# Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning 

Jens Flemming

This is a printable version of the interactive ebook available at https://www.whz.de/ jef 19jdw/data-science-ai. Switch to the online HTML version to get all features.
I Data Visualization ..... 3
1 Matplotlib ..... 5
1.1 Matplotlib Basics ..... 5
1.2 3D Plots ..... 36
1.3 Animations ..... 41
1.4 Seaborn ..... 46
1.5 Maps ..... 50
2 Plotly ..... 57
2.1 Plotly Express ..... 57
2.2 Graph Object Interface ..... 58
2.3 Exporting Figures ..... 58
3 Folium ..... 59
3.1 Basic Usage ..... 59
3.2 Advanced Features ..... 60
3.3 Exporting a Map ..... 60
II Supervised Learning ..... 63
4 Machine Learning Overview ..... 65
5 General Considerations ..... 67
5.1 Problem and Workflow ..... 67
5.2 Introductory Example (k-NN) ..... 71
5.3 Quality Measures ..... 82
5.4 Scaling ..... 96
5.5 Feature Reduction ..... 97
5.6 Hyperparameter Tuning ..... 104
6 Linear Regression ..... 113
6.1 Approach ..... 113
6.2 Regularization ..... 129
6.3 Worked Example: House Prices I ..... 142
6.4 Worked Example: House Prices II ..... 175
6.5 Outliers ..... 184
7 Logistic Regression ..... 195
7.1 Binary Logistic Regression ..... 195
7.2 Multiclass Logistic Regression ..... 196
7.3 Logistic Regression with Scikit-Learn ..... 197
7.4 The Need for Regularization ..... 197
7.5 Decision Boundaries for Multiclass Classification ..... 202
8 Artificial Neural Networks ..... 207
8.1 ANN Basics ..... 207
8.2 Training ANNs ..... 216
8.3 ANNs with Keras ..... 233
8.4 Convolutional Neural Networks ..... 252
8.5 CNNs with Keras ..... 265
8.6 What did the CNN learn? ..... 275
8.7 Improving CNN performance ..... 324
9 Decision Trees ..... 341
9.1 Basics ..... 341
9.2 Regression Trees ..... 343
9.3 Classification Trees ..... 349
10 Ensemble Methods ..... 353
10.1 The Idea ..... 353
10.2 Stacking ..... 354
10.3 Bagging ..... 354
10.4 Boosting ..... 355
11 Support-Vector Machines ..... 361
11.1 Hard Margin SVMs ..... 361
11.2 Soft Margin SVMs ..... 366
11.3 Kernel SVMs ..... 369
11.4 SVMs with Scikit-Learn ..... 370
11.5 Support-Vector Regression (SVR) ..... 374
12 Naive Bayes Classification ..... 377
12.1 Principal Approach ..... 377
12.2 Estimating Label Probabilities ..... 378
12.3 Estimating Classes' Feature Probabilities ..... 378
12.4 Naive Bayes Classification with Scikit-Learn ..... 380
12.5 Related Projects ..... 380
13 Text Classification ..... 381
13.1 Preprocessing Text Data ..... 381
13.2 Training ..... 401
III Exercises ..... 435
14 Data Visualization ..... 437
14.1 Matplotlib Basics ..... 437
14.2 Advanced Matplotlib ..... 439
IV Projects ..... 441
15 Weather ..... 443
15.1 DWD Open Data Portal ..... 443
15.2 Getting Forecasts ..... 445
15.3 Climate Change ..... 447
15.4 Weather Animation ..... 448
16 MNIST Character Recognition ..... 451
16.1 The xMNIST Family of Data Sets ..... 451
16.2 Load QMNIST ..... 453
16.3 QMNIST Feature Reduction ..... 455
16.4 PCA and ANN for QMNIST ..... 456
16.5 CNN for QMNIST ..... 456
16.6 CNN Analysis for QMNIST ..... 456
16.7 IBAN recognition ..... 457
16.8 SVM for QMNIST ..... 459
17 Public Transport ..... 461
17.1 Get Data and Set Up the Environment ..... 461
17.2 Find Connections ..... 466
17.3 Interactive Map ..... 468
18 Corona Deaths ..... 471
18.1 Get some Data ..... 471
18.2 Load Data ..... 471
18.3 Death Rates ..... 472
19 Chemnitz Trees ..... 473
19.1 Download and Cleaning ..... 473
19.2 Short Names ..... 473
19.3 Extract Information ..... 474
19.4 Presentation ..... 474
20 Forged Banknotes ..... 475
20.1 Detecting Forgery with k-NN ..... 475
20.2 Quality Measures ..... 477
20.3 Hyperparameter Optimization ..... 479
20.4 Decision Tree ..... 479
20.5 Random Forest ..... 480
20.6 Support-Vector Machine ..... 481
20.7 Naive Bayes Classification ..... 481
21 House Prices ..... 483
21.1 House Prices GUI ..... 483
21.2 House Prices ANN ..... 484
21.3 A Random Forest for House Prices ..... 485
22 Hyperparameter Optimization for Cats and Dogs ..... 487
23 Blogs ..... 489
23.1 Blog Author Classification (Training) ..... 489
23.2 Blog Author Classification (Test) ..... 492
V Computer Science ..... 495
24 Cloud Computing ..... 497
24.1 Different Approaches for Remote Execution ..... 497

## Data Science and

## Artificial Intelligence



This book covers a wide range of topics in data science and artificial intelligence. It's an attempt to provide selfcontained 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, February 2024

[^0]
## Part I

## Data Visualization

## 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 ${ }^{8}$, providing lots of plot types and additional visualization features. It integrates well with and is the Swiss army knife of visualization tools.

- Matplotlib Basics (page 5)
- 3D Plots (page 36)
- Animations (page 41)
- Seaborn (page 46)
- Maps (page 50)

Related exercises:

- Matplotlib Basics (page 437)
- Advanced Matplotlib (page 439)

Related projects:

- Chemnitz Trees (page 473)
- Weather (page 443)
- Weather Animation (page 448)


### 1.1 Matplotlib Basics

Matplotlib provides two interfaces for plotting:

- MATLAB ${ }^{9}$ like state-based interface,
- object-oriented interface.


### 1.1.1 State-Based Plotting

The state-based interface is known as pyplot ${ }^{10}$.
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
```

[^1]
## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning

We first need a place where all the plotting is done. Matplotlib calls this a figure. To create one call figure ${ }^{11}$.

```
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 ${ }^{12}$.

```
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 ${ }^{13}$. 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.

[^2]```
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()
```

Our first plot


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 xlabe $l^{14}$, ylabel $l^{15}$ or title ${ }^{16}$. 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.

### 1.1.2 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 ${ }^{17}$ object, which encapsulates the coordinate system and all its surroundings. Then we can add a Line2D ${ }^{18}$ object via Axes. plot ${ }^{19}$.

[^3]```
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 ${ }^{20}$ 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()
```

[^4]
## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning



Details on pyplot. subplots will be given below.

### 1.1.3 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 ${ }^{21}$.

```
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()
```

[^5]
## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning



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. subplot $s^{22}$ 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 ${ }^{23}$. This method returns a GridSpec ${ }^{24}$ 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()
```

[^6]

Indexing a GridSpec object returns a SubplotSpec ${ }^{25}$ object which can be passed to Axes.add_subplot.
Subplots can be nested with SubplotSpec.subgridspec ${ }^{26}$. This method return a GridSpecFromSubplotSpec ${ }^{27}$ object for which indexing returns Subplot Spec 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()
```

[^7]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning

Note: The Figure. suptitle ${ }^{28}$ 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()
```

[^8]


## Data Science and Artificial Intelligence for Undergraduates

### 1.1.4 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 ${ }^{29}$ scaling and limits for both axes can be set at once. The Axes methods set_xlim ${ }^{30}$, set_ylim ${ }^{31}$, set_xscale ${ }^{32}$, set_yscale ${ }^{33}$ 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)

[^9]```
plt.show()
```



## Tick Positions and Labels

To modify tick positions and labels Axes objects provide methods set_xticks ${ }^{34}$, set_yticks ${ }^{35}$, set_xticklabels ${ }^{36}$, set_yticklabels ${ }^{37}$.

```
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()
```

[^10]
## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning



Matplotlib distinguishes minor and major ticks. Passing the parameter minor with value True or Fal se (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 ${ }^{38}$. 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 ${ }^{39}$. 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()
```

[^11]

## Different Scales

Sometimes one wants to have two plots with different y axis limits in one figure. This can be achieved with Axes . twiny ${ }^{40}$ (there is also a twinx ${ }^{41}$ ). 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()
```

[^12]

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 ${ }^{42}$, 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()
```

[^13]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning

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.

### 1.1.5 Colors and Colorbars

### 1.1.6 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: ${ }^{43}$, without prefix ${ }^{44}$, with prefix 'xkcd:' ${ }^{45}$ )

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])
```

[^14]```
for }n\mathrm{ 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 ${ }^{46}$ 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 ${ }^{47}$ 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 ${ }^{48}$ ). But custom colormaps can be created, too.

## Colorbars

Colorbars visualize the connection between data values and colors. They can be created with pyplot. colorbar ${ }^{49}$ or Figure.colorbar ${ }^{50}$.

```
fig, ax = plt.subplots(figsize=(8, 4))
plt.scatter(x, y, c=z, cmap='jet')
plt.colorbar()
plt.show()
```

[^15]Data Science and Artificial Intelligence for Undergraduates
Volume 2: Data Visualization, Supervised Learning

fig, $a x=$ plt.subplots(figsize=(8, 4))
scatter_plot $=$ ax.scatter ( $x, y, c=z, \quad$ map='jet')
fig. colorbar(scatter_plot, ax=ax, orientation='vertical')
plt.show()


### 1.1.7 Legends and Text

## Legends

Simple legends can be generated automatically with Axes. legend ${ }^{51}$ 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)

[^16]```
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 $\mathrm{TeX}^{52}$ 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 $\frac{s \coot o \codot m \codot e}{m+a+t+h}$')
plt.show()
```

[^17]

Full LaTeX ${ }^{53}$ typesetting is available, too, but requires an external LaTeX installation.

## Annotations

To add text to a plot use Axes . text ${ }^{54}$ 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()
```

[^18]
## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning



More advanced annotation is provided by Axes . annotate ${ }^{55}$.

```
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':
    \hookrightarrow'->'})
plt.show()
```

[^19]

### 1.1.8 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 ${ }^{56}$ 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()
```

[^20]

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 ${ }^{57}$ or at this example ${ }^{58}$ to get an idea of what shapes are provided next to rectangles.

### 1.1.9 Raster Images

Raster images can be represented in different ways:

- ( $\mathrm{m}, \mathrm{n}$ ) dimensional NumPy array (each entry is interpreted as grey level or, more generally, as argument to a colormap)
- ( $\mathrm{m}, \mathrm{n}, 3$ ) 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 ${ }^{59}$. To load an image from a file use matplotlib.image. imread ${ }^{60}$ or pyplot.imread ${ }^{61}$.

```
img = plt.imread('tux.png')
print(img.shape)
fig, ax = plt.subplots()
ax.imshow(img)
plt.show()
```

(943, 800, 4)

[^21]

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.

### 1.1.10 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 ${ }^{62}$ (line plots)
- scatter ${ }^{63}$ (point clouds)
- bar $^{64}$ (bar plots)
- pie ${ }^{65}$ (pie charts)
- hist ${ }^{66}$ (histograms)
- contour ${ }^{67}$ (contour plots)

More plot types are listed in Axes's documentation ${ }^{68}$.
To get an idea of what is possible with Matplotlib have a look at the gallery ${ }^{69}$.

## Saving Plots and Postprocessing

Plots can be saved to image files with Figure. savefig ${ }^{70}$.

```
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)
```

[^22]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning

After saving a figure postprocessing with external tools is possible. For raster images use, for instance, GIMP ${ }^{71}$. For vector graphics Inkscape ${ }^{72}$ is a good choice. Next to addition of advanced graphical effects postprocessing also includes composing several plots to factsheets.

### 1.2 3D Plots

Matplotlib comes without native support for 3-dimensional plots. But there is an extension ('toolkit') for 3D plots: Mplot3d ${ }^{73}$. This toolkit is part of the Matplotlib standard installation.

```
import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d as plt3d
```


### 1.2.1 Basics

For 3d plotting we need an Axes $3 D^{74}$ object. This can either be created explicitly by plt3d.axes 3 d . 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)))
```

[^23]

Type of ax: <class 'mpl_toolkits.mplot3d.axes3d.Axes3D'>

Now ax is of type Axes3D instead of Axes.
Plotting is very similar to 2 d 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()
```


## Data Science and Artificial Intelligence for Undergraduates



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 ${ }^{75}$ 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()
```

[^24]

### 1.2.2 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=
    G'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 ${ }^{76}$.

```
%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()
```



[^25]
### 1.2.3 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()
```



### 1.3 Animations

Matplotlib comes with different techniques for animated plots. Here we concentrate on FuncAnimation ${ }^{77}$ objects.

```
%matplotlib widget
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as anim
```

[^26]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised LearningAn 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.

### 1.3.1 Animations without Blitting

For figures with only few animated objects updating only these objects is efficient and yields readable code.

```
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 \%matplot lib 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()
```

```
    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>

### 1.3.2 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>

### 1.3.3 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')
```


### 1.4 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 ${ }^{78}$, a fictional television character. See also issue \#229 in Seaborns Github repository ${ }^{79}$.

```
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()
```

[^27]
## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning



### 1.4.1 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 ${ }^{80}$ and at the Seaborn gallery ${ }^{81}$.

```
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
```

[^28]```
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 |
| $\ldots$ | $\ldots$. | $\ldots$ | $\ldots$ | $\ldots$ |
| 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]


## Data Science and Artificial Intelligence for Undergraduates

### 1.5 Maps

There exist several Python modules to plot maps with Matplotlib. Here we cover Cartopy ${ }^{82}$ and Mplleaflet ${ }^{83}$. Cartopy is good for schematic maps and supports different coordinate and mapping systems. Mplleaflet combines Matplotlib drawings with interactive maps based on OpenStreetMap ${ }^{84}$ data.
In a separate chapter we'll meet Folium ${ }^{85}$ allowing for more advanced interactive maps.

### 1.5.1 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 ${ }^{86}$ 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 ${ }^{87}$. Note that GeoAxes is derived from the usual Matplotlib

[^29]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 ${ }^{88}$. Detail level of coastlines can be controlled with resolution argument, which has to be 110 m (default), 50 m or 10 m . 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()
```



### 1.5.2 Adding more Content

Cartopy gets its data from Natural Earth ${ }^{89}$, 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 ${ }^{90}$.

[^30]
## Data Science and Artificial Intelligence for Undergraduates

Volume 2: Data Visualization, Supervised Learning

```
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 ${ }^{91}$. Look at the filenames in the download links. Arguments other than name, resolution and category are passed on to Matplotlib.

[^31]
## Data Science and Artificial Intelligence for Undergraduates

### 1.5.3 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())
```

```
ax.coastlines()
lat1 = 40
lon1 = -90
lat2 = 30
1on2 = 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 ${ }^{92}$ we reset limits to show the whole map.

### 1.5.4 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 ${ }^{93}$ 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 ${ }^{94}$ and this solution ${ }^{95}$.

[^32]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning```
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)
```

[^33]
## PLOTLY

Plotly ${ }^{96}$ is a relatively new JavaScript based plotting library for Python and several other languages ${ }^{97}$. 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 ${ }^{98}$.

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.

```
import numpy as np
import pandas as pd
```


### 2.1 Plotly Express

The Plotly express ${ }^{99}$ interface to Plotly provides commands for complex standard tasks:

```
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=
    \hookrightarrow'result', size='result', width=800, height=600)
fig.show()
```

    <IPython.core.display.HTML object>
    [^34]
## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning

### 2.2 Graph Object Interface

For more advanced usage Plotly provides the graph object interface ${ }^{100}$. Here we first have to create a Figure ${ }^{101}$ 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>
    
### 2.3 Exporting Figures

Plotly plots can easily be exported to HTML with Figure.write_html ${ }^{102}$ :

```
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 ${ }^{103}$ from Plotly's GitHub repository.
Defautl behavior is that the exported file is opened automatically after export. To prevent this, use auto_open=False.

[^35]
## FOLIUM

Folium ${ }^{104}$ is a Python package for generating interactive maps. It's based on OpenStreetMap ${ }^{105}$ and the Leaflet ${ }^{106}$ JavaScript library.

```
import folium
```

Related projects:

- Public Transport (page 461)
- Interactive Map (page 468)


### 3.1 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 ${ }^{107}$.

[^36]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning
### 3.2 Advanced Features

Of particular interest for data science purposes are:

- image overlays ${ }^{108}$ (position raster image on a map)
- marker clusters ${ }^{109}$ (place many markers on a map)
- fast marker clusters ${ }^{110}$ (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 ${ }^{111}$ :

```
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>
    
### 3.3 Exporting a Map

To generate an HTML file containing the interactive Folium map, call save ${ }^{112}$. The save method is inherited by Foliums Map from branca.elements.Element as we see from Map's documentation ${ }^{113}$. Branca ${ }^{114}$ 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' ${ }^{115}$ :

```
m.get_root().html.add_child(folium.Element(
'''
<div style="position: absolute; z-index: 10000; top: 10px; left: 60px;
\hookrightarrowbackground: white; padding:5px 10px;">
<span style="font-size: 20pt; font-weight: bold; color: black;">Some Title</span>
4<br/>
<span style="font-size: 10pt; color: blue;">some text</span>
```

(continues on next page)

[^37]```
</div>
''
))
display(m)
m.save('folium.html')
    <folium.folium.Map at 0x7f203ef4fe80>
```


## Part II

## Supervised Learning

## MACHINE LEARNING OVERVIEW

Machine learning is the core technique behind data science in general and todays artificial intelligence. Revisit 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 ${ }^{116}$ 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 ${ }^{117}$ or Stable Diffusion ${ }^{118}$ 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.

[^38]
## GENERAL CONSIDERATIONS

Before we consider concrete classes of supervised learning methods, we have a look at basic principles common to (almost) all methods.

- Problem and Workflow (page 67)
- Introductory Example ( $k-N N$ ) (page 71)
- Quality Measures (page 82)
- Scaling (page 96)
- Feature Reduction (page 97)
- Hyperparameter Tuning (page 104)

Related projects:

- Forged Banknotes (page 475)
- Detecting Forgery with $k$ - NN (page 475)
- Quality Measures (page 477)
- Hyperparameter Optimization (page 479)
- QMNIST Feature Reduction (page 455)


### 5.1 Problem and Workflow

Here we introduce basic terminology used throughout the book and we present major problems arising when implementing supervised learning methods.

### 5.1.1 Statement of the Problem

## Abstract Formulation

Let $f: X \rightarrow Y$ 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 $f_{\text {approx }}$ 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 $y_{1}:=f\left(x_{1}\right), \ldots, y_{n}:=f\left(x_{n}\right)$ are called training labels. The pairs $\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)$ are the training examples or training data. Ultimately, the function $f_{\text {approx }}$ will be represented by a Python program or a Python function: it takes an input and yields some output.


Fig. 5.1: Given the a mapping $f$ on finitely many points supervised learning aims at finding a mapping $f_{\text {approx }}$ 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 $f_{\text {approx }}$ 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. 5.2: The sample $x_{1}$ is a vector with 784 components, one component (or feature) per pixel.

# Data Science and Artificial Intelligence for Undergraduates <br> Volume 2: Data Visualization, Supervised Learning 

## 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 $f_{\text {approx }}$ 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.

### 5.1.2 Principal Steps in Supervised Learning

The starting point for all supervised machine learning methods is a finite collection of correctly labeled inputs:

$$
\left\{\left(x_{1}, y_{1}\right),\left(x_{2}, y_{2}\right), \ldots,\left(x_{n}, y_{n}\right)\right\}
$$

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 $f_{\text {approx }}$, 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.

### 5.1.3 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 ${ }^{119}$ 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 ${ }^{120}$ 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 ${ }^{121}$ 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 ${ }^{122}$ is an alternative to Keras, which provides a more object-oriented programming interface.

[^39]
### 5.1.4 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 . Scikit-Learn supports one-hot encoding, too, with its OneHotEncoder ${ }^{123}$ 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.

### 5.2 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).

### 5.2.1 Method

$\mathrm{k}-\mathrm{NN}$ 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.

### 5.2.2 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 96).

## 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.

[^40]
## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

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 $\mathrm{k}-\mathrm{NN}$ 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.

### 5.2.3 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

$$
y_{\text {pred }}=\frac{\sum_{\kappa=1}^{k} \frac{1}{d_{\kappa}} y_{\kappa}}{\sum_{\kappa=1}^{k} \frac{1}{d_{\kappa}}}
$$

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:

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 $n$.

### 5.2.4 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 ${ }^{124}$ and k-NN classification in KNeighborsClassifier ${ }^{125}$. 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.

[^41]
## Regression Example

For demonstration we use a small one-feature synthetic data set.

```
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)
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 ${ }^{126}$ 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()
```

[^42]

In the evaluation phase we calculate some suitable metric on the test set. Here we use mean squared error $(\mathrm{MSE})^{127}$.

```
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 ${ }^{128}$ 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)

[^43]```
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)
```

```
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',ь
    4edgecolor='black', zorder=1000)
ax.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1], c='#00ff00',
    чedgecolor='black', zorder=1000)
ax.scatter(X_test[y_test == 0, 0], X_test[y_test == 0, 1], c='#ff0000', edgecolor=
    \hookrightarrow'white', zorder=1000)
ax.scatter(X_test[y_test == 1, 0], X_test[y_test == 1, 1], c='#00ff00', edgecolor=
    G'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)))
```



```
test accuracy: 0.9166666666666666
train accuracy: 0.9214285714285714
```

Here we used accuracy ${ }^{129}$ as quality measure.

### 5.2.5 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.

[^44]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 $\mathbf{k}$

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 $m=1 \mathrm{it}$ '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 ${ }^{130}$ ), we have to set $k=2^{m}$. Such heavy influence of space dimension is sometimes called the curse of dimensionality.


Fig. 5.3: Even if a point of interest only has neighbors at the vertices of a surrounding cube, number of neighbors grows exponentially with dimension.

[^45]
## 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,

$$
\frac{k}{n} \approx \text { volume of hypersphere } \quad \text { or } \quad k \approx n \cdot \text { volume of hypersphere. }
$$

Have a look at the volume of the hypersphere (Wikipedia) ${ }^{131}$ 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 ${ }^{132}$ 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()
```

[^46]

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 $m \rightarrow \infty$.
For spheric melons similar computations yield exactly the same result $0.9^{m}$


Fig. 5.4: 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 30dimensional 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 97).

## Important: Math matters!

Human imagination fails in high dimensions. Math still works.

### 5.3 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.

### 5.3.1 Quality Measures for Regression Problems

For regression tasks all quality measures have the form

$$
\frac{1}{n} \sum_{l=1}^{n} l\left(f_{\text {approx }}\left(x_{l}\right), y_{l}\right)
$$

where $\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)$ are the samples (inputs and correct labels) on which to compute $f_{\text {approx }}$ 's predition quality and $l: \mathbb{R} \rightarrow \mathbb{R}$ 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:

$$
\frac{1}{n} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2} .
$$


<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 ${ }^{133}$ 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):

$$
\frac{1}{n} \sum_{l=1}^{n}\left|f_{\text {approx }}\left(x_{l}\right)-y_{l}\right|
$$

[^47]
<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 ${ }^{134}$ 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

$$
\frac{1}{n} \sum_{l=1}^{n} h\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)
$$

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 $\varepsilon$ independent from the data's range.

[^48]
<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.

### 5.3.2 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 probabilitylike 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 $t \in(0,1)$ :

$$
\begin{array}{lll}
p \leq t & \Rightarrow & \text { class A }, \\
p>t & \Rightarrow & \text { class } .
\end{array}
$$

## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised LearningNext 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 $\left[0, \frac{1}{2}\right)$ and positive numbers to $\left(\frac{1}{2}, 1\right]$. The sigmoid function

$$
z \mapsto \frac{1}{1+\mathrm{e}^{-z}}, \quad z \in \mathbb{R}
$$

shows all those properties.
sigmoid function


Note that 0 and 1 do not belong to the range of the sigmoid function, but

$$
\lim _{z \rightarrow-\infty} \frac{1}{1+\mathrm{e}^{-z}}=0 \quad \text { and } \quad \lim _{z \rightarrow \infty} \frac{1}{1+\mathrm{e}^{-z}}=1
$$

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

$$
\begin{aligned}
& \text { probability for class } \mathrm{A}:=\frac{1}{1+\mathrm{e}^{-z}} \\
& \text { probability for class } \mathrm{B}:=1-\frac{1}{1+\mathrm{e}^{-z}}=\frac{\mathrm{e}^{-z}}{1+\mathrm{e}^{-z}}=\frac{1}{1+\mathrm{e}^{z}}
\end{aligned}
$$

sigmoid-based probabilities for 2 classes


For multiclass classification with classes $1, \ldots, C$ and class scores $z_{1}, \ldots, z_{C}$ we may use the softmax function:

$$
\text { probability for class } i:=\frac{\mathrm{e}^{z_{i}}}{\mathrm{e}^{z_{1}}+\cdots+\mathrm{e}^{z_{C}}}, \quad i=1, \ldots C .
$$

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.

### 5.3.3 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:

$$
\text { accuracy }:=\frac{\text { correct predictions }}{\text { number of samples }} \text {. }
$$

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;

$$
5 \text { in } \mathrm{A}, \quad 10 \text { in } \mathrm{B}, \quad 85 \text { in } \mathrm{C} .
$$

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

$$
50 \text { in } \mathrm{A}, \quad 40 \text { in } \mathrm{B}, \quad 10 \text { in } \mathrm{C} \text {, }
$$

then accuracy drops to 10 per cent, although it's still the same classifier. With a balanced test set, say

$$
33 \text { in } \mathrm{A}, \quad 33 \text { in } \mathrm{B}, \quad 34 \text { in } \mathrm{C} \text {, }
$$

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 ${ }^{135}$ 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

$$
\text { balanced accuracy }:=\frac{1}{C} \sum_{i=1}^{C} \frac{\text { number of samples correctly labeled as class } i}{\text { total number of samples in class } i} .
$$

[^49]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

$$
\begin{array}{rll}
5 \text { in } \mathrm{A}, \quad 10 \text { in } \mathrm{B}, \quad 85 \text { in } \mathrm{C} & \Rightarrow & \text { bal. acc. }=\frac{1}{3}\left(\frac{0}{5}+\frac{0}{10}+\frac{85}{85}\right)=\frac{1}{3} . \\
50 \text { in } \mathrm{A}, \quad 40 \text { in } \mathrm{B}, \quad 10 \text { in } \mathrm{C} & \Rightarrow & \text { bal. acc. }=\frac{1}{3}\left(\frac{0}{50}+\frac{0}{40}+\frac{10}{10}\right)=\frac{1}{3} . \\
33 \text { in } \mathrm{A}, \quad 33 \text { in } \mathrm{B}, \quad 34 \text { in } \mathrm{C} & \Rightarrow & \text { bal. acc. }=\frac{1}{3}\left(\frac{0}{33}+\frac{0}{33}+\frac{34}{34}\right)=\frac{1}{3} .
\end{array}
$$

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 ${ }^{136}$ 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 $1, \ldots, C$ 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 ${ }^{137}$ 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 $y_{1}, \ldots, y_{n}$ and the predicted probabilities that sample $l$ belongs to class $y_{l}$ (its true class) by $p_{1}, \ldots, p_{n}$. Then

$$
\log \operatorname{loss}:=-\frac{1}{n} \sum_{l=1}^{n} \log p_{l}=-\log \left(\left(\prod_{l=1}^{n} p_{l}\right)^{\frac{1}{n}}\right) .
$$

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 $\infty$ 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.

[^50][^51]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

$$
y_{l, i}:=\left\{\begin{array}{ll}
1, & \text { if } y_{l}=i, \\
0, & \text { else }
\end{array} \quad \text { for } \quad l=1, \ldots, n \quad \text { and } \quad i=1, \ldots, C\right.
$$

(one-hot encoding) and denote by $p_{l, i}$ the predicted probability that sample $l$ belongs to class $i$. Then

$$
\log \operatorname{loss}=-\frac{1}{n} \sum_{l=1}^{n} \sum_{i=1}^{C} y_{l, i} \log p_{l, i},
$$

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

$$
\text { binary } \log \operatorname{loss}=-\frac{1}{n} \sum_{l=1}^{n}\left(y_{l} \log p_{l, 1}+\left(1-y_{l}\right) \log \left(1-p_{l, 1}\right)\right) .
$$

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 ${ }^{138}$ 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

[^52]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 ${ }^{139}$ and recall_score ${ }^{140}$ in its metrics module.

## F1-score

F1-score (or F-score) combines precision and recall by taking their harmonic mean ${ }^{141}$ :

$$
\text { F1-score }=\frac{2}{\frac{1}{\text { precision }}+\frac{1}{\text { recall }}}=\frac{2 \cdot \text { precision } \cdot \text { recall }}{\text { precision }+ \text { recall }} .
$$

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 $f 1 \_s c o r e^{142}$ 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 ${ }^{143}$
- roc_curve ${ }^{144}$
- RocCurveDisplay.from_predictions ${ }^{145}$


## 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

$$
\operatorname{FPR}(t):=\frac{\text { number of negative samples with } p_{l}>t}{\text { total number of negative samples }}
$$

and the true positive rate (precision)

$$
\operatorname{TPR}(t):=\frac{\text { number of positive samples with } p_{l}>t}{\text { total number of positive samples }}
$$

Then the ROC 'curve' is the (finite) set of points

$$
\{(F P R(t), T P R(t)): t \in(0,1)\} .
$$

Often these points are connected by straight lines and the resulting curve is denoted as ROC curve. AUC is the area under that curve.

[^53]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised LearningExample: 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

$$
\mathrm{AUC}=8 \cdot \frac{1}{3} \cdot \frac{1}{4}=\frac{2}{3}
$$

## 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 $\mathrm{AUC}^{146}$, 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-

[^54]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 influcence 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.

### 5.4 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 ( $k-N N$ ) (page 71). More will follow when discussing more machine learning techniques. Here we have a look at two standard approaches to scaling: normalization and standardization.

### 5.4.1 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

$$
x_{\mathrm{new}}=\frac{x_{\mathrm{old}}-a}{b-a}
$$

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 ${ }^{147}$ 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 trans form 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:

```
from sklear.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:

```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[^55]
### 5.4.2 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 $\mu$ is

$$
\mu=\frac{1}{n} \sum_{l=1}^{n} x_{l}
$$

and standard deviation $\sigma$ is

$$
\sigma=\sqrt{\frac{1}{n} \sum_{l=1}^{n}\left(x_{l}-\mu\right)^{2}} .
$$

Corresponding transform reads

$$
x_{\text {new }}=\frac{x_{\text {old }}-\mu}{\sigma} .
$$

From the mathematical statistics view we are slighlt imprecise here. Our $\mu$ 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 $\sigma$. But in addition, our estimate $\sigma$ 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 ${ }^{148}$ in the preprocessing module for standardizing date. Usage is exactly the same as described above for normalization.

### 5.4.3 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. 5.5: 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.

### 5.5 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.

[^56]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.

### 5.5.1 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. 5.6: 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.

### 5.5.2 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 ${ }^{149}$ 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.

### 5.5.3 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, $\xi_{1}$ contains all values of the first feature, $\xi_{2}$ all values of the second feature, and so on; in formulas:

$$
\xi_{1}=\left[\begin{array}{c}
x_{1}^{(1)} \\
\vdots \\
x_{n}^{(1)}
\end{array}\right], \quad \ldots, \quad \xi_{m}=\left[\begin{array}{c}
x_{1}^{(m)} \\
\vdots \\
x_{n}^{(m)}
\end{array}\right] .
$$

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.

[^57]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

$$
|X a| \rightarrow \max _{|a|=1} .
$$

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 $\left\{u_{1}, \ldots, u_{n}\right\}$ and $\left\{v_{1}, \ldots, v_{m}\right\}$ 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:

- $v_{\kappa}=a_{\kappa}^{*}\left(\right.$ normalized eigenvectors of $\left.X^{\mathrm{T}} X\right)$,
- $u_{\kappa}=\frac{X a_{\kappa}^{*}}{\left|X a_{\kappa}^{*}\right|}$ (normalized principal components or eigenvectors of $X X^{\mathrm{T}}$ ),
- $s_{\kappa}=\left|X a_{\kappa}^{*}\right|$ (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,

$$
A=V, \quad \tilde{X}=X A=U S
$$

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).

## Data Science and Artificial Intelligence for Undergraduates

## PCA with Scikit-Learn

Scikit-Learn implements a PCA ${ }^{150}$ 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 trans form 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.

```
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.

```
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,
4 0.2]], n)
# plot data
```

(continues on next page)

[^58]```
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()
```



### 5.5.4 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.

### 5.6 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.

### 5.6.1 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 ${ }^{151}$ 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.
    \hookrightarrownormal(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()
```

[^59]

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])
```


### 5.6.2 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,
    4random_state=0)
print(y_train.size, y_val.size, y_test.size)
```

500300200

Now steps are as follows:

- specify grid for each hyperparameter,
- generate all combinations of parameter values with ParameterGrid ${ }^{152}$ 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)

[^60]```
    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


### 5.6.3 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 $\nu$ more or less equally sized subsets. Then $\nu$ different training validation splits are considered: one of the $\nu$ 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 $\nu$ 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 ${ }^{153}$. 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 ${ }^{154}$.
- 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_
    ५error', 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))
```

[^61]```
{'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.

### 5.6.4 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 Randomized.SearchCV ${ }^{155}$.

### 5.6.5 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 ${ }^{156}$ 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,v
    420), 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=
    \hookrightarrow'training scores')
ax.plot(train_sizes, np.mean(val_scores, axis=1), '-or', markersize=3, label=
    \hookrightarrow'validation scores')
ax.legend()
ax.set_xlabel('size of training set')
ax.set_ylabel('score')
plt.show()
```

[^62]

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 ${ }^{157}$ 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=
    G'training scores')
ax.plot(param_range, np.mean(val_scores, axis=1), '-or', markersize=3, label=
    \hookrightarrow'validation scores')
ax.legend()
ax.set_xlabel('neighbors')
ax.set_ylabel('score')
plt.show()
```

[^63]

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.

## 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.

- Approach (page 113)
- Regularization (page 129)
- Worked Example: House Prices I (page 142)
- Worked Example: House Prices II (page 175)
- Outliers (page 184)

Related projects:

- House Prices GUI (page 483)


### 6.1 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.

### 6.1.1 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 $f_{\text {approx }}$.
Given $n$ training samples $\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)$ we want to choose $a_{0}, a_{1}, \ldots, a_{m}$ such that

$$
f_{\text {approx }}\left(x_{l}\right) \approx y_{l} \quad \text { for } l=1, \ldots, n
$$

To achive this we solve the minimization problem

$$
\frac{1}{n} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2} \rightarrow \min _{a_{0}, \ldots, a_{m}}
$$

The function $g$ defined by

$$
g(u, v)=\frac{1}{n} \sum_{l=1}^{n}\left(u_{l}-v_{l}\right)^{2}
$$

## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

is a so called loss function expressing the distance between two vectors $u=\left(u_{1}, \ldots, u_{n}\right)$ and $v=\left(v_{1}, \ldots, v_{n}\right)$. There are several other loss functions in machine learning like mean absolute error and Huber loss, see Quality Measures (page 82). For the moment we content ourselves with the simplest one, the mean squared error.
Note that we cannot expect $f_{\text {approx }}\left(x_{l}\right)=y_{l}$, 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

$$
h\left(a_{0}, \ldots, a_{m}\right):=\frac{1}{n} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2}=\frac{1}{n} \sum_{l=1}^{n}\left(a_{0}+\sum_{k=1}^{m} a_{k} x_{l}^{(k)}-y_{l}\right)^{2} \rightarrow \min _{a_{1}, \ldots, a_{m}}
$$

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

$$
h\left(a_{0}, \ldots, a_{m}\right)=\frac{1}{n} \sum_{l=1}^{n}\left(\sum_{k=0}^{m} a_{k} x_{l}^{(k)}-y_{l}\right)^{2} .
$$

Taking the derivative with respect to $a_{i}$ for $i=0,1, \ldots, m$ we obtain

$$
\begin{aligned}
\frac{\partial}{\partial a_{i}} h\left(a_{1}, \ldots, a_{m}\right) & =\frac{1}{n} \sum_{l=1}^{n} 2\left(\sum_{k=0}^{m} a_{k} x_{l}^{(k)}-y_{l}\right) x_{l}^{(i)} \\
& =\frac{2}{n} \sum_{l=1}^{n} \sum_{k=0}^{m} a_{k} x_{l}^{(k)} x_{l}^{(i)}-\frac{2}{n} \sum_{l=1}^{n} y_{l} x_{l}^{(i)} \\
& =\frac{2}{n} \sum_{k=0}^{m} a_{k} \sum_{l=1}^{n} x_{l}^{(k)} x_{l}^{(i)}-\frac{2}{n} \sum_{l=1}^{n} y_{l} x_{l}^{(i)} .
\end{aligned}
$$

Thus, the gradient is zero if and only if

$$
\sum_{k=0}^{m} a_{k} \sum_{l=1}^{n} x_{l}^{(k)} x_{l}^{(i)}=\sum_{l=1}^{n} y_{l} x_{l}^{(i)} \quad \text { for } i=0,1, \ldots, m
$$

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,

$$
B:=\left[\begin{array}{cccc}
1 & x_{1}^{(1)} & \cdots & x_{1}^{(m)} \\
\vdots & \vdots & & \vdots \\
1 & x_{n}^{(1)} & \cdots & x_{n}^{(m)}
\end{array}\right]
$$

we see that the gradient of $h$ is zero if and only if

$$
B^{\mathrm{T}} B a=B^{\mathrm{T}} y
$$

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 ${ }^{158}$. The data set contains age, selling price, kilimeters driven and several other parameters of about 4000 cars.
The dataset has been obtained via Kaggle ${ }^{159}$ and is licenced under Database Contents License (DbCL) v1.0 ${ }^{160}$, including Open Database License (ODbL) v1.0 ${ }^{161}$.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
cars = pd.read_csv('cars.csv')
cars
```


[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)

[^64]| 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
    `'] ))
```

```
xmin = 0
xmax = 260000
ax.plot([xmin, xmax], [a_young[0] + a_young[1] * xmin, a_young[0] + a_young[1] * c
    4xmax], '-b', label='young')
ax.plot([xmin, xmax], [a_old[0] + a_old[1] * xmin, a_old[0] + a_old[1] * xmax], '-
\hookrightarrowr', label='old')
ax.set_xlabel('km')
ax.set_ylabel('price')
ax.legend()
plt.show()
```

    [ \(6.74201542 \mathrm{e}+05-1.68902485 \mathrm{e}+00\) ]
    [3.64966780e+05 1.81825528e-01]
    

Given the kilometers driven, the simple formula

$$
y=674202-1.69 x
$$

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

$$
y=364967+0.18 x
$$

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.

## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning

```
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(
```

```
    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

$$
y=769448-0.21 x_{1}-54786 x_{2}
$$

## 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 ${ }^{162}$ 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 ${ }^{163}$ in the Seaborn documentation for more details.

```
import seaborn as sns
sns.pairplot(cars_subset)
plt.show()
```

[^65]

Seaborn also provides linear regression directly. See Estimating regression fits ${ }^{164}$ for more details.

```
sns.regplot(x='year', y='selling_price', data=cars_subset)
    <Axes: xlabel='year', ylabel='selling_price'>
```

[^66]

## 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.

# Data Science and Artificial Intelligence for Undergraduates 

```
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
    
### 6.1.2 Linear Regression with Nonlinear Functions

Idea

We consider more general parameter dependent hypotheses of the form

$$
f_{\text {approx }}(x)=a_{1} \varphi_{1}\left(x^{(1)}, \ldots, x^{(m)}\right)+\cdots+a_{\mu} \varphi_{\mu}\left(x^{(1)}, \ldots, x^{(m)}\right)=\sum_{\kappa=1}^{\mu} a_{\kappa} \varphi_{\kappa}\left(x^{(1)}, \ldots, x^{(m)}\right)
$$

where $x=\left(x^{(1)}, \ldots, x^{(m)}\right)$ is a feature vector, $a_{1}, \ldots, a_{\mu}$ are the parameters of the hypothesis $f_{\text {approx }}$, and $\varphi_{1}, \ldots, \varphi_{\mu}$ are prescribed real-valued functions on $\mathbb{R}^{m}$.

Note that $f_{\text {approx }}$ 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

$$
\mu=3, \quad \varphi_{1}(x)=x^{2}, \quad \varphi_{2}(x)=x, \quad \varphi_{3}(x)=1 .
$$

## 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 $\mu$. On each subinterval we could do a linear regression, but require that resulting lines are connected at the interval boundaries.

$$
\mu=6
$$



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 $\kappa=1 \ldots, \mu$ :

$$
\varphi_{\kappa}(x)= \begin{cases}(\mu-1) x-\kappa+2, & \text { if } x \in\left[\frac{\kappa-2}{\mu-1}, \frac{\kappa-1}{\mu-1}\right] \\ -(\mu-1) x+\kappa, & \text { if } x \in\left[\frac{\kappa-1}{\mu-1}, \frac{\kappa}{\mu-1}\right] \\ 0, & \text { else. }\end{cases}
$$

$$
\mu=6
$$



The parameters $a_{1}, \ldots, a_{\mu}$ describe the high of the hats or the function values of $f_{\text {approx }}$ 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.
1d RBFs


```
<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

$$
\varphi_{1}(x)=\varphi\left(x-u_{1}\right), \quad \ldots, \quad \varphi_{\mu}(x)=\varphi\left(x-u_{\mu}\right)
$$

## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning
## 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 $\mu$ features by

$$
\tilde{x}_{1}^{(\kappa)}:=\varphi_{\kappa}\left(x_{1}\right), \quad \ldots, \quad \tilde{x}_{n}^{(\kappa)}:=\varphi_{\kappa}\left(x_{n}\right) \quad \text { for } \kappa=1, \ldots, \mu .
$$

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 ${ }^{165}$ class for creating transformer objects. The transformer object then provides a method fit_transform ${ }^{166}$ 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.

```
# 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.

```
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)

[^67]```
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.
    \hookrightarrowinspace(ymin, ymax, grid_size))
xyfeatures = np.stack((xcoords.reshape(-1), ycoords.reshape(-1)), axis=1)
zcoords = regression.predict(transformer.fit_transform(xyfeatures)).reshape(grid_
    \leftrightarrowsize, 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
    
### 6.2 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.

## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

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.

### 6.2.1 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!

### 6.2.2 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:

$$
\frac{1}{n} \sum_{k=1}^{n}\left(f_{\text {approx }}\left(x_{k}\right)-y_{k}\right)^{2},
$$

where $\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)$ 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
```

```
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
```

(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, $a x=p l t . s u b p l o t s()$
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))


$$
\left.\right]
$$

```
[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.

### 6.2.3 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.

```
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 $f_{\text {approx }}$ and the targets in the training data:

$$
\frac{1}{n} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2} \rightarrow \min _{a_{1}, \ldots, a_{\mu}},
$$

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 $\alpha \geq 0$ :

$$
\frac{1}{n} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2}+\alpha \frac{1}{\mu} \sum_{\kappa=1}^{\mu} a_{\kappa}^{2} \rightarrow \min _{a_{1}, \ldots, a_{\mu}}
$$

The regularization parameter $\alpha$ 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 ${ }^{167}$.

```
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)
```

[^68]```
# 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
```

```
fig, ax = plt.subplots()
ax.semilogx(alphas, errors, '-m')
ax.set_xlabel('$\\alpha$')
ax.set_ylabel('error')
plt.show()
```



For $\alpha$ close to zero overfitting leads to large errors. For large $\alpha$ 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 $\alpha$.

In practice we do not know the truth. But analogously to detecting overfitting we may find values for $\alpha$, where overfitting vanishes. We simply have to start with very small $\alpha$ leading to overfitting. Then we increase $\alpha$ 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))
```

```
    # 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 |
| :--- | ---: |
| $\left[\begin{array}{ll}{[0.07326089} & 0.10832048]\end{array}\right.$ |  |
| $[0.07075901$ | $0.10408418]$ |
| $[0.06954605$ | $0.10167556]$ |
| $[0.06871829$ | $0.10026634]$ |
| $[0.06773215$ | $0.09926158]$ |
| $[0.06630425$ | $0.09848887]$ |
| $[0.06443948$ | $0.09830744]$ |
| $[0.06251508$ | $0.09948163]$ |
| $[0.06100575$ | $0.10238347]$ |
| $[0.06000687$ | $0.10645098]$ |
| $[0.05927804$ | $0.1108613]$ |
| $[0.05859948$ | $0.11495806]$ |
| $[0.05785644$ | $0.11780122]$ |
| $[0.0569407$ | $0.11823884]$ |
| $[0.05579468$ | $0.11589774]$ |
| $[0.05456294$ | $0.11190603]$ |
| $[0.05352215$ | $0.10822899]$ |
| $[0.05277435$ | $0.10602967]$ |
| $[0.05218768$ | $0.10528655]$ |
| $[0.05163829$ | $0.10579957]$ |
| $[0.05114153$ | $0.10762232]$ |
| $[0.0507495$ | $0.11063545]$ |
| $[0.05043095$ | $0.11450807]$ |
| $[0.05011846$ | $0.11923569]$ |

```
```

[0.04980821 0.12506099]

```
```

[0.04980821 0.12506099]
[0.0495608 0.13166127]
[0.0495608 0.13166127]
[0.04941321 0.1379058 ]
[0.04941321 0.1379058 ]
[0.0493391 0.14279624]
[0.0493391 0.14279624]
[0.04929224 0.14614947]
[0.04929224 0.14614947]
[0.04923822 0.14819351]
[0.04923822 0.14819351]
[0.04915017 0.14892117]
[0.04915017 0.14892117]
[0.04899594 0.1478916 ]
[0.04899594 0.1478916 ]
[0.04873589 0.14449768]
[0.04873589 0.14449768]
[0.04834478 0.13855119]
[0.04834478 0.13855119]
[0.04785498 0.13111712]]

```
```

[0.04785498 0.13111712]]

```
```

Note that this way only values for $\alpha$ 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.

### 6.2.4 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

$$
\frac{1}{n} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2}+\alpha \frac{1}{3}\left(a_{0}^{2}+a_{1}^{2}+a_{2}^{2}\right) \rightarrow \min _{a_{0}, a_{1}, a_{2}}
$$

where $\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)$ are the training samples and $\alpha$ 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 $\alpha$ regardless of their range of values.

### 6.2.5 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):

$$
\alpha \frac{1}{\mu} \sum_{\kappa=1}^{\mu}\left|a_{\kappa}\right| .
$$

- Elastic Net:

$$
\alpha \frac{1}{\mu}\left(\beta \sum_{\kappa=1}^{\mu}\left|a_{\kappa}\right|+(1-\beta) \sum_{\kappa=1}^{\mu} a_{\kappa}^{2}\right),
$$

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 ${ }^{168}$, ElasticNet ${ }^{169}$.

### 6.3 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 ${ }^{170}$ (unreachable in 2023) already did this job at www.immobilienscout24.de ${ }^{171}$ and published the data set at www.kaggle.com ${ }^{172}$ (unreachable in 2023) under a Attribution-NonCommercial-ShareAlike 4.0 International License ${ }^{173}$.

```
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'
```


### 6.3.1 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 ${ }^{174}$ to None (that is, unlimited).

```
pd.set_option('display.max_columns', None)
data.head(10)
```

| Unnamed: | 0 | Price | Type | Living_space | Lot |
| ---: | ---: | ---: | ---: | ---: | ---: |
| 0 | 498000.0 | Multiple dwelling | 106.00 | 229.0 |  |
| 1 | 495000.0 | Mid-terrace house | 140.93 | 517.0 |  |
| 2 | 749000.0 | Farmhouse | 162.89 | 82.0 |  |
| 3 | 259000.0 | Farmhouse | 140.00 | 814.0 |  |
| 4 | 469000.0 | Multiple dwelling | 115.00 | 244.0 |  |
| 5 | 1400000.0 | Mid-terrace house | 310.00 | 860.0 |  |
| 6 | 3500000.0 |  | Duplex | 502.00 | 5300.0 |
| 7 | 630000.0 | Duplex | 263.00 | 406.0 |  |
| 8 | 364000.0 | Duplex | 227.00 | 973.0 |  |

(continues on next page)

[^69]| State | City | Place | Garages |  |
| :--- | ---: | ---: | ---: | ---: |
| 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 |
| Baden-Württemberg |  | Mannheim | Rheinau | 12.0 |

```
        Garagetype
Parking lot
Parking lot
        Garage
        Garage
        Garage
        Garage
Parking lot
        Garage
Parking lot
Parking lot
```

```
data.info()
```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10552 entries, 0 to 10551
Data columns (total 26 columns):

Price 10552 non-null float64
Type 10150 non-null object
Living_space 10552 non-null float64
Lot 10552 non-null float64
Usable_area 5568 non-null float64
Free_of_Relation 6983 non-null object
Rooms 10552 non-null float64
Bedrooms 6878 non-null float64
Bathrooms 8751 non-null float64
Floors 7888 non-null float64
Year_built 9858 non-null float64
Furnishing_quality 7826 non-null object
Year_renovated 5349 non-null float64
Condition 10229 non-null object
Heating 9968 non-null object
Energy_source 9325 non-null object
Energy_certificate 9797 non-null object
Energy_certificate_type 7026 non-null object
Energy_consumption 2433 non-null float64
Energy_efficiency_class 5733 non-null object
State 10551 non-null object
City 10551 non-null object
Place 10262 non-null object
8592 non-null float64
8592 non-null object
25 Garagetype 8592 non-nu
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',
    \hookrightarrow'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
    4')
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'],
```


## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning

(continued from previous page)

```
    dtype='object')
Index([' A ', ' A+ ', ' B ', ' C ', ' D ', ' E ', ' F ', ' G ', ' H '], dtype=
    G'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', 'E', '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

```
\begin{tabular}{llll}
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) &
\end{tabular}
```

(continued from previous page)

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}
    \hookrightarrow)
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']).
    cdropna())
```

    3479
    We should keep Bat hrooms, 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
    ```
Gas, Ol, 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
Ol, Holz
Öl, Kohle
```

```
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
    \hookrightarrow', '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_
    \hookrightarrowquality', 'Energy_source'])
data = data.dropna()
```


### 6.3.2 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 | $4.854000 \mathrm{e}+03$ | 4854.000000 | 4854.000000 | 4854.000000 | 4854.000000 |
| mean | $5.739566 e+05$ | 209.305740 | 1240.636904 | 7.051504 | 2.316028 |
| std | $5.880211 e+05$ | 118.252688 | 3806.518099 | 3.834865 | 1.595327 |
| min | $0.000000 \mathrm{e}+00$ | 0.000000 | 0.000000 | 1.000000 | 0.000000 |
| 25\% | $2.800000 \mathrm{e}+05$ | 135.000000 | 401.000000 | 5.000000 | 1.000000 |
| 50\% | $4.400000 \mathrm{e}+05$ | 180.000000 | 675.000000 | 6.000000 | 2.000000 |
| 75\% | $6.850000 \mathrm{e}+05$ | 248.000000 | 1042.000000 | 8.000000 | 3.000000 |
| max | $1.300000 \mathrm{e}+07$ | 1742.240000 | 143432.000000 | 84.000000 | 26.000000 |
|  |  |  |  |  |  |
| Flount | 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 ${ }^{175}$, 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()
```

[^70]

There seem to be same very small values.

```
data.loc[data['Price'] <= np.exp(7), :]
\begin{tabular}{rrrrrrr} 
& Price & Type & Living_space & Lot & Rooms & Bathrooms \\
6987 & 1.0 & Duplex & 459.0 & 2742.0 & 23.0 & 8.0
\end{tabular}
    4 Year_built 
    State Nary City Place Garages Garage_type
```

Those samples should be dropped because house prices below $\mathrm{e}^{7} \approx 1000$ 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 \} |  |
| :---: | :---: | :---: | :---: |
| $3.0 \quad 1900.0$ | 1900.0 | modernized |  |
| 1.02019 .0 | 2019.0 first | occupation |  |
| $3.0 \quad 1905.0$ | 1981.0 | refurbished |  |
| 3.01965 .0 | 2019.0 | modernized |  |
| Heating | State | City | Place |
| wood-pellet heating | Baden-Württemberg | Main-Tauber-Kreis | Ahorn |
| stove heating | Bayern | Regensburg (Kreis) | Hemau |
| stove heating | Nordrhein-Westfalen | Höxter (Kreis) | Höxter |
| stove heating | Rheinland-Pfalz | Neuwied (Kreis) | erdorf |


| Garages | Garage_type |
| ---: | ---: |
| 1.0 | Garage |
| 1.0 | Outside parking lot |
| 3.0 | Outside parking lot |
| 1.0 | Outside parking lot |

Lot size below $\mathrm{e}^{2}<8 \mathrm{~m}^{2}$ 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, :]
```


(continues on next page)


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, :]
```



```
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, :]
```



```
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 |



```
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()
```



## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning
## 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, :]
```



```
data = data.loc[data['Year_renovated'] <= 2020, :]
```

data.loc[:, 'Year_renovated'] = np.log(2021 - data['Year_renovated'].to_numpy())
sns.histplot(data['Year_renovated'])

```
plt.show()
```



## Garages Column

```
sns.histplot(data['Garages'])
plt.show()
```

Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning


```
data.loc[data['Garages'] >= 20, :]
```

|  | Price | e Type | Living_space | Lot | Rooms | Bathrooms |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 461 | 14.343193 | 3 Bungalow | 5.347108 | 88.932609 | 8.0 | 2.0 |
| 2579 | 14.220976 | 6 Duplex | 6.278521 | 18.027150 | 15.0 | 4.0 |
| 3761 | 15.009130 | 0 Bungalow | 6.476972 | 28.116716 | 10.0 | 2.0 |
| 3932 | 13.384728 | 8 Duplex | 5.375278 | $8 \quad 8.809714$ | 15.0 | 7.0 |
| 4175 | 14.066269 | 9 Duplex | 5.777652 | 26.884487 | 11.0 | 5.0 |
| 4243 | 11.407565 | 5 Duplex | 3.058707 | $7 \quad 7.342132$ | 1.0 | 1.0 |
| 7166 | 13.102161 | 1 Bungalow | 7.003065 | $5 \quad 9.239899$ | 12.0 | 3.0 |
| 7870 | 12.959844 | 4 Corner house | 5.828946 | 610.596635 | 14.0 | 2.0 |
| 9535 | 13.910821 | Duplex | 6.415097 | 78.097122 | 22.0 | 8.0 |
|  | Floors Y | Year_built | renovated | Condition |  | Heating |
| 461 | 3.0 | 3.332205 | 3.332205 m | modernized |  | ve heating |
| 2579 | 2.0 | 3.761200 | 2.397895 m | modernized | underfl | oor heating |
| 3761 | 3.0 | 5.293305 | 1.609438 | renovated |  | heat pump |
| 3932 | 4.0 | 4.510860 | 4.510860 m | maintained |  | ve heating |
| 4175 | 2.0 | 3.496508 | 0.000000 m | modernized |  | ove heating |
| 4243 | 4.0 | 3.912023 | 3.912023 m | modernized |  | heat pump |
| 7166 | 1.0 | 3.970292 | 2.639057 | modernized |  | gas heating |
| 7870 | 3.0 | 4.795791 | 1.609438 di | dilapidated |  | ove heating |
| 9535 | 3.0 | 3.091042 | 3.091042 m | modernized |  | ove heating |
|  |  | State |  |  | City | $\backslash$ |
| 461 | Baden-W | Württemberg |  | udwigsburg | Kreis) |  |
| 2579 |  | Bayern |  | Rottal-Inn | Kreis) |  |
| 3761 |  | Hessen |  | Frankfurt a | m Main |  |
| 3932 |  | Hessen | Darmstadt | dt-Dieburg | Kreis) |  |
| 4175 |  | Hessen | Limburg | g-Weilburg ( | Kreis) |  |
| 4243 |  | Hessen |  | Main-Taunus | -Kreis |  |
| 7166 | Nordrhein | n-Westfalen | Minden | n-Lübbecke ( | Kreis) |  |

(continues on next page)

Rheinland-Pfalz Altenkirchen (Westerwald) (Kreis)
Sachsen-Anhalt Salzlandkreis

| Place | Garages | Garage_type |
| ---: | ---: | ---: | :--- |
| Vaihingen an der Enz | 30.0 | Outside parking lot |
| Eggenfelden | 22.0 | Parking lot |
| Niederursel | 20.0 | Outside parking lot |
| Otzberg | 60.0 | Outside parking lot |
| Bad Camberg | 21.0 | Outside parking lot |
| Hattersheim am Main | 30.0 | Parking lot |
| Espelkamp | 65.0 | Parking lot |
| Birnbach | 50.0 | Outside parking lot |
| 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 Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning```
data = data.loc[~data['Condition'].isin(['as new', 'by arrangement']), :]
data['Condition'] = data['Condition'].cat.remove_categories(['as new', 'by`
    4arrangement'])
```


## 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^
    slot'
data.loc[data['Garage_type'] == 'Duplex lot', 'Garage_type'] = 'Outside parking_
    slot'
data.loc[data['Garage_type'] == 'Parking lot', 'Garage_type'] = 'Outside parkingr
    4lot'
data['Garage_type'] = data['Garage_type'].cat.remove_categories(['Car park lot',
    \hookrightarrow'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: int 64 |  |

### 6.3.3 Save Cleaned Data

We save cleaned data for future use.

```
data.to_csv(data_path.replace('.csv', '_preprocessed.csv'))
```


### 6.3.4 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 ${ }^{176}$. 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()
```



```
(continues on next page)
```

[^71](continued from previous page)


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_
    4size=0.2)
print(y_train.size, y_test.size)
```

3817955

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_
```


### 6.3.5 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.

### 6.3.6 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
```

```
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,
    property,
        _
        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: {:.Of} EUR'.format(y[0]))
```

```
predicted price: 115455 EUR
```


### 6.4 Worked Example: House Prices II

We try to improve prediction of house prices based on Erdogan Seref's ${ }^{177}$ (unreachable in 2023) German housing dataset from www.immobilienscout24.de ${ }^{178}$ published at www.kaggle.com ${ }^{179}$ (unreachable in 2023) under a Attribution-NonCommercial-ShareAlike 4.0 International License ${ }^{180}$.

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',
```

[^72]```
    'fixer-upper',
    'dilapidated'
])
```



### 6.4.1 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 ${ }^{181}$ provided by Statistische Ämter des Bundes und der Länder under the license Datenlizenz Deutschland - Namensnennung - Version 2.0 ${ }^{182}$. Clicking here and there we find two interesting tables:

- annual income per inhabitant ${ }^{183}$
- prices for construction ground ${ }^{184}$

From those tables we may compile a table with 4 columns for region id, region name, income, ground prices.

[^73]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 |  | , Kreisfreie Stadt | 19296 | 85.30 |
| 3 | 1002 | Kiel, | , Kreisfreie Stadt | 19263 | 85.3 |
| 4 | 1003 |  | , 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
\hookrightarrow'][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,
\hookrightarrowRegionalverband')
        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])
```

Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning
(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 |
| ---: | ---: | ---: | ---: | ---: | ---: |
| 13.118355 | Multiple dwelling | 4.663439 | 5.433722 | 5.5 | 1.0 |
| 13.526494 | Farmhouse | 5.093075 | 4.406719 | 5.0 | 2.0 |
| 12.464583 | Farmhouse | 4.941642 | 6.701960 | 4.0 | 2.0 |
| 12.804909 | Duplex | 5.424950 | 6.880384 | 10.0 | 4.0 |
| 14.375126 | Mid-terrace house | 5.347108 | 7.286192 | 6.0 | 2.0 |

Floors Year built

Year renovated
2.772589 modernized
2.079442 dilapidated
3.044522 fixer-upper
1.791759 modernized
1.945910 modernized Baden-Württemberg

Place Garages
Bermatingen 2.0 Outside parking lot
Ostfildern
Bonndorf im Schwarzwald
Neuenbürg
Schönberg

State
Baden-Wurttemberg Baden-Württemberg Baden-Württemberg Baden-Württemberg
1.0 Garage
1.0 Garage
8.0 Outside parking lot
2.0 Garage

| Region_id | Income | Land_prices |
| ---: | ---: | ---: |
| 8435 | 26548 | 172.95 |
| 8116 | 25449 | 387.06 |
| 8337 | 25304 | 111.64 |
| 8236 | 25496 | 192.18 |
| 8111 | 25559 | 1500.34 |

### 6.4.2 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 logscaling we would have a column with very large range, which may result in problems when using regularization (see Regularization (page 129)). 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())
```


## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning

### 6.4.3 Save Data

```
data.to_csv(data_path.replace('preprocessed', 'extended'))
```


### 6.4.4 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
    \hookrightarrow']).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_
$\rightarrow$ size=0.2)
print(y_train.size, y_test.size)
3817955

```
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_
```


### 6.4.5 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!

## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning
### 6.4.6 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__
    \hookrightarrowid']).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
```


### 6.5 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.


### 6.5.1 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}=
# 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_
    \hookrightarrowmask))
# 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 *_
Goutreg.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,
    4:]),
                                    squared=False)
```

```
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)
```



We clearly see, that the small number of outliers deteriorate results significantly.

### 6.5.2 Loss Functions

We already discussed three different loss functions in Quality Measures (page 82):

- 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.

## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning
## 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 ${ }^{185}$. 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,
    G:]),
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 ${ }^{186}$.
Scikit-Learn provides the HuberRegressor ${ }^{187}$.

```
huberreg = linear_model.HuberRegressor()
huberreg.fit(X, y)
hubererr = metrics.mean_squared_error(y[inlier_mask], huberreg.predict(X[inlier_
    \hookrightarrowmask, :]),
        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 |

[^74]
### 6.5.3 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

$$
\frac{\ln (1-p)}{\ln \left(1-(1-r)^{s}\right)}
$$

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 fullv
udata set
max_iter = int(np.log(1 - 0.99) / np.log(1 - (1 - estimated_outlier_rate) **\smile
    \hookrightarrowsubset_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)
    cdetermining a subset
initial_subset_masks = np.empty((max_iter, n), dtype=bool) # initial subset_
cused 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()
```


## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning(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, :]),
& [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.
    4predict(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]
```

```
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
    \hookrightarrow')
# 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=
    \hookrightarrow'threshold', linewidth=1)
ax.plot([xmin, xmax], [-threshold_value + init_b + xmin * init_a,
    -threshold_value + init_b + xmax * init_a], '--c',
    \hookrightarrowlinewidth=1)
# final fit
ax.plot([xmin, xmax], [models_b[index] + xmin * models_a[index],
    models_b[index] + xmax * models_a[index]], '-b', label=
    \hookrightarrow'final fit')
ax.legend()
plt.show()
```


## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning



## RANSAC with Scikit-Learn

Scikit-Learn offers the RANSACRegressor ${ }^{188}$.

```
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: \(\quad 0.00980412170408581\)
    RANSAC (Scikit-Learn): 0.00980412170408581
    [^75]
## 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.

## 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.

### 7.1 Binary Logistic Regression

### 7.1.1 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

$$
\frac{1}{1+\mathrm{e}^{-\left(a_{0}+a_{1} x^{(1)}+\cdots+a_{m} x^{(m)}\right)}}
$$

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

$$
\stackrel{\circ}{x}:=\left(1, x^{(1)}, \ldots, x^{(m)}\right) \quad \text { and } \quad \stackrel{\circ}{a}:=\left(a_{0}, a_{1}, \ldots, a_{m}\right)
$$

it reads

$$
\frac{1}{1+\mathrm{e}^{-\grave{a}^{\mathrm{T}} \dot{x}}} .
$$

For training samples $\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)$ corresponding log loss is

$$
\begin{aligned}
\log \operatorname{loss} & =-\frac{1}{n} \sum_{l=1}^{n}\left(y_{l} \log \frac{1}{1+\mathrm{e}^{-\dot{a}^{\mathrm{T}} \dot{x}_{l}}}+\left(1-y_{l}\right) \log \left(1-\frac{1}{1+\mathrm{e}^{-\dot{\alpha}^{\mathrm{T}} \dot{x}_{l}}}\right)\right) \\
& =\frac{1}{n} \sum_{l=1}^{n}\left(y_{l} \log \left(1+\mathrm{e}^{-\dot{a}^{\mathrm{T}} \dot{x}_{l}}\right)-\left(1-y_{l}\right)\left(-\dot{a}^{\mathrm{T}} \dot{x}_{l}-\log \left(1+\mathrm{e}^{-\dot{a}^{\mathrm{T}} \dot{x}_{l}}\right)\right)\right) \\
& =\frac{1}{n} \sum_{l=1}^{n}\left(\left(1-y_{l}\right) \dot{a}^{\mathrm{T}} \dot{x}_{l}+\log \left(1+\mathrm{e}^{-\dot{a}^{\mathrm{T}} \dot{x}_{l}}\right)\right) .
\end{aligned}
$$

Minimizing the loss function with respect to $\stackrel{\circ}{a}$ yields the model

$$
\frac{1}{1+\mathrm{e}^{-\delta^{T} \dot{x}}} .
$$

for predicting the probability that some sample $x$ belongs to class 1 . From that probability we may derive a class table by thresholding.

### 7.1.2 Interpretation

Thresholding predicted probabilities at some level $t \in(0,1)$ yields class 1 if and only if

$$
\frac{1}{1+\mathrm{e}^{-\dot{a}^{T} \dot{x}}} \geq t \quad \Leftrightarrow \quad \frac{1-t}{t} \geq \mathrm{e}^{-\dot{a}^{\mathrm{T}} \dot{x}} \quad \Leftrightarrow \quad \log \frac{t}{1-t} \leq \stackrel{\circ}{a}^{\mathrm{T}} \dot{x} .
$$

The equation

$$
\log \frac{t}{1-t}=\stackrel{\circ}{a}^{\mathrm{T}} \stackrel{\circ}{x} \quad \Leftrightarrow \quad a_{1} x^{(1)}+\cdots+a_{m} x^{(m)}=\log \frac{t}{1-t}-a_{0}
$$

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.

### 7.1.3 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

$$
p=\frac{1}{1+\mathrm{e}^{-\grave{a}^{\mathrm{T}} \bar{x}}} \Leftrightarrow \quad \Leftrightarrow \quad \log \frac{p}{1-p}=a_{0}+a_{1} x^{(1)}+\cdots+a_{m} x^{(m)} .
$$

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

$$
\lim _{v \rightarrow 0} g(v)=-\infty \quad \text { and } \quad \lim _{v \rightarrow 1} g(v)=\infty
$$

so we would have to transform labels 0 and 1 to $\pm \infty$, which cannot be handled in a linear regression setting.

### 7.2 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.

### 7.2.1 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.

### 7.2.2 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.

### 7.2.3 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

$$
\frac{\mathrm{e}^{\dot{\delta}_{1}^{\mathrm{T}} \dot{x}}}{\mathrm{e}^{\delta_{1}^{\mathrm{T}} \dot{x}}+\cdots+\mathrm{e}^{\dot{a}_{C-1}^{\mathrm{T}} \dot{x}}+1}, \quad \cdots, \quad \frac{\mathrm{e}^{\dot{a}_{C-1}^{\mathrm{T}} \dot{x}}}{\mathrm{e}^{\dot{a}_{1}^{T} \dot{x}}+\cdots+\mathrm{e}^{\dot{\delta}_{C-1}^{\mathrm{T}} \dot{x}}+1}, \quad \frac{1}{\mathrm{e}^{\dot{a}_{1}^{T} \dot{x}}+\cdots+\mathrm{e}^{\dot{\varepsilon}_{C-1}^{\mathrm{T}} \dot{x}}+1}
$$

with $(C-1)(m+1)$ parameters $a_{i}^{l}, i=1, \ldots, C-1, l=0,1, \ldots, m$.

### 7.3 Logistic Regression with Scikit-Learn

Scikit-Learn implements three classes for logistic regression in the linear_model module:

- LogisticRegression ${ }^{189}$ implements logistic regression with regularization. Strength of regularization can be controlled via the parameter $C$, which is the inverse of the usual regularization parameter $\alpha$. The higher $C$ the less regularization is applied. Different algorithms for solving the minimization problem can be chosen.
- SGDClassifier ${ }^{190}$ implements stochastic gradient descent (very efficient in case of many samples) for several loss functions. Passing loss $=1 \log$ ' yields regularized logistic regression. Regulatization is controlled by parameter alpha.
- LogisticRegressionCV ${ }^{191}$ combines LogisticRegression with cross validation for choosing the regularization parameter.
All three classes support different penalties (squares/12, LASSO/l1, elastic net).


### 7.4 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

[^76]
## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

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_
4max, 100))
    y_grid = 1 / (1 + np.exp(-(a + b * x0 + c * x1)))
    cm = matplotlib.colors.LinearSegmentedColormap.from_list('ryg', ['#ff0000',
\hookrightarrow#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

## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

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()
```

0.32751587417245732 .7427345646166641 .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
```

```
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()
```

$7.865065246478776538 .74250199983403-2.765877255810703$


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))
```

33.381878479995414149 .96785067058389 -9.427364096624917
$2.3449177053285104 e+126$

### 7.5 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

$$
\stackrel{\circ}{a}_{1}^{\mathrm{T}} \stackrel{\circ}{x} \geq \dot{a}_{2}^{\mathrm{T}} \stackrel{\circ}{x}, \quad \ldots, \quad \stackrel{\circ}{a}_{1}^{\mathrm{T}} \dot{x} \geq \stackrel{\circ}{a}_{C}^{\mathrm{T}} \stackrel{\circ}{x}
$$

where $\stackrel{\circ}{a}_{C}$ contains zeros (see above). Equivalently,

$$
\left(\stackrel{\circ}{a}_{1}-\stackrel{\circ}{a}_{2}\right)^{\mathrm{T}} \stackrel{\circ}{x} \geq 0, \quad \ldots, \quad\left(\stackrel{\circ}{a}_{1}-\stackrel{\circ}{a}_{C}\right)^{\mathrm{T}} \stackrel{\circ}{x} \geq 0 .
$$

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
```

```
# 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,
    400))
                                    (continues on next page)
```


## Data Science and Artificial Intelligence for Undergraduates

(continued from previous page)

```
Y_grid = logreg.predict(np.stack((x0.reshape(-1), x1.reshape(-1)), axis=1)).
    \hookrightarrowreshape(x0.shape)
cm = matplotlib.colors.LinearSegmentedColormap.from_list('rgb', ['#ff0000', '
    \hookrightarrow#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, '#OOff00', '--')
#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()
```



## 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).

- ANN Basics (page 207)
- Training ANNs (page 216)
- ANNs with Keras (page 233)
- Convolutional Neural Networks (page 252)
- CNNs with Keras (page 265)
- What did the CNN learn? (page 275)
- Improving CNN performance (page 324)

Related projects:
-

- House Prices ANN (page 484)
- PCA and ANN for QMNIST (page 456)
- CNN for QMNIST (page 456)
- CNN Analysis for QMNIST (page 456)
- Hyperparameter Optimization for Cats and Dogs (page 487)


### 8.1 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 $\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)$ satisfying $f\left(x_{l}\right)=y_{l}$ or at least $f\left(x_{l}\right) \approx y_{l}$ 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 $f_{\text {approx }}$ 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 $f_{\text {approx }}$.

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 $f_{\text {approx }}$, 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).

### 8.1.1 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.


Fig. 8.1: 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 ${ }^{192}$

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 ${ }^{193}$ and collective intelligence ${ }^{194}$.

[^77]
### 8.1.2 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 ${ }^{195}$. 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 $g: \mathbb{R} \rightarrow \mathbb{R}$, 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

$$
v=g\left(w^{\mathrm{T}} u\right)
$$

### 8.1.3 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 $f_{\text {approx }}$ as mathematical formula. Denote the weight vectors by $w, \widehat{w}, \stackrel{\circ}{w}, \tilde{w}, \bar{w}$ and the activation functions by $g, \hat{g}, \stackrel{\circ}{g}, \tilde{g}, \bar{g}$. Then we have

$$
f_{\text {approx }}(x)=\bar{g}\left(\bar{w}_{1} g\left(w^{\mathrm{T}} x\right)+\bar{w}_{2} \hat{g}\left(\hat{w}^{\mathrm{T}} x\right)+\bar{w}_{3} \stackrel{g}{g}\left(\dot{w}^{\mathrm{T}} x\right)+\bar{w}_{4} \tilde{g}\left(\tilde{w}^{\mathrm{T}} x\right)\right)
$$

[^78]

Fig. 8.2: 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

$$
\frac{1}{n} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2} \rightarrow \min _{\text {weights }}
$$

with training samples $\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)$. Below we will discuss how to solve such nonlinear minimization problems numerically.

### 8.1.4 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. 8.3: 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

$$
f_{\text {approx }}(x)=g_{3}\left(W_{3} g_{2}\left(W_{2} g_{1}\left(W_{1} x\right)\right)\right)
$$

where the activation functions are applied componentwise.

### 8.1.5 Training ANNs

Training an ANN means solving the minimization problem

$$
\frac{1}{n} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2} \rightarrow \min _{\text {weights }}
$$

The dependence of $f_{\text {approx }}$ 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.

### 8.1.6 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.

### 8.1.7 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.

### 8.1.8 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. 8.4: 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

$$
g\left(w_{0}+w^{\mathrm{T}} u\right)
$$

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

$$
g\left(w_{0}+w^{\mathrm{T}} u\right)= \begin{cases}1, & \text { if } w^{\mathrm{T}} u>-w_{0} \\ 0, & \text { else }\end{cases}
$$

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.

### 8.1.9 Approximation Properties

In linear regression it is obvious which types of functions can be represented by the ansatz for $f_{\text {approx }}$ (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 $f_{\text {approx }}$ 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.

### 8.1.10 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

$$
\frac{1}{n} \sum_{l=1}^{n}\left|f_{\text {approx }}\left(x_{l}\right)-y_{l}\right|^{2},
$$

where $f_{\text {approx }}\left(x_{l}\right)-y_{l}$ is a vector with $d$ components and $|\cdot|$ denotes the length of a vector.


Fig. 8.5: An ANN with three outputs. Number of outputs is independent of the ANN's overall structure.

### 8.1.11 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

$$
o_{i}=\frac{\mathrm{e}^{a_{i}}}{\sum_{j=1}^{C} \mathrm{e}^{a_{j}}} \quad \text { for } i=1, \ldots, C .
$$

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.

### 8.2 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.

### 8.2.1 Gradient Descent for Nonlinear Minimization Problems

Consider a function $h: \mathbb{R}^{p} \rightarrow \mathbb{R}$ and the corresponding minimization problem

$$
h\left(w_{1}, \ldots, w_{p}\right) \rightarrow \min _{w \in \mathbb{R}^{p}}
$$

The gradient

$$
\nabla h\left(w_{1}, \ldots, w_{p}\right)=\left[\begin{array}{c}
\frac{\partial}{\partial w_{1}} h\left(w_{1}, \ldots, w_{p}\right) \\
\vdots \\
\frac{\partial}{\partial w_{p}} h\left(w_{1}, \ldots, w_{p}\right)
\end{array}\right]
$$

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 $\nabla h\left(w^{(0)}\right)$ giving a new point $w^{(1)}$, substract $\nabla h\left(w^{(1)}\right)$, 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 $i=0,1, \ldots$ :
3. Choose a direction $r_{i}$.
4. Choose a step length $s_{i}$.
5. Set $w^{(i+1)}=w^{(i)}+s_{i} r_{i}$
6. If $\left|w^{(i+1)}-w^{(i)}\right|$ is small enough, then stop iteration.

In the simplest case we would choose

$$
r_{i}=-\nabla h\left(w^{(i)}\right) \quad \text { and } \quad s_{i}=s
$$

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.

### 8.2.2 Gradient Descent for ANNs

To apply gradient descent for training ANNs we have to compute the gradient of the objective

$$
h(w)=\frac{1}{n} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2}
$$

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

$$
\frac{\partial}{\partial w_{\kappa}} h(w)=\frac{1}{n} \frac{\partial}{\partial w_{\kappa}} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2}=\frac{1}{n} \sum_{l=1}^{n} \frac{\partial}{\partial w_{\kappa}}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right)^{2} .
$$

Chain rule yields

$$
\frac{\partial}{\partial w_{\kappa}} h(w)=\frac{1}{n} \sum_{l=1}^{n} 2\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right) \frac{\partial}{\partial w_{\kappa}} f_{\text {approx }}\left(x_{l}\right) .
$$

Thus,

$$
\nabla h(w)=\frac{2}{n} \sum_{l=1}^{n}\left(f_{\text {approx }}\left(x_{l}\right)-y_{l}\right) \nabla f_{\text {approx }}\left(x_{l}\right)
$$

If we take $\nabla f_{\text {approx }}\left(x_{1}\right), \ldots, \nabla f_{\text {approx }}\left(x_{n}\right)$ as columns of a matrix $G \in \mathbb{R}^{p \times n}$ with $p$ being the total number of weights and if we set

$$
y_{\text {pred }}:=\left[\begin{array}{c}
f_{\text {approx }}\left(x_{1}\right) \\
\vdots \\
f_{\text {approx }}\left(x_{n}\right)
\end{array}\right] \quad \text { and } \quad y=\left[\begin{array}{c}
y_{1} \\
\vdots \\
y_{n}
\end{array}\right]
$$

we obtain

$$
\nabla h(w)=\frac{2}{n} G\left(y_{\mathrm{pred}}-y\right)
$$

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 $\mu$ are denoted by $w_{0}^{\mathrm{in}, \mu}, w_{1}^{\mathrm{in}, \mu}, \ldots, w_{m}^{\text {in }, \mu}$, where $w_{0}^{\mathrm{in}, \mu}$ 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, $w_{0}^{\text {out }}$ is for the bias input and the others are for the inputs from the $q$ regular neurons. Activation functions are $g_{\text {in }}$ for all regular neurons and $g_{\text {out }}$ for the output neuron.
With this notation we have

$$
w=\left[\begin{array}{llllllllll}
w_{0}^{\mathrm{in}, 1} & \cdots & w_{m}^{\mathrm{in}, 1} & \cdots & w_{0}^{\mathrm{in}, q} & \cdots & w_{m}^{\mathrm{in}, q} & w_{0}^{\text {out }} & \cdots & w_{q}^{\text {out }}
\end{array}\right]^{\mathrm{T}} \in \mathbb{R}^{(m+1) q+q+1}
$$



Fig. 8.6: 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

$$
\begin{gathered}
\frac{\partial}{\partial w_{0}^{\text {in }, \mu}} f_{\text {approx }}(x)=w_{\mu}^{\text {out }} g_{\text {in }}^{\prime}\left(w_{0}^{\text {in }, \mu}+\sum_{k=1}^{m} w_{k}^{\text {in }, \mu} x^{(k)}\right) g_{\text {out }}^{\prime}\left(w_{0}^{\text {out }}+\sum_{\tilde{\mu}=1}^{q} w_{\tilde{\mu}}^{\text {out }} g_{\text {in }}\left(w_{0}^{\text {in }, \tilde{\mu}}+\sum_{k=1}^{m} w_{k}^{\text {in }, \tilde{\mu}} x^{(k)}\right)\right), \\
\frac{\partial}{\partial w_{k}^{\text {in }, \mu}} f_{\text {approx }}(x)=x^{(k)} w_{\mu}^{\text {out }} g_{\text {in }}^{\prime}\left(w_{0}^{\text {in }, \mu}+\sum_{\tilde{k}=1}^{m} w_{\tilde{k}}^{\text {in }, \mu} x^{(\tilde{k})}\right) g_{\text {out }}^{\prime}\left(w_{0}^{\text {out }}+\sum_{\tilde{\mu}=1}^{q} w_{\tilde{\mu}}^{\text {out }} g_{\text {in }}\left(w_{0}^{\text {in }, \tilde{\mu}}+\sum_{\tilde{k}=1}^{m} w_{\tilde{k}}^{\text {in }, \tilde{\mu}} x^{(\tilde{k})}\right)\right)
\end{gathered}
$$

and

$$
\begin{gathered}
\frac{\partial}{\partial w_{0}^{\text {out }}} f_{\text {approx }}(x)=g_{\text {out }}^{\prime}\left(w_{0}^{\text {out }}+\sum_{\mu=1}^{q} w_{\mu}^{\text {out }} g_{\text {in }}\left(w_{0}^{\text {in }, \mu}+\sum_{k=1}^{m} w_{k}^{\text {in }, \mu} x^{(k)}\right)\right) \\
\frac{\partial}{\partial w_{\mu}^{\text {out }}} f_{\text {approx }}(x)=g_{\text {in }}\left(w_{0}^{\text {in }, \mu}+\sum_{k=1}^{m} w_{k}^{\text {in }, \mu} x^{(k)}\right) g_{\text {out }}^{\prime}\left(w_{0}^{\text {out }}+\sum_{\tilde{\mu}=1}^{q} w_{\tilde{\mu}}^{\text {out }} g_{\text {in }}\left(w_{0}^{\text {in }, \tilde{\mu}}+\sum_{k=1}^{m} w_{k}^{\text {in }, \tilde{\mu}} x^{(k)}\right)\right) .
\end{gathered}
$$

Introducing activations

$$
a^{\mathrm{in}, \mu}:=w_{0}^{\mathrm{in}, \mu}+\sum_{k=1}^{m} w_{k}^{\mathrm{in}, \mu} x^{(k)}
$$

for the regular neurons and

$$
a^{\text {out }}:=w_{0}^{\text {out }}+\sum_{\mu=1}^{q} w_{\mu}^{\text {out }} g_{\text {in }}\left(a^{\mathrm{in}, \mu}\right)
$$

for the output neuron, we may rewrite those formulas as

$$
\begin{gathered}
\frac{\partial}{\partial w_{0}^{\text {in }, \mu}} f_{\text {approx }}(x)=w_{\mu}^{\text {out }} g_{\text {in }}^{\prime}\left(a^{\text {in }, \mu}\right) g_{\text {out }}^{\prime}\left(a^{\text {out }}\right), \\
\frac{\partial}{\partial w_{k}^{\text {in }, \mu}} f_{\text {approx }}(x)=x^{(k)} w_{\mu}^{\text {out }} g_{\text {in }}^{\prime}\left(a^{\text {in }, \mu}\right) g_{\text {out }}^{\prime}\left(a^{\text {out }}\right)
\end{gathered}
$$

and

$$
\begin{gathered}
\frac{\partial}{\partial w_{0}^{\text {out }}} f_{\text {approx }}(x)=g_{\text {out }}^{\prime}\left(a^{\text {out }}\right), \\
\frac{\partial}{\partial w_{\mu}^{\text {out }}} f_{\text {approx }}(x)=g_{\text {in }}\left(a^{\text {in }, \mu}\right) g_{\text {out }}^{\prime}\left(a^{\text {out }}\right) .
\end{gathered}
$$

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. 8.7: 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 $g_{\text {out }}$.

We have

$$
w=\left[\begin{array}{llllllllll}
w_{0}^{1,1} & \cdots & w_{q_{0}}^{1, q_{1}} & \cdots & w_{0}^{L, 1} & \cdots & w_{q_{L-1}}^{L, q_{L}} & w_{0}^{\text {out }} & \cdots & w_{q_{L}}^{\text {out }}
\end{array}\right]^{\mathrm{T}} \in \mathbb{R}^{p}
$$

with

$$
p=1+q_{L}+\sum_{\lambda=1}^{L} q_{\lambda}\left(1+q_{\lambda-1}\right)
$$

for the weight vector.
For each neuron corresponding activations are defined as follows:

$$
\begin{gathered}
a^{1, \mu}:=w_{0}^{1, \mu}+\sum_{\nu=1}^{q_{0}} w_{\nu}^{1, \mu} x^{(\nu)} \quad \text { for } \quad \mu=1, \ldots, q_{1}, \\
a^{\lambda, \mu}:=w_{0}^{\lambda, \mu}+\sum_{\nu=1}^{q_{\lambda-1}} w_{\nu}^{\lambda, \mu} g_{\lambda-1}\left(a^{\lambda-1, \nu}\right) \quad \text { for } \quad \mu=1, \ldots, q_{\lambda} \quad \text { and } \quad \lambda=2, \ldots, L, \\
a^{\text {out }}:=w_{0}^{\text {out }}+\sum_{\nu=1}^{q_{L}} w_{\nu}^{\text {out }} g_{L}\left(a^{L, \nu}\right) .
\end{gathered}
$$

With this notation the hypothesis is

$$
f_{\text {approx }}(x)=g_{\text {out }}\left(a^{\text {out }}\right)
$$

Partial derivatives with respect to the weights of the output neuron are

$$
\begin{gathered}
\frac{\partial}{\partial w_{0}^{\text {out }}} f_{\text {approx }}(x)=g_{\text {out }}^{\prime}\left(a^{\text {out }}\right) \frac{\partial}{\partial w_{0}^{\text {out }}} a^{\text {out }}=g_{\text {out }}^{\prime}\left(a^{\text {out }}\right), \\
\frac{\partial}{\partial w_{\nu}^{\text {out }}} f_{\text {approx }}(x)=g_{\text {out }}^{\prime}\left(a^{\text {out }}\right) \frac{\partial}{\partial w_{\nu}^{\text {out }}} a^{\text {out }}=g_{\text {out }}^{\prime}\left(a^{\text {out }}\right) g_{L}\left(a^{L, \nu}\right) \quad \text { for } \quad \nu=1, \ldots, q_{L} .
\end{gathered}
$$

For the other neurons we have
$\frac{\partial}{\partial w_{\nu}^{\lambda, \mu}} f_{\text {approx }}(x)=g_{\text {out }}^{\prime}\left(a^{\text {out }}\right) \frac{\partial}{\partial w_{\nu}^{\lambda, \mu}} a^{\text {out }}=g_{\text {out }}^{\prime}\left(a^{\text {out }}\right)\left(\sum_{\tilde{\nu}=1}^{q_{L}} w_{\tilde{\nu}}^{\text {out }} g_{L}^{\prime}\left(a^{L, \tilde{\nu}}\right) \frac{\partial}{\partial w_{\nu}^{\lambda, \mu}} a^{L, \tilde{\nu}}\right) \quad$ for $\quad \nu=0,1, \ldots, q_{\lambda-1}$.
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 $\lambda$ and it also has no influence on activations of neurons at layer $\lambda$ other than neuron $\mu$. So corresponding derivatives are zero. For activations $a^{\lambda, \mu}$ (neuron the weight belongs to) we get an explicit formula. For layers below layer $\lambda$ we obtain recursive formulas:

$$
\begin{aligned}
\frac{\partial}{\partial w_{\nu}^{\lambda, \mu}} a^{\tilde{\lambda}, \tilde{\mu}} & =\frac{\partial}{\partial w_{\nu}^{\lambda, \mu}}\left(w_{0}^{\tilde{\lambda}, \tilde{\mu}}+\sum_{\tilde{\nu}=1}^{q_{\tilde{\lambda}-1}} w_{\tilde{\nu}}^{\tilde{\lambda}, \tilde{\mu}} g_{\tilde{\lambda}-1}\left(a^{\tilde{\lambda}-1, \tilde{\nu}}\right)\right) \\
& = \begin{cases}0, & \text { if } \tilde{\lambda}<\lambda, \\
0, & \text { if } \tilde{\lambda}=\lambda, \tilde{\mu} \neq \mu, \\
1, & \text { if } \tilde{\lambda}=\lambda, \tilde{\mu}=\mu, \nu=0, \\
g_{\lambda-1}\left(a^{\lambda-1, \nu}\right), & \text { if } \tilde{\lambda}=\lambda, \tilde{\mu}=\mu, \nu>0, \\
q_{\tilde{\lambda}-1} \tilde{\lambda}_{\tilde{\lambda}, \tilde{\mu}} g_{\tilde{\lambda}-1}^{\prime}\left(a^{\tilde{\lambda}-1, \tilde{\nu}}\right) \frac{\partial}{\partial w_{\nu}^{\lambda, \mu}} a^{\tilde{\lambda}-1, \tilde{\nu}}, & \text { if } \tilde{\lambda}>\lambda .\end{cases}
\end{aligned}
$$

To get partial derivatives for activations on layer $L$ with respect to a weight at layer $\lambda$ we first have to calculate all derivatives for activations at layer $\lambda$, 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.

## Data Science and Artificial Intelligence for Undergraduates

## 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.

### 8.2.3 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, withoutu
$\rightarrow$ 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
$\rightarrow$ $\rightarrow$ 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, biasu
$\rightarrow$ 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] $=0 \quad \#$ 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))

```
    # activations (one matrix per layer, one row per sample,
    # columns correspond to neurons, column 0 is bias neurons with nan asw
\varsigmaactivation
    # to have first regular neuron at column 1)
    self.activations = [None] # no neurons with activations on layer o
    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 areu
\hookrightarrowfeature vectors) '''
    # outputs of layer O 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.
\hookrightarrowneurons[layer]))
        self.activations[layer][:, 0] = np.nan # bias neuron has nou
cactivation
        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
coutput layer
    def _flat(self, l, n, i):
    ''' calculate index for self.weight_vector from layer, neuron, inputu
->indices
    (in formulas above: l --> \lamboda, 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):
```

(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 thev
\hookrightarrowgradient.
    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.
\hookrightarrowactivations[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,
\iotaneuron)
    # 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 O (feature values)
    for layer in range(1, self.layers + 2):
        pd_of_acts.append(np.zeros((self.weight_vector.size, *self.
4activations[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,
\hookrightarrowi), :, 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)
```

```
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 3 d 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()
```


## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning

```
<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':
    \hookrightarrow'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 thisu
ヶ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:
```

```
        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_
\leftrightarrowssize, 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. $\rightarrow 6001853239675975$
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.
$\rightarrow 27437575660531127$
iteration 500, grad size 0.027526464130790823 , objective value 0 .
$\hookrightarrow 21478552687825506$
iteration 600, grad size 0.009114642672881857, objective value 0 . 419661760598408443
iteration 700, grad size 0.004846327193297142 , objective value 0.193206235258564
iteration 800, grad size 0.004872007975520194 , objective value 0.
$\rightarrow 19187781786194577$
iteration 900, grad size 0.007417776383947535 , objective value 0. $\rightarrow 19043590964259213$
iteration 1000, grad size 0.008885552556249987 , objective value 0. $\hookrightarrow 18837353720412067$
iteration 1100, grad size 0.009568037192828935 , objective value 0. 418563996550698536
iteration 1200, grad size 0.010012054354481381 , objective value 0. $\rightarrow 18220440430619483$
iteration 1300, grad size 0.010418178877651126 , objective value 0. 417803497660056844
iteration 1400, grad size 0.01075639712936235 , objective value 0 . 417321340025738413
iteration 1500, grad size 0.010947923113323852 , objective value 0 . $\rightarrow 16795120443934383$
iteration 1600, grad size 0.010934279401701321 , objective value 0. 416253664459446235
iteration 1700, grad size 0.010698097971309628 , objective value 0. $\hookrightarrow 1572670641107871$
iteration 1800, grad size 0.010262207608537792 , objective value 0 . $\rightarrow 1523889646805046$
iteration 1900, grad size 0.009676890023421739 , objective value 0. 414806085026090485
iteration 2000, grad size 0.00900891008384827 , objective value 0.14434372346886
iteration 2100 , grad size 0.00833508321112536 , objective value 0.
$\hookrightarrow 14120819144319677$
iteration 2200, grad size 0.0077223848466977055, objective value 0.

```
\hookrightarrow13856023629741945
continues on next page)
```

(continued from previous page)

```
iteration 2300, grad size 0.0072026400917086, objective value 0.
    413628658605000918
iteration 2400, grad size 0.006774949415083191, objective value 0.
    41342867512079747
iteration 2500, grad size 0.4381234445624277, objective value 0.
    415948243205052437
iteration 2600, grad size 0.3040936511232076, objective value 0.1459351832762161
iteration 2700, grad size 0.2935434609394815, objective value 0.
    414490588894469558
iteration 2800, grad size 0.28580780347654683, objective value 0.
    414391902900710235
iteration 2900, grad size 0.27746029701818486, objective value 0.
    414245411381443282
iteration 3000, grad size 0.29328182758454185, objective value 0.
    41400263806581496
iteration 3100, grad size 0.3103952148237437, objective value 0.1357177431813228
iteration 3200, grad size 0.3089037082357916, objective value 0.
    412854283894561383
iteration 3300, grad size 0.2906689025696309, objective value 0.
    412016943141244246
iteration 3400, grad size 0.2703828281297573, objective value 0.
    411274969322635943
iteration 3500, grad size 0.25129739381599697, objective value 0.106501641601027
iteration 3600, grad size 0.24135077314606818, objective value 0.
    410141104705971411
iteration 3700, grad size 0.2421217908293113, objective value 0.
    409778612839471253
iteration 3800, grad size 0.25711890185003916, objective value 0.
    409568766417502658
iteration 3900, grad size 0.284722246737777, objective value 0.0947076864898635
iteration 4000, grad size 0.3296932782682503, objective value 0.
    409403499028986839
iteration 4100, grad size 0.3798402640685176, objective value 0.
    409248875950132941
iteration 4200, grad size 0.4245701332943749, objective value 0.
    408898185624750109
iteration 4300, grad size 0.45647057405936353, objective value 0.
    408323629490675982
iteration 4400, grad size 0.4776166938744331, objective value 0.
    407560503586718813
iteration 4500, grad size 0.4756286013031701, objective value 0.
    406580074692959394
iteration 4600, grad size 0.4237758050301021, objective value 0.
    405261848784999673
iteration 4700, grad size 0.32792804776147605, objective value 0.
    403712512520184369
iteration 4800, grad size 0.23425395310685446, objective value 0.
    402445699644936167
iteration 4900, grad size 0.18114834289066026, objective value 0.
    ๑017227636682214578
iteration 5000, grad size 0.16603929903997156, objective value 0.
    ¢0139897055034033
iteration 5100, grad size 0.1623863909382122, objective value 0.
    4012446982807328397
iteration 5200, grad size 0.1586237216430685, objective value 0.
    \hookrightarrow011479020481321035
iteration 5300, grad size 0.15370335463078733, objective value 0.
    \hookrightarrow010749620797045056
iteration 5400, grad size 0.1484288710518272, objective value 0.
    4010166610010989872
iteration 5500, grad size 0.14333675997862402, objective value 0.
    4009690543922116402
iteration 5600, grad size 0.13870582731696862 , objective value 0. \(\hookrightarrow 009298915283085041\)
iteration 5700, grad size 0.13458156869585244 , objective value 0. \(\rightarrow 008972069187505413\)
iteration 5800, grad size 0.1309219560547808, objective value 0. \(\hookrightarrow 008694452506876942\)
iteration 5900, grad size 0.12764751905438942 , objective value 0. \(\hookrightarrow 008453672061747604\)
iteration 6000, grad size 0.12467834200270841 , objective value 0 \(\checkmark 008240424124501034\)
iteration 6100, grad size 0.12194550534874606 , objective value 0. ↔008047859024417934
iteration 6200, grad size 0.11939480580182533, objective value 0. \(\hookrightarrow 007871041114899354\)
iteration 6300, grad size 0.11698545180964257 , objective value 0. \(\hookrightarrow 0077064369028187205\)
iteration 6400, grad size 0.11468761611433648 , objective value 0 . \(\hookrightarrow 0075515233404994725\)
iteration 6500, grad size 0.11247971986898214, objective value 0 \(\hookrightarrow 007404486914100332\)
iteration 6600, grad size 0.11034623722964698 , objective value 0. \(\rightarrow 007264013112872696\)
iteration 6700, grad size 0.10827590974319178 , objective value 0. \(\hookrightarrow 007129133998917132\)
iteration 6800, grad size 0.10626050161504519 , objective value 0. ↔006999126177998285
iteration 6900, grad size 0.10429386528215825 , objective value 0. \(\rightarrow 0068734380340564755\)
iteration 7000, grad size 0.10237132613873698 , objective value 0. ↔006751641659555333
iteration 7100, grad size 0.10048923678407225 , objective value 0. \(\hookrightarrow 006633398560380885\)
iteration 7200, grad size 0.09864468941543514 , objective value 0. \(\hookrightarrow 006518436751616424\)
iteration 7300, grad size 0.09683531271854097 , objective value 0 . \(\hookrightarrow 0064065343126327\)
iteration 7400, grad size 0.09505913998032933 , objective value 0. \(\hookrightarrow 0062975080366066164\)
iteration 7500, grad size 0.0939519377661455, objective value 0 . ↔006191205164444023
iteration 7600, grad size 0.09285496082438308 , objective value 0. \(\hookrightarrow 0060874973754314445\)
iteration 7700, grad size 0.09176074803899621 , objective value 0. \(\hookrightarrow 0059862762779628554\)
iteration 7800, grad size 0.0906688020220651 , objective value 0. \(\varsigma 005887449924631518\)
iteration 7900, grad size 0.08957875395805977 , objective value 0 . \(\hookrightarrow 005790940069909926\)
iteration 8000, grad size 0.08849035491376321 , objective value 0. 4005696679930145535
iteration 8100, grad size 0.08740346792360068 , objective value 0. \(\rightarrow 0056046123270596685\)
iteration 8200, grad size 0.08631806015976771 , objective value 0. ↔005514688110645113
iteration 8300, grad size 0.0852341950423185, objective value 0. \(\hookrightarrow 005426864801858907\)
iteration 8400, grad size 0.0841520242567544 , objective value 0. \(\hookrightarrow 005341105413835451\)
iteration 8500, grad size 0.0830717796635153 , objective value 0. ↔005257377420520317
iteration 8600, grad size 0.0819937652102004 , objective value 0. ム005175651854001437
(continued from previous page)
```

iteration 8700, grad size 0.08091834891682995, objective value 0.
4005095902514808226
iteration 8800, grad size 0.07984595502241544, objective value 0
ヶ005018105283735866
iteration 8900, grad size 0.07877705637468307, objective value 0
¢004942237526011821
iteration 9000, grad size 0.0777121671202834, objective value 0.
4004868277579470936
iteration 9100, grad size 0.07665183574576094, objective value 0.
ヶ0047962043195361485
iteration 9200, grad size 0.07559663850276738, objective value 0.
4004725996794204519
iteration 9300, grad size 0.0745471732378859, objective value 0.
\hookrightarrow004657633922577992
iteration 9400, grad size 0.0735040536377324, objective value 0.
400459109425080472
iteration 9500, grad size 0.07246790389135159, objective value 0.
๑004526355759560328
iteration 9600, grad size 0.07143935376462186, objective value 0.
40044633957174459735
iteration 9700, grad size 0.07041903407678365, objective value 0.
400440219057498265
iteration 9800, grad size 0.06940757256578538, objective value 0.
4004342715894193019
iteration 9900, grad size 0.06840559012591425, objective value 0.
\hookrightarrow0042849463090558135
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(

```
```

    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':
G'center'})
fig.show()

```
    <IPython.core.display.HTML object>

\subsection*{8.2.4 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 \({ }^{196}\) 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)

\footnotetext{
\(196 \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html
}

\section*{Data Science and Artificial Intelligence for Undergraduates} Volume 2: Data Visualization, Supervised Learning
(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':
\hookrightarrow'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()

```


\subsection*{8.3 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 \({ }^{197}\) 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 \({ }^{198}\) 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.

\footnotetext{
197 https://keras.io
198 https://www.tensorflow.org/
}
```

import tensorflow.keras as keras
2023-04-25 10:26:31.847362: I tensorflow/core/platform/cpu_feature_guard.
\hookrightarrowcc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
Library (oneDNN) to use the following CPU instructions in performance-
\hookrightarrowcritical operations: SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriateь
compiler flags.

```

\subsection*{8.3.1 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 \(28 \times 28\). 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

\section*{Loading Data}

For loading QMNIST data we may reuse code from Load QMNIST (page 453) 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/',ь
\hookrightarrowsubset='train')
test_images, test_labels, _' _ = qmnist.load('../../../../datasets/qmnist/',
suubset='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

\section*{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 ). As a by-product of centering bounding boxes each image will have 4 unused pixels at the boundary. Thus, we may crop images to \(20 \times 20\) 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, :] foru
\iotaefficiency 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()

```

\section*{Data Science and Artificial Intelligence for Undergraduates} Volume 2: Data Visualization, Supervised Learning
```

(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,
420)])

```
```

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 \({ }^{199}\).
```

train_labels = keras.utils.to_categorical(train_labels)
test_labels = keras.utils.to_categorical(test_labels)
train_labels.shape, test_labels.shape

```
    \(((60000,10),(60000,10))\)

\footnotetext{
\({ }^{199}\) https://keras.io/api/utils/python_utils/\#tocategorical-function
}

\section*{Defining the ANN}

Keras has a Mode \(1^{200}\) 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 \({ }^{201}\) 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 \({ }^{202}\) layer and several Dense \({ }^{203}\) layers. Layers can be added one by one with Sequential. add \({ }^{204}\).

Input layers accept multi-dimensional inputs. Thus, we do not have to convert the \(20 \times 20\) images to vectors with 400 components. But Dense layers want to have one-dimensional input. Thus, we use a Flatten \({ }^{205}\) 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 \({ }^{206}\) 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_
\hookrightarrowdiagnostics.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_
cdiagnostics.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_
\iotadiagnostics.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.
๑cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
\hookrightarrowibrary (oneDNN) to use the following CPU instructions in performance-
ccritical operations: SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriatev
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

```

\footnotetext{
\({ }^{200} \mathrm{https}: / /\) keras.io/api/models/model/
\({ }^{201} \mathrm{https}: / / \mathrm{keras} . i o / a p i / m o d e l s / s e q u e n t i a l / ~\)
\(202 \mathrm{https}: / /\) keras.io/api/layers/core_layers/input/
\({ }^{203} \mathrm{https}: / / \mathrm{keras} . i o / a p i / l a y e r s / c o r e \_l a y e r s / d e n s e / ~\)
\({ }^{204} \mathrm{https}: / /\) keras.io/api/models/sequential/\#add-method
\({ }^{205} \mathrm{https}: / / \mathrm{keras.io}\) /api/layers/reshaping_layers/flatten/
\(206 \mathrm{https}: / / \mathrm{keras} . i o / a p i /\) 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()

```
\begin{tabular}{|c|c|c|}
\hline Layer (type) & Output Shape & Param \# \\
\hline flatten (Flatten) & (None, 400) & 0 \\
\hline dense1 (Dense) & (None, 10) & 4010 \\
\hline dense2 (Dense) & (None, 10) & 110 \\
\hline out (Dense) & (None, 10) & 110 \\
\hline \multicolumn{3}{|l|}{Total params: 4,230} \\
\hline \multicolumn{3}{|l|}{Trainable params: 4,230} \\
\hline Non-trainable params: 0 & & \\
\hline
\end{tabular}

\section*{Training the ANN}

Parameters for training are set with Model.compile \({ }^{207}\). Here we may define an optimization routine. Next to gradient descent there are several other optimizers available \({ }^{208}\). 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 \({ }^{209}\). 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 \({ }^{210}\) 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. \(f i t^{211}\). 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 \(f\) it will be discussed below.
```

history = model.fit(train_images, train_labels, batch_size=100, epochs=5,
svalidation_split=0.2)

```

\footnotetext{
\({ }^{207} \mathrm{https}: / / \mathrm{keras.io} / \mathrm{api} /\) models/model_training_apis/\#compile-method
\(208 \mathrm{https}: / /\) keras.io/api/optimizers/\#available-optimizers
\({ }^{209} \mathrm{https}: / / \mathrm{keras} . i o / a p i /\) losses/\#available-losses
210 https://keras.io/api/metrics/
\({ }^{211} \mathrm{https}: / / \mathrm{keras} . i o / a p i / m o d e l s / m o d e l \_t r a i n i n g \_a p i s / \# f i t-m e t h o d ~\)
}
```

Epoch 1/5
480/480 [===============================] - 1s 2ms/step - loss: 2.0348 --
categorical_accuracy: 0.3197 - val_loss: 1.6531 - val_categorical_accuracy: 0.
4480
Epoch 2/5
480/480 [==============================] - 1s 1ms/step - loss: 1.2421 --
\hookrightarrowcategorical_accuracy: 0.6128 - val_loss: 0.8575 - val_categorical_accuracy: 0.
7488
Epoch 3/5
480/480 [================================] - 1s 1ms/step - loss: 0.7235 --
\hookrightarrowcategorical_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.
\&720
Epoch 5/5
480/480 [==============================] - 1s 1ms/step - loss: 0.4769 -七
\iotacategorical_accuracy: 0.8661 - val_loss: 0.4172 - val_categorical_accuracy: 0.
4847

```

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.

\section*{Incremental Training}

The Model.fit method returns a Hist ory 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
4accuracy')
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=5,七
\hookrightarrowvalidation_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.
4942
Epoch 3/5
480/480 [==============================] - 1s 1ms/step - loss: 0.4045 --
\hookrightarrowcategorical_accuracy: 0.8891 - val_loss: 0.3674 - val_categorical_accuracy: 0.
>992
Epoch 4/5
480/480 [===============================] - 1s 1ms/step - loss: 0.3920 --
categorical_accuracy: 0.8930 - val_loss: 0.3564 - val_categorical_accuracy: 0.
>9999
Epoch 5/5
480/480 [===============================] - 1s 1ms/step - loss: 0.3813 --
categorical_accuracy: 0.8954 - val_loss: 0.3486 - val_categorical_accuracy: 0.
4016

```
```

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()

```


\section*{Evaluation and Prediction}

To get loss and metrics on the test set call Model. evaluate \({ }^{212}\).
```

test_loss, test_metric = model.evaluate(test_images, test_labels)
test_loss, test_metric

```
```

1875/1875 [================================] - 2s 968us/step - loss: 0.3572 --
\hookrightarrowcategorical_accuracy: 0.9006
(0.3571886718273163, 0.9005666375160217)

```

For predictions call Model. predict \({ }^{213}\).
```

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()

```


\footnotetext{
\({ }^{212} \mathrm{https}: / / \mathrm{keras} . i o / a p i / m o d e l s / m o d e l \_t r a i n i n g \_a p i s / \# e v a l u a t e-m e t h o d ~\)
\({ }^{213} \mathrm{https}: / /\) 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)

```
```

    5 9 8 7
    [ 8 18 33 ... 59974 59998 59999]
    ```
\(i d x=7\)
show_image (test_images[idx])
print('truth: \{\}, prediction: \{\}'.format(true_digits[idx], pred_digits[idx]))


\footnotetext{
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 \({ }^{214}\), sns.heatmap \({ }^{215}\) ).
```

conf_matrix = pd.crosstab(pd.Series(true_digits, name='truth'),
pd.Series(pred_digits, name='prediction'))
print(conf_matrix)

```
\begin{tabular}{lrrrrrrrrrr} 
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
\end{tabular}
```


# 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 scaledv
4labels)
plt.show()

```

\footnotetext{
\(214 \mathrm{https}: / /\) pandas.pydata.org/docs/reference/api/pandas.crosstab.htm
\({ }^{215} \mathrm{https}: / /\) seaborn.pydata.org/generated/seaborn.heatmap.html
}


\subsection*{8.3.2 Hyperparameter Optimization}

Keras itself offers no hyperparameter optimization routines. But there is the keras-tuner \({ }^{216}\) 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 \({ }^{217}\) 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)

\footnotetext{
\({ }^{216} \mathrm{https}: / /\) keras-team.github.io/keras-tuner/
\({ }^{217}\) 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 \({ }^{218}\) object and call its search \({ }^{219}\) method. Several subclasses are available. RandomSearch \({ }^{220}\) 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_
\hookrightarrowcategorical_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

```

\footnotetext{
218 https://keras-team.github.io/keras-tuner/documentation/tuners/
\(219 \mathrm{https}: / / \mathrm{keras} . i o / a p i / k e r a s \_t u n e r / t u n e r s / b a s e \_t u n e r / \# s e a r c h-m e t h o d\)
\({ }^{220}\) https://keras.io/api/keras_tuner/tuners/random/\#randomsearch-class
}
```

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 \({ }^{221}\), which returns a sorted (best first) list of trained Model instances. Alternatively, we may call Tuner.get_best_hyperparameters \({ }^{222}\) 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"
\begin{tabular}{lll}
\hline \begin{tabular}{l} 
Layer (type) \\
\(=============================\) \\
flatten_1 (Flatten)
\end{tabular} & \begin{tabular}{c} 
Output Shape \\
(None, 400)
\end{tabular} & Param \# \\
dense (Dense) & (None, 30) & 0
\end{tabular}
(continues on next page)

\footnotetext{
\({ }^{221} \mathrm{https}: / /\) keras.io/api/keras_tuner/tuners/base_tuner/\#get_best_models-method
\({ }^{222} \mathrm{https}: / /\) keras.io/api/keras_tuner/tuners/base_tuner/\#get_best_hyperparameters-method
}
```

dense_3 (Dense)

```
(None, 10)

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 --
\iotacategorical_accuracy: 0.7655
Epoch 2/10
1875/1875 [=================================] - 2s 1ms/step - loss: 0.3384 --
\hookrightarrowcategorical_accuracy: 0.9027
Epoch 3/10
1875/1875 [=================================] - 3s 1ms/step - loss: 0.2765 --
\hookrightarrowcategorical_accuracy: 0.9202
Epoch 4/10
1875/1875 [================================] - 2s 1ms/step - loss: 0.2381 --
\hookrightarrowcategorical_accuracy: 0.9314
Epoch 5/10
1875/1875 [================================] - 2s 1ms/step - loss: 0.2127 --
\hookrightarrowcategorical_accuracy: 0.9384
Epoch 6/10
1875/1875 [================================] - 2s 1ms/step - loss: 0.1937 --
\iotacategorical_accuracy: 0.9430
Epoch 7/10
1875/1875 [===============================] - 2s 1ms/step - loss: 0.1786 --
\hookrightarrowcategorical_accuracy: 0.9478
Epoch 8/10
1875/1875 [================================] - 2s 1ms/step - loss: 0.1668 --
ccategorical_accuracy: 0.9514
Epoch 9/10
1875/1875 [=================================] - 2s 1ms/step - loss: 0.1571 --
ccategorical_accuracy: 0.9545
Epoch 10/10
1875/1875 [================================] - 2s 1ms/step - loss: 0.1478 --
\hookrightarrowcategorical_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 [================================]-2s1000us/step-1oss:0.1729--1
    \(\rightarrow c a t e g o r i c a l \_a c c u r a c y: 0.9482\)
    (0.17285792529582977, 0.948199987411499)

\section*{Data Science and Artificial Intelligence for Undergraduates Volume 2：Data Visualization，Supervised Learning}

\section*{8．3．3 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 \({ }^{223}\) ．We simply have to create a suitable Callback object and pass it to the fit method．Stopping criteria can be implemented with EarlyStop－ ping \({ }^{224}\) 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，re－ store＿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,
\hookrightarrowrestore_best_weights=True)
best_model.fit(train_images, train_labels, validation_split=0.2, epochs=1000,
ccallbacks=[es])

```
    Epoch 1/1000
    1500/1500 [===============================1]-2s2ms/step-10ss:0.1421-七
        ↔categorical_accuracy: 0.9586 - val_loss: 0.1316 - val_categorical_accuracy: 0.
        \(\rightarrow 9625\)
    Epoch 2/1000
    1500/1500 [ \(==============================1\) - \(2 \mathrm{~s} 1 \mathrm{~ms} / \mathrm{step}-\mathrm{loss}: 0.1361\)-七
        \(\rightarrow\) categorical_accuracy: 0.9603 - val_loss: 0.1337 - val_categorical_accuracy: 0.
        \(\rightarrow 9603\)
    Epoch 3/1000
    1500/1500 [===============================]-2s1ms/step-10ss:0.1307-七
        ↔categorical_accuracy: 0.9615 - val_loss: 0.1365 - val_categorical_accuracy: 0.
        ¢9598
    Epoch 4/1000
    1500/1500 [===============================1]-2s1ms/step-10ss:0.1250-七
        \(\hookrightarrow\) categorical_accuracy: 0.9633 - val_loss: 0.1283 - val_categorical_accuracy: 0.
        \(\hookrightarrow 9613\)
    Epoch 5/1000
    1500/1500 [ \(==============================1\) - \(2 \mathrm{~s} 1 \mathrm{~ms} / \mathrm{step}-\mathrm{loss}: 0.1196\) -
        \(\rightarrow\) categorical_accuracy: 0.9649 - val_loss: 0.1277 - val_categorical_accuracy: 0.
        49620
    Epoch 6/1000
    1500/1500 [===============================]-2s1ms/step-10ss:0.1153-七
        \(\hookrightarrow\) categorical_accuracy: 0.9659 - val_loss: 0.1260 - val_categorical_accuracy: 0.
        49629
    Epoch 7/1000
    1500/1500 [ \(==============================1\) - \(2 \mathrm{~s} 1 \mathrm{~ms} / \mathrm{step}-\mathrm{loss}: 0.1112\) -
        \(\rightarrow\) categorical_accuracy: 0.9673 - val_loss: 0.1286 - val_categorical_accuracy: 0 .
        \(\hookrightarrow 9628\)
    Epoch 8/1000
    1500/1500 [===============================1 - 2s 1ms/step - loss: 0.1070-七
        \(\rightarrow\) categorical_accuracy: 0.9683 - val_loss: 0.1286 - val_categorical_accuracy: 0.
        \(\rightarrow 9614\)
    Epoch 9/1000
    1500/1500 [===============================]-2s 2ms/step-loss:0.1032-七
        \(\hookrightarrow\) categorical_accuracy: 0.9697 - val_loss: 0.1299 - val_categorical_accuracy: 0.
        \(\rightarrow 9618\)
    <keras.callbacks.History at 0x7f86bc0224a0>

Resulting accuracy on test set：

\footnotetext{
\({ }^{223} \mathrm{https}: / /\) keras．io／api／callbacks／
\(224 \mathrm{https}: / /\) 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 -七
\hookrightarrowcategorical_accuracy: 0.9562
(0.14791476726531982, 0.9561833143234253)

```

\subsection*{8.3.4 Saving and Loading Models}

Keras models provide a save \({ }^{225}\) to a model to a file. To load a model use load_model \({ }^{226}\).
```

best_model.save('keras_save_best_model')

```
    WARNING:absl:Found untraced functions such as _update_step_xla while saving_
    \(\hookrightarrow(\) 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')

```

\subsection*{8.3.5 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 \({ }^{227}\) 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,
ヶ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])

```

\footnotetext{
\(225 \mathrm{https}: / / \mathrm{keras} . i o / a p i / m o d e l s / m o d e l \_s a v i n g \_a p i s / m o d e l \_s a v i n g \_a n d \_l o a d i n g / \# s a v e-m e t h o d ~\)

\(227 \mathrm{https}: / /\) keras.io/api/callbacks/tensorboard/
}
```

Epoch 1/1000
1500/1500 [================================] - 2s 2ms/step - loss: 0.1106 -七
\hookrightarrowcategorical_accuracy: 0.9669 - val_loss: 0.1313 - val_categorical_accuracy: 0.
4596
Epoch 2/1000
1500/1500 [===============================] - 2s 2ms/step - loss: 0.1069 --
4categorical_accuracy: 0.9683 - val_loss: 0.1273 - val_categorical_accuracy: 0.
49618
Epoch 3/1000
1500/1500 [=================================] - 2s 1ms/step - loss: 0.1036 --
\hookrightarrowcategorical_accuracy: 0.9689 - val_loss: 0.1255 - val_categorical_accuracy: 0.
49626
Epoch 4/1000
1500/1500 [================================] - 2s 1ms/step - loss: 0.0994 -七
¢categorical_accuracy: 0.9700 - val_loss: 0.1300 - val_categorical_accuracy: 0.
49615
Epoch 5/1000
1500/1500 [===============================] - 2s 1ms/step - loss: 0.0970 --
\hookrightarrowcategorical_accuracy: 0.9715 - val_loss: 0.1274 - val_categorical_accuracy: 0.
49628
Epoch 6/1000
1500/1500 [================================] - 2s 1ms/step - loss: 0.0940 --
\hookrightarrowcategorical_accuracy: 0.9726 - val_loss: 0.1297 - val_categorical_accuracy: 0.
\hookrightarrow9611
<keras.callbacks.History at 0x7f869ae5ae60>

```
```

%load_ext tensorboard
%tensorboard --logdir tensorboard_data

```

\subsection*{8.4 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.

\subsection*{8.4.1 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 \(w_{\nu}\) 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. 8.8: 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. 8.9: 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.

\subsection*{8.4.2 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.

\(u * w=\)\begin{tabular}{|c|c|c|c|c|}
\hline-11 & 4 & 3 & 3 & -2 \\
\hline 9 & -10 & -3 & 4 & -1 \\
\hline 4 & 7 & -10 & 5 & -2 \\
\hline 9 & -6 & 0 & 3 & 3 \\
\hline
\end{tabular}


Fig. 8.10: Computations for 2 d convolutions follow the same scheme as for 1 d 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()

```


\section*{Data Science and Artificial Intelligence for Undergraduates} Volume 2: Data Visualization, Supervised Learning

\subsection*{8.4.3 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 \({ }^{228}\) ). 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)


\footnotetext{
\({ }^{228}\) 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.
sshape[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:(mb
\hookrightarrow+ 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()

```


\subsection*{8.4.4 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. 8.11: 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.

\subsection*{8.4.5 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 \(2 \times 2\) 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. 8.12: CNNs typically consist of serveral convolution stacks and a small number of dense layers.

\subsection*{8.5 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 \(\mathrm{CCO}^{229}\) for competition on www.kaggle.com \({ }^{230}\). 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/'

```

\footnotetext{
\({ }^{229} \mathrm{https}: / /\) creativecommons.org/publicdomain/zero/1.0/
\(230 \mathrm{https}: / / \mathrm{www}\). kaggle.com/tongpython/cat-and-dog
\({ }^{231} \mathrm{https}: / / \mathrm{xkcd} . c o m / 1425\)
}


INCS, IT CAN BE HARD TO EXPLAIN THE DIFFERENCE BETWEEN THE EASY AND THE VIRTUALUY IMPOSSIBLE.

Fig. 8.13: 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 231
```

2023-07-03 05:33:58.382411: I tensorflow/core/platform/cpu_feature_guard.
¢C:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network`
\hookrightarrowLibrary (oneDNN) to use the following CPU instructions in performance-
\hookrightarrowcritical 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\smile
(-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_
\hookrightarrowgpu_executor.cc:981] successful NUMA node read from SysFS had negative value\smile
(-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\checkmark
\hookrightarrow(-1), but there must be at least one NUMA node, so returning NUMA node zero

```

\subsection*{8.5.1 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 \({ }^{232}\), which returns a Tensorflow Dataset \({ }^{233}\) 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)

\footnotetext{
\({ }^{232} \mathrm{https}: / /\) keras.io/api/preprocessing/image/
\({ }^{233} \mathrm{https}: / / w w w . t e n s o r f l o w . o r g / a p i \_d o c s / p y t h o n / t f / d a t a / D a t a s e t ~\)
}
```

)
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.
        \(\hookrightarrow C c: 193]\) This TensorFlow binary is optimized with oneAPI Deep Neural Network
        \(\hookrightarrow\) Library (oneDNN) to use the following CPU instructions in performance-
        \(\hookrightarrow c r i t i c a l ~ o p e r a t i o n s: ~ S S E 4.1 ~ S S E 4.2 ~ A V X ~ A V X 2 ~ F M A ~\)
To enable them in other operations, rebuild TensorFlow with the appropriater
    compiler flags.
2023-07-03 05:34:01.437382: I tensorflow/compiler/xla/stream_executor/cuda/cuda_
    \(\left.\hookrightarrow g p u \_e x e c u t o r . c c: 981\right]\) successful NUMA node read from SysFS had negative value
    \(\hookrightarrow(-1)\), 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_
    \(\left.\hookrightarrow g p u \_e x e c u t o r . c c: 981\right]\) successful NUMA node read from SysFS had negative value■
    \(\hookrightarrow(-1)\), 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」
    \(\hookrightarrow(-1)\), 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」
    \(\hookrightarrow(-1)\), 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↔
    \(\hookrightarrow(-1)\), 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
    \(\leftrightarrow(-1)\), 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.
    \(\hookrightarrow C C: 1613]\) Created device /job:localhost/replica:0/task:0/device:GPU:0 with
    \(\hookrightarrow 1649\) MB memory: -> device: 0, name: NVIDIA GeForce MX130, pci bus id:
    \(\hookrightarrow 0000: 01: 00.0\), 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 Dat aset 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 128 x128 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()

```

dog


\subsection*{8.5.2 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 \({ }^{234}\), MaxPooling2D \({ }^{235}\) and Dense \({ }^{236}\).

To scale the image's range to \([0,1]\) we may use a Rescaling layer \({ }^{237}\).

Important: Due to a bug in TensorFlow 2.9 and above training of Keras models with preprocessing layers is

\footnotetext{
\({ }^{234} \mathrm{https}: / /\) keras.io/api/layers/convolution_layers/convolution2d/
\({ }^{235} \mathrm{https}: / /\) keras.io/api/layers/pooling_layers/max_pooling2d/
\({ }^{236} \mathrm{https}: / / \mathrm{keras} . i o / a p i / l a y e r s / c o r e \_l a y e r s / d e n s e / ~\)
\({ }^{237} \mathrm{https}: / /\) keras.io/api/layers/preprocessing_layers/image_preprocessing/rescaling/
}

\title{
Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning
}
extremely slow. See TensorFlow issue \({ }^{238}\) 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"
\begin{tabular}{|c|c|c|c|}
\hline Layer (type) & Output & Shape & Param \# \\
\hline rescaling (Rescaling) & (None, & 128, 128, 3) & 0 \\
\hline conv1 (Conv2D) & (None, & 126, 126, 16) & 448 \\
\hline conv2 (Conv2D) & (None, & 124, 124, 16) & 2320 \\
\hline pool1 (MaxPooling2D) & (None, & 62, 62, 16) & 0 \\
\hline conv3 (Conv2D) & (None, & 60, 60, 32) & 4640 \\
\hline conv4 (Conv2D) & (None, & 58, 58, 32) & 9248 \\
\hline pool2 (MaxPooling2D) & (None, & 29, 29, 32) & 0 \\
\hline flatten (Flatten) & (None, & 26912) & 0 \\
\hline dense1 (Dense) & (None, & & 269130 \\
\hline dense2 (Dense) & (None, & & 110 \\
\hline out (Dense) & (None, & 2) & 22 \\
\hline
\end{tabular}

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 ( \(3 \times 3\) or \(5 \times 5\) ) filters and several convolutional layers are better than few large filters. From layer to layer number of filters may be

\footnotetext{
\({ }^{238}\) 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.

\subsection*{8.5.3 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 notь
guarantee that XLA will be used). Devices:
2023-07-03 05:34:08.843578: I tensorflow/compiler/xla/service/service.cc:181]
\hookrightarrowStreamExecutor 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.
๑c:477] Compiled cluster using XLA! This line is logged at most once for theఒ
\hookrightarrowlifetime 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 --
\hookrightarrowcategorical_accuracy: 0.6987 - val_loss: 0.5256 - val_categorical_accuracy: 0.
7410
Epoch 3/10
469/469 [===============================] - 60s 126ms/step - loss: 0.4951 --
\hookrightarrowcategorical_accuracy: 0.7602 - val_loss: 0.4864 - val_categorical_accuracy: 0.
7690
Epoch 4/10
469/469 [===============================] - 60s 127ms/step - loss: 0.4416 --
\iotacategorical_accuracy: 0.7971 - val_loss: 0.4803 - val_categorical_accuracy: 0.
47660
(continued from previous page)

```
Epoch 5/10
469/469 [================================] - 60s 128ms/step - loss: 0.3980 --
    \hookrightarrowcategorical_accuracy: 0.8213 - val_loss: 0.4730 - val_categorical_accuracy: 0.
    47786
Epoch 6/10
469/469 [===============================] - 60s 128ms/step - loss: 0.3507 --
    \hookrightarrowcategorical_accuracy: 0.8469 - val_loss: 0.4731 - val_categorical_accuracy: 0.
    \hookrightarrow924
Epoch 7/10
469/469 [===============================] - 60s 128ms/step - loss: 0.3054 --
    \hookrightarrowcategorical_accuracy: 0.8683 - val_loss: 0.5329 - val_categorical_accuracy: 0.
    47846
Epoch 8/10
469/469 [================================] - 60s 128ms/step - loss: 0.2576 --
    ccategorical_accuracy: 0.8925 - val_loss: 0.6474 - val_categorical_accuracy: 0.
    4658
Epoch 9/10
469/469 [===============================] - 60s 128ms/step - loss: 0.2156 --
    \hookrightarrowcategorical_accuracy: 0.9115 - val_loss: 0.6397 - val_categorical_accuracy: 0.
    4986
Epoch 10/10
469/469 [================================] - 60s 128ms/step - loss: 0.1751 --
    ccategorical_accuracy: 0.9311 - val_loss: 0.7653 - val_categorical_accuracy: 0.
    \hookrightarrow7898
```

```
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 ${ }^{239}$ 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

[^79]
## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

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 ${ }^{240}$.

```
model.save('cnnmodel')
```

```
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _
    \hookrightarrowjit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_
    Convolution_op while saving (showing 4 of 4). These functions will not be,
    cdirectly callable after loading.
INFO:tensorflow:Assets written to: cnnmodel/assets
INFO:tensorflow:Assets written to: cnnmodel/assets
```

```
model = keras.models.load_model('cnnmodel')
```


### 8.5.4 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 --
    ccategorical_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)
```

$1 / 1[==============================1$ - $0 \mathrm{~s} 134 \mathrm{~ms} / \mathrm{step}$

```
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()
```




[^80]```
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,
    41]))
ax.axis('off')
plt.show()
```

cat: 0.78 , dog: 0.72


### 8.6 What did the CNN Iearn?

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.
    \iotacc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Network
    \hookrightarrowLibrary (oneDNN) to use the following CPU instructions in performance-
    \hookrightarrowcritical operations: SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriateь
    compiler flags.
```

Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

```
# 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_
sdriver.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_
$\rightarrow$ 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_
$\rightarrow$ 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_
$\rightarrow$ 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_

2023-07-03 06:38:05.645067: I tensorflow/core/platform/cpu_feature_guard.
$\varsigma c c: 193]$ This TensorFlow binary is optimized with oneAPI Deep Neural Network
$\hookrightarrow$ 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 appropriateu
$\hookrightarrow$ 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

```
Trainable params: 285,918
Non-trainable params: 0
```


### 8.6.1 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 ${ }^{241}$ 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_
    4name).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 ${ }^{242}$. This function returns a PIL image object ${ }^{243}$ 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.float 32)
```

(continues on next page)

[^81]```
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)
```


$1 / 1[===============================1$ - $0 \mathrm{~s} 203 \mathrm{~ms} / \mathrm{step}$ $(126,126,16)$

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
```




$\times 3$

$\times 1$

$\times 2$


$\times 2$

$\times 3$

$\times 3$


$\times 1$

$\times 5$


### 8.6.2 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 ${ }^{244}$. 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)
```

[^82]
## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning

```
(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

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()


### 8.6.3 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, ) +L
    neuron])
submodel.summary()
```

```
Model: "model_5"
```



```
    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 ${ }^{245}$ 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 Gradient Tape ${ }^{246}$. The flow of all variables marked for watching with GradientTape. watch ${ }^{247}$ is recorded for all graph executions inside the with block. After executing the graph we get the gradient from GradientTape.gradient ${ }^{248}$. 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

[^83]
## Data Science and Artificial Intelligence for Undergraduates

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
0.07358889 0.0003213614400010556
30.07369768 0.00030644104117527604
4 0.07383166 0.00031520603806711733
50.07399155 0.00030644104117527604
0.074129954 0.00031513432622887194
0.0742565 0.00030454201623797417
0.07441732 0.0003239291545469314
9 0.07457861 0.00030454201623797417
10 0.07470061 0.00030454201623797417
```

```
11 0.07487879 0.00030454201623797417
12 0.07501966 0.00030454201623797417
0.07516515 0.0003058453439734876
40.075305745 0.00030454201623797417
0.0754803 0.0003058453439734876
0.07562959 0.00030454201623797417
0.07579061 0.00030454201623797417
0.07593748 0.0003058453439734876
0.07606852 0.00030454201623797417
0.076212205 0.00030454201623797417
0.07638178 0.0003369719488546252
0.076535545 0.00030454201623797417
0.07666714 0.0003113079583272338
0.07682478 0.00030454201623797417
0.07697314 0.00030454201623797417
0.07714298 0.00030454201623797417
0.077249855 0.0003259675286244601
0.07743089 0.00030454201623797417
0.07758624 0.0003369719488546252
0.07774404 0.00030454201623797417
0.0778713 0.00031524914084002376
0.07803682 0.0003058453439734876
0.078191474 0.0003058453439734876
0.07833725 0.00036290791467763484
0.07847955 0.00030454201623797417
0.07863803 0.00030454201623797417
0.07877792 0.0003113079583272338
0.07892162 0.00030454201623797417
0.07909869 0.00030454201623797417
0.07921913 0.0003113079583272338
0.07938554 0.00030454201623797417
0.07953825 0.00030454201623797417
0.07970761 0.00030454201623797417
0.07983406 0.00031524914084002376
0.07996544 0.0003058453439734876
0.08013553 0.00030454201623797417
0.0802946 0.00030454201623797417
0.08043927 0.0003058453439734876
0.08059702 0.00030454201623797417
0.08072535 0.00030454201623797417
0.08089805 0.00030454201623797417
0.08103597 0.0003369719488546252
0.08119964 0.0003058453439734876
0.08135997 0.00030454201623797417
0.08145313 0.0003259675286244601
0.08164934 0.00030454201623797417
0.081796795 0.00030454201623797417
0.08194393 0.00030454201623797417
0.08208322 0.00033827530569396913
0.08226341 0.00030454201623797417
0.08238993 0.00031524914084002376
0.082546435 0.00030454201623797417
0.08268981 0.00030454201623797417
0.082855605 0.0003113079583272338
0.08297971 0.00030454201623797417
0.08315368 0.00030454201623797417
0.08331067 0.00030454201623797417
0.08345787 0.0003048637881875038
0.083590515 0.0003035604313481599
0.0837581 0.0003232989402022213
0.08390607 0.0003113079583272338
```

```
72 0.084045365 0.0003247601562179625
0.08419961 0.0003035604313481599
0.08436007 0.0003035604313481599
0.08451763 0.0003048637881875038
0.08465027 0.0003060454619117081
0.08482045 0.0003035604313481599
0.084977776 0.0003035604313481599
0.08512163 0.0003035604313481599
0.08527139 0.0003035604313481599
0.08544078 0.0003048637881875038
0.085562915 0.0003035604313481599
0.085736886 0.0003048637881875038
0.085881226 0.0003035604313481599
0.086030394 0.00033729374990798533
0.086165085 0.0003035604313481599
0.086344495 0.0003197602345608175
0.08646924 0.0003035604313481599
0.08664642 0.0003035604313481599
0.086805105 0.0003035604313481599
0.086939305 0.0003035604313481599
0.08710596 0.00031557094189338386
0.08726247 0.0003035604313481599
0.08738602 0.0003035604313481599
0.08757089 0.0003035604313481599
0.087709084 0.0003048637881875038
0.08785117 0.0003172186843585223
0.08802833 0.0003197602345608175
0.08816219 0.0003035604313481599
0.08833459 0.0003359904221724719
0.0884886 0.0003035604313481599
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
1 1 0 0 . 0 8 9 8 7 5 9 5 ~ 0 . 0 0 0 3 0 4 8 6 3 7 8 8 1 8 7 5 0 3 8 )
111 0.09004665 0.0003035604313481599
112 0.090225294 0.0003035604313481599
1 1 3 0 . 0 9 0 3 7 1 7 9 5 ~ 0 . 0 0 0 3 1 5 5 7 0 9 4 1 8 9 3 3 8 3 8 6 ~
114 0.09052053 0.0003035604313481599
1 1 5 0 . 0 9 0 6 8 3 3 1 ~ 0 . 0 0 0 3 0 3 5 6 0 4 3 1 3 4 8 1 5 9 9 ~
116 0.0908356 0.0003259675286244601
117 0.09098149 0.0003172186843585223
118 0.091161326 0.0003035604313481599
1 1 9 0 . 0 9 1 3 2 4 3 7 ~ 0 . 0 0 0 3 0 3 5 6 0 4 3 1 3 4 8 1 5 9 9 ~
120 0.09145064 0.00031426758505403996
1 2 1 0 . 0 9 1 6 3 6 4 7 ~ 0 . 0 0 0 3 0 4 9 6 3 9 6 3 5 7 1 9 3 5 9 ~
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
1300.093041636 0.0003049639635719359
131 0.0932492 0.000548863725271076
132 0.093575545 0.000548863725271076
```

```
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
1520.10162692 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
1580.10411946 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
1690.108925685 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
1760.112955004 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
```

Data Science and Artificial Intelligence for Undergraduates

```
1940.123206116 0.0009152829879894853
195 0.12377263 0.0009152829879894853
196 0.12432422 0.0009152829879894853
1970.1248651 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
2120.134650290.001039297436363995
2130.13540941 0.001039297436363995
214 0.13616672 0.001039297436363995
2150.13694865 0.001039297436363995
2160.137729290.001039297436363995
2170.13847396 0.001039297436363995
2180.13922313 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
2300.14767927 0.0010269630001857877
2310.14838807 0.0010282662697136402
2320.14909413 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
2390.15398586 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
2520.16297057 0.0010334550170227885
2530.16366643 0.001034758286550641
254 0.16435352 0.0010334550170227885
```

```
255 0.16506052 0.0010422063060104847
256 0.1657235 0.0010422063060104847
257 0.16641217 0.0010409029200673103
258 0.16713175 0.0011633450631052256
2590.16799636 0.0011633450631052256
260 0.16887681 0.0011633450631052256
2 6 1 0 . 1 6 9 7 5 3 3 6 ~ 0 . 0 0 1 1 6 3 3 4 5 0 6 3 1 0 5 2 2 5 6
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
2 7 0 0 . 1 7 7 5 8 4 0 5 ~ 0 . 0 0 1 1 6 3 3 4 5 0 6 3 1 0 5 2 2 5 6
271 0.17844957 0.0011633450631052256
2 7 2 0 . 1 7 9 2 9 2 5 6 ~ 0 . 0 0 1 1 6 3 3 4 5 0 6 3 1 0 5 2 2 5 6
273 0.1801504 0.0011646484490484
274 0.1810208 0.0011633450631052256
2 7 5 0 . 1 8 1 8 5 5 3 5 ~ 0 . 0 0 1 1 6 3 3 4 5 0 6 3 1 0 5 2 2 5 6 ~
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
2 8 1 0 . 1 8 7 0 2 0 7 5 ~ 0 . 0 0 1 1 6 3 3 4 5 0 6 3 1 0 5 2 2 5 6
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
2890.19395226 0.0011633450631052256
290 0.19479892 0.0011646484490484
291 0.1956524 0.0011633450631052256
2920.19652039 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
3 1 1 0 . 2 1 2 7 0 5 9 4 ~ 0 . 0 0 1 1 5 4 7 6 0 9 4 6 5 1 9 6 7 2 9 ~
312 0.21352428 0.0011547609465196729
313 0.21437383 0.0011547609465196729
314 0.21522999 0.0011547609465196729
315 0.21606508 0.0011547609465196729
```

Data Science and Artificial Intelligence for Undergraduates

```
316 0.21693018 0.0011547609465196729
317 0.21776438 0.0011547609465196729
318 0.21860445 0.0011547609465196729
3190.21946377 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
3300.22871679 0.0011547609465196729
331 0.2295802 0.0011547609465196729
3320.23042527 0.0011547609465196729
333 0.23126474 0.0011547609465196729
334 0.23213267 0.0011547609465196729
3350.23293126 0.0011547609465196729
336 0.23380387 0.00119047611951828
337 0.23467505 0.00119047611951828
338 0.23550901 0.00119047611951828
3390.23637694 0.0011949328472837806
340 0.23720443 0.0011949328472837806
341 0.23803565 0.0011949328472837806
342 0.23889732 0.0011949328472837806
3430.23972046 0.0011859633959829807
344 0.24053669 0.0011859633959829807
345 0.24140269 0.0011859633959829807
346 0.2422227 0.0011859633959829807
3470.24304807 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
3550.24963304 0.0011859633959829807
356 0.25048518 0.0011801046784967184
357 0.25129056 0.0011693075066432357
3580.25210515 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
3710.26280937 0.0011801046784967184
372 0.2636469 0.0011801046784967184
373 0.2644574 0.0011801046784967184
3740.26530296 0.0011775860330089927
3750.26613316 0.0011883832048624754
376 0.26697263 0.0011775860330089927
```

```
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
4 0 3 ~ 0 . 2 8 9 2 7 7 0 5 ~ 0 . 0 0 1 1 7 5 6 1 5 9 3 6 5 1 7 7 1 5 5
404 0.2900658 0.0011756159365177155
4 0 5 0 . 2 9 0 9 0 8 4 3 ~ 0 . 0 0 1 1 7 5 6 1 5 9 3 6 5 1 7 7 1 5 5 ~
4 0 6 0 . 2 9 1 6 8 6 1 5 ~ 0 . 0 0 1 1 7 5 6 1 5 9 3 6 5 1 7 7 1 5 5 ~
4 0 7 0 . 2 9 2 4 9 9 5 4 ~ 0 . 0 0 1 1 7 5 6 1 5 9 3 6 5 1 7 7 1 5 5 ~
4 0 8 0 . 2 9 3 3 1 2 4 6 ~ 0 . 0 0 1 1 7 5 6 1 5 9 3 6 5 1 7 7 1 5 5 ~
4 0 9 ~ 0 . 2 9 4 1 2 6 3 3 ~ 0 . 0 0 1 1 7 5 6 1 5 9 3 6 5 1 7 7 1 5 5 ~
4 1 0 0 . 2 9 4 9 3 6 8 4 ~ 0 . 0 0 1 1 7 5 6 1 5 9 3 6 5 1 7 7 1 5 5
4 1 1 0 . 2 9 5 7 5 1 7 2 ~ 0 . 0 0 1 1 7 5 6 1 5 9 3 6 5 1 7 7 1 5 5 ~
4 1 2 0 . 2 9 6 5 5 3 8 5 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
413 0.2974703 0.0011670526582747698
4 1 4 0 . 2 9 8 3 8 0 2 3 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 1 5 0 . 2 9 9 2 6 3 2 4 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 1 6 0 . 3 0 0 2 1 1 3 4 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 1 7 0 . 3 0 1 1 1 3 6 4 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 1 8 0 . 3 0 2 0 1 5 6 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 1 9 ~ 0 . 3 0 2 9 5 1 8 4 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 2 0 0 . 3 0 3 8 5 1 6 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 2 1 ~ 0 . 3 0 4 7 4 4 4 8 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
422 0.3056714 0.0011670526582747698
4 2 3 ~ 0 . 3 0 6 5 8 3 0 2 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 2 4 ~ 0 . 3 0 7 5 1 7 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 2 5 ~ 0 . 3 0 8 4 2 8 4 4 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 2 6 ~ 0 . 3 0 9 3 1 1 1 2 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 2 7 0 . 3 1 0 2 1 4 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 2 8 ~ 0 . 3 1 1 1 2 6 2 3 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 2 9 ~ 0 . 3 1 2 0 5 4 2 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 3 0 0 . 3 1 2 9 2 1 3 2 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 3 1 0 . 3 1 3 8 4 3 6 7 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 3 2 0 . 3 1 4 7 6 3 3 7 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 3 3 0 . 3 1 5 6 5 4 5 8 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
434 0.3165592 0.0011670526582747698
4 3 5 0 . 3 1 7 4 9 3 9 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 3 6 0 . 3 1 8 3 8 9 0 3 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 3 7 0 . 3 1 9 2 9 6 4 8 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
```

Data Science and Artificial Intelligence for Undergraduates

```
4 3 8 ~ 0 . 3 2 0 1 9 0 5 2 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 3 9 0 . 3 2 1 0 9 9 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
440 0.32200316 0.0011670526582747698
4 4 1 0 . 3 2 2 9 1 1 1 4 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 4 2 0 . 3 2 3 7 9 9 5 5 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
443 0.32469973 0.0011670526582747698
444 0.32562268 0.0011670526582747698
4 4 5 0 . 3 2 6 5 0 7 5 7 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
446 0.32741365 0.0011670526582747698
44 0.3283232 0.0011670526582747698
448 0.3292402 0.0011670526582747698
4 4 9 0 . 3 3 0 1 2 9 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 5 0 0 . 3 3 1 0 3 3 0 5 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 5 1 ~ 0 . 3 3 1 9 3 3 6 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
452 0.3328155 0.0011670526582747698
4 5 3 0 . 3 3 3 7 4 7 3 3 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 5 4 ~ 0 . 3 3 4 6 4 7 8 6 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
4 5 5 0 . 3 3 5 5 3 8 9 2 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
456 0.3364692 0.0011670526582747698
4 5 7 0 . 3 3 7 3 7 8 4 4 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
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
4 6 3 0 . 3 4 2 7 8 7 6 2 ~ 0 . 0 0 1 1 6 7 0 5 2 6 5 8 2 7 4 7 6 9 8 )
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
4 7 1 0 . 3 4 9 7 5 7 5 ~ 0 . 0 0 1 0 7 9 0 6 5 2 5 9 5 4 6 0 4 1 5 ~
4 7 2 0 . 3 5 0 5 8 0 7 ~ 0 . 0 0 1 0 7 9 0 6 5 2 5 9 5 4 6 0 4 1 5 ~
473 0.3514599 0.0010790652595460415
4 7 4 0 . 3 5 2 2 8 3 7 ~ 0 . 0 0 1 0 7 9 0 6 5 2 5 9 5 4 6 0 4 1 5 ~
4 7 5 0 . 3 5 3 1 4 7 4 5 ~ 0 . 0 0 1 0 7 9 0 6 5 2 5 9 5 4 6 0 4 1 5 ~
476 0.35401186 0.0010790652595460415
477 0.35486692 0.0010790652595460415
4 7 8 0 . 3 5 5 6 8 1 5 4 ~ 0 . 0 0 1 0 7 9 0 6 5 2 5 9 5 4 6 0 4 1 5 ~
479 0.35657242 0.0010790652595460415
480 0.35737127 0.0010154407937079668
481 0.3581802 0.0010154407937079668
4 8 2 ~ 0 . 3 5 8 9 6 8 6 8 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
483 0.35978597 0.0010154407937079668
4 8 4 0 . 3 6 0 5 8 8 7 6 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
4 8 5 0 . 3 6 1 3 7 6 5 5 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
486 0.36217743 0.0010154407937079668
4 8 7 0 . 3 6 2 9 6 7 9 7 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
488 0.3637755 0.0010154407937079668
4 8 9 ~ 0 . 3 6 4 5 6 7 1 6 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
4 9 0 0 . 3 6 5 3 8 3 8 6 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
491 0.36614984 0.0010154407937079668
492 0.36697516 0.0010154407937079668
4 9 3 0 . 3 6 7 7 5 0 5 6 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
4 9 4 ~ 0 . 3 6 8 5 4 5 1 1 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
495 0.3693443 0.0010154407937079668
4 9 6 0 . 3 7 0 1 4 8 9 6 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
4 9 7 0 . 3 7 0 9 5 1 9 5 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
498 0.37172404 0.0010154407937079668
```

```
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
5 1 2 0 . 3 8 2 9 2 4 8 5 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
513 0.38370124 0.0010154407937079668
514 0.3845234 0.0010154407937079668
5 1 5 0 . 3 8 5 3 0 7 4 3 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
516 0.38611794 0.0010154407937079668
5 1 7 0 . 3 8 6 9 2 1 6 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
518 0.3876673 0.0010154407937079668
519 0.38849553 0.0010154407937079668
520 0.3892975 0.0010154407937079668
5 2 1 0 . 3 9 0 0 6 5 8 8 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
5 2 2 0 . 3 9 0 9 0 3 6 2 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
5 2 3 0 . 3 9 1 6 7 3 6 2 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
5 2 4 0 . 3 9 2 4 7 1 1 3 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
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
5 3 0 0 . 3 9 7 2 5 8 6 7 ~ 0 . 0 0 1 0 1 5 4 4 0 7 9 3 7 0 7 9 6 6 8 )
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
5390.40445825 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
5 5 3 ~ 0 . 4 1 5 7 0 6 3 4 ~ 0 . 0 0 1 0 1 0 5 1 3 3 9 8 7 9 6 3 2
554 0.4165171 0.00101051339879632
55 0.4173471 0.00101051339879632
5 5 6 0 . 4 1 8 1 3 0 6 4 ~ 0 . 0 0 1 0 1 0 5 1 3 3 9 8 7 9 6 3 2
5 5 7 0 . 4 1 8 9 3 3 8 ~ 0 . 0 0 1 0 1 0 5 1 3 3 9 8 7 9 6 3 2 )
558 0.41971493 0.00101051339879632
559 0.42053598 0.00101051339879632
```

```
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
6 1 1 0 . 4 6 1 5 9 6 0 . 0 0 1 0 5 9 0 2 8 9 0 2 0 9 1 0 8 6 ~
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
6 1 7 0 . 4 6 6 8 4 8 5 5 ~ 0 . 0 0 1 0 5 9 0 2 8 9 0 2 0 9 1 0 8 6 ~
618 0.46774 0.001059028902091086
6 1 9 0 . 4 6 8 6 3 3 2 ~ 0 . 0 0 1 0 5 9 0 2 8 9 0 2 0 9 1 0 8 6 ~
620 0.46951178 0.001059028902091086
```

```
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
6 2 8 0 . 4 7 6 8 8 9 9 7 ~ 0 . 0 0 1 0 3 2 6 9 8 8 9 9 5 0 7 5 2 2 6 ~
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
6550.5020299 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
6 6 6 0 . 5 1 1 9 7 2 8 4 ~ 0 . 0 0 1 0 3 2 6 9 8 8 9 9 5 0 7 5 2 2 6 ~
6670.51285535 0.0010326988995075226
6 6 8 0 . 5 1 3 7 7 3 1 4 ~ 0 . 0 0 1 0 3 2 6 9 8 8 9 9 5 0 7 5 2 2 6
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
6770.52187335 0.0010326988995075226
678 0.5227336 0.0010326988995075226
679 0.5236664 0.0010326988995075226
680 0.52454185 0.0010326988995075226
681 0.52543604 0.0010326988995075226
```

```
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
6970.5391491 0.0010307517368346453
698 0.5400001 0.0010307517368346453
6990.54090506 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
7 1 7 0 . 5 5 6 3 5 8 6 ~ 0 . 0 0 0 9 9 1 9 8 6 0 1 6 7 4 2 8 8 5 1 '
718 0.55718243 0.0009919860167428851
719 0.55800784 0.0010421713814139366
720 0.5589189 0.0010421713814139366
7 2 1 0 . 5 5 9 7 8 4 ~ 0 . 0 0 1 0 2 2 3 6 7 1 5 6 1 1 0 7 0 4 ~
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
7360.57226485 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
```

```
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 0.585659560.0009668168495409191
754 0.5864584 0.0009668168495409191
755 0.58722854 0.0009668168495409191
756 0.5880059 0.0009668168495409191
7 5 7 0 . 5 8 8 7 9 7 7 5 ~ 0 . 0 0 0 9 6 6 8 1 6 8 4 9 5 4 0 9 1 9 1 ~
758 0.5895954 0.0009668168495409191
7 5 9 0 . 5 9 0 3 7 6 5 6 ~ 0 . 0 0 0 9 6 6 8 1 6 8 4 9 5 4 0 9 1 9 1 ~
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
7 7 1 0 . 5 9 9 8 2 7 2 3 ~ 0 . 0 0 0 9 6 6 8 1 6 8 4 9 5 4 0 9 1 9 1 ~
772 0.6006173 0.0009668168495409191
7 7 3 0 . 6 0 1 4 0 6 0 4 ~ 0 . 0 0 0 9 6 6 8 1 6 8 4 9 5 4 0 9 1 9 1 ~
7 7 4 0 . 6 0 2 1 9 7 2 3 ~ 0 . 0 0 0 9 6 6 8 1 6 8 4 9 5 4 0 9 1 9 1 ~
775 0.6029881 0.0009668168495409191
776 0.6037439 0.0009668168495409191
7 7 7 0 . 6 0 4 5 5 0 . 0 0 0 9 6 6 8 1 6 8 4 9 5 4 0 9 1 9 1 ,
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
7 8 4 ~ 0 . 6 1 0 0 7 4 9 4 ~ 0 . 0 0 0 9 6 6 8 1 6 8 4 9 5 4 0 9 1 9 1 ~
85 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
7 9 2 0 . 6 1 6 3 5 3 2 ~ 0 . 0 0 0 9 8 8 6 8 9 8 3 3 3 2 0 6 7 7 3 ~
793 0.6171217 0.0009886898333206773
794 0.6179187 0.0009886898333206773
7 9 5 0 . 6 1 8 6 8 6 2 6 ~ 0 . 0 0 0 9 8 8 6 8 9 8 3 3 3 2 0 6 7 7 3 ~
796 0.6194822 0.0009886898333206773
797 0.6202694 0.0009886898333206773
7 9 8 0 . 6 2 1 0 4 6 3 6 ~ 0 . 0 0 0 9 8 8 6 8 9 8 3 3 3 2 0 6 7 7 3 ~
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
```

```
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
8120.6320016 0.0009886898333206773
813 0.63276213 0.0009979268070310354
814 0.6335460.0009979268070310354
8150.6343409 0.0009979268070310354
8160.63511616 0.0009979268070310354
8170.63590056 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
8360.6517256 0.0009945675265043974
8370.6525289 0.0009945675265043974
838 0.65336 0.0009945675265043974
8390.65417266 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
8450.6590987 0.0009945675265043974
8460.65990216 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
8560.66809356 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
```

```
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
8740.6827903 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
8850.6916016 0.0009995902655646205
886 0.6924194 0.0009995902655646205
8 8 7 0 . 6 9 3 2 1 1 ~ 0 . 0 0 0 9 9 9 5 9 0 2 6 5 5 6 4 6 2 0 5 ~
888 0.6940029 0.0009995902655646205
889 0.6947848 0.0009995902655646205
890 0.6955754 0.0009995902655646205
81 0.69638306 0.0009995902655646205
892 0.69717735 0.0009995902655646205
893 0.697969 0.0009995902655646205
894 0.69876575 0.0009995902655646205
895 0.6995406 0.0009995902655646205
8960.7003432 0.0009995902655646205
897 0.70114625 0.0009995902655646205
898 0.70194095 0.0009995902655646205
8990.70272666 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
9110.7122621 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 0.0010522683151066303
918 0.71787304 0.0010522683151066303
919 0.7187354 0.0010522683151066303
920 0.7195928 0.0010522683151066303
921 0.72045165 0.0010522683151066303
922 0.7212997 0.0010522683151066303
923 0.7221617 0.0010522683151066303
924 0.72301835 0.0010522683151066303
925 0.72386307 0.0010522683151066303
```

```
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
9320.72984874 0.0010522683151066303
933 0.7307101 0.0010522683151066303
934 0.7315547 0.0010522683151066303
9350.73242086 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
9520.7464003 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
9690.7597001 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
```

| 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)
```

Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

```
bb_row_start = row_mask.argmax()
b.b_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 ${ }^{249}$ from 2015 uses the techniques discussed above. A similar application of dreaming CNNs is neural style transfer ${ }^{250}$, also appearing in 2015.

[^84]
### 8.6.4 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, :, :,r
\hookrightarrowfmap_index]))
img = 255 * np.random.default_rng(0).normal(0.5, 0.2, size=(img_size, img_size,
    ५3))
# 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()
```

```
0.010571016 2.397079242655309e-06
0.0129878872.784569460345665e-06
0.01555704 2.6928230454359436e-06
0.018290414 2.2864280708745355e-06
0.021174435 2.192614147134009e-06
0.024220329 2.4407656837865943e-06
0.027513845 2.4178386865969514e-06
0.03095166 2.223908268206287e-06
0.03454724 2.0893653527309652e-06
0.038174365 2.028772314588423e-06
0.041862242 2.097629021591274e-06
0.045696992 2.3129020974010928e-06
0.049698126 2.385100970059284e-06
0.05387558 2.3095196866051992e-06
0.058383714 2.396672925897292e-06
0.063163 2.3852505819377257e-06
0.068157524 2.510322474336135e-06
0.07321679 2.4331638996955007e-06
0.07840866 2.3761479042150313e-06
0.08379156 2.492263092790381e-06
0.089350894 2.4447754185530357e-06
0.09505746 2.397075149929151e-06
0.10108615 2.545427378208842e-06
0.10727019 2.3443969894287875e-06
0.11355108 2.2837014057586202e-06
0.11987263 2.431382426948403e-06
0.12621516 2.6055895432364196e-06
0.13275892 2.548260454204865e-06
0.13940799 2.536550937293214e-06
0.14614432 2.5595429633540334e-06
0.15306608 2.4765708985796664e-06
0.1601255 2.4590774501120904e-06
0.16713977 2.3455713744624518e-06
0.17422041 2.3978557237569476e-06
0.18140693 2.474193479429232e-06
```

```
350.1886376 2.5417300548724597e-06
0.19593534 2.533454107833677e-06
0.20329633 2.704197186176316e-06
0.21070997 2.6751181394502055e-06
0.21831936 2.7035023322241614e-06
0.2259554 2.7090159164799843e-06
0.23365016 2.4669375306984875e-06
0.24140108 2.3419049739459297e-06
0.24929094 2.5573299353709444e-06
0.25731435 2.5564454517734703e-06
0.26534614 2.4323860543518094e-06
0.2732757 2.296659886269481e-06
0.28120157 2.4815644792397507e-06
0.28913957 2.3013376448943745e-06
0.29706028 2.140596961908159e-06
0.30498374 1.9981957848358434e-06
0.3129182 2.2039150735508883e-06
0.32090297 2.156832579203183e-06
0.32904193 2.1195842236920726e-06
0.33714753 2.443357743686647e-06
0.34530818 2.8032877708028536e-06
0.3534469 2.4665589535288746e-06
0.36151606 2.3440138647856656e-06
0.3696367 2.518916517146863e-06
0.37779084 2.2818105662736343e-06
0.38606593 2.8909905722684925e-06
0.39434585 2.2529472971655196e-06
0.4026392 2.309225692442851e-06
0.4110172 2.1857999854546506e-06
0.41925484 2.1461057713167975e-06
0.42749077 2.2192639335116837e-06
0.4357061 2.193360160163138e-06
0.44399276 2.109282604578766e-06
0.45219928 2.2869414806336863e-06
0.46037853 2.239397417724831e-06
0.4686118 2.8001759346807376e-06
0.47689474 2.122434580087429e-06
0.48513776 2.1441801436594687e-06
0.4933584 2.334981900276034e-06
0.5015519 2.120641738656559e-06
0.50966257 2.1056046080047963e-06
0.5177873 2.0936017790518235e-06
0.525847 2.438975798213505e-06
0.53396064 2.3880647859186865e-06
0.5421345 2.4256362394226016e-06
0.5503483 2.482177706042421e-06
0.55855685 2.787142193483305e-06
0.56671786 3.04928016703343e-06
0.5748916 2.9708369311265415e-06
0.5830835 2.8886945528938668e-06
0.5912361 2.8094241315557156e-06
0.59929144 2.7368155315343756e-06
0.60734797 2.5096640001720516e-06
0.6153681 2.514457037250395e-06
0.6233643 2.5291581096098525e-06
0.6313867 2.600254674689495e-06
0.6394314 2.527786591599579e-06
0.64746135 2.4455698621750344e-06
0.6554675 2.67075483861845e-06
0.6634189 2.3975730982783716e-06
0.67137367 2.2373308183887275e-06
```

```
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
1240.89243096 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
1391.000673 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
```

```
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
1791.266821 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 1.6312409343299805e-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
```

```
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
2491.6890433 1.728237066345173e-06
250 1.6949303 1.6089852579170838e-06
251 1.7008134 1.7228853721462656e-06
252 1.7066984 1.6063656858023023e-06
2 5 3 1 . 7 1 2 5 7 5 7 ~ 1 . 6 8 7 8 4 4 7 0 1 3 3 6 8 7 1 4 e - 0 6 ~
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 1.7193692656292114e-06
268 1.8004186 1.6063656858023023e-06
269 1.8062667 1.6225779972955934e-06
270 1.8121052 1.7510262750874972e-06
271 1.8179452 1.6642696891722153e-06
272 1.8237971 1.6239438309639809e-06
273 1.8296481 1.6642696891722153e-06
2 7 4 1 . 8 3 5 4 7 4 1 ~ 1 . 6 6 7 8 8 0 1 5 5 7 6 4 5 7 8 4 e - 0 6 ~
275 1.8413198 1.620687726244796e-06
276 1.8471508 1.6626680690023932e-06
277 1.8529727 1.5862756299611647e-06
278 1.8588016 1.5581568959532888e-06
```

```
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
3302.1606586 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
```

```
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
3502.2767446 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
```

```
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
4 1 7 ~ 2 . 6 6 8 4 5 1 8 ~ 1 . 7 2 5 3 1 3 8 3 6 6 8 6 8 3 7 e - 0 6 ~
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
4 2 5 ~ 2 . 7 1 5 6 0 2 4 ~ 1 . 7 1 9 4 0 5 0 7 6 9 8 3 0 9 3 7 e - 0 6 ~
426 2.7214992 1.689453029030119e-06
4 2 7 ~ 2 . 7 2 7 3 9 ~ 1 . 7 9 0 3 7 4 6 5 3 8 6 5 0 0 1 2 e - 0 6 ~
4 2 8 ~ 2 . 7 3 3 2 8 7 3 ~ 1 . 7 2 8 3 6 5 5 3 2 4 7 1 7 9 8 4 e - 0 6 ~
429 2.739193 1.6751293969718972e-06
430 2.7450848 1.6778164990682853e-06
4 3 1 2 . 7 5 0 9 8 0 6 ~ 1 . 6 3 5 7 3 0 5 4 1 2 3 8 4 4 4 9 e - 0 6 ~
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
4 3 8 ~ 2 . 7 9 2 2 6 4 ~ 1 . 6 3 9 1 8 6 7 3 4 7 9 2 0 1 9 7 e - 0 6 ~
4 3 9 2 . 7 9 8 1 5 4 ~ 1 . 7 0 0 5 8 8 9 9 5 8 4 5 4 6 4 5 e - 0 6 ~
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
4 5 5 ~ 2 . 8 9 2 5 4 8 8 ~ 1 . 7 0 6 4 1 0 5 5 7 7 4 4 6 7 5 3 e - 0 6 ~
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
```

```
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
4 7 2 ~ 2 . 9 9 3 0 7 3 5 ~ 1 . 7 0 4 8 6 2 7 1 1 4 4 9 0 9 5 5 e - 0 6 ~
473 2.9990044 1.669786229285819e-06
4 7 4 ~ 3 . 0 0 4 9 4 0 3 ~ 1 . 7 5 6 1 0 0 2 3 2 3 4 1 8 5 0 6 e - 0 6 ~
4 7 5 3 . 0 1 0 8 8 6 ~ 1 . 6 7 8 6 9 0 5 2 3 4 7 6 6 8 9 1 e - 0 6 ~
4 7 6 ~ 3 . 0 1 6 8 4 0 2 ~ 1 . 7 1 3 4 4 3 9 0 7 6 4 7 1 6 0 8 e - 0 6 ~
477 3.0227835 1.8284858924744185e-06
4 7 8 ~ 3 . 0 2 8 7 3 5 2 ~ 1 . 6 5 1 4 8 8 3 3 2 7 5 4 5 2 4 9 e - 0 6 ~
4 7 9 3 . 0 3 4 6 8 5 9 ~ 1 . 6 6 1 6 1 0 3 2 6 6 6 4 2 3 1 3 e - 0 6 ~
480 3.0406318 1.7071628235498792e-06
4 8 1 ~ 3 . 0 4 6 5 8 4 6 ~ 1 . 7 4 2 0 7 1 5 0 3 9 4 0 6 7 8 7 e - 0 6 ~
482 3.0525298 1.64506320743385e-06
483 3.058484 1.6777397604528232e-06
4 8 4 ~ 3 . 0 6 4 4 3 6 ~ 1 . 6 9 6 5 2 2 1 9 0 2 8 6 4 8 6 9 e - 0 6 ~
485 3.0704076 1.6948403072092333e-06
4 8 6 ~ 3 . 0 7 6 3 5 5 5 ~ 1 . 7 1 4 2 3 8 8 0 6 0 1 6 5 1 0 4 e - 0 6 ~
4 8 7 3 . 0 8 2 3 2 7 8 ~ 1 . 6 5 7 4 9 6 3 4 1 0 6 7 5 9 9 1 e - 0 6 ~
4 8 8 ~ 3 . 0 8 8 2 7 2 6 ~ 1 . 7 4 0 2 8 6 6 2 0 5 8 8 4 4 9 3 e - 0 6 ~
4 8 9 3 . 0 9 4 2 4 2 3 ~ 1 . 6 5 3 7 6 9 2 3 1 7 7 9 7 3 3 7 e - 0 6 ~
4 9 0 ~ 3 . 1 0 0 2 0 8 5 ~ 1 . 7 1 8 9 3 7 0 2 8 2 7 2 1 9 3 9 e - 0 6 ~
491 3.106177 1.703623070170579e-06
492 3.112145 1.641988546907669e-06
4 9 3 ~ 3 . 1 1 8 1 2 7 6 ~ 1 . 6 6 0 8 7 2 3 8 5 4 0 0 5 8 0 2 e - 0 6 ~
4 9 4 ~ 3 . 1 2 4 1 0 3 5 ~ 1 . 6 8 8 6 1 7 9 9 9 2 0 7 0 5 3 8 e - 0 6 ~
495 3.1300848 1.80542645011883e-06
4 9 6 ~ 3 . 1 3 6 0 7 7 2 ~ 1 . 6 6 6 8 7 1 7 5 3 5 1 3 9 8 7 6 e - 0 6 ~
4 9 7 3 . 1 4 2 0 6 3 1 ~ 1 . 7 4 8 4 9 5 7 1 9 7 6 6 6 5 1 7 e - 0 6 ~
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
```

Data Science and Artificial Intelligence for Undergraduates

```
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
```

```
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
6394.0072265 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
```

```
645 4.0442743 1.6988828974717762e-06
646 4.050464 1.6111703189380933e-06
6474.056641 1.8734641571427346e-06
648 4.0628304 1.6050108797571738e-06
6494.069009 1.7011698218993843e-06
6504.0751905 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
6554.1061044 1.6589945062150946e-06
6564.1122856 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
6694.1927905 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
6784.248543 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
6904.3229656 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
6964.360256 1.7122670215030666e-06
6974.3664637 1.7800831528802519e-06
698 4.3726835 1.6175806649698643e-06
6994.378895 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
```

```
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
7194.503376 1.7222489532287e-06
7204.509608 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
7294.5657253 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
7354.603169 1.6479848454764578e-06
736 4.6094174 1.7051301028914168e-06
77 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
7494.690659 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
7594.7532816 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
```

```
7674.803445 1.7297710428465507e-06
768 4.809731 1.720574573482736e-06
7694.816003 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
7754.853673 1.6988705056064646e-06
7764.859957 1.6846032622197527e-06
777 4.866243 1.7799635543269687e-06
778 4.8725348 1.7368907947457046e-06
7794.8788166 1.573215854477894e-06
780 4.885112 1.7900595139508368e-06
7814.891404 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
7904.948056 1.6644481775074382e-06
7914.954363 1.8117095805791905e-06
792 4.9606543 1.640500386201893e-06
793 4.9669495 1.6357402046196512e-06
794 4.9732513 1.7153088265331462e-06
7954.9795537 1.693157855697791e-06
7964.985849 1.6430649338872172e-06
7974.992156 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
8065.0488806 1.6543017409276217e-06
807 5.05519 1.8300238480151165e-06
808 5.0615 1.6380139413740835e-06
8095.0678115 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
8205.1372495 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
8265.175143 1.8991477190866135e-06
827 5.1814575 1.7334813264824334e-06
```

```
828 5.1877775 1.6132496512000216e-06
8295.1941004 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
8475.3080063 1.7163685015475494e-06
848 5.3143487 1.7763649111657287e-06
849 5.3206854 1.6711087482690345e-06
8505.3270345 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
8565.3650904 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
```

```
8895.574962 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
9295.830119 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
9405.9005475 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 5.94539361.7010621604640619e-06
948 5.9518075 1.670299525358132e-06
949 5.958211 1.6554276953684166e-06
```

```
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
9696.086478 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
9806.1570954 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
9906.221336 1.7308377664448926e-06
991 6.2277627 1.7490103800810175e-06
992 6.2341957 1.73156490745896e-06
9936.2406282 1.7092773987315013e-06
994 6.247062 1.7839250858742162e-06
995 6.253493 1.768219590303488e-06
9966.2599254 1.6105500435514841e-06
997 6.2663507 1.867201490313164e-06
998 6.2727757 1.5852081105549587e-06
9996.2792096 1.7314188198724878e-06
```



### 8.6.5 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,
    4786, 907, 1696
img = np.asarray(img, dtype=np.float 32)
```

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
```

Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning


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 $3 \times 3$ filter. Thus a $3 \times 3$ region is the preimage of the pixel. One layer up we have a $5 \times 5$ region (convolution with $3 \times 3$ filter again). Then there is a pooling layer. So the ROI's size before pooling is $10 \times 10$. Then again two $3 \times 3$ convolutions, yielding a $14 \times 14$ ROI.
The CAM is $58 \times 58$. The original image is $128 \times 128$. Centers of all ROIs have to be placed equally spaced in the $128 \times 128$ 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)

```
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()
```



### 8.7 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.

### 8.7.1 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 ${ }^{251}$.

### 8.7.2 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 ${ }^{252}$ 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.

### 8.7.3 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.

[^85]
### 8.7.4 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.

### 8.7.5 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.
    \iotacc:193] This TensorFlow binary is optimized with oneAPI Deep Neural Networkь
    Library (oneDNN) to use the following CPU instructions in performance-
    Gcritical 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.float 32 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 Dat aset 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(
```


# Data Science and Artificial Intelligence for Undergraduates 

 Volume 2: Data Visualization, Supervised Learning```
    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_
$\rightarrow$ 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_
$\rightarrow d i a g n o s t i c s . c c: 176]$ 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_
$\hookrightarrow d i a g n o s t i c s . c c: 310]$ kernel version seems to match DSO: 470.161.3
2023-05-08 07:12:34.818982: I tensorflow/core/platform/cpu_feature_guard.
$\varsigma C C: 193]$ This TensorFlow binary is optimized with oneAPI Deep Neural Network
$\hookrightarrow$ 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
$\hookrightarrow$ 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.
$\hookrightarrow C C: 392]$ 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.
५cc:417] Shuffle buffer filled.
2023-05-08 07:12:46.304312: W tensorflow/tsl/framework/cpu_allocator_impl.
५cc:82] Allocation of 2949120000 exceeds $10 \%$ of free system memory.
2023-05-08 07:12:51.302941: W tensorflow/tsl/framework/cpu_allocator_impl.
$\hookrightarrow C C: 82]$ Allocation of 983040000 exceeds $10 \%$ of free system memory.
2023-05-08 07:12:56.156447: W tensorflow/tsl/framework/cpu_allocator_impl.
$\rightarrow c c: 82]$ 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 ${ }^{253}$ for current discussion.

[^86]```
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) | $\begin{aligned} & \text { [(None, } 128,128,3 \\ & \text { )] } \end{aligned}$ | 0 | [] |
| $\begin{aligned} & \text { block1_conv1 (Conv2D) } \\ & \left.\hookrightarrow^{\prime}\right] \end{aligned}$ | (None, 63, 63, 32) | 864 | ['input_2[0][0] |
| ```block1_conv1_bn (BatchNormaliz \hookrightarrowconv1[0][0]'] ation)``` | (None, 63, 63, 32) | 128 | ['block1_ |
| $\begin{aligned} & \text { block1_conv1_act (Activation) } \\ & \text { ↔bn[0][0]'] } \end{aligned}$ | (None, 63, 63, 32) | 0 | ['block1_conv1_ |
| $\begin{aligned} & \text { block1_conv2 (Conv2D) } \\ & \text { ↔act[0][0]'] } \end{aligned}$ | (None, 61, 61, 64) | 18432 | ['block1_conv1_ |
| ```block1_conv2_bn (BatchNormaliz cconv2[0][0]'] ation)``` | (None, 61, 61, 64) | 256 | ['block1_ |
| ```block1_conv2_act (Activation) 4bn[0][0]']``` | (None, 61, 61, 64) | 0 | ['block1_conv2_ |
| ```block2_sepconv1 (SeparableConv aact[0][0]'] 2D)``` | (None, 61, 61, 128) | 8768 | ['block1_conv2_ |
| ```block2_sepconv1_bn (BatchNorma ↔sepconv1[0][0]'] lization)``` | (None, 61, 61, 128) | 512 | ['block2_ |
| ```block2_sepconv2_act (Activatio ムsepconv1_bn[0][0]'] n)``` | (None, 61, 61, 128) | 0 | ['block2_ |

(continues on next page)

```
block2_sepconv2 (SeparableConv
4sepconv2_act[0][0]']
2D)
block2_sepconv2_bn (BatchNorma
4sepconv2[0][0]']
lization)
conv2d (Conv2D)
4act[0][0]']
block2_pool (MaxPooling2D)
ssepconv2_bn[0][0]']
batch_normalization (BatchNorm
\hookrightarrow']
alization)
add (Add)
4pool[0][0]',
\hookrightarrownormalization[0][0]']
block3_sepconv1_act (Activatio
n)
block3_sepconv1 (SeparableConv
4sepconv1_act[0][0]']
2D)
block3_sepconv1_bn (BatchNorma
\hookrightarrowsepconv1[0][0]']
lization)
block3_sepconv2_act (Activatio
4sepconv1_bn[0][0]']
n)
block3_sepconv2 (Sepa
block3_sepconv2_bn (BatchNorma (None, 31, 31, 256) 1024
\hookrightarrowsepconv2[0][0]']
lization)
conv2d_1 (Conv2D)
block3_pool (MaxPooling2D)
4sepconv2_bn[0][0]']
batch_normalization_1 (BatchNo
41[0][0]']
rmalization)
add_1 (Add)
    (None, 16, 16, 256) 0
block3_
4pool[0][0]',
\leftrightarrowsnormalization_1[0][0]']
block4_sepconv1_act (Activatio (None, 16, 16, 256) 0
```

```
n)
block4 sepconv1 (SeparableConv
↔sepconv1_act[0][0]']
2D)
block4_sepconv1_bn (BatchNorma
\leftrightarrowssepconv1[0][0]']
lization)
block4_sepconv2_act (Activatio
\hookrightarrowsepconv1_bn[0][0]']
n)
block4_sepconv2 (SeparableConv (None, 16, 16, 728) 536536 ['block4_
4sepconv2_act[0][0]']
2D)
block4_sepconv2_bn (BatchNorma (None, 16, 16, 728) 2912
\hookrightarrowsepconv2[0][0]']
lization)
conv2d_2 (Conv2D)
block4_pool (MaxPooling2D)
\hookrightarrowsepconv2_bn[0][0]']
batch_normalization_2 (BatchNo
->2[0][0]']
rmalization)
add_2 (Add)
->pool[0][0]',
\leftrightarrowsnormalization_2[0][0]']
n)
block5_sepconv1 (SeparableConv (None, 8, 8, 728) 536536 ['block5_
\hookrightarrowsepconv1_act[0][0]']
2D)
block5_sepconv1_bn (BatchNorma (None, 8, 8, 728) 2912
ヶsepconv1[0][0]']
lization)
block5_sepconv2_act (Activatio (None, 8, 8, 728) 0 ['block5
\hookrightarrowsepconv1_bn[0][0]']
n)
block5_sepconv2 (SeparableConv (None, 8, 8, 728) 536536 ['block5_
\hookrightarrowsepconv2_act[0][0]']
2D)
block5_sepconv2_bn (BatchNorma (None, 8, 8, 728) 2912
4sepconv2[0][0]']
lization)
block5_sepconv3_act (Activatio (None, 8, 8, 728) 0
['block5_
```

```
n)
block5 sepconv3 (SeparableConv
4sepconv3_act[0][0]']
2D)
block5_sepconv3_bn (BatchNorma (None, 8, 8, 728) 2912
\hookrightarrowsepconv3[0][0]']
lization)
add_3 (Add)
ヶsepconv3_bn[0][0]',
block6_sepconv1_act (Activatio
n)
block6_sepconv1 (SeparableConv
ssepconv1_act[0][0]']
2D)
block6_sepconv1_bn (BatchNorma
\hookrightarrowsepconv1[0][0]']
lization)
block6_sepconv2_act (Activatio
4sepconv1_bn[0][0]']
n)
block6_sepconv2 (SeparableConv
4sepconv2_act[0][0]']
2D)
block6_sepconv2_bn (BatchNorma
ssepconv2[0][0]']
lization)
block6_sepconv3_act (Activatio
ムsepconv2_bn[0][0]']
n)
block6_sepconv3 (SeparableConv
2D)
block6_sepconv3_bn (BatchNorma (None, 8, 8, 728) 2912
4sepconv3[0][0]']
lization)
add_4 (Add)
ムsepconv3_bn[0][0]',
block7_sepconv1_act (Activatio (None, 8, 8, 728)
n)
block7_sepconv1 (SeparableConv (None, 8, 8, 728) 536536 ['block7_
ムsepconv1_act[0][0]']
2D)
block7_sepconv1_bn (BatchNorma (None, 8, 8, 728) 2912
['block7_
ssepconv1[0][0]']
    None, 8, 8, 728) 0
```

```
(continued from previous page)
lization)
block7_sepconv2_act (Activatio
&sepconv1_bn[0][0]']
n)
block7_sepconv2 (SeparableConv
\leftrightarrowsepconv2_act[0][0]']
2D)
block7_sepconv2_bn (BatchNorm
4sepconv2[0][0]']
lization)
block7_sepconv3_act (Activatio
\hookrightarrowsepconv2_bn[0][0]']
n)
block7_sepconv3 (SeparableConv
4sepconv3_act[0][0]']
2D)
block7_sepconv3_bn (BatchNorma
↔sepconv3[0][0]']
lization)
add_5 (Add)
4sepconv3_bn[0][0]',
n)
block8_sepconv1 (Sepa
2D)
block8_sepconv1_bn (BatchNorma (None, 8, 8, 728) 2912
ムsepconv1[0][0]']
lization)
block8_sepconv2_act (Activatio (None, 8, 8, 728)
«sepconv1_bn[0][0]']
n)
block8_sepconv2 (SeparableConv (None, 8, 8, 728) 536536 ['block8_
\hookrightarrowsepconv2_act[0][0]']
2D)
block8_sepconv2_bn (BatchNorma (None, 8, 8, 728) 2912
↔sepconv2[0][0]']
lization)
block8_sepconv3_act (Activatio (None, 8, 8, 728)
ssepconv2_bn[0][0]']
n)
block8_sepconv3 (SeparableConv (None, 8, 8, 728) 536536 ['block8_
\hookrightarrowsepconv3_act[0][0]']
2D)
```

```
block8_sepconv3_bn (BatchNorma
\leftrightarrowssepconv3[0][0]']
lization)
add_6 (Add)
ssepconv3_bn[0][0]',
block9_sepconv1_act (Activatio
n)
block9_sepconv1 (SeparableConv
↔sepconv1_act[0][0]']
2D)
block9_sepconv1_bn (BatchNorma
\hookrightarrowsepconv1[0][0]']
lization)
block9_sepconv2_act (Activatio
ムsepconv1_bn[0][0]']
n)
block9_sepconv2 (SeparableConv
\hookrightarrowsepconv2_act[0][0]']
2D)
block9_sepconv2_bn (BatchNorma
↔sepconv2[0][0]']
lization)
block9_sepconv3_act (Activatio
ムsepconv2_bn[0][0]']
n)
block9_sepconv3 (Sepa
block9_sepconv3_bn (BatchNorma (None, 8, 8, 728) 2912
\hookrightarrowsepconv3[0][0]']
lization)
add_7 (Add)
ムsepconv3_bn[0][0]',
block10_sepconv1_act (Activati (None, 8, 8, 728) 0
on)
block10_sepconv1 (SeparableCon (None, 8, 8, 728) 536536 ['block10_
ssepconv1_act[0][0]']
v2D)
block10_sepconv1_bn (BatchNorm (None, 8, 8, 728) 2912
Gsepconv1[0][0]']
alization)
block10_sepconv2_act (Activati (None, 8, 8, 728) 0
๑sepconv1_bn[0][0]']
on)
['block8_
    'add_5[0][0]']
['add_6[0][0]']
['block9_
['block9_
['block9_
['block9_
['block9_
['block9_
['block9_
['block9_
['block9_
    'add_6[0][0]']
['add_7[0][0]']
['block10_
['block10_
```

（continued from previous page）

```
['block8_
```

```
['block8_
```

（continues on next page）

```
block10_sepconv2 (Sep
v2D)
```

block10_sepconv2_bn (BatchNorm (None, 8, 8, 728)
$\rightarrow$ sepconv2[0][0]']
alization)
block10_sepconv3_act (Activati (None, 8, 8, 728) 0
$\rightarrow$ sepconv2_bn[0][0]']
on)
block10_sepconv3 (SeparableCon
↔sepconv3_act[0][0]']
v2D)
block10_sepconv3_bn (BatchNorm
↔sepconv3[0][0]']
alization)
add_8 (Add)
ssepconv3_bn [0] [0]',
on)
block11_sepconv1 (SeparableCon
$\rightarrow$ sepconv1_act[0][0]']
v2D)
block11_sepconv1_bn (BatchNorm
↔sepconv1[0][0]']
alization)
block11_sepconv2_act (Activati (None, 8, 8, 728) 0
$\rightarrow$ sepconv1_bn[0][0]'
on)
block11_sepconv2 (SeparableCon
↔sepconv2_act[0][0]']
v2D)
block11_sepconv2_bn (BatchNorm
↔sepconv2[0][0]']
alization)
block11_sepconv3_act (Activati (None, 8, 8, 728) 0
$\rightarrow$ sepconv2_bn[0][0]']
on)
block11_sepconv3 (Sep
sepconv3_act [0][0]']
v2D)
block11_sepconv3_bn (BatchNorm
↔sepconv3[0][0]']
alization)
add_9 (Add)
$\rightarrow$ sepconv3_bon[0][0]'


```
(continued from previous page)
    'add_8[0][0]']
```

['add_9[0][0]']

```
['add_9[0][0]']
['block12_
['block12_
['block12_
['block12_
['block12_
['block12_
['block12_
['block12_
['block12_
['block12_
4sepconv2[0][0]']
4sepconv2[0][0]']
alization)
alization)
block12_sepconv3_act (Activati (None, 8, 8, 728) 0
block12_sepconv3_act (Activati (None, 8, 8, 728) 0
4sepconv2_bn[0][0]']
4sepconv2_bn[0][0]']
on)
```

on)

```
```

block12_sepconv3 (SeparableCon (None, 8, 8, 728) 536536

```
block12_sepconv3 (SeparableCon (None, 8, 8, 728) 536536
```

block12_sepconv3 (SeparableCon (None, 8, 8, 728) 536536

```
block12_sepconv3 (SeparableCon (None, 8, 8, 728) 536536
\leftrightarrowssepconv3_act[0][0]']
\leftrightarrowssepconv3_act[0][0]']
\leftrightarrowssepconv3_act[0][0]']
\leftrightarrowssepconv3_act[0][0]']
v2D)
v2D)
v2D)
v2D)
\hookrightarrowsepconv3[0][0]']
\hookrightarrowsepconv3[0][0]']
\hookrightarrowsepconv3[0][0]']
\hookrightarrowsepconv3[0][0]']
alization)
alization)
alization)
alization)
add_10 (Add)
add_10 (Add)
add_10 (Add)
add_10 (Add)
    (None, 8, 8, 728)
    (None, 8, 8, 728)
    (None, 8, 8, 728)
    (None, 8, 8, 728)
«sepconv3_bn[0][0]',
«sepconv3_bn[0][0]',
«sepconv3_bn[0][0]',
«sepconv3_bn[0][0]',
block13_sepconv1_act (Activati (None, 8, 8, 728)
block13_sepconv1_act (Activati (None, 8, 8, 728)
block13_sepconv1_act (Activati (None, 8, 8, 728)
block13_sepconv1_act (Activati (None, 8, 8, 728)
`']
`']
`']
`']
on)
on)
on)
on)
block13_sepconv1 (SeparableCon (None, 8, 8, 728) 536536
block13_sepconv1 (SeparableCon (None, 8, 8, 728) 536536
block13_sepconv1 (SeparableCon (None, 8, 8, 728) 536536
block13_sepconv1 (SeparableCon (None, 8, 8, 728) 536536
4sepconv1_act[0][0]']
4sepconv1_act[0][0]']
4sepconv1_act[0][0]']
4sepconv1_act[0][0]']
v2D)
v2D)
v2D)
v2D)
block13_sepconv1_bn (BatchNorm (None, 8, 8, 728) 2912
block13_sepconv1_bn (BatchNorm (None, 8, 8, 728) 2912
block13_sepconv1_bn (BatchNorm (None, 8, 8, 728) 2912
block13_sepconv1_bn (BatchNorm (None, 8, 8, 728) 2912
4sepconv1[0][0]']
4sepconv1[0][0]']
4sepconv1[0][0]']
4sepconv1[0][0]']
alization)
alization)
alization)
alization)
block13_sepconv2_act (Activati (None, 8, 8, 728) 0
block13_sepconv2_act (Activati (None, 8, 8, 728) 0
block13_sepconv2_act (Activati (None, 8, 8, 728) 0
block13_sepconv2_act (Activati (None, 8, 8, 728) 0
ssepconv1_bn[0][0]']
ssepconv1_bn[0][0]']
ssepconv1_bn[0][0]']
ssepconv1_bn[0][0]']
on)
on)
on)
on)
block13_sepconv2 (SeparableCon (None, 8, 8, 1024) 752024 ['block13_
block13_sepconv2 (SeparableCon (None, 8, 8, 1024) 752024 ['block13_
block13_sepconv2 (SeparableCon (None, 8, 8, 1024) 752024 ['block13_
block13_sepconv2 (SeparableCon (None, 8, 8, 1024) 752024 ['block13_
4sepconv2_act[0][0]']
4sepconv2_act[0][0]']
4sepconv2_act[0][0]']
4sepconv2_act[0][0]']
v2D)
```

v2D)

```
```

block12 sepconv1 act (Activati (None, 8, 8, 728)

```
block12 sepconv1 act (Activati (None, 8, 8, 728)
on)
on)
block12_sepconv1 (SeparableCon (None, 8, 8, 728) 536536
block12_sepconv1 (SeparableCon (None, 8, 8, 728) 536536
↔sepconv1_act [0] [0]']
↔sepconv1_act [0] [0]']
v2D)
v2D)
\hookrightarrowsepconv1[0][0]']
\hookrightarrowsepconv1[0][0]']
alization)
alization)
block12_sepconv2_act (Activati (None, 8, 8, 728) 0
block12_sepconv2_act (Activati (None, 8, 8, 728) 0
\hookrightarrowsepconv1_bn[0][0]']
\hookrightarrowsepconv1_bn[0][0]']
on)
on)
block12 sepconv2 (SeparableCon (None, 8, 8, 728) 536536
block12 sepconv2 (SeparableCon (None, 8, 8, 728) 536536
Gsepconv2_act[0][0]']
Gsepconv2_act[0][0]']
v2D)
v2D)
~
```

~

```
~
```

~

```
```

```
block13_sepconv2_bn (BatchNorm (None, 8, 8, 1024) 4096
```

```
block13_sepconv2_bn (BatchNorm (None, 8, 8, 1024) 4096
ssepconv2[0][0]']
ssepconv2[0][0]']
alization)
alization)
conv2d_3 (Conv2D) (None, 4, 4, 1024)}74547
conv2d_3 (Conv2D) (None, 4, 4, 1024)}74547
' ]
' ]
block13_pool (MaxPooling2D)
block13_pool (MaxPooling2D)
(None, 4, 4, 1024) 0
(None, 4, 4, 1024) 0
\hookrightarrowsepconv2_bn[0][0]']
\hookrightarrowsepconv2_bn[0][0]']
batch_normalization_3 (BatchNo
batch_normalization_3 (BatchNo
ヶ3[0][0]']
ヶ3[0][0]']
rmalization)
rmalization)
add_11 (Add)
add_11 (Add)
4pool[0][0]',
4pool[0][0]',
\hookrightarrownormalization_3[0][0]']
\hookrightarrownormalization_3[0][0]']
block14_sepconv1 (SeparableCon (None, 4, 4, 1536) 1582080
block14_sepconv1 (SeparableCon (None, 4, 4, 1536) 1582080
G']
G']
v2D)
v2D)
block14_sepconv1_bn (BatchNorm (None, 4, 4, 1536) 6144 ['block14_
block14_sepconv1_bn (BatchNorm (None, 4, 4, 1536) 6144 ['block14_
\hookrightarrowsepconv1[0][0]']
\hookrightarrowsepconv1[0][0]']
alization)
alization)
block14_sepconv1_act (Activati (None, 4, 4, 1536) 0
block14_sepconv1_act (Activati (None, 4, 4, 1536) 0
\hookrightarrowsepconv1_bn[0][0]']
\hookrightarrowsepconv1_bn[0][0]']
on)
on)
block14_sepconv2 (SeparableCon (None, 4, 4, 2048) 3159552
block14_sepconv2 (SeparableCon (None, 4, 4, 2048) 3159552
Gsepconv1_act[0][0]']
Gsepconv1_act[0][0]']
v2D)
v2D)
block14_sepconv2_bn (BatchNorm (None, 4, 4, 2048) 8192 ['block14_
block14_sepconv2_bn (BatchNorm (None, 4, 4, 2048) 8192 ['block14_
\hookrightarrowsepconv2[0][0]']
\hookrightarrowsepconv2[0][0]']
alization)
alization)
block14_sepconv2_act (Activati (None, 4, 4, 2048) 0
block14_sepconv2_act (Activati (None, 4, 4, 2048) 0
↔sepconv2_bn[0][0]']
↔sepconv2_bn[0][0]']
on)
```

on)

```
```

(None, 4, 4, 1024)}74547

```
```

(None, 4, 4, 1024)}74547

```
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: \(4 \times 4 \times 2048\). That is, we obtain 2048 feature maps each of size \(4 \times 4\). 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 \({ }^{254}\) function doing necessary preprocessing.

\footnotetext{
\({ }^{254} \mathrm{https}: / / \mathrm{www} . t e n s o r f l o w . o r g / a p i \_d o c s / p y t h o n / t f /\) keras/applications/xception/preprocess_input
}

\section*{Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning}

To apply this function to all data flowing through our model we use a Lambda layer \({ }^{255}\).
```

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"
\begin{tabular}{|c|c|c|}
\hline Layer (type) & Output Shape & Param \# \\
\hline lambda (Lambda) & (None, 128, 128, 3) & 0 \\
\hline xception (Functional) & (None, 4, 4, 2048) & 20861480 \\
\hline flatten (Flatten) & (None, 32768) & 0 \\
\hline dense1 (Dense) & (None, 10) & 327690 \\
\hline dense2 (Dense) & (None, 10) & 110 \\
\hline out (Dense) & (None, 2) & 22 \\
\hline
\end{tabular}

\footnotetext{
\({ }^{255} \mathrm{https}: / /\) keras.io/api/layers/core_layers/lambda/
}

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 -
\(\rightarrow\) categorical_accuracy: 0.9325 - val_loss: 0.1298 - val_categorical_accuracy: 0.
\(\hookrightarrow 9496\)
Epoch 2/5
150/150 [===============================] - 577s 4s/step - loss: 0.1285 --
↔categorical_accuracy: 0.9529 - val_loss: 0.1015 - val_categorical_accuracy: 0.
\(\leftrightarrows 9592\)
Epoch 3/5
150/150 [ \(==============================1\) - \(571 \mathrm{~s} 4 \mathrm{~s} / \mathrm{step}\) - loss: 0.1036 -ー
\(\hookrightarrow c a t e g o r i c a l \_a c c u r a c y: 0.9629\) - val_loss: 0.1071 - val_categorical_accuracy: 0. \(\rightarrow 9580\)
Epoch 4/5

\(\rightarrow c a t e g o r i c a l \_a c c u r a c y: ~ 0.9661 ~-~ v a l \_l o s s: ~ 0.1296-~ v a l \_c a t e g o r i c a l \_a c c u r a c y: ~ 0 . ~\) \(\leftrightarrows 9604\)
Epoch 5/5
150/150 [==============================1 - 556s 4s/step - loss: 0.0838-七
↔categorical_accuracy: 0.9673 - val_loss: 0.1161 - val_categorical_accuracy: 0. \(\rightarrow 9592\)
```

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')

```
```

ax.legend()
plt.show()

```



\footnotetext{
model.save('cnnmodelimproved')
}

\title{
Data Science and Artificial Intelligence for Undergraduates
}

Volume 2: Data Visualization, Supervised Learning
```

WARNING:absl:Found untraced functions such as _update_step_xla, _jit_compiled_
\hookrightarrowconvolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op,
\hookrightarrowjit_compiled_convolution_op while saving (showing 5 of 41). These functions
\hookrightarrowwill 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)

```
\(157 / 157\) [==============================1 - \(136 \mathrm{~s} 860 \mathrm{~ms} / \mathrm{step}-\mathrm{loss}: 0.1460-\) -
    ↔categorical_accuracy: 0.9516
0.9516000151634216

\section*{DECISION TREES}

Decision trees are a class of relatively simple yet powerful machine learning methods suited for both regression and classification.
- Basics (page 341)
- Regression Trees (page 343)
- Classification Trees (page 349)

Related projects:
- Forged Banknotes (page 475)
- Decision Tree (page 479) (project)

\subsection*{9.1 Basics}

Decision trees, also known as classification and regression trees (CART), are a class of machine learning techniques based on tree data structures.

\subsection*{9.1.1 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. 9.1: 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. 9.2: 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.

\subsection*{9.1.2 Training}

Training a decision tree is a relatively complex task. We start general remarks and then provide concrete algorithms in subsequent sections.

\section*{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.

\section*{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.

\subsection*{9.2 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.

\subsection*{9.2.1 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 \(\left\{\left(x_{1}, y_{1}, \ldots,\left(x_{n}, y_{n}\right)\right\}\right.\). Let
\[
I_{S}:=\left\{k \in\{1, \ldots, n\}:\left(x_{k}, y_{k}\right) \in S\right\}
\]
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
\[
v(S):=\frac{1}{|S|} \sum_{k \in I_{S}}\left(y_{k}-m(S)\right)^{2},
\]
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 \(v(S)\).

\section*{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:
```

import numpy as np
import matplotlib.pyplot as plt

# targets of samples in set }

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
\[
\frac{|L|}{|S|} v(L)+\frac{|R|}{|S|} v(R)=\frac{1}{|S|}\left(\sum_{k \in I_{L}}\left(y_{k}-m(L)\right)^{2}+\sum_{k \in I_{R}}\left(y_{k}-m(R)\right)^{2}\right) .
\]

If we take into account that the mean is the best constant approximation to the targets of a set of samples, we immediately see
\[
\sum_{k \in I_{L}}\left(y_{k}-m(L)\right)^{2} \leq \sum_{k \in I_{L}}\left(y_{k}-m(S)\right)^{2} \quad \text { and } \quad \sum_{k \in I_{R}}\left(y_{k}-m(R)\right)^{2} \leq \sum_{k \in I_{R}}\left(y_{k}-m(S)\right)^{2} .
\]

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.

\subsection*{9.2.2 Cost-Complexity Pruning}

Hint: There exist several variants of cost-complexity pruning. Here we describe the variant implemented in ScikitLearn.

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 \(v\) (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:
\[
v(\text { subtree }):=\sum_{\substack{\text { leaves of } \\ \text { subtree }}} \frac{\mid \text { leaf } \mid}{\mid \text { subtree } \mid} v(\text { leaf }),
\]
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):
\[
C C M(\text { subtree }):=v(\text { subtree })+\alpha \cdot \text { leaves in subtree. }
\]

The first summand expresses the prediction error, the second the complexity of the subtree. The regularization parameter \(\alpha\) controls the trade-off between error and complexity. CCM of a leaf (a subtree containing only one node) is
\[
C C M(\text { leaf })=v(\text { leaf })+\alpha
\]

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 \(C C M\) (subtree) \(\leq C M M\) (subtree replaced by leaf), because prediction error dominates CCM. For very large \(\alpha\) we have \(C C M\) (subtree) \(>C C M\) (subtree replaced by leaf), because complexity dominates CCM.

Cost-complexity pruning for each non-leaf node replaces the corresponding subtree by a leaf if \(C C M(\) subtree \()>\) \(C C M\) (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 \(\alpha\) the smaller the resulting tree will be. The hyperparameter \(\alpha\) 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 \(\alpha\) such that
\[
C C M(\text { subtree })=C C M(\text { subtree replaced by leaf })
\]

Such an \(\alpha\) is called effective \(\alpha\). The formula is
\[
\alpha=\frac{v(\text { subtree replaced by leaf })-v(\text { subtree })}{\text { leaves in subtree }-1}
\]

For the effective \(\alpha\) CCM does not change if we replace a subtree by a leaf. Small effective \(\alpha\) indicates that prediction error changes only slightly while complexity is changed much more when replacing the subtree. Based on that observation we compute effective \(\alpha\) for all subtrees and remove subtrees with effective \(\alpha\) below some predefined upper bound. That upper bound is identical to the hyperparameter \(\alpha\) above.

\subsection*{9.2.3 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
\[
\operatorname{loss}(\text { model(inputs) }, \text { targets })+\alpha \cdot \text { penalty }(\text { inputs }) \rightarrow \underset{\text { model }}{\min }
\]
over a certain class of models. Following this idea, in regression tree learning we could ask for a tree minimizing
\[
\operatorname{loss}(\text { model(inputs) }, \text { targets })+\alpha \cdot \text { number of leaves } \rightarrow \min _{\text {model }}
\]
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 \(\alpha\) in the objective is the lower bound for effective \(\alpha\) values to keep. In this sense cost-complexity pruning fits well into the usual regularization framework.

\subsection*{9.2.4 Regression Trees with Scikit-Learn}

Scikit-Learn implements regression trees in DecisionTreeRegressor \({ }^{256}\).
```

import numpy as np
import matplotlib.pyplot as plt
import sklearn.tree as tree
rng = np.random.default_rng(0)

```
```

def truth(x):

```
def truth(x):
    return x + np.cos(2 * np.pi * x)
    return x + np.cos(2 * np.pi * x)
xmin = 0
xmin = 0
xmax = 1
xmax = 1
x = np.linspace(xmin, xmax, 200)
x = np.linspace(xmin, xmax, 200)
n = 100 # number of data points to generate
n = 100 # number of data points to generate
noise_level = 0.3 # standard deviation of artificial noise
noise_level = 0.3 # standard deviation of artificial noise
# simulate data
# simulate data
X = (xmax - xmin) * rng.random((n, 1)) + xmin
X = (xmax - xmin) * rng.random((n, 1)) + xmin
y = truth(X).reshape(-1) + noise_level * rng.standard_normal(n)
y = truth(X).reshape(-1) + noise_level * rng.standard_normal(n)
# plot truth and data
# plot truth and data
fig, ax = plt.subplots()
fig, ax = plt.subplots()
ax.plot(x, truth(x), '-b', label='truth')
ax.plot(x, truth(x), '-b', label='truth')
ax.plot(X.reshape(-1), y, 'or', markersize=3, label='data')
ax.plot(X.reshape(-1), y, 'or', markersize=3, label='data')
ax.legend()
ax.legend()
plt.show()
```

plt.show()

```


\footnotetext{
\({ }^{256} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html
}

\section*{Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning}

DecisionTreeRegressor takes several parameters for stopping growth of the tree and also supports costcomplexity pruning. For the latter the ccp_alpha parameter has to be specified. Nodes with smaller effective \(\alpha\) 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 \({ }^{257}\) function to visualize decision trees.
```

fig, ax = plt.subplots(figsize=(12, 12))
tree.plot_tree(reg, ax=ax, filled=True, rounded=True)
plt.show()

```

\footnotetext{
\({ }^{257} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html
}


\subsection*{9.3 Classification Trees}

Classification trees work much the same like regression trees, but there are some more standard splitting rules for growing a tree.

\subsection*{9.3.1 Splitting Strategies}

All splitting rules described here define some error measure and choose the split which minimizes the error.

\section*{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.

\section*{Example}

Consider 200 samples, 100 belonging to class A, 100 belonging to class B , and two splits
\[
25 \mathrm{~A} / 75 \mathrm{~B} \text { and } 75 \mathrm{~A} / 25 \mathrm{~B}, \quad 50 \mathrm{~A} / 100 \mathrm{~B} \text { and } 50 \mathrm{~A} / 0 \mathrm{~B} .
\]

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.

\section*{Example}

Consider three samples with feature values \(x_{1}=1, x_{2}=2, x_{3}=3\) belonging to classes \(\mathrm{A}, \mathrm{B}, \mathrm{A}\), respectively. Then the original node and the two possible splits
\[
\mathrm{A} \text { and } \mathrm{BA}, \quad \mathrm{AB} \text { and } \mathrm{A}
\]
have exactly one missclassified sample. Thus, splitting does not reduce missclassification.

\section*{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.

\section*{Example}

Consider the 200 samples from above again, 100 belonging to class A, 100 belonging to class B, and two splits
\[
25 \mathrm{~A} / 75 \mathrm{~B} \text { and } 75 \mathrm{~A} / 25 \mathrm{~B} \quad 50 \mathrm{~A} / 100 \mathrm{~B} \text { and } 50 \mathrm{~A} / 0 \mathrm{~B} .
\]

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.

\section*{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
\[
\mathrm{A} \text { and } \mathrm{BA}, \quad \mathrm{AB} \text { and } \mathrm{A}
\]
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.

\section*{Splitting by entropy}

Entropy is an alternative to and very similar to Gini impurity. The formula for entropy of a leaf is
\[
-\sum_{i=1}^{C} \frac{n_{i}}{n} \log \frac{n_{i}}{n} \quad \text { with } \quad 0 \cdot \log 0:=0 .
\]

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
\[
\log _{a} p=\left(\log _{a} b\right) \log _{b} p
\]

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.

\subsection*{9.3.2 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 343) for details.

\subsection*{9.3.3 Classification Trees with Scikit-Learn}

Scikit-Learn provides the DecisionTreeClassifier \({ }^{258}\) class.

\footnotetext{
\({ }^{258}\) https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
}

\section*{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.
- The Idea (page 353)
- Stacking (page 354)
- Bagging (page 354)
- Boosting (page 355)

Related projects:
- Forged Banknotes (page 475)
- Random Forest (page 480)
- House Prices (page 483)
- A Random Forest for House Prices (page 485)

\subsection*{10.1 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.

\subsection*{10.2 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 \({ }^{259}\)
Scikit-Learn supports stacking with StackingRegressor \({ }^{260}\) from the ensemble module.

\subsection*{10.3 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.

\subsection*{10.3.1 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.

\subsection*{10.3.2 Bagging with Scikit-Learn}

Scikit-Learn supports bagging for regression tasks with BaggingRegressor \({ }^{261}\) and BaggingClassifie \({ }^{262}\) from ScikitLearn'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.

\footnotetext{
\({ }^{259} \mathrm{https}: / / b l o g . g o o g l e / t e c h n o l o g y / a i / i n t r o d u c i n g-p a t h w a y s-n e x t-g e n e r a t i o n-a i-a r c h i t e c t u r e / ~\)
\({ }^{260} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingRegressor.html
\({ }^{261} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html
\(262 \mathrm{https}: / /\) 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.

\subsection*{10.3.3 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 \({ }^{263}\) and RandomForestClassifier \({ }^{264}\).

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.

\subsection*{10.3.4 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_265 attribute of the RandomForestRegressor or RandomForestClassifier object after training the forest.

\subsection*{10.4 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.

\footnotetext{
\({ }^{263} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
\({ }^{264} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
\({ }^{265} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html?highlight=randomforest\#sklearn. ensemble.RandomForestRegressor.feature_importances_
}

\subsection*{10.4.1 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 \({ }^{266}\) 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 \(\left(x_{1}, y_{1},\right), \ldots,\left(x_{n}, y_{n}\right)\) and initialize all weights to \(\frac{1}{n}\).
Step 2 (subset selection): Calculate probabilities
\[
p_{l}:=\frac{w_{l}}{\sum_{\lambda=1}^{n} w_{\lambda}}, \quad l=1, \ldots, n
\]
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
\[
e_{l}:=\frac{\left(y_{\text {pred }, l}-y_{l}\right)^{2}}{\sum_{\lambda=1}^{n}\left(y_{\text {pred }, \lambda}-y_{\lambda}\right)^{2}} \in[0,1] .
\]

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 \(\left[0, \frac{1}{2}\right)\) 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 \(f_{\text {approx }}\) as follows:

\footnotetext{
\({ }^{266}\) 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 \(a_{1}, \ldots, a_{q}\) :
\[
f_{\text {approx }}(x):=\inf \left\{y \in \mathbb{R}: \quad \sum_{\substack{\mu=1 \\ f_{\text {approx, },(x) \leq y}}}^{q} a_{\mu} \geq \frac{1}{2} \sum_{\mu=1}^{q} a_{\mu}\right\} .
\]

Rules for weight update, stopping and aggregation are somewhat arbitrary, but yield good results in practice.
Scikit-Learn implements adaptive boosting in AdaBoostRegressor \({ }^{267}\) in the ensemble module.

\subsection*{10.4.2 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 \({ }^{268}\) 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 \(f_{\text {approx }}\) as follows:
- Given a sample \(x\) for each model \(f_{\text {approx }, \mu}\) define a vector \(a_{\mu} \in\{0,1\}^{C}\) with components
\[
a_{\mu}^{(i)}:=\left\{\begin{array}{ll}
0, & \text { if } f_{\text {approx }, \mu}(x) \neq i, \\
1, & \text { if } f_{\text {approx }, \mu}(x)=i,
\end{array} \quad i=1, \ldots, C\right.
\]

Vector \(a_{1}, \ldots, a_{q}\) can be regarded as one-hot encoded predictions of corresponding models.

\footnotetext{
\({ }^{267} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html
\({ }^{268}\) 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 \(e_{\mu}\) 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 \(a\).

\subsection*{10.4.3 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 \(\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)\) and let \(L: \mathbb{R}^{n} \rightarrow \mathbb{R}\) be the loss with respect to the true labels \(y_{1}, \ldots, y_{n}\). For mean squared error loss we have
\[
L\left(z_{1}, \ldots, z_{n}\right)=\frac{1}{n} \sum_{l=1}^{n}\left(z_{l}-y_{l}\right)^{2}
\]
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
\[
y_{\text {pred }, j}:=\left(f_{\text {approx }, j}\left(x_{1}\right), \ldots, f_{\text {approx }, j}\left(x_{n}\right)\right) .
\]

We want to improve the training loss \(L\left(y_{\text {pred, }, j}\right)\) by doing a gradient step. In other words, how to modify predictions \(y_{\text {pred }, j}\) to make \(L\) smaller? Step direction is \(-\nabla L\left(y_{\text {pred }, j}\right)\) and with step length \(s\) we obtain the updated predictions
\[
y_{\text {pred }, j}-s \nabla L\left(y_{\text {pred }, j}\right) .
\]

The problem is that in general this is not a prediction of some model of interest. Thus, we fit a new model to samples
\[
\left(x_{1}, \frac{\partial L}{\partial z_{1}}\left(y_{\text {pred }, j}\right)\right), \ldots,\left(x_{n}, \frac{\partial L}{\partial z_{n}}\left(y_{\text {pred }, j}\right)\right)
\]
and set
\[
f_{\text {approx }, j+1}:=f_{\text {approx }, j}-s f_{\text {grad }},
\]
where \(f_{\text {grad }}\) 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
\[
\nabla L\left(y_{\text {pred }, j}\right)=\frac{2}{n}\left(y_{\text {pred }, j}-y\right)
\]
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 \(f_{\text {grad }}\) and step length \(s=\frac{n}{2}\) we would have
\[
f_{\text {approx }, j+1}\left(x_{l}\right)=f_{\text {approx }, j}\left(x_{l}\right)-s f_{\text {grad }}\left(x_{l}\right)=f_{\text {approx }, j}\left(x_{l}\right)-\left(f_{\text {approx }, j}\left(x_{l}\right)-y_{l}\right)=y_{l}
\]
for \(l=1, \ldots, n\). In other words, the next model would perfectly fit the training data.

Scikit-Learn implements gradient boosting in GradientBoostingRegressor \({ }^{269}\) and GradientBoostingClassifier \({ }^{270}\) 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.

\subsection*{10.4.4 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
\[
\tilde{y}:=\left(\tilde{y}^{(1)}, \ldots, \tilde{y}^{(C)}\right), \quad \tilde{y}^{(i)}:= \begin{cases}1, & \text { if } y=i, \\ -\frac{1}{C-1}, & \text { if } y \neq i,\end{cases}
\]
and the loss function for gradient boosting is
\[
L\left(z_{1}, \ldots, z_{n}\right):=\frac{1}{n} \sum_{l=1}^{n} \mathrm{e}^{-\frac{1}{C}\left(\tilde{y}_{l}^{(1)} \tilde{z}_{l}^{(1)}+\cdots+\tilde{y}_{l}^{(C)} \tilde{z}_{l}^{(C)}\right)} .
\]

The full proof that AdaBoost-SAMME is equivalent to gradient boosting is provided by the authors of AdaBoostSAMME in the above mentioned article.

\subsection*{10.4.5 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 \({ }^{271}\) for details on the algorithm and XGBoost Python Package \({ }^{272}\) for a Python package.

\footnotetext{
\({ }^{269} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html
\({ }^{270} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
271 https://arxiv.org/abs/1603.02754
\(272 \mathrm{https}: / / \mathrm{xgboost}\). readthedocs.io/en/latest/python/index.html
}

\section*{ELEVEN}

\section*{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.
- Hard Margin SVMs (page 361)
- Soft Margin SVMs (page 366)
- Kernel SVMs (page 369)
- SVMs with Scikit-Learn (page 370)
- Support-Vector Regression (SVR) (page 374)

Related projects:
- Forged Banknotes (page 475)
- Support-Vector Machine (page 481)
- MNIST Character Recognition (page 451)
- SVM for QMNIST (page 459)

\subsection*{11.1 Hard Margin SVMs}

Given training samples \(\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)\) 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. 11.1: Typically, the margin w.r.t. a hyperplane is smaller than the distance between two sets.

\subsection*{11.1.1 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 \(\left|h_{a, b}(x)\right|\) 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. 11.2: A hyperplane can be regarded as the zero level set of a linear function.

\footnotetext{
Hint: For brevity we'll write 'the hyperplane \(h_{a, b}(x)=0\) ' instead of the more correct 'the hyperplane \(\left\{x \in \mathbb{R}^{m}\right.\) : \(\left.h_{a, b}(x)=0\right\}^{\prime}\).
}

\subsection*{11.1.2 Separating Hyperplanes}

Let \(L^{+}\)and \(L^{-}\)be the index sets of positive and negative samples, respectively. That is,
\[
L^{+}:=\left\{l \in\{1, \ldots, n\}: y_{l}=1\right\} \quad \text { and } \quad L^{-}:=\left\{l \in\{1, \ldots, n\}: y_{l}=-1\right\} .
\]

A hyperplane \(h_{a, b}(x)=0\) is called separating (with respect to the given data set) if
\[
h_{a, b}\left(x_{l}\right)>0 \quad \text { for } \quad l \in L^{+} \quad \text { and } \quad h_{a, b}\left(x_{l}\right)<0 \quad \text { for } \quad l \in L^{-} .
\]

We may rewrite this condition as
\[
y_{l} h_{a, b}\left(x_{l}\right)>0 \quad \text { for all } l \text {. }
\]

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 \(y_{l} h_{a, b}\left(x_{l}\right)<0\) 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
\[
\begin{aligned}
\operatorname{margin} & =\min _{l \in L^{+}} \frac{\left|a^{\mathrm{T}} x_{l}+b\right|}{|a|}+\min _{l \in L^{-}} \frac{\left|a^{\mathrm{T}} x_{l}+b\right|}{|a|} \\
& =\min _{l \in L^{+}} \frac{a^{\mathrm{T}} x_{l}+b}{|a|}+\min _{l \in L^{-}} \frac{-\left(a^{\mathrm{T}} x_{l}+b\right)}{|a|} \\
& =\min _{l \in L^{+}} \frac{a^{\mathrm{T}} x_{l}+b}{|a|}-\max _{l \in L^{-}} \frac{a^{\mathrm{T}} x_{l}+b}{|a|} \\
& =\min _{l \in L^{+}} \frac{a^{\mathrm{T}} x_{l}}{|a|}-\max _{l \in L^{-}} \frac{a^{\mathrm{T}} x_{l}}{|a|} \\
& =\min _{l \in L^{+}}\left(\frac{a}{|a|}\right)^{\mathrm{T}} x_{l}-\max _{l \in L^{-}}\left(\frac{a}{|a|}\right)^{\mathrm{T}} x_{l} .
\end{aligned}
\]

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
\[
d_{a}^{+}:=\min _{l \in L^{+}}\left(\frac{a}{|a|}\right)^{\mathrm{T}} x_{l} \quad \text { and } \quad d_{a}^{-}:=\max _{l \in L^{-}}\left(\frac{a}{|a|}\right)^{\mathrm{T}} x_{l} .
\]

Then
\[
\operatorname{margin}=d_{a}^{+}-d_{a}^{-}
\]


Fig. 11.3: Margin with respect to a separating hyperplane with corresponding notation.

\subsection*{11.1.3 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:
\[
\begin{aligned}
\text { distance to positive class } & =\min _{l \in L^{+}} \frac{\left|a^{\mathrm{T}} x_{l}+b\right|}{|a|}=\min _{l \in L^{+}} \frac{a^{\mathrm{T}} x_{l}-|a| \frac{d_{a}^{+}+d_{a}^{-}}{2}}{|a|}=d_{a}^{+}-\frac{d_{a}^{+}+d_{a}^{-}}{2}=\frac{d_{a}^{+}-d_{a}^{-}}{2} \\
\text { distance to negative class } & =\min _{l \in L^{-}} \frac{\left|a^{\mathrm{T}} x_{l}+b\right|}{|a|}=\min _{l \in L^{-}} \frac{-\left(a^{\mathrm{T}} x_{l}-|a| \frac{d_{a}^{+}+d_{a}^{-}}{2}\right)}{|a|}=-\max _{l \in L^{-}} \frac{a^{\mathrm{T}} x_{l}-|a| \frac{d_{a}^{+}+d_{a}^{-}}{2}}{|a|} \\
& =-\left(d_{a}^{-}-\frac{d_{a}^{+}+d_{a}^{-}}{2}\right)=-\frac{d_{a}^{-}-d_{a}^{+}}{2}=\frac{d_{a}^{+}-d_{a}^{-}}{2}
\end{aligned}
\]

\subsection*{11.1.4 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
\[
\min _{l \in L^{+}}\left(\frac{a}{|a|}\right)^{\mathrm{T}} x_{l}-\max _{l \in L^{-}}\left(\frac{a}{|a|}\right)^{\mathrm{T}} x_{l} \rightarrow \max _{a \in \mathbb{R}^{m}}
\]
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 \({ }^{273}\) and Chervonenkis \({ }^{274}\) ) 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
\[
|a|^{2} \rightarrow \min _{a \in \mathbb{R}^{m}} \quad \text { with constraints } \quad \begin{cases}a^{\mathrm{T}} x_{l}+b \geq 1, & \text { for } l \in L^{+} \\ a^{\mathrm{T}} x_{l}+b \leq-1 & \text { for } l \in L^{-}\end{cases}
\]

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
\[
a^{*}:=\frac{2}{d_{a}^{+}-d_{a}^{-}} \frac{a}{|a|} \quad \text { and } \quad b^{*}:=-\frac{d_{a}^{+}+d_{a}^{-}}{d_{a}^{+}-d_{a}^{-}}
\]

\footnotetext{
\({ }^{273} \mathrm{https}: / /\) en.wikipedia.org/wiki/Vladimir_Vapnik
\(274 \mathrm{https}: / / \mathrm{en}\).wikipedia.org/wiki/Alexey_Chervonenkis
}


Fig. 11.4: 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
\[
h_{a^{*}, b^{*}}\left(x_{l}\right) \geq \frac{2}{d_{a}^{+}-d_{a}^{-}} d_{a}^{+}-\frac{d_{a}^{+}+d_{a}^{-}}{d_{a}^{+}-d_{a}^{-}}=\frac{d_{a}^{+}-d_{a}^{-}}{d_{a}^{+}-d_{a}^{-}}=1 \quad \text { for } \quad l \in L^{+}
\]
and
\[
h_{a^{*}, b^{*}}\left(x_{l}\right) \leq \frac{2}{d_{a}^{+}-d_{a}^{-}} d_{a}^{-}-\frac{d_{a}^{+}+d_{a}^{-}}{d_{a}^{+}-d_{a}^{-}}=\frac{d_{a}^{-}-d_{a}^{+}}{d_{a}^{+}-d_{a}^{-}}=-1 \quad \text { for } \quad l \in L^{-},
\]
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 \(|a| \geq\left|a^{*}\right|\). 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
\[
\left|a^{*}\right|=\frac{2}{d_{a}^{+}-d_{a}^{-}} \leq \frac{2}{\frac{1-b}{|a|}-\frac{-1-b}{|a|}}=|a|
\]

The final step is to show that \(\left|a^{*}\right|\) is the smaller the larger the margin is. But this follows immediately from
\[
\left|a^{*}\right|=\frac{2}{d_{a}^{+}-d_{a}^{-}}
\]
because \(d_{a}^{+}-d_{a}^{-}\)is the margin.

\subsection*{11.1.5 Support Vectors}

Given the centered separating hyperplane \(h_{a, b}(x)=0\) with maximum margin each sample \(x_{l}\) satisfying \(h_{a, b}\left(x_{l}\right)=\) \(\pm 1\) 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.

support vectors

Fig. 11.5: 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.

\subsection*{11.2 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.

\subsection*{11.2.1 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:
\[
\text { measure for contraint violation }+\alpha|a|^{2} \rightarrow \min _{a, b} .
\]

The parameter \(\alpha\) controls the trade-off between satisfaction of constraints and size of the margin. Small \(\alpha\) yields well satisfied constraints (almost all samples on the correct side of the margin) but small margin (large \(|a|\) ). Large \(\alpha\) leads to a wide margin but violated constraints (incorrect predictions on training set).


Fig. 11.6: Margin width and separation quality depend on the parameter \(\alpha\).
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 \(\left|a^{\mathrm{T}} x+b\right|\) 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 \(L: \mathbb{R} \times\{-1,1\} \rightarrow \mathbb{R}\) we want to solve
\[
\frac{1}{n} \sum_{l=1}^{n} L\left(a^{\mathrm{T}} x_{l}+b, y_{l}\right)+\alpha|a|^{2} \rightarrow \min _{a, b} .
\]

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 \(y_{l}\left(a^{\mathrm{T}} x_{l}+b\right) \geq 1\). 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 \(1-y_{l}\left(a^{\mathrm{T}} x_{l}+b\right)\) 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
\[
\frac{1}{n} \sum_{l=1}^{n} \max \left\{0,1-y_{l}\left(a^{\mathrm{T}} x_{l}+b\right)\right\}+\alpha|a|^{2} \rightarrow \min _{a, b}
\]

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.

\subsection*{11.2.2 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
\[
|a|^{2} \rightarrow \min _{a \in \mathbb{R}^{m}} \quad \text { with constraints } \quad y_{l}\left(a^{\mathrm{T}} x_{l}+b\right) \geq 1
\]
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
\[
y_{l}\left(a^{\mathrm{T}} x_{l}+b\right) \geq 1-s_{l} \quad \text { and } \quad s_{l} \geq 0
\]

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
\[
\frac{1}{n} \sum_{l=1}^{n} s_{l}+\alpha|a|^{2} \rightarrow \min _{s, a, b} \quad \text { with constraints } \quad y_{l}\left(a^{\mathrm{T}} x_{l}+b\right) \geq 1-s_{l}, \quad s_{l} \geq 0
\]

For each \(a\) and \(b\) constraints can be satisfied by choosing \(s_{1}, \ldots, s_{n}\) large enough. The smallest feasible \(s_{l}\) is
\[
\max \left\{0,1-y_{l}\left(a^{\mathrm{T}} x_{l}+b\right)\right\} .
\]

Thus, solving the minimization problem with respect to \(s_{1}, \ldots, s_{n}\) (with fixed \(a\) and \(b\) ) yields the optimal value
\[
\min _{a, b} \frac{1}{n} \sum_{l=1}^{n} \max \left\{0,1-y_{l}\left(a^{\mathrm{T}} x_{l}+b\right)\right\}+\alpha|a|^{2} .
\]

This shows that the quadratic problem with slack variables is equivalent to the original non-differentiable problem.

\subsection*{11.2.3 Another Reformulation}

Applying some mathematical standard techniques for transforming optimization problems (Lagrange duality \({ }^{275}\) ) we may derive another reformulation of the soft margin SVM minimization problem:
\[
\begin{aligned}
& \qquad \sum_{l=1}^{n} c_{l}-\frac{1}{2} \sum_{l=1}^{n} \sum_{\lambda=1}^{n} y_{l} y_{\lambda}\left(x_{l}^{\mathrm{T}} x_{\lambda}\right) c_{l} c_{\lambda} \rightarrow \max _{c_{1}, \ldots, c_{n}} \\
& \text { with constraints } \quad \sum_{l=1}^{n} c_{l} y_{l}=0 \quad \text { and } \quad 0 \leq c_{l} \leq \frac{1}{2 n \alpha}, l=1, \ldots, n .
\end{aligned}
\]

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):
\[
a=\sum_{l=1}^{n} c_{l} y_{l} x_{l}, \quad b=y_{\lambda}-\sum_{l=1}^{n} c_{l} y_{l} x_{l}^{\mathrm{T}} x_{\lambda} \quad \text { for some } \lambda \text { with } 0<c_{\lambda}<\frac{1}{2 n \alpha} .
\]

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.

\subsection*{11.2.4 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. 11.7: 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.

\footnotetext{
\({ }^{275} \mathrm{https}: / /\) en.wikipedia.org/wiki/Duality_(optimization)
}

\subsection*{11.3 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.

\subsection*{11.3.1 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 \(\mathbb{R}^{\mu}\).

\section*{Example}

For \(m=3\) polynomial features of degree 2 are given by
\[
\varphi(x)=\left(\frac{1}{\sqrt{2}}, x^{(1)}, x^{(2)}, x^{(3)}, \frac{1}{\sqrt{2}}\left(x^{(1)}\right)^{2}, \frac{1}{\sqrt{2}}\left(x^{(2)}\right)^{2}, \frac{1}{\sqrt{2}}\left(x^{(3)}\right)^{2}, x^{(1)} x^{(2)}, x^{(1)} x^{(3)}, x^{(2)} x^{(3)}\right) .
\]

Why we use \(\sqrt{2}\) here will become clear below.


Fig. 11.8: Data set and margin in original and in transformed space.

\subsection*{11.3.2 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 \(\varphi\) only appears in expressions
\[
K(x, \tilde{x}):=\varphi(x)^{\mathrm{T}} \varphi(\tilde{x})
\]

Given some feature transform \(\varphi\) 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 \(\varphi(\tilde{x})\).

\section*{Example}

For \(m=3\) polynomial features of degree 2 as above yield the kernel
\[
K(x, \tilde{x})=\frac{1}{2}\left(x^{\mathrm{T}} \tilde{x}+1\right)^{2} .
\]

The \(\sqrt{2}\) in the transform \(\varphi\) 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.

\subsection*{11.3.3 More Kernels}

Next to inhomogeneous polynomial kernels
\[
\left(x^{\mathrm{T}} \tilde{x}+1\right)^{p} \quad \text { with some } p \in \mathbb{N} .
\]
and homogeneous polynomial kernels
\[
\left(x^{\mathrm{T}} \tilde{x}\right)^{p} \quad \text { with some } p \in \mathbb{N} \text {. }
\]
there are several other kernels used in practise. The most important one is the Gaussian kernel
\[
\mathrm{e}^{-\gamma|x-\tilde{x}|^{2}} \quad \text { with some } \gamma>0
\]
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
\[
\sum_{l=1}^{n} c_{l} y_{l} K\left(x_{l}, x\right)+\text { constant }
\]

This function is a weighted sum of bells centered at \(x_{1}, \ldots, x_{n}\) with weights zero for non-support vectors \(x_{l}\).

\subsection*{11.4 SVMs with Scikit-Learn}

With LinearSVC \({ }^{276}\) Scikit-Learn offers a fast and well scaling implementation of linear SVMs for classification. For kernel SVMs there is SVC \({ }^{277}\).

SVC by default uses the RBF kernel with \(\gamma\) 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.

\footnotetext{
\({ }^{276} \mathrm{https}: / /\) scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
\({ }^{277} \mathrm{https}: / /\) 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 \(\alpha\) and \(\gamma\) ．The higher \(\alpha\) the wider the margin．The lower \(\alpha\) the more accurate the classifications．Small \(\gamma\) yields smooth but possibly imprecise decision boundaries．For large \(\gamma\) decision boundaries are fit more tightly to the training data，resulting in more accurate predictions on training data．
Small \(\alpha\) and／or large \(\gamma\) 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,
4 100))
y_grid = svc.decision_function(np.stack((x0.reshape(-1), x1.reshape(-1)),七
ムaxis=1)).reshape(100, 100)
max_y = np.max(np.abs(y_grid))
cm = matplotlib.colors.LinearSegmentedColormap.from_list('ryg', ['\#ff0000',
ム\#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*['
4\#808080'])

```
# 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()
```



## Data Science and Artificial Intelligence for Undergraduates

### 11.5 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 ${ }^{278}$. 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?


### 11.5.1 SVR with Scikit-Learn

Scikit-Learn implements SVR in SVR ${ }^{279}$.

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.svm as svm
rng = np.random.default_rng(0)
```

```
def truth(x):
```

    return \(x+n p \cdot \cos (2\) * np.pi * \(x)\)
    xmin $=0$
$x \max =1$
$x=n p . l i n s p a c e(x m i n, ~ x m a x, ~ 100)$
$\mathrm{n}=100$ \# number of data points to generate
noise_level $=0.3$ \# standard deviation of artificial noise
\# simulate data
$\mathrm{X}=$ (xmax -xmin$) ~ * ~ r n g . r a n d o m((n, 1))+x m i n$
$y=$ truth (X).reshape(-1) + noise_level * rng.standard_normal(n)
\# plot truth and data
fig, $a x=p l t . s u b p l o t s()$
ax.plot(x, truth(x), '-b', label='truth')
ax.plot (X.reshape(-1), y, 'or', markersize=3, label='data')
ax.legend()
plt.show()

[^87]

When creating the SVR object we may provide the width of the $\varepsilon$-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.

## 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 ${ }^{280}$ for conditional probabilities and on the (naive) assumption that features are mutually independent ${ }^{281}$ if interpreted as random variables.

Given training samples $\left(x_{1}, y_{1}\right), \ldots,\left(x_{n}, y_{n}\right)$ 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 $x_{0}$.

### 12.1 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

$$
P\left(Y=i \mid X=x_{0}\right) \rightarrow \max _{i \in\{1, \ldots, C\}}
$$

That is, we choose $y_{0}$ to be the maximizer of $P\left(Y=i \mid X=x_{0}\right)$ with respect to $i$.
The probabilities $P\left(Y=i \mid X=x_{0}\right)$ are not accessible. But Bayes' theorem states

$$
P\left(Y=i \mid X=x_{0}\right)=\frac{p_{X \mid Y=i}\left(x_{0}\right) P(Y=i)}{p_{X}\left(x_{0}\right)}
$$

for all $i \in\{1, \ldots, C\}$. Here, $p_{X \mid Y=i}$ is a density of the conditional probability measure $P(\cdot \mid Y=i)$ 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 $p_{X \mid Y=i}\left(x_{0}\right)$ and $p_{X}\left(x_{0}\right)$ can be replaced by the probabilities $P\left(X=x_{0} \mid Y=i\right)$ and $P\left(X=x_{0}\right)$, respectively. If $X$ contains continuous components, then we have to work with densities.
The denominator $p_{X}\left(x_{0}\right)$ in Bayes' theorem does not depend on $i$ and, thus, does not influence maximization. Knowing $p_{X \mid Y=i}\left(x_{0}\right)$ and $P(Y=i)$ for all $i$ we may compute

$$
p_{X}\left(x_{0}\right)=\sum_{i=1}^{C} p_{X \mid Y=i}\left(x_{0}\right) P(Y=i)
$$

because the sum of all $P\left(Y=i \mid X=x_{0}\right)$ has to be 1 . The conditional density $p_{X \mid Y=i}\left(x_{0}\right)$ and the probabily $P(Y=i)$ can be estimated from the training samples.

[^88]
### 12.2 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

$$
P(Y=i)=\frac{\left|\left\{l \in\{1, \ldots, n\}: y_{l}=i\right\}\right|}{n}
$$

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.

### 12.3 Estimating Classes' Feature Probabilities

To estimate the densities $p_{X \mid Y=i}\left(x_{0}\right)$ 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

$$
p_{X \mid Y=i}\left(x_{0}\right)=p_{X^{(1)} \mid Y=i}\left(x_{0}^{(1)}\right) \cdots p_{X^{(m) \mid} \mid Y=i}\left(x_{0}^{(m)}\right) .
$$

So we only have to estimate one-dimensional densities. To simplify notation we write $p_{U \mid Y=i}\left(u_{0}\right)$ as placeholder for one of the $m$ densities.

### 12.3.1 Gaussian Probabilities

If $U$ is a continuous random variable we may assume that $p_{U \mid Y=i}$ is a Gaussian density with mean $\mu_{i}$ and standard deviation $\sigma_{i}$ :

$$
p_{U \mid Y=i}\left(u_{0}\right)=\frac{1}{\sigma_{i} \sqrt{2 \pi}} \exp \left(-\frac{1}{2}\left(\frac{u_{0}-\mu_{i}}{\sigma_{i}}\right)^{2}\right)
$$

Parameters $\mu_{i}$ and $\sigma_{i}$ can be estimated from the training samples in class $i$ with standard techniques (see statistics lecture).

### 12.3.2 Bernoulli Probabilities

If $U$ takes values in $\{0,1\}$ then $p_{U \mid Y=i}\left(u_{0}\right)$ in Bayes' theorem has to be replaced by $P\left(U=u_{0} \mid Y=i\right)$ with

$$
P(U=0 \mid Y=i)=1-p_{i} \quad \text { and } \quad P(U=1 \mid Y=i)=p_{i} .
$$

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.

### 12.3.3 Multinomial Probabilities

If $X^{(1)}, \ldots, X^{(m)}$ count the occurrences of $m$ possible events for several independent trials of an experiment, then the corresponding random vector follows a multinomial distribution ${ }^{282}$ with parameters $p_{1}, \ldots, p_{m}$ (probabilities of events) and $\nu$ (number of trials):

$$
P\left(X=x_{0}\right)=\frac{\nu!}{x_{0}^{(1)}!\cdots x_{0}^{(m)}!} p_{1}^{x_{0}^{(1)}} \cdots p_{m}^{x_{0}^{(m)}},
$$

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 $p_{i, 1}, \ldots, p_{i, m}$ for each class $i$. Then

$$
P\left(X=x_{0} \mid Y=i\right)=\frac{\nu!}{x_{0}^{(1)}!\cdots x_{0}^{(m)}!} p_{i, 1}^{x_{0}^{(1)} \cdots p_{i, m}^{x_{0}^{(m)}}, \text {. }}
$$

with $\nu=x_{0}^{(1)}+\cdots+x_{0}^{(m)}$. The factor

$$
c\left(x_{0}\right):=\frac{\left(x_{0}^{(1)}+\cdots+x_{0}^{(m)}\right)!}{x_{0}^{(1)}!\cdots x_{0}^{(m)}!}
$$

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

$$
P\left(X=x_{0} \mid Y=i\right)=\prod_{k=1}^{m} c\left(x_{0}\right)^{\frac{1}{m}} p_{i, k}^{x_{0}^{(k)}}
$$

of the naive Bayes approach, although components of $X$ are not independent. Moreover, we do not have to compute $c\left(x_{0}\right)$ explicitely. It's a scaling factor which can be computed subsequently from the fact that all probabilities have to sum to 1 (cf. $p_{X}\left(x_{0}\right)$ 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 $P\left(X=x_{0} \mid Y=i\right)$ 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.

[^89]
### 12.3.4 Kernel Density Estimates

If the underlying probability model is not know in advance, the densities $p_{X^{(k)} \mid Y=i}$ can be estimated from training data using kernel density estimation.

## Self-study task

Read about kernel density estimation at Wikipedia ${ }^{283}$ (introduction, definition, example).

### 12.4 Naive Bayes Classification with Scikit-Learn

Different variants of naive Bayes classification are implemented in Scikit-Learns's naive_bayes module ${ }^{284}$.

### 12.5 Related Projects

- Forged Banknotes (page 475)
- Naive Bayes Classification (page 481) (project)

[^90]
## CHAPTER

## 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.

- Preprocessing Text Data (page 381)
- Training (page 401)

Related projects:

- Blogs (page 489)
- Blog Author Classification (Training) (page 489)
- Blog Author Classification (Test) (page 492)


### 13.1 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 ${ }^{285}$, 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 ${ }^{286}$ (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/'
```


### 13.1.1 Getting and Restructuring the Data Set

Data comes as ZIP file from the above mentioned website ( 313 MB ). The ZIP file contains one XML ${ }^{287}$ 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.

[^91]
## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning

## 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/
    \(\hookrightarrow 1005076\).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('.
\hookrightarrow') [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},ь
    4index=blog_ids)
```

    skipping blogs/
    blogs

| 1000866 | f | 17 | Student |
| :--- | ---: | ---: | ---: |
| 1004904 | m | 23 | Arts |
| 1005076 | f | 25 | Arts |
| 1005545 | m | 25 | Engineering |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| 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 <br> unknown |
| :--- | ---: | ---: | ---: |
| 1000331 | f | 37 | Student |
| 1000866 | f | 17 | Arts |
| 1004904 | m | 23 | Arts |
| 1005076 | f | 25 | Engineering |
| 1005545 | m | 25 | $\ldots$ |
| $\ldots$ | $\ldots$ | $\ldots$ | Telecommunications |
| 996147 | f | 36 | unknown |
| 997488 | m | 25 | unknown |
| 998237 | f | 16 | unknown |
| 998966 | m | 27 | Internet |
| 999503 | m | 25 |  |
|  |  |  |  |

```
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. et ree. Element Tree ${ }^{288}$ 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:

[^92]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning- Files are littered with random white space characters.
- Enconding is unknown and may vary from file to file.
- Files contain non-printable characters ( $00 \mathrm{~h}-1 \mathrm{Fh}$ ) 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 nonUnicode 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 nonprintable 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 ${ }^{289}$.

```
# cell execution may take several hours due to language detection
month_num = {'january': 1, 'february': 2, 'march': 3, 'april': 4, 'may': 5, 'june
    4': 6,
    'july': 7, 'august': 8, 'september': 9, 'october': 10, 'november':\smile
    411, '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
    \hookrightarrow': 6,
        'julio': 7, 'agosto': 8, 'septiembre': 9, 'octubre': 10, 'noviembre
    \hookrightarrow': 11, 'diciembre': 12,
        # Portuguese
        'janeiro': 1, 'fevereiro': 2, 'maio': 5, 'junho': 6,
```

(continues on next page)

[^93](continued from previous page)

```
    'julho': 7, 'agosto': 8, 'setembro': 9, 'outubro': 10, 'novembro':
411, 'dezembro': 12,
    # German
    'januar': 1, 'februar': 2, 'märz': 3, 'april': 4, 'mai': 5, 'juni':๖
\hookrightarrow,
    'juli': 7, 'august': 8, 'september': 9, 'oktober': 10, 'november':`
411, '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,
\hookrightarrow'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'
```

```
    if len(month) == 0:
    month = 'unknown'
    if len(year) == 0:
        year = '0'
    text = xml_post[(xml_post.find('<post>') + 6):(xml_post.find('</post>
    4'))]
    text = re.sub(r'[\x00-\x1F]+', ' ', text) # non-printable\smile
\hookrightarrowcharacters
    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,
    \hookrightarrow'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 |  |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |  |
| 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 |  |

$0 \quad$ 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
[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 |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| 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 |


[653702 rows x 5 columns]

## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning

### 13.1.2 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
    690 690
\begin{tabular}{lrr}
13 & 690 & 690 \\
14 & 1246 & 1246
\end{tabular}
\(15 \quad 1771 \quad 1771\)
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)
```



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_
    \varsigmaobserved=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 tor
    sadopt 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 |


| $\begin{array}{rll} (30,100] & 1497 \\ & \mathrm{f} & 1497 \end{array}$ | $\begin{aligned} & 1497 \\ & 1497 \end{aligned}$ |
| :---: | :---: |
| blogs.groupby('industry').count () [' |  |
| 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'>
```


## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning

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 31 st.

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 ofv
    \hookrightarrowobserved=False is deprecated and will be changed to True in a future version\smile
    ๑f pandas. Pass observed=False to retain current behavior or observed=True to\smile
    sadopt 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
    (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↔
    \rightarrow a d o p t ~ t h e ~ f u t u r e ~ d e f a u l t ~ a n d ~ s i l e n c e ~ t h i s ~ w a r n i n g . ~
        blogs.groupby(['gender', 'age_group'])['chars'].sum()
gender age_group
f (0, 20] 117123221
    (20, 30] 179750611
    (30, 100] 74878617
    (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.

### 13.1.3 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 ${ }^{290}$ 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 nonzero 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.

[^94]
## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

Stop words, stemming, and lemmatization are, for instance, implemented in the nltk Python package (Natural Language Toolkit $)^{291}$. 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, aposthrophe). Regular expressions ${ }^{292}$ 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']
```

```
O 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...
6 7 6 0 2 5 ~ S c i e n c e ~ i s ~ l i k e ~ s e x ~ o c c a s i o n a l l y ~ s o m e t h i n g ~ p r a . . . ~
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 WordNet Lemmat izer ${ }^{293}$ of NLTK. WordNet ${ }^{294}$ 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 ${ }^{295}$, too. Searching WordNet database with NLTK or online includes some stemming-like preprocessing steps ${ }^{296}$.

```
import nltk
```

Before first use of WordNetLemmatizer we have to download the database.

[^95]```
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 WordNet Lemmat izer by default looks for nouns. A second argument to lemmat ize 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)
```


## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning```
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
4'])
    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
            for idx in posts.index:
        ----> 5 posts.loc[idx, 'tokenized'] = nltk.tokenize.word_tokenize(posts.
        \hookrightarrowloc[idx, 'text'])
            6 posts.loc[idx, 'tagged'] = nltk.tag.pos_tag(posts.loc[idx,
        \hookrightarrow'tokenized'])
    File /opt/conda/envs/python3/lib/python3.11/site-packages/pandas/core/indexing.
        \hookrightarrowpy:849, in _LocationIndexer.__setitem__(self, key, value)
            84 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
```

```
    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,七
    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 )
    1 8 9 6 ~ e l i f ~ l p l a n e \_ i n d e x e r ~ = = ~ 0 ~ a n d ~ l e n ( v a l u e ) ~ = = ~ l e n ( s e l f . o b j . i n d e x ) : ~
    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
```

(continued from previous page)

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, |
| :---: | :---: |
| 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. |

```
# 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']
```

$0 \quad[($ Well, RB), (, , ), (everyone, NN), (got, VBD...
1 [(My, PRP\$), (fouryear, JJ), (old, JJ), (never...

```
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 ${ }^{297}$. There we see the following relations:

| NLTK POS tag | WordNet POS tag |
| :--- | :--- |
| JJ... | ADJ |
| RB $\ldots$ | ADV |
| NN... | NOUN |
| VB $\ldots$ | VERB |

All NLTK POS tags have at least two characters. So we may use the following conversion function.

```
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 lemmat ize 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 ${ }^{298}$ 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 unsual (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.

```
nltk.corpus.wordnet._morphy('children', nltk.corpus.reader.wordnet.VERB)
```


## []

[^96]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_
๑tag)
                if len(morphy_result) > 0:
                    lemmas.append(morphy_result[0])
                else:
                            morphy_result_all = nltk.corpus.wordnet._morphy(modified_token,七
\hookrightarrownltk.corpus.reader.wordnet.NOUN)
                                    + nltk.corpus.wordnet._morphy(modified_token,
↔nltk.corpus.reader.wordnet.VERB) \
                                    + nltk.corpus.wordnet._morphy(modified_token,
\hookrightarrowltk.corpus.reader.wordnet.ADJ) \
                            + nltk.corpus.wordnet._morphy(modified_token,
九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, -
    \(\varsigma^{\prime} ' t, ~ s k y d o m e, ~ o c c a s s i o n a l l y, ~ g o l d e y e s, ~ n ' t, ~ ' m, ~, ~ n ' t, ~ e v e r y o n e, ~ g a m e b o y, ~ n ' t, ~\)
    \(\hookrightarrow\) n't, n't, everything, 've, hmm, something, else, mcnally, something, anyone,
    \(\hookrightarrow ' v e, ~ ' v e, ~ b r e a d m a k e r, ~ n ' t, ~ n ' t, ~ ' v e, ~ i m d b, ~ e v e r y t h i n g, ~ s o m e t h i n g, ~ s o m e t h i n g, ~\)
    \(\hookrightarrow ' m, ~ i n e v e i t a b l e, ~ n ' t, ~, ~ n ' t, ~ ' m, ~ ' v e, ~ k e l, ~ ' m, ~ y a, ~ d i, ~ a m o r e, ~ ' m, ~ . ., ~ n ' t, ~ n ~\)
    
$\leftrightarrow ' t, ~ e p v m, ~ n ' t, ~ p o o p e y h e a d, ~ n a t h a n, ~ ' r e, ~ n ' t, ~ n a t h a n, ~ n a t h a n, ~ a l s t a d t, ~ w e, ~ w e, ~ w ~$
$\rightarrow$ dan, dave, we, dan, , , .., n't, 'm, thnk, 'm, epvm, umm, heh, n't, arg, , 'm,
$\hookrightarrow ' m, ~ m a d s e n, ~ e e k, ~ u m m, ~ . ., ~ b r e t t, ~ a l e x, ~ j o a n n e, ~ n o w . t h e y, ~ ' r e, ~ a l e x, ~, ~ g o s h, ~ u ~$
$\rightarrow d a r n i t, ~ i m p r o v, ~ ' m, ~ ' m, ~ ' r e, ~ n ' t, ~ . . . ., ~ y a, ~ d i, ~ a m o r e, ~ ' m, ~ ' m, ~ n ' t, ~ y a, ~ d i, ~ ᄂ ~$
$\rightarrow$ amore, , something, grrr, , 'm, enjoyig, hey, catie, momly, n't, duper, n't,
$\rightarrow ' m, ~ a n y o n e, ~ ' m, ~ t h i n g a m a b o b b e r, ~ ' r e, ~ ' r e, ~ ' r e, ~ ' v e, ~ d i, ~ a m o r e, ~ n ' t, ~ ' m, ~ ' m, ~ u ~$
$\hookrightarrow, ~ e v e r y o n e, ~ b r a n d o n, ~ j a k e, ~ n ' t, ~ m e h, ~, ~ o k ., ~ m y, ~ n ' t, ~ a ., ~ ' m, ~ k a n e, ~ u m m m, ~ ᄂ ~$
$\leftrightarrow$ something, else, psh, n't, n't, heh, anything, 'm, ya, di, amore, 'm, $\quad$
$\leftrightarrows$ superduper, ...., 're, contraversial, blahdeblahdeblah, nequa, naperville,
$\leftrightarrows$ 're, 're, 're, n't, heck, kathryn, how, poopy, 'm, sleepyland, g(condindeglot next page)
everyone, ya, di, amore, 'm, superduper, ...., , 're, contraversial,
13.1. Preprocesșing Jext Data, goodnight, everyone, ya, di, amore, hey, alex, 'm,
↔everything, 've, heh, spontanious, catie, rar, ah, n't, something, everything,

## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning(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...
6 7 6 0 2 5 \text { science be sex occasionally practical come not...}
6 7 6 0 2 6 ~ d o g ~ t o y ~ m a r i t a l ~ a i d ~ m a n a g e ~ l i t t l e ~ q u i z ~ s e e ~ w e l . . .
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... |
| $\ldots$ | $\ldots$. |  |
| 676023 | 999503 | today celebrate independence day honor event $9 .$. |
| 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,v
    4rows=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 $\leftrightarrows b e d$ book cat have be lot rain people have wet basement be lake be golf course $\quad$, $\hookrightarrow f i e l d s$ be green green green be suppose be degree friday be deal mosquito next $\hookrightarrow$ 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...

### 13.2 Training

In this second chapter on text classification we train several machine learning models on the preprocessed data from the first chapter.

### 13.2.1 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 |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| 996147 | f | 36 | Telecommunications |
| 997488 | m | 25 | unknown |
| 998237 | f | 16 | unknown |
| 998966 | m | 27 | unknown |
| 999503 | m | 25 | Internet |
|  |  |  |  |

```
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 |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| 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 |

Well, everyone got up and going this morning...
My four-year old never stops talking. She'll ...
Actually it's not raining yet, but I bought $15 \ldots$.
Ha! Just set up my RSS feed - that is so easy!...

```
4 Oh, which just reminded me, we were talking ab...
676023 Today we cermen
6 7 6 0 2 3 \text { Today we celebrate our independence day. In...}
676024 Ugh, I think I have allergies... My nose has ...
6 7 6 0 2 5 ~ " S c i e n c e ~ i s ~ l i k e ~ s e x ; ~ o c c a s i o n a l l y ~ s o m e t h i n g ~ p . . .
6 7 6 0 2 6 ~ D o g ~ t o y ~ o r ~ m a r i t a l ~ a i d ~ I ~ m a n a g e d ~ 1 0 / 1 4 ~ o n ~ t h . . . ~
676027 I had a dream last night about a fight when I ...
    lemmatized
    well get up go morning still raining okay sort...
    old never stop talk say mom say say oh yeah do...
    actually not raining yet buy ticket game mom b...
    ha just set r feed be so easy do do enough tod...
    just remind be talk can food coffee break morn...
.
6 7 6 0 2 3 ~ t o d a y ~ c e l e b r a t e ~ i n d e p e n d e n c e ~ d a y ~ h o n o r ~ e v e n t ~ g . . .
676024 think have allergy nose have be stuff week mak...
6 7 6 0 2 5 \text { science be sex occasionally practical come not...}
6 7 6 0 2 6 ~ d o g ~ t o y ~ m a r i t a l ~ a i d ~ m a n a g e ~ l i t t l e ~ q u i z ~ s e e ~ w e l . . . ~
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 |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| 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 |  | www.teenopendiary.com | J.A.M.C. | NaN |
| 1166 |  |  | ----> 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 | Ugh. |  | YAYness! | NaN |
| 674723 |  | http://www.cnn.com/2003/US/Southwest/... whoa im tlaking to kevin daly whoa |  | NaN |
| 675851 |  |  |  | NaN |

```
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 |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$. |
| 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 |



### 13.2.2 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.

## Data Science and Artificial Intelligence for Undergraduates

## 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
sstay home bed book cat have be lot rain people have wet basement be lake be
$\leftrightarrows$ golf course fields be green green green be suppose be degree friday be dealu
$\hookrightarrow$ mosquito next week hear winnipeg describe old testament city cbc radio one৬
$\rightarrow l a s t$ week sort rings true flood infestation',
'old never stop talk say mom say say oh yeah do lady bug hide rain hearu
$\hookrightarrow$ own voice very very exhaust now remember be go work sigh',
'actually not raining yet buy ticket game mom birthday tonight be
$\hookrightarrow$ suppose rain do cancel baseball game rain ballpark be beautiful ai use go jay
$\leftrightarrows$ game live toronto really take kid game now do know blue jay ticket cost now
$\rightarrow$ 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 hitu
$\leftrightarrows$ 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 |
| $\ldots$ | $\ldots$ | $\cdots$ | $\cdots$ | $\ldots$ |
| 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 |


|  | Well 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
2
3

4
...

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',
    \hookrightarrow'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, )

## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning
## 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_
    4size=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 ${ }^{299}$. 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 $t$ rans form 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.

[^97]The number of samples containing a fixed word is denoted as document frequency $(D F)$ of the word. The word count within one sample is denoted as term frequency ( $T F$ ) 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 |
| $\ldots$ | $\ldots$ | $\ldots$ | $\ldots$ |
| 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 $\mathrm{X} \_$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']
```

```
3 9 2 3 4 ~ d a s h i k i
39075 peerage
39076 pictograph
39079 diamante
39288 3d
5 0 9 4 8 ~ t a i p e h ~
50949 razorbill
5 0 9 5 0 ~ p o r i f e r a
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) -
    \iotamore (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.
    42) work (0.21) today (0.21) here (0.21) only (0.21) look (0.21) even (0.20) -
    \hookrightarrowtoo (0.19) other (0.19) last (0.19) way (0.19) still (0.18) right (0.18) need\smile
    (0.17) new (0.17) love (0.17) tell (0.17) feel (0.17) things (0.17) try (0.
    417) first (0.17) very (0.16) life (0.16) start (0.16) give (0.16) thing (0.
    416) 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.
    413) most (0.13) long (0.13) week (0.13) let (0.13) mean (0.13) like (0.12) ఒ
    4keep (0.12) as (0.12) read (0.12) end (0.12) better (0.12) find (0.12) greatr
    (0.12) guy (0.11) ever (0.11) always (0.11) next (0.11) few (0.11) write (0.
    41) watch (0.11) post (0.11) hope (0.11) seem (0.11) actually (0.11) play (0.
    ๑11) put (0.11) sure (0.11) up (0.11) ask (0.11) bad (0.11) many (0.11) maybeь
    (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 ${ }^{300}$.

```
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 ${ }^{301}$. Usage is identical to CountVectorizer.

[^98]```
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)
    
### 13.2.3 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)
\hookrightarrow'.format(metrics.accuracy_score(y, y_pred)))
    print('correct classification rate (gender): {:.4f} (random guessing: 0.5000)
G'.format (metrics.accuracy_score(y // 3, y_pred // 3)))
    print('correct classification rate (age): {:.4f} (random guessing: 0.3333)
\hookrightarrow'.format(metrics.accuracy_score(y - 3 * (y // 3), y_pred - 3 * (y_pred // 3))))
```

                                    (continues on next page)
    ```
    # calculate confusion matrices
    cm = pd.crosstab(pd.Series(y, name='truth'), pd.Series(y_pred, name=
\hookrightarrow'prediction'))
    cm = cm.reindex(index = [0, 1, 2, 3, 4, 5], columns=[0, 1, 2, 3, 4, 5], fill_
\iotavalue=0)
    cm_gender = pd.crosstab(pd.Series(y // 3, name='truth'), pd.Series(y_pred //v
43, 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_
4pred - 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[
Glabel'].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)
```

    [ \(88173123839 \quad 45420 \quad 92169122165\) 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 $k$ th word in our vocabulary) in class $i$. The MultinomialNB objects provides the logarithm of these probabilities as a 2 d 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, ...,
```

(continued from previous page)

$$
\begin{aligned}
& \text { 5.40897601e-06, 1.52174765e-06, 1.86291326e-06], } \\
& \text { [3.16922035e-06, 8.23294289e-06, 1.46964824e-05, ..., } \\
& 5.55744801 \mathrm{e}-06,3.46527245 \mathrm{e}-06,2.94293686 \mathrm{e}-06] \text {, } \\
& \text { [4.33136042e-06, 1.01679660e-05, 2.14931631e-05, ..., } \\
& 4.23157086 e-06,1.69074347 e-06,2.65137775 e-06] \text {, } \\
& \text { [2.44322058e-06, 7.15660286e-06, 1.61798918e-05, ..., } \\
& 1.32512575 e-05,1.94623715 e-06,2.01389749 e-06] \text {, } \\
& {[3.11215272 \mathrm{e}-06,7.82176886 \mathrm{e}-06,1.72747277 \mathrm{e}-05, \ldots,} \\
& 7.67104041 e-06,5.87706081 e-06,3.72104924 e-06]])
\end{aligned}
$$

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 0.002445372958287036
    time 0.00244219262808092
    work 0.0023443172188228716
    now 0.0022574399274471963
    see 0.002239260855123209
    day 0.002179888002166481
    more 0.0021602842162711912
    take 0.002147657978906127
    want 0.00208513406619077
```

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 14.51950422377979854
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

## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

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-
4bin_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()
```



### 13.2.4 Support Vector Machine

Scikit-Learn provides different implementations of support vector machines for classification:

- $S V C^{302}$ is the most general one. It supports linear and kernel SVMs, but is relatively slow.
- LinearSVC ${ }^{303}$ is a more efficient implementation supporting only linear SVMs.
- SGDClassifier ${ }^{304}$ uses gradient descent to minimize some loss function, the hinge loss for instance. It supports penalty terms for regularization and, thus, SVMs.

[^99]
## 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_
    4jobs=-1)
gs.fit(X_train, y_train)
print(gs.best_params_)
```

/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_ $\leftrightarrows$ stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration $\leftrightarrows r e a c h e d ~ b e f o r e ~ c o n v e r g e n c e . ~ C o n s i d e r ~ i n c r e a s i n g ~ m a x \_i t e r ~ t o ~ i m p r o v e ~ t h e ~ f i t . ~$ warnings.warn(
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_
↔stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration
$\hookrightarrow r e a c h e d ~ b e f o r e ~ c o n v e r g e n c e . ~ C o n s i d e r ~ i n c r e a s i n g ~ m a x \_i t e r ~ t o ~ i m p r o v e ~ t h e ~ f i t . ~$ warnings.warn (
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_ sstochastic_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/_ $\hookrightarrow$ stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration sreached 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 sreached before convergence. Consider increasing max_iter to improve the fit. warnings.warn (
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_ $\leftrightarrows$ stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration $\smile$ ↔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/_ $\hookrightarrow$ stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration sreached 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 $\leftrightarrows$ reached before convergence. Consider increasing max_iter to improve the fit.
(continues on next 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/_ sstochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration $\leftrightarrow r e a c h e d ~ b e f o r e ~ c o n v e r g e n c e . ~ C o n s i d e r ~ i n c r e a s i n g ~ m a x \_i t e r ~ t o ~ i m p r o v e ~ t h e ~ f i t . ~$ warnings.warn (
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_ sstochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration $\leftrightarrow r e a c h e d ~ b e f o r e ~ c o n v e r g e n c e . ~ C o n s i d e r ~ i n c r e a s i n g ~ m a x \_i t e r ~ t o ~ i m p r o v e ~ t h e ~ f i t . ~$ warnings.warn (
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_ ↔stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration $\leftrightarrows$ 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/_ $\leftrightarrows$ 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 $\hookrightarrow r e a c h e d ~ b e f o r e ~ c o n v e r g e n c e . ~ C o n s i d e r ~ i n c r e a s i n g ~ m a x \_i t e r ~ t o ~ i m p r o v e ~ t h e ~ f i t . ~$ warnings.warn (
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_ sstochastic_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/_ sstochastic_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/_
sstochastic_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/_ $\leftrightarrows$ stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration sreached before convergence. Consider increasing max_iter to improve the fit. warnings.warn (
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_ sstochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration $\hookrightarrow$ 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/_
sstochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration $\hookrightarrow r e a c h e d ~ b e f o r e ~ c o n v e r g e n c e . ~ C o n s i d e r ~ i n c r e a s i n g ~ m a x \_i t e r ~ t o ~ i m p r o v e ~ t h e ~ f i t . ~$ 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 $\hookrightarrow$ 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/_ $\leftrightarrows$ 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 $\hookrightarrow r e a c h e d ~ b e f o r e ~ c o n v e r g e n c e . ~ C o n s i d e r ~ i n c r e a s i n g ~ m a x \_i t e r ~ t o ~ i m p r o v e ~ t h e ~ f i t . ~$ warnings.warn (
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_ ↔stochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration $\hookrightarrow r e a c h e d ~ b e f o r e ~ c o n v e r g e n c e . ~ C o n s i d e r ~ i n c r e a s i n g ~ m a x \_i t e r ~ t o ~ i m p r o v e ~ t h e ~ f i t . ~$ warnings.warn (
/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_ $\leftrightarrows$ 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 $\hookrightarrow r e a c h e d ~ b e f o r e ~ c o n v e r g e n c e . ~ C o n s i d e r ~ i n c r e a s i n g ~ m a x \_i t e r ~ t o ~ i m p r o v e ~ t h e ~ f i t . ~$ warnings.warn(

```
{'alpha': 1.00000000000000004e-06}
```

/opt/conda/envs/python3/lib/python3.11/site-packages/sklearn/linear_model/_ sstochastic_gradient.py:713: ConvergenceWarning: Maximum number of iteration sreached 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
(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

```
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.
```

(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
(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

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
```

(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
(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:\longleftarrow
    \rightarrow C o n v e r g e n c e W a r n i n g : ~ S o l v e r ~ t e r m i n a t e d ~ e a r l y ~ ( m a x \_ i t e r = 1 0 0 ) . ~ C o n s i d e r ~ p r e - ~
    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) |




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 ${ }^{305}$, 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 $\alpha$ and $\gamma$ below were obtained via hyperparameter optimization (took several days).

[^100]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning```
# 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 concurrente $\leftrightarrows$ 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.
-- 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

```
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 $2^{37000}=\left(2^{37}\right)^{1000}>\left(10^{11}\right)^{1000}=10^{11000}$ samples. It's very likely that our classes can be seprated by hyperplanes without need for nonlinear separation.

### 13.2.5 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 ${ }^{306}$ 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)
```

${ }^{306} \mathrm{https}: / /$ 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) |  |





## Random Forest

Applying bagging to decision trees is known as random forests. Scikit-Learn's RandomForestClassifier ${ }^{307}$ 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)
```

[^101]



| 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 ScikitLearn's AdaBoostClassifier ${ }^{308}$.

```
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) |

[^102]

| 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 III

## Exercises

## DATA VISUALIZATION

- Matplotlib Basics (page 437)
- Advanced Matplotlib (page 439)


### 14.1 Matplotlib Basics

Before solving these exercises you should have read Matplotlib Basics (page 5).

```
import numpy as np
import matplotlib.pyplot as plt
```


### 14.1.1 Simple Function Plot

Plot the sine function for angles from 0 to $2 \pi$. Label the axes and show a title. Use a red line without markers.

## Solution:

```
# your solution
```


### 14.1.2 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
```


### 14.1.3 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
```


## Data Science and Artificial Intelligence for Undergraduates

### 14.1.4 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
```


### 14.1.5 Ticks and Grids

Plot $f(x)=\sin (x)$ for $x \in[-1,7]$. Place major tick labels at multiples of $\pi$ 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
```


### 14.1.6 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
```


### 14.1.7 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
```


### 14.1.8 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 ${ }^{309}$ ).

## Solution:

```
# your solution
```

[^103]
### 14.1.9 Pixelbased Postprocessing

Plot $f(x)=\sin x$ for $x \in[-2 \pi, 2 \pi]$. Fade out line ends with GIMP.

## Solution:

```
* your solution
```


### 14.1.10 Vectorbased Postprocessing

Plot $f(x)=\sin x$ for $x \in[-1,7]$ and plot the sine approximations $g(x)=-\frac{4}{\pi^{2}}\left(x-\frac{\pi}{2}\right)^{2}+1$ and $h(x)=$ $-\frac{4}{\pi^{2}}\left(x-\frac{3 \pi}{2}\right)^{2}-1$. Plot the region $x \in\left[\pi-\frac{1}{2}, \pi+\frac{1}{2}\right]$ in a separate figure. Save both figures as vector graphics and add the detail view to the sine plot with Inkscape.

## Solution:

```
# your solution
```


### 14.2 Advanced Matplotlib

Before solving these exercises you should have read Matplotlib (page 5).

### 14.2.1 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
```


## Part IV

## Projects

## CHAPTER

## FIFTEEN

## WEATHER

We will work through several projects related to weather data and forecasting. Data will be obtained from Deutscher Wetterdienst ${ }^{310}$.

- DWD Open Data Portal (page 443)
- Getting Forecasts (page 445)
- Climate Change (page 447)
- Weather Animation (page 448)


### 15.1 DWD Open Data Portal

Deutscher Wetterdienst (DWD) ${ }^{311}$ 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 ${ }^{312}$ 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.

### 15.1.1 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
```

[^104]
## Data Science and Artificial Intelligence for Undergraduates

### 15.1.2 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
```


### 15.1.3 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 ${ }^{313}$. What's the most recent measurement (timestamp and temperatur)? What's the oldest measurement available?

## Solution:

```
# your answer
```


### 15.1.4 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!

[^105]
### 15.2 Getting Forecasts

The Open Data Portal ${ }^{314}$ of Deutscher Wetterdienst (DWD) ${ }^{315}$ 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 ${ }^{316}$. 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 .

### 15.2.1 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 ${ }^{317}$. 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
```


### 15.2.2 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:

[^106]
## Data Science and Artificial Intelligence for Undergraduates

## Volume 2: Data Visualization, Supervised Learning

- 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 ${ }^{318}$.
- Read and write in binary mode because KMZ files aren't text files.


## Solution:

```
# your solution
```


### 15.2.3 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 1 xml 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 $1 \times \mathrm{xl}$.

## Solution:

```
# your solution
```


### 15.2.4 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
```

[^107]
### 15.3 Climate Change

In this project we download historic weather data from DWD Open Data Portal ${ }^{319}$ 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 and as well as on knowledge obtained in the DWD Open Data Portal (page 443) project.
We use the DWD data set Historical daily station observations for Germany ${ }^{320}$, see description ${ }^{321}$.

### 15.3.1 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 ${ }^{322}$, 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 $f^{323}$ is your friend.

## Solution:

```
# your solution
```


### 15.3.2 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 ${ }^{324}$.

## 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).
[^108]
## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

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

### 15.3.3 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
```


### 15.3.4 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
```


### 15.4 Weather Animation

Based on the data collected in the Climate Change (page 447) 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 41).
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.


### 15.4.1 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 447).
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
```


### 15.4.2 Visualize Data

Task: Load the data saved above and create a map of Germany with Cartopy. Plot with GeoAxes.scatter ${ }^{325}$ and use Matplotlib's animat ion 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 ${ }^{326}$ 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

[^109]
## 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 ${ }^{327}$

- The xMNIST Family of Data Sets (page 451)
- Load QMNIST (page 453)
- QMNIST Feature Reduction (page 455)
- PCA and ANN for QMNIST (page 456)
- CNN for QMNIST (page 456)
- CNN Analysis for QMNIST (page 456)
- IBAN recognition (page 457)
- SVM for QMNIST (page 459)


### 16.1 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.

### 16.1.1 NIST special database 19

Task: Learn about NIST special data base 19 from

- NIST Special Database 19, Handprinted Forms and Characters Database ${ }^{328}$ (sections 1 and 2)
- NIST Special Database $19^{329}$

Answer the following questions:

- Who collected the data?
- 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?

[^110]
## Solution:

```
your answers
```


### 16.1.2 MNIST

Task: Learn about MNIST data set from

- Wikipedia ${ }^{330}$
- The MNIST database of handwritten digits ${ }^{331}$

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
```


### 16.1.3 QMNIST

Task: Learn about QMNIST data set from

- Cold Case: the Lost MNIST Digits ${ }^{332}$
- QMNIST ${ }^{333}$

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?
- Is QMNIST a superset of MNIST?


## Solution:

```
# your answers
```

Task: Download the QMNIST data set.

[^111]
### 16.1.4 EMNIST

Task: Learn about EMNIST data set from

- EMNIST: an extension of MNIST to handwritten letters ${ }^{334}$ (section I and subsections A, B, C of section II)
- The EMNIST Dataset ${ }^{335}$


## 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
```


### 16.2 Load QMNIST

In this project we develop a Python module for loading and preprocessing QMNIST images and metadata. Prerequisites:
-

- The xMNIST Family of Data Sets (page 451)


### 16.2.1 Reading Data

Task: Get QMNIST training and test data from QMNIST GitHub repository ${ }^{336}$ (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:

- NumPy array of shape ( $60000,28,28$ ) or list of 60000 NumPy arrays of shape (28, 28) (range $0 \ldots 1$, type float16), depending on parameter as_list.
- NumPy array of shape (60000, ) containing classes (type uint 8 ).
- NumPy array of shape (60000, ) containing series IDs (type uint 8 ).
- NumPy array of shape (60000, ) containing writer IDs (type uint16).

[^112]
## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

Test your function and show first and last images of training and test data. Print corresponding metainformation. You may use the code from 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 ${ }^{337}$ 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
```


### 16.2.2 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

### 16.2.3 Python Module

Task: Create a Python module qmnist. py providing both functions load and preprocess.

## Solution:

```
# your solution
```

[^113]
### 16.3 QMNIST Feature Reduction

In this project we apply PCA to QMNIST data. Read about Feature Reduction (page 97) before you start.

### 16.3.1 Preprocessing

Task: Use the Python module developed in Load QMNIST (page 453) to load the first 5000 QMNIST training images.

## Solution:

```
# your solution
```

Task: Use the above module to apply the following preprocessing steps from

- auto crop,
- center in $20 \times 20$ image


## Solution:

```
your solution
```


### 16.3.2 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 ${ }^{338}$ or implement calculations manually).

## Solution:

```
# your solution
```

[^114]
## Data Science and Artificial Intelligence for Undergraduates Volume 2: Data Visualization, Supervised Learning

### 16.3.3 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
```


### 16.4 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 233).
You may reuse code from Load QMNIST (page 453) and .
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
```


### 16.5 CNN for QMNIST

In this project we train a CNN for handwritten digit recognition. Read Convolutional Neural Networks (page 252) and CNNs with Keras (page 265) before you start.

You may reuse code from Load QMNIST (page 453) and .
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 $28 \times 28$ images with centered bounding boxes. Save your model to a file.

## Solution:

```
# your solution
```


### 16.6 CNN Analysis for QMNIST

In this project we analyse the CNN trained in CNN for QMNIST (page 456).
You may reuse code from What did the CNN learn? (page 275).
Task: Load the CNN trained and saved in CNN for QMNIST (page 456).

## 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
```


### 16.7 IBAN recognition

We aim at recognizing handwritten IBANs ${ }^{339}$. 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
```


### 16.7.1 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 $x x$ 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 $28 \times 560$ ( 20 images of size $28 \times 28$ placed next to each other) and does not contain the letters DE. Each $28 \times 28$ box contains exactly one $20 \times 20$ 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.

[^115]```
# 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
```


### 16.7.2 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 ${ }^{340}$. Then write a function is_iban which takes an IBAN string and returns True or Fal se 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.

[^116]
# Data Science and Artificial Intelligence for Undergraduates 

Volume 2: Data Visualization, Supervised Learning

```
# your solution
```


### 16.8 SVM for QMNIST

We want to classify QMNIST images with an SVM.

### 16.8.1 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 $20 \times 20$ pixels.

## Solution:

```
# your solution
```


### 16.8.2 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 ${ }^{341}$ and again with SGDClassifier ${ }^{342}$.

## Solution:

```
# your solution
```


### 16.8.3 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
```

[^117]
## PUBLIC TRANSPORT

In this series of projects we visualize and analyze public transport networks based on open data.

- Get Data and Set Up the Environment (page 461)
- Find Connections (page 466)
- Interactive Map (page 468)


### 17.1 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.

### 17.1.1 Download Timetable Data

Timetable data for public transport operators in Germany is available in GTFS format ${ }^{343}$.
Task: Go to gtfs.de ${ }^{344}$. 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 ${ }^{345}$. 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
```

[^118]
### 17.1.2 Download OpenStreetMap Data

To compute walking distances between neighboring public transport stops we'll use data from OpenStreetMap $(\mathrm{OSM})^{346}$. The OSM website provides download of (too) small regions or the whole planet (about 60 GB ). Geofabrik $\mathrm{GmbH}^{347}$ 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
```


### 17.1.3 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 ${ }^{348}$. 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_
    4file=DESTINATION_FILE
```

Task: Extract your region of interest with Osmosis. Note the full terminal command.

## Solution:

```
# your notes
```


### 17.1.4 Conda Environment for GTFS Processing

We want to use the $g t f_{s p y}{ }^{349}$ 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 ${ }^{350}$ (also see corresponding GitHub repo ${ }^{351}$ ).

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 ${ }^{352}$, don't forget to create a corresponding ipykernel for Jupyter and to switch your notebook's kernel to the new one.

## Solution:

[^119]```
# your notes
```


### 17.1.5 Install osmread

The gtfspy package depends on osmread ${ }^{353}$ 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
```


### 17.1.6 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
```


### 17.1.7 Patch gtfspy

The gt fspy 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
```

[^120]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning
## 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)
$\square$
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 zerou
\hookrightarrow(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)"
```


### 17.1.8 Create GTFS Data Base

To speed up routing gt fspy stores all data in an SQLite ${ }^{354}$ data base. That's a usual file with extension sqlite. First step in working with gt fspy 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
```


### 17.1.9 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 $g t f s p y$ 's $g t f s$ module.
2. Create a FilterExtract object, defined in gtfspy's filter module.
3. Call the FilterExtract object's filter method.

Task: Have look at gt fspy's source to learn how to use the above mentioned objects and functions. Then filter the data base by region (hint: 'buffer zone' in $g t$ fspy ' s source is the region of interest).

## Solution:

```
# your solution
```


### 17.1.10 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

[^121]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.

### 17.1.11 Use the Data Base

To use the SQLite data base we have to create a GTFS object, definded in $g t f s p y$ 's $g t f s$ 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
```


### 17.2 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 461) project. Basic Pandas knowledge is required to solve the tasks (read , , before you start, may be of interest, too).

### 17.2.1 Data Base and Time Frame

Task: Connect to the data base, that is, create a gt fspy.gt fs. GTFS object.

## Solution:

```
your solution
```

The routing algorithm of gt fspy looks for public transport connections in a user-defined time frame. Start and end time have to be provided in Unix time ${ }^{355}$.

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:

$355 \mathrm{https}: / /$ en.wikipedia.org/wiki/Unix_time

```
# your solution
```


### 17.2.2 Arrival Stop

The routing algorithm of gtfspy computes public transport connections from all stops in the data base to a userdefined 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
```


### 17.2.3 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 (argumentstransit_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 gt fspy'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
```


## Data Science and Artificial Intelligence for Undergraduates

### 17.2.4 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
```


### 17.2.5 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
```


### 17.2.6 Save Results

Task: Save your stops data frame to a CSV file.

## Solution:

```
* your solution
```


### 17.3 Interactive Map

In this project we visualize the results of the Find Connections (page 466) project on an interactive map. Have a look at Folium (page 59) before you start.

### 17.3.1 Prepare Data

Task: Load the CSV file created in the Find Connections (page 466) 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
```


### 17.3.2 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
```


### 17.3.3 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 ${ }^{356}$ instead of folium.plugins.
Markercluster ${ }^{357}$, because the latter may be very slow if you have many stops. Because FastMark-
erCluster 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 FastMark-
erclusters 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:

[^122]```
<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
```


### 17.3.4 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 ${ }^{358}$ 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
```

[^123]
## CORONA DEATHS

In this project we collect and/or compute death rates before and during the Corona pandemic in Germany. You should read before you start.

### 18.1 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) ${ }^{359}$ in CSV format:

- Destatis, table 12613-0006360
- Destatis, table 12411-0001 ${ }^{361}$
- Destatis, Sonderreihe mit Beiträgen für das Gebiet der ehemaligen DDR, Heft $3^{362}$
- Destatis, table 12411-0020363

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 $1 \ldots 12$ ), men, women


### 18.2 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).

[^124]
## Solution:

\# your solution

### 18.3 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 15 th 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

## 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 5).

### 19.1 Download and Cleaning

Information on public trees in Chemnitz are available online: Chemnitz trees data set ${ }^{364}$.
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
```


### 19.2 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
```

[^125]
### 19.3 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 ${ }^{365}$ showing the fraction of total population for each dominant species. Include one slice for all non-dominant trees.

Create a stacked bar plot ${ }^{366}$ 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
```


### 19.4 Presentation

Task: Create PDF file in A4 format showing all information extracted above. Use whatever software you like. LibreOffice Write ${ }^{367}$ 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.

[^126]
## FORGED BANKNOTES

In this series of projects we apply several machine learning concepts and methods to automatically identify forged banknotes.

- Detecting Forgery with $k-N N$ (page 475)
- Quality Measures (page 477)
- Hyperparameter Optimization (page 479)
- Decision Tree (page 479)
- Random Forest (page 480)
- Support-Vector Machine (page 481)
- Naive Bayes Classification (page 481 )


### 20.1 Detecting Forgery with k-NN

Banknotes have lots of security features, some well known (see Deutsche Bundesbank ${ }^{368}$ ) and some less known like the EURion constellation ${ }^{369}$. 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 ${ }^{370}$. That technique produces extremely sharp edges if steel plates ${ }^{371}$ are used and cannot be realized with off-the-shelf machines. Researchers from Institut für industrielle Informationstechnik ${ }^{372}$ at Technischen Hochschule Ostwestfalen-Lippe ${ }^{373}$ created a data set for training banknote authentication systems based on scanned images of banknotes.

### 20.1.1 The Data Set

The data set ${ }^{374}$ is available from UCI Machine Learning Repository ${ }^{375}$. 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 ${ }^{376}$ to find out how to interpret class labels.

## Solution:

[^127]```
# 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 ${ }^{377}$ 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
```


### 20.1.2 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
```


#### Abstract

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.


### 20.1.3 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
```

[^128]
### 20.2 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.

### 20.2.1 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 ${ }^{378}$ 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
```


### 20.2.2 Quality Measures

## Confusion Matrix

Task: Generate and print the confusion matrix for the test set. Compare results to Scikit-Learn's confusion_matrix ${ }^{379}$.

```
# your solution
```


## Accuracy and Balanced Accuracy

Task: Calculate unbalanced and balanced accuracy on the test set. Compare results to Scikit-Learn's accuracy_score ${ }^{380}$ and balanced_accuracy_score ${ }^{381}$.

## Solution:

```
# your solution
```

[^129]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised LearningPrecision, Recall, F1-score

Task: Calculate and print precision, recall and F1-score on the test set. Compare results to Scikit-Learn's precision_score ${ }^{382}$, recall_score ${ }^{383}$, f1_score ${ }^{384}$.

## 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 ${ }^{385}$.

```
# 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 ${ }^{386}$. Compare results to Scikit-Learn's roc_curve ${ }^{387}$.

```
# your solution
```

Task: Plot the ROC curve for the test set. Compare results to Scikit-Learn's RocCurveDisplay. from_predictions ${ }^{388}$.

```
# your solution
```

Task: Calculate and print AUC for the test set. Compare results to Scikit-Learn's roc_auc_score ${ }^{389}$.

```
# your solution
```

[^130]
### 20.3 Hyperparameter Optimization

In this project we revisit Detecting Forgery with $k-N N$ (page 475) and add hyperparameter optimization.

### 20.3.1 Load Data Set

Task: Load the banknotes data set (cf. Detecting Forgery with $k$ - $N N$ (page 475)). Drop the entropy column.

```
your solution
```


### 20.3.2 Grid Search with Cross Validation

Task: Create a $k$-NN model with Scikit-Learn's KNeighborsClassifier ${ }^{390}$. 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
```


### 20.3.3 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 ${ }^{391}$ or contourf $f^{392}$.

## Solution:

```
# your solution
```


### 20.4 Decision Tree

In this project we solve the banknote classification task with a decision tree.

### 20.4.1 Load Data Set

Task: Load the banknotes data set (cf. Detecting Forgery with $k$ - $N N$ (page 475)). Drop the entropy column.

```
# your solution
```

[^131]
## Data Science and Artificial Intelligence for Undergraduates

### 20.4.2 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
```


### 20.4.3 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 ${ }^{393}$ or contourf $f^{394}$.

## Solution:

```
your solution
```


### 20.5 Random Forest

In this project we solve the banknote classification task with a random forest.

### 20.5.1 Load Data Set

Task: Load the banknotes data set (cf. Detecting Forgery with $k$ - $N N$ (page 475)). Drop the entropy column.

```
# your solution
```


### 20.5.2 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
```


### 20.5.3 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 ${ }^{395}$ or contourf $f^{396}$.

## Solution:

```
# your solution
```

[^132]
### 20.6 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.

### 20.6.1 Load Data Set

Task: Load the banknotes data set (cf. Detecting Forgery with $k-N N$ (page 475)). Drop the entropy column.

```
# your solution
```


### 20.6.2 SVM

Task: Use Scikit-Learns's SVC ${ }^{397}$ 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
```


### 20.6.3 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
```


### 20.7 Naive Bayes Classification

### 20.7.1 Load Data Set

Task: Load the banknotes data set (cf. Detecting Forgery with $k-N N$ (page 475)). Drop the entropy column.

```
# your solution
```

[^133]
## Data Science and Artificial Intelligence for Undergraduates

 Volume 2: Data Visualization, Supervised Learning
### 20.7.2 Naive Bayes

Task: Use Scikit-Learns's GaussianNB ${ }^{398}$ to create a model for banknote authentication.
Solution:

```
# your solution
```


### 20.7.3 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
```

[^134]
## HOUSE PRICES

This series of projects extends the results obtained in Worked Example: House Prices I (page 142) and Worked Example: House Prices II (page 175).

- House Prices GUI (page 483)
- House Prices ANN (page 484)
- A Random Forest for House Prices (page 485)


### 21.1 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 175).

### 21.1.1 The Model

In Worked Example: House Prices II (page 175) 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
```


### 21.1.2 GUI

We use the ipywidgets ${ }^{399}$ module to create a GUI in Jupyter Lab. Have a look at the following documentation pages:

- Simple Widget Introduction ${ }^{400}$
- Text ${ }^{401}$
- Combobox ${ }^{402}$
- RadioButtons ${ }^{403}$
- Button ${ }^{404}$
- Output ${ }^{405}$

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
```


### 21.2 House Prices ANN

In this project we solve the house price prediction problem from Worked Example: House Prices II (page 175) with an ANN. Read ANN Basics (page 207) and Training ANNs (page 216) before your start.

### 21.2.1 The ANN

In Worked Example: House Prices II (page 175) 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
```

${ }^{399} \mathrm{https}: / /$ ipywidgets.readthedocs.io/en/stable/
$400 \mathrm{https}: / /$ ipywidgets.readthedocs.io/en/stable/examples/Widget\ Basics.html
${ }^{401} \mathrm{https}: / /$ ipywidgets.readthedocs.io/en/stable/examples/Widget\ List.html\#text
$402 \mathrm{https}: / / \mathrm{ipywidgets} . r e a d t h e d o c s . i o / e n /$ stable/examples/Widget\%20List.html\#combobox
${ }^{403} \mathrm{https}: / /$ ipywidgets.readthedocs.io/en/stable/examples/Widget\ List.html\#radiobuttons
$404 \mathrm{https}: / /$ ipywidgets.readthedocs.io/en/stable/examples/Widget\ List.html\#button
${ }^{405} \mathrm{https}: / /$ ipywidgets.readthedocs.io/en/stable/examples/Widget\ List.html\#output

### 21.2.2 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
```


### 21.3 A Random Forest for House Prices

In this project we solve the house price prediction problem from Worked Example: House Prices II (page 175) with a random forest. Read Bagging (page 354) 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
```


## HYPERPARAMETER OPTIMIZATION FOR CATS AND DOGS

In CNNs with Keras (page 265) we trained a CNN for classifying images of cats and dogs. Using the knownledge from ANNs with Keras (page 233) 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
```

In this series of projects we analyze a large corpus of blogs from the world-wide web.

- Blog Author Classification (Training) (page 489)
- Blog Author Classification (Test) (page 492)


### 23.1 Blog Author Classification (Training)

Instead of deriving author information from single blog posts like in Text Classification (page 381) 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 492)), 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.

### 23.1.1 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
```


## Data Science and Artificial Intelligence for Undergraduates

### 23.1.2 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 $0,1,2, \ldots$ lists do the job. For unk nown 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
```


### 23.1.3 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
```


### 23.1.4 Text to Numbers

Task: Use Scikit-Learn's TFidfVectorizer ${ }^{406}$ 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 $t f i d f \_v e c t . v o c a b u l a r y \_$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.

[^135]
# Data Science and Artificial Intelligence for Undergraduates 

Volume 2: Data Visualization, Supervised Learning

## Solution:

your solution

### 23.1.5 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
```


### 23.1.6 Age Model

Task: Train naive Bayes and SVM models for age prediction.

## Solution:

```
your solution
```


### 23.1.7 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
```


## Data Science and Artificial Intelligence for Undergraduates

### 23.1.8 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
```


### 23.2 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 489) and we have to apply all the necessary preprocessing steps to the downloaded posts.

### 23.2.1 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 ${ }^{407}$ and join all posts to one string.

## Solution:

```
# your solution
```

[^136]
# Data Science and Artificial Intelligence for Undergraduates 

Volume 2: Data Visualization, Supervised Learning

### 23.2.2 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
```


### 23.2.3 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

## Part V

## Computer Science

## CLOUD COMPUTING


#### Abstract

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.


### 24.1 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.

### 24.1.1 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.

### 24.1.2 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. 24.1: Programs and libraries on a physical machine.

### 24.1.3 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. 24.2: 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.

### 24.1.4 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.

## computer



Fig. 24.3: 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.


[^0]:    ${ }^{1} \mathrm{https}: / /$ datascience.fh-zwickau.de
    ${ }^{2}$ https://www.fh-zwickau.de
    ${ }^{3}$ https://www.fh-zwickau.de/~jef 19jdw
    ${ }^{4} \mathrm{https}: / /$ www.python.org
    ${ }^{5} \mathrm{https}: / / j u p y t e r b o o k . o r g$
    ${ }^{6} \mathrm{https}: / /$ executablebooks.org
    ${ }^{7}$ https://www.fh-zwickau.de/~jef 19jdw

[^1]:    ${ }^{8}$ https://matplotlib.org/
    ${ }^{9} \mathrm{https}: / / \mathrm{en}$.wikipedia.org/wiki/MATLAB
    ${ }^{10} \mathrm{https}: / /$ matplotlib.org/stable/api/pyplot_summary.html

[^2]:    ${ }^{11} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.pyplot.figure.html
    $12 \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.pyplot.plot.html
    ${ }^{13} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.pyplot.show.html

[^3]:    ${ }^{14} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.pyplot.xlabel.html
    ${ }^{15} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.pyplot.ylabel.html
    ${ }^{16} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.pyplot.title.html
    ${ }^{17} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.html
    ${ }^{18} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.lines.Line2D.html
    ${ }^{19} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.plot.html

[^4]:    ${ }^{20} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/figure_api.html\#matplotlib.figure.Figure.add_axes

[^5]:    ${ }^{21} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/figure_api.html\#matplotlib.figure.Figure.add_subplot

[^6]:    22 https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html
    ${ }^{23} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/figure_api.html\#matplotlib.figure.Figure.add_gridspec
    ${ }^{24} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.gridspec.GridSpec.html

[^7]:    ${ }^{25} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.gridspec.SubplotSpec.html
    ${ }^{26} \mathrm{https}: / /$ matplotlib.org/stable/api/_as_gen/matplotlib.gridspec.SubplotSpec.html\#matplotlib.gridspec.SubplotSpec.subgridspec
    ${ }^{27} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.gridspec.GridSpecFromSubplotSpec.html

[^8]:    ${ }^{28} \mathrm{https}: / /$ matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.suptitle.html

[^9]:    ${ }^{29} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.axis.html
    ${ }^{30} \mathrm{https}: / / \mathrm{matplotlib} . o r g / s t a b l e / a p i / \_a s \_g e n / m a t p l o t l i b . a x e s . A x e s . s e t \_x l i m . h t m l ~$
    ${ }^{31} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.set_ylim.html
    $32 \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.set_xscale.html
    ${ }^{33} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.set_yscale.html

[^10]:    ${ }^{34} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.set_xticks.html
    $35 \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.set_yticks.html
    ${ }^{36} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.set_xticklabels.html
    ${ }^{37} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.set_yticklabels.html

[^11]:    ${ }^{38} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.tick_params.html
    ${ }^{39} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.grid.html

[^12]:    ${ }^{40} \mathrm{https}: / /$ matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.twiny.html
    ${ }^{41} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.twinx.html

[^13]:    ${ }^{42} \mathrm{https}: / /$ matplotlib.org/stable/api/projections/polar.html\#matplotlib.projections.polar.PolarAxes

[^14]:    ${ }^{43} \mathrm{https}: / /$ matplotlib.org/stable/_images/sphx_glr_named_colors_002.png
    ${ }^{44} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/_images/sphx_glr_named_colors_003.png
    ${ }^{45} \mathrm{https}: / / \mathrm{i}$. stack.imgur.com/nCk6u.jpg

[^15]:    ${ }^{46} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.scatter.html
    ${ }^{47} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.colors.Normalize.html
    ${ }^{48} \mathrm{https}: / /$ matplotlib.org/stable/gallery/color/colormap_reference.html
    ${ }^{49} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.pyplot.colorbar.html
    ${ }^{50} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/figure_api.html\#matplotlib.figure.Figure.colorbar

[^16]:    ${ }^{51} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.legend.html

[^17]:    ${ }^{52}$ https://en.wikipedia.org/wiki/TeX

[^18]:    ${ }^{53}$ https://en.wikipedia.org/wiki/LaTeX
    ${ }^{54} \mathrm{https}: / / \mathrm{matplotlib} . o r g / s t a b l e / a p i / \_a s \_g e n / m a t p l o t l i b . a x e s . A x e s . t e x t . h t m l ~$

[^19]:    ${ }^{55} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.annotate.html

[^20]:    ${ }^{56} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.add_artist.html

[^21]:    ${ }^{57} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/patches_api.html
    
    ${ }^{59} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.imshow.html
    ${ }^{60} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/image_api.html\#matplotlib.image.imread
    ${ }^{61} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.pyplot.imread.html

[^22]:    ${ }^{62} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.plot.html
    ${ }^{63} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.scatter.html
    ${ }^{64} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.bar.html
    ${ }^{65} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.pie.html
    ${ }^{66} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.hist.html
    ${ }^{67} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.contour.html
    ${ }^{68} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/axes_api.html\#plotting
    ${ }^{69} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/gallery/index.html
    ${ }^{70} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/figure_api.html\#matplotlib.figure.Figure.save_fig

[^23]:    ${ }^{71} \mathrm{https}: / / w w w . g i m p . o r g$
    ${ }^{72} \mathrm{https}: / / i n k s c a p e . o r g$
    ${ }^{73} \mathrm{https}: / /$ matplotlib.org/stable/api/toolkits/mplot3d.html
    ${ }^{74} \mathrm{https}: / /$ matplotlib.org/stable/api/toolkits/mplot3d/axes3d.html

[^24]:    ${ }^{75} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/mpl_toolkits.mplot3d.axes3d.Axes3D.scatter.html

[^25]:    ${ }^{76}$ https://github.com/matplotlib/ipympl

[^26]:    ${ }^{77} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.animation.FuncAnimation.html

[^27]:    ${ }^{78} \mathrm{https}: / / \mathrm{en}$. wikipedia.org/wiki/Sam_Seaborn
    ${ }^{79} \mathrm{https}: / /$ github.com/mwaskom/seaborn/issues/229

[^28]:    ${ }^{80} \mathrm{https}: / /$ seaborn.pydata.org/tutorial.html
    ${ }^{81} \mathrm{https}: / /$ seaborn.pydata.org/examples/index.html

[^29]:    $82 \mathrm{https}: / /$ scitools.org.uk/cartopy/docs/latest/
    ${ }^{83} \mathrm{https}: / / \mathrm{github} . c o m / j w a s s / m p l l e a f l e t$
    ${ }^{84} \mathrm{https}: / /$ www.openstreetmap.org
    ${ }^{85} \mathrm{https}: / /$ python-visualization.github.io/folium/
    ${ }^{86} \mathrm{https}: / /$ scitools.org.uk/cartopy/docs/latest/reference/generated/cartopy.mpl.geoaxes.GeoAxes.html
    ${ }^{87}$ https://scitools.org.uk/cartopy/docs/v0.15/crs/projections.html

[^30]:    ${ }^{88} \mathrm{https}: / /$ scitools.org.uk/cartopy/docs/latest/reference/generated/cartopy.mpl.geoaxes.GeoAxes.html\#cartopy.mpl.geoaxes.GeoAxes.set_ extent
    ${ }^{89} \mathrm{https}: / / \mathrm{www}$. naturalearthdata.com
    ${ }^{90} \mathrm{https}: / /$ scitools.org.uk/cartopy/docs/latest/reference/generated/cartopy.mpl.geoaxes.GeoAxes.html\#cartopy.mpl.geoaxes.GeoAxes.add_ feature

[^31]:    ${ }^{91} \mathrm{https}: / / \mathrm{www}$. naturalearthdata.com/downloads

[^32]:    ${ }^{92} \mathrm{https}: / /$ scitools.org.uk/cartopy/docs/latest/reference/generated/cartopy.mpl.geoaxes.GeoAxes.html\#cartopy.mpl.geoaxes.GeoAxes.set_ global
    ${ }^{93} \mathrm{https}: / / \mathrm{www}$. osm.org
    ${ }^{94} \mathrm{https}: / /$ github.com/jwass/mplleaflet/issues/80
    $95 \mathrm{https}: / / \mathrm{github} . c o m /$ plotly/plotly.py/issues/2913\#issuecomment-730619757

[^33]:    /home/jef19jdw/anaconda3/envs/ds_book/lib/python3.10/site-packages/IPython/core/ $\rightarrow d i s p l a y . p y: 431: ~ U s e r W a r n i n g: ~ C o n s i d e r ~ u s i n g ~ I P y t h o n . d i s p l a y . I F r a m e ~ i n s t e a d ~$ warnings.warn("Consider using IPython.display.IFrame instead")
    <IPython.core.display.HTML object>

[^34]:    ${ }^{96} \mathrm{https}: / /$ plotly.com/python/
    ${ }^{97}$ https://plotly.com/graphing-libraries/
    $98 \mathrm{http}: / /$ plotly.com
    ${ }^{99} \mathrm{https}: / /$ plotly.com/python/plotly-express

[^35]:    ${ }^{100} \mathrm{https}: / /$ plotly.com/python-api-reference/plotly.graph_objects.html
    ${ }^{101} \mathrm{https}: / /$ plotly.com/python-api-reference/generated/plotly.graph_objects.Figure.html
    102 https://plotly.com/python-api-reference/generated/plotly.graph_objects.Figure.html\#plotly.graph_objects.Figure.write_html
    ${ }^{103} \mathrm{https}: / / \mathrm{github} . c o m /$ plotly/plotly.js/blob/master/dist/plotly.min.js

[^36]:    ${ }^{104} \mathrm{https}: / /$ python-visualization.github.io/folium/
    $105 \mathrm{https}: / / \mathrm{www} . o s m . o r g$
    $106 \mathrm{https}: / /$ leafletjs.com/
    ${ }^{107} \mathrm{https}: / /$ python-visualization.github.io/folium/quickstart.html

[^37]:    ${ }^{108} \mathrm{https}: / /$ python-visualization.github.io/folium/modules.html\#folium.raster_layers.ImageOverlay
    $109 \mathrm{https}: / /$ python-visualization.github.io/folium/plugins.html\#folium.plugins.MarkerCluster
    ${ }^{110} \mathrm{https}: / /$ python-visualization.github.io/folium/plugins.html\#folium.plugins.FastMarkerCluster
    ${ }^{111} \mathrm{https}: / /$ python-visualization.github.io/folium/modules.html\#folium.map.LayerControl
    $112 \mathrm{https}: / /$ python-visualization.github.io/branca/element.html
    ${ }^{113} \mathrm{https}: / /$ python-visualization.github.io/folium/modules.html\#folium.folium.Map
    114 https://github.com/python-visualization/branca
    $115 \mathrm{https}: / /$ python-visualization.github.io/branca/element.html\#branca.element.Element.get_root

[^38]:    116 https://en.wikipedia.org/wiki/AlphaGo_Zero
    117 https://en.wikipedia.org/wiki/ChatGPT
    ${ }^{118} \mathrm{https}: / /$ en.wikipedia.org/wiki/Stable_Diffusion

[^39]:    119 https://scikit-learn.org/stable/
    120 https://www.tensorflow.org/
    121 https://keras.io/
    122 https://pytorch.org/

[^40]:    ${ }^{123} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

[^41]:    ${ }^{124} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html
    ${ }^{125} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[^42]:    ${ }^{126} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[^43]:    ${ }^{127} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.htm
    ${ }^{128} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html\#sklearn.neighbors.KNeighborsRegressor. predict

[^44]:    129 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

[^45]:    ${ }^{130} \mathrm{https}: / /$ en.wikipedia.org/wiki/Orthant

[^46]:    ${ }^{131}$ https://en.wikipedia.org/wiki/Volume_of_an_n-ball
    $132 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise_distances.html

[^47]:    $133 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html

[^48]:    ${ }^{134} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html\#sklearn.metrics.mean_absolute_error

[^49]:    $135 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html

[^50]:    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.

[^51]:    ${ }^{136} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html
    ${ }^{137} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

[^52]:    $138 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html

[^53]:    $139 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
    $140 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
    $141 \mathrm{https}: / /$ en.wikipedia.org/wiki/Harmonic_mean
    $142 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html
    ${ }^{143}$ https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html
    $144 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
    $145 \mathrm{https}: / / \mathrm{scikit}-$ learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html\#sklearn.metrics.RocCurveDisplay.from_ predictions

[^54]:    ${ }^{146} \mathrm{http}: / /$ madrury.github.io/jekyll/update/statistics/2017/06/21/auc-proof.html

[^55]:    ${ }^{147}$ https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[^56]:    ${ }^{148}$ https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[^57]:    ${ }^{149} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html

[^58]:    ${ }^{150}$ https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[^59]:    ${ }^{151} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

[^60]:    ${ }^{152} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.model_selection.ParameterGrid.html

[^61]:    ${ }^{153} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
    $154 \mathrm{https}: / /$ scikit-learn.org/stable/modules/model_evaluation.html\#scoring-parameter

[^62]:    ${ }^{155} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
    156 https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html

[^63]:    ${ }^{157} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.model_selection.validation_curve.html

[^64]:    $158 \mathrm{https}: / /$ www.cardekho.com
    $159 \mathrm{https}: / / \mathrm{www}$. kaggle.com
    $160 \mathrm{http}: / /$ opendatacommons.org/licenses/dbcl/1.0
    $161 \mathrm{https}: / /$ opendatacommons.org/licenses/odbl/summary

[^65]:    $162 \mathrm{https}: / /$ en.wikipedia.org/wiki/Anscombe\%27s_quartet
    ${ }^{163} \mathrm{https}: / /$ seaborn.pydata.org/tutorial/axis_grids.html\#plotting-pairwise-data-relationships

[^66]:    $164 \mathrm{https}: / /$ seaborn.pydata.org/tutorial/regression.html

[^67]:    ${ }^{165} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html
    $166 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html\#sklearn.preprocessing. PolynomialFeatures.fit_transform

[^68]:    167 https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

[^69]:    ${ }^{168} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html
    ${ }^{169} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html
    ${ }^{170} \mathrm{https}: / / \mathrm{www} . k a g g l e . c o m /$ scriptsultan
    $171 \mathrm{https}: / / \mathrm{www} . i m m o b i l i e n s c o u t 24 . d e$
    $172 \mathrm{https}: / / \mathrm{www} . k a g g l e . c o m / s c r i p t s u l t a n / g e r m a n-h o u s e-p r i c e s ~$
    $173 \mathrm{https}: / /$ creativecommons.org/licenses/by-nc-sa/4.0
    $174 \mathrm{https}: / /$ pandas.pydata.org/pandas-docs/stable/user_guide/options.html\#frequently-used-options

[^70]:    $175 \mathrm{https}: / / \mathrm{en}$. wikipedia.org/wiki/Log-normal_distribution

[^71]:    $176 \mathrm{https}: / /$ pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.cat.codes.html

[^72]:    ${ }^{177}$ https://www.kaggle.com/scriptsultan
    178 https://www.immobilienscout24.de
    $179 \mathrm{https}: / / \mathrm{www} . k a g g l e . c o m / s c r i p t s u l t a n / g e r m a n-h o u s e-p r i c e s ~$
    $180 \mathrm{https}: / /$ creativecommons.org/licenses/by-nc-sa/4.0

[^73]:    ${ }^{181} \mathrm{https}: / / w w w . r e g i o n a l s t a t i s t i k . d e$
    182 https://www.govdata.de/dl-de/by-2-0
    $183 \mathrm{https}: / / \mathrm{www}$. regionalstatistik.de/genesis/online?operation=abruftabelleBearbeiten\&levelindex=1\&levelid= 1614168819692 \&auswahloperation=abruftabelleAuspraegungAuswaehlen\&auswahlverzeichnis=ordnungsstruktur\&auswahlziel= werteabruf\&code=AI-S-01\&auswahltext=\&werteabruf=Werteabruf\#abreadcrumb
    $184 \mathrm{https}: / / \mathrm{www}$. 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

[^74]:    ${ }^{185} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html
    $186 \mathrm{https}: / /$ scikit-learn.org/stable/modules/linear_model.html\#huber-regression
    $187 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html

[^75]:    $188 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.linear_model.RANSACRegressor.html

[^76]:    ${ }^{189} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
    $190 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
    ${ }^{191} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegressionCV.html

[^77]:    $192 \mathrm{https}: / / x k c d . c o m / 2173$
    ${ }^{193} \mathrm{https}: / /$ en.wikipedia.org/wiki/Cellular_automaton
    $194 \mathrm{https}: / /$ en.wikipedia.org/wiki/Collective_intelligence

[^78]:    $195 \mathrm{https}: / /$ en.wikipedia.org/wiki/Activation_function\#Comparison_of_activation_functions

[^79]:    ${ }^{239} \mathrm{https}: / /$ keras.io/api/layers/regularization_layers/dropout/

[^80]:    ${ }^{240} \mathrm{https}: / /$ keras.io/api/layers/regularizers/

[^81]:    ${ }^{241} \mathrm{https}: / / \mathrm{www}$. tensorflow.org/api_docs/python/tf/Tensor
    ${ }^{242} \mathrm{https}: / / \mathrm{keras} . i o /$ api/preprocessing/image/\#loadimg-function
    ${ }^{243} \mathrm{https}: / /$ pillow.readthedocs.io/en/stable/reference/Image.html\#PIL.Image.Image

[^82]:    ${ }^{244} \mathrm{https}: / /$ keras.io/api/layers/base_layer/\#getweights-method

[^83]:    ${ }^{245} \mathrm{https}: / / \mathrm{www} . t e n s o r f l o w . o r g / a p i \_d o c s / p y t h o n / t \mathrm{f} /$ convert_to_tensor
    ${ }^{246} \mathrm{https}: / /$ www.tensorflow.org/api_docs/python/tf/GradientTape
    ${ }^{247} \mathrm{https}: / / \mathrm{www} . t e n s o r f l o w . o r g / a p i \_d o c s / p y t h o n / t f / G r a d i e n t T a p e \# w a t c h ~$
    

[^84]:    ${ }^{249} \mathrm{https}: / / e n . w i k i p e d i a . o r g / w i k i / D e e p D r e a m ~$
    $250 \mathrm{https}: / /$ en.wikipedia.org/wiki/Neural_Style_Transfer

[^85]:    ${ }^{251} \mathrm{https}: / / \mathrm{keras} . i o / a p i / l a y e r s /$ preprocessing_layers/image_augmentation/
    252 https://keras.io/api/applications/

[^86]:    ${ }^{253} \mathrm{https}: / /$ github.com/tensorflow/tensorflow/issues/55639

[^87]:    ${ }^{278}$ https://alex.smola.org/papers/2004/SmoSch04.pdf
    $279 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html

[^88]:    ${ }^{280} \mathrm{https}: / / e n . w i k i p e d i a . o r g / w i k i / B a y e s \% 27 \_t h e o r e m ~$
    ${ }^{281} \mathrm{https}: / / \mathrm{en}$. wikipedia.org/wiki/Independence_(probability_theory)

[^89]:    $282 \mathrm{https}: / /$ en.wikipedia.org/wiki/Multinomial_distribution

[^90]:    ${ }^{283} \mathrm{https}: / /$ en.wikipedia.org/wiki/Kernel_density_estimation
    ${ }^{284} \mathrm{https}: / /$ scikit-learn.org/stable/modules/classes.html\#module-sklearn.naive_bayes

[^91]:    ${ }^{285} \mathrm{https}: / / \mathrm{u} . \mathrm{cs}$. biu.ac.il/~koppel/BlogCorpus.htm
    ${ }^{286}$ https://u.cs.biu.ac.i1/~schlerj/schler_springsymp06.pdf
    ${ }^{287}$ https://en.wikipedia.org/wiki/XML

[^92]:    ${ }^{288} \mathrm{https}: / /$ docs.python.org/3/library/xml.etree.elementtree.html

[^93]:    ${ }^{289} \mathrm{https}: / /$ github.com/Mimino666/langdetect

[^94]:    ${ }^{290} \mathrm{https}: / /$ en.wikipedia.org/wiki/Word2vec

[^95]:    ${ }^{291} \mathrm{https}: / / w w w . n l t k . o r g$
    292 https://docs.python.org/3/library/re.html\#regular-expression-syntax
    ${ }^{293} \mathrm{https}: / / \mathrm{www} . n l t k . o r g / a p i / n l t k . s t e m . w o r d n e t . h t m l$
    $294 \mathrm{https}: / /$ wordnet.princeton.edu
    $295 \mathrm{http}: / /$ wordnetweb.princeton.edu/perl/webwn
    $296 \mathrm{https}: / /$ wordnet.princeton.edu/documentation/morphy7wn

[^96]:    ${ }^{297} \mathrm{https}: / / \mathrm{www} . l i n g . u p e n n . e d u /$ courses/Fall_2003/ling001/penn_treebank_pos.html
    ${ }^{298}$ https://www.nltk.org/_modules/nltk/stem/wordnet.html

[^97]:    ${ }^{299} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

[^98]:    ${ }^{300} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.preprocessing.normalize.html
    ${ }^{301} \mathrm{https}: / / \mathrm{scikit-learn.org} /$ stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[^99]:    ${ }^{302} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
    ${ }^{303} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
    ${ }^{304} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

[^100]:    ${ }^{305} \mathrm{https}: / /$ people.eecs.berkeley.edu/~brecht/papers/07.rah.rec.nips.pdf

[^101]:    ${ }^{307} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[^102]:    ${ }^{308}$ https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[^103]:    ${ }^{309} \mathrm{https}: / /$ matplotlib.org/stable/api/_as_gen/matplotlib.colors.LinearSegmentedColormap.html\#matplotlib.colors. LinearSegmentedColormap.from_list

[^104]:    ${ }^{310} \mathrm{https}: / / \mathrm{www} . d w d . d e$
    $311 \mathrm{https}: / /$ www.dwd.de
    $312 \mathrm{https}: / / \mathrm{www} . \mathrm{dwd}$.de/opendata

[^105]:    ${ }^{313} \mathrm{https}: / / \mathrm{www} . o p e n s t r e e t m a p . o r g / \# m a p=16 / 50.6879 / 12.4329$

[^106]:    314 https://www.dwd.de/opendata
    315 https://www.dwd.de
    ${ }^{316} \mathrm{https}: / / \mathrm{www} . d w d . d e / E N / o u r s e r v i c e s / m e t \_a p p l i c a t i o n \_m o s m i x / m e t \_a p p l i c a t i o n \_m o s m i x . h t m l ~$
    $317 \mathrm{https}: / / \mathrm{www} . d w d . d e / E N / o u r s e r v i c e s / m e t \_a p p l i c a t i o n \_m o s m i x / m e t \_a p p l i c a t i o n \_m o s m i x . h t m l ~$

[^107]:    ${ }^{318} \mathrm{https}: / /$ docs.python.org/3/library/os.path.html\#os.path.isfile

[^108]:    $319 \mathrm{https}: / / w w w . d w d . d e / o p e n d a t a$
    ${ }^{320} \mathrm{https}: / /$ opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/historical
    ${ }^{321} \mathrm{https}: / / o p e n d a t a . d w d . d e / c l i m a t e \_e n v i r o n m e n t / C D C / o b s e r v a t i o n s \_g e r m a n y / c l i m a t e / d a i l y / k l / h i s t o r i c a l / D E S C R I P T I O N \_o b s g e r m a n y \_~$ climate_daily_kl_historical_en.pdf
    322 https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/historical/KL_Tageswerte_Beschreibung_ Stationen.txt
    ${ }^{323} \mathrm{https}: / /$ pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_fwf.html
    ${ }^{324} \mathrm{https}: / / o p e n d a t a . d w d . d e / c l i m a t e \_e n v i r o n m e n t / C D C / o b s e r v a t i o n s \_g e r m a n y / c l i m a t e / d a i l y / k l / h i s t o r i c a l / ~$

[^109]:    ${ }^{325} \mathrm{https}: / /$ scitools.org.uk/cartopy/docs/latest/reference/generated/cartopy.mpl.geoaxes.GeoAxes.html\#cartopy.mpl.geoaxes.GeoAxes.scatter
    ${ }^{326} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.artist.Artist.remove.html

[^110]:    327 https://www.nist.gov
    328 https://www.nist.gov/system/files/documents/srd/nistsd19.pdf
    329 https://www.nist.gov/srd/nist-special-database-19

[^111]:    ${ }^{330} \mathrm{https}: / / e n . w i k i p e d i a . o r g / w i k i / M N I S T \_d a t a b a s e$
    $331 \mathrm{http}: / / y a n n . l e c u n . c o m / e x d b / m n i s t /$
    $332 \mathrm{https}: / /$ arxiv.org/pdf/1905.10498.pdf
    ${ }^{333} \mathrm{https}: / /$ github.com/facebookresearch/qmnist

[^112]:    ${ }^{334} \mathrm{https}: / /$ arxiv.org/pdf/1702.05373.pdf
    $335 \mathrm{https}: / / \mathrm{www}$. nist.gov/itl/products-and-services/emnist-dataset
    336 https://github.com/facebookresearch/qmnist

[^113]:    ${ }^{337}$ https://docs.python.org/3/library/gzip.html

[^114]:    ${ }^{338} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html\#sklearn.decomposition.PCA.inverse_transform

[^115]:    339 https://en.wikipedia.org/wiki/International_Bank_Account_Number

[^116]:    ${ }^{340} \mathrm{https}: / /$ en.wikipedia.org/wiki/International_Bank_Account_Number\#Validating_the_IBAN

[^117]:    ${ }^{341} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
    ${ }^{342} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

[^118]:    ${ }^{343} \mathrm{https}: / /$ en.wikipedia.org/wiki/GTFS
    344 https://www.gtfs.de
    $345 \mathrm{https}: / /$ www.gtfs.de

[^119]:    346 https://www.osm.org
    347 http://www.geofabrik.de/
    348 https://wiki.openstreetmap.org/wiki/Osmosis
    349 https://github.com/CxAalto/gtfspy
    350 https://www.nature.com/articles/sdata201889
    ${ }^{351} \mathrm{https}: / /$ github.com/CxAalto/gtfs_data_pipeline
    352 https://gauss.fh-zwickau.de

[^120]:    ${ }^{353}$ https://github.com/dezhin/osmread

[^121]:    354 https://www.sqlite.org

[^122]:    $356 \mathrm{https}: / /$ python-visualization.github.io/folium/plugins.html\#folium.plugins.FastMarkerCluster
    ${ }^{357}$ https://python-visualization.github.io/folium/plugins.html\#folium.plugins.MarkerCluster

[^123]:    ${ }^{358}$ https://python-visualization.github.io/folium/modules.html\#folium.raster_layers.ImageOverlay

[^124]:    $359 \mathrm{https}: / / \mathrm{www} . d e s t a t i s . d e$
    ${ }^{360} \mathrm{https}: / / \mathrm{www}-$ genesis.destatis.de/genesis//online?operation=table\&code=12613-0006
    ${ }^{361} \mathrm{https}: / / \mathrm{www}-$ genesis.destatis.de/genesis//online?operation=table\&code=12411-0001
    $362 \mathrm{https}: / / \mathrm{www}$. statistischebibliothek.de/mir/servlets/MCRFileNodeServlet/DEMonografie_derivate_00000961/Heft_3.pdf
    $363 \mathrm{https}: / / \mathrm{www}$-genesis.destatis.de/genesis//online?operation=table\&code=12411-0020

[^125]:    ${ }^{364} \mathrm{https}: / /$ portal-chemnitz.opendata.arcgis.com/datasets/baeume

[^126]:    ${ }^{365} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/gallery/pie_and_polar_charts/pie_features.html\#sphx-glr-gallery-pie-and-polar-charts-pie-features-py
    ${ }^{366} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/gallery/lines_bars_and_markers/bar_stacked.html
    367 https://www.libreoffice.org

[^127]:    ${ }^{368}$ https://www.bundesbank.de/de/aufgaben/bargeld/falschgeld/falschgelderkennung/50-euro-europa-serie
    $369 \mathrm{https}: / / e n . w i k i p e d i a . o r g / w i k i / E U R i o n \_c o n s t e l l a t i o n ~$
    370 https://en.wikipedia.org/wiki/Intaglio_(printmaking)
    $371 \mathrm{https}: / / \mathrm{en}$. wikipedia.org/wiki/Steel_engraving
    372 https://www.init-owl.de/
    373 https://www.th-owl.de
    374 https://archive-beta.ics.uci.edu/dataset/267/banknote+authentication
    375 https://archive-beta.ics.uci.edu/
    376 https://jamesmccaffrey.wordpress.com/2020/08/18/in-the-banknote-authentication-dataset-class-0-is-genuine-authentic/

[^128]:    ${ }^{377} \mathrm{https}: / / \mathrm{www} . r e s e a r c h g a t e . n e t / p r o f i l e / E u g e n-G i l l i c h-2 / p u b l i c a t i o n / 266673146 \_B a n k n o t e \_A u t h e n t i c a t i o n / l i n k s / ~$ 5436b8140cf2643ab9887bca/Banknote-Authentication.pdf

[^129]:    ${ }^{378} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html\#sklearn.neighbors.KNeighborsClassifier. predict_proba
    ${ }^{379} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
    ${ }^{380} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html
    $381 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html

[^130]:    ${ }_{382}^{382} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
    $383 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
    ${ }^{384} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.f 1 _score.html
    $385 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html
    $386 \mathrm{https}: / /$ matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.step.html
    $387 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
    ${ }^{388} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.RocCurveDisplay.html\#sklearn.metrics.RocCurveDisplay.from_ predictions
    ${ }^{389} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

[^131]:    ${ }^{390} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
    ${ }^{391} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.contour.html
    $392 \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.contourf.html

[^132]:    ${ }^{393} \mathrm{https}: / /$ matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.contour.html
    ${ }^{394} \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.contourf.html
    $395 \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.contour.html
    $396 \mathrm{https}: / / \mathrm{matplotlib} . o r g /$ stable/api/_as_gen/matplotlib.axes.Axes.contourf.html

[^133]:    397 https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[^134]:    $398 \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

[^135]:    ${ }^{406} \mathrm{https}: / /$ scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[^136]:    ${ }^{407}$ https://www.crummy.com/software/BeautifulSoup/bs4/doc/\#get-text

