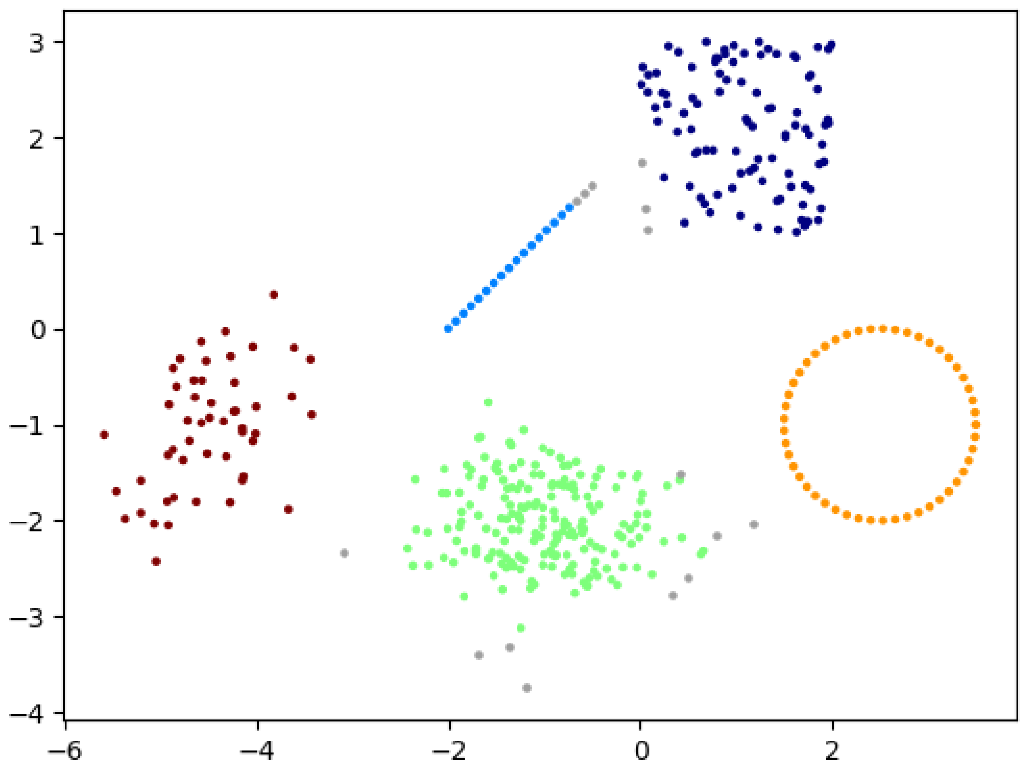
Here we do not go into the details of automatically identifying clusters.

```
labels = np.full(n, -1, dtype=int)

labels[order[0:97]] = 0
labels[order[103:120]] = 1
labels[order[120:310]] = 2
labels[order[319:369]] = 3
labels[order[369:]] = 4
```

```
mask = labels != -1

fig, ax = plt.subplots(figsize=(12, 4))
ax.plot(dists[order] , '-b', markersize=3)
ax.scatter(np.arange(0, n)[mask[order]], dists[order][mask[order]],
           s=20, c=labels[order][mask[order]], cmap='jet')
ax.set_xlabel('points in order of processing')
ax.set_ylabel('shortest reachability distance')
plt.show()
```



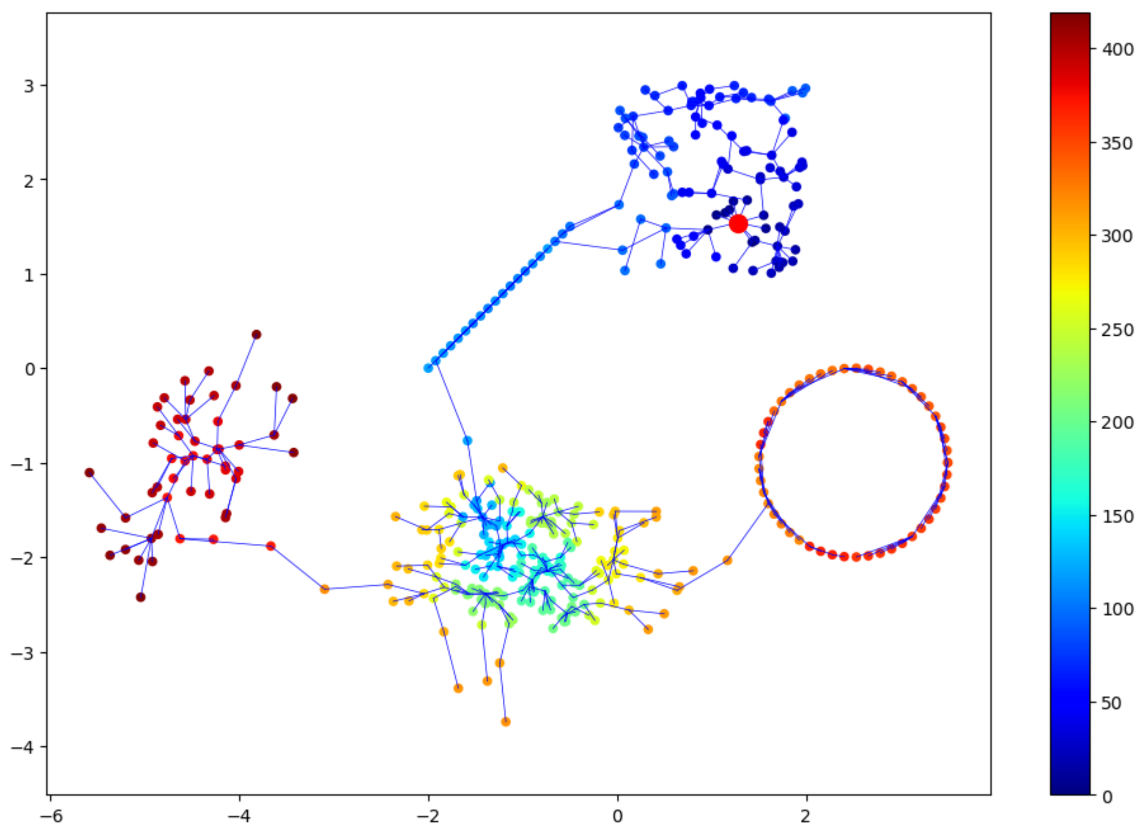
```
fig, ax = plt.subplots()
mask = labels != -1
ax.scatter(X[mask, 0], X[mask, 1], s=5, c=labels[mask], cmap='jet')
ax.scatter(X[mask == False, 0], X[mask == False, 1], s=5, c='#A0A0A0')
ax.axis('equal')
plt.show()
```



36.2.4 Visualizing Processing Order and Reachability

For getting some more insight into OPTICS algorithm we may visualize the processing order. On the one hand we could color each point by its position in the chain of processed points. On the other hand we could connect each point with its closest core point, that is, with the point used in the computation of the shortest reachability distance.

```
fig, ax = plt.subplots(figsize=(12, 8))

# connections
for l in range(0, n):
    if previous[l] != -1:
        ax.plot([X[previous[l], 0], X[l, 0]], [X[previous[l], 1], X[l, 1]], '-b',␣
↪markersize=3, linewidth=0.5)

# colored points
c = np.empty(n)
c[order] = np.arange(0, n)
plot = ax.scatter(X[:, 0], X[:, 1], s=20, c=c, cmap='jet')
fig.colorbar(plot, ax=ax)

# starting point
ax.plot(X[start_idx, 0], X[start_idx, 1],'-or', markersize=10)

ax.axis('equal')
plt.show()
```



All connections together constitute a graph which is tree shaped and spans the data set, a so called *spanning tree*. This graph may be used for further investigation of the data set, for instance, to visualize high dimensional data sets in two dimensions without loosing too much structure.

Unfortunately there seems to be no Python package supporting tree visualization with prescribed edge lengths and having a simple API. Thus, we write our own tree visualization function.

```
def plot_tree(prevs, dists, core_dists, ax, root_x=0, root_y=0):
    ''' Plot tree with prescribed edge lengths.
    Only plots the edges, not the nodes. Node positions are
    returnd as NumPy array of shape (n_samples, 2)

    prevs ... 1d array with predecessor (parent) for each node (root has -1)
    dists ... distance to predecessor
    core_dists ... distance considered as close to the node (close leaves will
                   be visualized closer than prescribed by dists)
    '''

    n = len(prevs)

    # for each node create list of children
    children = [[] for i in range(0, n)]
    for i in range(0, n):
        if prevs[i] != -1:
            children[prevs[i]].append(i)
        else:
            root = i

    # randomly permutate children of each node (for better visual impression)
    for i in range(0, n):
        if len(children[i]) > 0:
            children[i] = list(np.random.default_rng(0).permuted(children[i]))
```

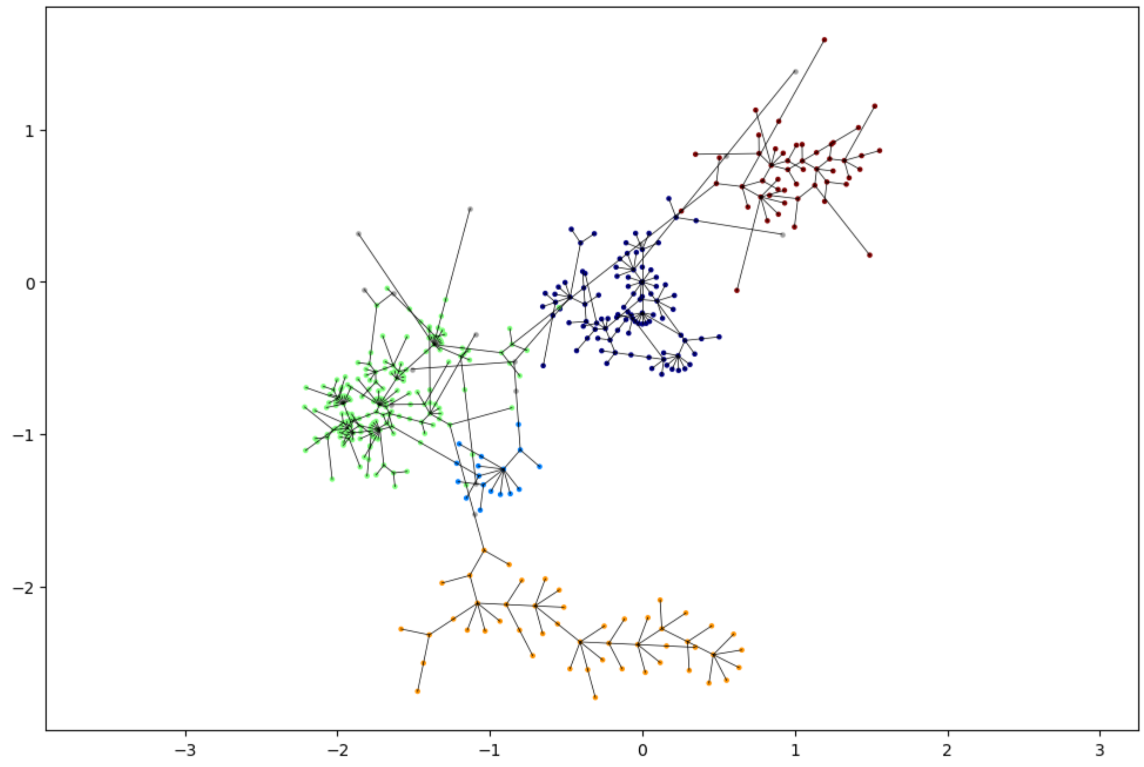
(continues on next page)

(continued from previous page)

```
    # array for node positions
    pos = np.empty((n, 2))

    # place root on stack of nodes to process
    pos[root, 0] = 0
    pos[root, 1] = 0
    stack = [root]

    # process nodes until stack is empty
    while len(stack) > 0:
        node = stack.pop()
        for i, child in enumerate(children[node]):

            # radius (distance to parent node)
            r = dists[child]
            if r <= core_dists[node]:
                r = 0.3 * core_dists[node]

            # angle
            if node == root:
                # evenly distribute children around root
                angle = 2 * np.pi / len(children[node]) * (i + 0.5)
            else:
                # evenly distribute children in a 180° region around node,
                # rotate 180° region to look away from parent of current node
                parent_angle = np.arctan2(pos[node, 1] - pos[prevs[node], 1],␣
↪pos[node, 0] - pos[prevs[node], 0])
                angle = np.pi / len(children[node]) * (i + 0.5) - 0.5 * np.pi +␣
↪parent_angle

            # child position
            pos[child, 0] = pos[node, 0] + r * np.cos(angle)
            pos[child, 1] = pos[node, 1] + r * np.sin(angle)
            stack.append(child)

            # plot edge to child
            ax.plot([pos[node, 0], pos[child, 0]], [pos[node, 1], pos[child, 1]],
↪'-k', linewidth=0.5)

    return pos
```

At the moment we only have 2d data. Thus, direct visualization of clusters is possible. Neththertheless we should have a look at corresponding tree visualization to get an idea of how to interpret it.

```
fig, ax = plt.subplots(figsize=(12, 8))

pos = plot_tree(previous, dists, epss, ax)

mask = labels != -1
ax.scatter(pos[mask, 0], pos[mask, 1], s=5, c=labels[mask], cmap='jet')
ax.scatter(pos[mask == False, 0], pos[mask == False, 1], s=5, c='#A0A0A0')

ax.axis('equal')
plt.show()
```



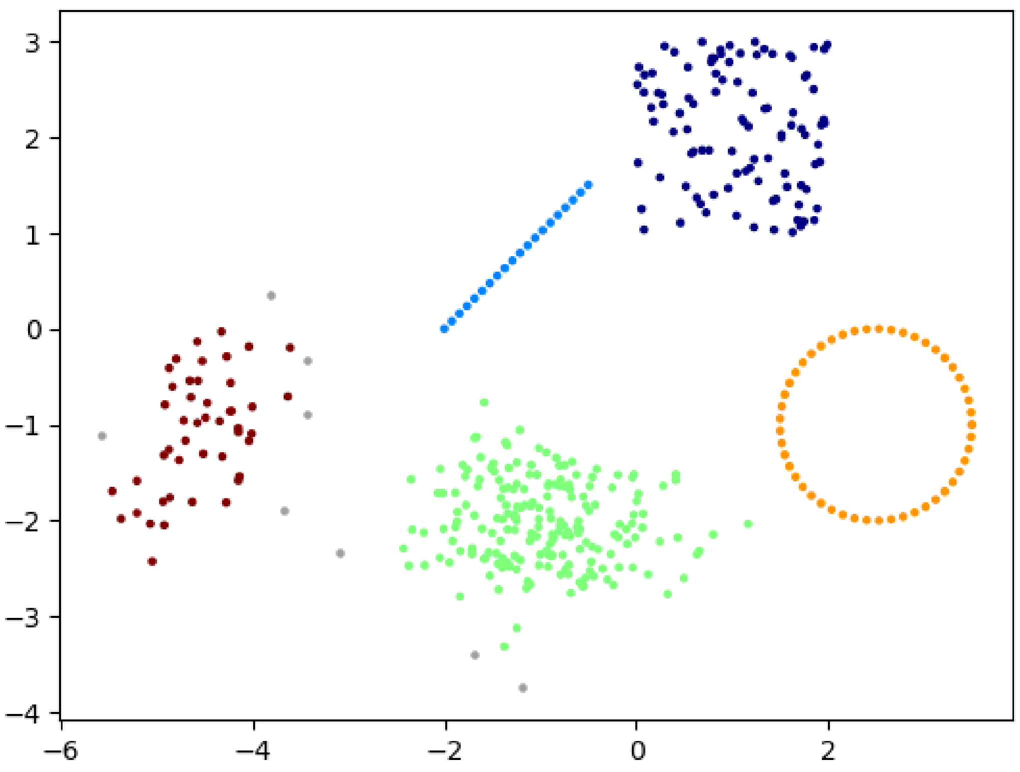
36.2.5 OPTICS Algorithm with Scikit-Learn

Scikit-Learn provides the `OPTICS`[591] class. The only relevant parameter is `min_samples` (our N). There are two cluster identification routines available. One based on DBSCAN (with sensitivity parameter `eps`) and another one with sensitivity parameter `xi`. Note, that automatic cluster identification often yields poor results.

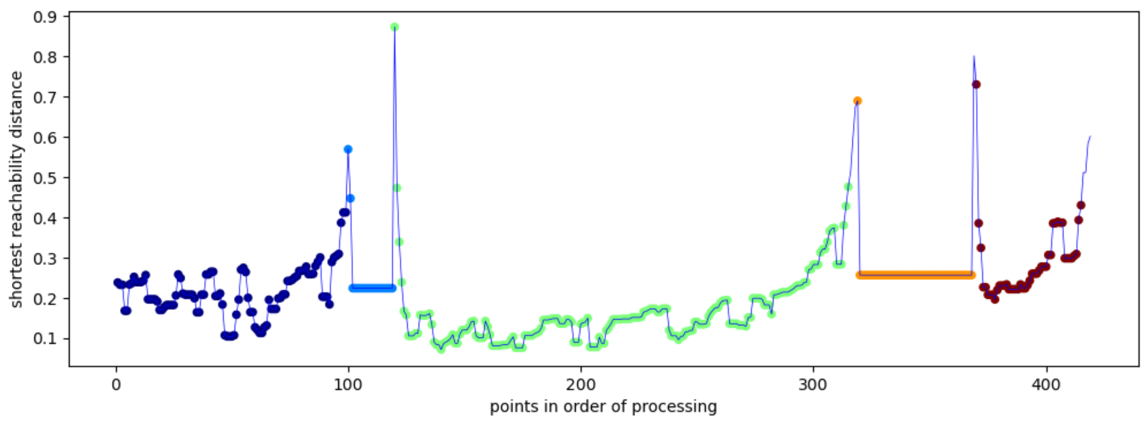
```
import sklearn.cluster as cluster
```

```
opt = cluster.OPTICS(min_samples=5, cluster_method='dbscan', eps=0.5)
opt.fit(X)

fig, ax = plt.subplots()
mask = opt.labels_ != -1
ax.scatter(X[mask, 0], X[mask, 1], s=5, c=opt.labels_[mask], cmap='jet')
ax.scatter(X[mask == False, 0], X[mask == False, 1], s=5, c='#A0A0A0')
ax.axis('equal')
plt.show()
```

[591] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.OPTICS.html



```
fig, ax = plt.subplots(figsize=(12, 4))
ax.plot(opt.reachability_[opt.ordering_] , '-b', linewidth=0.5)
ax.scatter(np.arange(0, n)[mask[opt.ordering_]], opt.reachability_[opt.ordering_
↪][mask[opt.ordering_]], s=20, c=opt.labels_[opt.ordering_][mask[opt.ordering_]],
↪ cmap='jet')
ax.set_xlabel('points in order of processing')
ax.set_ylabel('shortest reachability distance')
plt.show()
```



CHAPTER

THIRTYSEVEN

DISTRIBUTION-BASED CLUSTERING

In distribution-based clustering each cluster $1, \ldots, k$ is represented by a probability distribution. The underlying assumption is that there exist k probability distributions $p_1, \ldots, p_k$ from which each cluster's data points have been drawn. Given training data one aims to find the probability distribution for each cluster. Usually Gaussian distributions are used, but others may be appropriate as well.

The difficulty is to derive (for fixed k) the distribution parameters from the data set and, thus, cluster positions, sizes, shapes. Resulting model of the data set is denoted as a *mixture model* because the data set is represented as a mixture of points drawn from different distributions. In case of Gaussian distributions the searched for parameters are the k mean vectors and the k covariance matrices. Resulting model then is a *Gaussian mixture model*.

Mixture models are an example of *generative models*, because they allow to generate new samples with similar properties like samples in the training set. Here we do not obtain a model of the training set's clusters themselves, but a model to generate data sets similar to the training set. From this generative model we may extract information about clusters in the training data (distribution parameters), but partitioning of training samples into concrete clusters is an extra step based on information extracted from the model.

Related projects:

- *Generating Handwritten Digits* (page 942)

37.1 Expectation-Maximization

On the one hand we want to find parameters of the k underlying distributions such that the distributions fit well to the data. On the other hand we want to label training samples (assign them to clusters). The questions are how to formalize 'fits well' and how to derive the labels. A common approach is to choose parameters and labels such that the training set is the most probable data set within all data sets which can be drawn from distributions with the chosen parameters respecting the cluster assignments. This approach (and its algorithmic realization below) is known as *expectation-maximization (EM)*.

37.1.1 Notation

The EM approach requires that we consider training samples $x_1, \ldots, x_n$ and labels as realizations of random variables $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$, respectively. The X_l are continuous random variables with values in $\mathbb{R}^m$. The Y_l are discrete random variable taking values in $\{1, \ldots, k\}$.

Clusters $1, \ldots, k$ are described by probability densities $p_i : \mathbb{R}^m \rightarrow [0, \infty)$, $i = 1, \ldots, k$ containing parameters $\vartheta_1, \ldots, \vartheta_k$, respectively. Here, ϑ_i is a vector if the ith density has more than one parameter. If we know the cluster of some sample l, that is, $Y_l = y_l$ for some $y_l \in \{1, \ldots, k\}$, then we know the distribution of the lth sample: it's p_{y_l}. In other words, the densities p_i are our model for describing generation of random samples given preset cluster assignments.

Note that for continuous random variables working with probability densities is much more comfortable than working with the probability measures directly, because $P(X = x) = 0$ for each x, whereas $p(x)$ carries the relevant information as long os p is continuous, which is usually taken for granted when working with densities without explicitly mentioning this important assumption.

For the Y_l we do not specify a model. The Y_l are discrete random variables taking k different values. Thus, the distribution of each Y_l can be described by k non-negative numbers:

$$q_{l,1} := P(Y_l = 1), \quad \ldots, \quad q_{l,k} := P(Y_l = k) \qquad \text{for } l = 1, \ldots, n.$$

The $q_{l,i}$ describe the generation of random cluster assignments for all samples. By definition they satisfy

$$\sum_{i=1}^{k} q_{l,i} = 1 \qquad \text{for } l = 1, \ldots, n.$$

37.1.2 Maximization Problem

The aim is to choose all parameters describing the generation of random samples (that is, the ϑ_i and the $q_{l,i}$) in such a way that the probability to observe the available training data becomes maximal with respect to all possible parameter values:

$$p(x_1, \ldots, x_n) \to \max_{\vartheta_1, \ldots, \vartheta_k, q_{1,1}, \ldots, q_{n,k}}.$$

Here, p is the probability density function of $(X_1, \ldots, X_n)$.

The structure of p is quite involved and corresponding maximization problem hard to solve, analytically as well as numerically.

37.1.3 Idea of EM Algorithm

Solving the maximization problem with respect to all parameters is almost impossible. But if we fix the $q_{l,i}$ to certain special but still reasonable values then optimal $\vartheta_1, \ldots, \vartheta_k$ can be obtained in a straight-forward way. The other way round, if we fix $\vartheta_1, \ldots, \vartheta_k$, then calculating optimal $q_{l,i}$ becomes viable.

We may alternate both steps: Fix (initial, almost random) $q_{l,i}$ and get optimal ϑ_i. Then for those ϑ_i get optimal $q_{l,i}$ and so on until some stopping criterion is satisfied.

37.1.4 Optimal Labels

Given some fixed values for $\vartheta_1, \ldots, \vartheta_k$ we want so solve above maximization problem with respect to the $q_{l,i}$.

The law of total probability allows to rewrite p as

$$p(x_1, \ldots, x_n) = \sum_{y_1=1}^{k} \cdots \sum_{y_n=1}^{k} P(y_1, \ldots, y_n)\, p(x_1, \ldots, x_n \,|\, y_1, \ldots, y_n),$$

where $p(x_1, \ldots, x_n \,|\, y_1, \ldots, y_n)$ is the conditional probability density for $(X_1, \ldots, X_n)$ given conrete values $y_1, \ldots, y_n$ for $Y_1, \ldots, Y_n$.

Mathematical side note

The somewhat unusual mix of densities and probability measure P is mathematically correct. This can be seen by considering a subset A of $\mathbb{R}^m$ instead of only one element, allowing to write everything without densities:

$$P(A) = \sum \cdots \sum P(y_1, \ldots, y_n)\, P(A \,|\, y_1, \ldots, y_n).$$

Replacing P by integrals over corresponding densities we obtain

$$\begin{aligned}\int_A p(x_1, \ldots, x_n)\, \mathrm{d}(x_1, \ldots, x_n) &= \sum \cdots \sum P(y_1, \ldots, y_n) \int_A p(x_1, \ldots, x_n \,|\, y_1, \ldots, y_n)\, \mathrm{d}(x_1, \ldots, x_n) \\ &= \int_A \sum \cdots \sum P(y_1, \ldots, y_n)\, p(x_1, \ldots, x_n \,|\, y_1, \ldots, y_n)\, \mathrm{d}(x_1, \ldots, x_n).\end{aligned}$$

This integral equation holds for all sets A. Thus, integrants on both sides are equal.

Remember

$$P(y_1,\ldots,y_n)=q_{1,y_1}\cdots q_{n,y_n}$$

by definition of the $q_{l,i}$. The conditional density may be rewritten in terms of conditional densities for each sample (we reuse p here although joint and per-sample densities are different functions):

$$p(x_1,\ldots,x_n\,|\,y_1,\ldots,y_n)=p(x_1\,|\,y_1,\ldots,y_n)\cdots p(x_n\,|\,y_1,\ldots,y_n).$$

The distribution of X_l only depends on Y_l, not on Y_λ for $\lambda\neq l$. Thus,

$$p(x_l\,|\,y_1,\ldots,y_n)=p(x_l\,|\,y_l).$$

The value $p(x_l\,|\,y_l)$ is the probability of observing x_l if we know that x_l belongs to cluster y_l. With the cluster densities $p_1,\ldots,p_k$ we thus have

$$p(x_l\,|\,y_1,\ldots,y_n)=p_{y_l}(x_l)$$

The maximization problem to solve now reads

$$\sum_{y_1=1}^{k}\cdots\sum_{y_n=1}^{k}q_{1,y_1}\cdots q_{n,y_n}\,p_{y_1}(x_1)\cdots p_{y_n}(x_n)\to\max_{q_{1,1},\ldots,q_{n,k}}.$$

From the properties of the $q_{l,i}$ (non-negative, sum over i is 1) we immediately see that $q_{1,y_1}\cdots q_{n,y_n}\geq 0$ for each $(y_1,\ldots,y_n)$ and

$$\sum_{y_1=1}^{k}\cdots\sum_{y_n=1}^{k}q_{1,y_1}\cdots q_{n,y_n}=1.$$

Thus, our maximization problem has the structure

$$\sum_j a_j\,z_j\to\max_{a_j},\qquad a_j\geq 0,\qquad \sum_j a_j=1.$$

The solution of such problems is known (and easily verified) to be $a=(0,\ldots,0,1,0,\ldots,0)$ with the 1 at the position of the maximal z_j.

In our setting: Let $y_1^*,\ldots,y_n^*$ be the set of labels maximizing $p_{y_1}(x_1)\cdots p_{y_n}(x_n)$. Then $q_{1,y_1^*}\cdots q_{n,y_n^*}$ has to be 1 (which is only possible if all q_{l,y_l^*} equal 1) and all other $q_{l,i}$ have to be 0. In other words, the optimal distribution of $(Y_1,\ldots,Y_n)$ is deterministic with the whole probability mass on $(Y_1,\ldots,Y_n)=(y_1^*,\ldots,y_n^*)$. Note that $p_{y_1}(x_1)\cdots p_{y_n}(x_n)$ becomes maximal if each factor becomes maximal. Thus,

$$y_l^*=\operatorname{argmax}_i p_i(x_l)\qquad\text{for }l=1,\ldots,n.$$

37.1.5 Optimal Per-Cluster Distributions

Given fixed $q_{l,i}$ we want to solve the above maximization problem with respect to the parameters $\vartheta_1,\ldots,\vartheta_k$ of the per-cluster probability densities $p_1,\ldots,p_k$. We do not solve the maximization problem for arbitraty $q_{l,i}$ but for $q_{l,i}$ with the structure obtained as optimal solution in the previous subsection. That is, the $q_{l,i}$ describe a deterministic probability distribution with all mass on some fixed $(y_1,\ldots,y_n)$.

Given the concrete set of labels $y_1,\ldots,y_n$ the probability to observe our training samples $x_1,\ldots,x_n$ (that is, the objective of the maximization problem) is

$$p(x_1,\ldots,x_n)=\prod_{l=1}^{n}p_{y_l}(x_l)=\prod_{i=1}^{k}\prod_{l:y_l=i}p_i(x_l).$$

Each factor $\prod_{l:y_l=i}p_i(x_l)$ only depends on ϑ_i and not on ϑ_j for $j\neq i$. Thus, we may maximize factors independently.

Finding ϑ_i which maximizes $\prod_{l:y_l=i}p_i(x_l)$ is a standard task in maximum likelihood estimation[592]. Depending on the structure of p_i there exist explicit solutions (see below for Gaussion densities).

[592] https://en.wikipedia.org/wiki/Maximum_likelihood_estimation

37.2 The EM Algorithm

The procedure derived above can be summarized as follows:

1. Randomly choose initial labels $y_1, \ldots, y_n$.
2. Calculate optimal $\vartheta_1, \ldots, \vartheta_k$ based on the training samples in each cluster.
3. Update labels by assigning each x_l the cluster i with the highest $p_i(x_l)$.
4. Go to 2 if stopping criterion not satisfied.

Common stopping criteria are to stop if the y_l or the ϑ_i do not change anymore.

The process is known to converge to a stationary point (gradient is zero) of the original maximization problem. Thus, it may get stuck in local maxima or even in saddle points.

37.3 Gaussian Mixtures

If $p_1, \ldots, p_k$ are Gaussian probability densities, p_i has the mean vector and the covariance matrix as parameters. The products

$$\prod_{l:y_l=i} p_i(x_l), \qquad i = 1, \ldots, k$$

in the above derivation are maximized with respect to the distribution parameters, if we choose mean and covariance of

$$\{x_l \,:\, y_l = i\} \qquad \text{(cluster } i\text{)}$$

as parameters.

EM with Gaussian distributions is a very fast algorithm. The two alternating steps boil down to:

- calculate probabilities $p_{l,i} := p_i(x_l)$ for all pairs of samples and clusters,
- for each l set $y_l :=$ argmax$_{i=1,\ldots,k} p_{l,i}$,
- calculate empirical mean vectors and empirical covariance matrices for each cluster.

37.4 Relation to k-Means

The idea of alternating two steps, each solving the problem partially, is very similar to k-means:

- EM: for fixed labels get optimal parameters.k-means: for fixed labels get centroid of each cluster.
- EM: for fixed parameters get optimal labels.k-means: for fixed centroids get optimal labels.

One easily verifies that the training of Gaussian mixture models with identity covariance matrices is equivalent to k-means clustering.

37.5 Choosing k

Like for k-means the number of clusters has to be known in advance. The larger k, the better the data can be represented by the model, but interpretation of clusters becomes more difficult (overfitting).

Silhouette score and Davies-Bouldin index may help choosing k. For elbow method one can use the optimal value of the objective function from the above maximization problem. Another approach is known as *Bayesian information criterion*. Here one chooses k to minimize

$$t\,\log n - \log p(x_1, \ldots, x_n).$$

The second summand is the negative logarithm of the optimal value of the original maximization problem and the first summand is a penalty containing the number t of model parameters. Obviously, t is a multiple of k if all k distributions are of equal type (e.g., Gaussian). The more clusters the model has, the higher t and the larger the penalty. Minimizing the sum yields a value for k which is a compromise between number of clusters and fitting error. For more detailed motivation of BIC see Wikipedia on BIC[593].

37.6 Soft Clustering

EM allows for soft clustering, that is, each sample we may assign probabilities to belong to the different clusters. The $p_i(x_l)$ can be interpreted as score for x_l to belong to cluster i. Probability-like scores can be obtained the usual way as

$$\frac{p_i(x_l)}{p_1(x_l)+\cdots+p_k(x_l)}.$$

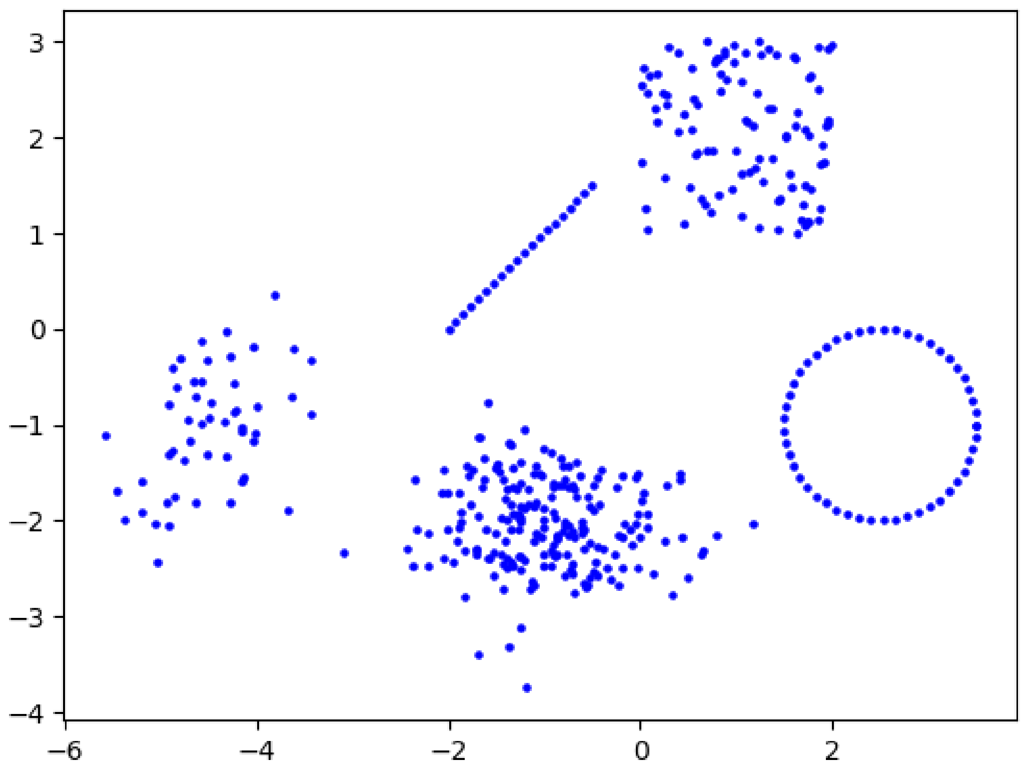
37.7 Gaussian Mixture Models with Scikit-Learn

Scikit-Learn provides the GaussianMixture[594] class in its `mixture` module. The only relevant parameter is `n_components` (our k).

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import sklearn.mixture as mixture
import sklearn.metrics as metrics

rng = np.random.default_rng(0)
```

```
# 5 clusters in 2 dimensions

n1, n2, n3, n4, n5 = 100, 200, 20, 50, 50
n = n1 + n2 + n3 + n4 + n5

X1 = rng.uniform((0, 1), (2, 3), (n1, 2))
X2 = rng.multivariate_normal((-1, -2), ((0.5, 0), (0, 0.2)), n2)
X3 = np.linspace(-1, 0.5, n3).reshape(-1, 1) * np.ones((1, 2)) + np.array([[-1,␣
↪1]])
X4 = rng.multivariate_normal((-4.5, -1), ((0.2, 0.1), (0.1, 0.4)), n4)
phi = np.linspace(0, 2 * np.pi, n5).reshape(-1, 1)
X5 = np.array([[2.5, -1]]) + np.concatenate((np.cos(phi), np.sin(phi)), axis=1)

X = np.concatenate((X1, X2, X3, X4, X5))

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], s=5, c='b')
ax.axis('equal')
plt.show()
```

593 https://en.wikipedia.org/wiki/Bayesian_information_criterion
594 https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html



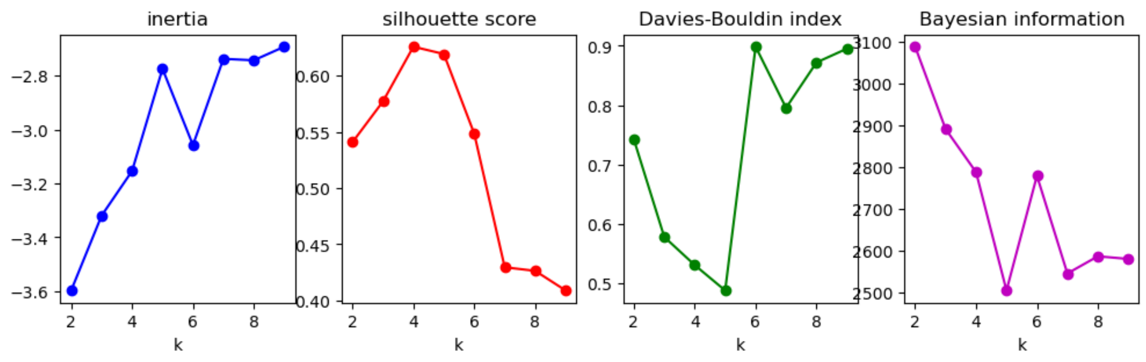
```
ks = range(2, 10)

obj = []
sil = []
db = []
bic = []
for k in ks:

    print(k)

    gm = mixture.GaussianMixture(n_components=k)
    gm.fit(X)

    labels = gm.predict(X)
    obj.append(gm.lower_bound_)
    sil.append(metrics.silhouette_score(X, labels))
    db.append(metrics.davies_bouldin_score(X, labels))
    bic.append(gm.bic(X))

obj = np.array(obj)
sil = np.array(sil)
db = np.array(db)
bic = np.array(bic)

fig, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(12, 3))
ax1.plot(ks, obj, '-ob')
ax2.plot(ks, sil, '-or')
ax3.plot(ks, db, '-og')
ax4.plot(ks, bic, '-om')
ax1.set_xlabel('k')
ax2.set_xlabel('k')
ax3.set_xlabel('k')
ax4.set_xlabel('k')
ax1.set_title('inertia')
```

(continues on next page)

(continued from previous page)

```
ax2.set_title('silhouette score')
ax3.set_title('Davies-Bouldin index')
ax4.set_title('Bayesian information')
plt.show()
```

```
2
3
4
5
6
7
8
9
```



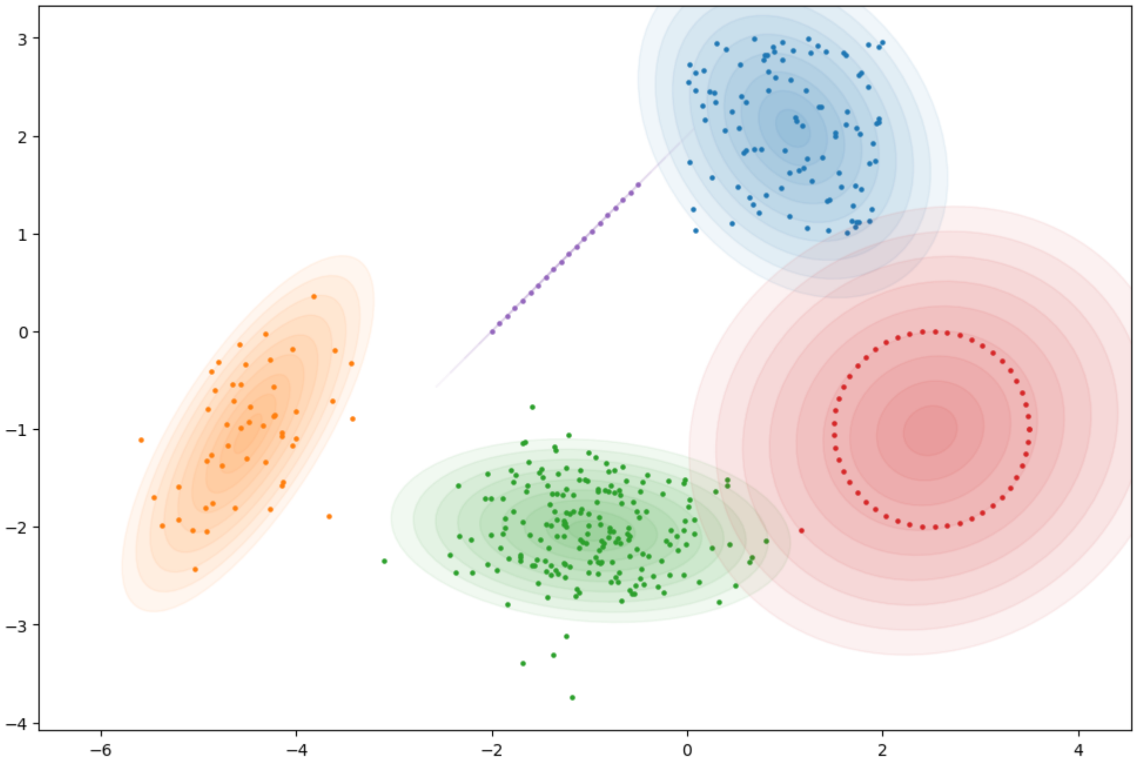
```
import matplotlib as mpl
```

```
k = 5

gm = mixture.GaussianMixture(k)
gm.fit(X)

colors = list(mpl.colors.TABLEAU_COLORS.values())

fig, ax = plt.subplots(figsize=(12, 8))

for label in range(0, k):
    mean = gm.means_[label, :]
    cov = gm.covariances_[label, :, :]

    # do some calculations for drawing an ellipse
    evals, evecs = np.linalg.eigh(cov)
    if evecs[0, 0] == 0:
        angle = 0.5 * np.pi
    else:
        angle = np.arctan(evecs[1, 0] / evecs[0, 0])

    # draw ellipses (level sets of Gaussian density)
    for fac in range(1, 10):
        ell = mpl.patches.Ellipse(mean, fac * evals[0], fac * evals[1],␣
↪angle=angle / np.pi * 180,
                                  color=colors[label] + '10')
        ax.add_artist(ell)

    # plot cluster
    labels = gm.predict(X)
```

(continues on next page)

(continued from previous page)

```
    mask = labels == label
    ax.scatter(X[mask, 0], X[mask, 1], s=5, c=colors[label])

ax.axis('equal')
plt.show()
```



CHAPTER

THIRTYEIGHT

AUTOENCODERS

An autoencoder is a special ANN mainly used for dimensionality reduction and anomaly detection. In principle, autoencoders can be used for generating new samples similar to training samples (generative model), but there exist much better generative models nowadays. We introduce autoencoders here, because they are a relatively simple ancestor of variational autoencoders, one of two very prominent generative machine learning techniques (the other prominent technique are generative adversarial networks).

Related projects:

- *MNIST Character Recognition* (page 931)
    - *Autoencoder for QMNIST* (page 943)

38.1 ANN Structure

Like PCA, autoencoders aim at reducing feature space dimension without loosing relevant information. PCA follows a clear mathematical reasoning to reduce feature space dimension. In particular, for PCA the transform between original and reduced feature space is linear (multiplication by a matrix). Autoencoders use ANNs for transforming features and also for the inverse transform. Thus, transformation may be highly nonlinear and, thus, much more flexible.

Encoder ANN and decoder ANN are trained simultaneously by joining them to form one large ANN. The input neurons of the decoder are connected to the output neurons of the encoder. Training data consists of (unlabeled) samples which are used both as inputs and as outputs of the ANN. So the ANN learns to reproduce its inputs!

The output of the encoder sometimes is denoted as *code*. If the number of encoder output neurons (dimension of reduced feature space or code space) is smaller than the original feature space dimension, then the ANN is forced to drop information not relevant for reproducing inputs.

There is nothing special about training an autoencoder. Usually mean squared error is used as loss. Regularization may be used to enforce certain special properties of the codes. An example are sparse codes (resulting from ℓ^1-penalty), which may be used together with high dimensional code spaces to identify most relevant original features.

38.2 Implementing an Autoencoder

Scikit-Learn does not offer autoencoders (at the moment), so we have to use Keras and create an ANN with special autoencoder structure manually.

We first create a simple toy data set for demonstration.

```
import numpy as np
import matplotlib.pyplot as plt

rng = np.random.default_rng(0)
```

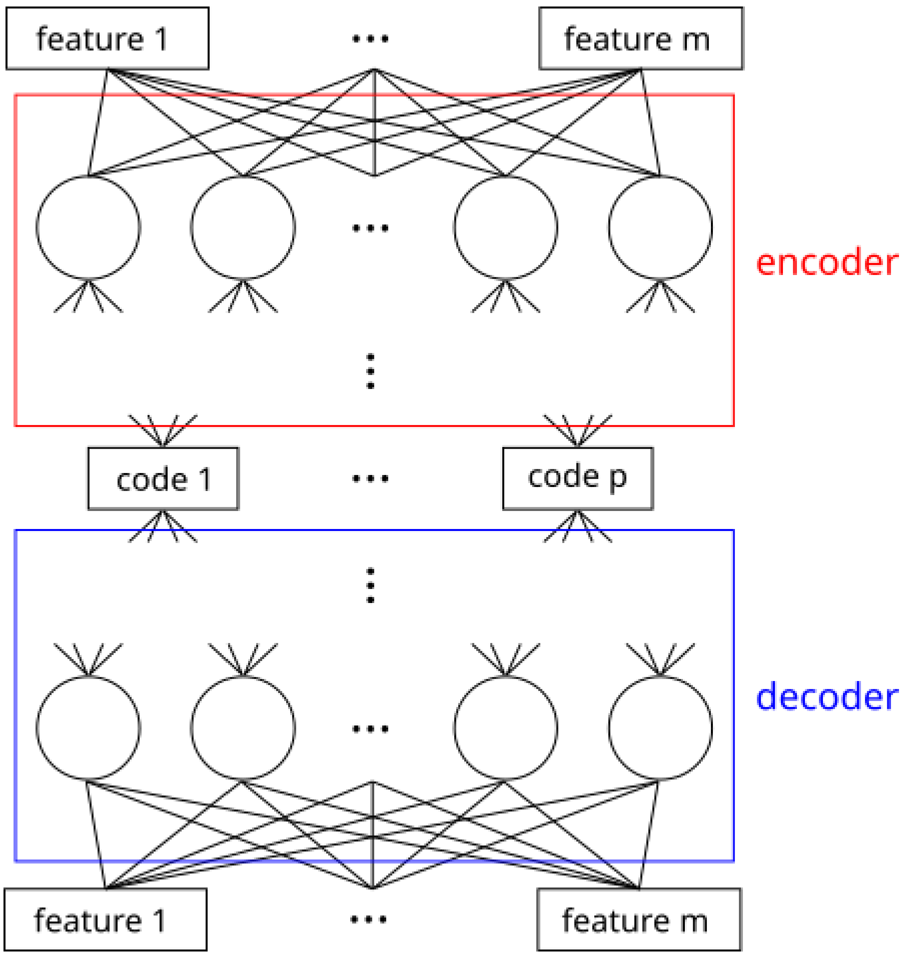
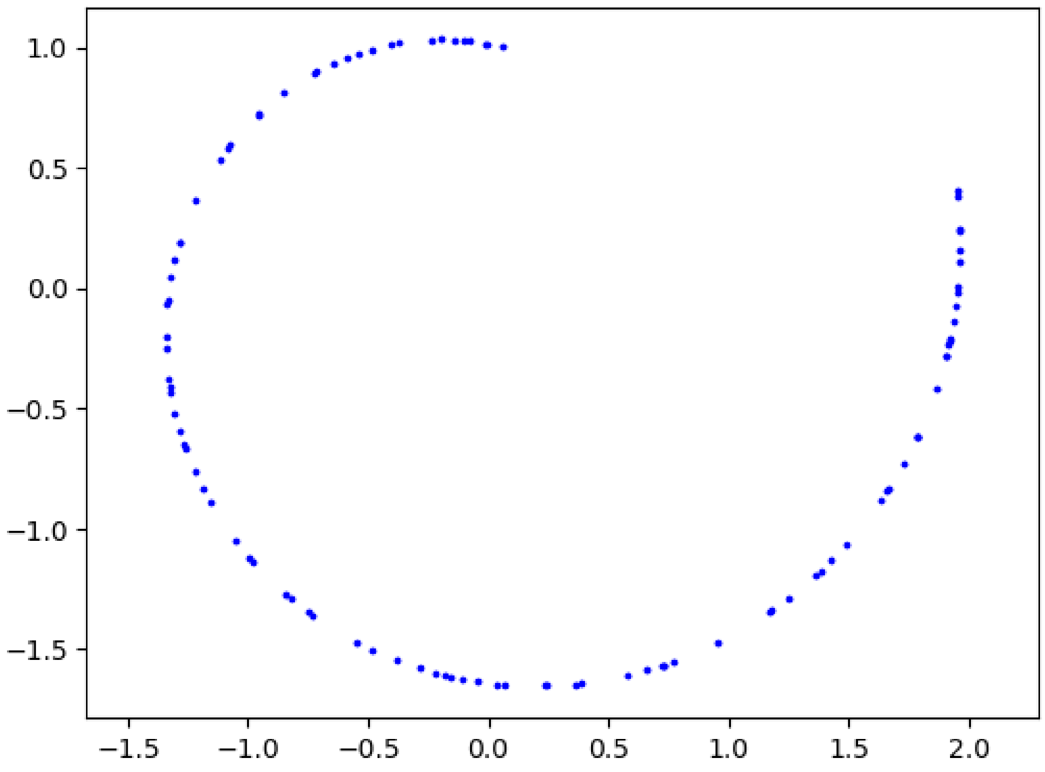


Fig. 38.1: Autoencoders are ANNs with as many outputs as inputs.

```
n = 100

t = rng.uniform(0, 1, n)
X = np.empty((n, 2))
X[:, 0] = (t + 1) * np.cos(5 * t + 1.5)
X[:, 1] = (t + 1) * np.sin(5 * t + 1.5)

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], c='b', s=3)
ax.axis('equal')
plt.show()
```



Original data space has two dimensions here. Human eye immediately sees that the data set lives on a one dimensional manifold. Thus, we could unwind the curve to map the data set into a one dimensional space without loss of information. So code space should have one dimension and the encoder ANN has to learn the unwinding operation. We give it a try with two small layers. Using only one layer would yield a more or less linear transform, which obviously cannot unwind the curve.

```
# disable GPU if TensorFlow with GPU causes problems
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

import tensorflow.keras as keras
```

```
2024-03-07 08:07:50.236721: I tensorflow/core/platform/cpu_feature_guard.
 ↪cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network␣
 ↪Library (oneDNN) to use the following CPU instructions in performance-
 ↪critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate␣
 ↪compiler flags.
```

```
encoder = keras.Sequential(name='encoder')
encoder.add(keras.Input(shape=(2,)))
encoder.add(keras.layers.Dense(10, activation='relu', name='enc1'))
encoder.add(keras.layers.Dense(10, activation='relu', name='enc2'))
encoder.add(keras.layers.Dense(1, activation='linear', name='code'))

encoder.summary()
```

```
Model: "encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 enc1 (Dense)                (None, 10)                30        
                                                                 
 enc2 (Dense)                (None, 10)                110       
                                                                 
 code (Dense)                (None, 1)                 11        
                                                                 
=================================================================
Total params: 151
Trainable params: 151
Non-trainable params: 0
_________________________________________________________________
```

```
2024-03-07 08:07:52.104978: E tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪driver.cc:267] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capable␣
↪device is detected
2024-03-07 08:07:52.105057: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:169] retrieving CUDA diagnostic information for host: WHZ-46349
2024-03-07 08:07:52.105077: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:176] hostname: WHZ-46349
2024-03-07 08:07:52.105370: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:200] libcuda reported version is: 525.147.5
2024-03-07 08:07:52.105437: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:204] kernel reported version is: 525.147.5
2024-03-07 08:07:52.105454: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
↪diagnostics.cc:310] kernel version seems to match DSO: 525.147.5
2024-03-07 08:07:52.106016: I tensorflow/core/platform/cpu_feature_guard.
↪cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network␣
↪Library (oneDNN) to use the following CPU instructions in performance-
↪critical operations:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate␣
↪compiler flags.
```

The decoder ANN looks the same because it as to learn the inverse of the encoder. So complexity of the mappings to learn by encoder and decoder is comparable.

```
decoder = keras.Sequential(name='decoder')
decoder.add(keras.Input(shape=(1,)))
decoder.add(keras.layers.Dense(10, activation='relu', name='dec1'))
decoder.add(keras.layers.Dense(10, activation='relu', name='dec2'))
decoder.add(keras.layers.Dense(2, activation='linear', name='reconstruction'))

decoder.summary()
```

```
Model: "decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
```

(continues on next page)

(continued from previous page)

```
 dec1 (Dense)                (None, 10)                20

 dec2 (Dense)                (None, 10)                110

 reconstruction (Dense)      (None, 2)                 22

=================================================================
Total params: 152
Trainable params: 152
Non-trainable params: 0
_________________________________________________________________
```

Now we join encoder and decoder. Keras models can be used as layers of a new model. So we only have to create a sequential model with encoder and decoder as layers. Important: all three models use the same underlying computation graph.

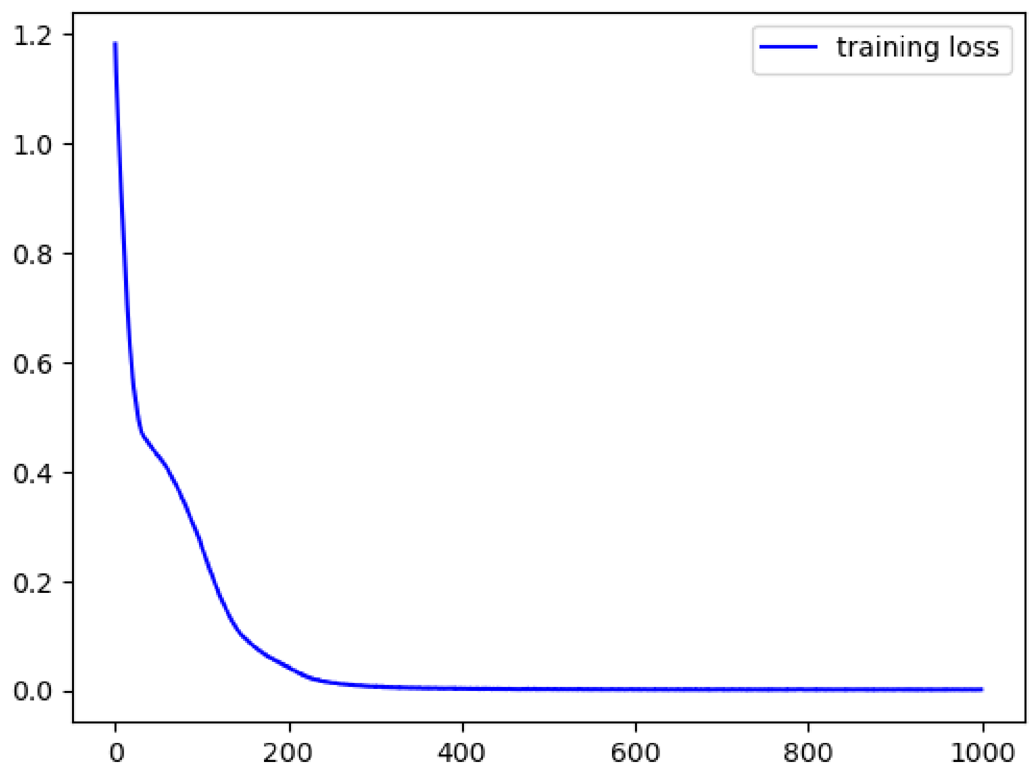
```
autoencoder = keras.Sequential(name='autoencoder')
autoencoder.add(keras.Input(shape=(2,)))
autoencoder.add(encoder)
autoencoder.add(decoder)

autoencoder.summary()
```

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder (Sequential)        (None, 1)                 151

 decoder (Sequential)        (None, 2)                 152

=================================================================
Total params: 303
Trainable params: 303
Non-trainable params: 0
_________________________________________________________________
```

We see that the number of weights equals the sum of the weights of encoder and decoder. The autoencoder model shares weights with encoder and decoder models. If we train the autoencoder the weights of encoder and decoder get modified, too (because there is only one computation graph).

```
autoencoder.compile(loss='mean_squared_error')

loss = []
```

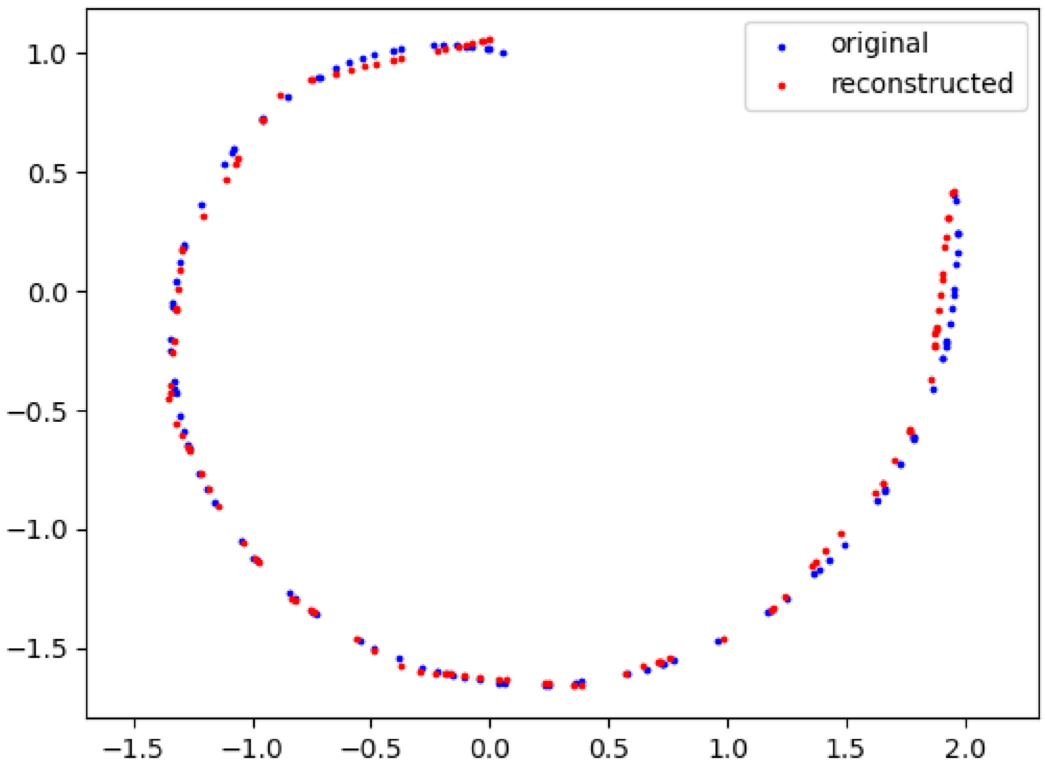
```
history = autoencoder.fit(X, X, epochs=1000, verbose=0)

loss.extend(history.history['loss'])

fig, ax = plt.subplots()
ax.plot(loss, '-b', label='training loss')
ax.legend()
plt.show()
```



To see whether training has been successful we may calculate root mean squared error and plot original and reconstructed data.

```
X_pred = autoencoder.predict(X)

print('RMSE:', np.sqrt(((X - X_pred) ** 2).sum() / n))

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], c='b', s=3, label='original')
ax.scatter(X_pred[:, 0], X_pred[:, 1], c='r', s=3, label='reconstructed')
ax.axis('equal')
ax.legend()
plt.show()
```

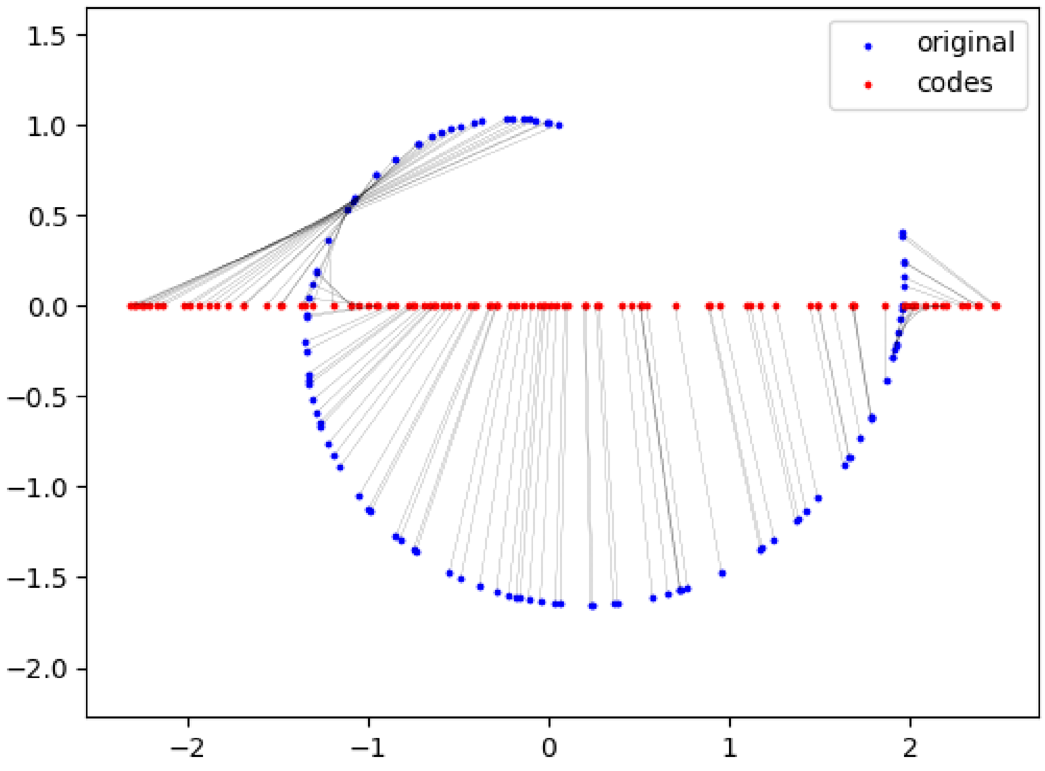
```
4/4 [==============================] - 0s 2ms/step
RMSE: 0.038026978404184425
```



Codes live in one dimension. To visualize the mapping of original data to codes we embed the one dimensional code space into the original data space and connect each sample with its code by a line.

```
C = encoder.predict(X)

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], c='b', s=3, label='original')
ax.scatter(C, np.zeros_like(C), c='r', s=3, label='codes')
for l in range(0, n):
    ax.plot([X[l, 0], C[l, 0]], [X[l, 1], 0], 'k', linewidth=0.1)
ax.axis('equal')
ax.legend()
plt.show()
```

```
4/4 [==============================] - 0s 2ms/step
```



From the visualization we see that the autoencoder preserves order. This is not (!) a general property of autoencoders, but here it works.

38.3 Comparison to PCA

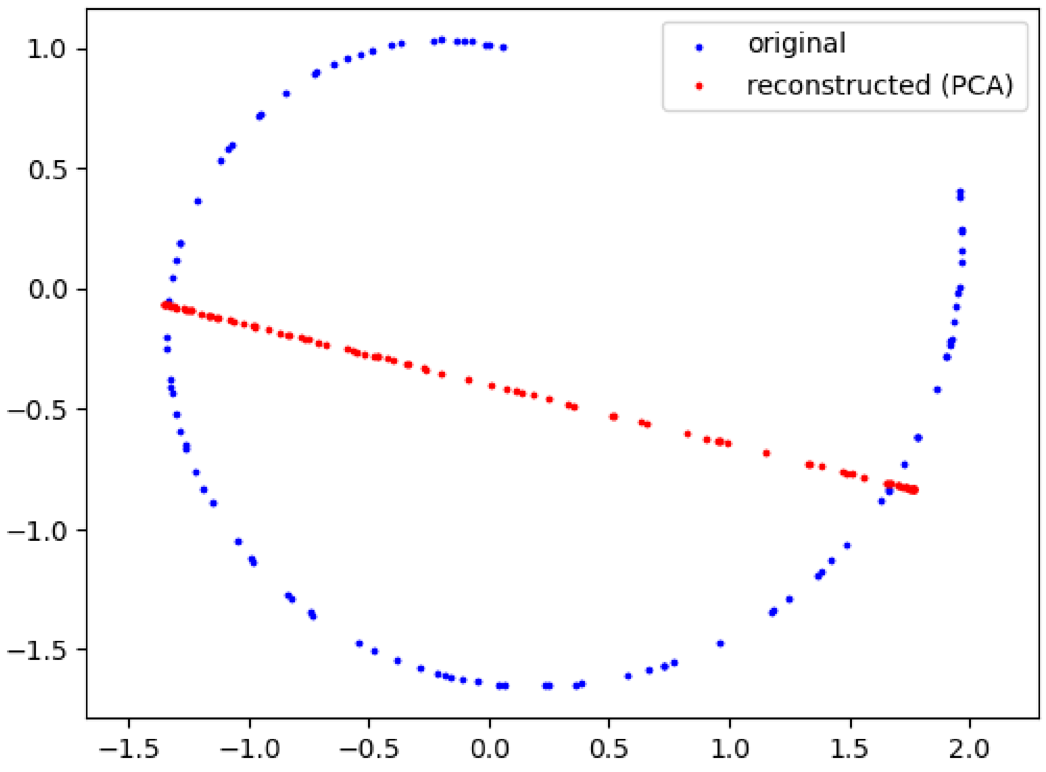
For comparison we use PCA to reduce feature space dimesion.

```
import sklearn.decomposition as decomposition
```

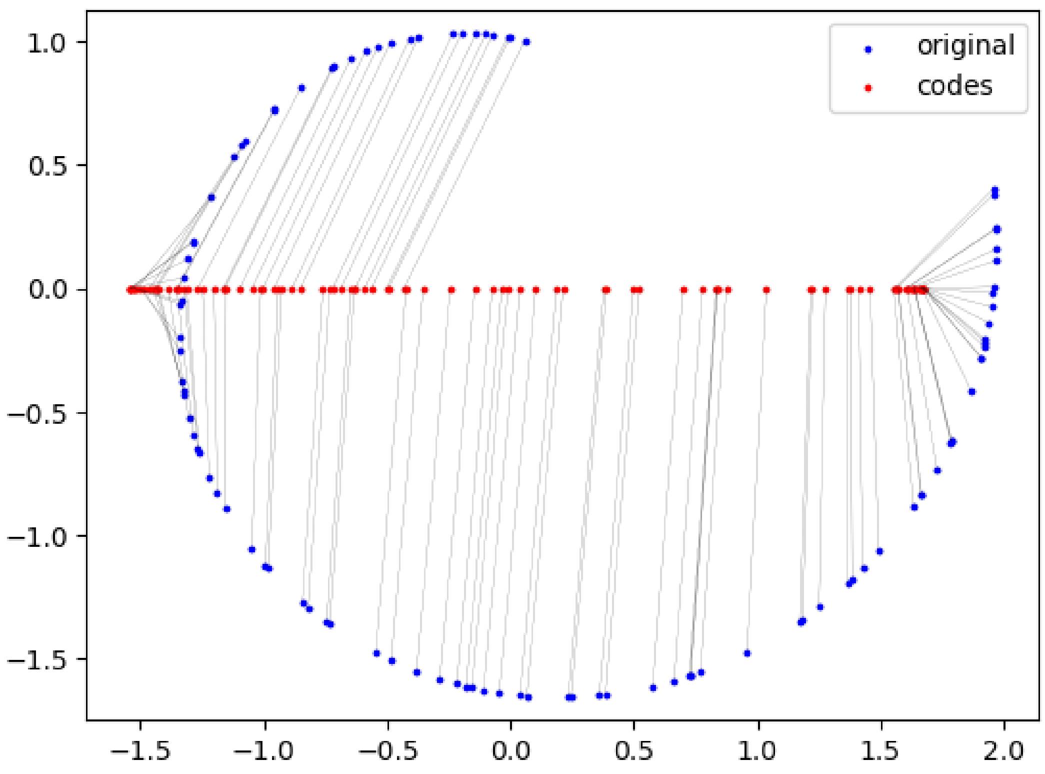
```
pca = decomposition.PCA(1)
C_pca = pca.fit_transform(X)     # encoder
X_pca = pca.inverse_transform(C_pca)     # decoder
```

```
print('RMSE:', np.sqrt(((X - X_pca) ** 2).sum() / n))

fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], c='b', s=3, label='original')
ax.scatter(X_pca[:, 0], X_pca[:, 1], c='r', s=3, label='reconstructed (PCA)')
ax.axis('equal')
ax.legend()
plt.show()
```

```
RMSE: 0.9049436506773132
```



```
fig, ax = plt.subplots()
ax.scatter(X[:, 0], X[:, 1], c='b', s=3, label='original')
ax.scatter(C_pca, np.zeros_like(C_pca), c='r', s=3, label='codes')
for l in range(0, n):
    ax.plot([X[l, 0], C_pca[l, 0]], [X[l, 1], 0], 'k', linewidth=0.1)
ax.axis('equal')
ax.legend()
plt.show()
```



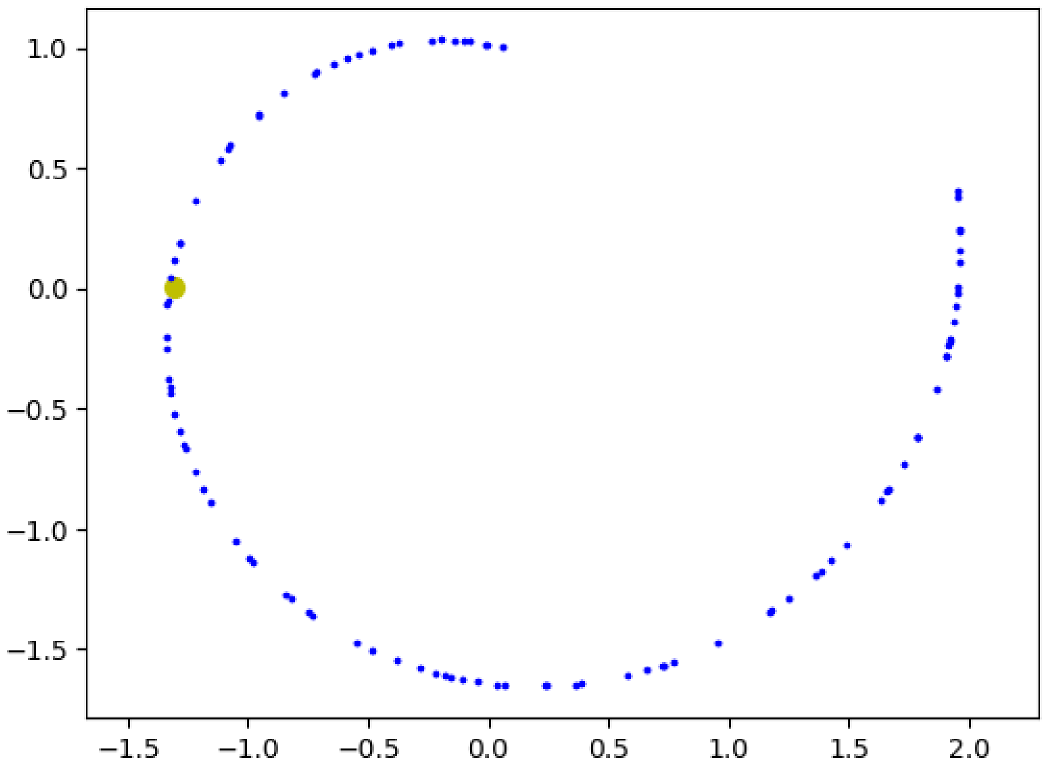
PCA maps distant points to similar codes because PCA maps original data to codes by orthogonal projection.

38.4 Generating New Samples

Autoencoders can be used as generative models. If we feed some input (code) into the decoder, it will yield some output. At least if the input is in the range of codes generated from training data output should be close to original data, that is, similar to training samples.

```
C_new = np.array([[-1]])
#C_new = np.linspace(-10, 10, 100).reshape(-1, 1)

X_new = decoder.predict(C_new)

fig, ax = plt.subplots()
ax.scatter(X_new[:, 0], X_new[:, 1], c='y', s=50)
ax.scatter(X[:, 0], X[:, 1], c='b', s=3)
ax.axis('equal')
plt.show()
```

```
1/1 [==============================] - 0s 43ms/step
```

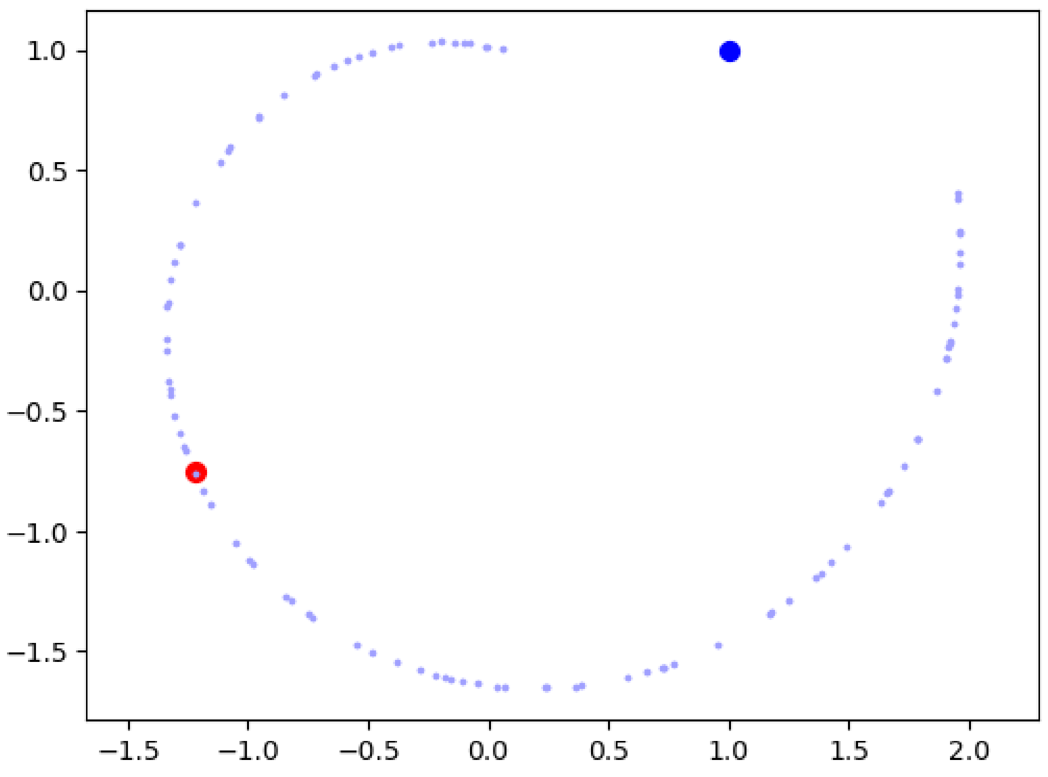


38.5 Anomaly Detection

Autoencoders typically yield good reconstructions only on data similar to training data. If we feed an autoencoder with samples very different from training data reconstructions won't look like inputs. This behavior can be used for anomaly detection. If an autoencoder is trained on 'positive' data only (legitimate credit card transactions for instance) it is likely to yield bad reconstructions on 'negative' samples (fraudulent credit card transactions).

```
X_bad = np.array([[1, 1]])
#X_bad = np.array([[-1, -1]])

X_bad_pred = autoencoder.predict(X_bad)

print('RMSE:', np.sqrt(((X_bad - X_bad_pred) ** 2).sum() / len(X_bad)))

fig, ax = plt.subplots()
ax.scatter(X_bad[:, 0], X_bad[:, 1], c='b', s=50)
ax.scatter(X_bad_pred[:, 0], X_bad_pred[:, 1], c='r', s=50)
ax.scatter(X[:, 0], X[:, 1], c='#a0a0ff', s=3)
ax.axis('equal')
plt.show()
```

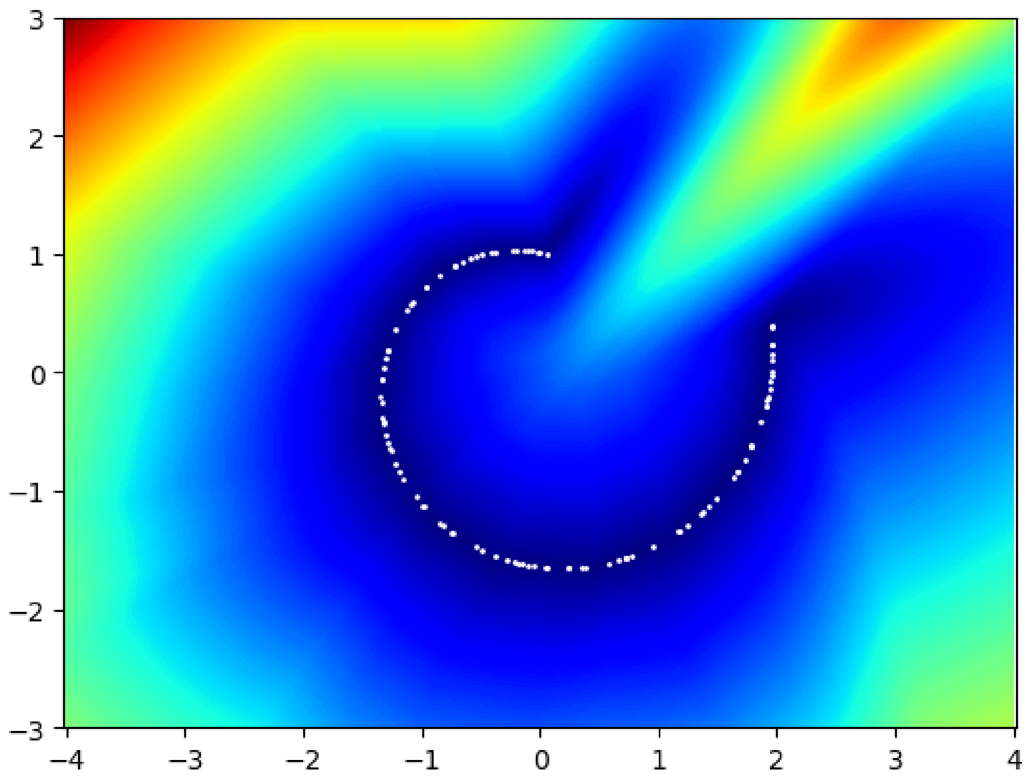
```
1/1 [==============================] - 0s 63ms/step
RMSE: 2.8292557635876245
```



Calculating RMSE between original and reconstructed samples for a grid of samples we get a visual impression of the autoencoder mapping.

```
n_grid = 200

u, v = np.meshgrid(np.linspace(-4, 4, n_grid), np.linspace(-3, 3, n_grid))
X_grid = np.concatenate((u.reshape(-1, 1), v.reshape(-1, 1)), axis=1)

X_grid_pred = autoencoder.predict(X_grid)

errors = np.sqrt(((X_grid - X_grid_pred) ** 2).sum(axis=1) / len(X_grid))

fig, ax = plt.subplots()
ax.contourf(u, v, errors.reshape(n_grid, n_grid), cmap='jet', levels=200)
ax.scatter(X[:, 0], X[:, 1], c='w', s=1)
ax.axis('equal')
plt.show()
```

```
1250/1250 [==============================] - 2s 1ms/step
```



38.6 Convolutional Autoencoders

For image processing task CNNs (convolutional ANNs) are known to perform much better than fully connected CNNs. Using a CNN as encoder is straight forward, but how to 'invert' a CNN for decoding? The decoder has to upsample the code until the original image size is reached. For this upsampling process we have to options:

- simple upsampling (e.g. duplicating each pixel),
- transposed convolution, also known as backward convolution.

A transposed convolution layer has inverse structure of a convolution layer (inputs and outputs switched). Like a usual convolutional layer, it is defined by a stack of filters (shared weights) and a stride value (step size for moving the filter to the next position). Note that a transposed convolution layer only inverts the structure (connections between neurons), not the calculation. So chaining convolution and transposed convolution is not the identity, but only the chain's input shape equals its output shape.

From the computation scheme we see that transposed convolution can be regarded as usual convolution with zero padded inputs (add zeros on left and right side).

While stride is applied to inputs for usual convolution, for transposed convolution stride is applied to the outputs.

A decoder symmetric to a CNN encoder can be constructed by inverting the encoder's layer stack and replacing pooling by upsampling and convolutions by transposed convolutions. Corresponding Keras layers are `UpSampling2D`[595] and `Conv2DTranspose`[596]. 1d and 2d variants exist as well, but `Conv1DTranspose` not before TensorFlow 2.3.

[595] https://keras.io/api/layers/reshaping_layers/up_sampling2d
[596] https://keras.io/api/layers/convolution_layers/convolution2d_transpose

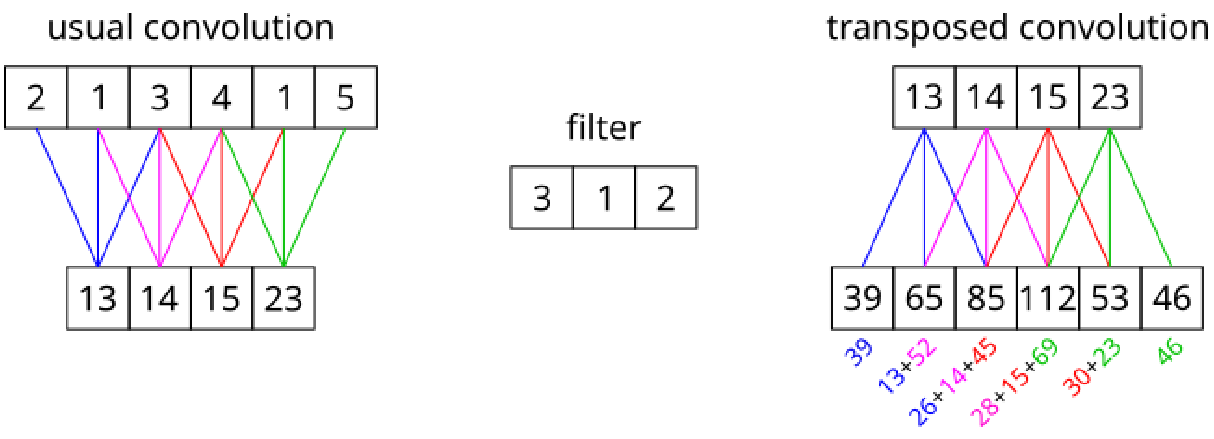


Fig. 38.2: Transposed convolutions do NOT invert convolutions but corresponding shapes only.

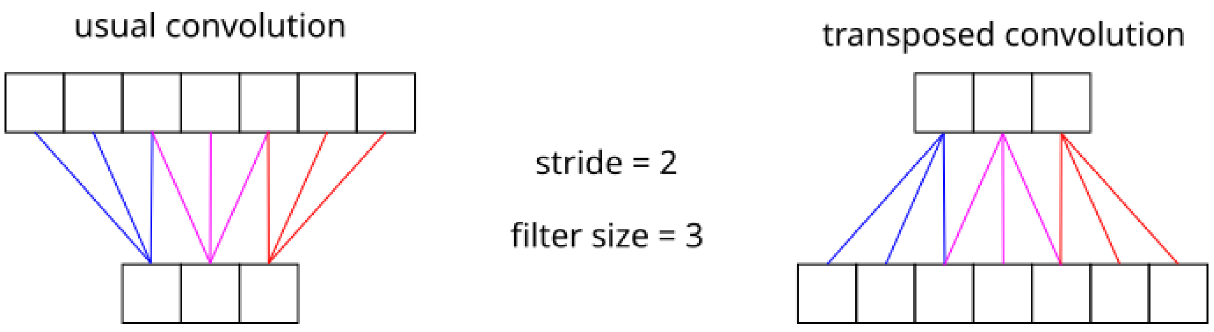


Fig. 38.3: Striding for transposed convolutions is realized such that resulting that input and output shapes match ouput and input shapes of corresponding usual convolution.

CHAPTER

THIRTYNINE

NONLINEAR DIMENSIONALITY REDUCTION OVERVIEW

Up to now we only considered PCA and autoencoders for dimensionality reduction. PCA is a linear technique, that is, it applies a linear transform (matrix multiplication) to the data. Autoencoders are nonlinear and, thus, more flexible.

There exist many other nonlinear techniques for dimensionality reduction. Dimensionality reduction is very important for visualizing high dimensional data. Some of them turned out to be more or less equivalent, some are different realizations of the same idea. The following scheme provides an overview:

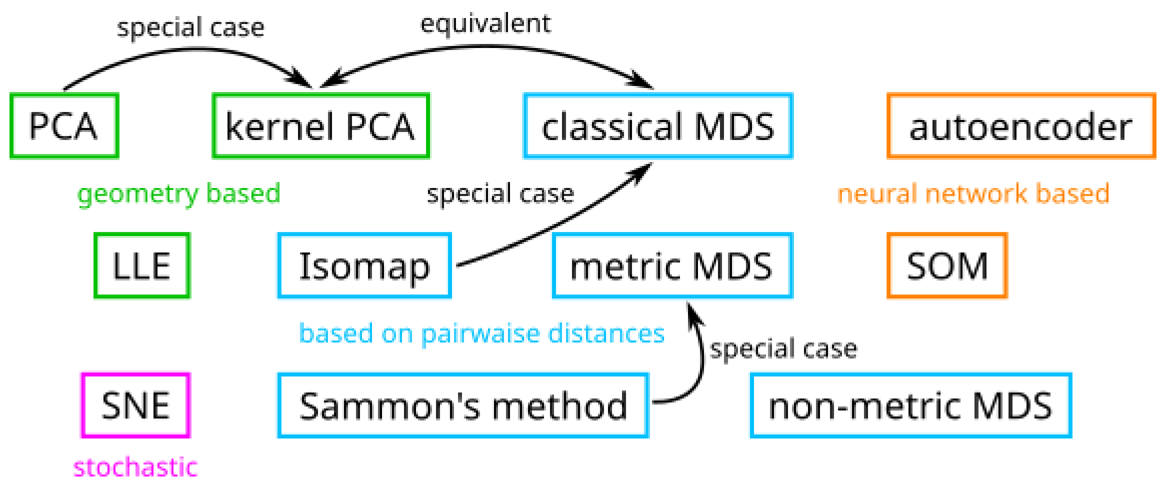


Fig. 39.1: Nonlinear dimensionality reduction is a wide field, but several methods turn out to be equivalent after careful inspection.

39.1 Toy Example 'Omega'

The first toy example for testing nonlinear dimensionality reduction is an Ω-shaped two dimensional manifold in $\mathbb{R}^3$. Data points lie (up to some noise) on this nonlinear manifold.

To each generated sample we assign a different color. Thus, after embedding the manifold into 2d space we can reconstruct from where in 3d space the sample came.

```
import numpy as np
import plotly.graph_objects as go

rng = np.random.default_rng()
```

```
n1 = 75     # number of grid points in first dimension
n2 = 40     # number of gird points in second dimension
noise = 0.005     # noise level for moving samples away from the manifold
```

(continues on next page)

(continued from previous page)

```
# parameter space
S, T = np.meshgrid(np.linspace(0, 1, n1), np.linspace(0, 1, n2))
S = S.reshape(-1)
T = T.reshape(-1)

# noise in parameter space to destroy rigid grid structure
S = S + rng.normal(0, 1 / (2 * n1), S.size)
T = T + rng.normal(0, 1 / (2 * n2), T.size)

# cut-off to keep parameters in [0, 1]
S = np.clip(S, 0, 1)
T = np.clip(T, 0, 1)

# samples in 3d
x = S + 0.15 * np.sin(4 * np.pi * S)
y = T
z = 5 * np.maximum(0, -np.abs(S - 0.5) + 0.5) ** 1 + 1 * T ** 2

# colors
red = np.sin(4 * np.pi * S)
green = np.sin(2 * np.pi * T)
blue = np.sin(2 * np.pi * (S + T))
red = (255 * (red - red.min()) / (red - red.min()).max()).astype(int)
green = (255 * (green - green.min()) / (green - green.min()).max()).astype(int)
blue = (255 * (blue - blue.min()) / (blue - blue.min()).max()).astype(int)

# some noise
x = x + rng.normal(0, noise, x.size)
y = y + rng.normal(0, noise, y.size)
z = z + rng.normal(0, noise, z.size)

# plot
fig = go.Figure(layout_width=800, layout_height=600, layout_scene_aspectmode='cube
↪')
fig.add_trace(go.Scatter3d(
    x=x, y=y, z=z,
    mode='markers',
    marker={'size': 2, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
    hoverinfo = 'none'
))
fig.show()
```

```
<IPython.core.display.HTML object>
```

```
np.savez('omega.npz', x=x, y=y, z=z, red=red, green=green, blue=blue)
```

39.2 Toy Example 'Sphere'

The next toy example is a sphere shaped 2d manifold in 3d space. This manifold cannot be mapped into 2d space without cuts or overlaps.

```
n = 40      # number of stacked circles
noise = 0.05     # noise level for moving samples away from the manifold

# phi is latitude angle
```

(continues on next page)

(continued from previous page)

```
# theta is longitude angle
# number of longitudes depends on latitude (more points per latitude on equator␣
↪than near poles)
x = []
y = []
z = []
red = []
green = []
blue = []
for phi in np.linspace(0, np.pi, n + 2)[1:-1]:
    m = int(2 * n * np.abs(np.sin(phi)))
    for i in range(0, m):
        phi_noisy = phi + rng.normal(0, np.pi / (2 * n))
        r = np.sin(phi_noisy)
        theta = i * 2 * np.pi / m + rng.normal(0, np.pi / m)
        x.append(r * np.cos(theta))
        y.append(r * np.sin(theta))
        z.append(np.cos(phi_noisy))
        red.append(np.sin(2 * phi_noisy))
        green.append(np.sin(2 * theta))
        blue.append(np.sin(2 * (phi_noisy + theta)))

x = np.array(x)
y = np.array(y)
z = np.array(z)

red = np.array(red)
green = np.array(green)
blue = np.array(blue)
red = (255 * (red - red.min()) / (red - red.min()).max()).astype(int)
green = (255 * (green - green.min()) / (green - green.min()).max()).astype(int)
blue = (255 * (blue - blue.min()) / (blue - blue.min()).max()).astype(int)

# some noise
x = x + rng.normal(0, noise, x.size)
y = y + rng.normal(0, noise, y.size)
z = z + rng.normal(0, noise, z.size)

# plot
fig = go.Figure(layout_width=800, layout_height=600, layout_scene_aspectmode='cube
↪')
fig.add_trace(go.Scatter3d(
    x=x, y=y, z=z,
    mode='markers',
    marker={'size': 2, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
    hoverinfo = 'none'
))
fig.show()
```

```
<IPython.core.display.HTML object>
```

```
np.savez('sphere.npz', x=x, y=y, z=z, red=red, green=green, blue=blue)
```

39.3 Toy Example 'Cube'

The next example is a filled 3d cube, which cannot be mapped into two dimensions without destroying its structure.

```
n = 15     # number of grid points per axis

x, y, z = np.meshgrid(np.linspace(0, 1, n), np.linspace(0, 1, n), np.linspace(0,␣
↪1, n))
x = x.reshape(-1)
y = y.reshape(-1)
z = z.reshape(-1)

# some noise
x = x + rng.normal(0, 1 / (2 * n), x.size)
y = y + rng.normal(0, 1 / (2 * n), y.size)
z = z + rng.normal(0, 1 / (2 * n), z.size)

# colors
red = np.cos(2 * np.pi * x)
green = np.cos(2 * np.pi * y)
blue = np.cos(2 * np.pi * z)
red = (255 * (red - red.min()) / (red - red.min()).max()).astype(int)
green = (255 * (green - green.min()) / (green - green.min()).max()).astype(int)
blue = (255 * (blue - blue.min()) / (blue - blue.min()).max()).astype(int)

# plot
fig = go.Figure(layout_width=800, layout_height=600, layout_scene_aspectmode='cube
↪')
fig.add_trace(go.Scatter3d(
    x=x, y=y, z=z,
    mode='markers',
    marker={'size': 2, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
    hoverinfo = 'none'
))
fig.show()
```

```
<IPython.core.display.HTML object>
```

```
np.savez('cube.npz', x=x, y=y, z=z, red=red, green=green, blue=blue)
```

39.4 Toy Example 'Clouds'

The next example data set consists of four 2d point clouds in 3d space. With this data set we can investigate the behavior of nonlinear dimensionality reduction techniques for nonconnected data sets.

```
X1 = rng.multivariate_normal((-1, -1, -1), ((0.1, 0.1, 0.01), (0.1, 0.2, 0.1), (0.
↪01, 0.1, 0.1)), 300)
X2 = rng.multivariate_normal((1, 1, 1), ((0.1, 0.09, 0.01), (0.09, 0.2, 0.08), (0.
↪01, 0.08, 0.05)), 300)
X3 = rng.multivariate_normal((-1, 1, -1), ((0.2, 0.1, 0.1), (0.1, 0.4, 0.1), (0.1,
↪ 0.1, 0.08)), 500)
X4 = rng.multivariate_normal((1, 1, -1), ((0.1, 0.1, 0.01), (0.1, 0.2, 0.1), (0.
↪01, 0.1, 0.1)), 300)

X1 = (1 + X1 / np.abs(X1).max()) / 2
X2 = (1 + X2 / np.abs(X2).max()) / 2
```

(continues on next page)

(continued from previous page)

```
X3 = (1 + X3 / np.abs(X3).max()) / 2
X4 = (1 + X4 / np.abs(X4).max()) / 2

X = np.concatenate((X1, X2, X3, X4))
x = X[:, 0]
y = X[:, 1]
z = X[:, 2]

dists1 = np.sum(np.abs(X1 - X1.mean(axis=0)) ** 0.8, axis=1)
dists2 = np.sum(np.abs(X2 - X2.mean(axis=0)) ** 0.8, axis=1)
dists3 = np.sum(np.abs(X3 - X3.mean(axis=0)) ** 0.8, axis=1)
dists4 = np.sum(np.abs(X4 - X4.mean(axis=0)) ** 0.8, axis=1)

red1 = 1 - dists1 / dists1.max()
green1 = np.ones(X1.shape[0])
blue1 = np.zeros(X1.shape[0])

red2 = np.zeros(X2.shape[0])
green2 = 1 - dists2 / dists2.max()
blue2 = np.ones(X2.shape[0])

red3 = np.ones(X3.shape[0])
green3 = np.zeros(X3.shape[0])
blue3 = 1 - dists3 / dists3.max()

red4 = np.ones(X4.shape[0])
green4 = 1 - dists4 / dists4.max()
blue4 = np.zeros(X4.shape[0])

red = np.concatenate((red1.reshape(-1, 1), red2.reshape(-1, 1), red3.reshape(-1,␣
↪1), red4.reshape(-1, 1)), axis=0).reshape(-1)
green = np.concatenate((green1.reshape(-1, 1), green2.reshape(-1, 1), green3.
↪reshape(-1, 1), green4.reshape(-1, 1)), axis=0).reshape(-1)
blue = np.concatenate((blue1.reshape(-1, 1), blue2.reshape(-1, 1), blue3.reshape(-
↪1, 1), blue4.reshape(-1, 1)), axis=0).reshape(-1)
red = (255 * (red - red.min()) / (red - red.min()).max()).astype(int)
green = (255 * (green - green.min()) / (green - green.min()).max()).astype(int)
blue = (255 * (blue - blue.min()) / (blue - blue.min()).max()).astype(int)

# some noise
x = x + rng.normal(0, noise, x.size)
y = y + rng.normal(0, noise, y.size)
z = z + rng.normal(0, noise, z.size)

# plot
fig = go.Figure(layout_width=800, layout_height=600, layout_scene_aspectmode='cube
↪')
fig.add_trace(go.Scatter3d(
    x=x, y=y, z=z,
    mode='markers',
    marker={'size': 2, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
    hoverinfo = 'none'
))
fig.show()
```

```
<IPython.core.display.HTML object>
```

```
np.savez('clouds.npz', x=x, y=y, z=z, red=red, green=green, blue=blue)
```

39.5 PCA for Toy Examples

To compare results from nonlinear dimensionality reduction to the linear standard technique PCA we plot 2d PCA projections for all toy examples.

```
import sklearn.decomposition as decomposition
from plotly.subplots import make_subplots
```

```
data_files = ['omega.npz', 'sphere.npz', 'cube.npz', 'clouds.npz']

for file in data_files:

    loaded = np.load(file)
    x = loaded['x']
    y = loaded['y']
    z = loaded['z']
    red = loaded['red']
    green = loaded['green']
    blue = loaded['blue']

    pca = decomposition.PCA(2)
    U = pca.fit_transform(np.stack((x, y, z), axis=1))

    fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'scatter3d'}, {'type':
↪'xy'}]])
    fig.update_layout(width=1000, height=600, scene_aspectmode='cube')
    fig.add_trace(go.Scatter3d(
        x=x, y=y, z=z,
        mode='markers',
        marker={'size': 1.5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,
↪ green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=1)
    fig.add_trace(go.Scatter(
        x=U[:, 0], y=U[:, 1],
        mode='markers',
        marker={'size': 5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=2)
    fig.show()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

CHAPTER

FORTY

KERNEL PCA

Principal component analysis (PCA) contructs a new coordinate system in the data space such that variance is maximal along the first axis of this new coordinate system. Variance along the other axes decreases with increasing axis index. Coordinates of the data set with respect to the new coordinate system have zero covariance.

Such data adapted coordinate system at hand data dimensionality can be reduced by dropping axes of low variance or, in other words, by projecting onto the subspace spanned by the first few axes.

While classical PCA is a linear technique (data transformation by matrix multiplication) kernel PCA extends the basic idea to nonlinear transforms by exploiting the kernel trick. We already met the kernel trick when discussing support vector machines. In theory data is (nonlinearly) embedded into a higher dimensional space and PCA together with corresponding dimensionality reduction is performed there. But in practice, all computations are carried out in the original space by replacing inner products in high dimensions by special kernel functions.

In the following we always count eigenvalues with their multiplicity. So each real symmetric matrix has as many eigenvalues as it has rows and columns.

40.1 Recap of PCA

Denote by $X \in \mathbb{R}^{n\times m}$ the matrix containing all samples $x_1,\ldots,x_n$ as rows. Here m is the dimension of the data space. We assume that data is centered, that is,

$$\sum_{l=1}^{n} x_l = 0.$$

The (up to a factor $\frac{1}{n-1}$) covariance matrix $X^{\mathrm{T}}X \in \mathbb{R}^{m\times m}$ is symmetric and positive semi-definite. Thus, there exist m nonnegative numbers

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_m \geq 0 \qquad \text{(eigenvalues)}$$

and m mutually orthogonal normalized vectors $a_1,\ldots,a_m$ (eigenvectors) such that

$$X^{\mathrm{T}}X\,a_k = \lambda_k\,a_k \qquad \text{for } k = 1,\ldots,m.$$

The vectors a_k form the coordinate axes of the PCA coordinate system and the values λ_k are the variances (up to a factor $\frac{1}{n-1}$) of the data set along those axes. Denoting by $A \in \mathbb{R}^{m\times m}$ the orthogonal matrix of eigenvectors (as columns) the coordinate matrix $\tilde{X}$ with respect to the new coordinate system is given by

$$\tilde{X} = X\,A.$$

Equivalently, for each sample x_l and its transformed variant $\tilde{x}_l$ we have the relations

$$\tilde{x}_l = \begin{bmatrix} {x_l}^{\mathrm{T}}\,a_1 \\ \vdots \\ {x_l}^{\mathrm{T}}\,a_m \end{bmatrix} \qquad \text{and} \qquad x_l = \tilde{x}_l^{(1)}\,a_1 + \cdots + \tilde{x}_l^{(m)}\,a_m.$$

Dimensionality reduction now boils down to replacing vectors a_k corresponding to small λ_k by zero vectors. The transform from x_l to $\tilde{x}_l$ and back to x_l then does not yield x_l again, but its projection onto the subspace spanned by the remaining a_k.

40.2 PCA with Kernel Trick

Kernel PCA appeared first in the 1996 report Nonlinear Component Analysis as a Kernel Eigenvalue Problem[597]. Introducing a (nonlinear) mapping $\Phi:\mathbb{R}^m\to\mathbb{R}^q$ from the m dimensional data space into a much higher dimensional embedding space of dimension q with $q\geq m$ the authors propose to perform PCA based dimensionality reduction for $\{\Phi(x_1),\ldots,\Phi(x_n)\}$, the embedded data set. They have shown that PCA transformed data (in the q dimensional space) can be computed without touching q dimensional vectors and without expensive computations in q dimensions.

40.2.1 The Kernel PCA Theorem

Given the embedding function Φ as above we define the kernel matrix $K\in\mathbb{R}^{n\times n}$ by

$$K_{l,\lambda}:=\Phi(x_l)^{\mathrm{T}}\,\Phi(x_\lambda).$$

By $U\in\mathbb{R}^{n\times q}$ we denote the matrix containing $\Phi(x_1),\ldots,\Phi(x_n)$ as rows (by analogy with X for usual PCA in data space). Then one can show (and we will do this below) that

- **the eigenvalues of $U^{\mathrm{T}}\,U$ and K coincide,**
- **the eigenvectors of K contain the coordinates of $\Phi(x_1),\ldots,\Phi(x_n)$ with respect to the PCA coordinate system in the q dimensional space.**

In other words, we only have to compute eigenvalues and eigenvectors of the kernel matrix K when performing PCA for the embedded data. Size of K depends on number of samples, which might be large, but often much smaller than q. Remember: for polynomial kernel of degree 2 and 1000 dimensional data (images!) we would have $q=501501$ (see chapter on SVMs).

The first statement above requires some clarifying remarks.

- The (up to a factor) covariance matrix $U^{\mathrm{T}}\,U$ is of size $q\times q$. Thus, PCA yields q eigenvalues and q eigenvectors. But the embedded data lives in a subspace of dimension at most n. Thus, there are at most n nonzero eigenvalues for $U^{\mathrm{T}}\,U$.
- Writing $K=U\,U^{\mathrm{T}}$ we see that K is a symmetric matrix. Thus, the number of nonzero eigenvalues equals the rank of K (standard math result). Because $U\,U^{\mathrm{T}}$ and $U^{\mathrm{T}}\,U$ have identical rank (another standard math result), K and $U^{\mathrm{T}}\,U$ have the same number of nonzero eigenvalues.
- The maximum number of nonzero eigenvalues of K and $U^{\mathrm{T}}\,U$, and thus also the maximum number of nontrivial features after dimensionality reduction via kernel PCA, is $\min\{n,q\}$.

The second statement above on the eigenvectors of K requires some clarification, too. In addition, we should formulate it more precisely.

- The new data adapted coordinate system in q dimensions has q axes.
- Embedded data $\{\Phi(x_1),\ldots,\Phi(x_n)\}$ uses not more than the first $\min\{n,q\}$ axes. Coordinates for remaining axes are zero.
- In context of dimensionality reduction we are interested in the coordinates with respect to the first p axes only. Here p is small, in particular $p\leq\min\{n,q\}$.
- If $B\in\mathbb{R}^{q\times q}$ contains the eigenvectors of $U^{\mathrm{T}}\,U$ (in columns), then by analogy to usual PCA the coordinates with respect to the new coordinate system are given by

$$\tilde{U}:=U\,B.$$

- From the previous two items we see that the desired result from dimensionality reduction via kernel PCA is contained in the first p columns of $\tilde{U}$.
- The kernel PCA theorem states that the columns of $\tilde{U}$ are the eigenvectors of K.

Before we come to the proof, we have to think about centering embedded data. Remember: PCA is sensible for centered data only.

[597] https://www.face-rec.org/algorithms/Kernel/kernelPCA_scholkopf.pdf

40.2.2 Centering Embedded Data

Before applying PCA we have to center the data. Given the embedding function Φ we have to apply PCA to data

$$\Phi(x_1) - \frac{1}{n}\sum_{l=1}^{n}\Phi(x_l) \quad , \quad \ldots \quad , \quad \Phi(x_n) - \frac{1}{n}\sum_{l=1}^{n}\Phi(x_l).$$

The new embedding function

$$\Phi_{\text{center}}(x) := \Phi(x) - \frac{1}{n}\sum_{l=1}^{n}\Phi(x_l)$$

automatically yields centered embedded data.

We may rewrite this as

$$\Phi_{\text{center}}(x) := \Phi(x) - U^{\mathrm{T}} \begin{bmatrix} \frac{1}{n} \\ \vdots \\ \frac{1}{n} \end{bmatrix}$$

and introducing the matrix

$$M := \begin{bmatrix} \frac{1}{n} & \ldots & \frac{1}{n} \\ \vdots & & \vdots \\ \frac{1}{n} & \ldots & \frac{1}{n} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

we obtain

$${U_{\text{center}}}^{\mathrm{T}} = U^{\mathrm{T}} - U^{\mathrm{T}} M = U^{\mathrm{T}} (I - M).$$

The kernel corresponding to Φ_{center} then is

$$K_{\text{center}} = U_{\text{center}}\, {U_{\text{center}}}^{\mathrm{T}} = (I - M)\, U\, U^{\mathrm{T}} (I - M) = (I - M)\, K\, (I - M).$$

Centering the embedded data reduces to a simple transform of the kernel. If the kernel K is positive semi-definite (which is a standard assumption), then the new kernel K_{center} is positive semi-definite, too. So centering the embedded data set does not imply any troubles. The kernel trick still works.

Passing remark: The matrix $I - M$ is known as *centering matrix*. Multiplying a Matrix K from the right by $I - M$ centers its columns (sum of all columns is zero vector). Multiplication from the left centers its rows (sum of all rows is zero vector). Applying both multiplications yields the *double centered* version of K. The mean over all entries of a double centered matrix is zero, too.

40.2.3 Proof of Kernel PCA Theorem

We assume $n \leq q$ (which is the typical situation in practice) to avoid repreated use of min$\{n, q\}$ in the formulas. But the proof also works for $n > q$ if most n are replaced by min$\{n, q\}$. Further we assume that K has rank n, that is, all eigenvalues are nonzero. So we may write n instead of rank K to simplify formulas. All arguments also work for rank below n if considerations are restricted to nonzero eigenvalues of K.

Denoting the nonzero eigenvalues of the kernel matrix K by $\mu_1, \ldots, \mu_n$ and corresponding eigenvectors by $\omega_1, \ldots, \omega_n \in \mathbb{R}^n$ we define vectors $b_1, \ldots, b_n \in \mathbb{R}^q$ by

$$b_i := \frac{1}{\mu_i}\sum_{l=1}^{n}\omega_i^{(l)}\,\Phi(x_l), \qquad i = 1, \ldots, n.$$

To proof the theorem it suffices to show that

- $b_1, \ldots, b_n$ are eigenvectors of $U^{\mathrm{T}} U$ and $\mu_1, \ldots, \mu_n$ are corresponding eigenvalues,
- $\omega_1, \ldots, \omega_n$ contain the transformed coordinates, more precisely,

$$\omega_i = U\, b_i, \qquad i = 1, \ldots, n.$$

For the first statement we have to show $U^{\mathrm{T}}\,U\,b_i=\mu_i\,b_i$. This follows from the definition of b_i, some simple manipulations and the fact that ω_i is an eigenvector of K to eigenvalue μ_i:

$$\begin{aligned}
U^{\mathrm{T}}\,U\,b_i &= \frac{1}{\mu_i}\sum_{l=1}^{n}\omega_i^{(l)}\,U^{\mathrm{T}}\,U\,\Phi(x_l)=\frac{1}{\mu_i}\sum_{l=1}^{n}\omega_i^{(l)}\,U^{\mathrm{T}}\begin{bmatrix}\Phi(x_1)^{\mathrm{T}}\,\Phi(x_l)\\ \vdots\\ \Phi(x_n)^{\mathrm{T}}\,\Phi(x_l)\end{bmatrix}\\
&= \frac{1}{\mu_i}\sum_{l=1}^{n}\omega_i^{(l)}\sum_{\lambda=1}^{n}\underbrace{\Phi(x_\lambda)^{\mathrm{T}}\,\Phi(x_l)}_{\in\mathbb{R}}\,\Phi(x_\lambda)\\
&= \frac{1}{\mu_i}\sum_{l=1}^{n}\sum_{\lambda=1}^{n}\omega_i^{(l)}\,\Phi(x_\lambda)^{\mathrm{T}}\,\Phi(x_l)\,\Phi(x_\lambda)\\
&= \frac{1}{\mu_i}\sum_{\lambda=1}^{n}\left(\sum_{l=1}^{n}\omega_i^{(l)}\,\Phi(x_\lambda)^{\mathrm{T}}\,\Phi(x_l)\right)\,\Phi(x_\lambda)=\frac{1}{\mu_i}\sum_{\lambda=1}^{n}\left(\sum_{l=1}^{n}K_{\lambda,l}\,\omega_i^{(l)}\right)\,\Phi(x_\lambda)\\
&= \frac{1}{\mu_i}\sum_{\lambda=1}^{n}[K\,\omega_i]_\lambda\,\Phi(x_\lambda)\\
&= \frac{1}{\mu_i}\sum_{\lambda=1}^{n}\mu_i\,\omega_i^{(\lambda)}\,\Phi(x_\lambda)=\mu_i\,b_i.
\end{aligned}$$

The second statement is a direct consequence of the definition of b_i:

$$U\,b_i=\frac{1}{\mu_i}\sum_{l=1}^{n}\omega_i^{(l)}\,U\,\Phi(x_l)=\frac{1}{\mu_i}\sum_{l=1}^{n}\omega_i^{(l)}\,K_{\bullet,l}=\frac{1}{\mu_i}\,K\,\omega_i=\omega_i.$$

40.3 Kernel PCA with Scikit-Learn

Scikit-Learn provides the `KernelPCA`[598] class in its `decomposition` module.

```python
import numpy as np
import sklearn.decomposition as decomposition
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```python
data_files = ['omega.npz', 'sphere.npz', 'cube.npz', 'clouds.npz']

for file in data_files:

    loaded = np.load(file)
    x = loaded['x']
    y = loaded['y']
    z = loaded['z']
    red = loaded['red']
    green = loaded['green']
    blue = loaded['blue']

    kpca = decomposition.KernelPCA(2, kernel='rbf', gamma=5)
    U = kpca.fit_transform(np.stack((x, y, z), axis=1))

    fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'scatter3d'}, {'type':
↪'xy'}]])
    fig.update_layout(width=1000, height=600, scene_aspectmode='cube')
    fig.add_trace(go.Scatter3d(
        x=x, y=y, z=z,
        mode='markers',
```

(continues on next page)

598 https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html

(continued from previous page)

```
        marker={'size': 1.5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,
↪ green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=1)
    fig.add_trace(go.Scatter(
        x=U[:, 0], y=U[:, 1],
        mode='markers',
        marker={'size': 5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=2)
    fig.show()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

40.4 Why Kernel PCA?

Kernel PCA only yields useful results if the embedding map Φ somehow unroles the nonlinear structure of the data set such that in higher dimensions the data set is almost linear. Kernel PCA is rarely used in practise, because one does not know how to choose the kernel. But we will see in subsequent chapters that other nonlinear dimensionality reduction techniques turn out to be equivalent to kernel PCA with specific data set adapted kernel.

CHAPTER

FORTYONE

MULTIDIMENSIONAL SCALING

We already considered techniques for dimensionality reduction in the context of feature reduction. There the aim was to construct more expressive features and to remove unnecessary ones. Now the focus is on visualization, that is, reduction of data sets to 2 or 3 dimensions. This way me may get a better understanding of a data set and we also may check results obtained from clustering methods.

We already met two techniques for dimensionality reduction:

- (Kernel) principal component analysis (PCA), which simply projects a data set orthogonally onto a lower dimensional linear manifold,
- Autoencoders, which try to find two mappings between the high and a low dimensional space such that their composition resamples the data set in the high dimensional space as good as possible.

PCA is a linear dimensionality reduction technique, because it performs a linear transform (multiplication by matrix). Kernel PCA and autoencoders are nonlinear dimensionality reduction techniques, because the mapping from high to low dimensions cannot be expressed by matrix multiplication.

Multidimensional scaling (MDS) refers to a third fundamental concept for dimensionality reduction. Here we try to find a set of points in low dimensions such that pairwise distances are as close as possible to pairwise distances in the high dimensional space. MDS comes in several variants differing in the choice of the distance measure and also in the weighting of pairwise distances.

Related projects:

- *Color Perception* (page 991)
- *Forest Fires* (page 995)

41.1 Abstract Mathematical Formulation

To make things precise, denote by $(x_1,\ldots,x_n)$ the data set in the high dimensional space $\mathbb{R}^m$ and by $(u_1,\ldots,u_n)$ a set of points in a lower dimensional space $\mathbb{R}^p$. Note that both sets are of equal size n. Further let d_m and d_p be similarity measures in high and low dimensions, respectively. By $w_{l,\lambda}\in(0,\infty)$, $l,\lambda=1,\ldots,n$ we denote some weights. Then MDS aims at solving

$$\sum_{l=1}^{n}\sum_{\lambda=1}^{n} w_{l,\lambda}\left(d_m(x_l,x_\lambda)-d_p(u_l,u_\lambda)\right)^2\to\min_{u_1,\ldots,u_n}.$$

Squaring is somewhat arbitrary here. We could apply any positive and increasing function to the difference of pairwise distances. But in each conrete MDS variant the square is the most fortunate choice, because it ensures differentiability and simplifies computation of gradients.

MDS solely relies on similarities $d_m(x_l,x_\lambda)$ between samples and does not touch samples directly. This observation allows for applications with arbitrary types of data (text data, for instance) as long as there is some notion of similarity. In addition, the similarity measure in high dimensions is not required to be some precise mathematical construct. Human scoring is possible, too, for instance.

41.2 Metric MDS and Sammon's Method

In metric MDS the low dimensional similarity measure is the squared Euclidean distance

$$d_p(u_l, u_\lambda) = |u_l - u_\lambda|^2.$$

So metric MDS tries to preserve pairwise Euclidean distances (if distances in high dimensions are Euclidean, too).

The minimization problem has to be solved numerically by gradient descent or Newton-type methods (iterative methods using second order derivatives). Results may be inaccurate due to local minima or saddle points. Starting guess may be determined by PCA or chosen at random.

Without weights (all weights set to 1) large distances will dominate the objective, because an error in distance fitting of 10 per cent has more influence on the objective for large distances than for small distances. Thus, without weights the local structure of the data set is not well reconstructed in low dimensions.

Usually one is more interested in the local structure than in the global one. For instance, the shape of a cluster or the boundary region between closely spaced clusters are more interesting than the correct distance between clusters far apart from each other.

Weighting by inverse distances solves this issue and puts emphasis on local structures. Typically, weights are

$$w_{l,\lambda} = \frac{1}{d_m(x_l, x_\lambda)}.$$

This weighted variant of metric MDS is known as *Sammon's method*.

```
import numpy as np
import sklearn.manifold as manifold
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```
data_files = ['omega.npz', 'sphere.npz', 'cube.npz', 'clouds.npz']

for file in data_files:

    loaded = np.load(file)
    x = loaded['x']
    y = loaded['y']
    z = loaded['z']
    red = loaded['red']
    green = loaded['green']
    blue = loaded['blue']

    mds = manifold.MDS(2, normalized_stress='auto')
    U = mds.fit_transform(np.stack((x, y, z), axis=1))

    fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'scatter3d'}, {'type':
↪'xy'}]])
    fig.update_layout(width=1000, height=600, scene_aspectmode='cube')
    fig.add_trace(go.Scatter3d(
        x=x, y=y, z=z,
        mode='markers',
        marker={'size': 1.5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,
↪ green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=1)
    fig.add_trace(go.Scatter(
        x=U[:, 0], y=U[:, 1],
        mode='markers',
        marker={'size': 5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
```

(continues on next page)

(continued from previous page)

```
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=2)
    fig.show()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

41.3 Classical MDS

Like metric MDS classical MDS aims at fitting Euclidean distances, but avoids iterative minimization. We will see that applying a linear transform to the Euclidean distance matrices in both low and high dimensions before fitting allows for analytical minimization. Here, by Euclidean distance matrix we denote a matrix containing in row i and column j the squared Euclidean distance between sample i and sample j.

We will discuss several aspects step by step:

- the transform applied to Euclidean distances to simplify minimization,
- reconstruction of the (lower dimensional) data set from transformed distances
- formulation and solution of the minimization problem,
- modifications to allow for non-Euclidean distances in high dimensions,
- relations to PCA and kernel PCA.

41.3.1 From Distances to Inner Products

Let $z_1,\ldots,z_n\in\mathbb{R}^q$ be a collection of points in q dimensions, think of $q=m$ with $z_l=x_l$ or $q=p$ with $z_l=u_l$. Corresponding Euclidean distance matrix $D\in\mathbb{R}^{n\times n}$ is given by

$$D_{l,\lambda}:=|z_l-z_\lambda|^2$$

and the inner product matrix $S\in\mathbb{R}^n$ of the centered data set $z_1-\bar{z},\ldots,z_n-\bar{z}$ with $\bar{z}:=\frac{1}{n}\sum_{l=1}^n z_l$ is defined by

$$S_{l,\lambda}:=(z_l-\bar{z})^{\mathrm{T}}\,(z_\lambda-\bar{z}).$$

With

$$M:=\begin{bmatrix}\frac{1}{n}&\cdots&\frac{1}{n}\\ \vdots&&\vdots\\ \frac{1}{n}&\cdots&\frac{1}{n}\end{bmatrix}\in\mathbb{R}^{n\times n}$$

we now show

$$S=-\frac{1}{2}\,(I-M)\,D\,(I-M),$$

that is, double centering the Euclidean distance matrix yields the inner product matrix of the centered data set (up to some constant factor).

From the definition of matrix multiplication and then from

$$|z_l-z_\lambda|^2=|z_l|^2+|z_\lambda|^2-2\,{z_l}^{\mathrm{T}}\,z_\lambda$$

we see

$$\begin{aligned}[(I-M)\,D\,(I-M)]_{l,\lambda} &= D_{l,\lambda}-[M\,D]_{l,\lambda}-[D\,M]_{l,\lambda}+[M\,D\,M]_{l,\lambda}\\ &= D_{l,\lambda}-\frac{1}{n}\sum_{i=1}^{n}D_{i,\lambda}-\frac{1}{n}\sum_{j=1}^{n}D_{l,j}+\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}D_{i,j}\\ &= |z_l|^2+|z_\lambda|^2-2\,{z_l}^{\mathrm{T}}\,z_\lambda-\left(|z_\lambda|^2+\frac{1}{n}\sum_{i=1}^{n}(|z_i|^2-2\,{z_i}^{\mathrm{T}}\,z_\lambda)\right)\\ &\quad-\left(|z_l|^2+\frac{1}{n}\sum_{j=1}^{n}(|z_j|^2-2\,{z_l}^{\mathrm{T}}\,z_j)\right)+\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(|z_i|^2+|z_j|^2-2\,{z_i}^{\mathrm{T}}\,z_j).\end{aligned}$$

Summands $|z_l|^2$ and $|z_\lambda|^2$ each appear twice with different sign, thus vanish. The double sum can be simplified to

$$\begin{aligned}\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(|z_i|^2+|z_j|^2-2\,{z_i}^{\mathrm{T}}\,z_j) &= \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}|z_i|^2+\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}|z_j|^2-\frac{2}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}{z_i}^{\mathrm{T}}\,z_j\\ &= \frac{1}{n}\sum_{i=1}^{n}|z_i|^2+\frac{1}{n}\sum_{j=1}^{n}|z_j|^2-2\left(\frac{1}{n}\sum_{i=1}^{n}z_i\right)^{\mathrm{T}}\left(\frac{1}{n}\sum_{j=1}^{n}z_j\right),\end{aligned}$$

so that the sums over $|z_i|^2$ and $|z_j|^2$ cancel out, too. Thus, we have

$$\begin{aligned}[(I-M)\,D\,(I-M)]_{l,\lambda} &= -2\,{z_l}^{\mathrm{T}}\,z_\lambda+\frac{2}{n}\sum_{i=1}^{n}{z_i}^{\mathrm{T}}\,z_\lambda+\frac{2}{n}\sum_{j=1}^{n}{z_l}^{\mathrm{T}}\,z_j-2\,\bar{z}^{\mathrm{T}}\,\bar{z}\\ &= -2\,{z_l}^{\mathrm{T}}\,z_\lambda+2\,\bar{z}^{\mathrm{T}}\,z_\lambda+2\,{z_l}^{\mathrm{T}}\,\bar{z}-2\,\bar{z}^{\mathrm{T}}\,\bar{z}\\ &= -2\,(z_l-\bar{z})^{\mathrm{T}}\,(z_\lambda-\bar{z})=-2\,S_{l,\lambda}.\end{aligned}$$

41.3.2 From Inner Products to Data

From an inner product matrix S we want to reconstruct original data $z_1,\dots,z_n$. As we will see, the mean $\bar{z}$ cannot be reconstructed from S. Choosing an arbitrary $\bar{z}$ amounts in a translation of the whole data compared to the original one. In addition there will be some freedom in rotating and mirrowing the reconstructed data set.

The data set can be reconstructed from the eigendecomposition of S. The matrix S is symmetric and positive semi-definite. So there exist n nonnegative numbers $\lambda_1\geq\dots\geq\lambda_n\geq 0$ (eigenvalues of S) and n vectors $a_1,\dots,a_n\in\mathbb{R}^n$ (eigenvectors of S) with

$$S=A\,\Lambda\,A^{\mathrm{T}},$$

where A contains the eigenvectors as columns and Λ is the diagonal matrix of eigenvalues.

If $Z\in\mathbb{R}^{n\times q}$ contains $z_1,\dots,z_n$ as rows, then we have

$$S=((I-M)\,Z)\,((I-M)\,Z)^{\mathrm{T}}=(I-M)\,Z\,Z^{\mathrm{T}}\,(I-M).$$

The rank of Z is at most q and, thus, the rank of S is at most q, too. So all but the first q eigenvalues of S are zero. Here we assume $q\leq n$. The case $q>n$ is not of interest to us, because for MDS we will have $q=p$ (dimension of low dimensional space, in practice 2 or 3).

With an arbitrary orthonormal matrix $R\in\mathbb{R}^{q\times q}$ (rotation and/or mirrowing) we set

$$\tilde{z}_l:=R\begin{bmatrix}\sqrt{\lambda_1}\,a_1^{(l)}\\ \vdots\\ \sqrt{\lambda_q}\,a_q^{(l)}\end{bmatrix}\qquad\text{for } l=1,\dots,n.$$

Denoting the diagonal matrix of square roots of the eigenvalues by $\lambda^{\frac{1}{2}}$ we see

$$\tilde{Z}=A\,\Lambda^{\frac{1}{2}}\,R^{\mathrm{T}}$$

and

$$\tilde{Z}\,\tilde{Z}^{\mathrm{T}} = A\,\Lambda^{\frac{1}{2}}\,R^{\mathrm{T}}\,R\,\Lambda^{\frac{1}{2}}\,A^{\mathrm{T}} = A\,\Lambda\,A^{\mathrm{T}} = S.$$

That is, S is the inner product matrix of $\tilde{z}_1, \dots, \tilde{z}_n$.

Now from

$$|\tilde{z}_l - \tilde{z}_\lambda|^2 = |\tilde{z}_l|^2 + |\tilde{z}_\lambda|^2 - 2\,\tilde{z}_l^{\mathrm{T}}\,\tilde{z}_\lambda = S_{l,l} + S_{\lambda,\lambda} - 2\,S_{l,\lambda} = |z_l - \bar{z} - (z_\lambda - \bar{z})|^2 = |z_l - z_\lambda|^2$$

we see that the reconstructed data set $\tilde{z}_1, \dots, \tilde{z}_n$ and the original data set $z_1, \dots, z_n$ have identical distance matrices.

41.3.3 The Minimization Problem

Classical MDS fits inner products, not distances. The following steps lead from a Euclidean distance matrix $D \in \mathbb{R}^{n\times n}$ in high dimensions to a set of points $u_1, \dots, u_n \in \mathbb{R}^p$ in low dimensions:

- Calculate the inner products matrix $S \in \mathbb{R}^{n\times n}$ in high dimensions:

$$S = -\frac{1}{2}\,(I - M)\,D\,(I - M).$$

Here we do not have to know the underlying data set $x_1, \dots, x_n \in \mathbb{R}^m$ explicitly, because inner products are determined by pairwise Euclidean distances (up to centering).

- Solve

$$\sum_{l=1}^{n}\sum_{\lambda=1}^{n}(S_{l,\lambda} - T_{l,\lambda})^2 \to \min_{T\in\mathbb{R}^{n\times n}} \qquad \text{considering only symmetric } T \text{ of rank } p.$$

- Interpret the optimal T as inner product matrix of n points in $\mathbb{R}^p$ and reconstruct corresponding points $u_1, \dots, u_n$.

In contrast to metric MDS classical MDS does not fit pairwise distances directly, but transforms distances to inner products and fits those inner products. Distance matrices and inner product matrices carry identical information (up to translation, rotation, mirrowing of the data set), but due to different objective functions both variants of MDS yield different results. Metric MDS minimizes the means squared error (MSE) of pairwise distances:

$$\sum_{l=1}^{n}\sum_{\lambda=1}^{n}\left(|x_l - x_\lambda|^2 - |u_l - u_\lambda|^2\right)^2.$$

Classical MDS minimizes (assuming centered data in high dimensions):

$$\sum_{l=1}^{n}\left(|x_l|^2 - |u_l|^2\right)^2 + \sum_{l=1}^{n}\sum_{\substack{\lambda=1\\ \lambda\neq l}}^{n}\left({x_l}^{\mathrm{T}}\,x_\lambda - {u_l}^{\mathrm{T}}\,u_\lambda\right)^2,$$

which is the MSE of vetor lengths (diagonal of inner product matrix) plus the MSE of angles (inner products are cosines of angles multiplied by both vector lengths).

It remains to answer the question how to solve the above minimization problem. We aim at a set of points in $\mathbb{R}^p$. So corresponding inner product matrix has rank of at most p. That's the reason why we restrict optimization to symmetric matrices of rank p. A standard result from linear algebra (Eckart-Young-Mirsky theorem)[599] tells us, that we have to look at the eigendecomposition of S:

$$S = A\,\Lambda\,A^{\mathrm{T}} \qquad \text{cf. above.}$$

Then the optimal T is (assuming $p \leq n$)

$$T = B\,\Theta\,B^{\mathrm{T}},$$

where $B \in \mathbb{R}^{n\times p}$ contains the first p eigenvectors of S as columns and $\Theta \in \mathbb{R}^{p\times p}$ is the diagonal matrix of the highest p eigenvalues of S. So solving the minimization problem reduces to an eigenvalue problem for S.

As a by-product of minimization we get the eigendecomposition of T, which is required for reconstructing the data set $u_1, \dots, u_n$ in low dimensions from T. Eigenvalues and eigenvectors of T coincide with the first p eigenvalues and eigenvectors of S.

[599] https://en.wikipedia.org/wiki/Low-rank_approximation#Basic_low-rank_approximation_problem

41.3.4 Non-Euclidean Distances

Classical MDS solely relies on the distance matrix D in high dimensions. The assumption that distances are Euclidean ensures that the transformed distance matrix $S=-\frac{1}{2}\,(I-M)\,D\,(I-M)$ is symmetric and positive semi-definite. So we may replace the assumption of Euclidean distances by the more direct assumption that the transformed distance matrix S is symmetric and positive semi-definite.

D may contain arbitrary (non-Euclidean) pairwise distances as long as $-\frac{1}{2}\,D$ is symmetric and positive semi-definite. So it can be regarded as a kernel matrix resulting from inner products of nonlinearly transformed data. The transform from $-\frac{1}{2}\,D$ to S is simple double centering, which is equivalent to centering the transformed data set.

We may even drop the assumption that S has to be positive semi-definite. If S is non-definite, some eigenvalues will be negative. When approximating S by some T of rank p we only consider the p largest positive eigenvalues. As long as the absolute values of negative eigenvalues is small, the error induced by ignoring them is not higher than the error resulting from low-rank approximation (that is, from dropping small positive eigenvalues). If D is based on some kind of distance, possible negative eigenvalues will be small.

As long as D is symmetric and is related to some almost arbitrary kind of distance classical MDS is applicable.

41.3.5 Relation to PCA and Kernel PCA

For centered data and Euclidean distances we have $S=X\,X^{\mathrm{T}}$. Else, S can be regarded as a double centered kernel matrix originating from a kernel matrix $-\frac{1}{2}\,D$. The transform from S to $u_1,\dots,u_n$ in classical MDS coincides with the transform from K to the data set's first p coordinates w.r.t. to the kernel PCA coordinate system. In both cases we use the same parts of the eigendecomposition of S or K, respectively.

Classical MDS and kernel PCA are two different motivations for one and the same algorithm. In case of centered data and Euclidean distances classical MDS and (non-kernel) PCA coincide. Classical Euclidean MDS originated in the 1950s. Kernel PCA appeared in 1996 and the connection to classical MDS had been discovered a few years later.

41.4 Non-Metric MDS

For the sake of completeness we mention a third variant of MDS known as non-metric MDS, but we do not go into the details. Non-metric MDS does not focus on getting the distances right, but only on preserving the ordering of distances.

The similarity measure in low dimensions is

$$d(u_l,u_\lambda)=f(|u_l-u_\lambda|^2).$$

with some monotonically increasing function f.

Next to $u_1,\dots,u_n$ the function f is a variable in the optimization process. A typical numerical approach is to alternate optimization steps for $u_1,\dots,u_n$ and f. Concrete algorithms following this idea are *smallest space analysis (SSA)* and *Shepard-Kruskal algorithm*[600] (German Wikipedia).

41.5 Isomap

Isomap assumes that the data set lies on low dimensional manifold in the high dimensional space. Instead of Euclidean distances it calculates (approximate) geodesic distances with respect to the manifold. In other words, it looks for the shortest distance between two points if a traveller between both points is not allowed to leave the manifold. Pairwise geodesic distances at hand classical MDS is applied to the distance matrix.

The only thing we have to discuss is how to find (approximate) geodesic distances. This requires two steps:

[600] https://de.wikipedia.org/wiki/Multidimensionale_Skalierung#Shepard-Kruskal_Algorithmus

- Create a neighborhood graph. The nodes are the samples. A node is connected to another node by an (undirected) edge if the other node belongs to the k nearest neighbors of the first one for some prescribed k. Each edge is assigned a weight, which equals the Euclidean distance between samples connected by the edge.
- Get length of shortest path in neighborhood graph between each pair of nodes. Length of a path is the sum of all edge weights belonging to the path. Finding shortest paths in a graph is a standard task and can be solved by Dijkstra's algorithm[601].

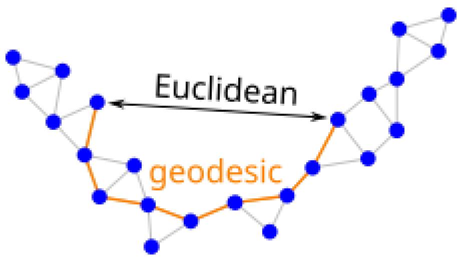


Fig. 41.1: Isomap uses approximate geodesic distances instead of Euclidean distances.

An alternative to the k nearest neighbors is to connect a node to all other nodes within a prescribed distance.

Next to high computational costs a major problem with Isomap is that there may be so called short-circuit errors. That is, the neighborhood graph contains edges betwenn non-neighboring points on the manifold. This happens especially for large k or sparse data sets.

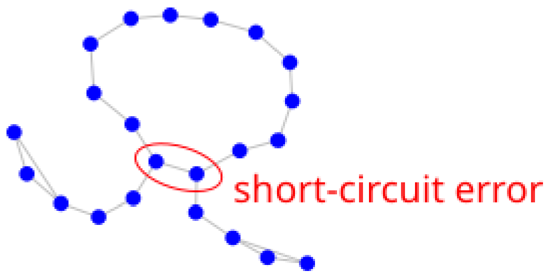


Fig. 41.2: Isomap may suffer from short-circuit errors.

Scikit-Learn has the `Isomap`[602] class in the `manifold` module.

```
data_files = ['omega.npz', 'sphere.npz', 'cube.npz', 'clouds.npz']

for file in data_files:

    loaded = np.load(file)
    x = loaded['x']
    y = loaded['y']
    z = loaded['z']
    red = loaded['red']
    green = loaded['green']
    blue = loaded['blue']

    im = manifold.Isomap(n_components=2, n_neighbors=5)
    U = im.fit_transform(np.stack((x, y, z), axis=1))

    fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'scatter3d'}, {'type':
↪'xy'}]])
    fig.update_layout(width=1000, height=600, scene_aspectmode='cube')
    fig.add_trace(go.Scatter3d(
        x=x, y=y, z=z,
```

(continues on next page)

[601] https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
[602] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.Isomap.html

(continued from previous page)

```
        mode='markers',
        marker={'size': 1.5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,
↪ green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=1)
    fig.add_trace(go.Scatter(
        x=U[:, 0], y=U[:, 1],
        mode='markers',
        marker={'size': 5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=2)
    fig.show()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
/home/jef19jdw/anaconda3/envs/ds_book/lib/python3.10/site-packages/sklearn/
 ↪manifold/_isomap.py:373: UserWarning:

The number of connected components of the neighbors graph is 2 > 1. Completing␣
 ↪the graph to fit Isomap might be slow. Increase the number of neighbors to␣
 ↪avoid this issue.

/home/jef19jdw/anaconda3/envs/ds_book/lib/python3.10/site-packages/scipy/sparse/
 ↪_index.py:103: SparseEfficiencyWarning:

Changing the sparsity structure of a csr_matrix is expensive. lil_matrix is␣
 ↪more efficient.
```

```
<IPython.core.display.HTML object>
```

CHAPTER

FORTYTWO

LOCALLY LINEAR EMBEDDING

The idea of locally linear embedding (LLE) appeared at the same time (and journal) as Isomap, uses Euclidean distances locally only like Isomap, and results in an eigenvalue problem (again, like Isomap). But details and motivation differ.

The basic assumption is, that the data set lies on a low dimensional manifold which can be decomposed into (approximately) linear snippets. LLE tries to arange those snippets in low (often 2) dimensions without modifying their neighborhood relations.

Finding an LLE requires two steps:

- Represent each sample as an affine combination of its neighbors.
- Arrange low dimensional points such that they can be represented by the same affine combination of neighbors as in high dimensions.

42.1 Local Affine Combinations

For each sample x_l the k nearest neighbors $x_{\lambda_1},\ldots,x_{\lambda_k}$ are determined. These neighbors span an *affine manifold* (that is, a translated subspace)

$$\{a_1\,x_{\lambda_1}+\cdots+a_k\,x_{\lambda_k}\,:\,a_1,\ldots,a_k\in\mathbb{R},\;a_1+\ldots+a_k=1\}.$$

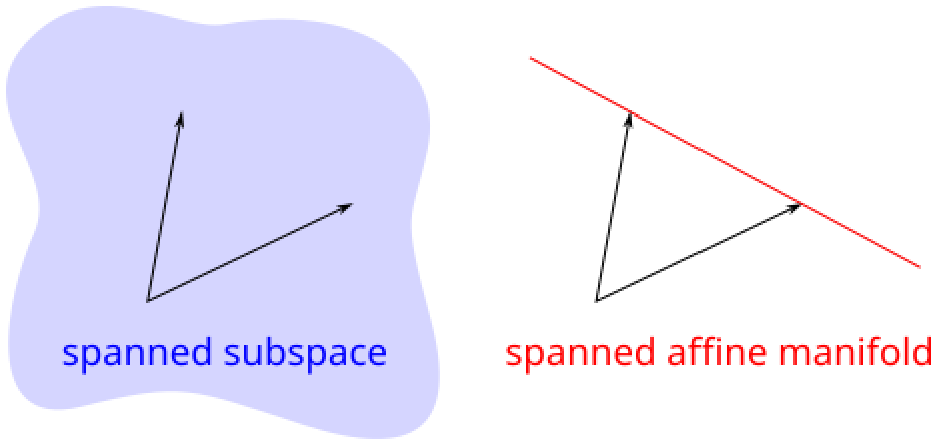


Fig. 42.1: Two vectors may span a two-dimensional subspace or a one-dimensional affine manifold depending on the set of coefficients considered.

Projecting x_l orthogonally onto the affine manifold spanned by its neighbors yields representation

$$x_l\approx w_{l,\lambda_1}\,x_{\lambda_1}+\cdots+w_{l,\lambda_k}\,x_{\lambda_k}.$$

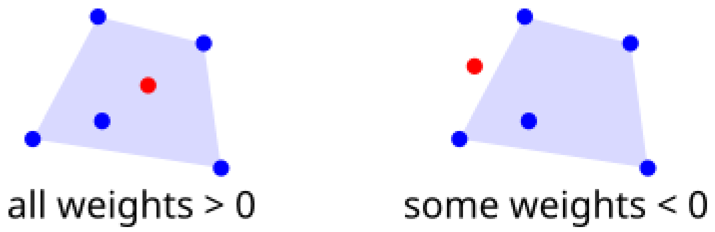


Fig. 42.2: All coefficients in an affine combination are nonnegative if and only if the resulting point belongs to the convec hull of the points being combined.

The coefficients $w_{l,\lambda_1},\ldots,w_{l,\lambda_k}$ may be regarded as weights, because their sum is 1 and often they are positive if x_l is surrounded by its neighbors (more precicely, if x_l lies in the convex hull of its neighbors.

If the data set is 'locally linear', then x_l equals the weighted sum and the error induced by representing samples as weighted sums of their neighbors is zero. The more nonlinear a data set behaves locally, the less reliable the low dimensional representation obtained via LLE.

This first step of LLE yields at most n^2 weights $w_{l,\lambda}$. Weights $w_{l,\lambda}$ for which corresponding samples aren't neighbors are set to zero. Weights are not symmetric, that is, $w_{l,\lambda}\neq w_{\lambda,l}$ in general.

To get the weights one first determines the indices $\lambda_1(l),\ldots,\lambda_k(l)$ of the k nearest neightbors for all samples x_l. Then one solves the constrained minimization problem

$$\begin{aligned}&\sum_{l=1}^{n}\left(w_{l,\lambda_1(l)}\,x_{\lambda_1(l)}+\cdots+w_{l,\lambda_k(l)}\,x_{\lambda_k(l)}-x_l\right)^2\to\min_{w_{l,\lambda}}\\&\text{with constraints}\quad w_{l,\lambda_1(l)}+\cdots+w_{l,\lambda_k(l)}=1\quad\text{for } l=1,\ldots,n\\&\text{(unused } w_{l,\lambda}\text{ are set to zero).}\end{aligned}$$

This minimization problem is quadratic and can be solved numerically in different efficient ways, for instance, by solving a system of linear equations.

42.2 Low Dimensional Fitting

Given weights $w_{l,\lambda}$ the second step of LLE is to find points $u_1,\ldots,u_n\in\mathbb{R}^p$ in low dimensions which solve

$$\begin{aligned}&\sum_{l=1}^{n}\left(w_{l,\lambda_1(l)}\,u_{\lambda_1(l)}+\cdots+w_{l,\lambda_k(l)}\,u_{\lambda_k(l)}-u_l\right)^2\to\min_{u_1,\ldots,u_n}\\&\text{with constraints}\quad u_1+\cdots+u_n=0\quad\text{and}\quad\text{covariance matrix is identity.}\end{aligned}$$

The objective is the same as in the first step, but now in low dimensions and with fixed weights. Thus, LLE tries to reconstruct the local linear structure from high dimensions in low dimensions. Without constraints choosing all low dimensional points to be zero would solve the minimization problem. The covariance constraint excludes such trivial solutions by requiring that featurewise variance is 1. Thus, solutions have to be scattered in space to some extent. Covariance of zero prevents some other trivial solutions and avoids solution non-uniqueness due to rotations. Non-uniqueness due to translations is avoided by the mean zero contraint.

The solution to the minimization problem can be obtained from an eigenvalue problem similar to kernel PCA.

42.3 LLE with Scikit-Learn

Scikit-Learn has the `LocallyLinearEmbedding`[603] class in the `manifold` module.

```
import numpy as np
import sklearn.manifold as manifold
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```
data_files = ['omega.npz', 'sphere.npz', 'cube.npz', 'clouds.npz']

for file in data_files:

    loaded = np.load(file)
    x = loaded['x']
    y = loaded['y']
    z = loaded['z']
    red = loaded['red']
    green = loaded['green']
    blue = loaded['blue']

    lle = manifold.LocallyLinearEmbedding(n_components=2, n_neighbors=30)
    U = lle.fit_transform(np.stack((x, y, z), axis=1))

    fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'scatter3d'}, {'type':
↪'xy'}]])
    fig.update_layout(width=1000, height=600, scene_aspectmode='cube')
    fig.add_trace(go.Scatter3d(
        x=x, y=y, z=z,
        mode='markers',
        marker={'size': 1.5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,
↪ green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=1)
    fig.add_trace(go.Scatter(
        x=U[:, 0], y=U[:, 1],
        mode='markers',
        marker={'size': 5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=2)
    fig.show()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[603] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.LocallyLinearEmbedding.html

CHAPTER

FORTYTHREE

STOCHASTIC NEIGHBOR EMBEDDING

Up to now we only considered deterministic dimensionality reduction methods. Stochastic neighbor embedding (SNE) does not use neighborhood relations and distances directly. Instead it estimates the probability that two samples are neighbors in high dimensions from pairwise distances. Then it tries to find points in low dimensions which yield identical neighborhood probabilities.

SNE appeared in 2002 and several variants have been developed since then. The most prominent one is known as t-SNE and originated in 2008. Like MDS, SNE does not require direct knowledge of the underlying data set $x_1, \ldots, x_n$, but only uses pairwise distances.

Related projects:

- *MNIST Character Recognition* (page 931)
  - *t-SNE for QMNIST* (page 945)

43.1 Basic idea

43.1.1 Probabilities in High Dimensions

Given pairwise distances $D_{l,\lambda}$ $(l, \lambda = 1, \ldots, n)$ in high dimensions we may fix a sample x_l and assign to all other samples probabilities $\tilde{p}_{l,\lambda}$ reflecting the neighborhood relations to x_λ. The closer x_λ to x_l the higher $\tilde{p}_{l,\lambda}$. We may use Gaussian probabilities:

$$\tilde{p}_{l,\lambda} := \frac{\mathrm{e}^{\frac{-(x_l-x_\lambda)^2}{2\,\sigma_l^2}}}{\sum\limits_{\substack{i=1\\ i\neq l}}^{n} \mathrm{e}^{\frac{-(x_l-x_i)^2}{2\,\sigma_l^2}}}, \qquad \lambda \neq l \qquad \text{and} \qquad \tilde{p}_{l,l} := 0.$$

The parameter σ_l is chosen numerically (by bisection, for instance) to fix the entropy of the neighborhood distribution of x_l at some prescribed value, which is independent of l. The entropy here is

$$-\sum_{\lambda=1}^{n} \tilde{p}_{l,\lambda} \, \log \tilde{p}_{l,\lambda}.$$

If σ_l is too small there will be only few neighbors of x_l with high probabilities. Then entropy is very low. If σ_l is too large many neighbors of x_l will have similar probabilities. Then entropy is high. Adjusting σ_l to get some prescibed medium entropy ensures that the probability distribution for the neighbors takes the data set's local density into account.

In general $\tilde{p}_{l,\lambda} \neq \tilde{p}_{\lambda,l}$. To enforce symmetry (which simplifies some computations) we set

$$p_{l,\lambda} := \frac{\tilde{p}_{l,\lambda} + \tilde{p}_{\lambda,l}}{2\,n}.$$

The sum of all these n^2 values equals 1. Instead of n probability distributions (one for each l) we now only have one distribution and this distribution is symmetric.

All in all we converted pairwise distances to pairwise probabilities that two samples are neighbors. But the conversions is not direct by proportionality, but also takes local density of the data set into account.

43.1.2 Probabilities in Low Dimensions

Given points $u_1, \ldots, u_n$ in low dimensions we may use the same construction as in high dimensions to obtain probabilities $q_{l,\lambda}$. Distances are Euclidean and σ-values can be set to one to get uniform local densities in low dimensions.

There exist different involved reasons to choose non-Gaussian probabilities in low dimensions. Several choices have been proposed yielding a range of different SNE variants. Below we give the details for a variant known as t-SNE.

43.1.3 Fitting Probabilites

SNE tries to find low dimensional points $u_1, \ldots, u_n$ such that the corresponding probability distribution fits the high dimensional probability distribution as good as possible. Instead of using MSE of both sets of probability values SNE prefers the Kullback-Leibler divergence[604]:

$$\sum_{l=2}^{n}\sum_{\lambda=1}^{l-1} p_{l,\lambda}\,\log\frac{p_{l,\lambda}}{q_{l,\lambda}(u_1,\ldots,u_n)} \to \min_{u_1,\ldots,u_n}.$$

This minimization problem can be solved numerically via gradient descent. There also exist some more efficient methods adapted to the specifics of SNE.

43.2 t-SNE

The t-SNE variant of SNE defines low dimensional probabilites $q_{l,\lambda}$ based on the Student's t-distribution[605]. There are two reasons for this choice:

- It's computationally more efficient because no exponentiation is required.
- Student's t-distribution decays slower than a Gaussian distribution, which compensates (to some degree) for effects caused by the curse of dimensionality. A ball around a sample in high dimensions has much higher volume than a same sized ball (same radius) in low dimensions. To get similar sample densities (neighbors per volume) in both high and low dimensions in low dimensions we have to assign higher probabilities to more distant samples than in high dimensions. Else the lower dimensional embedding would look much denser than the original data set and clusters may get indistinguishable.

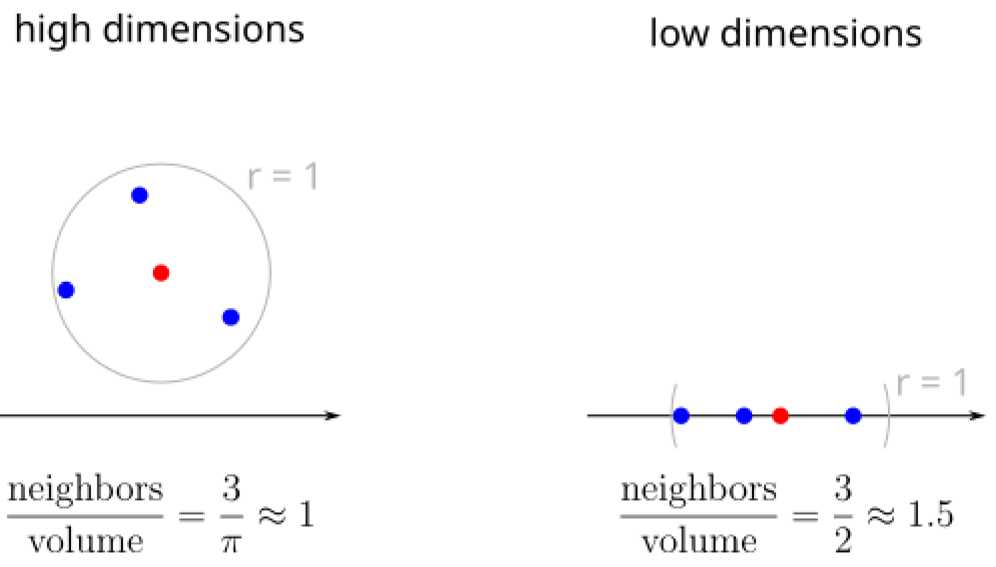

high dimensions
low dimensions
r = 1
r = 1
$\frac{\text{neighbors}}{\text{volume}} = \frac{3}{\pi} \approx 1$
$\frac{\text{neighbors}}{\text{volume}} = \frac{3}{2} \approx 1.5$

Fig. 43.1: Identical number of neighbors yields higher neighbor density in low dimensions than in high dimensions.

[604] https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence
[605] https://en.wikipedia.org/wiki/Student%27s_t-distribution

43.3 t-SNE with Scikit-Learn

Scikit-Learn has the `TSNE`[606] class in the `manifold` module. The `perplexity` parameter controls the desired entropy. Entropy is the base 2 logarithm of perplexity.

```python
import numpy as np
import sklearn.manifold as manifold
import plotly.graph_objects as go
from plotly.subplots import make_subplots
```

```python
data_files = ['omega.npz', 'sphere.npz', 'cube.npz', 'clouds.npz']

for file in data_files:

    loaded = np.load(file)
    x = loaded['x']
    y = loaded['y']
    z = loaded['z']
    red = loaded['red']
    green = loaded['green']
    blue = loaded['blue']

    sne = manifold.TSNE(n_components=2, perplexity=30)
    U = sne.fit_transform(np.stack((x, y, z), axis=1))

    fig = make_subplots(rows=1, cols=2, specs=[[{'type': 'scatter3d'}, {'type':
↪'xy'}]])
    fig.update_layout(width=1000, height=600, scene_aspectmode='cube')
    fig.add_trace(go.Scatter3d(
        x=x, y=y, z=z,
        mode='markers',
        marker={'size': 1.5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,
↪ green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=1)
    fig.add_trace(go.Scatter(
        x=U[:, 0], y=U[:, 1],
        mode='markers',
        marker={'size': 5, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
        hoverinfo = 'none',
        showlegend=False
    ), row=1, col=2)
    fig.show()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

[606] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

CHAPTER

FORTYFOUR

SELF-ORGANIZING MAPS

Self-organizing maps (SOMs), also known as Kohonen networks, appeared in the 1980s. In contrast to all other dimensionality reduction techniques discussed up to now, SOMs drop the requirement that the low dimensional version of a data set has to have as many points as the original data set. Instead, SOMs allow to choose the number of points in low dimensions and then try to find a topology preserving mapping between low and high dimensional data. Here, preservation of topology means preservation of neighborhood relations. Neighboring points in low dimensions are mapped to neighboring points in high dimensions.

A motivation for SOMs can be found in biology. Like artificial neural networks they are inspired by insights into the human brain, see lateral inhibition[607] for details. For this reason some people classify SOMs as a kind of ANNs, but structure and training differ a lot from usual ANNs.

44.1 SOM Structure

Let p be the dimension of the low dimensional space (almost always $p = 2$, sometimes $p = 1$ or $p = 3$) and let q be the desired number of data points $u_1, \ldots, u_q \in \mathbb{R}^p$ in low dimensions. Assign fixed locations to all low dimensional points in $\mathbb{R}^p$. Usually, the $u_1, \ldots, u_q$ are arranged in a regular rectangular or hexagonal grid. These positions will never change.

A SOM maps each u_i into the high dimensional data space by assigning a corresponding weight $w_i \in \mathbb{R}^m$. Although the images of the u_i are called weight, the w_i simply are points in the high dimensional data space. The aim of SOM training is to choose weights scattered over the whole high dimensional data set, but preserving the low dimensional neighborhood relations.

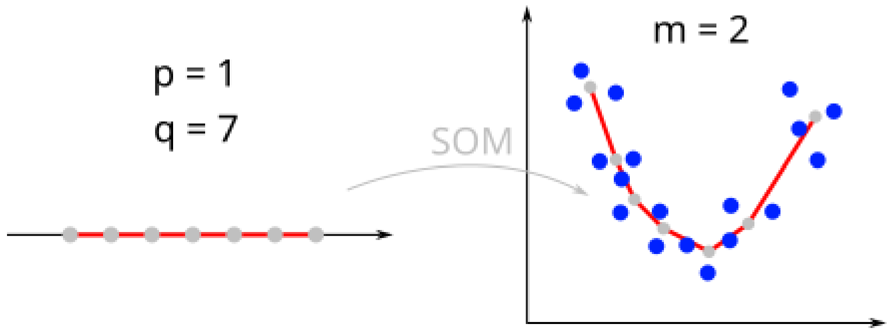


Fig. 44.1: SOMS approxiate high dimensional data by a low dimensional grid of points.

The u_i do not contain any information about the data set. To visualize a SOM we have to visualize properties of the weights w_i in the regular grid defined by the u_i (see below).

[607] https://en.wikipedia.org/wiki/Lateral_inhibition

44.2 Training

Fitting weights $w_1, \ldots, w_q$ to the data set $x_1, \ldots, x_n$ is done iteratively:

- Choose random initial weights (or create a grid in the p dimensional PCA manifold).
- Iterate over all samples:
  - Find the w_i closest to the current sample.
  - Move the best matching w_i and its neighbors closer to the current sample.
- Stop iteration if change of weights is small.

Moving weights towards the current sample is done via a neighborhood function $h : \mathbb{R}^p \to \mathbb{R}$, which typically takes values in $[0, 1]$ but in some cases may attain small negative values, too. Examples are Gaussian bells or the Mexican hat[608]. If w_i is the best matching weight for the current sample x_l, the update rule for all weights is

$$w_j^{\text{new}} := w_j^{\text{old}} + \alpha\, h\left(\frac{1}{r}\,(u_j - u_i)\right)\,(x_l - w_j^{\text{old}}).$$

The parameter $r > 0$ controls the size of the neighborhood. The parameter $\alpha \in (0, 1]$ controls the attracting force of x_l.

Training usually starts with high values for r and α to capture the data set's rough structure in few iterations. Then both values are decreased to allow for accurate fitting of the SOM to the data.

Batch training is possible, too. Here several samples are processed at once. For each sample the closest weight is determined. Then all weights are updated.

44.3 Prediction

With a trained SOM we may assign high dimensional data points to low dimensional grid points. This is also true for data not available during training. The chosen grid point is the one with weight closest to the sample under consideration.

Each low dimensional grid point may be regarded as a cluster of high dimensional points. The described mapping from high to low dimensions simply is 1-NN with the weights as training data.

44.4 Visualization

A straight forward SOM visualization is to color code a selected feature (component of the weights) in the low dimensional grid. This yields as many visualizations as there are dimensions in the data space.

To get some distance information from a SOM one may calculate Euclidean distances between weights of neighboring grid points and visualize those distances on the grid. The result is known as *unified distance matrix* (U-matrix). Dark (low values) areas indicate samples belonging to one cluster and light (high values) areas indicate boundaries between clusters. From the U-matrix one may extract clusters by applying standard image processing algorithms.

Many other kinds of visualizations may be useful, but details depend on the data set under consideration.

[608] https://en.wikipedia.org/wiki/Ricker_wavelet

44.5 SOMs with Python

Scikit-Learn does not support SOMs. But there are lots of Python modules for SOMs, for instance

- `sklearn-som`[609],
- `MiniSom`[610],
- `SOMPY`[611],
- `SimpSOM`[612],
- `somoclu`[613].

The first one focuses on clustering and lacks some more general features like direct access to the weights. `MiniSam` and `SOMPY` come without documentation. Usage has to be deduced from provided code examples. `SimpSOM` has at least some basic documention, but implementation is incomplete. The last one, `somoclu`, is a Python wrapper for an advanced stand-alone SOM software. It's well documented, so we use `somoclu` here. Implementation details may be found in somoclu: An Efficient Parallel Library for Self-Organizing Maps[614].

```
import numpy as np
import matplotlib.pyplot as plt
import plotly.graph_objects as go

import somoclu
```

```
file = 'clouds.npz'

loaded = np.load(file)
x = loaded['x']
y = loaded['y']
z = loaded['z']
red = loaded['red']
green = loaded['green']
blue = loaded['blue']

fig = go.Figure(layout_width=800, layout_height=600, layout_scene_aspectmode='cube
↪')
fig.add_trace(go.Scatter3d(
    x=x, y=y, z=z,
    mode='markers',
    marker={'size': 2, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
    hoverinfo = 'none'
))
fig.show()
```

```
<IPython.core.display.HTML object>
```

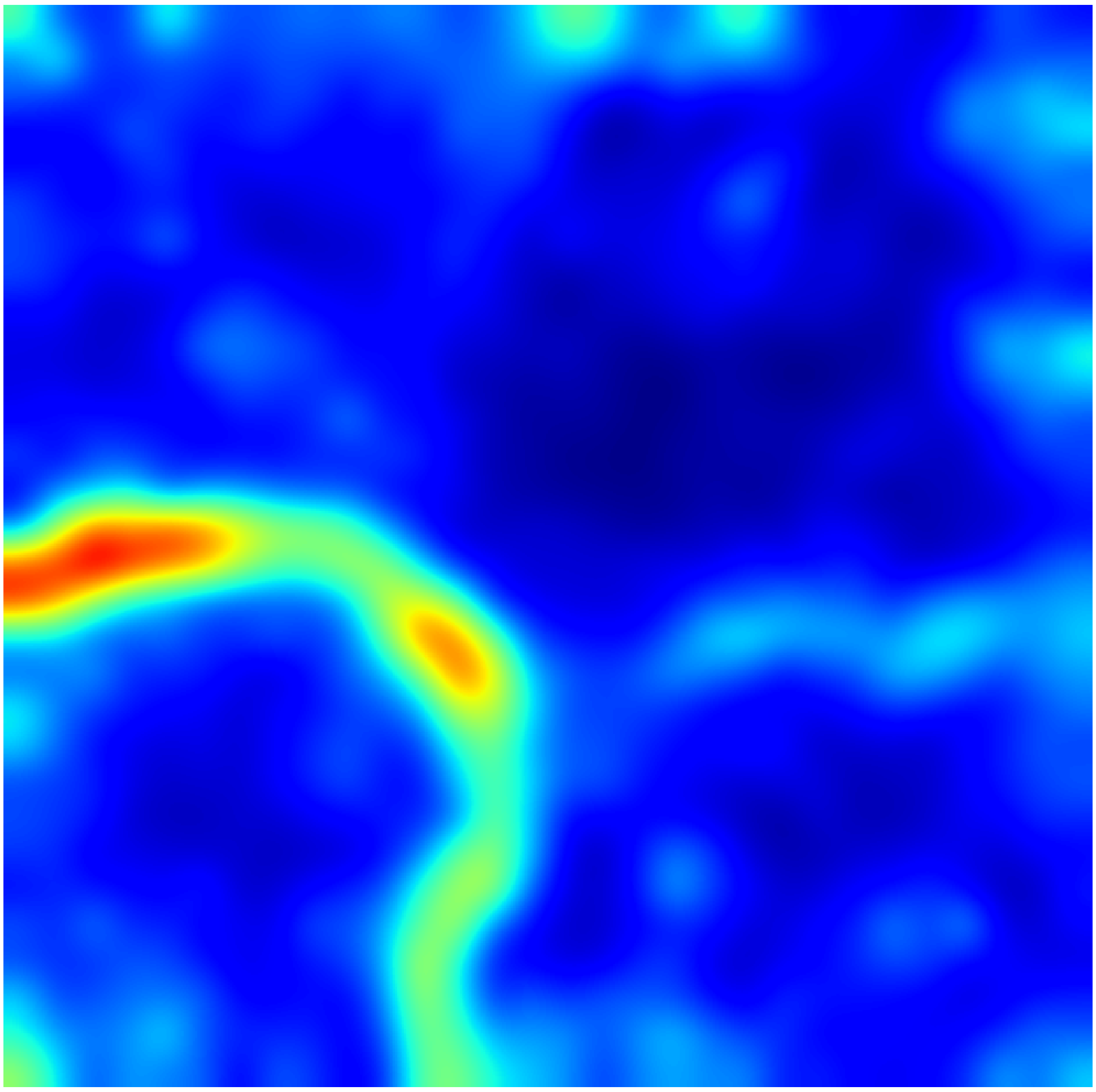
```
cols = 30
rows = 30

som = somoclu.Somoclu(cols, rows, initialization='pca')
som.train(np.stack((x, y, z), axis=1), epochs=1000)
```

[609] https://pypi.org/project/sklearn-som/
[610] https://github.com/JustGlowing/minisom
[611] https://github.com/sevamoo/SOMPY
[612] https://github.com/fcomitani/simpsom
[613] https://somoclu.readthedocs.io/en/stable/index.html
[614] https://www.jstatsoft.org/article/download/v078i09/1125

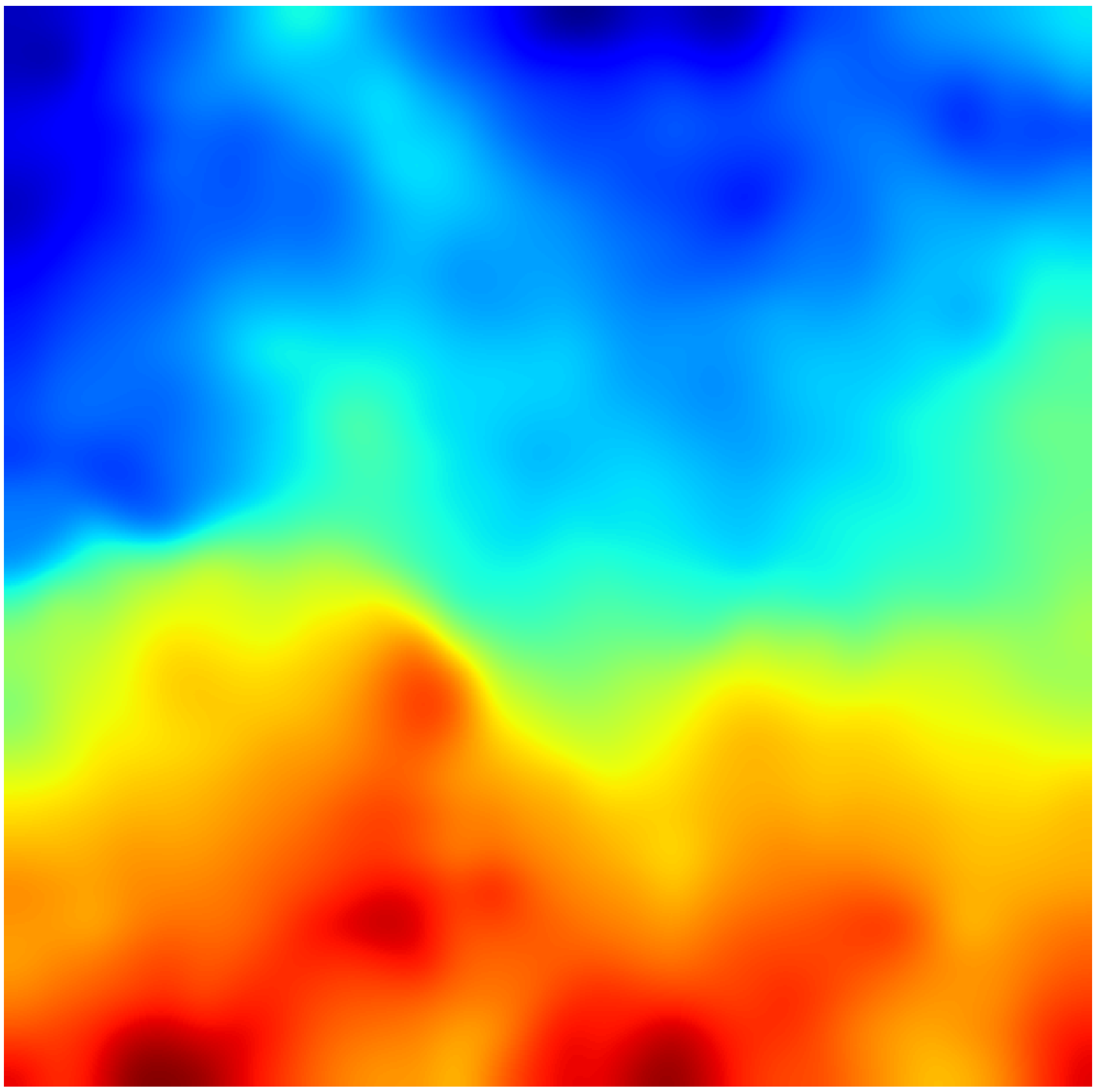
```
Warning: data was not float32. A 32-bit copy was made
```

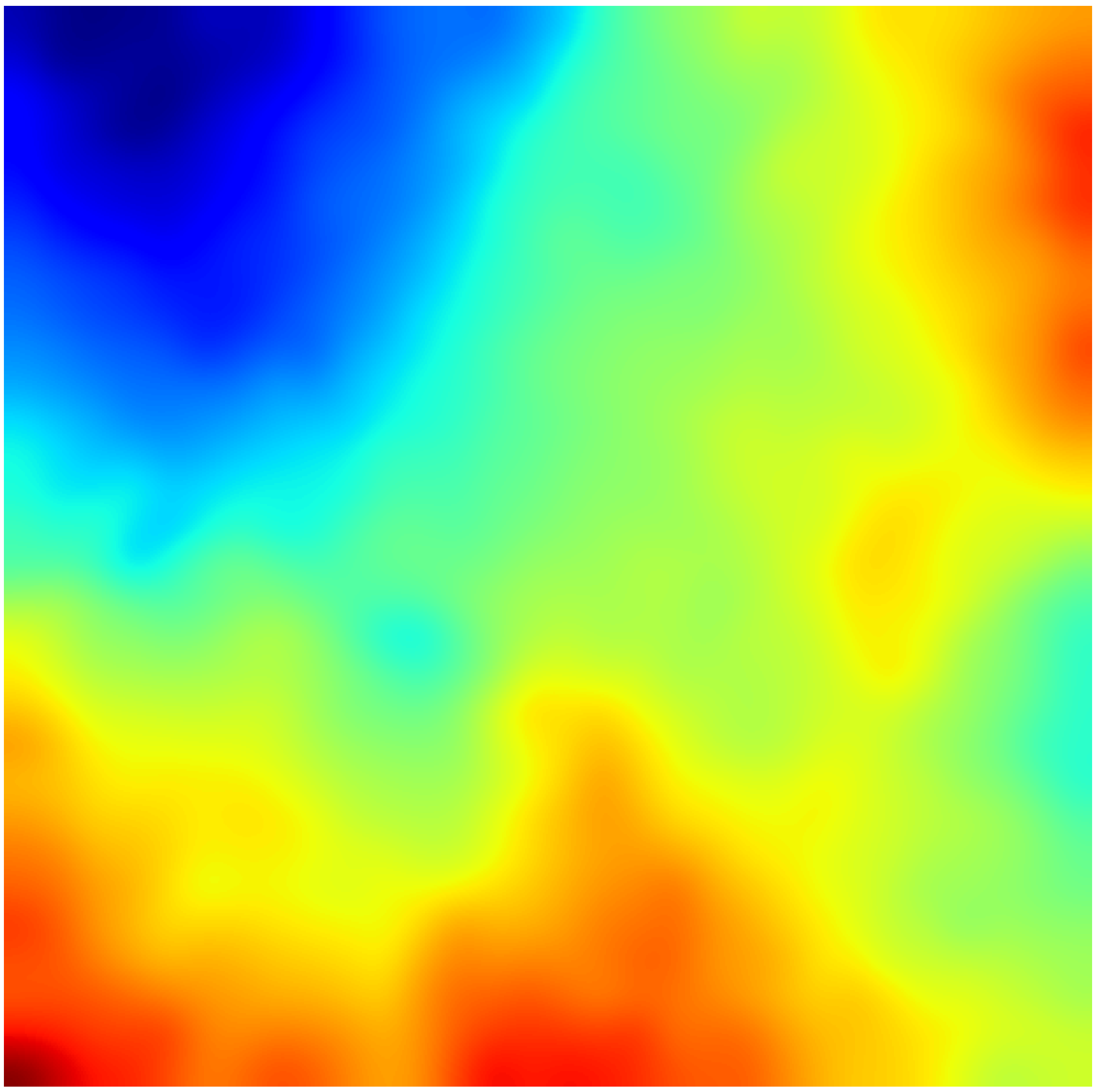
```
print('U-matrix')
som.view_umatrix(colormap='jet')

print('3x components of weight vectors')
som.view_component_planes(colormap='jet')
```

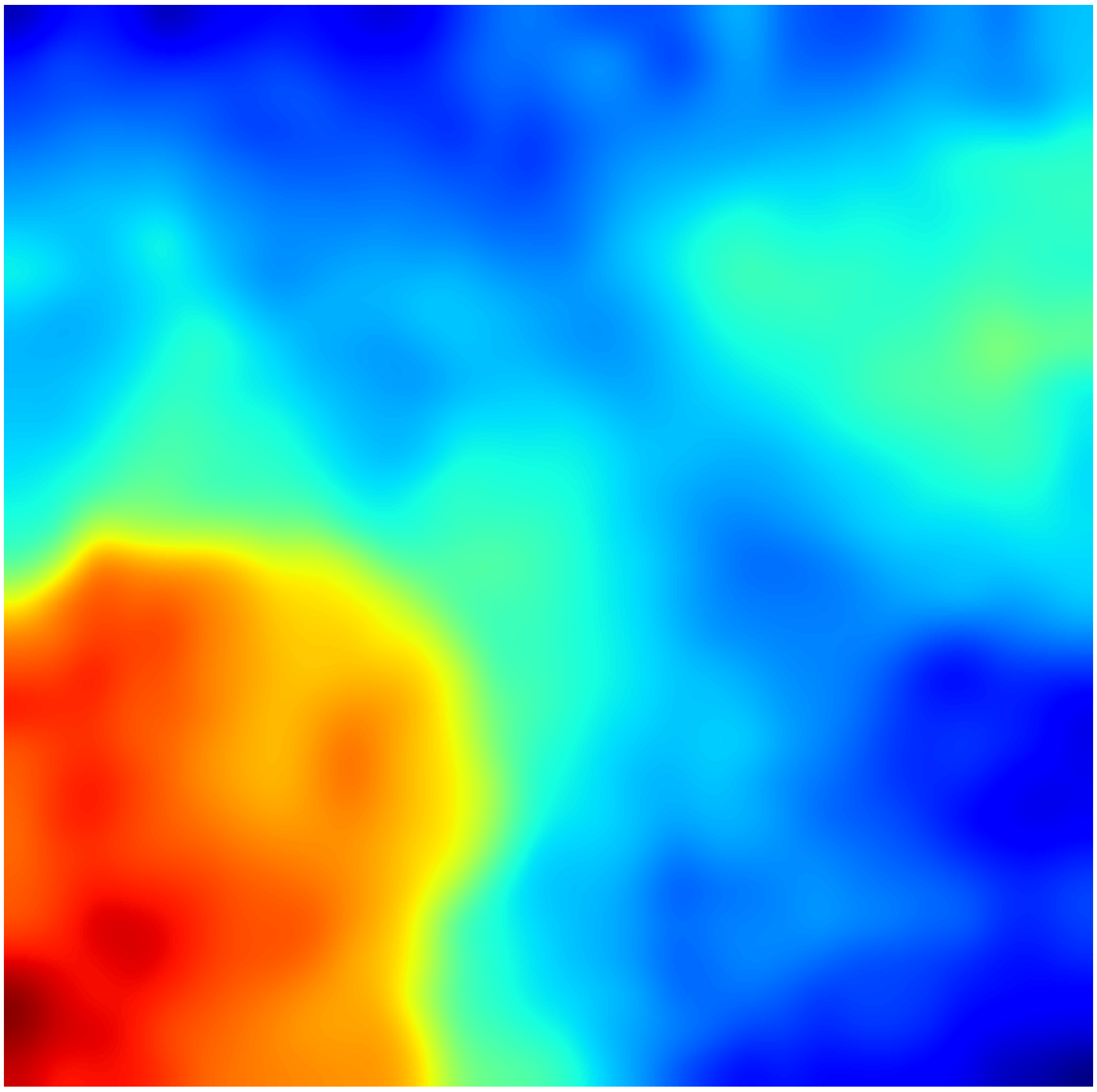
```
U-matrix
```



```
3x components of weight vectors
```





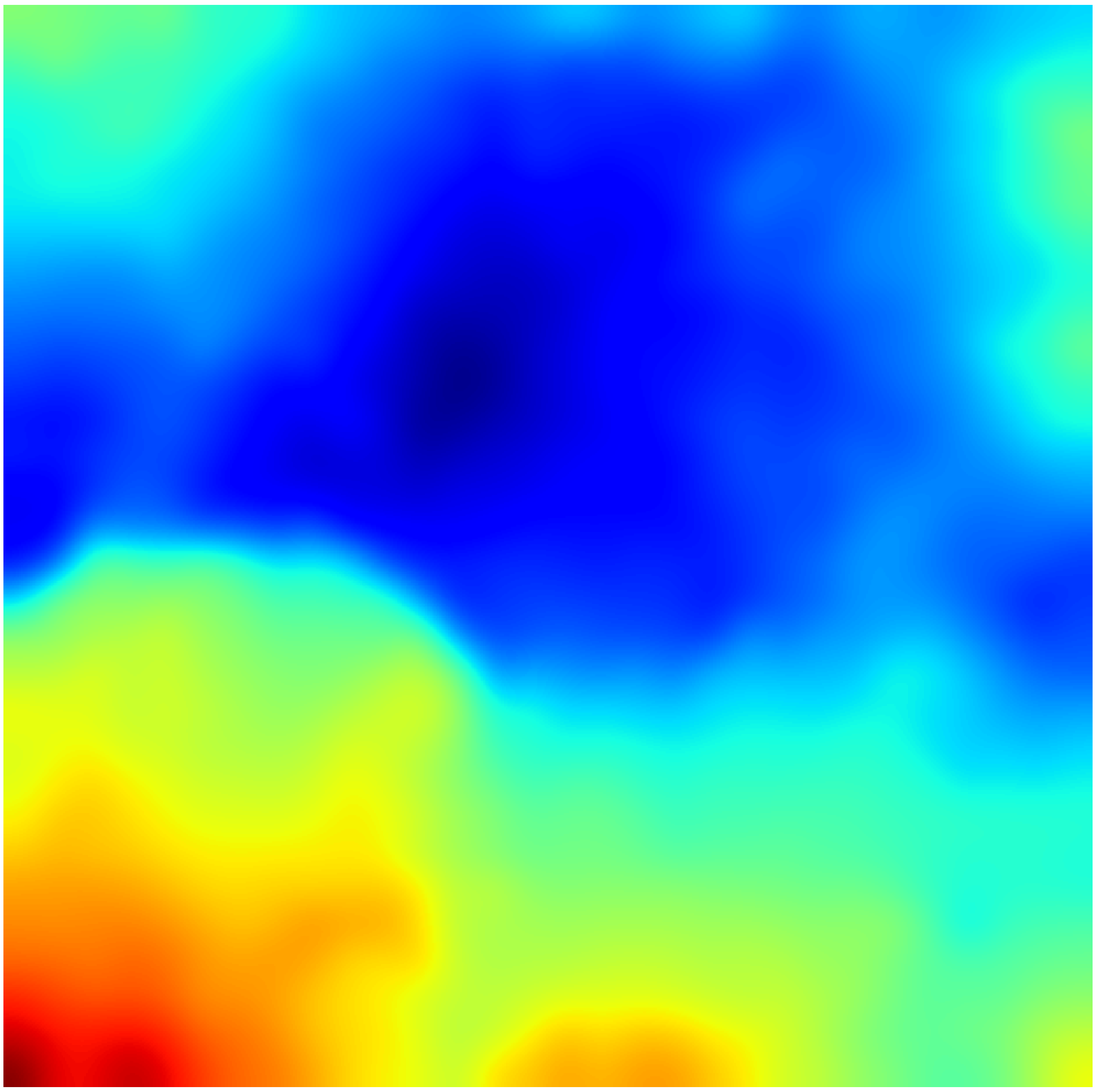


```
<module 'matplotlib.pyplot' from '/home/jef19jdw/anaconda3/envs/ds_book/lib/
↪python3.10/site-packages/matplotlib/pyplot.py'>
```

The `view_activation_map` computes the distance of a sample to all weights and visualizes corresponding matrix.

```
print('activation map')
som.view_activation_map(np.array([[x[700], y[700], z[700]]]), colormap='jet')
```

```
activation map
```



```
<module 'matplotlib.pyplot' from '/home/jef19jdw/anaconda3/envs/ds_book/lib/
↪python3.10/site-packages/matplotlib/pyplot.py'>
```

The `codebook` member variable contains the 3d NumPy array of the SOM's weights (rows x columns x features).

```
fig = go.Figure(layout_width=800, layout_height=600, layout_scene_aspectmode='cube
↪')

fig.add_trace(go.Scatter3d(
    x=x, y=y, z=z,
    mode='markers',
    marker={'size': 2, 'color': [f'rgb({r},{g},{b})' for r, g, b in zip(red,␣
↪green, blue)]},
    hoverinfo = 'none'
))


for r in range(0, rows):
    fig.add_trace(go.Scatter3d(
        x=som.codebook[r, :, 0], y=som.codebook[r, :, 1], z=som.codebook[r, :, 2],
        mode='lines',
        line={'color': 'rgb(0,0,0)'},
```

(continues on next page)

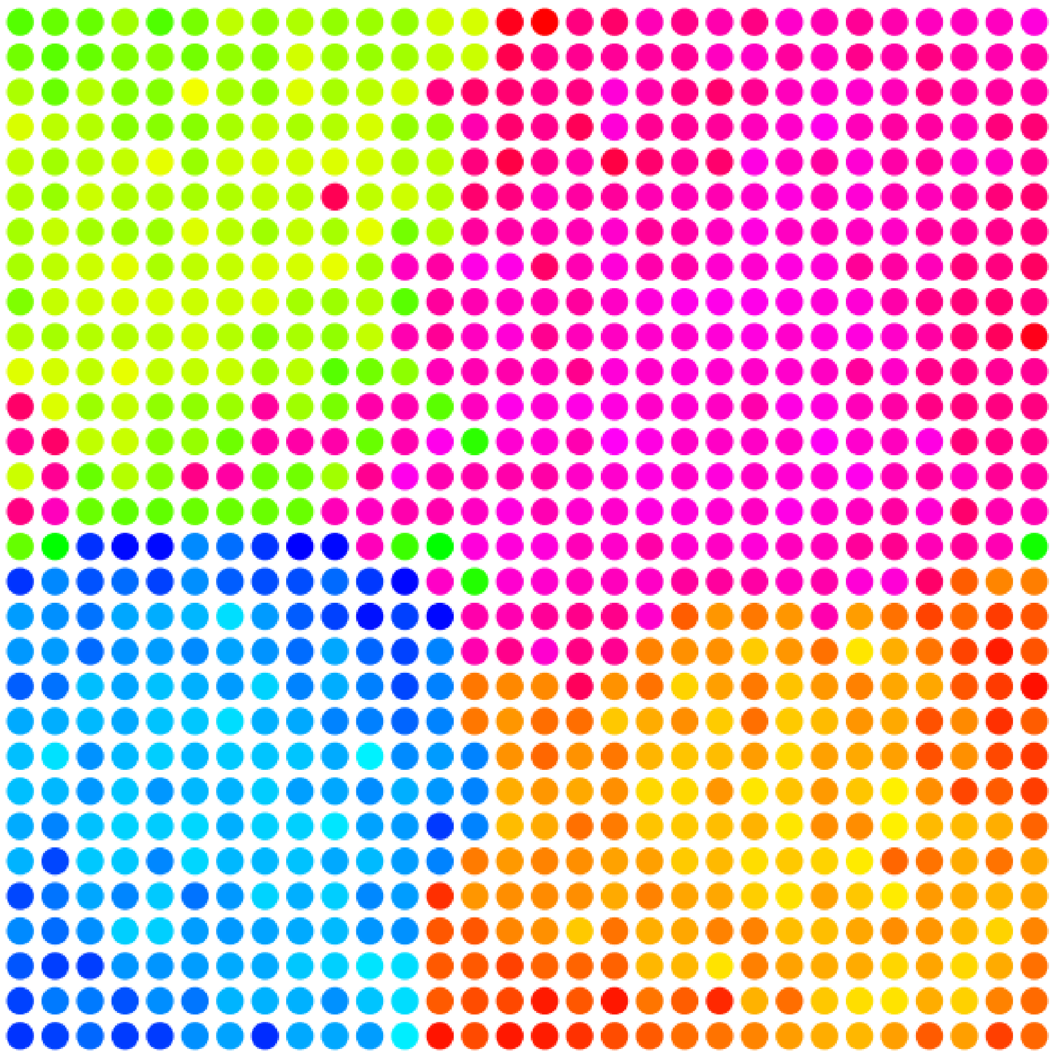
(continued from previous page)

```
        hoverinfo = 'none'
    ))
for c in range(0, cols):
    fig.add_trace(go.Scatter3d(
        x=som.codebook[:, c, 0], y=som.codebook[:, c, 1], z=som.codebook[:, c, 2],
        mode='lines',
        line={'color': 'rgb(0,0,0)'},
        hoverinfo = 'none'
    ))

fig.show()
```

```
<IPython.core.display.HTML object>
```

The `get_surface_state` methods yields the distances between each pair of weight vector and training sample. Return values is a 2d NumPy array (samples x weights). We may use it to visualize weights in 2d by assigning to each weight the closest training sample.

```
closest_sample = np.argmin(som.get_surface_state(), axis=0).reshape(rows, cols)

u1, u2 = np.meshgrid(np.linspace(0, 1, cols), np.linspace(1, 0, rows))
u1 = u1.reshape(-1)
u2 = u2.reshape(-1)

fig, ax = plt.subplots(figsize=(8, 8))
colors = np.stack((red[closest_sample].reshape(-1), green[closest_sample].
↪reshape(-1), blue[closest_sample].reshape(-1)), axis=1) / 255
ax.scatter(u1, u2, c=colors, s=100)
ax.axis('equal')
ax.axis('off')
plt.show()
```



Part VIII

Reinforcement Learning

CHAPTER

FORTYFIVE

OVERVIEW AND EXAMPLES

Reinforcement learning is the third fundamental concept in machine learning next to supervised and unsupervised learning. Although the basic idea (learning by trail and error) is simple, most methods are quite involved and require a sound understanding of underlying theoretical concepts. In this chapter we provide a first overview of relevant notions and ideas. Details will be discussed in in subsequent chapters.

Most of the material in this part of the book is loosely based on or inspired by the book Reinforcement Learning: An Introduction[615] by Richard S. Sutton and Andrew G. Barto (CC BY-NC-ND 2.0[616]).

Related exercises:

- *Modeling* (page 895)

45.1 What is Reinforcement Learning?

By reinforcement learning we denote methods to create computer programs which learn to do the desired tasks by trial and error. To fully understand the difference between reinforcement learning and other paradigms for modeling computational processes we should recap the different approaches for creating models of processes and data.

45.1.1 Approaches to Modeling

Classical Modeling

Especially in (classical) physics one looks at the outcome of a hand full of experiments and tries to create a model of underlying processes from the collected data. Here modeling is a combination of very few observations (data) and lots of thinking. Derived models usually cover a wide range of applications, much wider than covered by the available data.

Newton's law $F=m\cdot a$ is a typical example. We expect that it holds throughout the universe (at least up to some precision) although we only have observations of forces and accelerations from a tiny fraction of the universe.

Another examples is CT imaging[617]. Taking X-ray images of an object from different directions we may compute a 3d representation of the object's insides. The model describing the transformation from a series of 2d X-ray images to 3d is known as Radon transform[618]. This model originated in 1917 from human thinking without having any measurements at hand and proved to be correct later on with the advent of computers able to perform the transformation on real X-ray data.

[615] https://webspace.fh-zwickau.de/jef19jdw/teaching/pti01840/sutton_barto.pdf
[616] https://creativecommons.org/licenses/by-nc-nd/2.0/
[617] https://en.wikipedia.org/wiki/CT_scan
[618] https://en.wikipedia.org/wiki/Radon_transform

Models obtained via classical modeling may be used for understanding real-world processes, for reconstructing situations in the past, and for predicting future developments.

Statistical Modeling

Supervised and unsupervised machine learning use statistics to create computational models from large data sets. Models are fully data-based, (almost) no human thinking is involved. Corresponding models only cover situations represented directly or indirectly (via interpolation) by the underlying data.

A typical example for supervised learning is image classification. The transform from a collection of colored pixels to a label is too complex to derive it from few observations by simply thinking about the problem. But having lots of images and corresponding labels at hand we may train a machine learning model on the classification task. Of course, this model will only work on images similar to the images in the training data set. A model trained on cats and dogs won't recognize cars.

Models obtained from statistical modeling do not describe real-world phenomena but the training data set. We may use statistical modeling to find structures in data sets (unsupervised learning) or to describe mappings between data sets (supervised learning).

Reinforcement Learning

There are tasks that cannot be modeled by classical or statistical methods. They are too complex for classical modeling and require too much training data for successful statistical modelling. An example is autonomous driving, that is mapping lots of sensor data to car control commands. A car's environment is too complex to describe it accurately by a finite fixed data set.

For tasks like autonomous driving we need models that on the one hand solve the task without relying on human understanding of all relevant processes. On the other hand the model has to incorporate data collection mechanisms, because we cannot provide sufficiently rich training data in advance. This is what reinforcement learning is about: create models that collect (relevant) data and solve the desired task based on that data.

If we do not provide training data or preimplement models of relevant processes, how to tell the reinforcement learning system which task to solve? Detailed answers to this question will be given later on, but the principal idea is to provide the model with a fixed set of actions it may chose from and to answer each action with a numerical feedback. If the sequence of actions chosen by the model tends to solve the task feedback values will be high. If the model chooses actions resulting in useless behavior, feedback values will be smaller.

Reinforcement learning models describe how to collect relevant training data and how to map data to useful actions. Reinforcement learning is used to make computers solve complex control tasks.

45.1.2 The Model

What kind of task can solve with reinforcement learning? The answer is simple: all tasks fitting into the standard reinforcement learning model known as *Markov decision process (MDP)*. We will give a precise definition later on. Here we only provide basic notions and ideas.

The final computer program solving the desired task is denoted or understood as an *agent*. The agent performs *actions* in an *environment*. The environment provides information about its *state* to the agent and the environment provides numerical feedback (*reward*) to the agent's actions. Which action the agent chooses next is based on the environment's current state and on the feedbacks received by the agent in the past. The mapping from states to actions is called *policy*.

The aim of reinforcement learning is to find a good (maybe in some sense optimal) policy. A good policy should, in the long-run, yield high accumulated reward (denoted as *return*). Training starts with a random or default policy. Action by action (and, thus, reward by reward) the policy gets modified to yield better actions maximizing the return.

The basic algorithm a typical reinforcement learning model performs is:

1. Choose an initial policy.
2. Take some actions and get corresponding rewards/return.

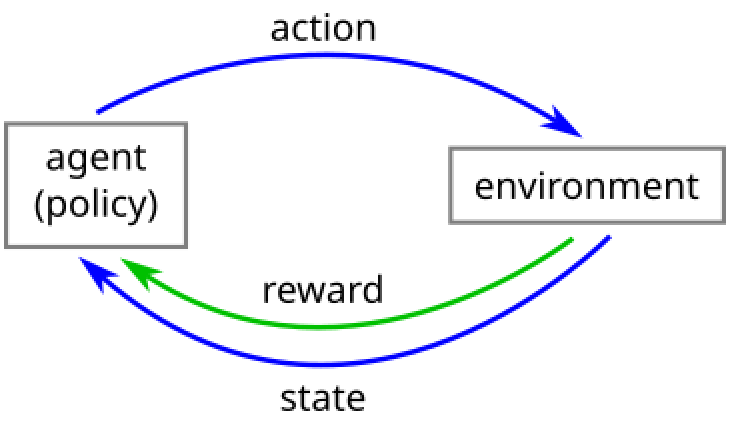


Fig. 45.1: The agent choses an action according to its policy. The environment changes its state and sends a reward signal to the agent.

3. Update/improve the policy.
4. Go to 2.

There's no dedicated training phase. Training and application of the model are one and the same. Of course, one may stop updating the policy if the policy is optimal or at least good enough for the task to be solved (convergence). But the more common situation is that the model improves its policy as long as it is used. This way the model may adapt to changing behavior of the environment.

Every task that can be formulated in such a framework based on an agent, actions, an evironment, and rewards can be tackled by reinforcement learning techniques.

45.2 Examples

Reinforcement learning has a wide range of applications from simple board games to autonomous robots. The simpler ones, especially board games, are good toy examples for testing and understanding important concepts.

To describe how some task can be solved by reinforcement learning we have to specify

- the environment,
- the agent,
- the set of states the environment can attain (or the agent's sensors can record),
- the set of actions available to the agent,
- calculation of rewards.

Further we may specify whether

- there is an end state with no more valid actions (*episodic task*) or
- there is no end state (*continuing task*).

Information about the environment (state) may be

- *complete* (observed state contains all relevant information) or
- *incomplete* (observed state does not contain all relevant information).

45.2.1 Board and Card Games in General

Environment: The board and all other material of the game. In some other players (humans or AI) may be relevant for making decisions. Then they belong to the environment, too. That's the especially the case for games where mimics of other players may reveal secret information.

Agent: The computer player.

States: Current board situation and all other relevant information provided by the environment.

Actions: Every allowed move. If the the agent does not know the games rules completely, but shall learn the rules, then moves not allowed by the rules may belong to the set of actions the agent may take.

Rewards: Depends on the game. In the simplest case the reward is 1 if the action yields immediate victory and 0 else. Other reward functions may also honour moves yielding an in some sense advantageous situation. For some games the aim is to collect as many rewards as possible, thus, there's a canonical reward function for reinforcement learning (Carcassonne[619],…).

Board and card games are episodic (there is an end state).

For some games information about the environment is complete (chess[620], connect four[621], Mensch ärgere dich nicht[622],…). For others information is incomplete (most card games, Scotland Yard[623],…).

In board and card games almost always there are only finitely many actions and states.

45.2.2 Autonomous Driving

Environment: Real world including pedestrians, other cars, butterflies,…

Agent: Computer/controler driving the car.

States: Everything the agent can observer (sensor data).

Actions: Signals to actors, like braking, steering commands,…

Reward: E.g. -1 if crash, 1 if destination reached,…

Autonomous driving often is a continuing task. There is no end state. The agent shall work forever.

Information about the environment is incomplete because the environment is too complex for storing its state in a digital computer (resolution of camera images,…).

The set of state set is infinite. The set of actions almost always is infinite (some freedom of modelling here, discretization).

45.2.3 Grid World

Grid worlds are discrete and heavily simplified variants of autonomous driving settings. A rectangular grid of cells defines possible locations for the agent or objects. Cells may be of different types (empty, wall,…). The aim of reinforment learning here is to train an agent that starting at an arbitrary location finds a (short) path to some destination cell.

The agent has to solve two typical problems in reinforcement learning:

- discover the environment,
- find a short path without hitting a wall or other restrictions.

[619] https://en.wikipedia.org/wiki/Carcassonne_(board_game)
[620] https://en.wikipedia.org/wiki/Chess
[621] https://en.wikipedia.org/wiki/Connect_Four
[622] https://en.wikipedia.org/wiki/Mensch_%C3%A4rgere_Dich_nicht
[623] https://en.wikipedia.org/wiki/Scotland_Yard_(board_game)

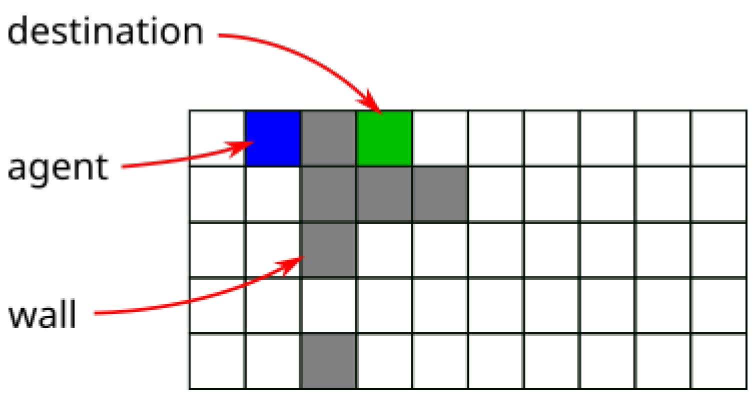


Fig. 45.2: The agent has to discover the grid world and find a short path to the destination cell.

Grid worlds may be dynamic (e.g., moving walls).

Environment: Finite grid of cells, maybe of different type.

Agent: A robot, for instance.

States: Position of agend in grid and type of surrounding cells.

Actions: Step left, right, up or down.

Reward: E.g., -1 for each move, 1 for reachng the destination.

Grid worlds are often used in combination with episodic tasks.

Information about the environment typically is complete.

Action and state sets usually are finite (at least if there are only finitely many cell types).

Automated ware houses sometimes are organized as grid worlds, allowing for arbitrary placement of goods or shelves in a grid.

45.2.4 Online Advertising

Targeted online advertising can be modeled as reinforcement learning problem. Everytime a user visits a website an algorithm decides what ad to show to the user. Ad selection may depend on individual users' behavior or on group behavior.

Environment: Customer websites and behavior of visting users.

Agent: Ad selection algorithm.

States: Ads shown on websites, visiting users, users' click behavior.

Actions: Show ad X to user on website Y.

Reward: E.g., 1, if user clicks add, 10 if user buys an advertised product.

Here we have a continuing task.

Whether information available to the agent is complete or incomplete depends onthe concrete setting.

Action and states sets are finite, but large.

45.3 Maximization of Return

After modeling a reinforcement learning problem including proper definition of the agent, the environment and corresponding actions, states and rewards, our aim is to find a policy which maximizes return, that is, accumulated rewards. To find a good policy the agent has to follow two objectives:

- **exploration** (collect information about the environment),
- **exploitation** (use collected information to get high rewards and to maximize return).

There exist several principal approaches for finding good policies:

- value function methods,
- genetic algorithms[624] and other evolutionary methods[625],
- simulated annealing[626] and other global optimization[627] methods.

In this book we only consider value function methods.

45.3.1 Basic Idea of Value Function Methods

Value function methods are based on scoring schemes (value functions) for states or state-action pairs. Rewards collected by the agent during exploration are used to compute or update scores for all states and actions seen and taken so far. Based on the scores furture actions are chosen. Typically, scores are estimates of expected return if corresponding action is chosen in corresponding state. The more data about the environment and its reward function is available (exploration!) the more accurate the estimates.

Depending on the size of state and action sets we distinguish between

- **tabular methods** (small state and action sets, fitting into memory),
- **approximate methods** (large state and/or action sets, scores cannot be kept in memory for all states/actions).

45.3.2 On-Policy and Off-Policy Methods

The agent may know only one policy, which is used for exploration (what to do next to get more information about the environment?) and gets optimized to solve the desired task (exploitation) at the same time. Such methods are said to be *on-policy*.

Alternatively, the agent may have two policies. One policy controls the agent's behavior during exploration. The other policy gets optimized and won't be used before its good enough.

45.3.3 Action-Values vs. State-Values

Scoring states and/actions in value function methods may follow one of two principal schemes:

- **Action-value methods:** Score each state-action pair. Given a state choose the action with highest score.
- **State-value methods:** Score each state. Given a state choose the action resulting in the state with highest score in all reachable states.

In principle, action-value methods canbe transformed into state-value methods and vice versa. But results obtained from both variants may be slightly different, because different actions (with different scores) may result in the same state (which has only one score) and, on the other hand, one action (with one score) may result in different states (with different scores) at different times (if environment has random features).

[624] https://en.wikipedia.org/wiki/Genetic_algorithm
[625] https://en.wikipedia.org/wiki/Evolutionary_algorithm
[626] https://en.wikipedia.org/wiki/Simulated_annealing
[627] https://en.wikipedia.org/wiki/Global_optimization

45.3.4 Standard Policies

There exist two frequently used standard policies for value function methods: the greedy policy and the ε-greedy policy.

The **greedy policy** always chooses the action with highest score (for action value methods) or the action leading to the state with highest score (for state-value methods). There is no dedicated exploration behavior. Decisions are based on up to now collected information. Actions with high return in the short-run but low return in the long-run may be preferred to actions with low short-run but high long-run return, because actions looking good at early stage will be chosen again and again while missing alternatives with better long-run return.

The **ε-greedy policy** behaves like the greedy policy only with probability $1-\varepsilon$ for some small fixed $\varepsilon > 0$. From time to time (with probability ε) it will choose a random action. Thus, the ε-greedy policy is able to find actions looking unfavorable in the short-run but may turn out to yield higher return than (at the moment) higher scored actions in the long-run. In this sense, there's explicit exploration.

In the short-run the ε-greedy policy yields lower return than the greedy policy because some actions chosen by the ε-greedy policy are simply wrong (do not help in solving the task). But in the long-run return will be better than for the greedy policy because the environment gets explored more extensively yielding better options for solving the desired task.

Consider moving to a new city a looking for a route to walk to the university. If you follow a greedy policy you decide for the way which looks shortest at first glance (possibly found by a routing algorithm). You'll never try a different way. If you instead follow an ε-greedy policy from time to time you try new routes. Most of them will be longer, but you may even find routes that are faster than the initial route (for instance, due to fewer traffic lights or otherwise simple/faster/fewer road crossings). In the long-run the ε-greedy approach will save you time.

The problem whether it's better to spend more time for exploration or to fully exploit current knowledge without further exploitation is known as the **exploration-exploitation dilemma**.

CHAPTER

FORTYSIX

STATELESS LEARNING TASKS (MULTI-ARMED BANDITS)

For introducing and demonstrating important concepts and ideas of reinforcement learning we start with a simplified setting here. We assume that the environment always has the same state from the agent's point of view. Of course, this one state does not matter for action selection of the agent. Thus, from the agent's point of view the task is stateless.

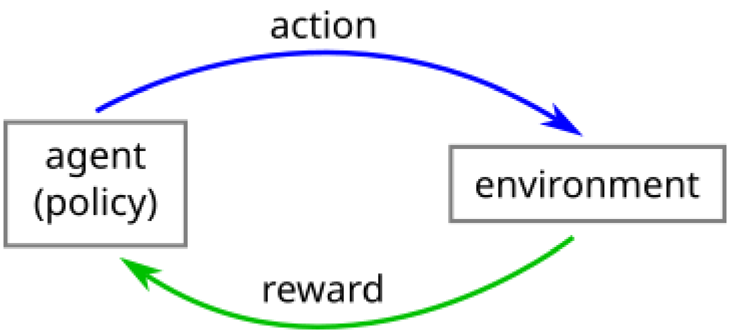


Fig. 46.1: The agent choses an action according to its policy. Then the environment sends a reward signal to the agent.

Related projects:

- *Online Advertising* (page 997)

46.1 The Detailed View

Although the definition of *stateless tasks* has already been given in the introductory paragraph, we should be more clear about the consequences. Each action of the agent yields a reward (but no new state). If the agent knows k actions, then there will be at most k different rewards.

But take care, reward may be different each time even if the agent always chooses one and the same action. An agent seeing always the same state does NOT imply that the environment does not change. Maybe the agent lacks relevant sensors. Thus, rewards have to be regarded as random numbers with distributions depending on the action chosen. In other words, the sequence of rewards for stateless tasks behaves like a sequence of random numbers drawn from k distributions and distributions are chosen by the agent's action.

Such settings are known as multi-armed bandits. The k actions correspond to the k levers of k one-armed bandits. Rewards correspond to the total coin output of the bandit machines after pulling one of the levers. Each one-armed bandit may have a different probability distribution for yielding coins. Some time instead of *multi-armed bandit* the term *k-armed* bandit is used.

[628] https://commons.wikimedia.org/wiki/File:Slot_machines_at_Wookey_Hole_Caves.JPG

Fig. 46.2: A real-world four-armed bandit made of four one-armed bandits. Source: Wikipedia (public domain)[628], edited by the author.

46.2 Stationarity

We already mentioned, that although the agent always sees the same state, the environment may change as a result of the agent's actions. But environment may change due to other influences, too, especially over time. If reward distributions are fixed, that is, they do not change over time, then the learning task is *stationary*. If reward distributions may change over time, then the task is *non-stationary*.

Stationary task only need exploration until reward distributions are known in sufficient detail to the agent. Then gathered knowledge can be exploited without ony further exploration. Further exploration would would decrease return (long-term accumulated rewards) without any use.

For non-stationary task exploration should never stop. Else the agent's behavior cannot adapt to changing reward distributions. Without exploration return will become lower and lower over time.

46.3 Example: Online Advertising

Let's consider a simple online advertising model for demonstrating first methods for stateless reinforcement learning tasks. A webpage shows an ad everytime some user loads the page. The ad to show is chosen from a pool of k ads. If the ad gets clicked by the user, the webpage provider earns some money. Of course, the website provider wants to show ads being clicked more often than others. But how to identify successful ads without prior knowledge on user behavior.

The setting is as follows:

- **actions:** show one of k available ads (so we have k actions to choose from),
- **reward:** 1 if ad is clicked, 0 else.

To make things precise (and stateless) we have to impose some restrictions:

- We do not collect user-specific information and we do not care about webpage content. These two assumptions yield a stateless environment.
- There are only few users. Thus, the next visits the website not before the previous user has decided whether to click the ad or not.
- Users have independent click behavior (they do not talk to each other).

Click behavior for each ad may be represented by an unknown probability value whether the ad gets clicked or not. The k values may be constant over time (stationary task) or they may change due to external factors (non-stationary task). Here 'external' means that those factors are not included in the state information provided by the environment (in stateless tasks no state information is provided). Typical external factors may be seasonal effects, for instance.

There exist several other real world examples for stateless reinforcement learning tasks. An important one is routing in computer networks. Data packages try to find the fastest or the most reliable route to the destination machine. But travel times via different routes are unknown in advance and may change over time. Whether transmission will be successful or not is unknown, too. The routing software has to learn properties (average speed, reliability) of available routes over time.

46.4 Solution without Rl (A/B Testing)

A standard approach for solving stateless learning tasks is A/B testing. It's a two phase approach:

- In phase 1 all actions are chosen equally often and average rewards for each action are calculated. In this phase we have 100% exploration, but no exploitation.
- In phase 2 the action with highest reward in phase 1 is chosen all the time. Now we have 100% exploitation, but no further exploration.

This approach has two important drawbacks:

- In phase 1 actions with low average reward are shown too often. That's a problem especially of long exploration phases, because return (accumulated rewards) will be significantly lower than with reinforcement learning methods.
- If average rewards change over time (non-stationary tasks), either the model is not able to adapt to the new sitatuation or another exploration phase is required. Both variants will decrease the return. Without adaption decrease will be smaller in the short-run, but higher in the long-run. With new exploration phase decrease of return will be high in the short-run, but low (maybe no decrease at all) in the long-run.

Let's consider the everyday example of finding a route from your home to some new destination. If you use A/B testing to find the fastest route, you take every available route (including the slow ones!) several times. Then you decide for the fastest route for the rest of your life. Even if there appear new and faster routes or if your chosen route becomes slower (more traffic,...), you won't readjust your decision. That's not what humans really do.

46.5 Solution with Rl (Sample Averaging)

To solve stateless learning tasks with reinforcement learning methods we have to formalize things a bit more. First, we restrict our attention to stationary tasks. Then we adapt the introduced method to non-stationary tasks.

46.5.1 Action Values

As already mentioned in *Maximization of Return* (page 798) we consider value function methods only. Thus, we have to define a state-value function or an action-value function. Of course, state values are useless in a stateless task (actions do not alter that). Thus, we use action values.

Action values should express return expected from choosing corresponding actions. If return is simply the sum of all rewards over time and if we have a reward function taking values 0 and 1 only like in the online advertising example above, then for continuing tasks return will be infinite. For episodic tasks return will be the product of the number of steps in the episode and average return. Thus, in both cases average reward is a sensible action value:

$$q^*: A \to \mathbb{R}, \qquad q^*(a) := \text{expected reward for action } a,$$

where A denotes the set of all available actions. Note that by 'expected reward' we denote the theoretical average reward, that is, the mean of the underlying reward distribution. By 'average reward' we denote the observed average, that is, the empirical mean of the reward distribution.

If we would know q^*, we could simply choose the action with highest $q^*(a)$ (that is, we could follow the greedy policy with respect to q^*). But q^* is unknown and has to be estimated from interacting with the environment.

In A/B testing we would follow the random choice policy in phase 1 (exploration) to get estimates for all $q^*(a)$. Then, in phase 2 (exploitation), we would follow the greedy policy w.r.t. q^*.

In reinforcement learning we use the ε-greedy policy w.r.t. some estimate of q^* and improve the estimate in each step.

46.5.2 Estimating Action Values

Assume that we are at time step $t \in \mathbb{N}$, that is, we've already chosen $t-1$ actions $a_1, \ldots, a_{t-1}$ and we have observed corresponding $t-1$ rewards $r_1, \ldots, r_{t-1}$. Then a straightforward estimate for the inaccessible action-value function q^* at time t is

$$Q_t: A \to \mathbb{R}, \qquad Q_t(a) := \frac{\text{sum of rewards obtained from choosing } a \text{ in first } t-1 \text{ steps}}{\text{number of times } a \text{ has been chosen in first } t-1 \text{ steps}},$$

where we set $\frac{0}{0} := 0$. This is the empirical mean reward for each action.

Note that setting initial estimates $Q_1(a)$ to zero for all actions a is somewhat arbitrary. Below we'll discuss the influence of initial values on the agent's behavior and how to choose appropriate ones.

If we combine this value function estimate with the ε-greedy policy, we obtain the following algorithm for solving stateless learning tasks:

Algorithm: sample averaging

```
1. Choose $\varepsilon \in (0, 1)$.
2. For $t = 1, 2, \ldots$ repeat:
    1. With probability $1 - \varepsilon$ take the action maximizing $Q_t$. With probability $\varepsilon$ take a random action.
    2. Observe reward and calculate $Q_{t+1}$.
```

46.5.3 Efficient Implementation

Calculation of the value function estimate Q_t given by the formula above requires storing all actions taken und rewards observed so far. The sums to calculate become longer and longer over time. But a simple reformulation of the formula allows to compute Q_t from Q_{t-1} without the need for storing all actions and rewards.

Fix an action a and a time step t. For $t = 1$ we set $Q_1(a) := 0$. For $t = 2, 3, \ldots$ the value of $Q_t(a)$ differs from $Q_{t-1}(a)$ only if action a has been chosen in step $t-1$. Now assume that in step $t-1$ action a has been chosen and denote by $t_1, t_2, \ldots, t_n \in \{2, \ldots, t-1\}$ the time steps where we already updated the Q value for a (that is, a has been chosen n times so far, in steps $t_1 - 1, \ldots, t_n - 1$). Then

$$\begin{aligned} Q_t(a) &= \frac{1}{n+1}\left(r_{t-1} + \sum_{i=1}^{n} r_{t_i-1}\right) \\ &= \frac{1}{n+1}\left(r_{t-1} + n\,Q_{t_n}(a)\right) \\ &= Q_{t_n}(a) + \frac{1}{n+1}\left(r_{t-1} - Q_{t_n}(a)\right). \end{aligned}$$

Because $Q_{t_n}(a) = Q_{t_n+1}(a) = \ldots = Q_{t-1}(a)$ we may rewrite this as

$$Q_t(a) = Q_{t-1}(a) + \frac{1}{n+1}\left(r_{t-1} - Q_{t-1}(a)\right).$$

Take care with notation here: n counts how often action a has been chosen before step t and, thus, depends on t and a. Time steps $t_1, \ldots, t_n$ are specific to a, too.

From the structure of the incremental formula for $Q_t(a)$ we see that $Q_t(a)$ adapts to the most recent reward for action a with damping factor $\frac{1}{n+1}$, which decreases over time.

46.5.4 The Non-stationary Case

Sample averaging adapts only very slowly to changing reward probabilities, because the more steps we already did the less influence newly observed rewards will have.

There are two simple methods to make sample averaging more adaptive and, thus, more suited for non-stationary tasks:

- Use moving averages, that is, only consider the last N rewards for estimating the value function q^* with some fixed N.
- Use weighted averages with weights the smaller the older a reward is.

Implementation of the first idea is straightforward. We have to store the fixed number of most recent rewards and have to use the original formula for calculating average reward, not the incremental one.

The second idea turns out to be a simple, but not so obvious modification of the incremental formula for Q_t. We simply have to replace the factor $\frac{1}{n+1}$ by a constant $\alpha \in (0, 1]$:

$$Q_t(a) = Q_{t-1}(a) + \alpha\left(r_{t-1} - Q_{t-1}(a)\right).$$

Writing this in a non-incremental way with the notation from above we see the weights:

$$\begin{aligned}
Q_t(a) &= Q_{t_n}(a) + \alpha\,(r_{t-1} - Q_{t_n}(a))\\
&= \alpha\,r_{t-1} + (1-\alpha)\,Q_{t_n}(a)\\
&= \alpha\,r_{t-1} + (1-\alpha)\,(\alpha\,r_{t_n-1} + (1-\alpha)\,Q_{t_{n-1}}(a))\\
&= \alpha\,r_{t-1} + \alpha\,(1-\alpha)\,r_{t_n-1} + (1-\alpha)^2\,Q_{t_{n-1}}(a)\\
&= \alpha\,r_{t-1} + \alpha\,(1-\alpha)\,r_{t_n-1} + (1-\alpha)^2\,(\alpha\,r_{t_{n-1}-1} + (1-\alpha)\,Q_{t_{n-2}}(a))\\
&= \ldots\\
&= (1-\alpha)^{n+1}\,Q_1(a) + \alpha\,r_{t-1} + \sum_{i=0}^{n-1}\alpha\,(1-\alpha)^{i+1}\,r_{t_{n-i}-1}\\
&= \alpha\,r_{t-1} + \sum_{i=1}^{n}\alpha\,(1-\alpha)^{n+1-i}\,r_{t_i-1}.
\end{aligned}$$

The weights $\alpha\,(1-\alpha)^{n+1-i}$ are the smaller the smaller i is, that is, old rewards have less influence an $Q_t(a)$ than more recent rewards. If $\alpha = 1$, all weight is at the most recent reward. If α is very close to 0, then all weights are almost identical.

46.5.5 Optimistic Initialization

Initialization of Q_1 to zero is somewhat arbitrary. Usually rewards are positive and initialization to zero corresponds to 'no rewards up to now'. But we may use non-zero initial action values to foster exploration in early steps.

If we initialize Q_1 to some fixed positive number greater than reward values for all actions, then even with the greedy policy the agent will choose all actions several times before settling to the action with highest average reward. The first action chosen (by chance, because all Q_1 values are equal) will lower the high initial value for that action. In the second step all other actions then have higher action values than the action chosen first. Thus, another action will be chosen. Regular behavior of the policy will not start until after all actions have been chosen several times and the influence of initial action values vanishes.

The idea to use high initial action values is known as *optimistic initialization*. Of course, it's not suited for non-stationary tasks, because increased exploration tendency is restricted to the first steps. But for stationary tasks optimistic initialization can be an alternative to the ε-greedy policy or it may allow for smaller ε.

CHAPTER

FORTYSEVEN

MARKOV DECISION PROCESSES

Finite Markov decision processes are the standard model for reinforcement learning tasks. Although their simple structure almost all tasks fit into this framework. Having a rigorous mathematical model at hand, we are able to solve all relevant task in a unified manner.

Related exercises:

47.1 Basic Notions

47.1.1 Finite Markov Decision Processes

Decision processes are processes which can be modeled by agent-environment interactions. The agent takes actions according to a policy and the environment provides feedback (state and reward) to the agent. With discrete time steps $t=0,1,\ldots$ this interaction yields a *trajectory*

$$s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, \ldots,$$

where a_t, r_t, s_t denote actions taken, rewards obtained, and states observed, respectively. Here s_0 is the environment's initial state and a_0 is the agent's first action yielding the reward r_1 and transforming the environment into state s_1. Note that there are continuous decision processes, too, but we restrict our attention the discrete setting without mentioning this every time.

Such a decision process is called a Markov decision processes (MDP) if it has the following property, known as **Markov property**: At each time step t obtained reward r_t and current state s_t only depend on the previous state s_{t-1} and on the previous action a_{t-1} (and chance). In other words: if we know s_{t-1} and a_{t-1}, then we also know r_t and s_t (up to random influences).

The Markov property ensures that state information is sufficiently rich. The environment does not have to look at historical developments to find the next state based on the current action. The Markov property can always be satisfied by appropriate modeling of the environment.

A *finite* Markov decision process is a Markov decision process with finite state set, finite action set, and a finite number of different reward values.

Below $\mathcal{S}$ denotes the state set, $\mathcal{A}(s)$ denotes the set of actions feasible in state s, and $\mathcal{R}\subseteq\mathbb{R}$ denotes the set of reward values.

47.1.2 Examples and Counter Examples

Continuously evolving processes like many physical phenomena aren't (discrete) Markov decision processes, because they lack discrete time steps. Processes with continuous time cannot be solved appropriately on a computer. From the analytical point of view continuous time is not a problem. To solve continuous time processes like driving a car with typical reinforcement learning algorithms we have to find ways for discretizing time.

Imagine a grid world in which the agent obtains an extra reward if it reaches the destination cell without touching a wall on its path to the destination cell. This model does not have the Markov property, because the reward cannot be determined from previous state and action. Instead, we would have to look at all previous states to determine whether the agent deserves the extra reward.

Similarly, the environment could react to the action 'go to the right' by moving the agent by one cell if it hit a wall some steps before and by two cells if the agent didn't hit a wall up to now. Then again the Markov property is violated because the next state (position of agent) cannot be determined from previous state and action. Instead, the environment would have to look at all previous states to determine the next state.

The grid world counter example can be turned into a Markov decision process by extending state information. The environment could have a flag in its state telling us whether the agent touched the wall or not. Reaching the destination cell, extra reward is determined by current state information.

An example for non-finite Markov decision processes is autonomous driving. Turning the steering wheel by an angle can be modeled as a set of infinitely many actions (one per angle). Most infinite action sets can be turned into finite sets by discretization. Instead of allowing for arbitrary angles we could restrict the action set to integer angles, for instance.

47.1.3 Environment Dynamics

Almost always we consider stochastic environments, that is, identical behavior of the agent may lead to different feedback (state, reward) from the environment at different times. The environment's behavior ('dynamics') is completely described by values

$$p(s',r,s,a)\in[0,1],$$

which are to be interpreted as probability that state s' and reward r are observed after taking action a in state s.

If we knew p for all combinations of arguments, we could solve corresponding reinforcement learning problem without exploration. But usually we don't know p. An example for complete knowledge of p are simple grid worlds. Here p encodes the map of the grid world.

For finite Markov decision processes p can be represented by a (four dimensional) table with finitely many cells.

47.1.4 Goals and Return

Reinforcement learning algorithms are designed to maximize accumulated rewards in the long-run. Thus, defining sensible rewards is the only way to direct the agent to a learning task's goal! Accumulated rewards are denoted as *return*. Return obtained after time step t will be denoted by g_t.

For episodic tasks with T time steps return is

$$\begin{aligned} g_t &:= r_{t+1}+r_{t+2}+\cdots+r_T\\ &= r_{t+1}+g_{t+1}. \end{aligned}$$

For continuing tasks a simple sum would have infinitely many summands and (the infinitely many) rewards in far future would outshine the (finitely many) near-future rewards. Thus, for continuing tasks we should use discounting:

$$\begin{aligned} g_t &:= r_{t+1}+\gamma\,r_{t+2}+\gamma^2\,r_{t+3}+\gamma^3\,r_{t+4}+\dots\\ &= r_{t+1}+\gamma\,g_{t+1}. \end{aligned}$$

Here $\gamma\in[0,1)$ is the discounting parameter. Small γ puts more weight on near future, whereas γ close to one also includes rewards obtained in far future. An alternative is to use the simple sum of a fixed number of future rewards.

For finding good policies we aren't interested in concrete returns obtained by the agent in past episodes, but in *expected return*. That is, we want to forecast average return over many episodes (episodic tasks) or long time spans (continuing tasks) obtained by following the policy under consideration. This is possible if we have complete knowledge of the environment dynamics. But often we do not have complete knowledge of environment dynamics. Then we have to find good estimates for expected return to properly direct the agent.

47.1.5 Policies and Value Functions

A *policy* describes the agent's behavior by mapping states to actions. Each state-action pair a probability-like score $\pi(a,s)$ is assigned to, which can be interpreted as probability that action a is chosen in state s. The goal of developing reinforcement learning algorithms is to find good policies (in terms of expected return).

Value functions express expected returns for all actions and/or states:

- The *state-value function* v_π with respect to a policy π assigns to each state s the expected return when starting from s and following π.
- The *action-value function* q_π with respect to a policy π assigns to each state-action pair (s,a) the expected return when starting from s with action a, then following π.

Writing down concrete formulas for value functions is quite difficult. We start with two implicit formulas showing the relation between state values and state-action values (p encodes environment dynamics, see above):

- The expected return in state s is the mean (weighted by probability) of expected returns obtained from all possible actions in state s:

$$v_\pi(s)=\sum_{a\in\mathcal{A}(s)}\pi(a,s)\,q_\pi(s,a)\qquad\text{for all } s\in\mathcal{S}.$$

- The expected return for the state-action pair (s,a) is the mean (weighted by probability) of expected returns of all possible follow-up states:

$$q_\pi(s,a)=\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,(r+\gamma\,v_\pi(s'))\qquad\text{for all } s\in\mathcal{S}\text{ and all } a\in\mathcal{A}(s).$$

  This formula is for continuing tasks only. For episodic tasks replace γ by 1.

From these equations we see, that computing state values from state-action values is always possible. But computing state-action values from state values requires knowledge of the environment dynamics.

The explicit formula for $v_\pi(s)$ has the structure

$$\begin{aligned}v_\pi(s)&=\text{expected reward after first action}\\&\quad+\gamma\times\text{expected reward after second action}\\&\quad+\gamma^2\times\text{expected reward after third action}\\&\quad+\ldots.\end{aligned}$$

For continuing tasks this sum has infinitely many summands, but finite value (expected rewards are bounded and $1+\gamma+\gamma^2+\ldots$ is a geometric series). For episodic tasks the sum has finitely many summands (and $\gamma=1$). Making 'expected reward after … action' explicit we obtain

$$\begin{aligned}v_\pi(s)=&\sum_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}\pi(a,s)\,p(s',r,s,a)\,r\\&+\gamma\sum_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}\sum_{a'\in\mathcal{A}(s')}\sum_{s''\in\mathcal{S}}\sum_{r'\in\mathcal{R}}\pi(a,s)\,p(s',r,s,a)\,\pi(a',s')\,p(s'',r',s',a')\,r'\\&+\gamma^2\sum_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}\sum_{a'\in\mathcal{A}(s')}\sum_{s''\in\mathcal{S}}\sum_{r'\in\mathcal{R}}\sum_{a''\in\mathcal{A}(s'')}\sum_{s'''\in\mathcal{S}}\sum_{r''\in\mathcal{R}}\pi(a,s)\,p(s',r,s,a)\,\pi(a',s')\,p(s'',r',s',a')\,\pi(a'',s'')\,p(s''',r\\&+\ldots.\end{aligned}$$

For $q_\pi(s,a)$ the explicit formula looks almost identical, simply remove $\sum_{a\in\mathcal{A}(s)}$ and $\pi(a,s)$ in each line.

On the one hand, value functions provide information about a policy's quality. On the other hand, value functions can be used to define policies:

- A *greedy policy* with respect to a value function only chooses actions with highest value (in case of an action-value function) or actions leading to a state with highest value (in case of a state-value function).
- An ε-*greedy policy* with respect to a value function behaves like a greedy policy with probability $1-\varepsilon$ and chooses a random action with probability ε.

Value function based reinforcement learning methods estimate value functions from information the agent gathers over time (exploration) resulting in a (value function based) policy solving the learning task (exploitation). If during exploration the agent is already controlled by the (iteratively changing) value function based policy to be optimized, the method is said to be *on-policy* (same policy for exploration and exploitation). If during exploration the agent is controlled by another policy (the random policy, for instance), then the method is *off-policy* (different policies for exploration and exploitation).

Whether to use state-values or action-values to define a policy depends on the concrete use case. Action value functions (or estimates thereof) require more memory, because they have to store expected returns for all pairs of states and actions. State value functions require less memory, but to choose an action based on state values we have to predict the next state, which may be impossible without complete knowledge of the environment.

47.2 Bellman Equations and Optimal Policies

Given complete knowledge of the environment dynamics p we may compute value functions for policies without exploration by the agent. Further, complete knowledge allows to find (in some sense) optimal policies.

47.2.1 Computing Value Functions (Bellman Equations)

Given a policy π, from the relations between state values and state-action values we immediately obtain equations

$$v_\pi(s)=\sum_{a\in\mathcal{A}(s)}\pi(a,s)\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,(r+\gamma\,v_\pi(s'))$$

for all $s\in\mathcal{S}$ and

$$q_\pi(s,a)=\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,\left(r+\gamma\sum_{a'\in\mathcal{A}(s')}\pi(a',s')\,q_\pi(s',a')\right)$$

for all $s\in\mathcal{S}$ and all $a\in\mathcal{A}(s)$. These are the *Bellman equations* for state values and state-action values, respectively.

The Bellman equations for state values and state-action values both are systems of linear equations allowing to compute the value functions for all arguments without any need for exploration (if we know the environment dynamics p). In case of state values there are as many equations as there are states. In case of state-action values there are as many equations as there are state-action pairs.

The Bellman equations always have a solution, because for each policy there is a value function (the series in the explicit formula for value functions always converges). But could there be more than one solution? For $\gamma<1$ the answer is 'no' (see next section for a proof). For $\gamma=1$ uniqueness cannot be assured. Note that existence of a solution for $\gamma=1$ is only guaranteed for episodic tasks, not for continuing tasks.

47.2.2 Proof of Uniqueness of Solution to Bellman Equations

Assume there are two solutions v_1 and v_2 to the Bellman equations for state values for some policy π. Then for each $s \in \mathcal{S}$ we have

$$\begin{aligned}
\bigl|v_1(s) - v_2(s)\bigr| &= \Biggl|\sum_{a\in\mathcal{A}(s)} \pi(a,s) \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a)\,(r + \gamma\, v_1(s')) \\
&\qquad - \sum_{a\in\mathcal{A}(s)} \pi(a,s) \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a)\,(r + \gamma\, v_2(s'))\Biggr| \\
&= \Biggl|\sum_{a\in\mathcal{A}(s)} \pi(a,s) \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a)\,(r + \gamma\, v_1(s') - r - \gamma\, v_2(s'))\Biggr| \\
&= \gamma\,\Biggl|\sum_{a\in\mathcal{A}(s)} \pi(a,s) \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a)\,(v_1(s') - v_2(s'))\Biggr| \\
&\leq \gamma \sum_{a\in\mathcal{A}(s)} \pi(a,s) \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a)\,\bigl|v_1(s') - v_2(s')\bigr| \\
&\leq \gamma \sum_{a\in\mathcal{A}(s)} \pi(a,s) \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a)\,\max_{s''\in\mathcal{S}}\bigl|v_1(s'') - v_2(s'')\bigr| \\
&= \gamma \sum_{a\in\mathcal{A}(s)} \pi(a,s)\,\max_{s''\in\mathcal{S}}\bigl|v_1(s'') - v_2(s'')\bigr| \\
&= \gamma\,\max_{s''\in\mathcal{S}}\bigl|v_1(s'') - v_2(s'')\bigr|.
\end{aligned}$$

Here we used that all p-values (for fixed s and a) sum to 1 and that $\sum_a \pi(a,s) = 1$. Now taking the maximum over all s we see

$$\max_{s\in\mathcal{S}}\bigl|v_1(s) - v_2(s)\bigr| \leq \gamma \max_{s\in\mathcal{S}}\bigl|v_1(s) - v_2(s)\bigr|.$$

But for $\gamma \in [0,1)$ this is only possible if $v_1(s) = v_2(s)$ for all s. Thus, there cannot be two different solutions to the Bellman equations.

47.2.3 Policy Improvement

Given a policy we are able to compute corresponding value functions. Next step is to derive a better policy from the state or state-action value function. Here we say that a policy π_1 *is at least as good as* π_2 if

$$v_{\pi_1}(s) \geq v_{\pi_2}(s) \qquad \text{for all states } s \in \mathcal{S}$$

(and it's *better* if strict inequality holds at least for one state).

It turns out that a greedy policy with respect to the state value function v_π always is at least as good as the original policy π. This result is known as the *policy improvement theorem*.

Note that the policy improvement theorem for episodic tasks with $\gamma = 1$ only holds if both original policy and corresponding greedy policy always reach the end state within finitely many steps.

47.2.4 Proof of the Policy Improvement Theorem

Proof of the policy improvement theorem is quite simple. We give it for continuing tasks, for episodic tasks it's almost identical. Denote the greedy action with respect to v_π for state s by $a_{\mathrm{g}}(s)$ (if there are more than one greedy action, choose one of them). Then

$$q_\pi(s,a_{\mathrm{g}}(s)) = \max_{a\in\mathcal{A}(s)} q_\pi(s,a) \qquad \text{for all } s\in\mathcal{S}$$

and, thus,

$$\begin{aligned} v_\pi(s) &= \sum_{a\in\mathcal{A}(s)} \pi(a,s)\, q_\pi(s,a) \\ &\leq \sum_{a\in\mathcal{A}(s)} \pi(a,s)\, q_\pi(s,a_{\mathrm{g}}(s)) \\ &= q_\pi(s,a_{\mathrm{g}}(s)) \end{aligned}$$

for all $s\in\mathcal{S}$. With this estimate for each $s\in\mathcal{S}$ we obtain

$$\begin{aligned} v_\pi(s) &\leq q_\pi(s,a_{\mathrm{g}}(s)) \\ &= \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a_{\mathrm{g}}(s))\,(r+\gamma\, v_\pi(s')) \\ &\leq \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a_{\mathrm{g}}(s))\,(r+\gamma\, q_\pi(s,a_{\mathrm{g}}(s'))) \\ &= \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a_{\mathrm{g}}(s))\, r + \gamma \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a_{\mathrm{g}}(s))\, q_\pi(s,a_{\mathrm{g}}(s')) \\ &= \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a_{\mathrm{g}}(s))\, r \\ &\qquad + \gamma \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a_{\mathrm{g}}(s)) \sum_{s''\in\mathcal{S}}\sum_{r'\in\mathcal{R}} p(s'',r',s',a_{\mathrm{g}}(s'))\,(r'+\gamma\, v_\pi(s'')) \\ &= \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}} p(s',r,s,a_{\mathrm{g}}(s))\, r \\ &\qquad + \gamma \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}\sum_{s''\in\mathcal{S}}\sum_{r'\in\mathcal{R}} p(s',r,s,a_{\mathrm{g}}(s))\, p(s'',r',s',a_{\mathrm{g}}(s'))\, r' \\ &\qquad + \gamma^2 \sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}\sum_{s''\in\mathcal{S}}\sum_{r'\in\mathcal{R}} p(s',r,s,a_{\mathrm{g}}(s))\, p(s'',r',s',a_{\mathrm{g}}(s'))\, v_\pi(s'') \\ &\leq \ldots, \end{aligned}$$

that is, an estimate of the form

$$\begin{aligned} v_\pi(s) \leq\ &\text{expected reward if first action is greedy action} \\ &+ \gamma \times \text{expected reward if first two actions are greedy} \\ &+ \gamma^2 \times \text{expected reward if first three actions are greedy} \\ &+ \ldots. \end{aligned}$$

The right-hand side is exactly the state value function of the greedy policies (w.r.t. v_π). Thus, each greedy policy is at least as good as the original policy π.

Note that from the mathematical point of view the transition above from the finite sum to the infinite sum be repeating the step again and again is dangerous. To have a complete and correct proof some details would have to be filled in there. From those details one would see the troubles caused by $\gamma = 1$.

From the prove we also see, that a greedy policy is strictly better (in at least one state) if the original policy π allows for non-greedy actions in some state, that is, if there are s and a such that $\pi(a,s) > 0$ and $q_\pi(s,a) < q_\pi(s,a_{\mathrm{g}}(s))$.

47.2.5 Optimal Policies

One can prove (see next section) that **there always is a best policy**, that is, a policy π_* with

$$v_{\pi_*}(s) \geq v_\pi(s) \qquad \text{for all } s \in \mathcal{S} \text{ and for all policies } \pi.$$

Thus, for each finite Markov decision process there is a best solution. This optimal policy yields higher return than any other policy regardless of the initial state of the environment.

In other words, it's not possible to have two policies, one better in one state, the other better in another state, and both cannot be improved further. In such situations there always will be a third policy yielding higher return than those two policies.

As a consequence of the policy improvement theorem each optimal policy is a greedy policy w.r.t. its value function (else a corresponding greedy policy would be strictly better).

Given existence of an optimal policy π_* the definition of optimal policies implies following formulas for the optimal policies' value functions:

$$v_*(s) := v_{\pi_*}(s) = \max_\pi v_\pi(s) \qquad \text{for all } s \in \mathcal{S},$$

$$q_*(s,a) := q_{\pi_*}(s,a) = \max_\pi q_\pi(s,a) \qquad \text{for all } s \in \mathcal{S} \text{ and all } a \in \mathcal{A}(s).$$

Note that if there are more than one optimal policy, then all these optimal policies share one and the same value function v_* or q_* (by the definition of optimality). Thus, we may denote v_* and q_* as *optimal value functions* without referring to a concrete policy.

47.2.6 Proof of Existence of Optimal Policies

There exist at least two different existence proofs for optimal policies, both requiring deeper knowledge of mathematics. Here we show the basic ideas only to showcase the power of abstract mathematics. One proof exploits the Banach fixed-point theorem[629], the other is heavily based on Zorn's lemma[630], which is equivalent to the hotly debated[631] axiom of choice[632]. We follow the second variant.

The set of all policies has no linear order, because we cannot compare every policy to every other policy in terms of 'better' or 'worse' (pointwise for all states as introduced above). But we may find pairs of policies for which we can say that the one policy is better than the other. From such pairs we may form increasing chains of policies (from bad to good). If we can show that every such chain of policies has a maximal element (a best policy), then the Zorn lemma yields existence of a maximal element w.r.t. to the whole set of policies. That is, Zorn's lemma then guarantees existence of an optimal policy.

How can we show that each chain of policies has a maximal element? For finite chains obviously the last element is the maximal one. For infinite chains we argue as follows:

- Each infinite chain is a bounded set of functions (policies take value in $[0,1]$).
- Bounded sets always contain a convergent sequence, say $\pi_1, \pi_2, \ldots$ (that's the Bolzano-Weierstrass theorem[633]).
- Let π be the pointwise limit of the sequence, that is

$$\pi(a,s) := \lim_{n\to\infty} \pi_n(a,s) \qquad \text{for all } s \in \mathcal{S} \text{ and all } a \in \mathcal{A}(s).$$

- Now π is a policy, too (values in $[0,1]$, sum over actions is 1).
- The sequence of corresponding state value functions $v_1, v_2, \ldots$ converges, too (bounded increasing sequences always converge). Denote the pointwise limit by v.
- From the Bellman equations we see that v is the value function of π (swap sums and limits in Bellman equations).
- Thus, π is a better policy than policies $\pi_1, \pi_2, \ldots$ and, consequently, π is better than all policies in the chain under consideration.

[629] https://en.wikipedia.org/wiki/Banach_fixed-point_theorem
[630] https://en.wikipedia.org/wiki/Zorn%27s_lemma
[631] https://en.wikipedia.org/wiki/Axiom_of_choice#Criticism_and_acceptance
[632] https://en.wikipedia.org/wiki/Axiom_of_choice
[633] https://en.wikipedia.org/wiki/Bolzano%E2%80%93Weierstrass_theorem

47.2.7 Computing Optimal Policies (Optimal Bellman Equations)

Above we met the Bellman equations for computing value functions for given policies. Optimal policies all share identical state and state-action value functions. Thus, we should try to find Bellman equations for computing optimal value functions without referring to a conrete policy. From the optimal value function then we easily obtain an optimal policy: a greedy policy w.r.t. the optimal value function (this is a simple consequence of the policy improvement theorem).

The key ingredient for computing optimal value functions is the equation

$$v_*(s)=\max_{a\in\mathcal{A}(s)} q_*(s,a)\qquad\text{for all } s\in\mathcal{S}.$$

This equation is, again, a simple consequence of the policy improvement theorem, because there is an optimal greedy policy w.r.t. to v_* and for this greedy policy the equation is obviously true. Combining this equation with the equations relating state and state-action values (cf. derivation of Bellman equations) we obtain

$$v_*(s)=\max_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,(r+\gamma\,v_*(s'))\qquad\text{for all } s\in\mathcal{S},$$

$$q_*(s,a)=\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,\left(r+\gamma\max_{a'\in\mathcal{A}(s')}q_*(s',a')\right)\qquad\text{for all } s\in\mathcal{S}\text{ and all } a\in\mathcal{A}(s).$$

These are the *optimal Bellman equations* for state values and for state-action values in case of continuing tasks. For episodic tasks replace γ by 1. Note that the optimal Bellman equations are identical to the Bellman equations for the greedy policy w.r.t. the optimal value function.

For deriving the optimal Bellman equations we did not use optimality of v_* directly. Instead we only needed that the underlying policy is greedy with respect to its own value function. Might there be non-optimal policies with this property, too? If yes, the optimal Bellman equations would have multiple solutions, some of them being non-optimal value functions. Luckily, the answer is 'no' (see next section for a proof). This uniqueness result for the optimal Bellman equations also implies that every policy which is greedy w.r.t. to its own value function has to be optimal.

The optimal Bellman equations are a system of nonlinear equations, which cannot be solved directly. Instead, (iterative) methods for solving nonlinear equations have to be used. We know, that there always is a solution of this system, because we've proven existence of an optimal policy above.

If we know the environment dynamics p, then the optimal Bellman equations allow for numerically computing an optimal policy without the need for exploration. If we do not know p, we have to look for approximate solutions to the optimal Bellman equations based on data collected by the agent.

47.2.8 Proof of Uniqueness of Solution to Optimal Bellman Equations

For the proof we need the inequality

$$\left|\max_x f(x)-\max_x g(x)\right|\leq\max_x\bigl|f(x)-g(x)\bigr|$$

for arbitrary functions f and g taking arguments x from a finite set. To see that this inequality is true, assume $\max_x f(x)\geq\max_x g(x)$ (else, switch the roles of f and g) and let $\bar{x}$ be a maximizer of $f(x)$. Then

$$\begin{aligned}\left|\max_x f(x)-\max_x g(x)\right|&=f(\bar{x})-\max_x g(x)\\&\leq f(\bar{x})-g(\bar{x})\\&\leq\max_x\bigl|f(x)-g(x)\bigr|.\end{aligned}$$

Now assume there are two solutions v_1 and v_2 to the optimal Bellman equations for state values. Then for each $s\in\mathcal{S}$ we have

$$\begin{aligned}
\bigl|v_1(s)-v_2(s)\bigr| &= \Biggl|\max_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,(r+\gamma\,v_1(s'))\\
&\qquad-\max_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,(r+\gamma\,v_2(s'))\Biggr|\\
&\le \max_{a\in\mathcal{A}(s)}\Biggl|\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,(r+\gamma\,v_1(s')-r-\gamma\,v_2(s'))\Biggr|\\
&=\gamma\max_{a\in\mathcal{A}(s)}\Biggl|\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,(v_1(s')-v_2(s'))\Biggr|\\
&\le\gamma\max_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,\bigl|v_1(s')-v_2(s')\bigr|\\
&\le\gamma\max_{a\in\mathcal{A}(s)}\sum_{s'\in\mathcal{S}}\sum_{r\in\mathcal{R}}p(s',r,s,a)\,\max_{s''\in\mathcal{S}}\bigl|v_1(s'')-v_2(s'')\bigr|\\
&=\gamma\max_{a\in\mathcal{A}(s)}\max_{s''\in\mathcal{S}}\bigl|v_1(s'')-v_2(s'')\bigr|\\
&=\gamma\max_{s''\in\mathcal{S}}\bigl|v_1(s'')-v_2(s'')\bigr|.
\end{aligned}$$

Here we used that all p-values (for fixed s and a) sum to 1. Now taking the maximum over all s we see

$$\max_{s\in\mathcal{S}}\bigl|v_1(s)-v_2(s)\bigr|\le\gamma\max_{s\in\mathcal{S}}\bigl|v_1(s)-v_2(s)\bigr|.$$

But for $\gamma\in[0,1)$ this is only possible if $v_1(s)=v_2(s)$ for all s. Thus, there cannot be two different solutions to the optimal Bellman equations.

Note that for episodic tasks with $\gamma=1$ uniqueness cannot be assured.

CHAPTER

FORTYEIGHT

DYNAMIC PROGRAMMING

The term ‘dynamic programming’ is somewhat missleading. Here ‘programming’ is to be read as ‘optimization’ and ‘dynamic’ emphasizes the fact that we won’t solve one large optimization problem but a sequence of smaller ones yielding the a solution to the overall problem.

The general considerations in *Bellman Equations and Optimal Policies* (page 810) give rise to two concrete algorithms:

- Use the Bellman equations and the policy improvement theorem to iteratively improve an initial policy (*policy iteration*).
- Solve the optimal Bellman equations to directly obtain an optimal policy (*value iteration*).

Both algorithms assume, that we have complete knowledge of the environment dynamics p, which is rarely seen in practise (except for grid worlds). But these two algorithms will be the starting point for developing data-driven (exploration!) variants of them in subsequent chapters. Almost all reinforcement learning algorithms we consider in this book will look quite similar to policy or value iteration.

Related projects:

- *Frozen Lake* (page 1001)
  - *Dynamic Programming* (page 1001)

48.1 Policy Iteration

Based on the Bellman equations and the policy improvement theorem we may assemble the following algorithm:

1. Choose a random initial policy.
2. Solve corresponding Bellman equations.
3. Replace current policy by a greedy policy w.r.t. the solution of 2.
4. Go to 2 as long as the policy has changed.

Step 2 is referred to as *policy evaluation*. Step 3 is the *policy improvement* step. The stopping criterion will be satisfied if and only if the policy is optimal (remember that a policy is optimal if and only if it’s a greedy policy w.r.t. its own value function).

To solve the Bellman equations in step 2 we could create the coefficient matrix for the system of linear equations and use some numerical solver. For large state spaces this matrix requires lots of memory, but most entries are zero. Employing sparse matrix representations and specialized solvers may help, but there’s also a very simple and suffienctly efficient algorithm for solving Bellman equations based on the concrete structure of the system.

The structure of the Bellman equations for state values is

$$\underline{v}_\pi = B_\pi(\underline{v}_\pi),$$

where $\underline{v}_\pi \in \mathbb{R}^{|\mathcal{S}|}$ is the vector of values for all states and B_π multiplies its argument by a $\mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$-matrix and then adds a constant vector to the result. In other words, the value vector $\underline{v}_\pi$ for π is a *fixed point* of B_π. From the proof of uniqueness of the solution we easily see that for an arbitrary vector $\underline{v}$ we have

$$|\underline{v}_\pi - B_\pi(\underline{v})|_\infty = |B_\pi(\underline{v}_\pi) - B_\pi(\underline{v})|_\infty \leq \gamma\,|\underline{v}_\pi - \underline{v}|_\infty.$$

Say we start with some vector $\underline{v}_0$ and repeatedly apply B_π to obtain a sequence $\underline{v}_1, \underline{v}_2, \ldots$, then we have

$$|\underline{v}_\pi - \underline{v}_k|_\infty \leq \gamma^k \, |\underline{v}_\pi - \underline{v}_0|_\infty$$

and $\gamma^k \to 0$ if $k \to \infty$. This observation yields the following algorithm for (approximately) solving the Bellman equations in step 2 above:

1. Choose an initial value vector $\underline{v}_0$ (random, all zero,...).
2. For $k = 1, 2, \ldots$ iteratively compute $\underline{v}_k := B_\pi(\underline{v}_{k-1})$.
3. Stop the iteration if $|\underline{v}_k - \underline{v}_{k-1}|_\infty \leq \delta$ for some preset bound $\delta > 0$.

This algorithm always stops after finitely many steps.

48.2 Value Iteration

Alternatively to policy iteration we may solve the optimal Bellman equations to obtain an optimal policy more directly. The optimal Bellman equations are a system of nonlinear equations, but the solution vector $\underline{v}_*$ is a fixed point again:

$$\underline{v}_* = B(\underline{v}_*),$$

where B is the mapping defined by the right-hand side of the optimal Bellman equations. In complete analogy to solving the Bellman equations for policy iteration above we obtain the following algorithm:

1. Choose an initial value vector $\underline{v}_0$ (random, all zero,...).
2. For $k = 1, 2, \ldots$ iteratively compute $\underline{v}_k := B(\underline{v}_{k-1})$.
3. Stop the iteration if $|\underline{v}_k - \underline{v}_{k-1}|_\infty \leq \delta$ for some preset bound $\delta > 0$.

This yields (an approximation of) the optimal value function v_*. Each greedy policy w.r.t. this value function is an optimal policy.

48.3 Efficiency Considerations

Fixed-point iteration for systems of linear equations usually converges faster to the solution than for nonlinear equations. Policy iteration solves one system of linear equations per iteration, but often requires only a handful of iterations. Value iteration takes much more steps, but each step is very cheap (apply B). Thus, it's not clear which one is preferable.

For both algorithms we may increase efficiency with a simple trick. We may update $\underline{v}$ in-place and componentwise:

$$\underline{v}^{(l)} := [B(\underline{v})]_l \qquad \text{for } l = 1, \ldots, |\mathcal{S}|.$$

More precisely:

$$\underline{v}_k^{(1)} := [B(\underline{v}_{k-1})]^{(1)},$$

$$\underline{v}_k^{(l)} := \left[B\left(\begin{bmatrix} \underline{v}_k^{(1)} \\ \vdots \\ \underline{v}_k^{(l-1)} \\ \underline{v}_{k-1}^{(l)} \\ \vdots \\ \underline{v}_{k-1}^{(|\mathcal{S}|)} \end{bmatrix} \right) \right]^{(l)} \qquad \text{for } l = 2, \ldots, |\mathcal{S}|.$$

Thus, already updated components are directly used for updating the next component instead of holding the new values back until all components have been updated. As a side effect we do not need to store two vectors in memory, but only one, which may become relevant in case of large state spaces. Fixed-point iteration with this update rule is known as *asynchronous policy evaluation*.

Both algorithms' computation time is polynomial in the number of states and actions. Thus, for large state and/or action spaces they are very slow, but much faster than exhaustively searching the whole policy space.

CHAPTER

FORTYNINE

MONTE CARLO METHODS

Related projects:

- *Frozen Lake* (page 1001)
  - *Monte Carlo Methods* (page 1002)

In statistics 'Monte Carlo' means `do something random'. Monte Carlo methods in reinforcement learning use (more or less) random policies to explore the environment and average observed returns to get estimates for value functions. While Monte Carlo methods may be extended to continuing tasks, here we restrict our attention to episodic tasks only.

Monte Carlo methods require running a relatively large number of episodes to find a good policy. Thus, whenever possible, we should use a simulator of the real environment to find a sufficiently good policy before running the agent in its real-world environment. If this isn't possible (because there is no simulator) Monte Carlo methods aren't a good choice.

The major question to answer when developing Monte Carlo methods is how to ensure that all states are visited sufficiently often to get accurate information about rewards and follow-up states (remember, the environment may have random features). Here we discuss three different approaches:

- exploring starts (choose random start states and actions),
- ε-soft policies (all actions have nonzero probability),
- off-policy methods (use separate policies for exploration and exploitation).

49.1 Exploring Starts

If we have a simulated environment at hand, for each episode we may choose initial state and action at random. This ensures thorough exploration of the environment if the number of episodes run is high enough. Starting with some initial random policy we alternate policy evaluation and policy improvement in the same way we did with the policy iteration algorithm in *Dynamic Programming* (page 817).

The policy improvement step choses a greedy policy w.r.t. the estimated value function. We do not need to use an ε-greedy policy, because exploration is guaranteed by random initial states and actions. To estimate state or action values we compute mean returns over relevant episodes or subtrajectories thereof.

To estimate the value (expected return) $q_\pi(s,a)$ of some state-action pair (s,a) if following policy π, take all (sub-)traces available so far starting in s with action a, calculate corresponding returns, and set $Q_\pi(s,a)$ to be the average of all those returns. Here, $Q_\pi(s,a)$ is the estimate for the unknown exact value $q_\pi(s,a)$. This procedure requires storing all traces, which may be infeasible due to memory limitations. Efficiency can be increased by processing a trace directly after finishing the episode and only storing observed returns for each state-action pair:

1. For each state-action pair (s,a) let $G(s,a)$ be an empty list (of observed returns).
2. After each episode with trace $s_0,a_0,r_1,s_1,\dots s_{T-1},a_{T-1},r_T,s_T$ do:
   1. Set $g:=0$.
   2. For $t=T-1,T-2,\dots,0$ do:

1. Replace the value of g by $r_{t+1}+\gamma\,g$.
2. Append g to $G(s_t,a_t)$.
3. Set $Q_\pi(s_t,a_t)$ to the average of all returns in $G(s_t,a_t)$.

Sometimes this procedure is formulated for the first occurrence of a state-action pair in a trace only. Ignoring repeated visits to a state-action pair within one episode makes theoretical investigation much simpler, but is disadvantageous in practice. We should exploit collected data as much as possible.

The exploring starts approach works with state values, too. But for the policy improvement step we have to find a greedy policy w.r.t. to the action value function. But calculating action values from state values requires (complete) knowledge or good estimates of the envrionment dynamics. Estimating the environment dynamics is more difficult than estimating action values directly. If the environment behaves deterministically, then estimating state values is feasible and more efficient than estimating action values (there are much fewer state values than action values).

Depending on the concrete task to solve, policy evaluation may be reduced to only one trace per improvement step, if traces are sufficiently ridge.

49.2 ε-Soft Policies

If, like in most real-world settings without simulation, the exploring starts approach is not feasible but one wants to stick to on-policy methods, one has to use policies allowing for exploration.

A policy π is ε-*soft* for some $\varepsilon\in(0,1)$ if each action has nonzero probability bounded below by

$$\pi(a,s)\geq\frac{\varepsilon}{|\mathcal{A}(s)|},$$

where $|\mathcal{A}(s)|$ denotes the number of actions available in state s. Obviously, each ε-greedy policy is ε-soft.

One can show that ε-greedy policies w.r.t. an action value function q_π of an ε-soft policy π always are at least as good as π. This is the 'ε-version' of the policy improvement theorem.

Here is a concrete Monte Carlo method based on ε-soft policies:

1. Choose initial state s_0 (same for all episodes).
2. Choose some initial ε-soft policy π (uniformly random, for instance).
3. Get an estimate Q_π of the value function q_π from running several episodes (identical to exploring starts approach).
4. Replace π by a policy ε-greedy w.r.t. the value function estimate.
5. Go to 3 as long as policy changes.

If $a_{\mathrm{g}}(s)$ maximizes Q_π for s in step 4, an ε-greedy policy is given by

$$\pi(a,s)=\begin{cases}1-\varepsilon+\frac{\varepsilon}{|\mathcal{A}(s)|}, & \text{if } a=a_{\mathrm{g}}(s),\\ \frac{\varepsilon}{|\mathcal{A}(s)|}, & \text{else.}\end{cases}$$

Learning should start with relatively large ε. Later on ε should be decreased.

49.3 Importance Sampling

Importance sampling is a technique to estimate value functions for one policy from returns obtained by following another policy. This allows to develop off-policy methods. Such methods use a more or less constant *behavior policy* to explore the environment and continuously improve a *target policy* from the collected data. After suffcient exploration the target policy can be used to control the agent. This is very similar to A/B testing in *Stateless Learning Tasks (Multi-armed Bandits)* (page 801). The target policy does not explore at all. Thus, for non-stationary tasks exploration phases for updating the target policy from time to time have to be incorporated into the agent's overall behavior.

The overall algorithm is like before: alternate policy evaluation (estimate value function) and policy improvement (replace policy by greedy policy). But in the policy evaluation step we now have to estimate the value function of the target policy from data collected via the behavior policy.

The advantage of off-policy methods is, that they are able to find optimal policies. In contrast, on-policy methods only yield policies which always do some (non-optimal) exploration, too. Exploration in off-policy methods is faster, because the behavior policy may make the agent behave more randomly than the (iteratively optimized) policy in on-policy methods. On the other hand, the transformation of returns obtained from the behavior policy to returns for the target policy may introduce inaccuracies requiring for more episodes than on-policy methods to obtain good results.

49.3.1 Estimating Action Values

If π denotes current target policy and b denotes current behavior policy, to get estimates for action values q_π from returns obtained by following b the behavior policy has to allow at least for all actions which may be chosen by π:

$$\pi(a,s)>0\quad\Rightarrow\quad b(a,s)>0\qquad\text{for all } s\in\mathcal{S},\, a\in\mathcal{A}(s).$$

This property is sometimes referred to as *coverage assumption*.

To find an estimate $Q_\pi(s,a)$ of the value $q_\pi(s,a)$ of a state action pair (s,a), that is, an estimate of the expected return if starting at s with action a and following π, we may collect sufficiently many traces starting at (s,a) but following policy b, compute their returns, and then use the standard formula for expected values:

$$q_\pi(s,a)\approx\sum_{\text{observed traces}}\text{probability of trace under }\pi\times\text{observed return.}$$

But what is the 'probability of trace under π'? For a concrete trace $s_0,a_0,r_1,s_1,\ldots s_{T-1},a_{T-1},r_T,s_T$ with environment dynamics p we have

$$\text{probability of trace under }\pi=p(s_1,r_1,s_0,a_0)\,\pi(a_1,s_1)\,p(s_2,r_2,s_1,a_1)\cdots\pi(a_{T-1},s_{T-1})\,p(s_T,r_T,s_{T-1},a_{T-1}).$$

The probability that we observe the exactly same trace if we follow b instead of π (what we actually do) is

$$\text{probability of trace under }b=p(s_1,r_1,s_0,a_0)\,b(a_1,s_1)\,p(s_2,r_2,s_1,a_1)\cdots b(a_{T-1},s_{T-1})\,p(s_T,r_T,s_{T-1},a_{T-1}).$$

Thus,

$$q_\pi(s,a)\approx\sum_{\text{observed traces}}\text{probability of trace under }b\times\frac{\pi(a_1,s_1)\cdots\pi(a_{T-1},s_{T-1})}{b(a_1,s_1)\cdots b(a_{T-1},s_{T-1})}\times\text{observed return.}$$

The term $\frac{\pi(a_1,s_1)\cdots\pi(a_{T-1},s_{T-1})}{b(a_1,s_1)\cdots b(a_{T-1},s_{T-1})}$ is the *importance sampling ratio* of the trace under consideration.

The probability of observing a trace under b is a theoretical value. In practice we have to replace it by the relative frequency of the trace within all episodes under consideration. If each trace is observed only once or if we use one summand per episode, even if multiple episodes yielded the same trace, then 'probability of trace under b' can be replaced by 1 over the number of episodes:

$$q_\pi(s,a)\approx\sum_{\text{observed traces}}\frac{1}{\text{number of episodes}}\times\frac{\pi(a_1,s_1)\cdots\pi(a_{T-1},s_{T-1})}{b(a_1,s_1)\cdots b(a_{T-1},s_{T-1})}\times\text{observed return.}$$

Although above derivation is straight-forward, in practice the obtained estimate for q_π yields very poor results. The estimate often shows extremely large or small values. The reason lies in the difference between the probability of observing a trace and the relative frequency of the trace within the set of all observed traces. Importance sampling ratios transform probabilities of traces under b to probabilities of traces under π. But in practice we transform relative frequencies to relative frequencies. For small number of observed traces transformed relative frequencies will not sum up to 1. Thus, transformed relative frequencies do not correspond to a probability distribution anymore. To overcome this problem we should divide transformed relative frequencies by their total sum. With

$$h(\text{trace}) := \frac{\pi(a_1, s_1) \cdots \pi(a_{T-1}, s_{T-1})}{b(a_1, s_1) \cdots b(a_{T-1}, s_{T-1})}$$

we thus set

$$Q_\pi(s, a) := \frac{\sum\limits_{\text{observed traces}} h(\text{trace}) \times \text{observed return}}{\sum\limits_{\text{observed traces}} h(\text{trace})}.$$

This way of estimating $q_\pi(s, a)$ is known as *weighted important sampling*. Without division by the sum of transformed relative frequencies it's *ordinary importance sampling*.

If π is 0 for some state-action pair of an observed trace, then the whole trace is observed with probability 0 if following π. Such traces may appear if following the behavior policy b, but can be savely ignored when computing Q_π, because such traces do not carry any information about expected returns for π. If π is non-zero for all state-action pairs of a trace, the coverage assumption formulated above ensures that the importance sampling ratio is well-definded for that trace.

49.3.2 The Algorithm

Before we discuss some important limitations of the off-policy approach, we state the full algorithm for the off-policy Monte Carlo method with weighted importance sampling. A major difference to the derivation above is that we also consider all subtraces in addition to the complete trace of each episode to get as much information as possible from available date.

1. Let b be an arbitrary policy with $b(a, s) > 0$ for all s and a.
2. Set $Q(s, a)$ to arbitrary values (usually all zeros).
3. Let π be a greedy policy with some positive probability mass on all Q-maximizing actions.
4. Set $c(s, a) := 0$ for all pairs (s, a) (sum of importance sampling ratios of all traces starting with (s, a)).
5. Run episodes following b. After each episode with trace $s_0, a_0, r_1, s_1, \ldots s_{T-1}, a_{T-1}, r_T, s_T$ do:
    1. Set $g := 0$ (return).
    2. Set $h := 1$ (importance sampling ratio).
    3. For $t = T-1, T-2, \ldots, 0$ do:
        1. If $\pi(s_t, a_t) = 0$, stop iteration and proceed with next episode.
        2. Replace the value of g by $r_{t+1} + \gamma\, g$.
        3. Replace the value of h by $\frac{\pi(a_t, s_t)}{b(a_t, s_t)}\, h$.
        4. Replace the value of $c(s, a)$ by $c(s, a) + h$.
        5. Replace $Q_\pi(s_t, a_t)$ by $Q_\pi(s_t, a_t) + \frac{h}{c(s,a)}\,(g - Q_\pi(s_t, a_t))$.
    4. Let π be a greedy policy with some positive probability mass on all Q-maximizing actions.
    5. Modify b if there is some reason for (depends on application).

The algorithm may be stopped if changes in Q_π are below some prescribed bound.

Why we do not use an arbitrary greedy policy for π but a non-deterministic one (if possible) will be discussed in the next subsection.

For each episode we handle each subtrace as a trace on its own. Calculation of returns and importance sampling ratios is done in an incremental manner. Remember that $Q_\pi(s,a)$ is the weighted sum of returns of all (sub-)traces starting at (s,a). If $h_1,\ldots,h_n$ are corresponding importance sampling ratios, $c_k := \sum_{l=1}^k h_l$ is the sum of the first k ratios, $g_1,\ldots,g_n$ are the returns, and $Q_k := \frac{1}{c_k}\sum_{l=1}^k h_l\,g_l$, then

$$c_{k+1} = c_k + h_{k+1}$$

and

$$\begin{aligned}
Q_{k+1} &= \frac{h_{k+1}\,g_{k+1} + \sum\limits_{l=1}^{k} h_l\,g_l}{c_{k+1}}\\
&= \frac{h_{k+1}}{c_{k+1}}\,g_{k+1} + \frac{c_k}{c_{k+1}}\,Q_k\\
&= \frac{h_{k+1}}{c_{k+1}}\,g_{k+1} + \frac{c_{k+1}-h_{k+1}}{c_{k+1}}\,Q_k\\
&= \frac{h_{k+1}}{c_{k+1}}\,g_{k+1} + \left(1-\frac{h_{k+1}}{c_{k+1}}\right)\,Q_k\\
&= Q_k + \frac{h_{k+1}}{c_{k+1}}\,(g_{k+1}-Q_k),
\end{aligned}$$

which is exactly what happens in the algorithm above.

Traces containing a state action pair (s,a) with $\pi(s,a)=0$ do not modify the estimate $Q_\pi(s,a)$. Thus, we stop processing of subtraces of an episode as soon as we recognize such a pair (remember that we process subtraces starting with the tails).

The policy b may be changed at any time, for instance to focus exploration on certain aspects or regions of the environment.

49.3.3 Limitations

Having too many traces with $\pi(s,a)=0$ for some state-action pair (s,a) should be avoided, because such traces are useless for improving π. If π is a deterministic policy, then a trace will contain a pair with zero π as soon as the trace differs from a trace obtained by following π. This renders the off-policy approach useless because all truely exploring traces will be dropped. Only traces identical to traces obtainable via π will have influnce on Q_π. But then we could use an on-policy approach directly.

The only way out is to keep π non-deterministic as long as possible. During first iterations that's not difficult. But after sufficiently many updates to Q_π there will be only one maximizing action for most states. In this stage the off-policy approach behaves almost like on-policy techniques with large overhead (drops most traces). For deterministic π exact behavior in each episode is:

1. Take a random action in first step (according to b).
2. Follow π for all further steps.

Traces or subtraces not following this scheme will be dropped in case of deterministic π.

The described problem may lead to very slow learning. But there also exist some modifications to (at least partially) overcome this limitation. See Sutton/Barto's book[634], section 5.8 and 5.9 for more information.

[634] https://webspace.fh-zwickau.de/jef19jdw/teaching/pti01840/sutton_barto.pdf

CHAPTER

FIFTY

TEMPORAL DIFFERENCE LEARNING (TD LEARNING)

Related projects:

- *Frozen Lake* (page 1001)
  - *SARSA* (page 1003)

With Monte Carlo methods we have to wait till the end of the first episode before the agent learns anything. Especially for tasks with very long episodes it would be preferable if the agent could already learn something during the episode. Take a grid world with walls, where the agent obtains positive reward for reaching a destination cell and negative reward for hitting a wall. Learning about the location of the destination cell requires at least one full episode. But learning about the location of the walls (and how to avoid hitting them) should be possible during an episode. Thus, the agent souldn't hit the same wall more than one or two times. This is possible with temporal difference learning.

The overall setting is as usual: alternate policy evaluation and policy improvement. The difference is in the policy evaluation step, that is, in estimating the action value function q_π.

50.1 Policy Evaluation

If we have a trace $s_0, a_0, r_1, s_1, a_1, r_2, s_2, ...$, immediately after deciding for the next action a_{t+1} and obtaining the reward r_{t+1} the estimate $Q_\pi(s_t, a_t)$ for $q_\pi(s_t, a_t)$ is updated via

$$Q_\pi(s_t, a_t) := Q_\pi(s_t, a_t) + \alpha\,(r_{t+1} + \gamma\,Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t)).$$

Here $\alpha \in (0, 1]$ is a step size and $r_{t+1} + \gamma\,Q_\pi(s_{t+1}, a_{t+1})$ is an estimate for the return $q_\pi(s_t, a_t)$.

This value function update works for both episodic and continuing tasks. One can show that for fixed policy π the estimate Q_π converges to q_π if α is chosen small enough and if each state-action pair gets visited infinitely many times.

Note that for TD learning each end state in an episodic task has to allow for at least one action, else there is no Q-value to estimate the value from for the state-action pair leading to the end state. But this is more or less an implementation detail only.

In TD learning the rewards flow through the set of state-action pairs in reverse direction of the traces. Take a 1-by-5 grid world, for instance. The agent always starts at the left most position and the goal is at the right most position. The policy is to always go to the right and never to the left. Reward for reaching the goal is 1. All other rewards are zero. The table below shows the development of state-action values for $\gamma = 1$ and $\alpha = 0.5$. Note that the agent does not collect any information about what happens if it goes to the left. Thus, we have no estimates for corresponding state-action pairs.

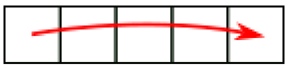


Fig. 50.1: The agent always moves from the left to the right.

episode	new state	$Q_\pi(1,\mathrm{R})$	$Q_\pi(2,\mathrm{R})$	$Q_\pi(3,\mathrm{R})$	$Q_\pi(4,\mathrm{R})$	$Q_\pi(5,\mathrm{R})$
-	-	0	0	0	0	0
1	2	0	0	0	0	0
1	3	0	0	0	0	0
1	4	0	0	0	0	0
1	5	0	0	0	0.5	0
2	2	0	0	0	0.5	0
2	3	0	0	0	0.5	0
2	4	0	0	0.25	0.5	0
2	5	0	0	0.25	0.75	0
3	2	0	0	0.25	0.75	0
3	3	0	0.125	0.25	0.75	0
3	4	0	0.125	0.625	0.75	0
3	5	0	0.125	0.625	0.875	0
4	2	0.0625	0.125	0.625	0.875	0
…	…	…	…	…	…	0

For TD learning to work the initial estimate for Q_π has to be zero for all end states.

50.2 SARSA (On-Policy TD Learning)

The SARSA algorithm alternates policy evaluation and policy improvement, but improves the policy after each step of above policy evaluation iteration instead of waiting for convergence of the value function. To ensure sufficient exploration an ε-greedy policy based on the current value function estimate is used. The name SARSA reflects the fact, that the algorithm requires $s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1}$ for the value function update.

1. Choose $\varepsilon > 0$ and $\alpha \in (0,1]$.
2. Choose initial π (uniformly random, for instance) and initial Q_π (all zero).
3. Run episodes (for continuing tasks there's only one episode):
    1. Choose initial state s and initial action a.
    2. For each step do:
        1. Take action a and observe reward r and new state s'.
        2. Choose a' according to π.
        3. Replace $Q_\pi(s,a)$ by $Q_\pi(s,a) + \alpha\,(r + \gamma\,Q_\pi(s',a') - Q_\pi(s,a))$.
        4. Set π to be ε-greedy w.r.t. Q.
        5. Replace (s,a) by (s',a').
        6. If s' is an end state, go to next episode.

The exploration rate ε should be decreased over time.

50.3 Q-Learning (Off-Policy TD Learning)

SARSA updates $Q_\pi(s,a)$ by looking at the value of the next state-action pair (s',a') the current (non-optimal) policy suggests. Because the policy is not greedy (exploration!) a' won't allways be the highest rated action. But for the update of Q_π we do not have to stick to the (non-optimal) behavior suggested by π. Instead we could simply use the best follow-up action to estimate current state-action pair's value:

$$Q(s,a) := Q(s,a) + \alpha \left(r + \gamma \max_{\tilde{a} \in \mathcal{A}(s')} Q(s',\tilde{a}) - Q(s,a) \right).$$

Note that we dropped the subscript π because Q no longer approximates the action values of the ε-greedy policy π used by the agent for choosing the next action. Q now approximates action values of the greedy policy w.r.t. Q from the previous iteration. SARSA with this modified update formula is an off-policy method denoted as *Q-learning*. Convergence of Q to an optimal value function typically is faster than for on-policy SARSA.

50.4 Double Q-Learning

Q-Learning tends to overestimate action values. SARSA at least in some steps chooses non-optimal actions. Thus, $Q(s',a')$ in the estimated $r + \gamma\, Q(s',a')$ sometimes is below the average and sometimes above. But Q-learning always updates with the highest estimated (!) Q value. Thus, estimates will be greater than the true action values on average.

To overcome this problem we should use two estimates Q_1 and Q_2. One of them is used to choose the optimal action. The other then is used to determine the value of the chosen action. Alternating both roles yields a technique which averages two estimates of the optimal action value. Thus, there's some chance that one estimate is below and the other above the true value.

Double Q-learning looks like Q-learning, but in each step we randomly choose one of two possible updates:

$$Q_1(s,a) := Q_1(s,a) + \alpha \left(r + \gamma\, Q_1 \left(s', \operatorname*{argmax}_{\tilde{a} \in \mathcal{A}(s')} Q_2(s',\tilde{a}) \right) - Q_1(s,a) \right)$$

or

$$Q_2(s,a) := Q_2(s,a) + \alpha \left(r + \gamma\, Q_2 \left(s', \operatorname*{argmax}_{\tilde{a} \in \mathcal{A}(s')} Q_1(s',\tilde{a}) \right) - Q_2(s,a) \right).$$

The ε-greedy policy then is based on the average of both Q_1 and Q_2.

Note that the idea of double learning also works for SARSA, but SARSA does not tend to overestimate action values as much as Q-learning does.

CHAPTER

FIFTYONE

APPROXIMATE VALUE FUNCTION METHODS

Instead of using tables (tabular methods) to represent value functions, for task with large state and action spaces approximate representations of value functions based on parametrizations should be used, especially if tabular methods render infeasible regarding memory consumption and computation times.

Related projects:

51.1 Principal Approach

Tabular value function methods are well suited for reinforcement learning tasks with few actions and not too many different states. The table assigning values (expected return) to each state-action pair has to fit into memory. But what about tasks with very large, maybe continuous or otherwise infinite action and/or state spaces? Here we cannot explicitely calculate and store estimates of expected returns for all state-action pairs.

Approximate value function methods use parametrized functions to represent value functions. Instead of estimating expected return for all state-action pairs, here we have to compute parameter values such that corresponding function yields a good estimate for expected return of all relevant state-action pairs. This way memory consumptions for storing a value function is independent of state and action space size.

All kinds of function approximation are available here: linear or polynomial functions, linear combinations of radial basis functions, decision trees, artificial neural networks (ANNs) and so on. In applications the main focus is on ANNs. Thus, instead of parameters we will use the word ‘weight’ to be consistent with the literature.

51.1.1 Challenges with Large State Spaces

Large state spaces often induce further problems (next to memory consumption). Especially for continuous state spaces, values of neighboring states will heavily influence each other. On the one hand, thus we cannot handle states independently. On the other hand, this allows to deduce a state's value from neighboring states' values. With parametrized value functions we exploit this *generalization* property of state values, because there are much fewer parameters than states. Each parameter controls many states, not allowing for individual values.

Controlling the learning process, that is, exploring the environment and estimating state or state-action values, is much more difficult for large state and action spaces. The agent will explore only a tiny fraction of the whole environment. Thus, we have to take care that this tiny fraction is as relevant as possible for 'understanding' the environment's structure.

51.1.2 Algorithm Structure

Algorithms based on approximate value functions follow the same structure like algorithms based on tabular value functions. Starting with an initial policy the policy's value function has to be estimated by exploring the environment (*policy evaluation*). Then, based on the value function estimate a new improved policy can be derived, an ε-greedy policy, for instance (*policy improvement*).

Like for tabular methods, policy evaluation and policy improvement can be combined in many different ways to obtain efficient algorithms. On-policy and off-policy approaches are available, too.

51.2 Policy Evaluation

In preparation of developing concrete reinforcement learning algorithms we think about strategies for computing the weights of a value function. We restrict our attention to state-action values (action values for short), because they are much more relevant in practice than state values. For state values everything looks quite similar and can easily be deduced from the considerations we describe for action values.

Given the environment (mapping state-action pairs to states and rewards) and a policy π we want to find the value function q_π, that is, the expected return for all state-action pairs. Since we aim at approximate value function methods, we look for an estimate Q_w of q_π, where $w \in \mathbb{R}^p$ is the vector of weights by which Q_w is completely determined.

51.2.1 Supervised Learning

Computing Q_w is a typical supervised learning problem: find weights w such that Q_w maps state-action pairs (inputs) to expected returns (outputs). We have to solve to problems:

- Collect training samples.
- Choose a concrete model for Q_w and calculate w from training samples.

Collecting trainings samples is equivalent to exploration in reinforcement learning. But the direct outcome of exploration are rewards, not expected returns. Thus, we need to give some thought to how to deduce expected returns from observed rewards.

In reinforcment learning the model Q_w almost always is an ANN. Thus, calculating weights w requires a loss function, which then is minimized with repspect to w via some gradient descent method.

51.2.2 Training Targets

The training targets, that is, the outputs of the model Q_w, are expected returns, but from exploration we only have rewards at hand. Thus, expected return has to be estimated from observed rewards. We already solved this problem in several different ways for tabular methods. Here we may use exactly the same ideas:

- **Monte Carlo methods:** Run a full episode and calculate actual return for each state-action pair visited in the episode. If a pair is visited multiple times (maybe in several episodes), use the mean of all observed returns as an estimate for the expected return, that is,

$$q_\pi(s,a) \approx \text{mean of observed returns for episodes starting at } (s,a).$$

- **SARSA (on-policy TD learning):** Look one step ahead, that is, if (s,a) results in reward r and state-action pair (s',a'), use

$$q_\pi(s,a) \approx r + \gamma\, Q_w(s',a').$$

- **Q-learning (off-policy TD learning):** Look one step ahead and use a greedy policy for action selection, that is, if (s,a) results in reward r and state s', use

$$q_\pi(s,a) \approx r + \gamma\, \max_{a'} Q_w(s',a').$$

Each of these three tabular methods gives rise to an approximate value function method. Full algorithms for SARSA and Q-learning will be developed in *Policy Improvement* (page 832) and *Deep Q-Learning* (page 837), respectively.

51.2.3 Loss Function and Gradient

Given training samples $(s_1,a_1,y_1),\ldots,(s_n,a_n,y_n)$ with inputs (s_l,a_l) and targets y_l we want to find the weigths w of Q_w by gradient descent. The loss function for training is usual mean squared error:

$$L(w) := \sum_{l=1}^{n} \bigl(Q_w(s_l,a_l) - y_l\bigr)^2$$

State-action pairs visited several times will occur several times in the training data, giving them more weight in the training process.

Computing the gradient of L with respect to w is difficult because the targets y_k may depend on w, too (in case of SARSA and Q-learing). A simple way out is to ignore this dependency completely:

$$\nabla L(w) = 2\sum_{l=1}^{n} \bigl(Q_w(s_l,a_l) - y_l\bigr)\,\nabla_w Q_w(s_l,a_l).$$

At the first glance this seems a bit dubious, but there's good justification for this approach: To get training samples we had to replace the true target (expected return) by some approximation. If we switch this approximation step and the computation of the gradient, targets in L do not depend on w anymore. Thus, computing the gradient of L is simple. But now the gradient contains expected rewards, which are inaccessible in practice. So we replace them by one of the approximations given above. The outcome is completely equivalent to the original approach (approximate targets, ignore targets' dependence on w).

Sometimes instead of gradient the term *semi-gradient* is used to emphasize the fact, that something is not standard here.

51.3 Policy Improvement

For approximate value functions policy improvement should work as usual: get a good estimate of the value function for some initial policy, get corresponding (ε-)greedy policy, get an estimate of the value function for the new policy, and so on. This procedure should yield better policies (higher expected returns) step by step. But looking at the details we are facing several problems.

51.3.1 Problem 1: Large Action Space

Given a value function estimate Q_w, following corresponding (ε-)greedy policy requires maximization of $Q_w(s,a)$ with respect to a (and fixed current state s). For large action spaces calculating $Q_w(s,a)$ for all a is too expensive or even impossible (due to memory limitations). Thus, numerical minimization techniques have to be employed, which typically are slow, expensive and unreliable.

With ANNs in mind gradient based numerical optimization with respect to actions is equivalent to maximizing ANN output with respect to ANN inputs. This requires automatic differentiation as well as one ANN evaluation per gradient step.

Examples for large action spaces are the steering angle in autonomous driving (continuous action space) and choosing web ads (each available ad is an action).

Up to now there is no commonly accepted reinforcement learning technique for large action spaces!

The only way out is to enforce sufficiently small discrete action spaces. Continuous action spaces have to be discretized and similar discrete actions possibly have to be joined into one action. For steering angles this could mean, for instance, that the only allowed angles are multiples of 10 degree.

51.3.2 Problem 2: No Optimal Policy

For tabular methods we know that there always is an optimal policy. Remember that a policy is optimal, if its value function is nowhere smaller than the value function of any other policy. For approximate value function methods this result no longer holds true.

The reason is not quite obvious. In approximate value function methods we first decide for a (then fixed) parametrization scheme. All value function estimates have to follow this scheme, that is, they all have the same structure with the same number of weights (parameters). The only difference are the concrete values for the weights. Although there might be an optimal value function, there's only little chance that it can be represented by the chosen parametrization scheme. Thus, value function estimates of all (!) policies will differ significantly from the optimal value function if the optimal value function is not representable by the chosen scheme. Even if we have found the optimal policy we won't recognize this fact, because its value function estimate will significantly differ from the optimal value function and, in addition, we do not have prior knowledge about the optimal value function in practice.

To illustrate this problem take a reinforcement learning task with four different states s_1, s_2, s_3, s_4 and five actions A, B, C, D, E. The figure below defines the behavior of the deterministic environment.

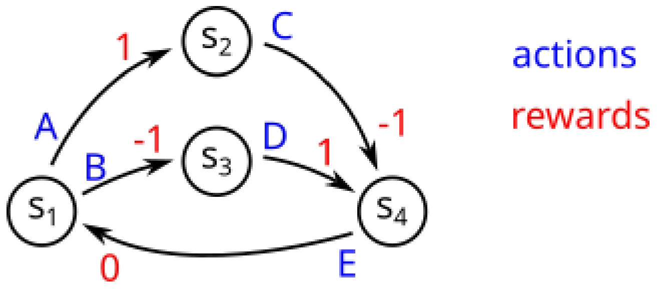


Fig. 51.1: The environment dynamics for the deterministic environment. Taking action B in state s_1 yields reward -1 and brings the environment into state s_3, for instance. In state s_1 only actions A and B are available, in state s_2 there's only one action, and so on.

Our model for value function estimates Q_w has three weights and following structure:

$$\begin{aligned} Q_w(s_1,\mathrm{A}) &:= w_1\\ Q_w(s_1,\mathrm{B}) &:= w_1\\ Q_w(s_2,\mathrm{C}) &:= w_2\\ Q_w(s_3,\mathrm{D}) &:= w_2\\ Q_w(s_4,\mathrm{E}) &:= w_3 \end{aligned}$$

Note that there are fewer weights than state-action pairs, which is the typical situation for large state and action spaces.

For simplicity we set the discounting parameter $\gamma := 0$, that is, return equals immediate reward. The optimal value function then assigns immediate rewards to all state-action pairs:

$$\begin{aligned} q_*(s_1,\mathrm{A}) &:= 1\\ q_*(s_1,\mathrm{B}) &:= -1\\ q_*(s_2,\mathrm{C}) &:= -1\\ q_*(s_3,\mathrm{D}) &:= 1\\ q_*(s_4,\mathrm{E}) &:= 0 \end{aligned}$$

For the task under consideration there are only two (deterministic) policies, because there's only one state (s_1) allowing for more than one action. Let π_A be the policy choosing action A in s_1 and let π_B be the other policy. Denote the weight vectors for corresponding value function estimates by w^A and w^B.

With π_A the state action pair (s_1,B) is never visited. Thus, w_1^A is determined by the reward for taking A in s_1. Pair (s_3,D) never gets visited, too (except for starting in s_3). Thus, w_2^A is determined by the reward for taking C in s_2. Analogous observations hold true for w_1^B (determined by taking B in s_1) and w_2^B (determined by taking D in s_3). Value function estimates are:

$$\begin{aligned} Q_{w^\mathrm{A}}(s_1,\mathrm{A}) &:= 1 & \qquad Q_{w^\mathrm{B}}(s_1,\mathrm{A}) &:= -1\\ Q_{w^\mathrm{A}}(s_1,\mathrm{B}) &:= 1 & Q_{w^\mathrm{B}}(s_1,\mathrm{B}) &:= -1\\ Q_{w^\mathrm{A}}(s_2,\mathrm{C}) &:= -1 & Q_{w^\mathrm{B}}(s_2,\mathrm{C}) &:= 1\\ Q_{w^\mathrm{A}}(s_3,\mathrm{D}) &:= -1 & Q_{w^\mathrm{B}}(s_3,\mathrm{D}) &:= 1\\ Q_{w^\mathrm{A}}(s_4,\mathrm{E}) &:= 0 & Q_{w^\mathrm{B}}(s_4,\mathrm{E}) &:= 0 \end{aligned}$$

Because

$$Q_{w^\mathrm{A}}(s_3,\mathrm{D}) < q_*(s_3,\mathrm{D}) \qquad \text{and} \qquad Q_{w^\mathrm{B}}(s_1,\mathrm{A}) < q_*(s_1,\mathrm{A}),$$

we cannot decide which of the two policies is the optimal one.

To solve the problem that with approximate value functions finding an optimal policy might be impossible we will have to weaken the notion of optimality, see below.

51.3.3 Problem 3: No Improvement

Improving the value estimate for one state-action pair does not necessarily improve corresponding (ε-)greedy policy, because other values will change, too. Maybe we can't even say which one is better, original or 'improved' one.

The only way out is to weaken the notion of 'better' when comparing policies.

51.3.4 Weakening the Notion of Optimality

Motivated by problems 2 and 3 above we want to weaken the notion of ‘better’ when comparing policies, which then implies a weaker notion of optimality, too.

Remember that for tabular methods the notion of optimality of a policy is based on pointwise properties of state value functions. Exactly this pointwise approach leads to problems 2 and 3 above for policies based on approximate value functions. A way out is to find a quality measure for policies consisting of only one real number. Then we may compare the quality of policies via that number.

Given the state value function v_π of a policy π the *average state value* of π is

$$V(\pi) := \sum_{s\in\mathcal{S}} \mu_\pi(s)\, v_\pi(s),$$

where $\mu_\pi(s)$ is the probability of visiting state s if starting at some state chosen uniformly at random and then following π. In case of infinitely many discrete states $V(\pi)$ is given by a series. In case of a continuous state space $V(\pi)$ is given by an integral and μ_π is a probability density function. In all what follows we assume that there are finitely many states only to simplify notation. But derived algorithms will work for infinite and continuous state spaces, too.

We now may say that a policy π_1 *is at least as good as* π_2 if $V(\pi_1) \geq V(\pi_2)$. A policy π is *optimal* if

$$V(\pi) \geq V(\tilde{\pi}) \qquad \text{for all policies } \tilde{\pi}.$$

With this weaker notion of optimality we do not see anymore in which state a non-optimal policy has to be improved. But at least we are able to compare policies again, which in turns allows to decide whether a policy can be improved further (or has been improved by some manipulation).

51.3.5 Deprecating Discounting

A surprising consequence of weakening the notion of optimality is that the discounting factor γ used in the return definition for continuing tasks does not have any influence on the ordering of policies: The Bellman equations for state values yield

$$\begin{aligned} V(\pi) &= \sum_{s\in\mathcal{S}} \mu_\pi(s)\, v_\pi(s) \\ &= \sum_{s\in\mathcal{S}} \mu_\pi(s) \sum_{a\in\mathcal{A}(\mathcal{S})} \pi(a,s) \sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}} p(s',r,s,a)\,(r+\gamma\, v_\pi(s')), \end{aligned}$$

where p denotes the environment dynamics. With

$$r(\pi) := \sum_{s\in\mathcal{S}} \mu_\pi(s) \sum_{a\in\mathcal{A}(s)} \pi(a,s) \sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}} p(s',r,s,a)\, r$$

and reordering summation this becomes

$$\begin{aligned} V(\pi) &= r(\pi) + \sum_{s\in\mathcal{S}} \mu_\pi(s) \sum_{a\in\mathcal{A}(\mathcal{S})} \pi(a,s) \sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}} p(s',r,s,a)\,\gamma\, v_\pi(s') \\ &= r(\pi) + \gamma \sum_{s'\in\mathcal{S}} v_\pi(s') \sum_{s\in\mathcal{S}} \mu_\pi(s) \sum_{\substack{a\in\mathcal{A}(s)\\ r\in\mathcal{R}}} \pi(a,s)\, p(s',r,s,a) \\ &= r(\pi) + \gamma \sum_{s'\in\mathcal{S}} v_\pi(s')\, \mu_\pi(s') \\ &= r(\pi) + \gamma\, V(\pi). \end{aligned}$$

Isolating $V(\pi)$ now yields

$$V(\pi) = \frac{1}{1-\gamma}\, r(\pi).$$

The discounting parameter γ is just a scaling factor in average state values. Thus, if some policy π_1 is better than π_2 for small γ (long-term rewards not of importance), π_1 will be better than π_2 for γ close to 1 (long-term rewards are of importance), too. Although for value functions γ does not matter anymore, for value function estimates the choice of γ may still have some influence on the quality of the estimate in terms of convergence to the true value function.

51.3.6 Differential Return

The quantity

$$r(\pi) := \sum_{s\in\mathcal{S}} \mu_\pi(s) \sum_{a\in\mathcal{A}(s)} \pi(a,s) \sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}} p(s',r,s,a)\, r$$

in above derivation is known as *average reward* of the policy π. It's the reward we can expect in each step if following π after starting at some state chosen uniformly at random.

Since the discounting factor γ does not influence ordering of policies given by average state values $V(\pi)$, we may use $r(\pi)$ for definition of optimality in case of approximate value functions: A policy is optimal if it maximizes average reward. And π_1 is at least as good as π_2 if $r(\pi_1) \geq r(\pi_2)$. With this, γ vanished from the definition of optimality without changing the meaning.

The question is whether we can get rid of γ in the definition of returns, too. Remember that we introduced discounting to avoid infinite returns in case of continuing tasks. So we have to find an alternative way for avoiding infinite returns. Indeed, instead of summing rewards directly we may sum differences between rewards and average reward: Given a trace $s_0, a_0, r_1, s_1, a_1, r_1, \ldots$ we look at

$$G(\text{trace}) := r_1 - r(\pi) + r_2 - r(\pi) + \ldots .$$

This quantity is known as *differential return*. Each summand $r_t - r(\pi)$ is close to zero in the sense that its expectation (or its average over many traces) is zero. Thus, there's good chance (but no guarantee!) that $G(\text{trace}) < \infty$.

The definition of value functions does not change. The value $v_\pi(s)$ of a state s still is the expected return if starting in s and following π. The value $q_\pi(s,a)$ of a state-action pair (s,a) still is the expected return if starting in s with action a and then following π. But instead of discounted return we now use differential return.

The state value $v_\pi(s)$ may be regarded as the difference between expected return obtained from starting at s and the policy's average expected return $V(\pi)$. Analogously, $q_\pi(s,a)$ may be regarded as the difference between expected return obtained from taking action a in start state s and the policy's average average expected return $V(\pi)$.

51.3.7 Discounted or Differential Return?

We now have two settings for computing return in continuing tasks, discounted return and differential return. The questions, which one to prefer, has no clear answer. Both variants are used in practice and no final decision has been made up to now by the reinforcement learning community.

We introduced differential return as a consequence of weakening the notion of optimality for policies, which in turn originated from considering approximate value functions. Although the discounted return setting lead to some trouble with theory (possible non-approximability of optimal value functions), algorithms based on discounting still work with approximate value functions. Of course, together with optimal policies we also lost the policy improvement theorem, which has been the motivation for all our algorithms. But for the differential return setting we do not have a policy improvement theorem, too. We have to live with the fact, that all algorithms described below for approximate value functions lack theoretical justification. We simply hope that the idea of the policy improvement theorem also works for approximate value functions sufficiently well.

In practice, differential return does not require us to choose a parameter for discounting. Having fewer parameters to choose values for generally is a very good thing. But in turn, with differential returns all past exploration data has identical influence on the agent's behavior. In the discounted setting the agent forgets about the long past and adapts to changing environments more rapidly (depending on the choice of γ). From the theoretical point of view there's no difference between forgetting and not forgetting the past. A policy either is (weakly) optimal in both cases or in non of both. But in practice, when expected returns in fact are estimates obtained from few observations, discounting may yield faster learning progress in some situations.

51.3.8 SARSA with Discounted Return

Now we are ready to formulate concrete algorithms. Here we focus on on-policy methods (SARSA). Off-policy methods (Q-learning) will be discussed later on.

1. Initialization
    1. Choose $\gamma \in [0,1)$ (discounting), $\alpha > 0$ (step size for gradient descent) and $\varepsilon \in (0,1)$ (for ε-greedy policy).
    2. Choose initial weights w for Q_w (at random, for instance).
    3. Choose initial state s and initial action a (at random, for instance).
2. Step
    1. Take action a and observe reward r and new state s'.
    2. Choose next action a' by ε-greedy policy w.r.t. Q_w.
3. Update
    1. Replace weights w by

$$w + \alpha\,(r + \gamma\,Q_w(s', a') - Q_w(s, a))\,\nabla_w\,Q_w(s, a).$$

    2. Replace s by s' and a by a'.
    3. Go to 2.1.

In step 3.1 the term $r + \gamma\,Q_w(s', a')$ is the new return estimate. Cf. *Policy Evaluation* (page 830) for the gradient of the loss function.

Note that there's no stopping criterion, because in case of continuing tasks the algorithm shall run forever. If the task is episodic, the algorithm stops if an end state has been reached (no more action to choose from). The last weight update then uses r instead of $r + \gamma\,Q_w(s', a')$, because return equals reward if reaching an end state. For each new episode the algorithm is rerun, but without steps 1.1 and 1.2.

51.3.9 SARSA with Differential Return

Computing differential return requires knowledge of the average reward $r(\pi)$, which in turn requires knowledge of the environment dynamics. If we do not have such knowledge (this is the common situation), we have to estimate $r(\pi)$ from data collected by the agent. If $r_1, r_2, \ldots$ is the sequence of observed rewards a simple estimate $\overline{r}_t$ after time step t for $r(\pi)$ is given by

$$\overline{r}_1 := r_1, \qquad \overline{r}_t := \overline{r}_{t-1} + \beta\,(r_t - \overline{r}_{t-1}) \quad \text{for } t > 1,$$

where $\beta > 0$ controls how fast the estimates adapts to most recent rewards. Note that this is not exactly an estimate for $r(\pi)$, because very old rewards have less influence on the estimate (cf. discounted return). But this estimate allows for better adaption to changing environment dynamics in non-stationary tasks.

To simplify algorithms one may use $\overline{r}_0 := 0$ instead of $\overline{r}_1 := r_1$ for initialization.

Initialization and step are identical (up to initialization of β and $\overline{r}$) for both SARSA with discounted return and SARSA with differential return. Only the weight update is different and for differential returns we have to update the estimate for the average reward in each step.

1. Initialization
    1. Choose $\alpha > 0$ (step size for gradient descent), $\beta > 0$ (step size for average reward update) and $\varepsilon \in (0,1)$ (for ε-greedy policy).
    2. Choose initial weights w for Q_w (at random, for instance) and set $\overline{r} := 0$.
    3. Choose initial state s and initial action a (at random, for instance).
2. Step

   1. Take action a and observe reward r and new state s'.
   2. Choose next action a' by ε-greedy policy w.r.t. Q_w.
3. Update
   1. Replace $\overline{r}$ by $\overline{r}+\beta\,(r-\overline{r})$.
   2. Replace weights w by

$$w+\alpha\,(r-\overline{r}+Q_w(s',a')-Q_w(s,a))\,\nabla_w\,Q_w(s,a).$$

   3. Replace s by s' and a by a'.
   4. Go to 2.1.

51.4 Deep Q-Learning

Tabular Q-learning works like tabular SARSA but uses the greedy action's value instead of the chosen action's value for updating the value function estimate Q. This makes Q-learning an off-policy method, because Q approximates the value function of its greedy policy while for exploration an ε-greedy policy is used.

In principle, Q-learning with approximate value functions can be implemented like SARSA for approximate value functions. The update formula then is

$$w+\alpha\,(r+\gamma\max_{a'\in\mathcal{A}(s')}Q_w(s',a')-Q_w(s,a))\,\nabla_w\,Q_w(s,a).$$

But it turns out that this doesn't work in a satisfactory manner. In 2015 DeepMind[635] (now Google DeepMind) suggested a successful Q-learning variant known as deep Q-learning.

Deep Q-learning implements some special features, discussed below in more detail:

- more stable gradient descent,
- more efficient ANN structure,
- less correlated training samples,
- reuse of training samples.

51.4.1 Two ANNs

Approximate value function methods solve a supervised learning problem via gradient descent. Training samples consist of state-action pairs (inputs) and estimates for expected returns (outputs). During training one and the same state-action pair may appear with many different outputs, because estimates for expected return will improve over time. Thus, labels of the training samples for supervised learning are very noisy, which typically yields poor results in combniation with gradient descent. Convergence often is slow and some gradient steps destroy the progress of previous steps. Although this is a problem for SARSA, too, it seems to be much more severe for Q-learning.

The deep Q-learning algorithm tries to stabilize the minimization procedure by using two ANNs, a *prediction ANN* and a *target ANN*. Both ANNs represent estimates for expected return. Denote the value function represented by the prediction ANN by Q_w and the one represented by the target ANN by $Q_{w'}$.

Q_w is updated after each step as usual. $Q_{w'}$ is only updated every C steps by copying current weights w of the prediction ANN. Here C is a parameter of the overall algorithm. The update formula for Q_w is

$$w+\alpha\,(r+\gamma\max_{a'\in\mathcal{A}(s')}Q_{w'}(s',a')-Q_w(s,a))\,\nabla_w\,Q_w(s,a),$$

that is, $Q_{w'}$ is used for computing expected return only. In other words, $Q_{w'}$ determines the outputs of the training samples. Because $Q_{w'}$ changes only every C steps, training samples now are less noisy and gradient descent tends to behave more stably.

[635] https://en.wikipedia.org/wiki/Google_DeepMind

There's no proof that this procedure always works. But experience shows that performance of gradient descent greatly improves.

Note that in deep Q-learning the ANN structure is slightly different than for SARSA. Only the state is used as input. On the output side one has as many neurons as there are actions. The advantage of this structure is that for getting values for all actions we have to evaluate the ANN only once, saving computation time.

51.4.2 Experience Replay

Especially for reinforcement learning tasks with continuous state space (autonomous driving, for instance) training samples of neighboring time steps look very similar. But for supervised learning we want to have very different, uncorrelated samples. Else learning progress will be slow.

One solution to this problem is to train several agents in parallel, each in its own environment, but using a common Q function. Then observations obtained from the agents will be quite different.

Deep Q-learning follows another approach, known as *experience replay*. Training samples are kept in memory and are reused many times. On the one hand, this solves the correlation problem to some extent, because older samples will differ from newer samples. On the other hand, reusing training samples increases information gain from explored data. Without experience replay each sample would be used for one gradient step only. With experience replay there will be many gradient steps per sample. In each gradient step we may now provide a batch of samples to the ANN, which significantly increases convergence speed with only little additional computational effort.

Training samples are kept in a fixed-size *replay buffer*. For each time step we have to store the tuple (s,a,r,s'). If the replay buffer is full, the oldest training sample is removed. For each gradient step we take a random batch of samples from the replay buffer.

CHAPTER

FIFTYTWO

POLICY GRADIENT METHODS

Related projects:

- *Cart Pole* (page 1011)
  - *Policy Gradient Method* (page 1013)

All methods considered so far for reinforcement learning were based on value functions. They tried to find good estimates for value functions and then used the (ε-)greedy policy with respect to a value function. No we have an introductory look at methods optimizing policies more directly.

Going on from tabular methods to approximate value function methods like deep Q-learning we lost the important policy improvement theorem, which is the theoretical foundation for tabular methods. Thus, for approximate methods we cannot be sure that they really work in all cases. For policy gradient methods theoretical justification will be slightly better. Apart from this advantage, practice shows that for some tasks value function methods yield better results and for other tasks policy gradient methods work better. There's no general rule which class to use.

52.1 The Idea

Remember that a policy π is a function mapping a state-action pair (s, a) to the probability that the agent chooses action a in state s. This function can be represented as a table with $|\mathcal{S}| \times |\mathcal{A}|$ cells whoes values are to be chosen to maximize some quality measure for policies. In case of large state or action spaces we may replace the tabular approach by some kind of parametrized policies. Here we skip the less relevant tabular case and focus an ANN based policies.

Let π_θ be a policy represented by an ANN (or some other parametrization) with weight vector θ. Optimizing π_θ to solve a given task means choosing good weights θ. By J we denote a quality measure mapping policies to real numbers. Good quality measures are all kinds of expected returns. For episodic tasks J is the mean total reward with respect to all possible traces:

$$J(\pi_\theta) = \sum_{\text{traces}} \text{probability of trace} \times \text{total reward of trace}.$$

For continuing tasks we use a policy's average reward $r(\pi_\theta)$ (cf. *Policy Improvement* (page 832)):

$$J(\pi_\theta) = \sum_{s \in \mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a \in \mathcal{A}(s)} \pi_\theta(a, s) \sum_{\substack{s' \in \mathcal{S} \\ r \in \mathcal{R}}} p(s', r, s, a)\, r.$$

The common idea of all kinds of policy gradient methods now is to maximize $J(\pi_\theta)$ with respect to θ via gradient ascent

$$\theta_{k+1} := \theta_k + \alpha\, \nabla_\theta J(\pi_\theta),$$

where $\alpha > 0$ is the step length. Thus, the major step in deriving concrete algorithms is the computation of the gradient $\nabla_\theta J(\pi_\theta)$.

52.2 Derivation of the Gradient

We derive the gradient $\nabla_\theta J(\pi_\theta)$ only for continuing tasks here. For episodic tasks result differs only by a factor related to average episode length. This factor is of little importance due to multiplication of gradients by step lengths.

For the derivation of the gradient we drop the subscript θ to keep formulas simple. So we write ∇ instead of ∇_θ, π instead of π_θ and so on. We cannot go the straight-forward way for deriving the gradient because this would introduce terms $\nabla\mu_\pi$ we cannot simplify further. Instead we use a little trick: we compute the gradient of some other quantity and then extract ∇J from corresponding formula. For the gradient of the differential return based state value function v_π and fixed arbitrary state s we have

$$\begin{aligned}\nabla v_\pi(s) &= \nabla\left(\sum_{a\in\mathcal{A}(s)}\pi(a,s)\,q_\pi(s,a)\right)\\ &= \sum_{a\in\mathcal{A}(s)}\left(\nabla\pi(a,s)\,q_\pi(s,a)+\pi(a,s)\,\nabla q_\pi(s,a)\right)\end{aligned}$$

by the product rule. With

$$\begin{aligned}\nabla q_\pi(s,a) &= \nabla\left(\sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}}p(s',r,s,a)\,(r-J(\pi)+v_\pi(s'))\right)\\ &= \sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}}p(s',r,s,a)\,(-\nabla J(\pi)+\nabla v_\pi(s'))\\ &= -\nabla J(\pi)+\sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}}p(s',r,s,a)\,\nabla v_\pi(s')\end{aligned}$$

we obtain

$$\begin{aligned}\nabla v_\pi(s) &= \sum_{a\in\mathcal{A}(s)}\left(\nabla\pi(a,s)\,q_\pi(s,a)+\pi(a,s)\left(-\nabla J(\pi)+\sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}}p(s',r,s,a)\,\nabla v_\pi(s')\right)\right)\\ &= -\nabla J(\pi)+\sum_{a\in\mathcal{A}(s)}\nabla\pi(a,s)\,q_\pi(s,a)+\sum_{a\in\mathcal{A}(s)}\pi(a,s)\sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}}p(s',r,s,a)\,\nabla v_\pi(s').\end{aligned}$$

Consequently,

$$\nabla J(\pi) = -\nabla v_\pi(s)+\sum_{a\in\mathcal{A}(s)}\nabla\pi(a,s)\,q_\pi(s,a)+\sum_{a\in\mathcal{A}(s)}\pi(a,s)\sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}}p(s',r,s,a)\,\nabla v_\pi(s').$$

for all s. Now we multiply this equation by $\mu_\pi(s)$ and sum over all s (remember that the sum of all $\mu_\pi(s)$ equals 1):

$$\begin{aligned}\nabla J(\pi) = &-\sum_{s\in\mathcal{S}}\mu_\pi(s)\,\nabla v_\pi(s)+\sum_{s\in\mathcal{S}}\mu_\pi(s)\sum_{a\in\mathcal{A}(s)}\nabla\pi(a,s)\,q_\pi(s,a)\\ &+\sum_{s\in\mathcal{S}}\mu_\pi(s)\sum_{a\in\mathcal{A}(s)}\pi(a,s)\sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}}p(s',r,s,a)\,\nabla v_\pi(s').\end{aligned}$$

Together with

$$\begin{aligned}&\sum_{s\in\mathcal{S}}\mu_\pi(s)\sum_{a\in\mathcal{A}(s)}\pi(a,s)\sum_{\substack{s'\in\mathcal{S}\\ r\in\mathcal{R}}}p(s',r,s,a)\,\nabla v_\pi(s')\\ &\quad=\sum_{s'\in\mathcal{S}}\nabla v_\pi(s')\sum_{s\in\mathcal{S}}\mu_\pi(s)\sum_{\substack{a\in\mathcal{A}(s)\\ r\in\mathcal{R}}}\pi(a,s)\,p(s',r,s,a)\\ &\quad=\sum_{s'\in\mathcal{S}}\nabla v_\pi(s')\,\mu_\pi(s')\end{aligned}$$

we arrive at

$$\nabla_\theta J(\pi_\theta) = \sum_{s\in\mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a\in\mathcal{A}(s)} \nabla_\theta \pi_\theta(a,s)\, q_{\pi_\theta}(s,a).$$

Here we see that the computation of $\nabla_\theta J$ reduces to the computation of the policy's gradient. Thus the name 'policy gradient methods'.

The other two quantities involved are the distribution μ_{π_θ} of visits to states, which will be automatically determined when computing estimates of $\nabla_\theta J$ from an agent's observations, and the action value function q_{π_θ}, which will be estimated from observations, too.

52.3 REINFORCE

REINFORCE uses the idea of *Monte Carlo Methods* (page 819) (sample empirical return many times) to estimate the gradient $\nabla_\theta J(\pi_\theta)$ and is, thus, restricted to episodic tasks.

Assuming $\pi_\theta(a,s) > 0$ for all a and all s. We may rewrite $\nabla_\theta J(\pi_\theta)$ as follows:

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \sum_{s\in\mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a\in\mathcal{A}(s)} q_{\pi_\theta}(s,a)\, \nabla_\theta \pi_\theta(a,s)\\ &= \sum_{s\in\mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a\in\mathcal{A}(s)} \pi_\theta(a,s)\, q_{\pi_\theta}(s,a)\, \frac{\nabla_\theta \pi_\theta(a,s)}{\pi_\theta(a,s)}\\ &= \sum_{s\in\mathcal{S}} \mu_{\pi_\theta}(s) \sum_{a\in\mathcal{A}(s)} \pi_\theta(a,s)\, q_{\pi_\theta}(s,a)\, \nabla_\theta \ln \pi_\theta(a,s).\end{aligned}$$

This is the expected value of $q_{\pi_\theta}(s,a)\,\nabla_\theta \ln \pi_\theta(a,s)$ with respect to all state-action pairs (s,a) visited under π_θ. So estimating $\nabla_\theta J(\pi_\theta)$ reduces to observing some traces by following π_θ, computing an estimate for q_{π_θ} (average observed return for (s,a)), and computing an estimate for $\nabla_\theta \ln \pi_\theta(a,s)$ (average over all visits to (s,a)).

Here is the full algorithm:

1. Choose step length $\alpha > 0$.
2. Choose initial weights θ (at random, for instance).
3. For each episode (following π_θ) with trace $s_0, a_0, r_1, s_1, \ldots, r_T, s_T$ do:
    1. For each $t = 0, 1, \ldots, T-1$ do:
        1. Update θ to $\theta + \alpha \left(\sum_{k=t+1}^{T} r_k\right) \nabla_\theta \ln \pi_\theta(a_t, s_t)$,

Here we compute the gradient after each step, that is, 'estimating' reduces to one sample only.

Note that the assumption $\pi_\theta(a,s) > 0$ is always satisfied for visited (s,a), else the pair would not have been visited.

52.4 Actor-Critic Methods

Actor-critic methods use the idea of *Temporal Difference Learning (TD Learning)* (page 825) (look one step ahead for value function estimates) to estimate the gradient $\nabla_\theta J(\pi_\theta)$ and are, thus, suitable for episodic and continuing tasks. Here we restrict our attention to a SARSA-like actor-critic method for continuing tasks with differential return.

Next to π_θ also a parametrized state value function estimate V_w with weights w will be computed. The policy is the *actor* choosing actions. The value function is the *critic* telling us whether the policy does a good job and how much (but not in which way) the policy should get improved. In contrast to REINFORCE q_{π_θ} is not estimated from one trace, but from all observations collected so far. The full algorithm is as follows:

1. Initialization
    1. Choose $\alpha_1 > 0$ (step length for updating V_w), $\alpha_2 > 0$ (step length for updating π_θ), $\beta \in (0,1]$ (step length for updating average reward.

   2. Choose initial weights w and θ (at random, for instance).
   3. Choose initial average reward $\overline{r}$ (zero, for instance).
   4. Choose initial state s.
2. Step
   1. Take action a according to π_θ.
   2. Observe reward r and next state s'.
3. Update
   1. Replace $\overline{r}$ by $\overline{r}+\beta\,(r-\overline{r})$.
   2. Replace w by $w+\alpha_1\,(r-\overline{r}+V_w(s')-V_w(s))\nabla_w V_w(s)$.
   3. Replace θ by $\theta+\alpha_2\,(r-\overline{r}+V_w(s'))\nabla_\theta \ln\pi_\theta(a,s)$.
   4. Replace s by s'.
   5. Go to 2.1.

Learning progress of actor-critic methods tends to be more stable, because estimates for q_{π_θ} are more stable than with REINFORCE. But computational costs are higher due to an additional ANN for representing the value function estimate V_w.

52.5 Baselines

When discussing REINFORCE, we derived the formula

$$\nabla_\theta J(\pi_\theta)=\sum_{s\in\mathcal{S}}\mu_{\pi_\theta}(s)\sum_{a\in\mathcal{A}(s)}\pi_\theta(a,s)\,q_{\pi_\theta}(s,a)\,\nabla_\theta\ln\pi_\theta(a,s)$$

for the gradient of the quality measure to be maximized. Here we see that the length of $\nabla_\theta J(\pi_\theta)$ and, thus, the behavior of gradient ascent depends on the values $q_{\pi_\theta}(s,a)$. The range of those values is determined by the range of rewards, which can be freely chosen. Stability and convergence of gradient ascent heavily depends on how we model the environment. To overcome potential problems resulting from this degree of freedom we introduce the concept of baselines.

A baseline is any function b mapping states to real numbers. A baseline is ot allows to depend on chosen actions in any way. For a baseline b we have

$$\begin{aligned}\sum_{s\in\mathcal{S}}\mu_{\pi_\theta}(s)\sum_{a\in\mathcal{A}(s)}\pi_\theta(a,s)\,b(s)\,\nabla_\theta\ln\pi_\theta(a,s)&=\sum_{s\in\mathcal{S}}\mu_{\pi_\theta}(s)\,b(s)\sum_{a\in\mathcal{A}(s)}\pi_\theta(a,s)\,\nabla_\theta\ln\pi_\theta(a,s)\\&=\sum_{s\in\mathcal{S}}\mu_{\pi_\theta}(s)\,b(s)\sum_{a\in\mathcal{A}(s)}\nabla_\theta\pi_\theta(a,s)\\&=\sum_{s\in\mathcal{S}}\mu_{\pi_\theta}(s)\,b(s)\,\nabla_\theta\left(\sum_{a\in\mathcal{A}(s)}\pi_\theta(a,s)\right)\\&=\sum_{s\in\mathcal{S}}\mu_{\pi_\theta}(s)\,b(s)\,\nabla_\theta 1\\&=\sum_{s\in\mathcal{S}}\mu_{\pi_\theta}(s)\,b(s)\cdot 0\\&=0.\end{aligned}$$

Thus, we may introduce a baseline in $\nabla_\theta J(\pi_\theta)$ without changing the gradient:

$$\nabla_\theta J(\pi_\theta)=\sum_{s\in\mathcal{S}}\mu_{\pi_\theta}(s)\sum_{a\in\mathcal{A}(s)}\pi_\theta(a,s)\,(q_{\pi_\theta}(s,a)-b(s))\,\nabla_\theta\ln\pi_\theta(a,s).$$

A common choice is

$$b(s)=v_{\pi_\theta}(s),$$

because then in REINFORCE and in the SARSA-based actor-critic method above the estimate for $\nabla_\theta J(\pi_\theta)$ depends on the difference between new and old value estimate instead of on the value estimate itself.

To use this baseline with REINFORCE an additional state value function estimate has to be introduced in the algorithm. For the actor-critic method the baseline fits for naturally. The only change is the update of weights θ, which becomes

$$\theta + \alpha_2 \left(r - \bar{r} + V_w(s') - V_w(s)\right) \nabla_\theta \ln \pi_\theta(a, s),$$

showing even stronger parallels to the update of w.

Part IX

Exercises

CHAPTER

FIFTYTHREE

COMPUTER BASICS

To solve these exercises on bits and bytes and representations of numbers you should have read *Computers and Programming* (page 43).

53.1 Bits and Bytes

53.1.1 Hard Disk Capacity

A hard disk is advertised to have a capacity of 2 TB. Give the hard disk's capacity in TiB.

Solution

```
# your solution
```

53.1.2 Uncompressed Image File

An image has a width of 4000 pixels and a height of 3000 pixels. For each pixel three color components (red, green, blue) have to be saved, where 8 bits per component are required. How much space does the image need when saving it to a file without compression?

Solution

```
# your solution
```

53.1.3 Books

A book with 500 pages and 1500 characters per page in average shall be saved to a file. Each character requires one byte of disk space. What is the file's total size?

A well known online encyclopedia has about 50 GB of English text (without images). How many books are needed for a print version?

How long does it take to transfer 50 GB of data with a transfer rate of 100 megabit per second?

Solution

```
# your solution
```

53.1.4 Uncompressed Video File

How much storage is required for an uncompressed 120 minutes video file with 30 frames (that is, images) per second and full HD resolution (1920x1080 pixels, 24 bits per pixel).

What's the compression rate if the video file has size 4.1 GB?

Solution

```
# your solution
```

53.2 Representation of Numbers

53.2.1 Binary to Decimal

Write the binary numbers as decimal numbers.

- 10000001
- 11010010

Solution

```
# your solution
```

53.2.2 Decimal to Binary

Write the decimal numbers as binary numbers:

- 15
- 16
- 17
- 123

Solution

```
# your solution
```

53.2.3 Decimal to Hexadecimal

Write the decimal numbers as hexadecimal numbers:

- 31
- 32
- 33
- 234
- 257

Solution

```
# your solution
```

53.2.4 Binary to Hexadecimal

Write the 32-digit binary number 10001001 11111110 00001010 01001101 as hexadecimal number.

Solution

```
# your solution
```

53.3 Memory vs. Storage

Give two differences between a computer's memory and a mass storage device.

Solution

```
# your solution
```

53.4 Compilers and Interpreters

What's the difference between compiled and interpreted computer programs?

Solution

```
# your solution
```

CHAPTER

FIFTYFOUR

PYTHON PROGRAMMING

54.1 Finding Errors

Computer programs contain lots of errors. Finding them is much more difficult than correcting them. In this series of exercises you have to find syntax and semantic errors in small programs. Before you start you should have read *Building Blocks* (page 56).

Hint: Simply run the programs and let the Python interpreter look for errors. Then correct all errors identified by the interpreter. If still something is not working as expected, have a closer look at the source code.

54.1.1 Simple 1

```
print('Python is a programming language.')
print('Many people love Python's approach to programming.')
print('Maybe it's the first programming language you learn to use.')
```

Solution:

```
# your modifications

print('Python is a programming language.')
print('Many people love Python's approach to programming.')
print('Maybe it's the first programming language you learn to use.')
```

54.1.2 Simple 2

```
a = 2
b = 3

if a < b
     print(a, '<', b)
```

Solution:

```
# your modifications

a = 2
b = 3

if a < b
     print(a, '<', b)
```

54.1.3 Simple 3

```
def say_something(text):

    print('I have to say:')
    print(text)

say_samething('Python is great!')
```

Solution:

```
# your modifications

def say_something(text):

    print('I have to say:')
    print(text)

say_samething('Python is great!')
```

54.1.4 Simple 4

```
a = 2
b = 3

if a = b:
    print(a, 'equals', b)
else:
    print('not equal')
```

Solution:

```
# your modifications

a = 2
b = 3

```

(continues on next page)

(continued from previous page)

```
if a = b:
    print(a, 'equals', b)
else:
    print('not equal')
```

54.1.5 Slightly More Difficult 1

```
primes = [2, 3, 5, 7, 11, 13]

print('third prime number is', primes(2))
```

Solution:

```
# your modifications

primes = [2, 3, 5, 7, 11, 13]

print('third prime number is', primes(2))
```

54.1.6 Slightly More Difficult 2

```
problem = '5 + 4'
solution = '5' + '4'

print(problem, '=', solution)
```

Solution:

```
# your modifications

problem = '5 + 4'
solution = '5' + '4'

print(problem, '=', solution)
```

54.1.7 Slightly More Difficult 3

```
my_list = [1, 3, 4, 2, 6]

print('last item of my list is', my_list[5])
```

Solution:

```
# your modifications

my_list = [1, 3, 4, 2, 6]

print('last item of my list is', my_list[5])
```

54.1.8 Slightly More Difficult 4

```
last = 5

print('printing all square numbers up to square of', last)
for k in range(1, last):
    print(k + k)
```

Solution:

```
# your modifications

last = 5

print('printing all square numbers up to square of', last)
for k in range(1, last):
    print(k + k)
```

54.1.9 More Difficult 1

```
print('Adding zero to 5 doesn't change anything:')
print(5 + O)
```

Solution:

```
# your modifications

print('Adding zero to 5 doesn't change anything:')
print(5 + O)
```

54.1.10 More Difficult 2

```
l = [2, 3, 2, 5, 7, 8, 9]

for i in range(0, len(l)):
    print('product of item', i, 'and item', i + 1, 'is', l[i] * l[i + 1])
```

Solution:

```
# your modifications

l = [2, 3, 2, 5, 7, 8, 9]

for i in range(0, len(l)):
    print('product of item', i, 'and item', i + 1, 'is', l[i] * l[i + 1])
```

54.1.11 More Difficult 3

```
l = [2, 4, 6, 3]

print('let\'s print the list', l, 'item by item:')
for i in range(1, len(l)):
    print(l[i - 1])
```

Solution:

```
# your modifications

l = [2, 4, 6, 3]

print('let\'s print the list', l, 'item by item:')
for i in range(1, len(l)):
    print(l[i - 1])
```

54.1.12 Difficult 1

```
my_lists = [[1, 2, 3], [4, 2, 5, 6, 7], [1, 4], [0]]

total_sum = 0
for i in range(0, len(my_lists)):
    for j in range(0, len(my_lists[0])):
        total_sum = total_sum + my_lists[j][i]

print(total_sum)
```

Solution:

```
# your modifications

my_lists = [[1, 2, 3], [4, 2, 5, 6, 7], [1, 4], [0]]

total_sum = 0
for i in range(0, len(my_lists)):
    for j in range(0, len(my_lists[0])):
        total_sum = total_sum + my_lists[j][i]

print(total_sum)
```

54.1.13 Difficult 2

```
last = 19

print('detecting prime numbers below or equal to', last)
for n in range(2, last):
    for k in range(2, n / 2):
        if n % k != 0:
            break
    else:
        print(n)
```

Solution:

```
# your modifications

last = 19

print('detecting prime numbers below or equal to', last)
for n in range(2, last):
    for k in range(2, n / 2):
        if n % k != 0:
            break
    else:
        print(n)
```

54.1.14 Difficult 3

```
def print_first_half(l):
    '''print first half of list, include center item if length of list is odd'''

    print('first half of', l, 'is')
    for i in range(0, len(l) / 2):
        print(l[i])


def print_second_half(l):
    '''print second half of list, omit center item if length of list is odd'''

    print('second half of', l, 'is')
    for i in range(len(l) / 2, len(l)):
        print(l[i])


l = [2, 4, 3, 6, 8, 5, 7]
print_first_half(l)
print_second_half(l)

l = [2, 4, 3, 6, 8, 5]
print_first_half(l)
print_second_half(l)

l = [1]
print_first_half(l)
print_second_half(l)

l = []
print_first_half(l)
print_second_half(l)
```

Solution:

```
# your modifications

def print_first_half(l):
    '''print first half of list, include center item if length of list is odd'''

    print('first half of', l, 'is')
    for i in range(0, len(l) / 2):
        print(l[i])


def print_second_half(l):
```

(continues on next page)

(continued from previous page)

```
    '''print second half of list, omit center item if length of list is odd'''

    print('second half of', l, 'is')
    for i in range(len(l) / 2, len(l)):
        print(l[i])


l = [2, 4, 3, 6, 8, 5, 7]
print_first_half(l)
print_second_half(l)

l = [2, 4, 3, 6, 8, 5]
print_first_half(l)
print_second_half(l)

l = [1]
print_first_half(l)
print_second_half(l)

l = []
print_first_half(l)
print_second_half(l)
```

54.2 Basics

Before solving these programming exercises you should have read *Building Blocks* (page 56). Only use Python features discussed there.

54.2.1 Python as a Calculator 1

If light travels 300 million meters per second, how many kilometers travels light in one hour?

Solution:

```
# your solution
```

54.2.2 Python as a Calculator 2

A 20 years old person started watching web videos at the age of 8. Everyday the person watches videos for 2 hours. If the person would have watched videos the same total number of hours, but all the day instead of only 2 hours, how many years of its young life would the person have wasted?

Take into account, that the person uses approximately 7 hours for sleeping and 3 ours for eating, grooming, and doing housework. Thus, time for watching videos is less than 24 hours a day.

Assume that each year has 365 days.

Solution:

```
# your solution
```

54.2.3 Integer Division

Get two integers from the user. Divide the first by the second. Use floor division and show the remainder, too. Avoid `ZeroDivisionError`.

Solution:

```
# your solution
```

54.2.4 Conditional Execution 1

Get three integers from the user and print a message if they are *not* in ascending order.

Solution:

```
# your solution
```

54.2.5 Conditional Execution 2

Get three integers from the user und print them in ascending order.

Solution:

```
# your solution
```

54.2.6 Functions 1

Write a function `is_ascending` which checks whether its three numeric arguments are in ascending order. Return a boolean value.

Test the function with an ascending sequence and a non-ascending sequence.

Solution:

```
# your solution
```

54.2.7 Functions 2

Write a function `cut_off_decimals` which takes two arguments, a float and a positive integer. The function shall cut off all but the specified number of decimal places of the float and then return the float. If the second argument is negative, the float shall remain untouched.

Note, that floor devision also works for floats. Thus, `x // 1` cuts off all decimal places.

Test the function with 1.2345 and 2 decimal places.

Solution:

```
# your solution
```

54.2.8 Loops 1

Print the numbers 0 to 9 with a while loop. In other words: Use a while loop to simulate a for loop.

Solution:

```
# your solution
```

54.2.9 Loops 2

Take a list and print its items in reverse order. Test your code with `[-1, 2, -3, 4, -5]`.

Solution:

```
# your solution
```

54.2.10 Loops 3

Take a list and print it item by item. After each item ask the user whether he wants to see the next item. If not, stop printing. If yes, go on.

Solution:

```
# your solution
```

54.2.11 Loops 4

Take a list and print it item by item. After each item ask the user whether he wants to see the next item. If not, stop printing. If yes, go on. If there are no more items left, start again with the first item.

Solution:

```
# your solution
```

54.3 More Basics

Solving this set of exercises increases your skills in algorithmic thinking and Python's syntax. Everything you need has been discussed in the *Crash Course* (page 53) chapter. Do not use additional Python features or modules.

54.3.1 Point Inside Rectangle?

Get two integers from the user and check whether the corresponding point lies inside the rectangle with corners at (-1, -1), (5, -1), (5, 2), (-1, 2). Print a message showing the result.

Solution:

```
# your solution
```

54.3.2 Square Numbers

Get an integer from the user and tell the user whether it's a square number or not. If you want to compute square roots, use `123 ** 0.5`. Print a message if the user gave a negative number.

Hint: Have a look at the output of `16.125 % 1`.

Solution:

```
# your solution
```

54.3.3 Unique Items

Write a function `no_duplicates` which returns `True` if the passed list contains no duplicates and `False` if there are duplicates.

Test your function with `[1, 4, 5, 6, 3]` and `[1, 3, 1]` (and with `[]`, of course).

Solution:

```
# your solution
```

54.3.4 Increasing Subsequence

Write a function `inc_subseq` which takes a list of numbers and prints all items except the ones which are smaller than their predecessor.

Test your function (at least) with `[1, 3, 2, 3, 4, -2, 9]` and `[3, 2, 1]`.

Solution:

```
# your solution
```

54.3.5 Area of a Circle

Get an integer radius of a circle from the user. Calculate and print the circle's area as well as the edge length of a square with identical area. Check user input for validity. You may use NumPy's `pi` constant.

Solution:

```
# your solution
```

54.3.6 Quadratic Equations

Solve the quadratic equation $a\,x^2 + b\,x + c = 0$ with user-specified a, b, c (integers). Give all real solutions.

Solution:

```
# your solution
```

54.3.7 Regular Polygons

Use Matplotlib and NumPy's `sin` and `cos` functions to plot a regular polygon. Ask the user for the number of vertices and check user input for validity.

Hint: For n vertices the kth vertex is at $(\cos\varphi, \sin\varphi)$ with $\varphi = 2\,\pi\,\frac{k}{n}$.

Solution:

```
# your solution
```

54.3.8 Stars

Draw a star with user-specified number of outside vertices. Radius for inner vertices is 0.3, for outer vertices it's 1.

Solution:

```
# your solution
```

54.4 Variables and Operators

Python's approach to variables and operators is simple and beautiful, although beginners need some time to see both simplicity and beauty. In each exercise below keep track of what's happening in detail behind the scenes. That is, track the creation of objects and which name is tied to which object. Before solving the exercises read the chapter on *Variables and Operators* (page 85) (sections *Operators as Member Functions* (page 98) and *Efficiency* (page 99) are not required here).

54.4.1 Global vs. Local Variables 1

The following code contains an error. Find it (by running the code) and correct it.

```
def do_something():
    n = n + 1
    print('something')

n = 0     # counter for function calls

for i in range(0, 100):
    if i % 11 == 0:
        do_something()

print('function called', n, 'times')
```

Solution:

```
# your modifications

def do_something():
    n = n + 1
    print('something')

n = 0     # counter for function calls

for i in range(0, 100):
    if i % 11 == 0:
```

(continues on next page)

(continued from previous page)

```
        do_something()

print('function called', n, 'times')
```

54.4.2 Global vs. Local Variables 2

The following code shall print 10 lines each containing 5 numbers. Make it work correctly.

```
def print_n_times_5_numbers(m, n, o, p, q):
    for k in range(0, n):
        print(m, n, o, p, q)

n = 10    # number of rows to print

print('printing', n, 'times 5 numbers:')
print_n_times_5_numbers(42, 23, 32, 24, 111)
```

Solution:

```
# your modifications

def print_n_times_5_numbers(m, n, o, p, q):
    for k in range(0, n):
        print(m, n, o, p, q)

n = 10    # number of rows to print

print('printing', n, 'times 5 numbers:')
print_n_times_5_numbers(42, 23, 32, 24, 111)
```

```
printing 10 times 5 numbers:
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
42 23 32 24 111
```

54.4.3 List of Squares

Make the following code work correctly.

```
a = [1, 2, 3, 4, 5]

squares = a
for i in range(0, len(squares)):
    squares[i] **= 2

print('squares of', a, 'are:')
print(squares)
```

Solution:

```
# your modifications

a = [1, 2, 3, 4, 5]

squares = a
for i in range(0, len(squares)):
    squares[i] **= 2

print('squares of', a, 'are:')
print(squares)
```

```
squares of [1, 4, 9, 16, 25] are:
[1, 4, 9, 16, 25]
```

54.4.4 Similar Code, Different Results

Why do the following two code cells yield different results? Explain in detail what's happening!

```
a = 2
b = a
a = 5
print(b)
```

```
2
```

```
a = [2]
b = a
a[0] = 5
print(b[0])
```

```
5
```

Solution:

```
# your answer
```

54.4.5 2 > 3?

Guess why the condition `2 <= 3 == True` evaluates to `False`. How to repair?

```
if 2 <= 3 == True:
    print('2 <= 3')
else:
    print('2 > 3, really?')
```

Solution:

```
# your modifications

if 2 <= 3 == True:
    print('2 <= 3')
else:
    print('2 > 3, really?')
```

```
2 > 3, really?
```

54.4.6 Minus Minus

Why do the following two code cells yield different results?

```
a = 5
b = 2
c = 3
print(a - b - c)
```

```
0
```

```
a = 5
b = 2
c = 3
a -= b - c
print(a)
```

```
6
```

Solution:

```
# your answer
```

54.5 Memory Management

Here you find some exercises on Python's optimization techniques for memory management discussed in *Efficiency* (page 99).

The last two tasks demonstrate effects of running out of memory and how to prevent such situations. Read *Garbage Collection* (page 101) first.

54.5.1 Similar Code, Different Results 1

Why do the following two code cells yield different results?

```
a = 100
b = 2 * a
c = 2 * a
print(b is c)
```

```
True
```

```
a = 100
b = 3 * a
c = 3 * a
print(b is c)
```

```
False
```

Solution:

```
# your answer
```

54.5.2 Similar Code, Different Results 2

Copy the following code to a text file and feed the file to the Python interpreter. Why do both variants yield different results? Why running in Jupyter yields identical results in both cases?

```
# variant 1
a = 1234
b = 1234
print(a is b)

# variant 2
c = 34
a = 1200 + c
b = 1200 + c
print(a is b)
```

```
False
False
```

Solution:

```
# your answer
```

54.5.3 Some More Code Optimization

Copy the following code to a text file and feed the file to the Python interpreter. Do you have an idea why the next code yields `True`?

```
a = 1200 + 34
b = 1200 + 34
print(a is b)
```

```
False
```

Solution:

```
# your answer
```

54.5.4 Running Out of Memory

Run the following code and observe what happens to your system. Use some system tool to monitor memory consumption. You may interrupt the Python kernel in Jupyter if necessary.

Warning: Save all your data you are currently editing. Depending on your system you may have to reboot your machine due to hanging the system.

```
stop = False
data = 'x'

while not stop:

    print('Hit Return to stop. Type an int to multiply memory consumption with.')
    fac = input()

    if fac == '':
        stop = True
    else:
        fac = int(fac)
        print('increasing memory usage to', fac * len(data), 'bytes...')
        new_data = ''
        for i in range(0, fac):
            new_data += data
        data = new_data
        del new_data
        print('...done')

del data
```

Note the `del data` line. This line is not required if the program is run as a stand-alone program (text file fed to the interpreter), because at exit the interpreter will free all memory used by the program. But in Jupyter the interpreter does not stop at the end of a code cell. All objects created by the cell remain in memory until the kernel dies or memory is explicitly freed via `del`.

Solution:

```
# your answer
```

54.5.5 Managing Memory

Write a function which returns a string with approximately 1 billion characters. Then write a program which can manage three data sets. Repeatedly ask the user whether he or she wants the exit the program or whether he or she wants to load/remove data set 1/2/3. Load and remove data sets according to the user's choice. Monitor memory consumption while running the program to see whether the program works as intended.

Solution:

```
# your solution
```

54.6 Lists and Friends

To solve these exercises you yould have read *Lists and Friends* (page 103). Only use features discussed there or in previous chapters.

54.6.1 Bad Coding Style

Consider the following code snipped. What numbers appear on screen when printing `a`, `b`, `e`, `g`, and `h` after executing the code? Don't run the code, interpret each line manually.

```
a = 1
b = 2
c = [a, b]
c[0] = 3
d = c
d[1] = 4
c.append(5)
e = c[-1]
f = c[0:-1]
g = f[-1]
h = d[0]
```

Solution:

```
# your answer
```

54.6.2 Squares and Sums of List of Lists

The following code yields incorrect outputs. Find the problem and solve it.

```
a = [1, 2, 3]
b = [4, 5, 6]
c = [7, 8, 9]
d = [a, b, c]

# compute squares
for i in range(0, 3):
    for j in range(0, 3):
        d[i][j] **= 2

# compute row sums
sum_a = a[0] + a[1] + a[2]
sum_b = b[0] + b[1] + b[2]
sum_c = c[0] + c[1] + c[2]
```

(continues on next page)

(continued from previous page)

```
# print results
print('squares of d:', d)
print('sum of a:', sum_a)
print('sum of b:', sum_b)
print('sum of c:', sum_c)
```

Solution:

```
# your modifications

a = [1, 2, 3]
b = [4, 5, 6]
c = [7, 8, 9]
d = [a, b, c]

# compute squares
for i in range(0, 3):
    for j in range(0, 3):
        d[i][j] **= 2

# compute row sums
sum_a = a[0] + a[1] + a[2]
sum_b = b[0] + b[1] + b[2]
sum_c = c[0] + c[1] + c[2]

# print results
print('squares of d:', d)
print('sum of a:', sum_a)
print('sum of b:', sum_b)
print('sum of c:', sum_c)
```

```
squares of d: [[1, 4, 9], [16, 25, 36], [49, 64, 81]]
sum of a: 14
sum of b: 77
sum of c: 194
```

54.6.3 Slicing Instead of Loops 1

Write a function `shift` which takes a list, increases each item's index by one (last item becomes first one), and returns the resulting list. Example: `[1, 3, 5, 7]`should become `[7, 1, 3, 5]`. Don't use loops, but slicing syntax.

Solution:

```
# your solution
```

54.6.4 Slicing Instead of Loops 2

Write a function `shift_n` which takes a list and a positive integer `n` and shifts the list `n` times (cf. previous task). Don't use loops. Do not forget to think about the case that `n` is larger than the length of the list. Examples:

- `shift_n([1, 2, 3, 4, 5], 3)` should be `[3, 4, 5, 1, 2]`.
- `shift_n([1, 2, 3, 4, 5], 5)` should be `[1, 2, 3, 4, 5]`.
- `shift_n([1, 2, 3, 4, 5], 6)` should be `[5, 1, 2, 3, 4]`.

Solution:

```
# your solution
```

54.6.5 Dictionary from Lists 1

Given two lists `keys` and `values` create a dictionary. Use a for loop to fill an empty dictionary item by item. Test case:

```
keys = ['a', 'b', 'c', 'd']
values = [1, 2, 3, 4]
```

Solution:

```
# your solution
```

54.6.6 Dictionary from Lists 2

Given two lists `keys` and `values` create a dictionary. Use a dictionary comprehension. Test case:

```
keys = ['a', 'b', 'c', 'd']
values = [1, 2, 3, 4]
```

Solution:

```
# your solution
```

54.6.7 Dictionary from Lists 3

Given two lists `keys` and `values` create a dictionary. Call `dict` and pass a list of key-value pairs. Test case:

```
keys = ['a', 'b', 'c', 'd']
values = [1, 2, 3, 4]
```

Solution:

```
# your solution
```

54.6.8 One-Liner 1

Given a list of numbers, write one line of code to create a new list containing only numbers greater than 3. Test case:

```
[4, 3, 2, 8, 6, 0, 4, 6, -2, 1]
```

Solution:

```
# your solution
```

54.6.9 One-Liner 2

In one line of code create a new list containing every second number from a given list, if the number is between 1 and 10 (both included). Test case:

```
[-2, 3, 2, 6, 23, 1, 42, 42, 5, 10, 1, 12, 6, 4, 3]
```

Solution:

```
# your solution
```

54.6.10 One-Liner 3

Write one line of code to square all numbers in a list of lists of numbers. Test case:

```
[[1, 2, 3], [7, 6, 5, 4], [8, 9]]
```

Solution:

```
# your solution
```

54.6.11 Manual Deep Copying

Make a copy of a list of lists of numbers. In the end, changing a number in the copy must not modify the original numbers. Test case:

```
[[1, 2, 3], [7, 6, 5, 4], [8, 9]]
```

Solution:

```
# your solution
```

54.7 Strings

Read *Strings* (page 115) before you start with the exercises.

54.7.1 Character Statistics

Print a list of all characters appearing in a string. Count how often each character appears **without** using `str.count`. Get the string from user input.

Solution:

```
# your solution
```

54.7.2 Unicode Fruits

Print the string `I like apples and melons.` where `apples` and `melons` shall be replaced by a suitable Unicode symbol.

Solution:

```
# your solution
```

54.7.3 Parser

Write a function which converts a string representation of a table of integers to a list of lists of integers. The elements of a row are separated by commas and rows are separated by semicolons. Test your function with

```
'1,2,3;4,5,6;7,8,9'
```

Result should be

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Bonus: Solve the task in one line.

Solution:

```
# your solution
```

54.7.4 Spoon Language

Write a Python script which prompts the user for some input and translate user input to spoon language[636].

Hint: Simple replacement does not work. If, for instance, at first *a* is replaced by *alewa*, subsequent replacement of *e* will alter the *lew* in *alewa*. There seems to be no replacement order for vowels working correctly in each case. Thus, in a first run replace all vowels by some code (unlikely to appear in a text), then, in a second run, replace the codes by the vowel's lew-version.

Solution:

```
# your solution
```

636 https://de.wikipedia.org/wiki/Spielsprache#L%C3%B6ffelsprache

54.8 File Access

Read *Accessing Data* (page 121) before you start with the exercises.

54.8.1 Lower Case Copy

Read some text file's content, convert it to lower case and save it to a new text file.

Solution:

```
# your solution
```

54.8.2 Reading CSV Files

Get a CSV file containing all public trees at Chemnitz from the Open data portal of Chemnitz[637]. Read the first 10 lines from the file and show them on screen.

Hint: If you encounter cumbersome symbols in the output, have a look at byte order marks[638] at Wikipedia.

Solution:

```
# your solution
```

54.8.3 Reading ZIP files

Get 'The Blog Authorship Corpus' from the web. The original source https://u.cs.biu.ac.il/~koppel/BlogCorpus.htm vanished in 2022. Use https://www.fh-zwickau.de/~jef19jdw/datasets/blogs.zip. The ZIP file contains an `info.txt` file and the original ZIP file.

Read `infos.txt` to get information about the file name format used in the ZIP file.

Write a Python program which extracts all 5 features from the file names and saves them in a CSV file.

Solution:

```
# your solution
```

54.8.4 Reading XML Files

Open the file `7596.male.26.Internet.Scorpio.xml` from 'The Blog Authorship Corpus' (see exercise above) without extracting it explicitly. Print the first and the last post in the file to screen.

Solution:

```
# your solution
```

[637] http://portal-chemnitz.opendata.arcgis.com
[638] https://en.wikipedia.org/wiki/Byte_order_mark

54.9 Functions

Read *Functions* (page 137) before you start with the exercises.

54.9.1 Inplace Squaring

Write a function which squares all numbers in a list. The list of numbers is the only parameter and there is no return value. The results have to be stored in the original list.

Test your code with

```
[1, 2, 3, 4, 5]
```

Solution:

```
# your solution
```

54.9.2 Arbitrary Keyword Arguments

Write a function taking an arbitrary number of keyword arguments and printing all of them.

Test your code by passing

```
a=1, b='test', c=(1, 2, 3)
```

This should yield the output

```
a: 1
b: test
c: (1, 2, 3)
```

Solution:

```
# your solution
```

54.9.3 Apply to All

Write a function `apply_to_all` which takes an arbitrary number of arguments.

- The first argument is a function taking a float and returning a float.
- All other arguments are floats.

The `apply_to_all` function shall apply the first argument (function) to all other arguments (floats). Results shall be returned as tuple of floats.

Test your code with a function, which calculates the square of a number. So calling `apply_to_all(lambda x: x * x, 3, 2, 5)` yields `(9, 4, 25)`.

Hint: For `apply_to_all` one short line of code suffices, but you may use more.

Solution:

```
# your solution
```

54.9.4 Composition

Write a function `apply_composition` which takes an arbitrary number of positional arguments.

- The first argument is mandatory. It's a float.
- All other arguments are optional. They are real functions of a real variable, which shall be applied to the first argument.

Example: If three functions are passed to your function, then the first is to be applied to the float, the second is to be applied to the result of the first function, and the third is to be applied to the result of the second function.

Test your code with `23.42` and the following functions:

- square a number
- add 3
- sine (from `math` module)
- multiply by 2

Result should be `-1.9784623024455807`.

Solution:

```
# your solution
```

54.9.5 Sorting

Sort a list of paths by file name. Use `list.sort` with custom sort key.

Test your code with

```
['/some/path/file.txt',
'/another/path/xfile.txt',
'file_without_path.xyz',
'../relative/path/abc.py',
'/no/extension/some_file.txt']
```

Result should be

```
['../relative/path/abc.py',
 '/some/path/file.txt',
 'file_without_path.xyz',
 '/no/extension/some_file.txt',
 '/another/path/xfile.txt']
```

Solution:

```
# your solution
```

54.9.6 Loop versus Recursion

Given an integer n calculate $n!$ (see *Combinatorics* (page 1019) for a definition) using a loop. Then calculate $n!$ by exploiting the recursive rule $n! = n \cdot (n-1)!$. Test both functions with $10! = 3628800$.

Solution:

```
# your solution
```

54.10 Object-Oriented Programming

Read *Inheritance* (page 153) before you start with the exercises.

54.10.1 Abstract Animals

Create a class `Animal` encapsulating an animal. Members:

- variable `legs` (number of legs provided to the constructor),
- variable `weight` (weight of the animal provided to the constructor,
- function `is_lightweight` (returns `True` if weight is below 10 kg, `False` else),
- function `show_up` (show an ASCII art[639] representation of the animal),
- function `say_something` (a virtual method).

The `show_up` method has to take into account the `legs` variable. Up to the number of legs we do not know anything about the animal's appearance. Be creativ and print an abstract animal with the correct number of legs!

No idea? What about this 8-legged abstract animal:

```
(########):
 //||||\\
```

Bonus: Everytime an `Animal` object is created a message 'An animal with … legs is hiding somewhere.'

Test your code by creating and showing animals with 0, 1,…, 8 legs.

Solution:

```
# your solution
```

54.10.2 Dogs

Derive a class `Dog` from the `Animal` class (cf. exercise above). Implement a proper `say_something` method ('Wau', for instance) and reimplement `show_up` to show a dog instead of an abstract animal.

Note, that the constructor only takes the dog's weight as argument. The constructor should call the base class' constructor to set the correct number of legs (and show the bonus message, if implemented).

In your test code also check if the dog is lightweight.

Solution:

```
# your solution
```

[639] https://en.wikipedia.org/wiki/ASCII_art

54.10.3 Sitting Dog

Add methods `sit_down` and `stand_up` to your `Dog` class. Depending on which of both methods was called last, `show_up` shall show a sitting or a standing dog.

Solution:

```python
# your solution
```

54.10.4 Fish

Derive a `Fish` class from `Animal`. Add a member function `to_sticks`. After calling this function once, `show_up` should show 10 fish sticks per kilogram of the fish instead of a live fish.

Don't forget to implement `say_something` (maybe 'Blub' or, more realistic, '').

Solution:

```python
# your solution
```

CHAPTER

FIFTYFIVE

MANAGING DATA

55.1 NumPy Basics

Before solving these basic NumPy exercises you should have read *NumPy Arrays* (page 165), *Array Operations* (page 171), *Advanced Indexing* (page 175), and *Vectorization* (page 176). Only use NumPy features discussed there.

```
import numpy as np
```

55.1.1 Maximum

Given two arrays print the largest value of all elements from both arrays. Use `np.max` or `np.maximum` or both.

Test your code with

$$\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix}\qquad\text{and}\qquad\begin{pmatrix}1&2&3\end{pmatrix}.$$

Solution:

```
# your solution
```

55.1.2 Largest Difference

Print the largest elementwise absolute difference between two equally shaped arrays.

Test your code with

$$\begin{pmatrix}1.3&2.4\\3.5&6.5\end{pmatrix}\qquad\text{and}\qquad\begin{pmatrix}1.25&2.34\\3.499&6.55\end{pmatrix}$$

Solution:

```
# your solution
```

55.1.3 Comparisons 1

Given an array of integers, test whether there is a one in the array. Print a message showing the result.

Test your code with

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \text{and also with} \quad \begin{pmatrix} 0 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

Solution:

```
# your solution
```

55.1.4 Comparisons 2

Test whether all elements of an array are positive. Print a message showing the result.

Test your code with

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \text{and also with} \quad \begin{pmatrix} 1 & -2 & 3 \\ -4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}.$$

Solution:

```
# your solution
```

55.1.5 Indexing 1

Set all elements between -1 and 1 to zero. Remember: Avoid loops wherever possible.

Test you code with

$$\begin{pmatrix} -2 & -0.5 & 0 \\ 0.4 & 5 & 0.9 \\ 1 & 8 & 9. \end{pmatrix}$$

Solution:

```
# your solution
```

55.1.6 Indexing 2

Write a function which takes an integer n and returns an $n \times n$ matrix with ones at the borders and zeros inside. Data type of the matrix should be `int8`.

Example for $n = 5$:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Solution:

```
# your solution
```

55.1.7 Indexing 3

Write a function which takes an arbitrarily sized matrix and returns a matrix having the same elements as the original matrix, but being bordered by additional zeros. The returned matrix should have the same data type as the original matrix.

Example:

$$\text{input:} \quad \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}, \qquad \text{output:} \quad \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Solution:

```
# your solution
```

55.1.8 Broadcasting

Take the first row of a matrix and add it to all other rows of the matrix. Print the resulting matrix.

Test your code with

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}.$$

Solution:

```
# your solution
```

55.1.9 Not as Easy as it Seems

Given a square matrix with odd number of rows/columns replace all but the boundary elements by zeros and write the mean of all boundary elements to the center element. Print the modified matrix.

Example:

$$\text{original matrix:} \quad \begin{pmatrix} -1 & 2 & -1 & 2 & -1 \\ 2 & -2 & 3 & -2 & 2 \\ -1 & 3 & -3 & 3 & -1 \\ 2 & -2 & 3 & -2 & 2 \\ -1 & 2 & -1 & 2 & -1 \end{pmatrix}, \qquad \text{result:} \quad \begin{pmatrix} -1 & 2 & -1 & 2 & -1 \\ 2 & 0 & 0 & 0 & 2 \\ -1 & 0 & 0.5 & 0 & -1 \\ 2 & 0 & 0 & 0 & 2 \\ -1 & 2 & -1 & 2 & -1 \end{pmatrix}.$$

Solution:

```
# your solution
```

55.2 Image Processing with NumPy

Images can be represented by NumPy arrays with two (grayscale) or three (color) dimensions. Thus, basic image processing like color transforms and cropping reduce to operations on NumPy arrays. Before solving the exercises you should have read *Efficient Computations with NumPy* (page 165). Only use NumPy features, no additional modules.

To show results (images) on screen use Matplotlib.

```
import numpy as np
import matplotlib.pyplot as plt
```

Call the following function whenever you want to see an image:

```
def show_image(img):
    ''' Show 2d or 3d NumPy array img as image (color range 0...1). '''

    # check range
    if img.min() < 0 or img.max() > 1:
        print('Color values out of range!')
        return

    # check dims
    if img.ndim == 2:     # disable color normalization for grayscale
        plt.imshow(img, vmin=0, vmax=1, cmap='gray')
    elif img.ndim == 3:
        plt.imshow(img)
    else:
        print('Wrong number of dimensions!')
        return

    plt.show()
```

Hint: Start with the first exercise and complete exercises one by one. Each exercise will use results from the previous one.

55.2.1 Load Images

Load all images (NumPy arrays) contained in `pasta.npz`. The file contains three color images. Name the arrays `img1`, `img2`, `img3`.

Print the images' shapes and show the images with `show_image`. What's the numeric range of the pixel values (floats 0…1 or integers 0…255)?

Solution:

```
# your solution
```

55.2.2 Convert to Grayscale

Write a function `rgb2gray` which takes a color image (3d array) and returns a grayscale image (2d array). Calculate gray levels as pixelwise mean of red, green and blue values.

Apply the function to the three images and show results.

Hint: NumPy has a `mean` function. The `axis` parameter could be of interest.

Solution:

```
# your solution
```

55.2.3 Cropping

Each image shows a piece of alphabet pasta on a perfect black background (color value 0). Write a function `auto_crop` which finds the bounding box of a grayscale image and returns a copy (not a view!) of the corresponding subarray. The bounding box is the rectangle that contains the pasta piece without black margin.

Apply the function to the grayscale pasta images and show the results.

Solution:

```
# your solution
```

55.2.4 Centering

Write a function `center` which takes a grayscale image and an integer `n`. The return value shall be a grayscale image of size `nxn` with black background and the passed image positioned in the new images center (identical margin width at all sides).

Place each cropped pasta image in a 50x50 image and show the results.

Solution:

```
# your solution
```

55.2.5 Vectorized Cropping

Implement the above `auto_crop` function without using loops, that is, fully vectorized. Compare execution times.

Solution:

```
# your solution
```

55.3 Pandas Basics

Before solving these basic Pandas exercises you should have read *Series* (page 200) and *Data Frames* (page 210).

For these exercises we use a dataset describing used cars obtained from kaggle.com[640]. Licences: Open Data Commons Database Contents License (DbCL) v1.0[641] and Open Data Commons Open Database License (ODbL) [642].

```
import pandas as pd

data = pd.read_csv('cars.csv')
```

[640] https://www.kaggle.com/nehalbirla/vehicle-dataset-from-cardekho
[641] http://opendatacommons.org/licenses/dbcl/1.0/
[642] https://opendatacommons.org/licenses/odbl/summary/

55.3.1 First Look

Basic Information

Print the following information about the data frame `data`:

- first 10 rows,
- number of rows,
- basic statistical information,
- column labels, data types, memory usage.

Solution:

```
# your solution
```

Missing Values

Are there missing values in `data`?

Solution:

```
# your answer
```

Value Counts

Use `DataFrame.nunique`[643] to get the number of different values per column.

Solution:

```
# your solution
```

Unique Car Models

Use `DataFrame.value_counts`[644] to get the number of unique `'name'`-`'year'` combinations.

Solution:

```
# your solution
```

55.3.2 Restructure Columns

New Columns

Append a column `'manual_trans'` containing `True` where column `'transmission'` shows `'Manual'`, else `False`.

Append a column `'age'` showing a car's age (now minus `'year'`).

Solution:

```
# your solution
```

[643] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.nunique.html
[644] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.value_counts.html

Remove Columns

Remove columns `'seller_type'`, `'transmission'`, and `'owner'`.

Solution:

```
# your solution
```

55.3.3 Mean Price

Series with String Index

Create a Pandas series `price` with column `'name'` as index and column `'selling_price'` as data.

Solution:

```
# your solution
```

Mean

Calculate mean price for model `'Maruti Swift Dzire VDI'`.

Solution:

```
# your solution
```

55.3.4 Kilometers per Year

Boolean Indexing

Use boolean row indexing to get a data frame `one_model` with columns `'km_driven'` and `'age'` containing only rows with `'name'` equal to `'Maruti Swift Dzire VDI'`.

Solution:

```
# your solution
```

New Column

Add a column `'km_per_year'` to the `one_model` data frame containing kilometers per year.

Solution:

```
# your solution
```

Mean

Get the mean of column `'km_per_year'` in `one_model`.

Solution:

```
# your solution
```

55.3.5 Oldest Car

Find the oldest car in `data` and print its name and manufacturing year. Have a look at Pandas' documentation[645] for suitable functions.

Solution:

```
# your solution
```

55.4 Pandas Indexing

Before solving these exercises you should have read *Advanced Indexing* (page 219) and *Dates and Times* (page 229).

```
import pandas as pd
```

55.4.1 Cars

For these exercises we use a dataset describing used cars obtained from kaggle.com[646]. Licences: Open Data Commons Database Contents License (DbCL) v1.0[647] and Open Data Commons Open Database License (ODbL) [648].

```
data = pd.read_csv('cars.csv')
```

Create Multi-Level Index

Create a multi-level index for the data frame from columns `'name'` and `'year'`.

Solution:

```
# your solution
```

[645] https://pandas.pydata.org/docs/
[646] https://www.kaggle.com/nehalbirla/vehicle-dataset-from-cardekho
[647] http://opendatacommons.org/licenses/dbcl/1.0/
[648] https://opendatacommons.org/licenses/odbl/summary/

Select Model

Print all rows for the `'Maruti Swift Dzire VDI'` 2018 model.

Solution:

```
# your solution
```

Diesel

Select all 2018 cars and use `value_counts`[649] to get the percentage of Diesel cars.

Solution:

```
# your solution
```

Old Cars

Print all cars with more than 100000 kilometers driven and manufactured before 2000.

Solution:

```
# your solution
```

55.4.2 E-Mails

Consider an email account receiving emails every day. Use the following code to generate a list `times` of time stamps representing arrival times of emails.

```
import numpy as np
rng = np.random.default_rng(0)

n_mails = 1000
start_time = pd.Timestamp('2019-01-01 00:00:00')
end_time = pd.Timestamp('2020-01-01 00:00:00')

total_seconds = int((end_time - start_time).total_seconds())
seconds = rng.integers(0, total_seconds, n_mails)
times = [start_time + pd.Timedelta(sec, unit='s') for sec in seconds]
del seconds
```

Mails per Day

Given the list of time stamps of incoming mails create a series with daily mail counts.

Solution:

```
# your solution
```

[649] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.value_counts.html

Mails per Morning

Every day the user only answers mails received not after 7:00am that day. From the list of time stamps create a series with daily mail counts at 7:00am. Hint: Have a look at the `offset` argument of `Series.resample`[650]; `label` might be of interest, too.

Solution:

```
# your solution
```

Mails per Business Day Morning

Assume the user reads and answers emails at business days only (again, at 7:00am). Create a series containing the numbers of mails to process at each business day.

Solution:

```
# your solution
```

Vacation

From the results of the previous task get the number of mails arriving during winter vacation in January and February. Use a variable for the year of interest:

```
year = 2019
```

Write code which works for all years (leap year or not).

Solution:

```
# your solution
```

55.5 Advanced Pandas

Before solving these exercises you should have read *Advanced Indexing* (page 219), *Dates and Times* (page 229), *Categorical Data* (page 235), and *Restructuring Data* (page 239).

```
import pandas as pd
import numpy as np
```

55.5.1 Grades

Use the following code to create a series containing student IDs as index and points in exam as data:

```
rng = np.random.default_rng(123)

n_students = 20
max_points = 40

student_ids = rng.integers(20000, 25000, n_students)
points = np.floor(rng.normal(0.6 * max_points, 0.2 * max_points, n_students))
```

(continues on next page)

[650] https://pandas.pydata.org/docs/reference/api/pandas.Series.resample.html

(continued from previous page)

```
points = points.clip(0, max_points).astype(np.int8)
exam_points = pd.Series(points, index=student_ids)

exam_points
```

Understand the Code

What do the two `points = ...` lines in the above code do in detail?

Solution:

```
# your answer
```

Points to Grades

Add a column to the series (resulting in a data frame) containing corresponding grades. Conversion from points to grade is as follows:

percent of points	grade
less than 40	5.0
at least 40	4.0
at least 54	3.7
at least 60	3.3
at least 68	3.0
at least 74	2.7
at least 80	2.3
at least 84	2.0
at least 88	1.7
at least 92	1.3
at least 96	1.0

Use `pd.cut`[651] to get the grades. Result should look as follows:

```
       points grade
id
20077      24   3.3
23411      11   5.0
22964      33   2.3
...
```

The `'grade'` column should be of categorical type.

Solution:

```
# your solution
```

[651] https://pandas.pydata.org/docs/reference/api/pandas.cut.html

Mean Grade

Get the mean grade for all students who passed the exam (grade better than 5).

Solution:

```
# your solution
```

55.5.2 Cafeteria

For these exercises we use the dataset obtained in the *Cafeteria* (page 947) project.

```
data = pd.read_csv('meals.csv', names=['date', 'category', 'name', 'students',
 ↪'staff', 'guests'])

data
```

Dates

Convert `'date'` column to `Timestamp`. Hint: the `pd.to_datetime`[652] function is very flexible.

Solution:

```
# your solution
```

Categories

Set type of `'category'` column to categorical. Print all categories.

Solution:

```
# your solution
```

Mean Price per Category

Get mean students/staff/guests prices per category. Sort results by students price.

Solution:

```
# your solution
```

Prices over Time

Drop rows with `nan` or `0.0` prices. Then get minimum, average, maxmium students prices per day. Create a data frame with three columns `'min'`, `'mean'` `'max'` and `DatetimeIndex`. Call the data frame's `plot`[653] method (works without arguments) to visualize the results.

Solution:

```
# your solution
```

[652] https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html
[653] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html

Restructuring

Create a data frame showing prices only, no meal names. Use dates for row index. Column index shall be multi-level with first level showing the category and second level showing the price level (students/staff/guests).

Solution:

```
# your solution
```

55.6 Pandas Vectorization

Before solving these exercises you should have read *High-Level Data Management with Pandas* (page 199).

```
import pandas as pd
```

For these exercises we use the dataset obtained in the *Cafeteria* (page 947) project.

```
data = pd.read_csv('meals.csv', names=['date', 'category', 'name', 'students',
↪'staff', 'guests'])

data
```

55.6.1 Preprocessing

Data Types

Convert `'date'` and `'category'` columns to `Timestamp` and `category`, respectively (see *Advanced Pandas* (page 886) exercises).

Solution:

```
# your solution
```

Count Categories

For each category give the number of meals.

Solution:

```
# your solution
```

Remove Categories

Remove Categories which do not contain full-fledged meals descriptions (e.g., `'Salat Bar'`).

Solution:

```
# your solution
```

55.6.2 Remove Allergens and Additives

Most meals contain information on alergenes and additives (numbers in parantheses). Remove the information to get more readably meal descriptions. Implement the removal procedure twice: without and with vectorized string operations. Get and compare execution times.

Solution:

```
data['name_backup'] = data['name']
```

```
%%timeit
# your solution without vectorized string operations
```

```
data['name'] = data['name_backup']
data = data.drop(columns=['name_backup'])
```

```
%%timeit
# your solution with vectorized string operations
```

55.6.3 Simplify Meal Descriptions

Create a new column `'simple'` from the `'name'` column by removing all lower-case words, all punctuation marks and so on. Only words starting with an upper-case letter are allowed.

Solution:

```
# your solution
```

55.6.4 All Meals with...

Given a key word (e.g., `'Kartoffel'`) get the number of meals containing the keyword and print all meal descriptions.

Solution:

```
# your solution
```

55.6.5 Meal Plot

For each day get the number of meals containing some keyword (e.g., `'Kartoffel'`). Call `Series.plot` to visualize the result.

Solution:

```
# your solution
```

CHAPTER

FIFTYSIX

DATA VISUALIZATION

56.1 Matplotlib Basics

Before solving these exercises you should have read *Matplotlib Basics* (page 251).

```
import numpy as np
import matplotlib.pyplot as plt
```

56.1.1 Simple Function Plot

Plot the sine function for angles from 0 to 2π. Label the axes and show a title. Use a red line without markers.

Solution:

```
# your solution
```

56.1.2 Synchronized Axes

Create a figure containing $\sin(x)$ and $2\,\cos(x)$ for $x\in[0,2\,\pi]$ next to each other and having identical vertical axes.

Solution:

```
# your solution
```

56.1.3 Plotting Poles

Plot $f(x)=\tan(x)$ for $x\in[-\pi,\pi]$ and $f(x)\in[-10,10]$. Add dashed vertical lines at the poles.

Solution:

```
# your solution
```

56.1.4 Parametric Curves

Plot a curve $(x(t), y(t))$ in the plane. Align plots of $x(t)$ and $y(t)$ properly below and beside the curve plot, respectively. As an example you could use $x(t) = \cos t + 2\,\sin(2\,t)$ and $y(t) = \sin(t) + 2\,\cos(3\,t)$ for $t \in [0, 2\,\pi]$.

Solution:

```
# your solution
```

56.1.5 Ticks and Grids

Plot $f(x) = \sin(x)$ for $x \in [-1, 7]$. Place major tick labels at multiples of π and minor tick labels at multiples of $\pi/2$. The y axis should have major ticks at $-1, 0, 1$ and minor ticks in between. Activate grid lines for minor and major ticks on x axis and for major ticks on y axis. Further, use increased line width for grid lines at major ticks.

Solution:

```
# your solution
```

56.1.6 Circular Colorbar

Visualize the function $f(x, y) = \arcsin(x) + \arccos(y) + \pi/2$ for $(x, y) \in [-1, 1] \times [-1, 1]$ with a scatter plot of uniformly distributed points with color given by the function values. Function values are angles in $[0, 2 \cdot \pi]$. Thus, colors at both ends of the colorbar should coincide. Create a circular colorbar next to the scatter plot.

Solution:

```
# your solution
```

56.1.7 Bent Arrows

Plot $f(x) = \cos x$ for $x \in [-2\,\pi, 2\,\pi]$. Add the text 'two local minima' to the plot and connect it with two arrows to the minima. Take care that the arrows do not intersect with the graph.

Solution:

```
# your solution
```

56.1.8 Color Channels

Load the image `tux.png` and plot it four times: original, red channel, green channel, blue channel (hint: a simple way to create custom colormaps is `matplotlib.colors.LinearSegmentedColormap.from_list`[654]).

Solution:

```
# your solution
```

[654] https://matplotlib.org/stable/api/_as_gen/matplotlib.colors.LinearSegmentedColormap.html#matplotlib.colors.LinearSegmentedColormap.from_list

56.1.9 Pixelbased Postprocessing

Plot $f(x)=\sin x$ for $x\in[-2\,\pi,2\,\pi]$. Fade out line ends with GIMP.

Solution:

```
# your solution
```

56.1.10 Vectorbased Postprocessing

Plot $f(x)=\sin x$ for $x\in[-1,7]$ and plot the sine approximations $g(x)=-\frac{4}{\pi^2}\,(x-\frac{\pi}{2})^2+1$ and $h(x)=-\frac{4}{\pi^2}\,(x-\frac{3\,\pi}{2})^2-1$. Plot the region $x\in[\pi-\frac{1}{2},\pi+\frac{1}{2}]$ in a separate figure. Save both figures as vector graphics and add the detail view to the sine plot with Inkscape.

Solution:

```
# your solution
```

56.2 Advanced Matplotlib

Before solving these exercises you should have read *Matplotlib* (page 251).

56.2.1 Cycloids

Create a never ending animation which plots a curve in the following way: the pen is rotating around a center, which itself rotates around the origin. Use different radiuses and rotation speeds for both rotations.

Solution:

```
# your solution
```

CHAPTER

FIFTYSEVEN

REINFORCEMENT LEARNING

57.1 Modeling

Before solving these exercises you should have read *Overview and Examples* (page 793).

57.1.1 Tic-Tac-Toe

Task: How many actions has Tic-tac-toe[655] seen from an agent's perspective?

Solution:

```
# your answer
```

Task: How many states has Tic-tac-toe seen from an agent's perspective, that is, if the next move is the agent's move? Include end states, too (although the agent can't choose any valid action based on an end state). Give a rough estimate (upper bound), then try to find better estimates. Bonus: write some Python code to get the exact value.

Solution:

```
# your answer
```

57.1.2 Connect Four

Task: How many actions has Connect four[656] seen from an agent's perspective?

Solution:

```
# your answer
```

Task: Give a simple and maybe also a more precise upper bounds for the number of states in Connect four.

Solution:

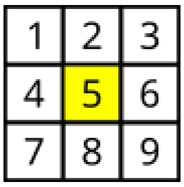
```
# your answer
```

[655] https://en.wikipedia.org/wiki/Tic-tac-toe
[656] https://en.wikipedia.org/wiki/Connect_Four

57.2 Markov Decision Processes

Before solving these exercises you should have read *Markov Decision Processes* (page 807).

We consider a tiny 3-by-3 grid world.



State information contains the agent's position only. Actions are 'go left', 'go right', 'go up', 'go down'. Invalid actions do not change state. Moving to cell 5 yields reward 1 and places the agent randomly on 2, 4, 6 or 8. All other rewards are 0.

57.2.1 Environment

Task: How many different states do we have in this grid world? Is moving in this grid world an collecting as much reward as possible an episodic or a continuing task? Is the task stationary or non-stationary?

Solution:

```
# your answer
```

Task: Write down the environment dynamics p for the grid world defined above.

Solution:

```
# your answer
```

57.2.2 Policies

Task: Write down the uniformly random policy for the grid world.

Solution:

```
# your answer
```

Task: Find a deterministic policy that maximizes return (with discount factor $\gamma < 1$).

Solution:

```
# your answer
```

Task: Are there policies that maximize return for $\gamma = 1$, but not for $\gamma < 1$?

Solution:

```
# your answer
```

57.2.3 Bellman Equations

Task: Write down the Bellman equations for state values for the deterministic policy 1→2→3→6→9→8→7→4→5. Solve the equations for different γ (try $\gamma = 1$, too). Then calculate action values from the state values.

Solution:

```
# your answer
```

Task: Calculate the state-value function for your return maximizing policy from above. Prove that the policy is indeed optimal.

Solution:

```
# your answer
```

Part X

Projects

CHAPTER

FIFTYEIGHT

INSTALL AND USE PYTHON

There are lots of different ways for installing and using Python and related tools. Choose the one which suits your needs and your approach to work.

58.1 Working with JupyterLab

In this project you learn basic usage of JupyterLab[657], a web interface for Python programming. Read *Python and Jupyter* (page 37) to get some information on the relation between Python and JupyterLab.

This project is available as video (with German audio and subtitles only):

Note: JupyterLab is not restricted to Python programming, but supports almost every other programming language.

58.1.1 Start JupyterLab

Task: Open a webbrowser and go to some JupyterLab provider. Students of Zwickau University should go to Gauss[658].

Hint: You may also run a local JupyterLab instance. See *Install Jupyter Locally* (page 905) project for installation and start.

After starting JupyterLab you should see a file manager sidebar on the left and the launcher tab in the working area on the right.

Task: Create a new notebook file by clicking the Python button in the launcher tab's Notebook section.

This creates a file `Untitled.ipynb` in the current directory. To change the file name click 'File' and 'Rename Notebook…' in the top menu.

[657] https://jupyter.org
[658] https://gauss.fh-zwickau.de

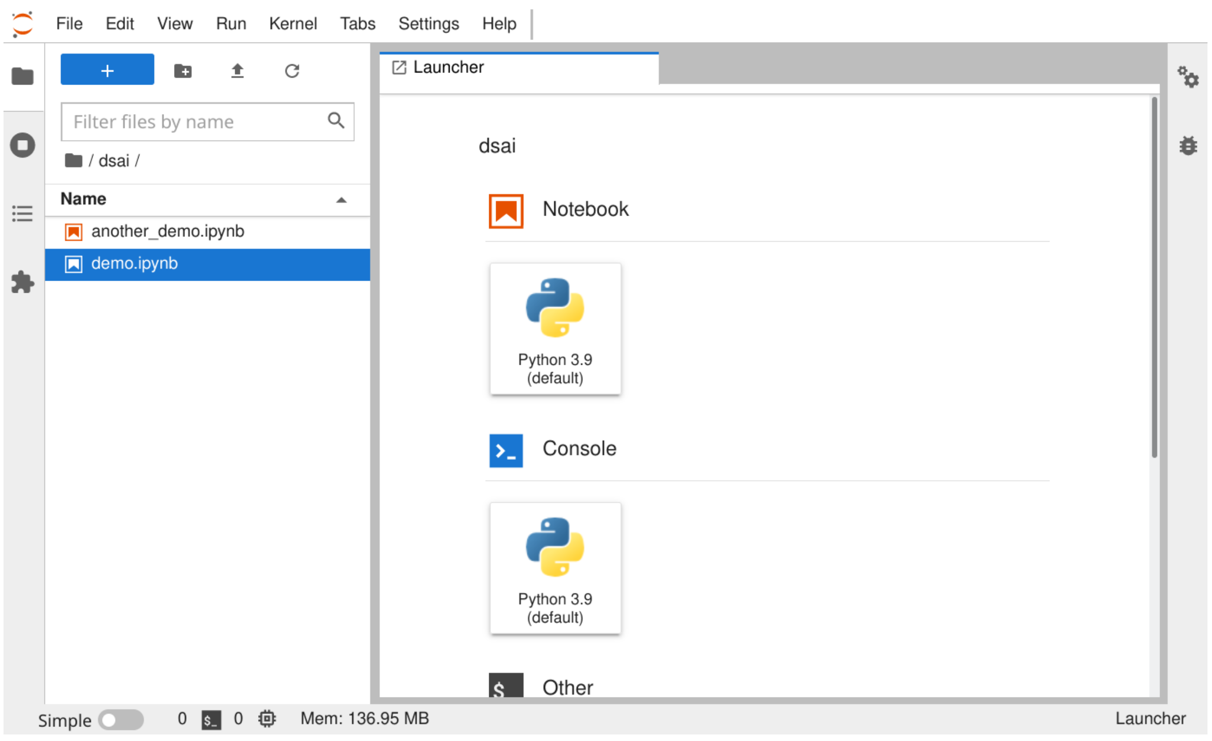


Fig. 58.1: File manager and launcher tab are shown in JupyterLab's initial view.

58.1.2 Write and Execute Code

The new notebook file shows an empty code cell.

Task: Write the following Python code to the code cell.

```
a = 2
b = 3
print(a + b)
```

To run the code do one of the following:

- Click the 'run selected cells' button in the toolbar (triangle button).
- Press Ctrl + Return on your keyboard (runs code and keeps focus on current cell).
- Press Shift + Return on your keyboard (runs code and goes to next cell).

One and the same cell may be executed many times (with or without modifiying the code).

Task: Run your code.

A cell's output is shown directly below the cell.

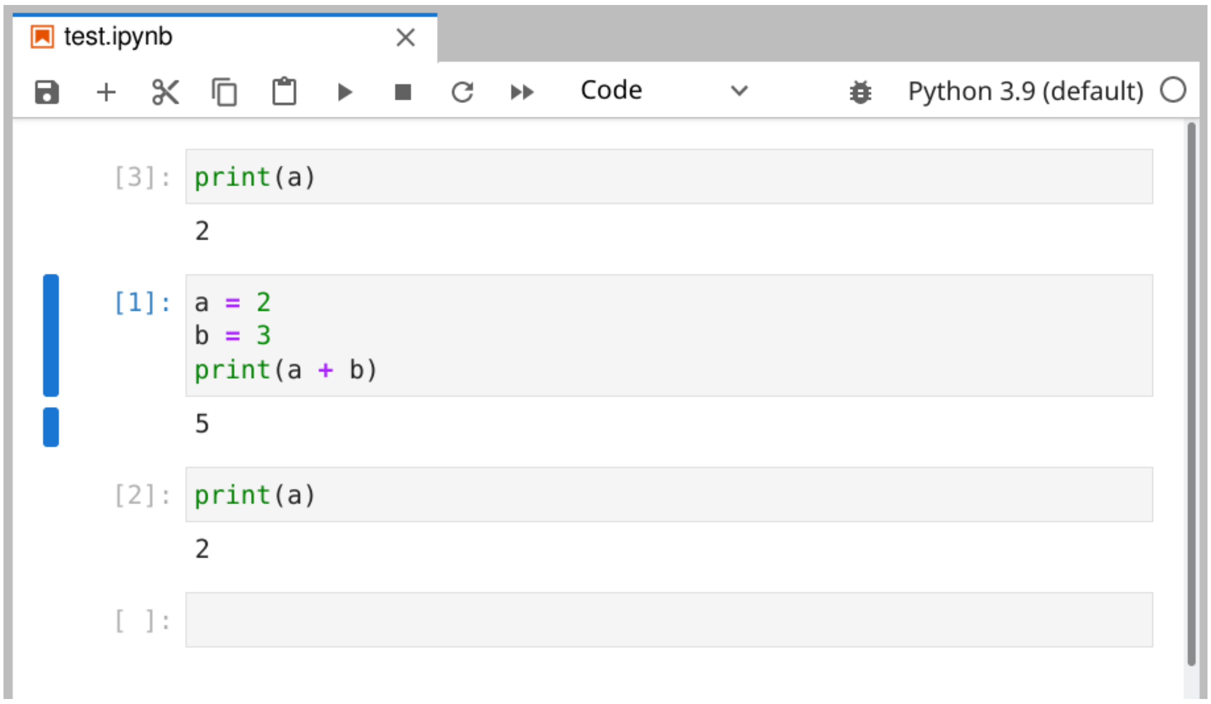
Task: Write `print(a)` in a new cell, execute the cell and observe the output.

Each cell knows what happend in other cells.

Task: Click into the first cell. Then insert a new cell by pressing ESC, then A on your keyboard (not ESC + A). Write `print(a)`, execute and observe output.

Order of cells in the notebook is not of importance. Whether a cell knows about other cells depends on the order of execution!

Task: Save your notebook.



```
[3]: print(a)
     2

[1]: a = 2
     b = 3
     print(a + b)
     5

[2]: print(a)
     2

[ ]:
```

Fig. 58.2: Your notebook should look as depicted here if you've completed all tasks above.

58.1.3 Kernels

JupyterLab is the connection between you and Python. The code you write to a code cell is send to Python for execution. JupyterLab watches for outputs of your program and displays them to you. The background Python part is referred to as Python kernel.

The connection between a notebook in JupyterLab and the Python kernel is very loose. You may open a notebook file without running a kernel. Then code execution isn't possible. Or you may have a running Python kernel although you closed your notebook.

Task: Close your notebook file's tab. Then click the square symbol in the sidebar.

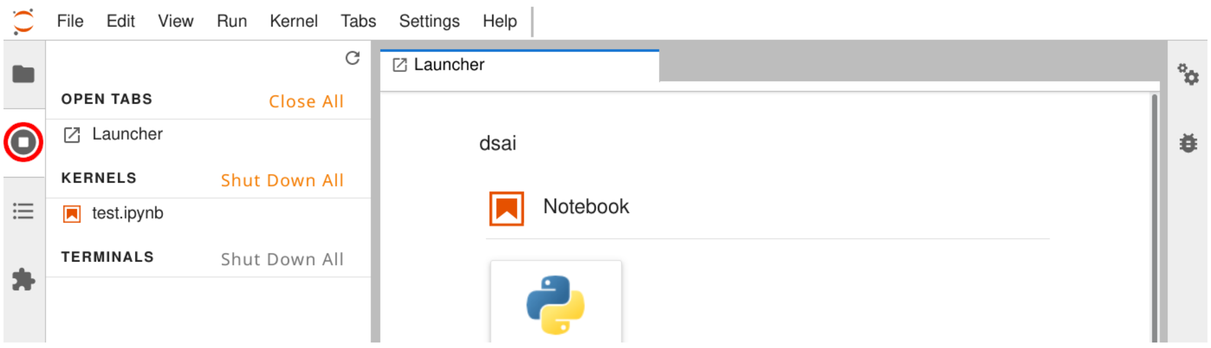


Fig. 58.3: The square button brings up a list of running kernels and some other information.

The Python kernel of our test notebook is still running. Clicking the kernel line reopens the notebook tab. The X button (only shown on hover) shuts the kernel down.

Task: Shut down the kernel.

Hint: Instead of closing a tab and its kernel separately you may also click 'File' and 'Close and Shutdown Notebook'

Reopening a notebook runs a fresh kernel.

Task: Switch to the file manager in the sidebar and open your notebook (double click its name).

Task: Execute the first (that is, top most) cell.

Python complains about an unknown name. The fresh kernel hasn't seen the `a = 2` line up to now, because we did not execute it. Executing the second cell makes the first work correctly.

Important: Always try to create notebooks which can be executed in the same order as cells appear in the notebook!

Hint: If your code takes too long to run or if it won't stop for some reason, click 'Kernel' and 'Interrupt Kernel' in the menu. This stops code execution and makes the kernel wait for new code.

Hint: To test your notebook's behavior after launching a fresh kernel but without reopening the file, click 'Kernel' and 'Restart Kernel…' in the menu.

58.1.4 Cell Types

Up to now we only used code cells. Another important cell type are Markdown cells. Markdown is a markup language for writing formatted text.

```
# Some Heading

## A Subsection

Text without formatting. Here's a [link to nowhere](https://nowhere). And a list:
* first item
* second item

This is **bold** and *italic* text.
```

Task: Create a new cell (ESC, then A or B to insert new cell above or below current cell). Switch cell type to 'Markdown', either via dropdown in toolbar or via ESC, then M.

Task: Write some Markdown code to the cell and execute the cell. To modify Markdown code double click the cell. Then edit and execute again.

58.1.5 Terminals

JupyterLab allows to run one or more terminals. A terminal is a text interface to the computer's operating system. There you can use operating system features and programs not accessible through JupyterLab. Exampels are copying files are deleting non-empty directories.

Note: Almost all remote JupyterLab instances run on Linux machines. So in a terminal you have to use Linux commands, not Windows. Linux, macOS, OpenBSD and many other operating systems share a common set of commands. Only Windows has its own set.

Task: Open a terminal (click the Terminal button in the launcher tab's Other section). Then type `ls` to get a list of all files in the current directory. Then type `logout` to close the terminal.

The `pwd` command prints the current directory's path. Commands for working with files are `cd` (change directory), `cp` (copy), `mv` (move, rename), `rm` (remove), `mkdir` (create directory), `rmdir` (remove directory). See Unix Commands[659] for more commands and usage information.

Hint: If you close a terminal tab without typing `logout` the terminal remains active in the background. To reopen it or to shut it down click the 'Running Terminals and Kernels' button in the sidebar.

58.1.6 Quit JupyterLab

To quit JupyterLab click 'File' and 'Log Out'. Closing the browser tab without logging out may cause security problems.

Before leaving JupyterLab shut down all kernels and terminals. This saves resources on the server. Most Jupyter providers (including Gauss at Zwickau University) will shut down inactive kernels and JupyterLab sessions after some hours automatically.

Note: It's possible to log out from JupyterLab and have a kernel running. Coming back some hours or days later one can fetch the outputs of long running tasks. Corresponding workflow is described in the *Long-Running Tasks* (page 912) project.

58.2 Install Jupyter Locally

We want to set up an extensible system for Python development and data science in general, including Jupyter as one component. Here we only install the base system. From time to time tools and Python libraries can be added on demand.

Hint: A Python library is a collection of Python code files extending Python's set of commands.

58.2.1 Conda

Before we start, we should become aware of two problems:

- A Python development environment consists of many different tools, because many small tools are more flexible than one monolithic all-in-one tool. In principle, one could install all of them manually. On Windows systems this would be very time-consuming. Other operating systems, which adhere to the small tools approach (Unix-like systems), have a package manager for efficient software installation.
- Different programming tasks could require different tools. Sometimes a tool prevents installation of another tool or of another version of itself. This is especially the case for some Python libraries, because some libraries depend on specific versions of others. Installing an additional library could require updating an existing one, but this in turn could corrupt dependencies of already installed libraries.

To circumvent both problems there exist package managers which create and manage multiple separate Python environments. So we may have several different sets of tools and libraries in parallel, switching between them whenever appropriate.

A widely used package manager for Python is Conda[660]. It's part of the Anaconda[661] and Miniconda[662] Python distributions. A Python distribution is a collection of tools and libraries for Python development.

[659] https://en.wikibooks.org/wiki/Guide_to_Unix/Commands
[660] https://conda.io
[661] https://www.anaconda.com
[662] https://docs.conda.io/en/latest/miniconda.html

Miniconda is a light-weight version of Anaconda with fewer tools and libraries pre-installed.

58.2.2 Install Miniconda and Anaconda Navigator

Task: Go to Miniconda Installer List[663] and download a suitable installer for your system. Then follow the install instructions[664] for your system.

Conda is a command line tool. If you feel more comfortable with GUI tools, install Anaconda Navigator[665].

Task: Open a terminal and run `conda install anaconda-navigator` in it. This installs Anaconda Navigator.

Depending on your system now there should be an entry for Anaconda Navigator in your system's app menu. If not add an entry manually. Anaconda Navigator executable should be in `bin` subdirectory of Miniconda's installation directory. On Non-Windows run `which anaconda-navigator` in a terminal to get the path.

Important: Before you install any additional tools with Anaconda Navigator or Conda, read on!

58.2.3 Create a Python Environment

At the moment there is only one Python environment on your system, called `base`. Don't install additional packages to this environment. Create a separate environment for each kind of task, for instance, an environment you use to work through projects and exercises in this book.

Task: Create a new Python environment `ds-book`. Either run `conda create -n ds-book` in a terminal or go to 'Environments' page in Anaconda Navigator. Then click the plus button and follow the GUI instructions.

To switch between environments use Anaconda Navigator or run `conda activate environment_name` in a terminal.

58.2.4 Install JupyterLab

Now that we have a Python environment, it's time to install Jupyter.

Task: In Anaconda Navigator set package filtering to 'All'. Then head for 'jupyterlab' in the package list and mark it for install. A click on 'Apply' starts installation. Alternatively, run `conda install jupyterlab` in a terminal (make sure you have activated the correct environment).

Note: Although you selected only one package for install, many more will be installed due to dependencies. JupyterLab requires a number of other packages and those packages may require others again. Conda manages such dependencies for us.

[663] https://docs.conda.io/en/latest/miniconda.html#latest-miniconda-installer-links
[664] https://conda.io/projects/conda/en/latest/user-guide/install/index.html#regular-installation
[665] https://docs.anaconda.com/anaconda/navigator/
[666] https://xkcd.com/1987

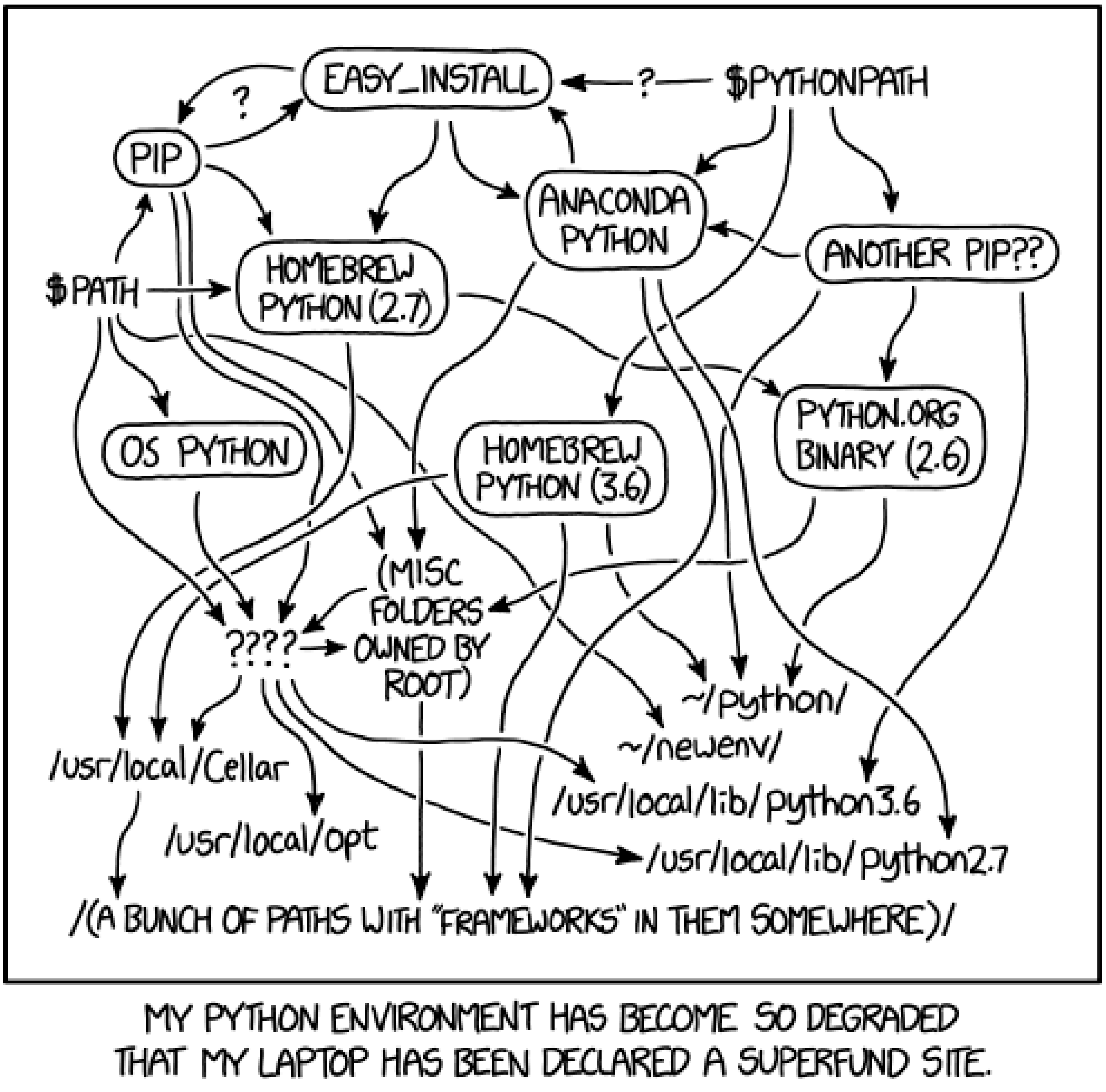


Fig. 58.4: The Python environmental protection agency wants to seal it in a cement chamber, with pictorial messages to future civilizations warning them about the danger of using sudo to install random Python packages.. Source: Randall Munroe, xkcd.com/1987[666]

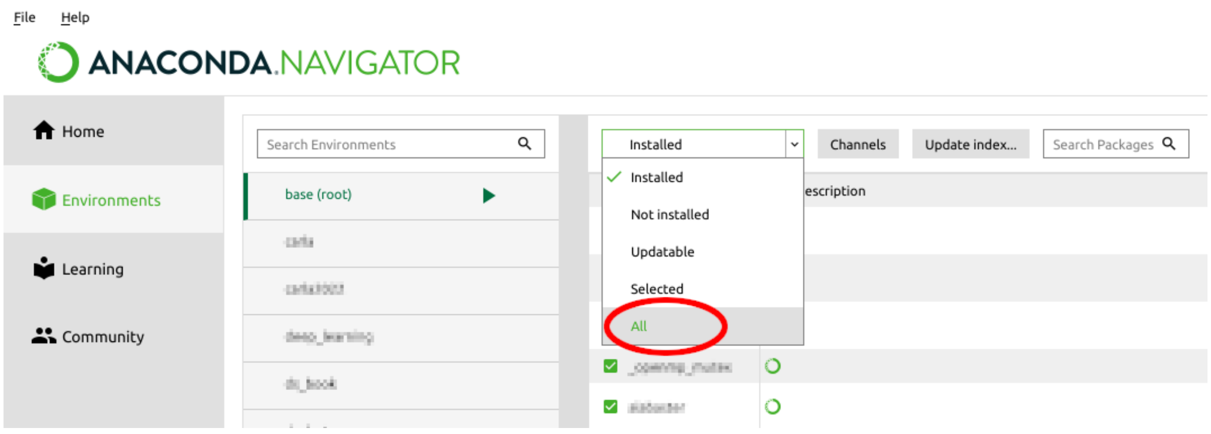


Fig. 58.5: The filter dropdown provides several filters for the package list.

58.2.5 Launch JupyterLab

The Home page of Anaconda Navigator shows a launch button for JupyterLab. Make sure you have selected the correct environment in the dropdown above the launch buttons.

Task: Launch JupyterLab via Anaconda Navigator or from the command line: `jupyter lab` or `jupyter-lab`.

Freshly installed JupyterLab lives in the environment you installed it in. Creating a new Python environment requires to install JupyterLab in this environment, too (if you want to use JupyterLab there).

Default behavior of JupyterLab is to show you your home directory and to disallow visiting directories outside your home directory via JupyterLab. To access a different directory, run JupyterLab from a terminal. Then JupyterLab will show the directory active in the terminal when launching JupyterLab.

58.2.6 Install Python Packages

There's already a basic Python installation in your environment. So you can use Python in JuyterLab. Additional packages (math, visualization,…) can be installed on demand in the same way we installed JupyterLab. Always keep an eye on the environment name when installing. So things will end up in the correct environment.

Hint: Next to Conda there exist other package managers. A very prominent one is Pip[667]. Conda automatically installs Pip in each environment. To install a package with Pip simply write `pip install package_name` in a terminal. Conda will take care of packages installed with Pip, too.

Some packages are only available for install with Conda or only for install with Pip. So both package manager have to be used in parallel.

[667] https://pypi.org/project/pip/

58.2.7 JupyterLab Desktop App

Recently, a JupyterLab Desktop App[668] has been released. This brings the look and feel of usual GUI apps to JupyterLab's start-up process. After start-up there's no difference to browser based JupyterLab.

Handling of different Python environments is somewhat more difficult than with plain JupyterLab.

58.3 Python Without Jupyter

Jupyter is only one of many Python *IDEs* (*integrated development environments*). Although Jupyter is well suited for data science applications, because text, visualizations and code can be mixed in one and the same documents, other tools may be appropriate, too.

58.3.1 Interactive Python Without IDE

Python (better: the Python interpreter) is a stand-alone program for running Python code on the command line. It has an interactive mode, which executes each line of code immediately after writing. Alternatively, we may provide a file containing Python code and the Python interpreter executes the file's content.

Task: Open a terminal and type `python` (don't forget to activate the correct environment with Conda or Anaconda Navigator).

Now the Python interpreter runs in interactive mode. All Python commands are allowed. It's, for instance, a powerful replacement for a calculator.

Task: Type the following line by line

```
1 + 2 * 3
a = 2
b = 3
(a + b) ** 3
```

The result of each line is printed on screen immediately after hitting the return key.

To quit the interpreter we have to call the Python function for leaving a program.

Task: Type `exit()`.

58.3.2 Text Editor Plus Python Interpreter

We may write Python code to a text file and hand it over to the Python interpreter for execution.

There are lots of text editors with additional features for coding, like syntax highlighting, automatic indentation, line numbers. Two common ones are Kate[669] (Linux, MacOS, Windows) and Notepad++[670] (Windows). Others you may hear about are Emacs[671], Vim[672] and Nano[673], especially if you work on non-Windows machines (cloud!).

Task: Create a text file `bye.py` with following content:

```
code = None

while code != "bye":
```

(continues on next page)

[668] https://github.com/jupyterlab/jupyterlab-desktop
[669] https://kate-editor.org/
[670] https://notepad-plus-plus.org
[671] https://www.gnu.org/software/emacs/
[672] https://www.vim.org/
[673] https://www.nano-editor.org/
[674] https://xkcd.com/378

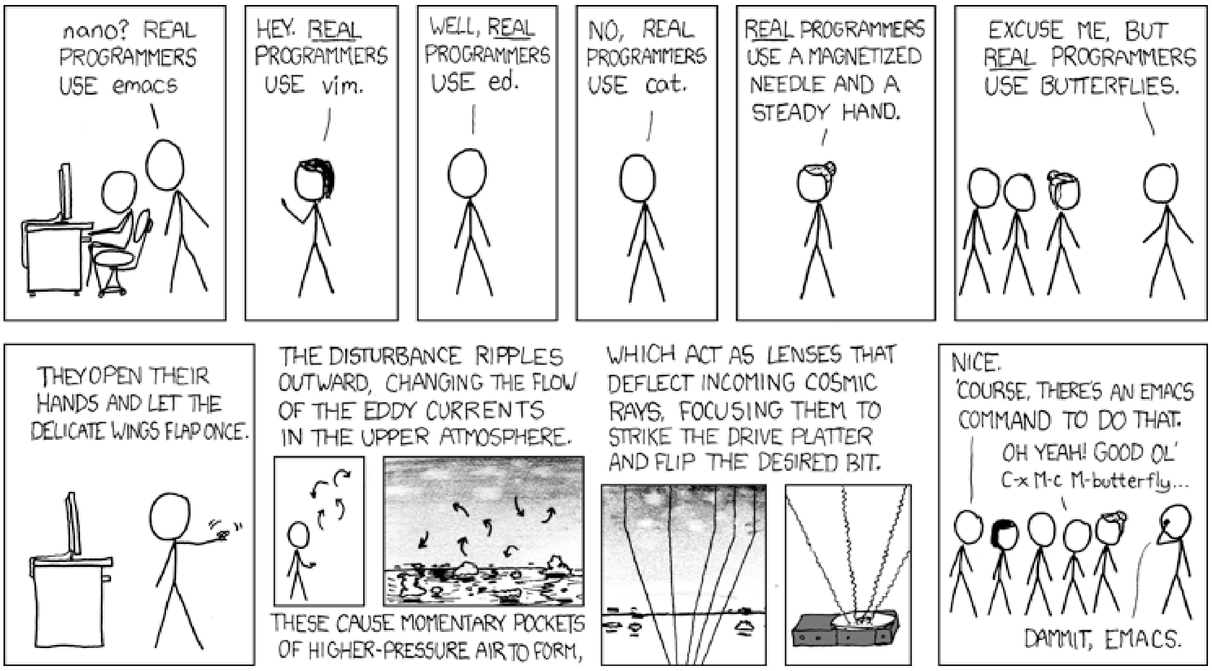

nano? REAL PROGRAMMERS USE emacs
HEY. REAL PROGRAMMERS USE vim.
WELL, REAL PROGRAMMERS USE ed.
NO, REAL PROGRAMMERS USE cat.
REAL PROGRAMMERS USE A MAGNETIZED NEEDLE AND A STEADY HAND.
EXCUSE ME, BUT REAL PROGRAMMERS USE BUTTERFLIES.
THEY OPEN THEIR HANDS AND LET THE DELICATE WINGS FLAP ONCE.
THE DISTURBANCE RIPPLES OUTWARD, CHANGING THE FLOW OF THE EDDY CURRENTS IN THE UPPER ATMOSPHERE.
THESE CAUSE MOMENTARY POCKETS OF HIGHER-PRESSURE AIR TO FORM,
WHICH ACT AS LENSES THAT DEFLECT INCOMING COSMIC RAYS, FOCUSING THEM TO STRIKE THE DRIVE PLATTER AND FLIP THE DESIRED BIT.
NICE.
'COURSE, THERE'S AN EMACS COMMAND TO DO THAT.
OH YEAH! GOOD OL' C-x M-c M-butterfly...
DAMMIT, EMACS.

Fig. 58.6: Real programmers set the universal constants at the start such that the universe evolves to contain the disk with the data they want. Source: Randall Munroe, xkcd.com/378[674]

(continued from previous page)

```
    print("I'm a Python program. Type 'bye' to stop me:")
    code = input()     # get some input from user

    if code == "":
        print("To lazy to type?")
    print("")

print("Bye")
```

Indentation matters! Python uses indentations (white space) to structure code. So don't modify indentation depth here.

Task: Open a terminal, go to your code file's directory and run `python bye.py`.

The Python interpreter now runs your program. When reaching the end of the file, execution stops. Only output from your program is shown. The Python interpreter itself doesn't print anything as long as there are no errors in your program.

58.3.3 Spyder

Spyder[675] is a Python IDE for scientific programming with look and feel similar to Octave and Matlab. To use Spyder install the `spyder` package with Anaconda Navigator or via `conda install spyder` in a terminal.

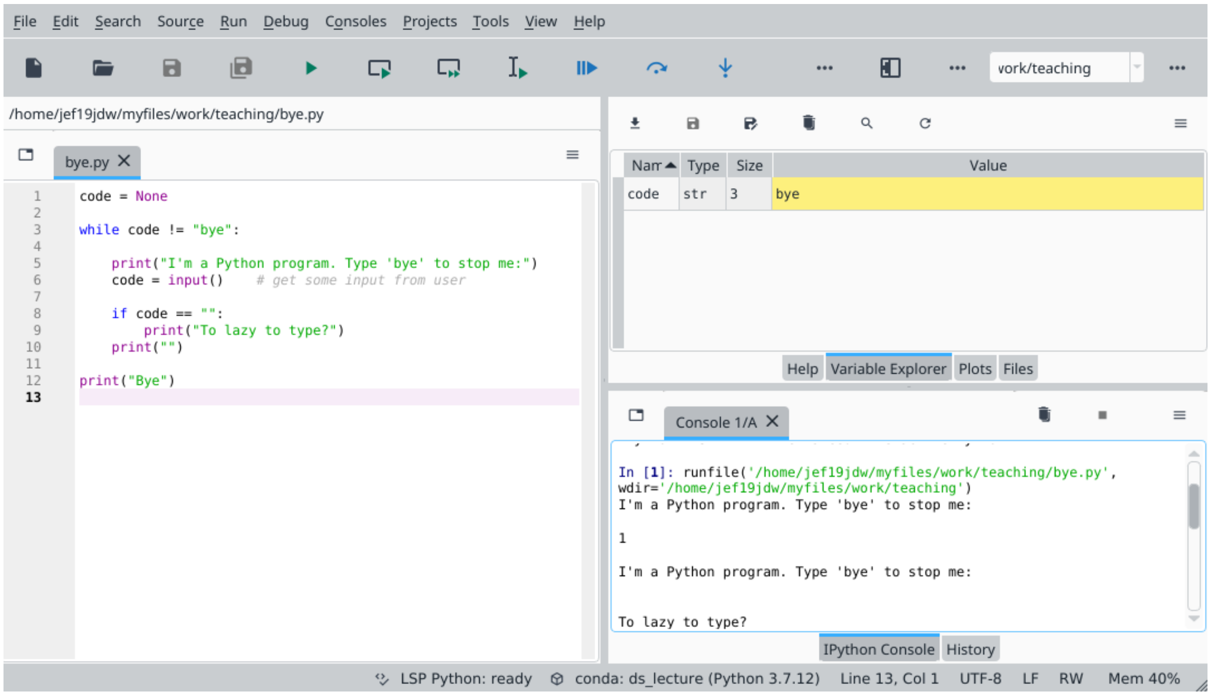
Next to text editor, interactive Python interpreter and separate plotting area Spyder provides tools for debugging (find errors in programs) and for code profiling (measure execution time and memory consumption).

Task: Run Spyder (click button in Anaconda Navigator or type `spyder` in terminal).

Task: Run the Spyder tour (Help > Show Tour).

[675] https://www.spyder-ide.org/

Task: Open `bye.py` in Spyder. Run the program by clicking the 'run file' button in the toolbar (triangle symbol).


File Edit Search Source Run Debug Consoles Projects Tools View Help
vork/teaching
/home/jef19jdw/myfiles/work/teaching/bye.py
bye.py
Help Variable Explorer Plots Files
Console 1/A
IPython Console History
LSP Python: ready
conda: ds_lecture (Python 3.7.12)
Line 13, Col 1
UTF-8
LF
RW
Mem 40%

```
code = None

while code != "bye":

    print("I'm a Python program. Type 'bye' to stop me:")
    code = input()    # get some input from user

    if code == "":
        print("To lazy to type?")
    print("")

print("Bye")
```

Name	Type	Size	Value
code	str	3	bye

```
In [1]: runfile('/home/jef19jdw/myfiles/work/teaching/bye.py',
wdir='/home/jef19jdw/myfiles/work/teaching')
I'm a Python program. Type 'bye' to stop me:

1

I'm a Python program. Type 'bye' to stop me:


To lazy to type?
```

Fig. 58.7: Spyder after running `bye.py`. The variable inspector shows that now there is a variable `code` set in the interactive interpreter.

Task: Use Spyder's interactive Python console for some computations.

58.3.4 Executables

Python code can be bundled together with the interpreter and all required libraries into one executable file. This is not recommended because this file will be relatively large, but it's the only way to provide Python programs to people which have not installed a Python interpreter on their machine.

There are several tools for this job. One is known as PyInstaller and ships with the Anaconda distribution. To convert your Python source code file into an executable file open a terminal and type

```
pyinstaller --onefile filename.py
```

This will create a directory named 'dist' containing the executable.

Hint: Install the `pyinstaller` package via Anaconda Navigator or via `conda install pyinstaller` in a terminal.

Task: Make an executable from

```
code = None

while code != "bye":

    print("I'm a Python program. Type 'bye' to stop me:")
    code = input()    # get some input from user

    if code == "":
```

(continues on next page)

(continued from previous page)

```
        print("To lazy to type?")
    print("")

print("Bye")
```

and run it.

58.4 Long-Running Tasks

On a typical JupyterHub a user's JupyterLab does not stop if the user logs out from the hub. Thus, it's possible to let some Python code (neural network training for instance) run several days without having the notebook open in a webbrowser all the time. In this project we test the workflow for such long running tasks and discuss some caveats.

58.4.1 A Long-Running Task

Task: Put the following Python code into a notebook on a JupyterHub:

```
import time

for i in range(60):
    print(f'iteration {i}')
    time.sleep(1)

print('finished')
```

Save the notebook, and let it run for 5 seconds. Then log out from the hub (without stopping the kernel), wait 5 seconds and log in again. Use a second cell to print a message. Wait until the message appears (may take up to 50 seconds). What do you learn from this experiment?

Solution:

```
# your answers
```

58.4.2 Simple Logging

Task: Now do the same with the following code:

```
import time

with open('log.txt', 'w') as f:

    for i in range(60):
        print(f'iteration {i}')
        f.write(f'iteration {i}\n')
        time.sleep(1)

print('finished')
```

After execution finished, open the file `log.txt`. What do you see?

Solution:

```
# your answers
```

58.4.3 Capturing All Output

Writing log files does not capture output from library code or error messages. Thus, we have to use another approach.

Task: Run the above test procedure with the following code:

```
%%capture cap

import time

for i in range(60):
    print(f'iteration {i}')
    time.sleep(1)

print('finished')
```

When finished, use `cap.show()` to see the captured output.

CHAPTER

FIFTYNINE

PYTHON PROGRAMMING

Programming is like riding a bike. If you want to learn it, you have to do it. Although nowadays there's code available for almost every standard task, implementing simple algorithms like searching and sorting ourselves is very instructive.

59.1 Simple List Algorithms

In this project we implement simple algorithms related to lists like sorting a list or finding special values. The purpose of the projecct is threefold:

- familiarize yourself with Python's syntax,
- learn to algorithmize, that is, how to combine available building blocks to solve a task,
- see and understand how basic algorithms frequently used in data science work.

Before you work through the project you should have read *Building Blocks* (page 56). Restrict yourself to Python features discussed there. Don't use ready-made library functions.

Important: Don't use `list` as name for a variable holding some list, although this would be quite expressive. Several names like `print` and `int` and `list` are already occupied by Python. Python won't complain about reusing some of it's predefined names as variables, but it's considered bad practice.

59.1.1 Maxmium of a List

Given a list of integers we want to find the list's greatest value.

Task: Describe a process (that is, algorithm) to find the maximum value of a list in your words, not Python code. Remember that the list may be of arbitrary length. For simplicity you may assume that the list is not empty.

Solution:

```
# your solution
```

Task: Implement your algorithm in Python. Proceed as follows:

1. Create a function `get_max` which takes a list as argument and, for the moment, always returns the length of the list.
2. Create a sample list, pass it to your function, and print the returned value to the screen.
3. If the framework is working correctly, implement your algorithm in `get_max` and make the function return the maximum value.

4. Test your code with several different sample lists. Include pathological cases like `[1, 1, 1]` and `[1]`.

Solution:

```
# your solution
```

Task: What happens if you test your code with an empty list? Now add some code to your function to check whether the list is empty. If the list is empty `get_max` should print a message and return 0.

Solution:

```
# your solution
```

59.1.2 Mean of a List

Given a list of floats we want to find the list's mean.

Task: Describe an algorithm for calculating the mean of a list. Assume that the list has at least one item.

Solution:

```
# your solution
```

Task: Implement your algorithm in Python. Proceed as follows:

1. Create a function `get_mean` which takes a list as argument and, for the moment, always returns the length of the list.
2. Create a sample list, pass it to your function, and print the returned value to the screen.
3. If the framework is working correctly, implement your algorithm in `get_mean` and make the function return the list's mean.
4. Test your code with several different sample lists. Include obvious cases like `[-1, -1, 1, 1]` and pathological cases like `[1]`.

Solution:

```
# your solution
```

Task: What happens if you test your code with an empty list? Now add some code to your function to check whether the list is empty. If the list is empty `get_mean` should print a message and return 0.

Solution:

```
# your solution
```

59.1.3 Count Values

Given a list of integers we want to count how often a given integer occurs in the list.

Task: Write a function `count_value` taking two arguments (the list and an integer) and returning the number of occurrences of the integer in the list. Proceed step by step as before. How to handle empty lists here?

Solution:

```
# your solution
```

59.1.4 Sorting a List

There exist plenty of algorithms for sorting[676] values in lists. Here we consider selection sort[677] for sorting a list of integers.

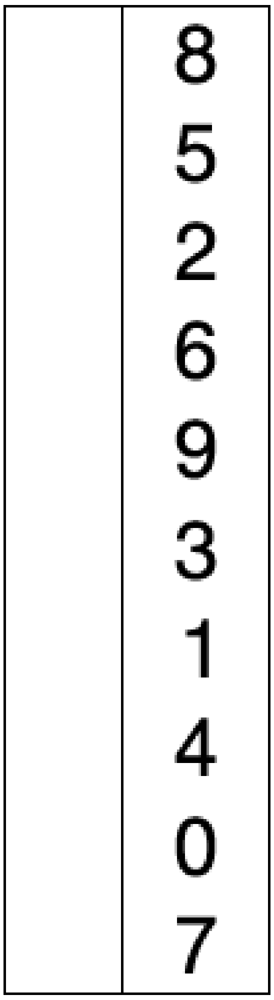


Fig. 59.1: Selection sorts devides the list into two parts: sorted items and unsorted items. It repeatedly walks (blue) through the unsorted items to find the smallest (red) unsorted item. Then it swaps the first unsorted item with the smallest unsorted item. The swapped item then belongs to the sorted part (yellow). Source: Joestape89, wikipedia.org[678], CC BY-SA 3.0[679], modified by the author.

Task: Write a function `sort` taking a list and returning the sorted list. Proceed step by step as before. Don't forget to extensively test your code!

Solution:

```
# your solution
```

[676] https://en.wikipedia.org/wiki/Sorting_algorithm
[677] https://en.wikipedia.org/wiki/Selection_sort
[678] https://en.wikipedia.org/wiki/File:Selection-Sort-Animation.gif
[679] https://creativecommons.org/licenses/by-sa/3.0/deed.en

59.2 Geometric Objects

To get a better feeling for the object-oriented approach to programming we implement classes for geometric objects, like roughly sketched in *Everything is an Object* (page 69). How to draw lines can be guessed from *Library Code* (page 67). Generally, you should have completed the *Crash Course* (page 53) chapter before starting with this project.

We'll need Matplotlib for this project. So we should import it. In principle, imports can be placed anywhere in the code, but it's considered good practice to have them in the first lines of a file.

```
import matplotlib.pyplot as plt
```

59.2.1 Points

Task: Create a class `Point` with two member variables `x` and `y`. To create a `Point` object we want to call `Point(3, 4)`, for instance.

Solution:

```
# your solution
```

59.2.2 Triangles

Task: Create a class `Triangle`. For creating a `Triangle` object we want to pass three `Point` objects.

Solution:

```
# your solution
```

Task: Add a `draw` member function to `Triangle` which draws the triangle on screen using Matplotlib. Keep the whole class definition in one code cell, because class definitions cannot be scattered over multiple cells.

Task: Create four points and draw two triangles between them (two points are used by both triangles). Remember to call `plt.show()` to show the plot on screen. Without `plt.show()` the plot will show up, too, due to some Jupyter magic. In plain Python you won't see the plot.

Solution:

```
# your solution
```

59.2.3 Rectangles

Task: Create a class `Rectangle` representing an paraxial rectangle. For creating a `Rectangle` object we want to pass two `Point` objects, the lower left corner and the upper right corner. Each of the rectangle's four points should be accessible as a member variable.

Solution:

```
# your solution
```

Task: Add a `draw` member function to `Rectangle` (use same code cell as before).

Task: Draw a series of 20 squares centered at the origin and growing from edge length 2 to 40. Call `plt.axis('equal')` before `plt.show()` to make the squares look like squares.

Solution:

```
# your solution
```

59.2.4 Houses

Task: Create a class `House` representing a house made of a rectangular body and a triangular roof. The roof has one third of the body's height and is 20 per cent wider than the body. For creation we want to provide the lower left corner as well as width of the body and total height of the house. Add a `draw` method.

Solution:

```
# your solution
```

Task: Draw three houses.

Solution:

```
# your solution
```

59.2.5 Moving Objects

Task: Add a `move` method to `Point` which takes two numbers and moves the point paraxially by the specified amounts. Then add `move` methods to `Triangle`, `Rectangle` and `House`, which call `Point`'s move method.

Task: Write a function `row_of_houses` for drawing a row of identical houses. Arguments are width and height of the houses as well as start and end coordinates of the row on the x axis. Create one `House` object inside the function. Draw and move the house in a while loop until the house leaves the specified interval.

Solution:

```
# your solution
```

Task: Draw a row of houses.

Solution:

```
# your solution
```

59.3 Vector Multiplication

The aim of this project is to develop a class (object type) for 3-dimensional vectors. The class shall provide typical calculations with vectors as Python operators. Before working through this project you should have read *Variables and Operators* (page 85) chapter. For a recap of vector operations see *Vectors* (page 1021).

This project consolidates your knowledge on defining custom object types and demonstrates the possibilities of Python's flexible approach to customization of operator's behavior.

59.3.1 Class Definition and String Representation

We start with a minimal orking example and then add more functionality step by step.

Task: Create a class `Vector` representing a vector in 3-dimensional space. The `__init__` method takes the 3 components as arguments.

Solution:

```
# your solution
```

Task: Add dunder functions `__str__` and `__repr__` returning human readable representations of a `Vector` object. Test your code by creating and printing a `Vector` object.

Solution:

```
# your test code
```

59.3.2 Addition

When implementing operations on custom objects we have to decide whether to modify an existing object or to create a new object holding the operation's result. For vectors, `a + b` shouldn't modify `a`, but return a new object.

Task: Implement vector addition with `+` operator. The operation should return a new object holding the result. If the second operand is not of type `Vector`, addition should return `NotImplemented`. At this point we do not need `__radd__` because we only accept `Vector` objects for addition. So the left-hand side operand will always be a `Vector` object and Python will call its `__add__` method. Test your code!

Solution:

```
# your test code
```

Task: Modify your implementation of addition to accept lists of length 3, too. Don't forget to implement `__radd__` now. Else list plus vector won't work, because the left-hand side `list` object doesn't know how to work with `Vector` objects. Thus, Python tries to call `__radd__` on the right-hand side operand. As always: test your code!

Solution:

```
# your test code
```

59.3.3 Multiplication by Scalar

Task: Implement multiplication of vectors by scalars via `*`. Both variants, scalar times vector and vector times scalar, should be supported.

Solution:

```
# your test code
```

59.3.4 Inner Products

Now we run out of operators. Of course, we could `*` again for inner products and check types to decide whether we have to do multiplication by scalars or to compute an inner product. The more readable alternative is implementing inner products without an operator.

Task: Implement a method `inner` taking a `Vector` or a `list` object as argument and returning the inner product with the vector whose `inner` method has been called. If the argument neither is a vector nor a 3 element list, return `NotImplemented` although the Python intepreter does not care about (because `inner` is a usual method, not an operator). If you know about Python's exception handling mechanism, you should raise `NotImplementedError` here.

Solution:

```
# your test code
```

59.3.5 Outer Products

Task: In analogy to `inner` implement a method `outer` returning the outer product of two vectors.

Solution:

```
# your test code
```

59.3.6 Equality

At the moment == behaves like `is`. But we want to make == compare vectors componentwise.

Task: Implement equality test via == between two `Vector` objects and a vector and a list. If the second operand is of incorrect type both operands are considered unequal.

Solution:

```
# your test code
```

59.3.7 List-Like Indexing

There's a dunder method `__getitem__` which is called by the Python interpreter whenever indexing syntax `[...]` is applied to an object. The index is passed to the method and the method is expected to return the corresponding item.

Task: Implement `__getitem__`. For invalid indices return `None`. If you know about Python's exception handling mechanism, you should raise `IndexError` here.

Solution:

```
# your test code
```

Task: Make the `len` function work on `Vector` objects. Return value should be 3.

Solution:

```
# your test code
```

CHAPTER SIXTY

WEATHER

We will work through several projects related to weather data and forecasting. Data will be obtained from Deutscher Wetterdienst[680].

60.1 DWD Open Data Portal

Deutscher Wetterdienst (DWD)[681] is Germany's public authority for collecting, managing and publishing weather data from around the world. DWD also creates weather forecasts for Germany and all other regions of the world. Some years ago DWD launched an Open Data Portal[682] and continually extends its services there.

DWD's open data portal provides lots of data and is very complex. In this project we explore part of its structure and locate data sources for subsequent projects.

60.1.1 Licensing

We want to use DWD's data for education and research. Before we delve into the data we should check whether we are allowed to do this and whether and how to attribute the source.

Task: Find out whether we are allowed to use DWD's data for education and research puposes.

Solution:

```
# your answer
```

Task: Do we have to refer to DWD as data provider? If yes, how?

Solution:

```
# your answer
```

[680] https://www.dwd.de
[681] https://www.dwd.de
[682] https://www.dwd.de/opendata

60.1.2 Weather Stations

There are lots of weather stations at Germany collecting weather data. To locate wheather data on the map, we need a list of stations and their geolocations.

Task: Find a list of all DWD weather stations at Germany measuring air temperature at least once per hour. The list should containing geolocations (longitude, latitude, altitude) and other parameters. Get the URL, so we can download it on demand.

Solution:

```
# your answer
```

Task: How many weather stations do we have in the list? List all parameters available for the stations.

Solution:

```
# your answer
```

Task: Get the station list for hourly precepitation measurements. How many stations do we have here?

Solution:

```
# your answer
```

60.1.3 Data

For each weather station we want to have access to all its historical and most recent measurements.

Task: Locate hourly temperature measurements for station Lichtentanne[683]. What's the most recent measurement (timestamp and temperatur)? What's the oldest measurement available?

Solution:

```
# your answer
```

60.1.4 Metadata

Each station comes with extensive metadata telling a story about the station and its measurements.

Task: Answer the following questions from metadata of Lichtentanne station:

- Did the station move? If yes, when? Was it's name changed, too?
- Has measurement equipment been replaced? If yes, when?
- What's the time zone of timestamps in the data?

Solution:

```
# your answer
```

Whenever you find abnormalities in measurements, first check metadata!

[683] https://www.openstreetmap.org/#map=16/50.6879/12.4329

60.2 Getting Forecasts

The Open Data Portal[684] of Deutscher Wetterdienst (DWD)[685] provides detailed forecasts for Germany and all other regions of the world in human and machine readable form. The machine readable service is called MOSMIX[686]. In this project we

- collect information on how to use MOSMIX,
- automatically download newly published MOSMIX data,
- convert MOSMIX files to CSV files.

In this project we heavily rely on techniques presented in *Accessing Data* (page 121).

60.2.1 Investigating and Understanding MOSMIX

DWD's open data portal is quite complex. Before we start downloading forecasts data we have to find information on data location and format.

Task: Read about MOSMIX at DWD's MOSMIX info page[687]. Follow relevant links and answer the following questions:

- What are the differences between MOSMIX S and MOSMIX L?
- What's the URL of the most recent MOSMIX L file for station 'Zwickau'?
- What standard file formats are used for MOSMIX files (KMZ files)?
- How long MOSMIX files are available at DWD's open data portal?

Solution:

```
# your answers
```

60.2.2 An Archive of Forecasts

MOSMIX data older than two days gets removed from DWD's open data portal. To be able to analyze quality of forecasts (that is, to compare them to real observations) we have to keep them in a local archive. For this purpose we would have to visit DWD's open data portal once a day and look for new MOSMIX files. Then we could download them and add them to our local archive. With Python we may automate this job.

Task: Write a function `get_available_mosmix_files` which scrapes a list of URLs of all currently available MOSMIX L files for a selected station from DWD open data portal. Arguments:

- station ID (string).

Return value:

- URLs (list of strings).

Solution:

```
# your solution
```

Now it's time to download the files. Maybe we already downloaded some of them yesterday. So we should have a look in our archive directory first to avoid downloading more files than necessary.

Task: Write a function `download_files` which downloads all new files from a list of URLs. Arguments:

[684] https://www.dwd.de/opendata
[685] https://www.dwd.de
[686] https://www.dwd.de/EN/ourservices/met_application_mosmix/met_application_mosmix.html
[687] https://www.dwd.de/EN/ourservices/met_application_mosmix/met_application_mosmix.html

- URLs (list of strings),
- archive path (string).

Return value:

- names of new files (list of strings).

Hints:

- To check whether a file already exists, have a look at `os.path.isfile`[688].
- Read and write in binary mode because KMZ files aren't text files.

Solution:

```
# your solution
```

60.2.3 KMZ to CSV

Now that we have MOSMIX files in our local storage we should convert them to CSV files. Each row shall contain all weather parameters for a fixed point of time. First column is the time stamp. All other columns contain all the weather parameters contained in the MOSMIX files.

Task: Write a function `kmz_to_csv` for converting a list of KMZ files to CSV files. Arguments:

- archive path (string),
- list of file names (list of strings).

No return value.

Hint: MOSMIX files use an XML feature known as namespaces. Consequently, tag names contain collons, which confuses Beautiful Soup's standard HTML parser (which also parses simple XML files). To get MOSMIX files parsed correctly, install the `lxml` module and provide a second argument `'xml'` to Beautiful Soup's constructor. This tells Beautiful Soup to use a dedicated XML parser, which by default is `lxml`.

Solution:

```
# your solution
```

60.2.4 Automatic Daily Download

To collect forecasts over a longer period of time we have to run the developed code once per day. We could implement a loop and use `time.sleep` to make Python wait one day before continuing with the next run. The better (simpler and more efficient) solution is to tell the operating system to run the Python program each day at a fixed time.

On Linux and macOS there is `cron` (and `anacron`) for scheduling tasks. On Windows there is the *Task Scheduler*.

Task: Find out the details about scheduling a daily task on your system. Then make a Python script file from your code above and let it run once per day.

Solution:

```
# your steps to schedule a task
```

[688] https://docs.python.org/3/library/os.path.html#os.path.isfile

60.3 Climate Change

In this project we download historic weather data from DWD Open Data Portal[689] and have a look at annual mean temperatures and other values at different locations in Germany.

In this project we heavily rely on techniques presented in *Accessing Data* (page 121) and *High-Level Data Management with Pandas* (page 199) as well as on knowledge obtained in the *DWD Open Data Portal* (page 923) project.

We use the DWD data set Historical daily station observations for Germany[690], see description[691].

60.3.1 Station List

The first step is to get a list of all weather stations in Germany.

Task: Download the station list from DWD Open Data Portal[692], make a nice data frame from it, and save it to a CSV file. Columns:

- `'id'` (DWD station ID, use as index, integer),
- `'name'` (string),
- `'latitude'` (float),
- `'longitude'` (float),
- `'altitude'` (integer),
- `'first'` (date of first measurement, timestamp),
- `'last'` (date of last measurement, timestamp).

Hint: `pandas.read_fwf`[693] is your friend.

Solution:

```
# your solution
```

60.3.2 Download Measurements

Task: Get a list of file names of all ZIP files of the data set.

Hint: A good idea is to construct file names from data in the station list (ID, first and last day of measurement). But it turns out that dates in the list in the file names do not coincide for several files. Thus, we have to scrape file names from the data set's file listing[694].

Solution:

```
# your solution
```

Task: Process all files. Processing steps are:

1. Download the file.
2. Read the data file contained in the ZIP file.
3. Drop and rename colums and adjust types (see below).

[689] https://www.dwd.de/opendata
[690] https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/historical
[691] https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/historical/DESCRIPTION_obsgermany_climate_daily_kl_historical_en.pdf
[692] https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/historical/KL_Tageswerte_Beschreibung_Stationen.txt
[693] https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_fwf.html
[694] https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/historical/

4. Write data to a CSV file (one large CSV file for data from all stations).

Columns for CSV file:

- `date` (timestamp of measurement),
- `id` (station ID, integer),
- `'wind_gust'`, `'wind_speed'`, `'precipitation'`, `'sunshine'`, `'snow'`, `'clouds'`, `'pressure'`, `'temperature'`, `'humidity'`, `'max_temp'`, `'min_temp'`, `'min_temp_ground'` (float).

Solution:

```
# your solution
```

60.3.3 Update Station List

Dates of first and last measurements are incorrect in the station list created above. Now, that we have the measurements, we should correct the list.

Task: For each station get dates for first and last measurement and write them to the station list CSV file. Drop all stations that do not have any measurements.

```
# your solution
```

60.3.4 Plots

Task: Use `Series.plot` to create different plots:

- mean annual temperature/precipitation/… for the station with highest number of years with measurements,
- mean annual temperature/precipitation/… in Germany (mean over all stations)
- minimum/maximum temperature in Germany for each year

```
# your solution
```

60.4 Weather Animation

Based on the data collected in the *Climate Change* (page 927) project here we want to create an animation showing air temperature (or other measurements) over a period of time for all weather stations on a map of Germany.

Before you start read *Animations* (page 287).

Following features should be implemented:

- simple (non-interacitve) map of Germany showing at least the borders,
- color-coded temperatures at weather stations (scatter plot),
- one animation frame per day,
- time stamp of currently shown data in each frame.

60.4.1 Prepare Data

For each frame we need following data:

- time stamp string to show during animation
- coordinates for all relevant weather stations,
- temperature for all stations.

Use data collected in *Climate Change* (page 927).

Task: Decide for a time period of several months to show in the animation. Collect required data. Create a list with one item per frame. Each item shall be a dictionary with following keys:

- `'text'` (time stamp string for frame),
- `'x'` (NumPy array with longitudes of all stations to show in the frame),
- `'y'` (NumPy array with latitudes of all stations to show in the frame),
- `'val'` (NumPy array with temperature for all stations).

Save the list with `pickle` to a file.

```
# your solution
```

60.4.2 Visualize Data

Task: Load the data saved above and create a map of Germany with Cartopy. Plot with `GeoAxes.scatter`[695] and use Matplotlib's `animation` module to animate the arrows. Don't forget to show time stamps in the animation. Save the animation to an MP4 video file.

Hint: To avoid troubles with Matplotlib consider the following:

- At the moment Matplotlib does not support blitting when saving animations, but redrawing the whole map for every frame is too slow. Thus, when updating the drawing remove artists not needed anymore by calling the artists `remove`[696] function and then create new artists. This way the background (map) is kept. It's kind of manual blitting.
- If your animation looks like the scatter dots are covered by parts of the map, then pass `zorder=100` (or some other high value) to `scatter`. This tells Matplotlib to draw the scatter object on top of all other objects (if all other objects have lower z-order).

Solution:

```
# your solution
```

[695] https://scitools.org.uk/cartopy/docs/latest/reference/generated/cartopy.mpl.geoaxes.GeoAxes.html#cartopy.mpl.geoaxes.GeoAxes.scatter
[696] https://matplotlib.org/stable/api/_as_gen/matplotlib.artist.Artist.remove.html

CHAPTER

SIXTYONE

MNIST CHARACTER RECOGNITION

A major application of data science and artificial intelligence is recognition of handwritten characters. I a series of projects we will implement different techniques for this task based on the famous MNIST data set (and related data sets) for training recognition systems. MNIST is provided by the National Institute of Standards and Technology[697]

61.1 The xMNIST Family of Data Sets

In the project we have a first look at the MNIST data set and related data sets. In subsequent projects we'll use these data sets for training machine learning models.

A major benefit from the project is, that we see how difficult data preparation can be. As we'll learn later on, obtaining unbiased data is extremely important for training machine learning algorithms.

61.1.1 NIST special database 19

Task: Learn about *NIST special data base 19* from

- NIST Special Database 19, Handprinted Forms and Characters Database[698] (sections 1 and 2)
- NIST Special Database 19[699]

Answer the following questions:

- Who collected the data?

[697] https://www.nist.gov
[698] https://www.nist.gov/system/files/documents/srd/nistsd19.pdf
[699] https://www.nist.gov/srd/nist-special-database-19

- What are the conditions for using the data set?
- Who wrote the characters and digits?
- How many images are in the data set?
- How much disk space is needed?

Solution:

```
# your answers
```

61.1.2 MNIST

Task: Learn about *MNIST data set* from

- Wikipedia[700]
- The MNIST database of handwritten digits[701]

Answer the following questions:

- Who collected the data?
- What are the conditions for using the data set?
- How many images are in the data set?
- What subset of symbols is shown on the images?
- What's the size of the images?
- How much disk space is needed?
- What preprocessing steps were done?
- What's the up to now best error rate for digit recognition based on MNIST?

Solution:

```
# your answers
```

61.1.3 QMNIST

Task: Learn about *QMNIST data set* from

- Cold Case: the Lost MNIST Digits[702]
- QMNIST[703]

Answer the following questions:

- Who collected the data?
- What are the conditions for using the data set?
- How many images are in the data set?
- What's the size of the images?
- How much disk space is needed?
- What preprocessing steps were done?

[700] https://en.wikipedia.org/wiki/MNIST_database
[701] http://yann.lecun.com/exdb/mnist/
[702] https://arxiv.org/pdf/1905.10498.pdf
[703] https://github.com/facebookresearch/qmnist

- Is QMNIST a superset of MNIST?

Solution:

```
# your answers
```

Task: Download the QMNIST data set.

61.1.4 EMNIST

Task: Learn about *EMNIST data set* from

- EMNIST: an extension of MNIST to handwritten letters[704] (section I and subsections A, B, C of section II)
- The EMNIST Dataset[705]

Answer the following questions:

- Who collected the data?
- What are the conditions for using the data set?
- How many images are in the data set?
- What's the size of the images?
- How much disk space is needed?
- What preprocessing steps were done?
- Why EMNIST was created?

Solution:

```
# your answers
```

61.2 Load QMNIST

In this project we develop a Python module for loading and preprocessing QMNIST images and metadata. Prerequisites:

- *Efficient Computations with NumPy* (page 165)
- *The xMNIST Family of Data Sets* (page 931)

61.2.1 Reading Data

Task: Get QMNIST training and test data from QMNIST GitHub repository[706] (4 files ending with `...idx3-ubyte.gz` or `...idx2-int.gz`) and find information on the file format.

Task: Write a function `load` which reads images and metadata from the QMNIST files. Parameters:

- `path`: defaulting to `''`, path of directory with data files.
- `subset`: defaulting to `'train'` (load training data), passing `'test'` loads test data.
- `as_list`: defaulting to `False` (return one large array), passing `True` returns a list of images.

Return values:

[704] https://arxiv.org/pdf/1702.05373.pdf
[705] https://www.nist.gov/itl/products-and-services/emnist-dataset
[706] https://github.com/facebookresearch/qmnist

- NumPy array of shape `(60000, 28, 28)` or list of 60000 NumPy arrays of shape `(28, 28)` (range 0…1, type `float16`), depending on parameter `as_list`.
- NumPy array of shape `(60000, )` containing classes (type `uint8`).
- NumPy array of shape `(60000, )` containing series IDs (type `uint8`).
- NumPy array of shape `(60000, )` containing writer IDs (type `uint16`).

Test your function and show first and last images of training and test data. Print corresponding metainformation. You may use the code from *Image Processing with NumPy* (page 879) to show images.

Hint: Going the obvious path via `zipfile` module and `np.fromfile` fails due to two problems:

1. Python's `zipfile` module has some trouble reading the QMNIST files. Try the `gzip` module[707] from Python's standard library instead.
2. NumPy's `fromfile` is not compatible with file objects created by the `gzip` module. The `fromfile` function will read compressed instead of uncompressed data (for some very knotty technical reasons). Thus, read with the file object's `read` method and use `np.frombuffer`.

Solution:

```
# your solution
```

61.2.2 Preprocessing

Before images can be used preprocessing steps might be appropriate. Given a list of preprocessing steps we would like to have a function which applies all the steps to all images.

Task: Write a function `preprocess` which applies a list of preprocessing steps to all images. Parameters:

- `images`: large NumPy array or list of arrays (images to be processed).
- `steps`: list of functions; each function takes an image and returns an image.
- `as_list`: `False` (default) returns images in large array (and fails if image sizes differ after applying preprocessing steps); `True` returns list of images.

Return values:

- list of processed images or large array of images, depening an parameter `as_list`.

Test your code with two preprocessing steps:

1. horizontal mirrowing,
2. color inversion (black to white, white to black).

Solution:

```
# your solution
```

[707] https://docs.python.org/3/library/gzip.html

61.2.3 Python Module

Task: Create a Python module `qmnist.py` providing both functions `load` and `preprocess`.

Solution:

```
# your solution
```

61.3 QMNIST Feature Reduction

In this project we apply PCA to QMNIST data. Read about *Feature Reduction* (page 343) before you start.

61.3.1 Preprocessing

Task: Use the Python module developed in *Load QMNIST* (page 933) to load the first 5000 QMNIST training images.

Solution:

```
# your solution
```

Task: Use the above module to apply the following preprocessing steps from *Image Processing with NumPy* (page 879)

- auto crop,
- center in 20x20 image

Solution:

```
# your solution
```

61.3.2 Relevant Components

Task: Perform a full PCA (no feature reduction) and plot standard deviations (square roots of variances) for all principal components.

Solution:

```
# your solution
```

Task: Show the data set's mean and the first 100 principal components as images. Scale principal components by corresponding standard deviation and use same color map for all images.

Solution:

```
# your solution
```

Task: Transform all images by PCA with 15 components. For one images plot original and transformed image side by side (you may use `PCA.inverse_transform`[708] or implement calculations manually).

Solution:

```
# your solution
```

[708] https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html#sklearn.decomposition.PCA.inverse_transform

61.3.3 Visualization

We may use feature reduction techniques to visualize high-dimensional data sets.

Task: Use PCA to reduce the data set to 3 features. Plot the now 3d data set. Color classes differently.

```
# your solution
```

Task: Use PCA to reduce the data set to 4 features. Make pair plots for combinations of two features (use same coloring as above).

```
# your solution
```

61.4 PCA and ANN for QMNIST

In this project we train a small ANN for handwritten digit recognition. Using PCA for preprocessing allows to reduce the ANN's size compared to the ANN in *ANNs with Keras* (page 479).

You may reuse code from *Load QMNIST* (page 933) and *Image Processing with NumPy* (page 879).

Task: Solve the QMNIST digit recognition task with a layered feedforward ANN and prior PCA feature reduction to 15 features. Try to get at least 90 percent correct classifications on the test set (without using the test set for training, of course).

Solution:

```
# your solution
```

61.5 CNN for QMNIST

In this project we train a CNN for handwritten digit recognition. Read *Convolutional Neural Networks* (page 498) and *CNNs with Keras* (page 511) before you start.

You may reuse code from *Load QMNIST* (page 933) and *Image Processing with NumPy* (page 879).

Task: Solve the QMNIST digit recognition task with a CNN. Try to get 99 percent correct classifications on the test set (without using the test set for training, of course). Remember that convolutions give more weight to image center. Thus, do not crop QMNIST images. Use 28x28 images with centered bounding boxes. Save your model to a file.

Solution:

```
# your solution
```

61.6 CNN Analysis for QMNIST

In this project we analyse the CNN trained in *CNN for QMNIST* (page 936).

You may reuse code from *What did the CNN learn?* (page 521).

Task: Load the CNN trained and saved in *CNN for QMNIST* (page 936).

Solution:

```
# your solution
```

Task: Visualize filters for the first convolutional layer in your CNN and try to interpret some of them (what features do they detect?).

Solution:

```
# your solution
```

Task: Take an image correctly classified to show a zero and modify it slightly to make you CNN 'think' that it's a five although human eye clearly sees a zero.

Solution:

```
# your solution
```

61.7 IBAN recognition

We aim at recognizing handwritten IBANs[709]. We first train a CNN an QMNIST for detecting single handwritten digits. Then we use the CNN to recognize handwritten IBANs.

Task: Train and evaluate a CNN with log loss and softmax activation for classifying handwritten digits. Use the QMNIST data set for training and testing. Use Keras and Keras-Tuner.

Solution:

```
# your solution
```

61.7.1 Simple IBAN recognition

We have a data set containing 10000 images of IBANs together with corresponding correct IBANs (strings). The data set only contains German IBANs of the form

```
DExxyyyyyyyyyyyyyyyyyy
```

with `xx` being a checksum (see below) and `yyyyyyyyyyyyyyyyyy` being 18 digits (0-9).

For our first attempt we ignore the checksum and try to recognize the IBAN digit by digit.

Each image has size 28x560 (20 images of size 28x28 placed next to each other) and does not contain the letters `DE`. Each 28x28 box contains exactly one 20x20 digit from the QMNIST test set randomly positioned in the box.

Task: Load the IBAN data set. Show an IBAN image and the corresponding correct IBAN.

```
# your solution
```

Task: Write a function `get_iban_simple` which takes an IBAN image and returns the IBAN as string (including `DE`).

```
# your solution
```

Task: Convert all IBAN images to strings and calculate the correct classification rate.

```
# your solution
```

Task: Based on the correct classification rate on the QMNIST test set calculate the probability that an IBAN is correctly recognized.

[709] https://en.wikipedia.org/wiki/International_Bank_Account_Number

```
# your solution
```

Task: Based on the correct classification rate on the QMNIST test set calculate the probability that a recognized IBAN has at most one wrong digit.

```
# your solution
```

Task: Calculate the probability that a recognized IBAN has at most two wrong digits.

```
# your solution
```

61.7.2 IBAN recognition with checksum check

The third and fourth digit of an IBAN is a checksum. The checksum allows to detect common typos (missing digits, interchanged digits, and others).

Task: Find out how to validate IBANs. For instance, have a look at Wikipedia on IBANs[710]. Then write a function `is_iban` which takes an IBAN string and returns `True` or `False` depending on the validity of the IBAN.

```
# your solution
```

If exactly one digit of an IBAN is incorrect, then the check sum check is guaranteed to fail. For two incorrect digits, the check almost always fails.

Task: Write a function `get_iban` which takes an IBAN image and returns the IBAN as string (including `DE`). The returned IBAN should be valid. If the first attempt yields an invalid IBAN use probabilities returned by the model to determine other IBANs. Proceed as follows:

- Generate a list of all IBANs which can be derived from the original one by replacing one or two digits.
- Calculate probabilities for all generated IBANs.
- Sort IBANs by probability.
- Check IBANs starting with the most probable one.

Provide the IBAN's probability as second return value of `get_iban`. Before you start: How many alternative IBANs will be generated in case of an invalid first attempt?

```
# your solution
```

Task: Recognize all IBANs and calculate correct classification rate.

```
# your solution
```

Task: Plot histograms of probabilities for correctly classified and for incorrectly classified IBANs. Use logarithmic binning.

```
# your solution
```

From the histograms we see, that it's not (!) a goog idea to look at the probabilities for deciding whether an IBAN is correctly recognized or not. There are correct IBANs with very small probability and incorrect IBANs with probability close to 1.

To further improve IBAN recognition one could use other checksums described in national IBAN specifications. For German IBANs there are separate checksum for routing number and account number.

Task: Visualize all incorrectly classified IBANs together with true and recognized IBANs.

[710] https://en.wikipedia.org/wiki/International_Bank_Account_Number#Validating_the_IBAN

```
# your solution
```

61.8 SVM for QMNIST

We want to classify QMNIST images with an SVM.

61.8.1 Load Data Set

Task: Load QMNIST training and test data to NumPy arrays as required by Scikit-Learn. Center bounding boxes of all images and crop images to 20x20 pixels.

Solution:

```
# your solution
```

61.8.2 Linear SVM

The data set is relatively small (60000 training samples) compared to the space dimension (400). So there is some chance that classes can be separated by hyperplanes instead of nonlinear hypersurfaces.

Task: Train and evaluate a linear SVM on the QMNIST training samples with Scikit-Learn's `LinearSVC`[711] and again with `SGDClassifier`[712].

Solution:

```
# your solution
```

61.8.3 Nonlinear SVM

To reduce computation time we should apply PCA to our data set. Thus, inner products are computed in, say, $\mathbb{R}^{15}$ instead of in $\mathbb{R}^{400}$.

Task: Apply PCA transform with 15 components to the data. Then train a kernel SVM with rbf kernel.

Solution:

```
# your solution
```

Task: Visualize some support vectors (original images, not PCA coefficients).

Solution:

```
# your solution
```

[711] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[712] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

61.9 Semisupervised Classification

The QMNIST data set contains 120000 labeled images of handwritten digits. It was created to train digit recognition systems via supervised learning methods. Obtaining such a massiv amount of labeled samples is an expensive and time consuming work. The creators designed forms for writing prescribed digits, which were filled in by hundreds of people. Then the forms were scanned and digits separated. Hopefully every writer wrote the correct digit in each field.

In the project we want to train a digit recognition model based an QMNIST images without using any labels. Obtaining scanned images of handwritten digits is cheap and simple. One does not need forms with prescribed digits and nobody has to manually label scanned images. There are lots of pages with handwritten digits out there. We would have to scan them and separate digits by standard image processing routines.

Of course, having only unlabeled data at hand we cannot use supervised learning methods. Unsupervised methods do not yield labels, but only unlabeled clusters of similar images. The obvious idea is to do some unsupervised clustering and then manually label each cluster. Adding (a small amount of) manual labels to an unsupervised learning procedure is known as *semi-supervised learning*.

61.9.1 Preprocessing

Task: Load QMNIST training images (without labels). Center and crop images to 20x20.

Solution:

```
# your solution
```

We have a 400 dimensional data space. This might be too much to obtain useful results from k-means (curse of dimensionality). Maybe we have to use PCA.

Task: Check whether our data suffers from the curse of dimensionality (don't use all samples!). Why isn't this the case?

Solution:

```
# your solution
```

61.9.2 Clustering

Although we know that there are 10 different classes there might be much more clusters. A cluster contains similar samples, but different people tend to write the same digits in several different ways. From this point of view it is not clear how many cluster we can expect.

We have to choose k by elbow method or silhouette score or Davies-Bouldin index. Calculating silhouette scores is very slow for large data sets (why?), so we calculate it only for a subset. We also should use mini-batch k-means to save computation time.

Task: Apply k-means to the data and find the best k based on elbow method, silhouette score (10000 samples) and Davies-Bouldin index. Make a first run with $k = 5, 10, 15, \ldots, 100$. Then choose a smaller intervall for k and run k-means for each k in this interval.

Solution:

```
# your solution
```

Task: Choose a good k and visualize all cluster centers together with cluster sizes (samples per cluster). Write a function for visualizing the cluster centers. The function shall take a NumPy array of cluster centers and a list of title strings as arguments.

Solution:

```
# your solution
```

61.9.3 Prediction

To use our model (set of cluster centers) for recognizing digits we have to label the cluster centers manually. For testing the prediction quality of our model we use QMNIST test images and labels.

Task: Load and preprocess QMNIST test images and labels.

Solution:

```
# your solution
```

Task: Create a mapping (1d array) from cluster labels (indices) to digit labels (manual labeling). Then label all test images and calculate correct classifiaction rate.

Solution:

```
# your solution
```

61.9.4 Inspection

Task: Calculate correct classification rate per cluster and show results together with corresponding cluster centroids.

Solution:

```
# your solution
```

61.9.5 More clusters?

Task: What do you think about using more clusters than suggested by silouette score and Davies-Bouldin-index?

Solution:

```
# your answer
```

Task: Try $k = 100$.

Task: What happens for $k = 60000$.

Solution:

```
# your answer
```

61.9.6 Random cluster centers

k-means with k manually labeled cluster centers is equivalent to 1NN with the cluster centers as training set. The idea behind k-means is that the cluster centers are not chosen at random, but much more sensible.

Task: Try 1NN with 100 randomly choosen and manually labeled training samples. Calculate correct classification rate on the test set.

Solution:

```
# your solution
```

61.10 Generating Handwritten Digits

Gaussian mixture models are generative models, that is, we may use a trained model to generate new samples looking similar to the training samples. If we train a Gaussian mixture model on QMNIST, then it should be possible to generate images of handwritten digits, which are not exactly equal to one of the QMNIST images.

61.10.1 Loading and Preprocessing Data

Task: Load QMNIST training images, labels and writer IDs. Center bounding boxes and crop images to 20x20 pixels.

Solution:

```
# your solution
```

Task: Use PCA with 50 components to reduce dimensionality of the data space. Else training the model will take too long. Show some image together with its projection onto the 50 dimensional subspace constructed by PCA.

Solution:

```
# your solution
```

61.10.2 Training the Model

Task: Fit a Gaussian mixture model to the data. What's a sensible number of clusters.

Solution:

```
# your solution
```

61.10.3 Generating Images

Task: Use GaussianMixture.sample[713] to generate 100 new images showing handwritten digits. Show all images.

Solution:

```
# your solution
```

61.10.4 Gaussian Mixture Unmixed

If we already know the clusters and only want to generate new images, we may fit a Gaussian distribution to each cluster manually. That is, for each cluster we compute mean vector and covariance matrix.

Task: Find the writer with the highest number of images available. Show all images for this writer.

Solution:

```
# your solution
```

Task: Get means and covariances for the ten classes of digitis written by the writer.

Solution:

[713] https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html#sklearn.mixture.GaussianMixture.sample

```
# your solution
```

Task: Use NumPy's random number generator to generate 10 new images per class.

Solution:

```
# your solution
```

61.11 Autoencoder for QMNIST

When clustering QMNIST images with Gaussian mixtures we had to apply PCA to get acceptable computation times. Now we have a nonlinear dimensionality reduction technique at hand which might yield lower quality loss than PCA.

Task: Load QMNIST training images (without further preprocessing).

Solution:

```
# your solution
```

61.11.1 Autoencoder Training

We use the following encoder model:

```
# disable GPU if TensorFlow with GPU causes problems
#import os
#os.environ["CUDA_VISIBLE_DEVICES"] = "-1"

import tensorflow.keras as keras
```

```
encoder = keras.Sequential(name='encoder')

encoder.add(keras.Input(shape=(28, 28, 1)))
encoder.add(keras.layers.Conv2D(4, 3, activation='relu', name='conv1'))
encoder.add(keras.layers.Conv2D(4, 3, activation='relu', name='conv2'))
encoder.add(keras.layers.MaxPooling2D(name='pool1'))
encoder.add(keras.layers.Conv2D(8, 3, activation='relu', name='conv3'))
encoder.add(keras.layers.Conv2D(8, 3, activation='relu', name='conv4'))
encoder.add(keras.layers.MaxPooling2D(name='pool2'))
encoder.add(keras.layers.Conv2D(12, 3, activation='relu', name='conv5'))

encoder.summary()
```

Task: Create a symmetric decoder and the autoencoder model.

Solution:

```
# your solution
```

Task: Train the autoencoder.

Solution:

```
# your solution
```

61.11.2 Evaluation

Code space has 48 dimensions. So we could compare results to PCA with 48 components to see whether the autoencoder yields better results than PCA.

Task: Transform all images with PCA with 48 components. Visualize for some samples original image, autoencoder reconstruction and PCA transformed image. Compute RMSE for autoencoder and PCA results.

Solution:

```
# your solution
```

Visualizing codes provides some information on how well clusters can be separated by looking at the codes.

Task: Load training labels and visualize 300 codes for each digit.

Solution:

```
# your solution
```

61.11.3 Anomaly Detection

Looking at RMSE for each image we may identify images not similar to most other images. Removing such outliers from the data set could increase training success for supervised learning methods.

Task: Calculate RMSE for each image (autoencoder only) and plot images with highest RMSE.

Solution:

```
# your solution
```

61.11.4 Generating New Images

Task: Generate images from random codes.

Solution:

```
# your solution
```

Task: Generate images from random perturbations of a training image's code.

Solution:

```
# your solution
```

Task: Generate a transition from one training image to another training image

- by interpolating images directly and
- by interpolating their codes.

Solution:

```
# your solution
```

Task: Create an animation of the code based transition.

Solution:

```
# your solution
```

61.12 t-SNE for QMNIST

Many projects ago we used PCA to get 2d and 3d visualization of QMNIST training images. Now we have more powerful dimensionality reduction techniques at hand. Let's try t-SNE.

Task: Load QMNIST training images, center bounding boxes, and crop images to 20x20.

Solution:

```
# your solution
```

Task: Use t-SNE to get a 2d visualization of QMNIST images.

Solution:

```
# your solution
```

Task: Load QMNIST training labels and color the 2d plot according to the labels.

Solution:

```
# your solution
```

CHAPTER

SIXTYTWO

CAFETERIA

Have a look at the Zwickau and Chemnitz Universities's menu[714] (cafeterias of both universities are operated by Studentenwerk Chemnitz-Zwickau[715]). In this project we want to scrape as much as possible historic menu data from that website. Read *Accessing Data* (page 121) before you start. Section *Web Access* (page 132) is of particular importance.

62.1 The API

Often web APIs come with some documentation. In our case we neither see an obvious API nor some documentation. Clicking through the menus of past weeks and watching the browser's address bar we see how date and other information is encoded in the URL. This is our key for scraping historic data.

In addition, there is a link an the lower right looking like information about the API. But it turns out, that there is not much API related information, but the useful hint on on XML interface[716] using the same parameter envoding like the HTML interface.

Task: Understand the arguments in the HTML URLs. Then try the XML API from your browser's address bar. Note all location IDs (for 'Mensa Ring' and so on) and the oldest available menu (by trial and error).

Solution:

```
# your answer
```

62.2 Legal Considerations

Have a look at the license information[717]. There we read that it's okay to use the data for our intended purposes.

Remember to not fire too many requests in short time to the server! This may trigger some protection mechanism making the server refuse any communication with us.

- Limit the number of requests per second by pausing your script after each request.
- While developing and testing automatic download limit the total number of requests to a hand full until you're certain that your script works correctly.

[714] https://www.swcz.de/bilderspeiseplan
[715] https://www.swcz.de
[716] https://www.swcz.de/bilderspeiseplan/xml.php
[717] https://www.swcz.de/bilderspeiseplan/lizenz.php

62.3 Getting Raw Data

We proceed in two steps:

- get all the XML files,
- parse all XML files.

Parsing will require lots of trial and error. Thus, first downloading all files and parsing in a second step avoids repeated requests to the server while developing and testing code for parsing.

Task: Write a Python script which downloads menu XML files for all week days and mensa IDs 3 and 4. Write all files into the same directory. Before you start: How many requests will be send to the server? How long will it take if we send two requests per second?

Solution:

```
# your solution
```

62.4 Parsing

Task: From all the downloaded files extract all meals including date, category, description, and prices for students, staff, guests. Save the data to a CSV file.

Solution:

```
# your solution
```

CHAPTER

SIXTYTHREE

PUBLIC TRANSPORT

In this series of projects we visualize and analyze public transport networks based on open data.

63.1 Get Data and Set Up the Environment

In this project we download public transport data and install several Python packages for its processing. Some basic knowledge in Python programming is required for this project.

63.1.1 Download Timetable Data

Timetable data for public transport operators in Germany is available in GTFS format[718].

Task: Go to gtfs.de[719]. Find available GTFS feeds. What types of transport are contained in each feed? What time periods are covered by the data? Are we allowed to use the data?

Solution:

```
# your answers
```

Task: Download all available data from gtfs.de[720]. Note download URLs and terminal commands (if you use the terminal).

Hint: For download via terminal in Linux use

```
curl URL -o DESTINATION_FILE_NAME
```

Solution:

```
# your notes
```

[718] https://en.wikipedia.org/wiki/GTFS
[719] https://www.gtfs.de
[720] https://www.gtfs.de

63.1.2 Download OpenStreetMap Data

To compute walking distances between neighboring public transport stops we'll use data from OpenStreetMap (OSM)[721]. The OSM website provides download of (too) small regions or the whole planet (about 60 GB). Geofabrik GmbH[722] provides regional downloads.

Task: Check OSM licence information. Then download OSM data for Europe in PBF format (Germany is not enough, because GTFS data may contain stops in neighboring countries, if German trains cross borders). Note the download URL and terminal commands.

Solution:

```
# your notes
```

63.1.3 Extract Region of Interest from OSM Data

Extracting walking distances from OSM data requires a lot of memory. Memory consumption grows with size of the region under consideration. Thus, we should extract our region of interest from Europe's OSM file.

Task: Find minimum and maximum latitude and longitude of your region of interest (go to OSM and look at the coordinates of some object on the border of your region of interest).

Solution:

```
# your answer
```

There exist many tools for processing OSM data. A very handy one is Osmosis[723]. You may use it as Python package or in terminal. The terminal command for data extraction is

```
osmosis --rb file=SOURCE_FILE --bb left=... right=... top=... bottom=... --wb␣
↪file=DESTINATION_FILE
```

Task: Extract your region of interest with Osmosis. Note the full terminal command.

Solution:

```
# your notes
```

63.1.4 Conda Environment for GTFS Processing

We want to use the `gtfspy`[724] Python package. It's unmaintained since 2019 (at least). Thus, installation is tricky due to outdated dependencies. But it's a nice package including fast public transport routing. It has been developed for creating A collection of public transport network data sets for 25 cities[725] (also see corresponding GitHub repo[726]).

To avoid messing up your everyday Conda environment with failed installations and broken dependencies create a new Conda environment for this project.

Task: Create a new Conda environment `gtfs`. If working on Gauss[727], don't forget to create a corresponding ipykernel for Jupyter and to switch your notebook's kernel to the new one.

Solution:

[721] https://www.osm.org
[722] http://www.geofabrik.de/
[723] https://wiki.openstreetmap.org/wiki/Osmosis
[724] https://github.com/CxAalto/gtfspy
[725] https://www.nature.com/articles/sdata201889
[726] https://github.com/CxAalto/gtfs_data_pipeline
[727] https://gauss.fh-zwickau.de

```
# your notes
```

63.1.5 Install `osmread`

The `gtfspy` package depends on `osmread`[728] package. But `osmread` isn't available via Conda. Via PyPI (that is, `pip`) we get an older version with outdated (unsatisfyable) dependencies. Thus, we have to install `osmread` from source.

Task: Find out what the following commands do. For each line write a short comment. Then run the commands (works on Linux, macOS and Co.; for Windows minor modifications may be required).

```
conda activate gtfs
pip install argparse lxml protobuf==3.20.1
git clone https://github.com/dezhin/osmread.git
cd osmread
python setup.py install
cd ..
rm -r osmread
```

Solution:

```
# your notes
```

63.1.6 Install `gtfspy`

The `gtfspy` package comes with outdated dependencies and several programming errors. Thus, we install it from source as a local package in our working directory. This way we may easily fix issues when they pop up.

Task: Find out what the following commands do. Why do we need the `mv` commands? For each line write a short comment. Then run the commands (works on Linux, macOS and Co.; for Windows minor modifications may be required).

```
pip install pandas networkx pyshp nose Cython shapely pyproj mopy geoindex geojson␣
 ↪matplotlib-scalebar
git clone https://github.com/CxAalto/gtfspy.git
mv gtfspy gtfspy_gitrepo
mv gtfspy_gitrepo/gtfspy gtfspy
rm -r gtfspy_gitrepo
```

Solution:

```
# your notes
```

63.1.7 Patch `gtfspy`

The `gtfspy` package uses several outdated library functions (mainly from `networkx` package) and contains some programming errors. Some patching is in order…

Task: Implement the modifications listed below and think about why they could be necessary (make short notes).

Solution:

```
# your notes
```

[728] https://github.com/dezhin/osmread

in `gtfspy/osm_tranfer.py`:

- replace (line 91)

  ```
  network_nodes = walk_network.nodes(data="true")
  ```

  by

  ```
  network_nodes = walk_network.nodes(data=True)
  ```

- replace (line 139)

  ```
      walk_network.add_path(way.nodes)
  ```

  by

  ```
      networkx.add_path(walk_network, way.nodes)
  ```

- replace (line 143-145)

  ```
  for node, degree in walk_network.degree().items():
      if degree is 0:
          walk_network.remove_node(node)
  ```

  by

  ```
  nodes_to_remove = []
  good_nodes = networkx.get_node_attributes(walk_network, 'lat').keys()
  for node, degree in walk_network.degree():
      if degree == 0:
          nodes_to_remove.append(node)
      elif node not in good_nodes:
          nodes_to_remove.append(node)
  for node in nodes_to_remove:
      walk_network.remove_node(node)
  ```

  (`good_nodes` contains all nodes with lat/lon data; nodes without data presumably belong to ways crossing the map's border (some nodes dropped by Osmosis, but way not shortened); prevents index errors when computing edge lengths some lines below)

in `gtfspy/networks.py`:

- replace (lines 267-270):

  ```
  events_df.drop('to_seq', 1, inplace=True)
  events_df.drop('shape_id', 1, inplace=True)
  events_df.drop('duration', 1, inplace=True)
  events_df.drop('route_id', 1, inplace=True)
  ```

  by

  ```
  events_df.drop('to_seq', axis=1, inplace=True)
  events_df.drop('shape_id', axis=1, inplace=True)
  events_df.drop('duration', axis=1, inplace=True)
  events_df.drop('route_id', axis=1, inplace=True)
  ```

`gtfspy/routing/node_profile_multiobjective.py` (line 78):

- replace

```
        assert dep_time_index is 0, "first dep_time index should be zero␣
↪(ensuring that all connections are properly handled)"
```

by

```
        assert dep_time_index == 0, "first dep_time index should be zero␣
↪(ensuring that all connections are properly handled)"
```

63.1.8 Create GTFS Data Base

To speed up routing `gtfspy` stores all data in an SQLite[729] data base. That's a usual file with extension `sqlite`. First step in working with `gtfspy` is to create the data base containing all relevant GTFS feeds.

Task: Have look at the `import_gtfs` function in `gtfspy`'s `import_gtfs` module. Use this function to transfer GTFS feeds of interest to you to an SQLite data base.

Solution:

```
# your solution
```

63.1.9 Extract Region from GTFS Data Base

If imported GTFS data covers a much larger region than the region you are interested in, you should filter the created data base by region. Else, routing becomes too expensive (in terms of computation time). The `gtfspy` package provides such filtering, but it's expensive, too. Thus, filtering should only be used if it reduces the data base's size significantly.

Filtering require three steps:

1. Open the data base to filter by creating a `GTFS` object, defined in `gtfspy`'s `gtfs` module.
2. Create a `FilterExtract` object, defined in `gtfspy`'s `filter` module.
3. Call the `FilterExtract` object's `filter` method.

Task: Have look at `gtfspy`'s source to learn how to use the above mentioned objects and functions. Then filter the data base by region (hint: 'buffer zone' in `gtfspy's` source is the region of interest).

Solution:

```
# your solution
```

63.1.10 Add OSM Walking Distances to Data Base

To get more realistic walking times between neighboring stops we may extract walking distances from OpenStreetMap. This step is optional. It requires a lot of memory and computation time, because the whole walk network (all walkable paths and streets) is extracted from the OSM file. Use OSM walking distances for small regions only. Without OSM data Euclidean distance are used.

Task: Have look at `add_walk_distances_to_db_python` in `gtfspy`'s `osm_transfer` module. Then use this function to get OSM walking distances. If your region is too large, have a look at hint below this task.

Solution:

```
# your solution
```

[729] https://www.sqlite.org

Hint: Without OSM walking distances the routing algorithm will complain about missing the key `d_walk` in a dictionary. That's presumably a bug. Workaround: Whenever you use your data base (without OSM distances) for routing, add the following lines to your code:

```
for u, v, data in walk_network.edges(data=True):
    data['d_walk'] = data['d']
```

Here `walk_network` is an object representing the walk network stored in the data base. It will be created as preparative step for routing and then passed to the routing algorithm. Place the code between creation of the walk network and passing the walk network to the routing algorithm.

If you use these two lines of code with OSM distance, OSM distances will be overwritten with Euclidean distances.

63.1.11 Use the Data Base

To use the SQLite data base we have to create a `GTFS` object, definded in `gtfspy`'s `gtfs` module. This object then provides lots of methods for accessing the data.

Task: Have a look at an `GTFS` objects `stops`, `get_min_date`, `get_max_date` methods. Call them to get a list of all stops and the date range covered by the GTFS data.

Solution:

```
# your solution
```

63.2 Find Connections

In this project we generate departure times for all stops in a region of interest for connections to one arrival stop with fixed (latest) arrival time.

The projects uses the `gtfspy` data base created in the *Get Data and Set Up the Environment* (page 949) project. Basic Pandas knowledge is required to solve the tasks (read *Series* (page 200), *Data Frames* (page 210), *Advanced Indexing* (page 219) before you start, *Performance Issues* (page 245) may be of interest, too).

63.2.1 Data Base and Time Frame

Task: Connect to the data base, that is, create a `gtfspy.gtfs.GTFS` object.

Solution:

```
# your solution
```

The routing algorithm of `gtfspy` looks for public transport connections in a user-defined time frame. Start and end time have to be provided in Unix time[730].

Task: Compute Unix times for start and end of your time frame of interest. Use the `GTFS` object's `get_day_start_ut` method to convert a date to it's 00:00 unix time. Then add hours and minutes to this value.

Hint: The Python standard library provides functions for getting Unix times. But `GTFS.get_day_start_ut` takes care of time zone information in the GTFS data.

Solution:

[730] https://en.wikipedia.org/wiki/Unix_time

```
# your solution
```

63.2.2 Arrival Stop

The routing algorithm of `gtfspy` computes public transport connections from all stops in the data base to a user-defined arrival stop. The arrival stop has to be specified by it's GTFS ID (column `'stop_I'` in the data frame returned by `GTFS.stops()`).

Task: Get the stops data frame. Use column `'stop_I'` (GTFS stop ID) as index. Rename the index column to `'id'` and the column `'stop_id'` to `'code'` (the stop's GTFS short name). Drop all columns but `'id'`, `'code'`, `'name'`, `'lat'`, `'lon'`.

Solution:

```
# your solution
```

Task: Write some code to find all stops containing some string (e.g., all stops containing `'Zwickau, Zentrum'`). Use the stops' geolocation and OpenStreetMap to decide for an arrival stop.

Hint: An advanced and very comfortable solution is to generate for each relevant stop a link to OSM (with marker at the stop). Rendering these links as HTML in Jupyter you simply have to click the stops' links to see where they are on the map.

- OSM link with marker: `https://www.osm.org/?mlat=MARKER_LAT&mlon=MARKER_LON`
- HTML rendering for links:

```
import IPython.display
display(IPython.display.HTML('<a href="URL">LINK_TEXT</a>'))
```

Solution:

```
# your solution
```

63.2.3 Routing

The routing API of `gtfspy` is relatively complex and unintuitive. To generate all connections to the arrival stop following steps are necessary:

1. Call `gtfspy.routing.helpers.get_transit_connections`.
2. Call `gtfspy.routing.helpers.get_walk_network(G, max_walk)`.
3. Create a `gtfspy.routing.multi_objective_pseudo_connection_scan_profiler.MultiObjectivePseudoCSAProfiler` object. Pass the results of steps 1 and 2 to the constructor (arguments `transit_events` and `walk_network`).
4. Call the `run` method of the object created in step 3.

Task: Follow the above steps. Have a look at `gtfspy`'s source for available arguments. A good walking speed is `1.5`. With `track_vehicle_legs` and `track_time` you (presumably) can influence whether connections with fewer transfers and lower travel time shall be preferred by the routing algorithm.

Solution:

```
# your solution
```

63.2.4 Best Connection

The `MultiObjectivePseudoCSAProfiler` object now contains information about all connections to the arrival stop in the specified time frame. The `stop_profiles` member variable is subscriptable with allowed indices returned by the `keys` member function. Indices are stop IDs. If `i` is a stop ID, then `stop_profiles[i].get_final_optimal_labels()` returns an iterable object with one item per connection from stop `i` to the arrival stop. Each item has a `departure_time` member containing the departure time of the connection in Unix time.

Task: Add a column to your stops data frame, which contains the difference between latest allowed arrival time and latest possible departure time from the considered stop in minutes. For stops without connection to the arrival stop use `-1`.

Solution:

```
# your solution
```

63.2.5 Grouping Stops

In the stops data frame most stops appear multiple times, e.g., each platform of a station has its own item in the data frame. For visualization nearby stops should be merged to one stop. The `GTFS` object's `get_stops_within_distance` method yields a data frame of nearby stops. The first argument is the considered stop's ID, the second argument is the distance in meters.

Task: Think about an algorithm for grouping stops and implement it. Add a column to your stops data frame, which contains a group ID for each stop. All stops with identical group ID are considered one and the same stop (in the visualization to create in a follow-up project).

Solution:

```
# your solution
```

Task: How many stop groups do you have? What's the largest group? Show all its stops.

Solution:

```
# your solution
```

63.2.6 Save Results

Task: Save your stops data frame to a CSV file.

Solution:

```
# your solution
```

63.3 Interactive Map

In this project we visualize the results of the *Find Connections* (page 954) project on an interactive map. Have a look at *Folium* (page 305) before you start.

63.3.1 Prepare Data

Task: Load the CSV file created in the *Find Connections* (page 954) project to a Pandas data frame.

Solution:

```
# your solution
```

Task: Create a data frame containing only one stop per group. In each group choose the stop with shortest travel time. Drop groups without any stops connected to the arrival stop.

Solution:

```
# your solution
```

63.3.2 Create Map

Task: Create Folium map centered at the analyzed region with a marker (with stop name in tooltip) at the arrival stop.

Solution:

```
# your solution
```

63.3.3 Add Departure Stops

Task: For each stop add a marker to the map showing stop name and departure time in a tooltip. Show only one stop per stop group, the one with the lastest departure.

Hint: Use `folium.plugins.FastMarkerCluster`[731] instead of `folium.plugins.MarkerCluster`[732], because the latter may be very slow if you have many stops. Because `FastMarkerCluster` generates markers dynamically only for the currently visible area of the map, tooltips have to be generated dynamically, too. For tooltip generation pass the following JavaScript function as string to `FastMarkerClusters` `callback` argument:

```
function (row) {
    var marker;
    marker = L.marker(new L.LatLng(row[0], row[1])).bindTooltip(row[2]);
    return marker;
};
```

The `data` argument then expects a list of tuples `(latitude, longitude, tooltip_text)` describing the markers.

Solution:

```
# your solution
```

Task: Marker clusters are colored depending on their size. We want to have constant color for all clusters. Thus, inject to following HTML snipped into your map:

[731] https://python-visualization.github.io/folium/plugins.html#folium.plugins.FastMarkerCluster
[732] https://python-visualization.github.io/folium/plugins.html#folium.plugins.MarkerCluster

```
<style>
.marker-cluster-small div, .marker-cluster-medium div, .marker-cluster-large div {
    background-color: #0000ff80;
}
.marker-cluster-small, .marker-cluster-medium, .marker-cluster-large {
    background-color: #0000ff30;
}
</style>
```

Solution:

```
# your solution
```

63.3.4 Color-Coded Distances

To visualize travel times we may color each point of the map depending on the distance to the next stop and on the next stop's departure time to the arrival stop. Color scheme is as follows:

- Circular areas with radius 1 kilometer around stops with connection to the arrival stop get colored. All other regions of the map remain uncolored.
- Color around a stop depends on departure time: green for late departure, yellow for medium departure time, red for early departure (continuous color scale).

One possible path to follow is:

1. Choose a rectangular region of interest on the map and divide it into a grid of rectangles (cells). Edge length should be about 100 meters.
2. For each stop identify the cell containing the stop.
3. For each cell get the latest departure time.
4. Interpret the grid of cells as 'image' and 'draw' a 1-kilometer disc of color 'departure time' around each cell. Initialize the image with some invalid value, then draw discs with increasing departure time (thus, late departures will overwrite early departures).
5. Add an `ImageOverlay`[733] to your map color coding the image of departure times.

Task: Implement the above steps or follow an alternative path for color-coding departure times.

Solution:

```
# your solution
```

[733] https://python-visualization.github.io/folium/modules.html#folium.raster_layers.ImageOverlay

CHAPTER

SIXTYFOUR

CORONA DEATHS

In this project we collect and/or compute death rates before and during the Corona pandemic in Germany. You should read *Dates and Times* (page 229) before you start.

64.1 Get some Data

We would like to have *monthly* death rates for an as long as possible period of time including very recent data.

Task: Download relevant data from Federal Statistical Office (Statistisches Bundesamt)[734] in CSV format:

- Destatis, table 12613-0006[735]
- Destatis, table 12411-0001[736]
- Destatis, Sonderreihe mit Beiträgen für das Gebiet der ehemaligen DDR, Heft 3[737]
- Destatis, table 12411-0020[738]

For each file write a short note on its content.

Solution:

```
# your notes
```

Task: Use your favorit spreadsheet tool to compile following CSV files from the downloaded files:

- `inhabitants-yearly.csv` with columns `year`, `FRG` (inhabitants FRG), `GDR` (inhabitants GDR, 0 from 1990 on)
- `inhabitants-quarterly.csv` with colums `date`, `inhabitants`
- `deaths-monthly.csv` with columns `year`, `months` (numeric 1…12), `men`, `women`

64.2 Load Data

We want to use dates as index for data frames. Numbers of inhabitants are related to precise timestamps (end of year or quarter). Numbers of deaths are related to periods (month).

Task: Read in the three CSV files. Use `DatetimeIndex` and `PeriodIndex` for data frames and series. In the end you should have two series:

- `inhabitants` with index `date` (timestamp of last day in year or quarter),
- `deaths` with index `date` (monthly period aligned at last day of month).

[734] https://www.destatis.de
[735] https://www-genesis.destatis.de/genesis//online?operation=table&code=12613-0006
[736] https://www-genesis.destatis.de/genesis//online?operation=table&code=12411-0001
[737] https://www.statistischebibliothek.de/mir/servlets/MCRFileNodeServlet/DEMonografie_derivate_00000961/Heft_3.pdf
[738] https://www-genesis.destatis.de/genesis//online?operation=table&code=12411-0020

Solution:

```
# your solution
```

64.3 Death Rates

For calculating monthly death rates we have to get the number of inhabitants on a monthly basis, i.e., the mean number of inhabitants per month. If we would have daily values for the number of inhabitants we could simply calculate the mean. But resolution is much coarser. Thus, we have to use (linear) interpolation. A good replacement for the monthly mean is the (interpolated) value at the 15th of the month.

Task: Use resampling to get interpolated number of inhabitants at the 15th of each month. From these values construct a series with period index (in analogy to the `deaths` series' index). Hint: instead of (integer) index based linear interpolation you may want to use timestamp based interpolation (see docs).

```
# your solution
```

Task: Calculate monthly death rates and plot results with `Series.plot()`.

```
# your solution
```

CHAPTER

SIXTYFIVE

CHEMNITZ TREES

The aim of this project is to create an information sheet about public trees at Chemnitz. Before you start, you should have read *Matplotlib Basics* (page 251).

65.1 Download and Cleaning

Information on public trees in Chemnitz are available online: Chemnitz trees data set[739].

Task: Find license information. Are we allowed to create an information sheet from the data set and to publish this information sheet?

Solution:

```
# your answers
```

Task: Download the data set in CSV format and read it into a data frame. Explore the data set (columns, data types, numerical ranges, row count,…) and apply standard cleaning steps as appropriate (adjust types, rename columns, drop useless columns,…).

Solution:

```
# your solution
```

65.2 Short Names

To get information about tree type distribution we have to unify tree names. Instead of full detailed names we want to have common short names (*Linde* instead of *Sommerlinde*, *Ahorn* instead of *Bergahorn*, *Flieder* instead of *Syringa reticulata Ivory Pink*,…).

Task: Add a column with common short names for all trees. There are many different ways to automatically derive short names. A good idea is to define a dictionary assigning short species names to search strings. Then full species names can be searched for those strings and, if there is a match, corresponding short names can be assigned. Find short names for all (!) trees. Use 'sonstige' for trees without species name in the data set.

Solution:

```
# your solution
```

[739] https://portal-chemnitz.opendata.arcgis.com/datasets/baeume

65.3 Extract Information

Task: Get the following information from the data set:

- five oldest trees,
- list of rare species (less than 5 trees),
- list of dominant species (at least 1000 trees).

Create a pie chart[740] showing the fraction of total population for each dominant species. Include one slice for all non-dominant trees.

Create a stacked bar plot[741] showing fractions for dominant species by age. Group ages by decade. The horizontal axis shows age in decades starting with 0 (for decade 2020 till 2029) at the right. Vertical axis shows fractions ('linear pie chart').

Solution:

```
# your solution
```

65.4 Presentation

Task: Create PDF file in A4 format showing all information extracted above. Use whatever software you like. LibreOffice Write[742] is a good starting point.

Pimp your pie and bar plots. Format lists of oldest and rarest trees nicely. Add some visual elements (lines, boxes,…) to structure the document and guide the viewer's eyes.

Feel free to add further information. For instance, try to find locations of old and rare trees.

[740] https://matplotlib.org/stable/gallery/pie_and_polar_charts/pie_features.html#sphx-glr-gallery-pie-and-polar-charts-pie-features-py
[741] https://matplotlib.org/stable/gallery/lines_bars_and_markers/bar_stacked.html
[742] https://www.libreoffice.org

CHAPTER

SIXTYSIX

FORGED BANKNOTES

In this series of projects we apply several machine learning concepts and methods to automatically identify forged banknotes.

66.1 Detecting Forgery with k-NN

Banknotes have lots of security features, some well known (see Deutsche Bundesbank[743]) and some less known like the EURion constellation[744]. Machine learning methods allow to investigate features not designed for human vision or simple algorithmic evaluation.

One approach is to have a closer look at the printing quality. Banknotes are printed using a technique know as intaglio[745]. That technique produces extremely sharp edges if steel plates[746] are used and cannot be realized with off-the-shelf machines. Researchers from Institut für industrielle Informationstechnik[747] at Technischen Hochschule Ostwestfalen-Lippe[748] created a data set for training banknote authentication systems based on scanned images of banknotes.

66.1.1 The Data Set

The data set[749] is available from UCI Machine Learning Repository[750]. It comes as a simple CSV file without header.

Task: Download the data set and read it into a Pandas data frame. Get column names from UCI webpage of the data set. Adjust data types if necessary. Look at this blog post by James D. McCaffrey[751] to find out how to interpret class labels.

Solution:

[743] https://www.bundesbank.de/de/aufgaben/bargeld/falschgeld/falschgelderkennung/50-euro-europa-serie
[744] https://en.wikipedia.org/wiki/EURion_constellation
[745] https://en.wikipedia.org/wiki/Intaglio_(printmaking)
[746] https://en.wikipedia.org/wiki/Steel_engraving
[747] https://www.init-owl.de/
[748] https://www.th-owl.de
[749] https://archive-beta.ics.uci.edu/dataset/267/banknote+authentication
[750] https://archive-beta.ics.uci.edu/
[751] https://jamesmccaffrey.wordpress.com/2020/08/18/in-the-banknote-authentication-dataset-class-0-is-genuine-authentic/

```
# your solution
```

Task: Determine class sizes.

```
# your solution
```

The data set does not contain scanned images, but some statistical information about the histograms.

Task: Read sections 1 and 2 of Banknote Authentication[752] to get a rough idea of what the features in the data set express. Note that wavelet transforms are similar to Fourier transforms.

```
# your notes
```

66.1.2 Visualization

Task: Create a pairplot of the 4 features with different colors for the two classes.

```
# your solution
```

Task: For each combination of 3 features create a 3d scatter plot (again different colors for classes).

```
# your solution
```

Note: From the 3d plots we see that variance, skewness, curtosis should suffice to separate forged from genuine banknotes. From this point of view the entropy feature can be neglected. Further, the data set description does not contain information on how the entropy was calculated. If we want to use a model trained on the data set to classify new samples, we do not know how to derive model inputs from scanned images. This is a second reason to drop the entropy feature.

66.1.3 k-NN Predictions

Task: Create model inputs and outputs in Scikit-Learn format. That is, create a NumPy array `X` with one row per sample and one column per feature (do not include entropy) and a one-dimensional NumPy array with outputs for all samples (use integers, no booleans).

```
# your solution
```

Task: Create a k-NN classifier with Scikit-Learn. For the moment we do not consider hyperparameter optimization. Choose $k = 11$ and no weighting. Test set size should be 30 per cent. Compute accuracy on test and training sets. Get the number of missclassified samples in the test set.

Solution:

```
# your solution
```

[752] https://www.researchgate.net/profile/Eugen-Gillich-2/publication/266673146_Banknote_Authentication/links/5436b8140cf2643ab9887bca/Banknote-Authentication.pdf

66.2 Quality Measures

In (projects:forged-banknotes:knn) we saw that k-NN works quite good for detecting forged banknotes from three features. Now we want to compute and interpret several quality measures. To get more wrong predictions (and thus more instructive data to play with) we reduce the number of features to two: variance and skewness.

66.2.1 Predictions

Task: Apply k-NN with $k = 11$ and without weighting to the banknotes classification problem with only two features (variance and skewness). Use test size 0.3. Print accuracy on train and test sets.

Solution:

```
# your solution
```

Task: Use `predict_proba`[753] to get a vector of predicted probabilities for class 1 on the test set. For k-NN classification probabilities represent class distribution in a sample's neighborhood.

Solution:

```
# your solution
```

66.2.2 Quality Measures

Confusion Matrix

Task: Generate and print the confusion matrix for the test set. Compare results to Scikit-Learn's `confusion_matrix`[754].

```
# your solution
```

Accuracy and Balanced Accuracy

Task: Calculate unbalanced and balanced accuracy on the test set. Compare results to Scikit-Learn's `accuracy_score`[755] and `balanced_accuracy_score`[756].

Solution:

```
# your solution
```

[753] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.predict_proba
[754] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
[755] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
[756] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html

Precision, Recall, F1-score

Task: Calculate and print precision, recall and F1-score on the test set. Compare results to Scikit-Learn's `precision_score`[757], `recall_score`[758], `f1_score`[759].

Solution:

```
# your solution
```

Task: Think about precision and recall from a practical point of view. What do models with low or high values for precision or recall imply for banknote authentication?

Solution:

```
# your answers
```

Log Loss

Task: Calculate and print the log loss on the test set. Compare results to Scikit-Learn's `log_loss`[760].

```
# your solution
```

Task: What is the log loss for perfect predictions (with your implementation and with Scikit-Learn)?

```
# your solution
```

AUC

Task: Calculate and plot false positive rate as well as true positive rate for the test set depending on the threshold t, where a sample is labeled 'positive' if the predicted probability is strictly greater than t. Remember that both functions are step functions. Use Matplotlib's `step`[761]. Compare results to Scikit-Learn's `roc_curve`[762].

```
# your solution
```

Task: Plot the ROC curve for the test set. Compare results to Scikit-Learn's `RocCurveDisplay.from_predictions`[763].

```
# your solution
```

Task: Calculate and print AUC for the test set. Compare results to Scikit-Learn's `roc_auc_score`[764].

```
# your solution
```

[757] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
[758] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
[759] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
[760] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html
[761] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.step.html
[762] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
[763] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html#sklearn.metrics.RocCurveDisplay.from_predictions
[764] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

66.3 Hyperparameter Optimization

In this project we revisit *Detecting Forgery with k-NN* (page 963) and add hyperparameter optimization.

66.3.1 Load Data Set

Task: Load the banknotes data set (cf. *Detecting Forgery with k-NN* (page 963)). Drop the entropy column.

```
# your solution
```

66.3.2 Grid Search with Cross Validation

Task: Create a k-NN model with Scikit-Learn's `KNeighborsClassifier`[765]. Use hyperparameter optimization based on accuracy for choosing k and to find appropriate weights (uniform or inverse distance). Evaluate the model on a test set. Print optimal parameters and accuracy on the test set.

Solution:

```
# your solution
```

66.3.3 Decision Surface

Task: Train a second model based on variance and skewness only. Plot the decision surface (the surface separating the classes) with Matplotlib's `contour`[766] or `contourf`[767].

Solution:

```
# your solution
```

66.4 Decision Tree

In this project we solve the banknote classification task with a decision tree.

66.4.1 Load Data Set

Task: Load the banknotes data set (cf. *Detecting Forgery with k-NN* (page 963)). Drop the entropy column.

```
# your solution
```

[765] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[766] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.contour.html
[767] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.contourf.html

66.4.2 Decision Tree

Task: Train and evaluate a decision tree for banknote classification. Use cost-complexity pruning with corresponding parameter chosen by hyperparameter optimization.

Solution:

```
# your solution
```

66.4.3 Decision Surface

Task: Train a second model based on variance and skewness only. Plot the decision surface (the surface separating the classes) with Matplotlib's `contour`[768] or `contourf`[769].

Solution:

```
# your solution
```

66.5 Random Forest

In this project we solve the banknote classification task with a random forest.

66.5.1 Load Data Set

Task: Load the banknotes data set (cf. *Detecting Forgery with k-NN* (page 963)). Drop the entropy column.

```
# your solution
```

66.5.2 Random Forest

Task: Train and evaluate a random forest with 50 trees. Use fixed depth to restrict tree sizes. Choose the depth by hyperparameter optimization.

Solution:

```
# your solution
```

66.5.3 Decision Surface

Task: Train a second forest based on variance and skewness only. Plot the decision surface (the surface separating the classes) with Matplotlib's `contour`[770] or `contourf`[771].

Solution:

```
# your solution
```

[768] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.contour.html
[769] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.contourf.html
[770] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.contour.html
[771] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.contourf.html

66.6 Support-Vector Machine

We want to train a kernel SVM for banknote authentication. Without kernel the decision surface will be a hyperplane more or less identical to the hyperplane obtained from logistic regression. Separation by a hyperplane works, but for both classes there are samples very close to the hyperplane. Looking for nonlinear separation should yield fewer ambiguous samples.

66.6.1 Load Data Set

Task: Load the banknotes data set (cf. *Detecting Forgery with k-NN* (page 963)). Drop the entropy column.

```
# your solution
```

66.6.2 SVM

Task: Use Scikit-Learns's `SVC`[772] to create a model for banknote authentication. Try different kernels (RBF, polynomials of different degrees). Use hyperparameter optimization for choosing the parameter `C`.

Solution:

```
# your solution
```

66.6.3 Decision Surface

Task: Train a second model based on variance and skewness only. Plot the decision surface (the surface separating the classes).Highlight all support vectors.

Solution:

```
# your solution
```

66.7 Naive Bayes Classification

66.7.1 Load Data Set

Task: Load the banknotes data set (cf. *Detecting Forgery with k-NN* (page 963)). Drop the entropy column.

```
# your solution
```

[772] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

66.7.2 Naive Bayes

Task: Use Scikit-Learns's `GaussianNB`[773] to create a model for banknote authentication.

Solution:

```
# your solution
```

66.7.3 Decision Surface and Distributions

Task: Train a second model which only uses variance and skewness. Plot the training data set and the decision curve as well as the two Gaussian densities estimated from the training samples by the naive Bayes classifier.

Solution:

```
# your solution
```

[773] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

CHAPTER

SIXTYSEVEN

HOUSE PRICES

This series of projects extends the results obtained in *Worked Example: House Prices I* (page 388) and *Worked Example: House Prices II* (page 421).

67.1 House Prices GUI

The aim of this project is to create a graphical user interface (GUI) for house price predictions based on the model trained in *Worked Example: House Prices II* (page 421).

67.1.1 The Model

In *Worked Example: House Prices II* (page 421) we saw that not all features or of importance for price prediction. Thus, here we restrict our attention to the important features only.

Task: Train a Ridge regression model for house price prediction based on following features:

- region's average land prices and income,
- lot size and living space,
- build and renovation year
- number of rooms and bath rooms,
- building type (duplex, villa, other).

Find optimal hyperparameters.

Solution:

```
# your solution
```

Task: Train the model with optimal hyperparameters on the full data set. Save the trained model to a file (use `pickle` module).

Solution:

```
# your solution
```

67.1.2 GUI

We use the `ipywidgets`[774] module to create a GUI in Jupyter Lab. Have a look at the following documentation pages:

- Simple Widget Introduction[775]
- Text[776]
- Combobox[777]
- RadioButtons[778]
- Button[779]
- Output[780]

Task: Load the model saved above. Create all required input fields and a button which starts the prediction process. After clicking the button the user should see the predicted house price (in an output widget). To simplify input of the region's land prices and income you could use a combobox for selecting a region.

Solution:

```
# your solution
```

67.2 House Prices ANN

In this project we solve the house price prediction problem from *Worked Example: House Prices II* (page 421) with an ANN. Read *ANN Basics* (page 453) and *Training ANNs* (page 462) before your start.

67.2.1 The ANN

In *Worked Example: House Prices II* (page 421) we saw that not all features or of importance for price prediction. Thus, here we restrict our attention to the important features only.

Task: Train an ANN for house price prediction based on following features:

- region's average land prices and income,
- lot size and living space,
- build and renovation year
- number of rooms and bath rooms,
- building type (duplex, villa, other).

Apply regularization (parameter choice via Scikit-Learn's hyperparameter optimization methods) and try different ANN sizes (number of layers, number of neurons per layer) as well as different activation functions.

Solution:

```
# your solution
```

[774] https://ipywidgets.readthedocs.io/en/stable/
[775] https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20Basics.html
[776] https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html#text
[777] https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html#combobox
[778] https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html#radiobuttons
[779] https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html#button
[780] https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20List.html#output

67.2.2 Some Theory

Task: Think about implications of very small batch sizes in case of noisy data. Are small batches good or bad here?

Solution:

```
# your notes
```

67.3 A Random Forest for House Prices

In this project we solve the house price prediction problem from *Worked Example: House Prices II* (page 421) with a random forest. Read *Bagging* (page 600) before you start.

Task: Use the extended and preprocessed German housing data set to predict house prices. Train a random forest regressor with Scikit-Learn. Try to get similar or better prediction quality as for Ridge regression.

```
# your solution
```

Task: Show feature importances based on `RandomForestRegressor.feature_importances_`.

```
# your solution
```

Task: Visualize one of the trees in the forest. Only show the first few depth levels.

```
# your solution
```

67.4 House Prices SOM

We already used regression techniques to predict house prices. Now it's time to visualize the underlying data set using SOMs.

Task: Load the preprocessed German housing data set generated in *Worked Example: House Prices I* (page 388). Convert categorical features to numeric features. For one hot encoding use as many code features as there are categories (do not drop one of them). Why?

```
# your solution
```

Task: Create a list of feature names, which we will use below to label plots. Create a NumPy array holding the data set.

```
# your solution
```

Task: Train a SOM on the data. Don't forget to standardize data.

```
# your solution
```

Task: Plot the U-matrix.

```
# your solution
```

Task: Visualize each (high dimensional) feature in 2d. Arrange all plots in a 4 by 6 grid.

```
# your solution
```

Task: Create a new sample and get its position (best matching unit) in the SOM. Show corresponding activation map and mark the best matching unit in the map.

```
# your solution
```

CHAPTER

SIXTYEIGHT

HYPERPARAMETER OPTIMIZATION FOR CATS AND DOGS

In *CNNs with Keras* (page 511) we trained a CNN for classifying images of cats and dogs. Using the knownledge from *ANNs with Keras* (page 479) abuot hyperparameter optimization we may improve prediction quality.

Task: Find a CNN for classifying cats and dogs using hyperparameter optimization for Keras. Do not use data augmentation or pre-trained ANNs. Try to get at least 85 percent classification accuracy.

Solution:

```
# your answer
```

CHAPTER

SIXTYNINE

BLOGS

In this series of projects we analyze a large corpus of blogs from the world-wide web.

- *Blog Author Classification (Training)* (page 977)
- *Blog Author Classification (Test)* (page 980)

69.1 Blog Author Classification (Training)

Instead of deriving author information from single blog posts like in *Text Classification* (page 627) we want to use all posts of a blog to derive age, gender and industry of the author from text data. We train three independent models for the three output variables.

Working with text data requires heavy preprocessing. If we want to apply a machine learning model to new data (see project *Blog Author Classification (Test)* (page 980)), we have to preprocess the new data in the same way as training data. This means that not only the model has to be saved for later use, but also the parameters of all preprocessing steps have to be accessible to the user of the trained model. This issue will be addressed in this project, too.

69.1.1 Getting the Data

We have to load blog author data and posts. All posts of a blog have to be joined to one long text.

Task: Load blog author data.

Solution:

```
# your solution
```

Task: Load the lemmatized blog posts. We only need blog IDs and lemmatized texts.

Solution:

```
# your solution
```

Task: Join all posts of one blog to a long string. Add a new column `text` to the blogs data frame containing blog texts. Then remove the posts data frame from memory to free some 100 MB of memory.

Solution:

```
# your solution
```

69.1.2 Model Inputs and Outputs

Gender, age and industry values have to be converted to integers. Conversion rules will be needed again for getting human readable outputs from our model. Thus, we should create some data structure holding the conversion rules. If we use integers starting from 0, 1, 2,… lists do the job. For `unknown` industry we should use the highest integer, because samples with unknown industry will be excluded from training the industry model.

Task: Convert gender (2 classes), age (3 classes) and industry (many classes) to integer values. Create 3 lists for converting integers to human readable strings.

```
# your solution
```

Task: Create a NumPy array with all outputs (3 columns).

Solution:

```
# your solution
```

Task: Create a NumPy array with all texts (1 column of type `object`).

Solution:

```
# your solution
```

69.1.3 Train-Test Split

Task: Split the data set into training (80 per cent) and test sets (20 per cent).

Solution:

```
# your solution
```

For training the industry model we will drop all samples with unknown industry. Here we have to take care, that this removel has similar influence on training and test sets. Else we would have to first remove the samples and then split the data, which would yield more complicated code than one split for all three models.

Task: Check that `unknown` industry got equally distributed to training and test sets.

Solution:

```
# your solution
```

69.1.4 Text to Numbers

Task: Use Scikit-Learn's `TFidfVectorizer`[781] to convert text data to numerical data.

Solution:

```
# your solution
```

We have to save the mapping from words to numbers if we want to use some model trained on the preprocessed data. The vocabulary (maps words to indices) is accessible through `tfidf_vect.vocabulary_` and can be passed to a fresh `TfidfVectorizer` object via the `vocabulary` argument. But vectorization also requires knowledge of the inverse document frequencies. These are accessible through `tfidf_vect.idf_`, but there is no way to pass them to a fresh `TfidfVectorizer` object. Thus, we have to save the whole object.

Task: Save the three lists with human readable labels and the vectorizer object to a file. Use the `pickle` module.

[781] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Solution:

```
# your solution
```

69.1.5 Gender Model

Task: Train and evaluate a multinomial naive Bayes classifier for predicting blog authors' gender with Scikit-Learn.

Solution:

```
# your solution
```

Task: Try a linear SVM for gender prediction.

Solution:

```
# your solution
```

69.1.6 Age Model

Task: Train naive Bayes and SVM models for age prediction.

Solution:

```
# your solution
```

69.1.7 Industry Model

Task: Select all training and test samples with kown industry.

Solution:

```
# your solution
```

Task: Train naive Bayes and SVM models for industry prediction.

Solution:

```
# your solution
```

Task: Calculate the accuracy for a model which always predicts 'Student' as industry.

Solution:

```
# your solution
```

69.1.8 Saving Models

Scikit-Learn does not provide functions for saving trained models (in contrast to Keras). But pickling Scikit-Learn objects should work.

Task: Save the three SVM models to a file.

Solution:

```
# your solution
```

Task: Why is the file containing three SVM models so small? Or: What has to be saved to fully specify a SVM model?

Solution:

```
# your answer
```

Task: What's the expected file size for a k-NN model?

Solution:

```
# your answer
```

69.2 Blog Author Classification (Test)

We want to write a script which takes a list of URLs to blog posts and yields predictions for gender, age and industry of the blog author. For this purpose we have to load our trained models from project *Blog Author Classification (Training)* (page 977) and we have to apply all the necessary preprocessing steps to the downloaded posts.

69.2.1 Getting some Blog Posts

Task: Collect URLs of posts of some blog in a list. Take a blog for which you know gender, age and industry of the author. So we will see whether our models yield good predictions.

Solution:

```
# your solution
```

Task: Download all webpages in the list. Strip HTML tags with Beautiful Soup's `get_text`[782] and join all posts to one string.

Solution:

```
# your solution
```

[782] https://www.crummy.com/software/BeautifulSoup/bs4/doc/#get-text

69.2.2 Preprocessing

Task: Repeat all preprocessing steps from part 1 of the text processing chapter (remove punctuation, tokenize, lemmatize).

Solution:

```
# your solution
```

Task: Load the three label lists and the vectorizer.

Solution:

```
# your solution
```

Task: Vectorize the lemmatized text.

Solution:

```
# your solution
```

69.2.3 Prediction

Task: Load the three saved SVC models.

Solution:

```
# your solution
```

Task: Predict the blog author's gender, age and industry. Provide the result in human readable form.

Solution:

```
# your solution
```

CHAPTER

SEVENTY

SUPERMARKET CUSTOMERS

To understand general customer behavior and for targeted advertising supermarket companies have to identify groups of customers with similar behavior. Sending all customers identical advertisments fails most customers needs. Producing individual advertisements for each potential customer would be too expensive. Thus, clustering customers into a handful of groups should be a sensible middle ground.

There are several data sets for supermarkets available. We use the one provided at Michele Coscia's website[783]. It is sufficiently ridge and has simple structure. The data set is free to use (private communication with M. Coscia). Data comes from italian Coop supermarkets. In Explaining the Product Range Effect in Purchase Data[784] the data set is described in more detail.

70.1 Understanding the Data Set

Task: Get the data set. Read section III of the accompanying paper till the end of the left column on page 3. Answer the following questions:

- How many shops?
- How many customers?
- Are there customers missing in the data?
- Which time interval?
- What's the detail level of products?
- How many products?

Solution:

```
# your answer
```

Task: Load prices and purchases data.

Solution:

```
# your solution
```

Task: Collect following product information:

- number of shops selling the product,
- total quantity sold,
- number of customers who bought the product,
- maximum quantity bought by one customer.

[783] https://www.michelecoscia.com/?page_id=379

[784] https://www.michelecoscia.com/wp-content/uploads/2013/09/geocoop.pdf

Get minimum, average, maximum for all values.

Solution:

```
# your solution
```

Task: Collect following customer information:

- number of shops visited,
- total number of items bought,
- number of different products bought.

Get minimum, average, maximum for all values.

Solution:

```
# your solution
```

70.2 Cleaning the Data Set

The aim of this project is to cluster the set of customers into a handful of groups for targeted advertising. Outliers are not of interest because else the number of groups would become to large and advertising too expensive.

We only are interested in average customers and products. For example, products bought only by very few customers or not available in all shops should be removed from the data set. Customers buying only occasionally at the shops should be removed, too.

Task: Remove all products, customers, purchases such that remaining data satisfies the following conditions:

- each product has been sold in all shops,
- each product has been sold at least 1000 times,
- each product has been bought by at least 100 different customers,
- each product has been bought at least 4 times by at least one customer,
- each customer bought at least 10 items per month (on average),
- each customer bought at least 20 different products.

How many products, customers, purchases do we have now?

Solution:

```
# your solution
```

70.3 Preparing Data for Clustering

We want to use Scikit-Learn's k-means implementation for clustering. Thus, we have to bring our data into the right shape.

Task: Create a NumPy array with one row per customer and one column per product. Store quantities of products bought by each customer in the array. Sort customers descending with respect to the total number of items they bought and products descending with respect to the total quantity sold (sorting may simplify visualization later on).

Solution:

```
# your solution
```

70.4 Scaling

Scaling the data will influence the clustering process. We have several options:

- Without scaling data, products sold in large quantities will dominate the Euclidean distance between the product vectors of two customers. Thus, customers will have small distance if the products they bought most often do coincide.
- Standardization per product ensures that the total quantity sold of a product does not matter. The distance between two product vectors is the mean squared difference of per product quantities bought by both customers. Customers buying similar quantities of each product will have small distance.
- If we are more interested in the selection of products of each customer than in the quantities bought, we should normalize the product vectors. Then the total quantity bought by each customer is identical and data only contains information on the composition of each customers shopping cart. Here ℓ^1-norm should be used. Then all product quantities will sum to 1 and can be interpreted as probability that a product is bought by the customer.

Task: Prepare productwise standardized data and customerwise normalized data.

Solution:

```
# your solution
```

70.5 Clustering

Task: Cluster the data set with k-means for unscaled, standardized, and normalized data. Choose some good k for each variant. Keep the three `KMeans` objects with best k for further analysis.

Solution:

```
# your solution
```

70.6 Analyzing the Clusters

Now that we have identified groups of customers with similar behavior, it's time to understand those groups. Remember, that we want to adapt our advertising campaign to customer behavior.

Task: Visualize cluster centers in a quantity versus product index plot. Don't forget to back scale the data.

Solution:

```
# your solution
```

Each cluster center can be regarded as a prototype customer of the cluster.

Task: Characterize the prototype customers for unscaled and standardized data in words a person designing advertising campaigns understands.

Solution:

```
# your answer
```

The first two clusterings are more or less trivial and useless for targeted advertising. The third clustering deserves further investigation.

Task: Get the 100 most popular products (highest average per customer quantity) per cluster for the third clustering. By how many products differ the top 100 products? Do the same for the first clustering and for a random clustering.

Solution:

```
# your solution
```

Task: Consider the third clustering only. Does one of the customer groups buy higher quantities than the other? Visualize the answer to this question for different price regions.

Solution:

```
# your solution
```

CHAPTER

SEVENTYONE

CHINESE CELADONS

This series of projects investigates the relations between different types of Chinese celadons (early porcelains).

71.1 Hierarchical Clustering

We want to find clusters in a set of celadons (early porcelains). A data set with material properties of a number of celadons found in China is available from UCI Machine Learning Repository[785]. The data set originates from research work published in Data-driven research on chemical features of Jingdezhen and Longquan celadon by energy dispersive X-ray fluorescence[786] by Ziyang He, Maolin Zhang, Haozhe Zhang. A free preprint PDF file[787] is available, too.

Task: Read (at least) the last paragraph of section 1 and section 2 of the afore mentioned preprint. How many celadon sample do you expect in the data set after reading?

Solution:

```
# your answer
```

71.1.1 Understanding the data

Data comes as a CSV file.

Task: Load the data to a data frame. Why are there so many samples?

Solution:

```
# your solution
```

Task: Create a NumPy array holding the data with one row per sample and one column per feature.

Solution:

```
# your solution
```

Task: Create a list of sample names.

Solution:

```
# your solution
```

[785] https://archive.ics.uci.edu/ml/datasets/Chemical+Composition+of+Ceramic+Samples
[786] https://www.sciencedirect.com/science/article/pii/S0272884215023135
[787] https://arxiv.org/pdf/1511.07825.pdf

71.1.2 Preprocessing

Units of measurement are weight per cent for body features and parts per million for glaze features. Since all features are equally important for finding similar celadons we should standardize features independently. Maybe the assumption of equal importance is not correct, but without further domain knowledge we cannot do better.

Task: Standardize all features.

Solution:

```
# your solution
```

71.1.3 Hierarchical Clustering

Task: Plot a dendrogram and determine a sensible number of clusters.

Solution:

```
# your solution
```

Task: Cluster data into the chosen number of clusters.

Solution:

```
# your solution
```

71.1.4 k-Means Clustering

Task: Use k-means for clustering. Determine a good k.

Solution:

```
# your solution
```

Task: Compare results from hierarchical and k-means clustering.

Solution:

```
# your solution
```

71.2 Density-based Clustering

Task: Load, rearrange, standardize the celadons data set.

Solution:

```
# your solution
```

Task: Plot a histogram of pairwise distances to get a feeling for distances in the data set.

Solution:

```
# your solution
```

Task: Use DBSCAN algorithm for clustering. Print sample names for each cluster (and outliers).

Solution:

```
# your solution
```

Task: Use OPTICS algorithm for clustering. Choose clusters manually.

Solution:

```
# your solution
```

Task: Visualize the data set exploiting the tree structure generated by OPTICS algorithm. Label each node with the sample name.

Solution:

```
# your solution
```

CHAPTER

SEVENTYTWO

COLOR PERCEPTION

Human color perception is not as trivial as it may look at first glance. The number of independent colors recognizable by humans, and thus the dimension of the human color space, is not obvious. In the project we want to use MDS to get some insights into human color perception.

The data set we want to analyze is from 1954 and can be found in a table on page 2 of Dimensions of Color Vision[788]. Corresponding CSV file ships with this project's notebook. It contains similarity scores for pairs of colors (light of different wave lengths).

Task: Read the following excerpt from Ekman's 1954 paper ('Table 1' is in the CSV file). How did Ekman obtaine the similarity scores?

Solution:

```
# your notes
```

Task: Load the data set. Create a list of wave lengths (for labeling plots below) and a distance (!) matrix.

Solution:

```
# your solution
```

72.1 1d Embedding

Wave lengths are real numbers and, thus, contained in a one dimensional linear manifold. If human perception of color differences is proportional to wave length, then the color space should have a nice embedding into 1d space.

Task: Use metric MDS to get a 1d embedding of the data set.

Solution:

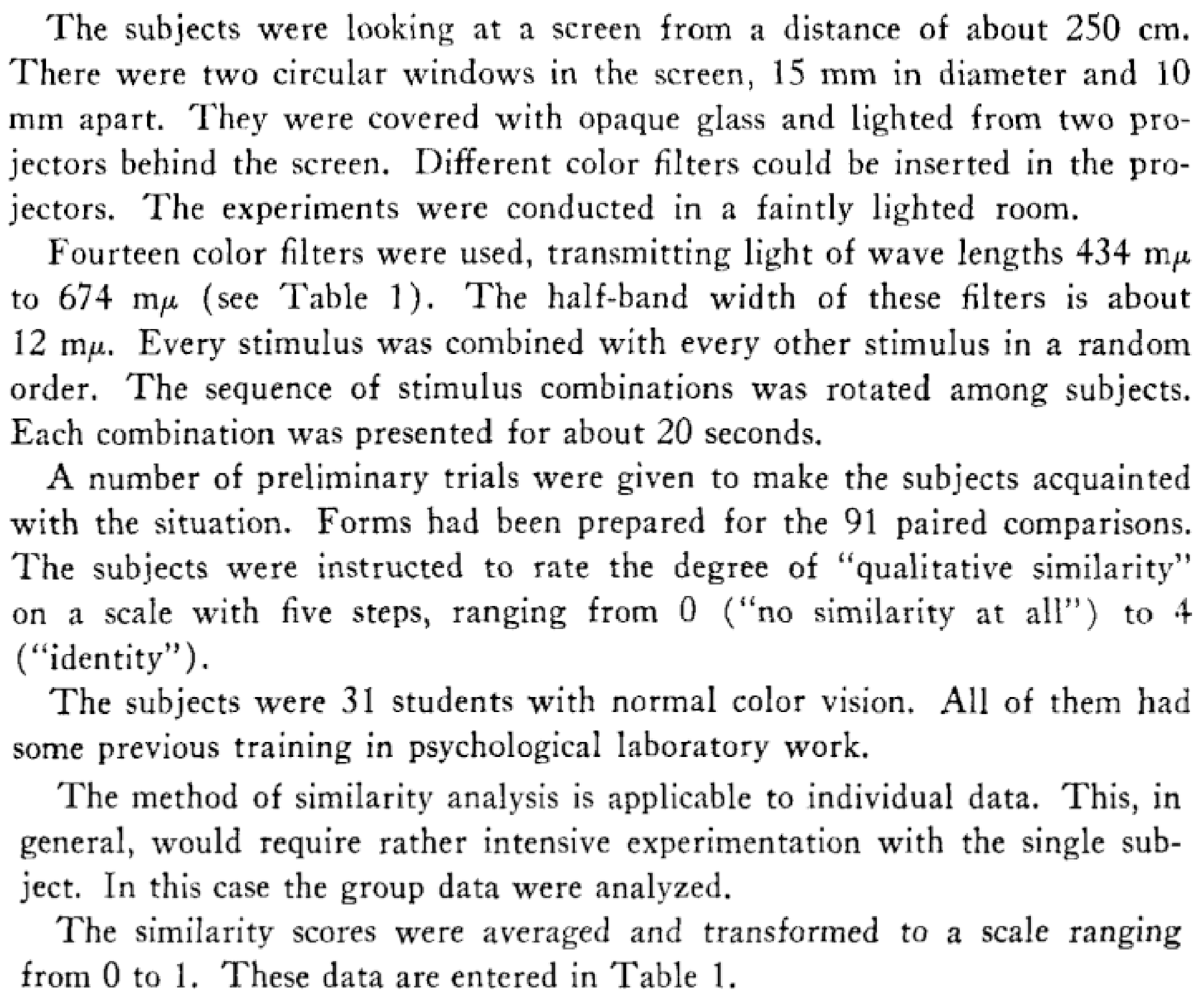
```
# your solution
```

Task: Get the distance matrix of the embedded data set. Visualize (dis)similarity of both distance matrices.

Solution:

```
# your solution
```

[788] https://doi.org/10.1080/00223980.1954.9712953



The subjects were looking at a screen from a distance of about 250 cm. There were two circular windows in the screen, 15 mm in diameter and 10 mm apart. They were covered with opaque glass and lighted from two projectors behind the screen. Different color filters could be inserted in the projectors. The experiments were conducted in a faintly lighted room.

Fourteen color filters were used, transmitting light of wave lengths 434 mμ to 674 mμ (see Table 1). The half-band width of these filters is about 12 mμ. Every stimulus was combined with every other stimulus in a random order. The sequence of stimulus combinations was rotated among subjects. Each combination was presented for about 20 seconds.

A number of preliminary trials were given to make the subjects acquainted with the situation. Forms had been prepared for the 91 paired comparisons. The subjects were instructed to rate the degree of "qualitative similarity" on a scale with five steps, ranging from 0 ("no similarity at all") to 4 ("identity").

The subjects were 31 students with normal color vision. All of them had some previous training in psychological laboratory work.

The method of similarity analysis is applicable to individual data. This, in general, would require rather intensive experimentation with the single subject. In this case the group data were analyzed.

The similarity scores were averaged and transformed to a scale ranging from 0 to 1. These data are entered in Table 1.

Fig. 72.1: Data table in Ekman's 1954 paper. The opposite of 'big data'.

72.2 2d Embedding

Task: Repeat steps from above for a 2d embedding.

Solution:

```
# your solution
```

72.3 3d Embedding

Task: Repeat steps from above for a 3d embedding.

```
# your solution
```

72.4 Higher Dimensions

Task: Use PCA to determine the number of dimensions needed to accurately model the space of human perceivable colors.

Solution:

```
# your solution
```

CHAPTER

SEVENTYTHREE

FOREST FIRES

In this project we want to obtain some insight into different types of forest fires in a Portuguese national park. Data is available at the UCI Machine Learning Repository[789]. It covers forest fires from January 2000 till December 2003 and provides several numerical features of soil moisture and weather.

Task: Get the data and have a look at Data Mining Approach to Predict Forest Firesusing Meteorological Data[790]. What are FFMC, DMC, DC, ISI?

Solution:

```
# your answer
```

Task: Load the data. Remove outliers. Scale data where necessary. Create a NumPy array containing for all samples FFMC, DMC, DC, ISI.

Solution:

```
# your solution
```

Task: Visualize data with 2d Isomap. Interpret the 2d embedding. Use visualizations of all the other features (not only FFMC, DMC, DC, ISI). Can you identify clusters or any other useful structure? Describe different types of forest fires.

Solution:

```
# your solution
```

Task: Use PCA to project data into 2 dimensions. Can you see different fire types here, too?

Solution:

```
# your solution
```

789 http://archive.ics.uci.edu/dataset/162/forest+fires
790 https://core.ac.uk/download/pdf/55609027.pdf

CHAPTER

SEVENTYFOUR

ONLINE ADVERTISING

In this project we implement the methods discussed in *Stateless Learning Tasks (Multi-armed Bandits)* (page 801) for the online advertising example given there.

Of course, we have to simulate the environment to train the agent. For stateless reinforcement learning tasks the environment from the agent's point of view looks like a fixed number of random number generators, each following a different probability distribution. Here we use the following functions to simulate an environment with 10 possible actions (ads to show):

```
import numpy as np

rng = np.random.default_rng(0)

class Env:

    def __init__(self, stationary=False):

        self.stationary = stationary

        # initial probabilities for reward 1 (click rates), Bernoulli distribution
        self.p = np.array([0.2, 0.3, 0.1, 0.4, 0.5, 0.5, 0.7, 0.9, 0.8, 0.4])

        if not self.stationary:

            # drift in p for simulation of non-stationary environments
            self.delta_p = 0.001 * np.array([1, -1, 1, 0, 1, 1, 0, -1, -1, 1])

    def action(self, a):
        ''' Take action a (0-9) and return reward. '''

        if not self.stationary:
            self.p = np.clip(self.p + self.delta_p, 0.05, 0.95)

        # reward
        return rng.binomial(1, self.p[a])
```

74.1 Sample Averaging

Task: Write a function `sample_averaging` that implement the sample averaging method for stateless learning tasks with the ε-greedy policy. Arguments:

- an `Env` object,
- the value for ε,
- the update factor α (where 0 indicates no weighting),
- list of initial action values,

- length of episode (number of steps to run).

Returns:

- list of average reward after each step.

Create a stationary `Env` object, run an episode of 1000 steps, and plot average reward vs. step. Note that the area under the curve is the return obtained in the episode.

Solution:

```
# your solution
```

74.2 Stationary Problem with ε-Greedy Policy

Task: Run episodes for $\varepsilon \in \{0, 0.01, 0.1, 0.5, 1\}$ with 5000 steps. For each ε run 100 episodes and plot corresponding mean average rewards (vs. step). What do you see and why?

Solution:

```
# your solution
```

```
# your observations
```

Task: Plot average reward vs. step for 100 episodes with $\varepsilon = 0$ and, in another plot, for 100 episodes with $\varepsilon = 1$ (5000 steps each). Use thin lines to see more lines in the plots. What do you learn from the plots about the averaged values in the previous task?

Solution:

```
# your solution
```

```
# your observations
```

74.3 Stationary Problem with Optimistic Initial Values

Task: Plot average reward vs. step for $\varepsilon = 0$ and optimistic initial values 0, 0.2, 0.4,…, 2. Use averages over 100 episodes as above.

What do you see and why?

Solution:

```
# your solution
```

```
# your observations
```

Task: Repeat the previous task with $\alpha = 0.5$.

Solution:

```
# your solution
```

```
# your observations
```

74.4 Non-Stationary Problem

Task: Run 20 episodes with $\varepsilon = 0$, $\alpha = 0.5$ and 10000 steps in a non-stationary environment. Plot average reward vs. step for all episodes and explain what you see.

Solution:

```
# your solution
```

```
# your observations
```

Task: Run episodes for $\varepsilon \in \{0, 0.01, 0.1, 0.2, 1\}$ with $\alpha = 0.5$ and 10000 steps. For each ε run 100 episodes and plot corresponding mean average rewards (vs. step). What do you see and why?

Solution:

```
# your solution
```

```
# your observations
```

CHAPTER

SEVENTYFIVE

FROZEN LAKE

Frozen Lake[791] is a simple grid world simulator contained in Gymnasium[792], an open source project for standardizing environment simulators.

75.1 Dynamic Programming

Read *Dynamic Programming* (page 817) before you start.

In this project (and all others of this project series) we use the grid world simulator Frozen Lake[793] originally developed by OpenAI[794] in their `gym` package (until they removed the 'non' in 'non-profit organization) and now maintained by the non-profit Farama Foundation[795] in their `gymnasium` package. See Announcing The Farama Foundation[796] for some more background information on this transition.

Task: Read about Gymnasium[797] and the Frozen Lake simulator[798]. Then install Gymnasium.

75.1.1 The Environment

Task: Create the standard 8-by-8 Frozen Lake environment object and render the environment. Use `is_slippery=True`

Solution:

```
# your solution
```

Task: Write your own `render` function for getting a more pleasant rendering. It should take the environment as argument. The frozen lake map is contained as list of lists (rows) in `env.desc`. Note that we want to test dynamic programming. So we do not care about a starting position, because we will solve the problem for all starting positions at once.

```
# your solution
```

Task: Get the environment dynamics for $p(s', r, s, a)$ for all arguments (four dimensional array). Relevant information is in `env.P`. Check that $p(0,0,0,0) = 2/3$, else your solution is not correct.

[791] https://gymnasium.farama.org/environments/toy_text/frozen_lake/
[792] https://gymnasium.farama.org/
[793] https://gymnasium.farama.org/environments/toy_text/frozen_lake/
[794] https://openai.com/
[795] https://farama.org/
[796] https://farama.org/Announcing-The-Farama-Foundation
[797] https://gymnasium.farama.org/
[798] https://gymnasium.farama.org/environments/toy_text/frozen_lake/

```
# your solution
```

75.1.2 Value Iteration

Task: Implement asynchronous value iteration to get the optimal state-value function v_*. Use $\gamma = 1$. Show values in an 8-by-8 grid.

```
# your solution
```

Task: From optimal state values get optimal action values.

```
# your solution
```

Task: Get an optimal policy from optimal action values.

```
# your solution
```

Task: Visualize the policy.

```
# your solution
```

Task: Run your code with the following parameters and explain what you see (optimal value function, optimal policy):

- `slippery=False`, $\gamma = 0.5$
- `slippery=False`, $\gamma = 0$
- `slippery=False`, $\gamma = 1$
- `slippery=True`, $\gamma = 0.5$
- `slippery=True`, $\gamma = 0$
- `slippery=True`, $\gamma = 1$

```
# your answer
```

75.1.3 Policy Iteration

Task: Implement policy iteration for state values. Use a deterministic random initial policy.

```
# your solution
```

75.2 Monte Carlo Methods

Read *Monte Carlo Methods* (page 819) before you start.

In this project we solve the *Frozen Lake* (page 1001) task with Monte Carlo methods. Thus, in contrast to the *Dynamic Programming* (page 817) project we do not need to know then environment dynamics. Instead, the environment will be explored by the agent.

Task: Implement a function `episode` running an episode of Frozen Lake. Arguments are the `env` object and a policy (2d NumPy array of probabilities for each state-action pair). The function shall return a list of state-action-reward tuples. Episodes always start in the environment's default initial state 0.

Solution:

```
# your solution
```

Task: Implement the on-policy Monte Carlo method with an ε-soft policy. Use incremental updates to the action value estimates. Visualize a greedy policy w.r.t. to the action value estimates.

Solution:

```
# your solution
```

Task: Visualize the number of visits per state in a heatmap. Did the agent discover the whole map? What about the policy for states in less discovered regions.

```
# your solution
```

75.3 SARSA

Read *Temporal Difference Learning (TD Learning)* (page 825) before you start.

In this project we solve the *Frozen Lake* (page 1001) task with SARSA (TD learning). We do not need to know then environment dynamics and implementation is much simpler than for *Monte Carlo Methods* (page 1002).

75.3.1 Random Starting Positions

By default Frozen Lake episodes always start in the upper left corner. But would like to start at a new random position in each episode. Looking at the source code[799] we see that the starting position is chosen randomly from all cells labeled `S`.

Task: Create a Frozen Lake environment with standard 8-by-8 map, extract the map via `Env.desc`, replace all `F` (frozen cell) by `S`, and create a new environment object from the new map.

Solution:

```
# your solution
```

75.3.2 SARSA implementation

Task: Implement the SARSA algorithm with decreasing ε. After each episode print `O` or `X` for goal reached or not (without line breaks to get a visual impression of training progress).

Solution:

```
# your solution
```

Task: Visualize the resulting (greedy) policy.

Solution:

```
# your solution
```

[799] https://github.com/Farama-Foundation/Gymnasium/blob/a2663125c426951a409f10d5d1297aaf267bc3b4/gymnasium/envs/toy_text/frozen_lake.py#L236

CHAPTER

SEVENTYSIX

Q-LEARNING FOR TIC-TAC-TOE

Read *Temporal Difference Learning (TD Learning)* (page 825) before you start.

The aim of this project is to create an AI player for Tic-tac-toe[800] using Q-learning. The AI player will not only learn how to win the game but he'll also have to learn the rules of the game.

We'll have a board object holding the state information of the game and two player objects interacting with the board. Interaction is not direct. Instead all information flow is controlled by code living outside board and player objects. Thus, board and players do not have to know how to control each other.

```
import numpy as np

rng = np.random.default_rng(0)
```

76.1 The Board

The board is 3-by-3 indexed rowwise:

```
0 1 2
3 4 5
6 7 8
```

The state of each field is represented by a one-character string. Symbols used are flexible, but `' '` indicates an empty field.

```
class Board:

    # some constants to increase readability of code
    OKAY = 0
    INVALID = 1
    WIN = 2
    DRAW = 3

    def __init__(self, symbol1='X', symbol2='O'):
        ''' Create empty board. Player 1 has first move. '''

        self.symbol1 = symbol1
        self.symbol2 = symbol2
        self.board = [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
        self.game_over = False
        self.next_player = 1  # player 1 always has first move

    def _win(self, symbol):
        ''' Check whether player with symbol won the game. '''
```

(continues on next page)

800 https://en.wikipedia.org/wiki/Tic-tac-toe

(continued from previous page)

```
        b = self.board  # shorthand
        s3 = 3 * (symbol, )

        if (b[0], b[1], b[2]) == s3 or \
           (b[3], b[4], b[5]) == s3 or \
           (b[6], b[7], b[8]) == s3 or \
           (b[0], b[3], b[6]) == s3 or \
           (b[1], b[4], b[7]) == s3 or \
           (b[2], b[5], b[8]) == s3 or \
           (b[0], b[4], b[8]) == s3 or \
           (b[2], b[4], b[6]) == s3:
            return True
        else:
            return False

    def take(self, field):
        ''' Take a field (0...8) for next player and return status. '''

        # no moves allowed if game is over
        if self.game_over:
            return Board.INVALID

        # valid move?
        try:
            field = int(field)
        except:
            return Board.INVALID
        if field < 0 or field > 8:
            return Board.INVALID
        if self.board[field] != ' ':
            return Board.INVALID

        self.game_over = True  # set to False below if appropriate

        # take field and check for win
        symbol = self.symbol1 if self.next_player == 1 else self.symbol2
        self.board[field] = symbol
        if self._win(symbol):
            return Board.WIN
        else:
            self.next_player = 2 if self.next_player == 1 else 1
        if ' ' not in self.board:
            return Board.DRAW

        # if we arrive here, game is not over
        self.game_over = False
        return Board.OKAY

    def render(self):
        ''' Print current state of the board. '''

        b = self.board  # shorthand
        print('+---+---+---+')
        print('| ' + b[0] + ' | ' + b[1] + ' | ' +  b[2] + ' |')
        print('+---+---+---+')
        print('| ' + b[3] + ' | ' + b[4] + ' | ' +  b[5] + ' |')
        print('+---+---+---+')
        print('| ' + b[6] + ' | ' + b[7] + ' | ' +  b[8] + ' |')
        print('+---+---+---+')
```

Task: For testing create a board, take some fields and render the board.

Solution:

```
# your solution
```

76.2 Players

We create an abtract player class from which we may derive different types of players (human player, AI player with random behavior, AI player controlled by Q-learning,…).

76.2.1 Abstract Player

```
class Player:

    def __init__(self, symbol):

        self.symbol = symbol

    def field(self, board):
        ''' Choose a field (0...8) to take. '''

        raise NotImplementedError

    def reward(self, r, board):
        ''' Numerical feedback for player and new board state.
        Called by controller after each call of field method. '''

        pass
```

76.2.2 Random Player

For testing purposes we implement an AI player who chooses one of the empty fields uniformly at random.

```
class RandomPlayer(Player):

    def field(self, board):
        ''' Choose a field (0...8) to take. '''

        free = np.array([board[i] == ' ' for i in range(9)])
        return rng.choice(np.arange(9)[free])
```

76.2.3 Human Player

In the end we want to play a game against our Q-learning AI player. Thus, we need a player object asking us for a field to choose whenever the player object's `field` method is called.

```
class HumanPlayer(Player):

    def field(self, board):

        print('| ' + board[0] + ' ' + board[1] + ' ' + board[2] + ' |   0 1 2')
        print('| ' + board[3] + ' ' + board[4] + ' ' + board[5] + ' |   3 4 5')
```

(continues on next page)

(continued from previous page)

```
        print('| ' + board[6] + ' ' + board[7] + ' ' + board[8] + ' |   6 7 8')
        return input('Which field? ')
```

76.3 Episodes

An episode of playing tic-tac-toe is realized by the following function, which returns the final game status and the last symbol added to the board. Note that an episode ends as soon as a player requests an invalid move.

```
rewards = {Board.WIN: 2, Board.DRAW: 1, Board.OKAY: 0, Board.INVALID: -10}

def episode(p1, p2):
    ''' Play an episode between two Player objects and return final status and␣
↪symbol. '''

    game = Board(p1.symbol, p2.symbol)
    status = Board.OKAY
    p = p2  # implies that p1 starts the game

    while status == Board.OKAY:
        p = p2 if p == p1 else p1
        status = game.take(p.field(game.board))
        p.reward(rewards[status], game.board)

    return status, p.symbol
```

Task: Create a board, a random AI player and a human player. Test play some episodes.

Solution:

```
# your solution
```

Task: Let two random players play 1000 games against each other. How many wins does each player have? How many draws?

Solution:

```
# your solution
```

76.4 Q-Learning AI Player

Task: Think about memory requirements and suitable data structures for implementing a Q-learning based AI player.

Solution:

```
# your notes
```

Task: Create an AI player class `QLearningPlayer`, which allows to train an AI player via Q-learning.

Solution:

```
# your solution
```

Task: Let the Q-learning AI player train by playing against a random AI player. Count and compare number of wins, draws and invalid moves. Print and reset counters every n episodes for suitable n. Also switch the roles of player 1 and 2 at these points, because the AI player should learn to play in both roles (remember that player 1 always has the first move).

Solution:

```
# your solution
```

Task: Train two AI players by letting them play against each other. Count and compare wins and draws.

Solution:

```
# your solution
```

Task: Play a game against the trained AI player.

Solution:

```
# your solution
```

CHAPTER

SEVENTYSEVEN

CART POLE

Cart Pole[801] is a physics simulator contained in Gymnasium[802], an open source project for standardizing environment simulators. It simulates movement of a pole mounted on top of a moving car.

77.1 The Cart Pole Environment

In this project we learn how to use `gymnasium`'s Cart Pole[803]. The cart pole environment simulates a pole mounted on top of a car. If the car moves (and even if it does not move), the pole most likely will fall down. But if we control the car's movements in the correct way, then we might be able to balance the pole.

Task: Read about the cart pole environment. Describe the set of states and the set of actions and the set of rewards. When does an episode end?

Solution:

```
# your answer
```

Task: Create a cart pole environment object and run one episode by choosing actions randomly. Display episode length. Don't try to render the environment. Rendering will be discussed below.

Solution:

```
# your solution
```

Rendering via `Env.render()` does not work in Jupyter. But we may record a video and then show the video in the notebook. This workflow requires `pygame`[804] and `moviepy`[805] to be installed.

Task: Have a look at Gymnasium doc's Recording Agents[806] and at `IPython.display.Video`[807]. Then record one episode and show the video in the notebook.

Solution:

```
# your solution
```

[801] https://gymnasium.farama.org/environments/classic_control/cart_pole/
[802] https://gymnasium.farama.org/
[803] https://gymnasium.farama.org/environments/classic_control/cart_pole/
[804] https://www.pygame.org
[805] https://zulko.github.io/moviepy/
[806] https://gymnasium.farama.org/main/introduction/record_agent/
[807] https://ipython.readthedocs.io/en/stable/api/generated/IPython.display.html#IPython.display.Video

77.2 SARSA

Read *Approximate Value Function Methods* (page 829) and also have a look at *The Cart Pole Environment* (page 1011) project before you start.

Task: Implement SARSA with differential return for the cart pole environment. Represent value function estimates by a two-layer ANN with 30 neurons per layer. Train about some 100 episodes and print obtained return after each episode.

Modify the algorithm given in *Approximate Value Function Methods* (page 829) as follows:

- Use TensorFlow's `Adam` optimizer instead of manually implementing the ANN weight update.
- Start with $\varepsilon = 1$ and decrease ε slightly after each episode. Take care that ε cannot become arbitrarily small.

Note that the cart pole environment always yields reward 1, even if the pole fell down. Maybe it's better to use reward 0 in that case.

Solution:

```
# your solution
```

Task: After sufficiently long training run an episode and render the agent's behavior.

Solution:

```
# your solution
```

77.3 Deep Q-Learning

Read *Approximate Value Function Methods* (page 829) and also have a look at *The Cart Pole Environment* (page 1011) project before you start.

Task: Implement deep Q-learning for the cart pole environment. Represent value function estimates by a two-layer ANN with 30 neurons per layer. Train about some 100 episodes and print obtained return after each episode.

Modify the algorithm given in *Approximate Value Function Methods* (page 829) as follows:

- Use TensorFlow's `Adam` optimizer instead of manually implementing the ANN weight update.
- Start with $\varepsilon = 1$ and decrease ε slightly after each episode. Take care that ε cannot become arbitrarily small.

Note that the cart pole environment always yields reward 1, even if the pole fell down. Maybe it's better to use reward 0 in that case.

Solution:

```
# your solution
```

Task: After sufficiently long training run an episode and render the agent's behavior.

Solution:

```
# your solution
```

77.4 Policy Gradient Method

Read *Policy Gradient Methods* (page 839) and also have a look at *The Cart Pole Environment* (page 1011) project before you start.

Task: Implement the SARSA-base actor-critic method with baseline for the cart pole environment. Represent value function estimates by a two-layer ANN with 30 neurons per layer. For the policy use a same-sized ANN. Train about some 100 episodes and print obtained return after each episode.

Use TensorFlow's `Adam` optimizer instead of manually implementing the ANN weight update.

Note that the cart pole environment always yields reward 1, even if the pole fell down. Maybe it's better to use reward 0 in that case.

Solution:

```
# your solution
```

Task: After sufficiently long training run an episode and render the agent's behavior.

Solution:

```
# your solution
```

Part XI

Mathematics

CHAPTER

SEVENTYEIGHT

LOGIC

The field of logic consideres truth values of mathematical expressions.

78.1 Logical Operators

Truth values can be transformed or combined by *logical operators*.

78.1.1 Not

The not operator inverts the truth values of an expression.

a	not a
true	false
false	true

78.1.2 And

The and operator yields true if and only if both operands are true.

a	b	a and b
true	true	true
true	false	false
false	true	false
false	false	false

78.1.3 Or (inclusive)

The or operator yields true if and only if at least one of both operands is true. This is sometimes called *inclusive or* because the and-case (both operands true) is included (that is, yields true).

a	b	a or b
true	true	true
true	false	true
false	true	true
false	false	false

78.1.4 Or (exclusive)

The xor operator yields true if and only if exactly one of both operands is true. This called *exclusive or* because the and-case is excluded.

a	b	a xor b
true	true	false
true	false	true
false	true	true
false	false	false

CHAPTER

SEVENTYNINE

COMBINATORICS

Combinatorics is the mathematical field of counting. An application are discrete distributions in simple probability theory.

79.1 Factorial

The factorial of a non-negative integer n is the integer

$$n! := 1 \cdot 2 \cdot \dots \cdot (n-1) \cdot n,$$

where $0! := 1$.

Obviously, $n! = (n-1)!\, n$.

CHAPTER

EIGHTY

LINEAR ALGEBRA

This chapter summarizes tools and results from linear algebra used throughout the book.

80.1 Vectors

The term *vector* is used in several different contexts, each coming with a slightly different definition. In basic linear algebra a vector often is considered a finite column of numbers. That's the approach we follow here.

80.1.1 Definition

For $d \in \mathbb{N}$ a d*-tuple* is an ordered list of real numbers, typically written as $(x_1, x_2, \ldots, x_d)$ with $x_1, \ldots, x_d$ denoting the numbers. Here 'ordered' means that swapping two unequal numbers in the list yields a different d-tuple. Example: $(1, 2, 3) \neq (1, 3, 2)$. By $\mathbb{R}^d$ we denote the set of d-tuples.

Vector is another term for d-tuple. In linear algebra vectors may be interpreted as points in space or as difference between two points (that is, describing a translation). Vectors often are written as columns:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}.$$

80.1.2 Length of a Vector

The (Euclidean) length of a vector x is defined as

$$|x| := \sqrt{x_1^2 + \cdots + x_d^2}.$$

80.1.3 Sum of Vectors

Sums of vectors are defined componentwise:

$$x+y:=\begin{bmatrix}x_1+y_1\\ \vdots\\ x_d+y_d\end{bmatrix}.$$

80.1.4 Multiples of Vectors

Products of real numbers and vectors are defined componentwise. For $a\in\mathbb{R}$ and $x\in\mathbb{R}^d$ we have

$$a\,x:=\begin{bmatrix}a\,x_1\\ \vdots\\ a\,x_d\end{bmatrix}.$$

80.1.5 Inner Products

The inner product of two vectors $x,y\in\mathbb{R}^d$ is

$$\langle x,y\rangle:=x_1\,y_1+\cdots+x_d\,y_d.$$

Inner products are closely related to angles between vectors.

80.1.6 Outer Products

The outer product of two vectors $x,y\in\mathbb{R}^3$ is

$$x\times y:=\begin{bmatrix}x_2\,y_3-x_3\,y_2\\ x_3\,y_1-x_1\,y_3\\ x_1\,y_2-x_2\,y_1\end{bmatrix}.$$

The outer product yields a vector orthogonal to both factors.

80.2 Matrices

A *matrix* is a rectangular scheme of numbers. Matrices frequently appear in almost all fields of mathematics because they can be used to represent abstract concepts like linear mappings numerically. Many abstract operations in mathematics boil down to matrix computations as soon as concrete numerical examples are considered.

80.2.1 Definition

For $m\in\mathbb{N}$ and $n\in\mathbb{N}$ a *matrix* is an m-tuple of n-tuples (see *Vectors* (page 1021) for definition of tuples). Example:

$$\big((1,4,5,-2),(3,2,-7,10),(-2,4,5,8)\big)\qquad\text{here}\;\;m=3\;\;\text{and}\;\;n=4.$$

To simplify notation matrices are written as rectangular schemes:

$$\begin{pmatrix}1&4&5&-2\\3&2&7&10\\-2&4&5&8\end{pmatrix}\qquad\text{or}\qquad\begin{bmatrix}1&4&5&-2\\3&2&7&10\\-2&4&5&8\end{bmatrix}$$

The set of all realvalued matrices with m rows and n columns is denoted as $\mathbb{R}^{m\times n}$.

Matrices usually are denoted by uppercase letters and a matrix' elements by corresponding lowercase letters with double index. The first index denotes the row, the second the column of the element. Example:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} = (a_{ij})_{i=1,\ldots,3}^{j=1,\ldots,4}.$$

Row i is denoted by $a_{i\bullet}$, column j by $a_{\bullet,j}$. Rows and columns can be regarded as vectors.

80.2.2 Special Matrices

A matrix with identical number of rows and columns ($m = n$) is called *square matrix*. The tuple $(a_{11}, a_{22}, \ldots, a_{mm})$ is called *main diagonal* of the matrix A.

A square matrix of the form

$$\begin{pmatrix} 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \ddots & & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & \ddots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \end{pmatrix}$$

is called *identity matrix*.

Matrices of the form

$$\begin{pmatrix} * & \cdots & \cdots & * \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & * \end{pmatrix} \quad \text{and} \quad \begin{pmatrix} * & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ * & \cdots & \cdots & * \end{pmatrix}$$

are called *upper-triangular* and *lower-triangular*.

80.2.3 Transpose

Given a matrix $A \in \mathbb{R}^{m\times n}$ its *transpose* is the matrix

$$A^{\mathrm{T}} := (a_{ji})_{i=1,\ldots,m}^{j=1,\ldots,n} \in \mathbb{R}^{n\times m},$$

that is, the same matrix as A, but with rows and columns interchanged.

The double transpose is equivalent to the original matrix:

$$\left(A^{\mathrm{T}}\right)^{\mathrm{T}} = A.$$

80.2.4 Matrix Multiplication

The product of two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$ is the matrix $C := A\,B \in \mathbb{R}^{m\times p}$ with entries

$$c_{ik} := \sum_{j=1}^{n} a_{ij}\, b_{jk}.$$

80.2.5 Inverse Matrix

A square matrix $A \in \mathbb{R}^{n\times n}$ is called *invertible* if there is a square matrix $B \in \mathbb{R}^{n\times n}$ such that

$$A\,B = I \qquad \text{and} \qquad B\,A = I,$$

where I is the identity matrix. The matrix B is called the *inverse* of A.

80.2.6 Determinants

The determinant $\det A$ of a square matrix $A \in \mathbb{R}^{n\times n}$ is the real number computed from the following iterative rule:

$$\det A := \begin{cases} a_{1,1}, & \text{if } n = 1,\\ \sum_{j=1}^{n}(-1)^{1+j}\,a_{1,j}\,\det A_{1,j}, & \text{if } n > 1, \end{cases}$$

where $A_{1,j}$ is the submatrix of A originating from removing row 1 and column j.

The determinant is non-zero if and only if the matrix is invertible. It is positive if the matrix columns constitute a right-handed coordinate system and negative if columns constitute a left-handed coordinate system.

For each fixed $i \in \{1, \dots, n\}$ we get an equivalent iterative definition:

$$\det A = \sum_{j=1}^{n}(-1)^{i+j}\,a_{i,j}\,\det A_{i,j} \qquad \text{for } n > 1,$$

where $A_{i,j}$ is A without row i and without column j.

One can show that

$$\det A = \det A^{\mathrm{T}},$$

which implies

$$\det A = \sum_{i=1}^{n}(-1)^{i+j}\,a_{i,j}\,\det A_{i,j} \qquad \text{for } n > 1$$

for all $j \in \{1, \dots, n\}$.

80.3 Systems of Linear Equations

A system of linear equations with m equations and n unknowns $x_1, \dots, x_n$ has the form

$$\begin{aligned} a_{1,1}\,x_1 + \dots + a_{1,n}\,x_n &= b_1\\ &\vdots\\ a_{m,1}\,x_1 + \dots + a_{m,n}\,x_n &= b_m \end{aligned}$$

with coefficients a_{ij} and right-hand sides $b_1, \dots, b_m$.

Setting

$$A := \begin{pmatrix} a_{1,1} & \cdots & a_{1,n}\\ \vdots & & \vdots\\ a_{m,1} & \cdots & a_{m,n} \end{pmatrix}, \qquad x := \begin{pmatrix} x_1\\ \vdots\\ x_n \end{pmatrix} \qquad \text{and} \qquad b := \begin{pmatrix} b_1\\ \vdots\\ b_m \end{pmatrix}$$

we may write a system of linear equations in its *matrix form*

$$A\,x = b.$$

Depending on coeefficient matrix A and right-hand side b a system of linear equations either has no solution or exactly one solution or infinitely many solutions.

Part XII

Computer Science

CHAPTER

EIGHTYONE

UNIFIED MODELING LANGUAGE (UML)

There's a standard for visualizing the design of software and other systems: unified modeling language (UML). Next to many other types of visualization it standardizes how to express relations between classes graphically. Especially, inheritance relations can be visualized. We do not go into the details here. But you should know that there is a standard and from time to time you should practice reading UML class diagrams, since they are used for planning and communicating larger software projects.

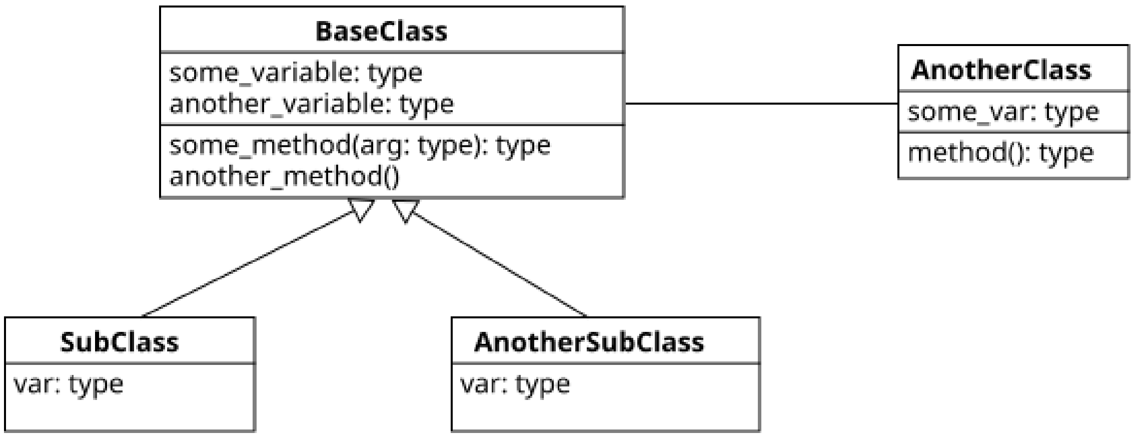


Fig. 81.1: Example of an UML class diagram.

To get an overview of UML class diagrams have a look at Class diagram[808] at Wikipedia.

An open source tool for drawing UML class diagrams is UMLet[809].

Other types of diagrams are shown in Wikipedia's article Unified Modeling Language[810].

[808] https://en.wikipedia.org/wiki/Class_diagram
[809] https://www.umlet.com
[810] https://en.wikipedia.org/wiki/Unified_Modeling_Language

CHAPTER

EIGHTYTWO

CLOUD COMPUTING

Almost all modern computers have several CPUs and one or more GPUs. CPUs are made for general purpose computations, GPUs are specialized chips for fast floating point computations and for matrix operations. Initially, GPUs were used for rendering 3d graphics, but they turned out to be very useful for training machine learning models. Training on a GPU is much faster than training on CPUs.

Powerful GPUs are very expensive and having several of them is even more expensive. Thus, training complex machine learning models should be done on remote computers operated by companies or research institutions. Compute time can be bought from Amazon, Google, Microsoft and many others.

82.1 Different Approaches for Remote Execution

There exist different techniques for executing programs on remote computers. Here we discuss three of them. To understand differences and implications we first have to have a closer look at the relationships between a program and the operating system.

82.1.1 Libraries

Each program uses some libraries (called 'modules' or 'packages' in Python). When shipping a program to the end user or to the cloud we have to decide whether and how to ship the libraries. We could include all libraries into our program (known as *static linkage*), but this would yield a very large program. Alternatively, we could ask the user or the cloud provider to install required libraries before running our program (*dynamic linkage*). This way we have a smaller program and libraries can be used in common by many different programs.

Static linkage is relatively rare nowadays. Dynamic linkage is the standard approach. The major drawback is that we cannot be sure that our program can be executed on the destination system. Even if we provide all required libraries for prior install, some libraries may conflict with ones already installed on the destination system. Concerning the question of how to handle such library problems, there exist three major techniques for executing programs on remote computers.

82.1.2 Physical Machine with Full Control

If we have full control over the remote computer, then we may install everything we need to run our programs. On modern multi-user systems like Linux and macOS (and to some extent also Windows) different users may work in parallel without influencing each other. Some of the libraries are available for all users, but each user may install user specific libraries, too.

Advantages are relatively simple administration and efficient program execution. A disadvantage is that users are forced to use the operating system installed on the remote computer.

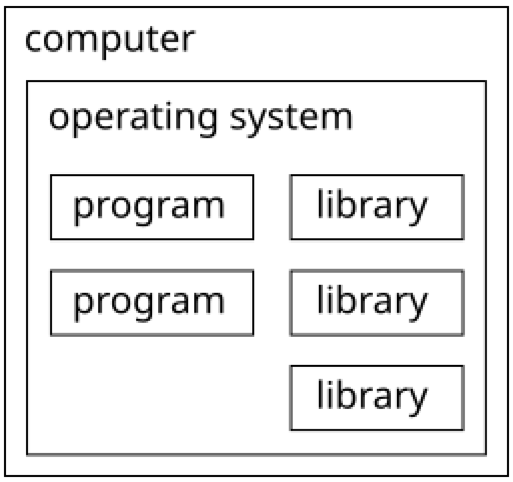


Fig. 82.1: Programs and libraries on a physical machine.

82.1.3 Virtual Machine

The cloud provider may install virtual machines. A virtual machine is a program simulating a whole computer. On such a simulated computer one may install a different operating system than the one installed on the underlying real computer. Several virtual machines may run in parallel and isolated from each other. Every user gets access to a different virtual machine and can do whatever he or she wants to do.

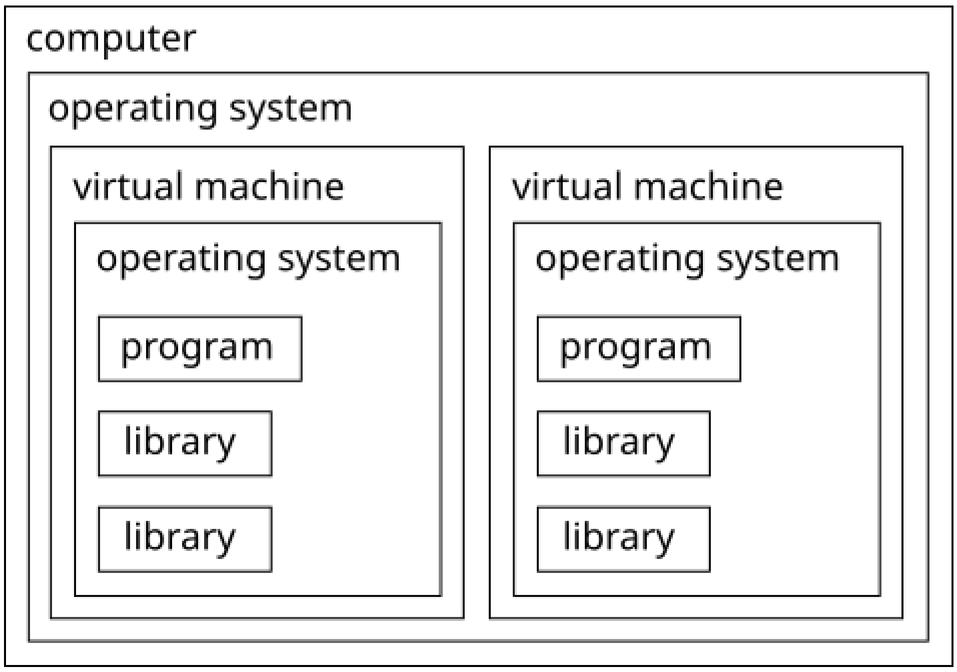


Fig. 82.2: Programs and libraries on two virtual machines.

This technique is rarely seen in practise because virtual machines run relatively slow and require lots of hardware resources. A typical use case is to run Linux software on a Windows machine and vice versa.

82.1.4 Containers (Docker)

Non-Windows operating systems allow for a third technique combining the advantages of direct access to a physical machine and virtual machines. A program and all required libraries can be packaged into a container for shipment. This container is then executed by the operating system in an isolated environment very similar to virtual machines, but much more efficiently.

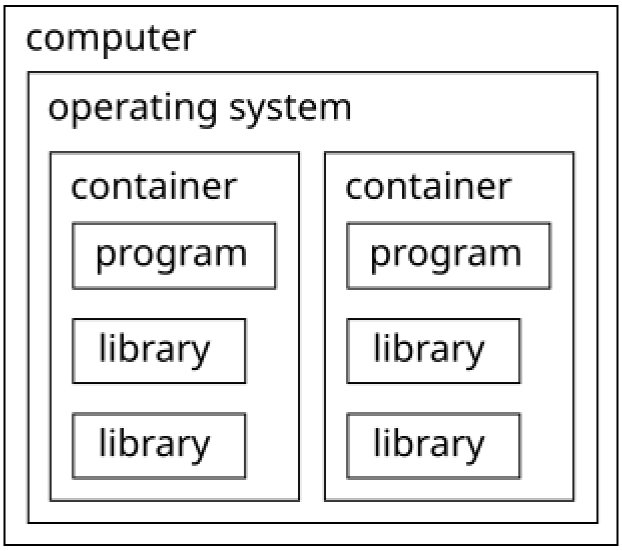


Fig. 82.3: Programs and libraries in two containers.

Docker is a widely used containerization software. Corresponding containers are created from so called Docker images (an image is a blueprint for a container). All major cloud providers can process Docker images. There exist several other containerization tools, Podman for instance.

Containerization uses specific features of non-Windows systems. Docker is available for Windows, too. But the installer installs a virtual machine containing a Linux system and then installs Docker in the virtual Linux.