Introduction to Machine Learning

Understanding How and Why

Supplementary notes

André Artelt

Version: July 10, 2019

Preface

These supplementary notes are roughly connected to the lecture "Introduction to Machine Learning".
The aim of these notes is to help students getting a deep understanding of the topics discussed in the lecture and exercises. However, we do not claim or guarantee that all topics (from the lecture/exercises) are covered in these notes (since there are minor changes each year). Therefore, the existence of these notes is no excuse for not visiting & attending the lecture and exercises. Furthermore, many topics are covered in far more details than (for just passing the exam) necessary. However, we think that these - more in depth - explanations might be interesting & useful to curious students who want to "dive deeper" into the material.

Machine Learning is applied math. Especially, we need a basic background knowledge of optimization (in particular convex optimization) as well as of probability theory & statistical inference. Because the past has shown that many students do not have the necessary knowledge of mathematics, we decided to put a short but comprehensive overview on (convex) optimization and probability theory & statistical inference in the appendix of these notes. (Convex) optimization is described in appendix A and appendix B is about probability theory and statistical inference.
You can either first work through appendices A and B or you can directly start reading the notes and go to the appendices whenever you are confronted with something you do not know or understand.

Although many smart people have read these notes and searched for mistakes, we are pretty sure that some mistakes have survived. In case that you find a mistake or have any other kind of feedback, please do not hesitate to contact us by writing to intromachlearn@techfak.uni-bielefeld.de. We really appreciate your feedback and hope that these notes are a useful resource for understanding and mastering the introduction to machine learning.

Finally, we want to thank all involved people (in particular Benjamin Paaßen) for proofreading, correcting mistakes as well as useful and inspiring discussions. Writing these notes would not have been possible without them.

André Artelt
Bielefeld, Germany
Spring 2019

Contents

Chapter 0

Notation

1. Vectors are represented by lower case letters with an arrow on top. E.g. $\vec{x}$.
2. Scalar values are denoted by lower case letters. E.g. a.
3. Matrices are represented by upper case bold letters. E.g. $\mathbf{A}$.
4. Sets are denoted by upper case calligraphic letters. E.g. $\mathcal{Y}$.
5. The set of real numbers is denoted by $\mathbb{R}$.
6. The set of positive real numbers is denoted by $\mathbb{R}_+$.
7. The d dimensional real valued vector space is denoted by $\mathbb{R}^d$.
8. Accessing an entry of a vector is denoted by putting the vector into parenthesis and putting a subindex at it. E.g. $(\vec{x})_i$ denotes the i-th entry of the vector $\vec{x}$.
   On the other hand we denote the i-th vector in a collection of vectors with a subindex i. E.g. $\vec{x}_i$.
9. By default, all vectors are column vectors.
10. The identity matrix is represented by $\mathbb{I}$.
11. The determinant of a matrix $\mathbf{A}$ is denoted by $\det(\mathbf{A})$.
12. The set of *symmetric positive semidefinite* matrices in $\mathbb{R}^{d\times d}$ is denoted by $\mathcal{S}^d$.
13. The expression $\mathbf{A} \succeq 0$ states that the matrix $\mathbf{A}$ is *symmetric positive semidefinite*, whereby the expression $\mathbf{B} \succ 0$ states that the matrix $\mathbf{B}$ is *symmetric positive definite*.

14. Accessing an entry of a matrix is denoted by putting the matrix into parentheses and putting a tuple of subindices at it.
E.g. $(\mathbf{A})_{i,j}$ denotes the entry in i-th row and j-th column of the matrix $\mathbf{A}$.
The i-th matrix in a set of matrices is denoted by the subindex i. E.g. $\mathbf{A}_i$

15. The indicator function $\mathbb{1}_c(x)$ is a function which is 1 if x satisfies the condition c and 0 otherwise.
E.g. an indicator for positive numbers is denoted as $\mathbb{1}_{>0}(x)$. If no condition c is specified, we implicitly assume the condition $= 1$ - e.g. $\mathbb{1}(x)$ is 1 if $x = 1$ and 0 otherwise. Furthermore, we might also write the condition into the argument - e.g. $\mathbb{1}(x = 1)$.

16. When summing or multiplying over a set of variables, we often just write $\sum_i$ or $\prod_i$ without completely specifying the value range of the index i. The value range of i should be obvious from the specific context.

17. Random variables are denoted as upper case letters. E.g. X.

18. Probabilities are always denoted by a capital P.
E.g. the probability that a random variable takes on a specific value (denoted as a lower case letter) is denoted as $P(X = x)$ or even simpler as $P(x)$.

19. The expected value of a random variable X is denoted with $\mathbb{E}[X]$. If not explicitly specified, the distribution, over which the expectation is taken, should be clear from the context.

20. The normal distribution is denoted by $\mathcal{N}$.

21. The uniform distribution is denoted by $\mathcal{U}$.

22. The probability mass function over a set of indices is denoted by $\mathcal{P}(\{p_i\})$ where p_i denotes the probability for the i-th index.

23. Estimates of a quantity are denoted by the "hat" symbol on top.
E.g. $\hat{\mu}$ denotes the estimate of μ.

24. The gradient of a function is denoted by using the ∇ symbol.
E.g. the gradient of a function $f(\vec{x})$ with respect to $\vec{x}$ is written as $\nabla_{\vec{x}} f(\vec{x})$.

25. The *Hessian* (second derivative) of a multivariate function $f(\vec{x})$ is denoted by $\boldsymbol{\nabla^2_{\vec{x}} f}$.

26. The 2-norm of a vector is denoted by $\|\cdot\|$ or explicitly by $\|\cdot\|_2$.

27. The 1-norm of a vector is denoted by $|\cdot|$ or explicitly by $|\cdot|_1$.

28. The Frobenius norm of a matrix is denoted by $\|\cdot\|_F$.

29. The domain of a function f is denoted by $\mathcal{D}(f)$.

30. The image of a function f is denoted by $\mathrm{Img}(f)$.

31. The natural logarithm is denoted by log.

Chapter 1

Basic concepts

1.1. Regression

Assume that we have a data set $\mathcal{D} = \{(\vec{x}_i, y_i)\}$, $|\mathcal{D}| = n$ where $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$. We are looking for a function $f : \mathcal{X} \mapsto \mathcal{Y}$ which predicts the output or dependent variable (here $\mathcal{Y}$) for a given input/predictor/regressor/independent variable (here $\mathcal{X}$). Such a function is also called *regression model*. Therefore the problem of finding such a function, for a given data set, is called a *regression problem*.

Note: In contrast to a classification problem 1.2 the set of possible outputs (here $\mathcal{Y}$) is *infinitely large* (usually continuous e.g. $\mathcal{Y} = \mathbb{R}$). This is a subtle but important difference.
Fig. 1.1 illustrates the difference between regression and classification.

Figure 1.1: Regression vs. Classification

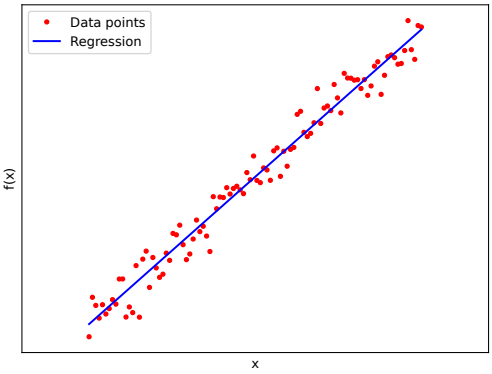


Figure 1.2: Regression

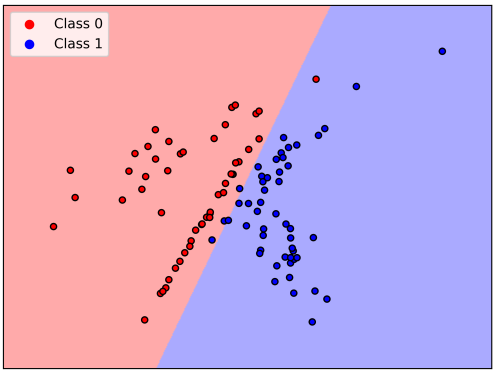


Figure 1.3: Classification

1.2. Classification

1.2.1. The classification problem

Assume that we have a data set $\mathcal{D} = \{(x_i, y_i)\}$, $|\mathcal{D}| = n$ where $x_i \in \mathcal{X}, y_i \in \mathcal{Y}$. A classification problem is a regression problem where $\mathcal{Y}$ is a *finite set*. If $\mathcal{Y} = \{0, 1\}$, we call the problem a *binary classification problem*. In case of classification problems, we call the outputs y_i *labels* or *classes* and we call the regressor $f : \mathcal{X} \mapsto \mathcal{Y}$ a *classifier* - often denoted by h instead of f.
In plain English: We would like to have a model which predicts the output y for a given input x. Therefore we are looking for a function $h : \mathcal{X} \mapsto \mathcal{Y}$ where we define h as

$$h(x) = \begin{cases} 1 & \text{if } x \text{ belongs to class 1} \\ 0 & \text{if } x \text{ belongs to class 0} \end{cases} \tag{1.1}$$

Note: A function like h is also called *classifier*.

1.2.2. Hypothesis

In the previous section we introduced and defined the classification problem. Before looking at particular functions realizing a classifier, which we will start doing in chapter 2, we spend some more time on investigating a classifier from a very formal and abstract point of view. Hence, we do not make any assumptions on the form or implementation of the classifier.

We assume that the items in the training set $\mathcal{D}$ are *samples from a random process and are identically distributed*. We introduce two sequences of random variables: One sequence for the inputs $\{X_i\}$ where $X_i : \mathcal{X} \mapsto \mathcal{X}$ with $X_i(x) = x$ and another sequence for the outputs $\{Y_i\}$ where $Y_i : \mathcal{Y} \mapsto \mathcal{Y}$ with $Y_i(y) = y$. Because we want to predict an output based on an input, we assume that the pairs X_i, Y_i are dependent. The task of classification would not make any sense without a dependency between the input and output.
In plain English: We define for each data point a pair of random variables, X_i and Y_i. Recall that a random variable is a *mapping* from a domain of possible atomic events to a set of possible outcomes for the variable. In our case, we consider the simplest possible random variables, namely identity mappings, that is: X_i maps from X to X with $X_i(x) = x$ and Y_i maps from Y to Y with $Y_i(y) = y$. Note that we assume that Y_i depends on X_i - otherwise, X_i would carry no information about Y_i and predictions would be impossible.

Often we will also assume that the *random process is memoryless*, which means that the random variables are independent of each other - meaning that all X_i are independent of each other and all Y_i are independent of each other. In this case we define two more random variables, belonging to the same random process, for the input and output. We will use them when reasoning about data from the same random process. $X : \mathcal{X} \mapsto \mathcal{X}$ with $X(x) = x$ for the input and $Y : \mathcal{Y} \mapsto \mathcal{Y}$ with $Y(y) = y$ for the output.
If we consider a memoryless random process we can use the two random variables X and Y only and claim that all $(x_i, y_i) \in \mathcal{D}$ are independent realizations of the two random variables. Working with just two random variables will make the math a lot easier.
Throughout this book we will always use this independence assumption - but we will declare it always again, to make sure that we understand/remember that this is an assumption we introduced.

A *hypothesis* (also called *concept*, *classifier* or *model*) is defined as

$$h : \mathcal{X} \mapsto \{0, 1\} \tag{1.2}$$

We can think of it as a classifier telling us whether a given element $x \in \mathcal{X}$ belongs to the concept ($h(x) = 1$) or not.
Furthermore, we define the *version space* as the set of all hypotheses which are consistent with our data $\mathcal{D}$. Thus, we are looking for hypotheses h such that

$$h(x) = y \quad \forall\, (x, y) \in \mathcal{D} \tag{1.3}$$

We define the version space $\mathcal{V}_\mathcal{D}$ as

$$\mathcal{V}_\mathcal{D} = \{h \mid h(x_i) = y_i \ \ \forall\, (x_i, y_i) \in \mathcal{D}\} \tag{1.4}$$

We denote the set of (all) hypotheses as $\mathcal{H}$.

1.2.3. Risk minimization

We define a *loss function* $L : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ as a measure of the quality of a prediction of a given hypothesis h for a given input. That is, we compare the output of the hypothesis $\hat{y} = h(x)$ for a given input $x \in \mathcal{X}$ to the ground truth y and assign a (usually positive) score to this pair $(y, \hat{y})$. A higher score means a "higher" error.
Different loss functions assign different values to $(y, \hat{y})$ pairs. Two common loss functions are

1. the *0-1 loss*:
$$L(y, \hat{y}) = \mathbb{1}(y \neq \hat{y}) \tag{1.5}$$
2. the *squared loss*[1]:
$$L(y, \hat{y}) = (y - \hat{y})^2 \tag{1.6}$$

In order to measure the quality of a hypothesis on a data distribution $p_{X,Y}$, we define the *risk* as

$$R(h) = \mathbb{E}_{(x,y)\sim p_{X,Y}}[L(y, h(x))] \tag{1.7}$$

By plugging the 0-1 loss 1.5 into the risk 1.7, we obtain *0-1 risk* of a hypothesis h as

$$\begin{aligned} R(h) &= \mathbb{E}_{(x,y)\sim p_{X,Y}}[\mathbb{1}(h(x) \neq y)] \\ &= P(h(X) \neq Y) \end{aligned} \tag{1.8}$$

In plain English: The 0-1 risk is the probability of making a mistake - assigning a wrong label to a given instance $x \in \mathcal{X}$. Hence, the 0-1 risk $R(h)$ is often considered as a natural way for assessing the quality of a given classifier h.

For computing the risk 1.7, we need to know the distribution of the data, which in practice might not be the case. In this case we approximate the risk by the *empirical risk* defined as

$$\hat{R}_\mathcal{D}(h) = \frac{1}{n} \sum_i L(y_i, h(x_i)) \tag{1.9}$$

[1] also called *quadratic loss*, *squared distance* or *squared error*

where we assume that we have a data set $\mathcal{D} = \{(x_i, y_i)\}$ of i.i.d. samples from the data distribution $p_{X,Y}$.
In case of the 0-1 loss, the empirical risk becomes

$$\hat{R}_{\mathcal{D}}(h) = \frac{1}{n} \sum_i \mathbb{1}(h(x_i) \neq y_i) \tag{1.10}$$

Because of the law of large numbers (see section B.8), we know that the empirical risk $\hat{R}(h)$ is a consistent estimator of the true risk $R(h)$. Furthermore, Hoeffding's inequality B.7.5 implies that the difference between the risk $R(h)$ and the empirical risk $\hat{R}_{\mathcal{D}}(h)$ decreases exponentially with the number of samples in the data set. In particular we have that

$$P(|\hat{R}_{\mathcal{D}}(h) - R(h)| \geq \epsilon) \leq 2 \exp\left(-2|\mathcal{D}|\epsilon^2\right) \quad \forall\, \epsilon > 0 \tag{1.11}$$

where we made use of the fact that the 0-1 loss is either 0 or 1 - recall that Hoeffding's inequality requires bounded random variables.

Next, we take a look at the set $\mathcal{H}$ containing *all possible* hypotheses. How large is $\mathcal{H}$ (how many elements are in $\mathcal{H}$)? Obviously, the size of $\mathcal{H}$ depends on $\mathcal{X}$. Because for each $x \in \mathcal{X}$ we have exactly two possible outcomes, 1 or 0, it follows that

$$|\mathcal{H}| = 2^{|\mathcal{X}|} \tag{1.12}$$

As one can imagine, $|\mathcal{H}|$ can be come very large - often infinitely large. This is why people often restrict the space of possible hypotheses and only look at this subset of $\mathcal{H}$. Such a restriction is called *inductive bias*. Usually, for practical purposes, people restrict the space of hypotheses (e.g. by only considering hypotheses of a special form) such that searching in this space for a suitable hypothesis h becomes feasible[2].
First, we assume that we are working with a finite set of hypotheses denoted by $\mathcal{H}$[3]. Later, we will introduce some of the basic theory for working with infinitely large hypotheses spaces - although, from a practical point of view nothing will change.

Because we want to make as few mistakes as possible, the common paradigm *risk minimization*, is to choose $h \in \mathcal{H}$ such that the corresponding risk $R(h)$ is minimized.

$$h = \arg\min_{h \in \mathcal{H}} R(h) \tag{1.13}$$

[2]Famous quote: "Learning = searching"

[3]later on we will see many infinitely large $\mathcal{H}$, but we will always make some kind of assumption such that we can search in $\mathcal{H}$ "efficiently"

Because we often do not know the distribution $p_{X,Y}$, we usually minimize the empirical risk instead of the true risk.

$$h = \arg\min_{h \in \mathcal{H}} \hat{R}_{\mathcal{D}}(h) \tag{1.14}$$

By applying the union bound B.7.7 and Hoeffding's inequality B.7.5, we note that the difference between 1.13 and 1.14 decreases exponentially with the number of samples. We can bound the *maximum* difference between the risk and empirical risk as

$$\begin{aligned} P\left(\max_{h \in \mathcal{H}} |\hat{R}_{\mathcal{D}}(h) - R(h)| \geq \epsilon\right) &= P\left(\bigcup_{h \in \mathcal{H}} \left|\hat{R}_{\mathcal{D}}(h) - R(h)\right| \geq \epsilon\right) \\ &\leq \sum_{h \in \mathcal{H}} |\hat{R}_{\mathcal{D}}(h) - R(h)| \\ &\leq 2|\mathcal{H}| \exp\left(-2|\mathcal{D}|\epsilon^2\right) \quad \forall\, \epsilon > 0 \end{aligned} \tag{1.15}$$

In case of an infinitely large hypothesis set, we can not use our bound 1.15 because $|\mathcal{H}| = \infty$. However, we can make some non-trivial statements about the difference between 1.13 and 1.14 in case of $|\mathcal{H}| = \infty$. Because the math to derive this is much more complicated, we only state the result and discard all proofs and derivations.
We define the *growth function*[4] $\text{growth}_{\mathcal{H}} : \mathbb{N} \mapsto \mathbb{N}$ on a hypotheses set $\mathcal{H}$ as the maximum number of distinct labelings of a data set of size $n \in \mathbb{N}$ that can be discriminated by hypotheses in $\mathcal{H}$. In case of binary classification[5], it holds that

$$\text{growth}_{\mathcal{H}}(n) \leq 2^n \tag{1.16}$$

In plain English: The growth function counts the maximum number of different labelings for a given number of data points which can be classified correctly - that is, all training data points would be classified correctly. For this purpose, we do not fix the location of the data points but allow them to be positioned such that we obtain a maximum shattering coefficient (output/result of the growth function).
For instance, consider the set of all discriminating lines in 2d

$$\mathcal{H} = \{\text{sign}(w_1 x + w_2) \mid w_1, w_2 \in \mathbb{R}\} \tag{1.17}$$

[4] also called *shatter coefficient* or *shattering number*

[5] the growth function can be defined for more than two labels - it is possible to use it for regression, too.

In this case, we can classify all binary labelings correctly for $n \leq 3$. Thus,

$$\text{growth}_{\mathcal{H}}(n) = \begin{cases} n & \text{if } n \leq 3 \\ < 2^n & \text{otherwise} \end{cases} \tag{1.18}$$

Next, we define the *VC-dimension* d_{VC} of a hypotheses set $\mathcal{H}$ as the maximum number of data points that can always be shattered. In case of binary classification, the VC-dimension can be formally defined as

$$d_{\text{VC}}(\mathcal{H}) = \max_{n \in \mathbb{N}} n \quad \text{s.t. } \text{growth}_{\mathcal{H}}(n) = 2^n \tag{1.19}$$

Lastly, we have *Sauer's lemma* which gives a non-trivial upper bound on the growth function - if $d_{\text{VC}}(\mathcal{H})$ is finite.

$$\text{growth}_{\mathcal{H}}(n) \leq \sum_{i}^{d_{\text{VC}}} \binom{n}{i} \leq \left(\frac{ne}{d_{\text{VC}}}\right)^{d_{\text{VC}}} \leq \mathcal{O}(n^{d_{\text{VC}}}) \tag{1.20}$$

Now, we are ready to bound the difference between minimizing the risk and empirical risk over an infinitely large hypotheses set. The *Vapnik-Chervonenkis bound*[6] states that

$$P\left(\max_{h \in \mathcal{H}} |\hat{R}_{\mathcal{D}}(h) - R(h)| \geq \epsilon\right) \leq 4\,\text{growth}_{\mathcal{H}}(2|\mathcal{D}|)\,\exp\left(-\frac{|\mathcal{D}|\epsilon^2}{8}\right) \tag{1.21}$$

Furthermore, we can bound the true minimizing risk as

$$\begin{aligned} R(h) &\leq \hat{R}_{\mathcal{D}}(h) + \sqrt{\frac{8\log(\text{growth}_{\mathcal{H}}(2|\mathcal{D}|)) + 8\log(\frac{4}{\delta})}{|\mathcal{D}|}} \\ &\leq \hat{R}_{\mathcal{D}}(h) + \sqrt{\frac{8d_{\text{VC}}\left(\log\left(\frac{2|\mathcal{D}|}{d_{\text{VC}}}\right) + 1\right) + 8\log(\frac{4}{\delta})}{|\mathcal{D}|}} \end{aligned} \tag{1.22}$$

where we set $\delta = 4\,\text{growth}_{\mathcal{H}}(2|\mathcal{D}|)\,\exp\left(-\frac{|\mathcal{D}|\epsilon^2}{8}\right)$ and applied Sauer's lemma 1.20.

1.2.4. Risk minimization and maximum likelihood

It can be shown that, under some mild conditions and for some loss functions - e.g. the 0-1 loss, minimizing the empirical risk is equivalent to maximizing the likelihood[7]. In this case we treat the hypothesis h as a parameter of a

[6] VC - Vapnik-Chervonenkis

[7] see [18]

parametric model - see section B.13.2 for a discussion of maximum likelihood for estimating parameters of a statistical model.
In this book, as well as in practice, we do not care about these "mild conditions" and treat minimizing the empirical risk of the 0-1 loss and maximizing the likelihood as the same thing. Thus

$$h = \arg\min_{h \in \mathcal{H}} \hat{R}_{\mathcal{D}}(h) = \arg\max_{h \in \mathcal{H}} \mathrm{L}_{\mathcal{D}}(h) \tag{1.23}$$

In this context, we define[8] the likelihood as

$$\begin{aligned} \mathrm{L}_{\mathcal{D}}(h) &= \prod_i P\left(y_i = h(x_i)\right) \\ &= \mathbb{1}(y_i = h(x_i)) \end{aligned} \tag{1.24}$$

where we used the assumption that the samples from our data set are *i.i.d.* (independent, identically distributed).

1.2.5. Bayesian model averaging

If we assume that the "true" hypothesis h is not fixed but random, we can use Bayesian inference B.13.3.
From *Bayes' formula* we know that the following holds

$$\begin{aligned} P(h \mid \mathcal{D}) &= \frac{P(\mathcal{D}, h)}{P(\mathcal{D})} \\ &= \frac{P(\mathcal{D} \mid h) P(h)}{P(\mathcal{D})} \end{aligned} \tag{1.25}$$

where $P(h \mid \mathcal{D})$ is called *posterior*, $P(\mathcal{D} \mid h) = \mathrm{L}_{\mathcal{D}}(h)$ is modeled to be the *likelihood*, $P(h)$ is called *prior* and $P(\mathcal{D})$ is called *evidence*.
We define the maximum a posteriori estimator as

$$h = \arg\max_{h \in \mathcal{H}} \mathrm{L}_{\mathcal{D}}(h) P(h) \tag{1.26}$$

Furthermore, we define *Bayesian model averaging* as

$$\begin{aligned} P(x \mid \mathcal{D}) &= \sum_{h \in \mathcal{H}} P(x \mid h) P(h \mid \mathcal{D}) \\ &= \sum_{h \in \mathcal{H}} P(x \mid h) \mathrm{L}_{\mathcal{D}}(h) P(h) \end{aligned} \tag{1.27}$$

[8] in subsequent chapters we will see slightly different definitions of likelihood - e.g. conditional likelihood. Furthermore, we will consider other loss functions than the 0-1 loss - e.g. cross entropy.

where we assume that $\mathcal{H}$ is finite.
In plain English: In Bayesian model averaging we compute the probability that an instance x belongs to a given data set $\mathcal{D}$ - meaning that both $\mathcal{D}$ and x belongs to the same concept. For this purpose, we sum over all hypothesis and compute for each hypothesis whether x belongs to the current hypothesis or not - this will be either 0 or 1, weighted with the likelihood for observing the given data set under the current hypothesis.

1.3. Outlook

In this chapter we learned about regression and classification problems. We introduced hypotheses which can be interpreted as a abstract classifier. In addition, we introduced risk and loss functions and looked at difference between the risk and the empirical risk. We observed that minimizing the true risk becomes approximately the same as minimizing the empirical risk when adding more and more data points to our data set. In this context, we introduced VC-theory and some of its bounds.
However, there is much more to say about hypotheses, risk and loss functions as well as computational/statistical learning theory (which includes the VC-stuff). If you want to know more about these topics, a good starting point might be [10] and [18].

1.4. Exercises

1. Decide for each of the following problems whether it can be seriously modeled as a classification or a regression problem.

   (a) Recognize faces of your friends on a picture.

   (b) Predict the lottery numbers of next week.

   (c) Identify a user of a website by just looking at the keystroke dynamics of this user.

   (d) Determine whether a text contains hate speech or not.

   (e) Predict the steering angle of a self-driving car based on a video stream from the driver's cab.

Chapter 2

Bayes classifier

Recall from section 1.2, that a binary classifier is a function like

$$h(x) = \begin{cases} 1 & \text{if } x \text{ belongs to class 1} \\ 0 & \text{if } x \text{ belongs to class 0} \end{cases} \tag{2.1}$$

How can we decide whether a given x belongs to class 0 or 1? We model this decision with a probability for each class.
Let $P(Y = 1 \mid X = x)$ be the probability for x belonging to class 1 and $P(Y = 0 \mid X = x) = 1 - P(Y = 1 \mid X = x)$ the probability for x belonging to class 0.
Note: Because there are two possibilities only, we can express both in terms of the single probability $P(Y = 1 \mid X = x)$.
With these definitions, we rewrite/redefine 2.1 as

$$h(x) = \begin{cases} 1 & \text{if } P(Y = 1 \mid X = x) > t \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

where t is the *discrimination threshold* and specifies the threshold (confidence-probability) at which we predict class 1. Typically $t = 0.5$ but other choices are possible and valid - see discussion about receiver operating characteristic in section 7.1.2.7.
If we do not know the true distribution, we replace $P(Y = 1 \mid X = x)$ by $\hat{P}(Y = 1 \mid X = x)$ as an estimate/approximation of the true underlying conditional distribution.
Note: Equation 2.2 is also called *Bayes classifier*.

The set of of points, which have a equal probability of belonging to class

zero or class one, is called *decision boundary*.

$$\text{Decision boundary} = \{x \in \mathcal{X} \mid P(Y = 1 \mid X = x) = P(Y = 0 \mid X = x) = t\} \tag{2.3}$$

where t is the discrimination threshold - often the default is $t = 0.5$.

2.1. The optimum Bayes classifier

If we have access to the *true distribution*[1] of the data - assuming that we know $P(Y \mid X)$, we can use this distribution to obtain the *optimum Bayes classifier*. The optimum Bayes classifier is exactly the Bayes classifier 2.2 where we know the conditional distribution $P(Y = 1 \mid X = x)$.
Similar to 2.2 (we simply replace the estimate of the probability with the true one) we define the optimum Bayes classifier as the function

$$h^*(x) = \begin{cases} 1 & \text{if } P(Y = 1 \mid X = x) > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Recall from section 1.2.2 the definition of risk. We call the corresponding risk of the optimum Bayes classifier 2.4 *Bayes risk* and denote it by

$$R^* = R(h^*) = P(h^*(X) \neq Y) \tag{2.5}$$

Furthermore, recall from section 1.2.2 the concept of risk minimization as a "natural" way of assessing the "quality" of a classifier.
As the next theorem states, the Bayes risk is a lower bound on the risk of any other classifier.

Theorem 1 (Optimality of the optimum Bayes classifier)**.** *The optimum Bayes classifier is the classifier with the lowest risk. There can not be any other classifier with a lower risk. Thus*

$$R(h) \geq R(h^*) \quad \forall\, h$$

Proof. We begin[2] by observing that $R(h) \geq R(h^*) \quad \forall\, h$ is equivalent to $R(h) - R(h^*) \geq 0 \quad \forall\, h$. Hence, we are going prove that the difference of the risk of any classifier h and the risk of the optimum Bayes classifier is

[1] in practice we usually do not know the true distribution
[2] this proof is heavily based on [12]

always greater or equal to 0, which implies that $R(h) \geq R(h^*) \quad \forall\, h$.
We can rewrite the risk 1.7 as

$$\begin{aligned} R(h) &= P(h(X) \neq Y) \\ &= \mathbb{E}[\mathbb{1}(h(X) \neq Y)] \\ &= \mathbb{E}[\mathbb{1}(h(X) = 0, Y = 1) + \mathbb{1}(h(X) = 1, Y = 0)] \\ &= \mathbb{E}[\mathbb{1}(h(X) = 0, Y = 1)] + \mathbb{E}[\mathbb{1}(h(X) = 1, Y = 0)] \end{aligned} \tag{2.6}$$

Furthermore, because the prediction of the model and the true label are independent we can write

$$\begin{aligned} \mathbb{E}[\mathbb{1}(h(X) = 0, Y = 0) \mid X] &= P(h(X) = 0 \mid X) P(Y = 0 \mid X) \\ &= \mathbb{1}(h(X) = 0) \mathbb{E}[y = 0 \mid X] \end{aligned} \tag{2.7}$$

where the last equality follows from the fact that the output of the model $h(X)$ does not depend on the true label Y and is eiter 0 or 1. The idea in 2.7 holds for all possible combinations of $h(X)$ and Y (there are 4 different possible combinations).
Next, we apply the law of total expectation to 2.6 and get

$$\begin{aligned} R(h) &= \mathbb{E}[\mathbb{1}(h(X) = 0, Y = 1)] + \mathbb{E}[\mathbb{1}(h(X) = 1, Y = 0)] \\ &= \mathbb{E}[\mathbb{E}[\mathbb{1}(h(X) = 0, Y = 1)] \mid X] + \mathbb{E}[\mathbb{E}[\mathbb{1}(h(X) = 1, Y = 0)] \mid X] \end{aligned} \tag{2.8}$$

By making use of 2.7, we can rewrite 2.8 as

$$\begin{aligned} R(h) &= \mathbb{E}[\mathbb{E}[\mathbb{1}(h(X) = 0, Y = 1)] \mid X] + \mathbb{E}[\mathbb{E}[\mathbb{1}(h(X) = 1, Y = 0)] \mid X] \\ &= \mathbb{E}[\mathbb{1}(h(X) = 0) \mathbb{E}[Y = 1 \mid X] + \mathbb{1}(h(X) = 1) \mathbb{E}[Y = 0 \mid X]] \\ &= \mathbb{E}[\mathbb{1}(h(X) = 0) P(Y = 1 \mid X) + \mathbb{1}(h(X) = 1) P(Y = 0 \mid X)] \end{aligned} \tag{2.9}$$

Next, we use this result and take a look at the expression $R(h) - R(h^*)$.

Recall that our goal is to show that this difference is always ≥ 0.

$$
\begin{aligned}
R(h) - R(h^*) &= \mathbb{E}[\mathbb{1}(h(X) = 0)P(Y = 1 \mid X) + \mathbb{1}(h(X) = 1)P(Y = 0 \mid X) - \\
&\quad \mathbb{1}(h^*(X) = 0)P(Y = 1 \mid X) - \mathbb{1}(h^*(X) = 1)P(Y = 0 \mid X)] \\
&= \mathbb{E}[\mathbb{1}(h(X) = 0)P(Y = 1 \mid X) + \mathbb{1}(h(X) = 1)(1 - P(Y = 1 \mid X)) - \\
&\quad \mathbb{1}(h^*(X) = 0)P(Y = 1 \mid X) - \mathbb{1}(h^*(X) = 1)(1 - P(Y = 1 \mid X))] \\
&= \mathbb{E}[P(Y = 1 \mid X)(\mathbb{1}(h(X) = 0) - \mathbb{1}(h^*(X) = 0)) + \\
&\quad (1 - P(Y = 1 \mid X)(\mathbb{1}(h(x) = 1) - \mathbb{1}(h^*(X) = 1))) \\
&= \mathbb{E}[P(Y = 1 \mid X)(\mathbb{1}(h(X) = 0) - \mathbb{1}(h^*(X) = 0)) + \\
&\quad (1 - P(Y = 1 \mid X))(1 - \mathbb{1}(h(X) = 0) - 1 + \mathbb{1}(h^*(X) = 0))] \\
&= \mathbb{E}[P(Y = 1 \mid X)(\mathbb{1}(h(X) = 0) - \mathbb{1}(h^*(X) = 0)) + \\
&\quad (1 - P(Y = 1 \mid X))(-\mathbb{1}(h(X) = 0) + \mathbb{1}(h^*(X) = 0))] \\
&= \mathbb{E}[P(Y = 1 \mid X)\mathbb{1}(h(X) = 0) - P(y = 1 \mid X)\mathbb{1}(h^*(X) = 0) + \\
&\quad - \mathbb{1}(h(X) = 0) + \mathbb{1}(h^*(X) = 0) + \\
&\quad P(Y = 1 \mid X)(\mathbb{1}(h(X) = 0) - P(Y = 1 \mid X)\mathbb{1}(h^*(X) = 0))] \\
&= \mathbb{E}[P(Y = 1 \mid X)\mathbb{1}(h(X) = 0) + P(Y = 1 \mid X)(\mathbb{1}(h(X) = 0) \\
&\quad - P(Y = 1 \mid X)\mathbb{1}(h^*(X) = 0) - P(Y = 1 \mid X)\mathbb{1}(h^*(X) = 0)) \\
&\quad - \mathbb{1}(h(X) = 0) + \mathbb{1}(h^*(X) = 0)] \\
&= \mathbb{E}[(2\,P(Y = 1 \mid X) - 1)(\mathbb{1}(h(X) = 0) - \mathbb{1}(h^*(X) = 0))] \\
&\geq 0
\end{aligned}
\tag{2.10}
$$

where the last inequality follows from the fact that the second last equation is ≥ 0 if $P(Y = 1 \mid X) > 0.5$ and ≥ 0 if $P(Y = 1 \mid X) <= 0.5$.
In particular, if $P(Y = 1 \mid X) > 0.5$, it follows that $h^*(X) = 1$. Therefore, $\mathbb{1}(h^*(X) = 0) = 0$. No matter what $h(X)$ outputs, the term $2\,P(Y = 1 \mid X) - 1)(\mathbb{1}(h(X) = 0) - \mathbb{1}(h^*(X) = 0))$ is always greater or equal to zero.
On the other hand, if $P(Y = 1 \mid X) \leq 0.5$, it follows that $h^*(X) = 1$. Hence, $\mathbb{1}(h^*(X) = 0) = 1$. Because of $2\,P(Y = 1 \mid X) - 1 \leq 0$, the term $2\,P(Y = 1 \mid X) - 1)(\mathbb{1}(h(X) = 0) - \mathbb{1}(h^*(X) = 0))$ is always greater or equal to zero.
Because this holds for all X and Y, it follows that the expectation over (X, Y) is always greater or equal to zero. Which concludes the proof. □

Note: We can only use/compute the optimum Bayes classifier if we know the true underlying distribution of the data. From a practical point of view one might think that this is useless, since in practice we almost never know the

true distribution - if we would do, we would not need to anything and are done. However, the optimum Bayes classifier turns out to be useful when we want to compare different classifiers on a synthetic dataset, where we know the true distribution. In this case, we can compute the optimum Bayes error 2.6 and obtain a lower bound on the error. Next, we could investigate how well the other classifiers - in particular our own - perform.

2.1.1. Outlook

In the previous section we learned that the optimum Bayes classifier is the "best" classifier (it has the lowest risk). However, we can not use it in practice because we do not know the true conditional distribution $P(Y = 1 \mid X)$. In the next sections/chapters we will look at many "instances" of Bayes classifiers which try to approximate the optimum Bayes classifier by approximating/estimating the needed probability $P(Y = 1 \mid X)$.
We will encounter the concept of likelihood many more times which should no longer be surprising since we know that maximum likelihood and risk minimization are strongly related to each other (under some conditions they are equivalent) and the Bayes classifier minimizes the risk.

2.2. Naive Bayes classifier

So far, we have made no assumption about the input domain $\mathcal{X}$. Now, we assume that $\mathcal{X}$ is some kind of *vector space*[3] (e.g. $\mathcal{X} = \mathbb{R}^d$). The naive Bayes classifier is a multi-class classifier and not restricted to binary classification. Thus, $\mathcal{Y} = \{0, 1, ..., k\}$ for a k-class classification problem.

The *naive Bayes classifier* is a *maximum a posteriori* (also called *MAP*) classifier.
Recall that Bayes formula for computing the posterior probability is given by

$$P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} \tag{2.11}$$

If we ignore $P(X)$, we can write the conditional probability $P(Y = 1 \mid X = \vec{x})$, which we need for the Bayes classifier, as

$$P(Y = 1 \mid X = \vec{x}) \propto P(X = \vec{x} \mid Y = 1)P(Y = 1) \tag{2.12}$$

[3] we need some structure with multiple entries/dimensions/features

Thus, the naive Bayes classfier can be written as

$$h(\vec{x}) = \underset{c \in \mathcal{Y}}{\arg\max}\, P(X = \vec{x} \mid Y = c)P(Y = c) \tag{2.13}$$

Furthermore, the *naive Bayes classifier* assumes that the features/dimensions of the input are independent of each other. Thus, we can rewrite the conditional probability $P(\vec{x} \mid Y = c)$ as

$$P(X = \vec{x} \mid Y = c) = \prod_k P((\vec{x})_k \mid Y = c) \tag{2.14}$$

Because the assumption of independence is naive[4], the classifier is called naive Bayes classifier.
Substituting the independence assumption into 2.13 yields

$$h(\vec{x}) = \underset{c \in \mathcal{Y}}{\arg\max} \prod_k P((\vec{x})_k \mid Y = c)P(Y = c) \tag{2.15}$$

or equivalently

$$h(\vec{x}) = \underset{c \in \mathcal{Y}}{\arg\max} \sum_k \log\Big(P((\vec{x})_k \mid Y = c)\Big) + \log\Big(P(Y = c)\Big) \tag{2.16}$$

By substituting different distribution for $P((\vec{x})_k, \mid y)$, we obtain different instances of the naive Bayes classifier.

2.2.1. Gaussian naive Bayes classifier

The *Gaussian naive Bayes classifier* assumes that

$$x_k^c \sim \mathcal{N}(\mu_k^c, \sigma_k^c) \quad \forall\, k \in \{1, \cdots, d\},\ c \in \mathcal{Y} \tag{2.17}$$

In plain English: All features follow a *class dependent normal distribution*, each specified by its own mean μ_k^c and variance $\sigma^{2^c}{}_k$.
Thus, the classifier 2.13 becomes

$$h(\vec{x}) = \underset{c \in \mathcal{Y}}{\arg\max} \prod_k \mathcal{N}\left((\vec{x})_k \mid \mu_k^c, \sigma^{2^c}{}_k\right) P(Y = c) \tag{2.18}$$

When we have a data set given, we simply estimate the unknown parameters μ_k^c and $\sigma^{2^c}{}_k$ from this given data set.

[4] note that independence might not always hold

From statistics (see appendix B) we know that an unbiased & consistent estimator of μ is given by

$$\hat{\mu}_k = \frac{1}{n} \sum_i (\vec{x}_i)_k \tag{2.19}$$

and an unbiased & consistent estimator of the variance σ^2 is given by

$$\hat{\sigma}_k^2 = \frac{1}{n-1} \sum_i \Big((\vec{x}_i)_k - \hat{\mu}_k \Big)^2 \tag{2.20}$$

Finally, we have the following equations for estimating the unknown model parameter

$$\begin{aligned} n_c &= \sum_i \mathbb{1}(y_i = c) \\ P(y = c) &= \frac{n_c}{n} \\ \mu_k^c &= \frac{1}{n_c} \sum_i \mathbb{1}(y_i = c)(\vec{x}_i)_k \\ {\sigma^2}_k^c &= \frac{1}{n_c - 1} \sum_i \mathbb{1}(y_i = c) \Big((\vec{x}_i)_k - \hat{\mu}_k \Big)^2 \end{aligned} \tag{2.21}$$

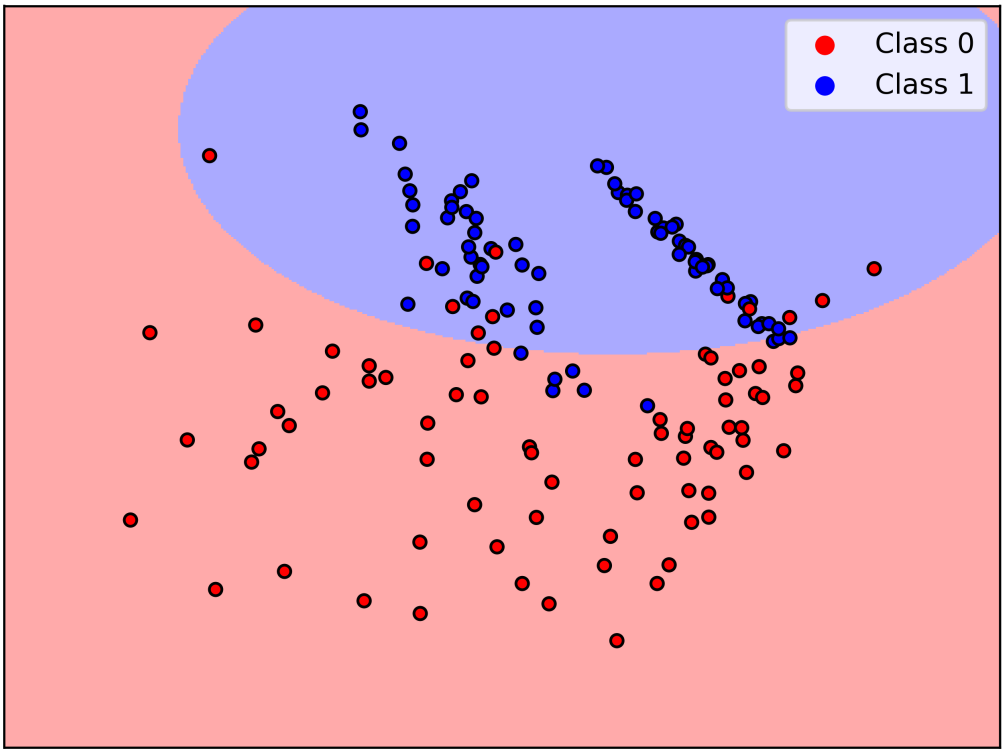


Figure 2.1: Decision boundary of a Gaussian naive Bayes classifier

2.2.2. Generative vs. discriminative models

The joint probability distribution of two random variables X and Y can be factored as

$$p(X, Y) = p(X \mid Y)p(Y) = p(Y \mid X)P(X) \tag{2.22}$$

A model which "learns" or models $p(X, Y)$ is called a *generative model* because it can be used for generating new data by sampling from the joint distribution $p(X, Y)$. An example of a generative model is the naive Bayes classifier.

Note that because of 2.22, every generative model can be (in theory) turned into a discriminiative model - we can obtain $p(Y \mid X)$ from $p(Y, X)$ but not vice versa.

On the other hand, a model which models the conditional distribution $p(Y \mid X)$ only is called *discriminative model*, because it can predict a value given the other value - it learns/models a relationship between the dependent and independent variable. Examples of discriminative models are the naive Bayes

classifier, linear regression, logistic regression and the knn model.

2.3. Exercises

1. We obtain the *Bernoulli naive Bayes classifier*, if we assume that $\vec{x} \in \{0,1\}^d$ is binary vector and the probability $P((\vec{x})_k, \mid y)$ is modeled with the Bernoulli distribution. Thus

$$\begin{aligned} P((\vec{x})_k, \mid y) &= \mathrm{Ber}((\vec{x})_k, p_k) \\ &= p_k^{(\vec{x})_k}(1-p_k)^{1-(\vec{x})_k} \end{aligned} \tag{2.23}$$

   Write down the classification formula/rule $h(\vec{x}) = ...$ of the Bernoulli naive Bayes classifier and derive formulas for estimating p_k using the maximum likelihood approach.

Chapter 3

K-nearest neighbors model

The *k-nearest neighbors* model (short: *KNN*) can be used both for regression (see section 1.1) and classification (see section 1.2). The rough idea behind the model is that when it has to compute a prediction it simply selects the k nearest training data points to the query point and merges their labels/outputs.

We assume that we have a data set $\mathcal{D} = \{(x_i, y_i)\}$ where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. In the binary classification problem we have $\mathcal{Y} = \{0, 1\}$ and in a regression problem we might have $\mathcal{Y} = \mathbb{R}$ (or smth. else which is continuous).

We define the set of the k closest neighbors of a point x as

$$\mathcal{E}(x, \mathcal{D}, k) = \{(x_j, y_j) \in \mathcal{D} \mid x_j \text{ belongs to the } k \text{ closest points around } x\} \tag{3.1}$$

where it is obvious that $k > 0$ is a good idea (otherwise the set is always empty) and "closeness" is measured by an meaningful[1] metric - if $\mathcal{X} = \mathbb{R}^d$ the Euclidean norm is a popular choice.

3.1. K-nearest neighbors classifier

Based on 3.1, we define the conditional class probability[2] as

$$P(Y = c \mid X = x, \mathcal{D}, k) = \frac{1}{k} |\{y_j \mid (x_j, y_j) \in \mathcal{E}(x, \mathcal{D}, k) \wedge y_j = c\}| \tag{3.2}$$

[1] depends on the data & application

[2] for that we introduce two random variables X and Y for the input and output

By making use of 3.2, we can define the KNN classifier as an instance of the Bayes classifier 2.2

$$h(x) = \begin{cases} 1 & \text{if } P(Y = 1 \mid X = x, \mathcal{D}, k) > t \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

where t is the discrimination threshold (often $t = 0.5$).

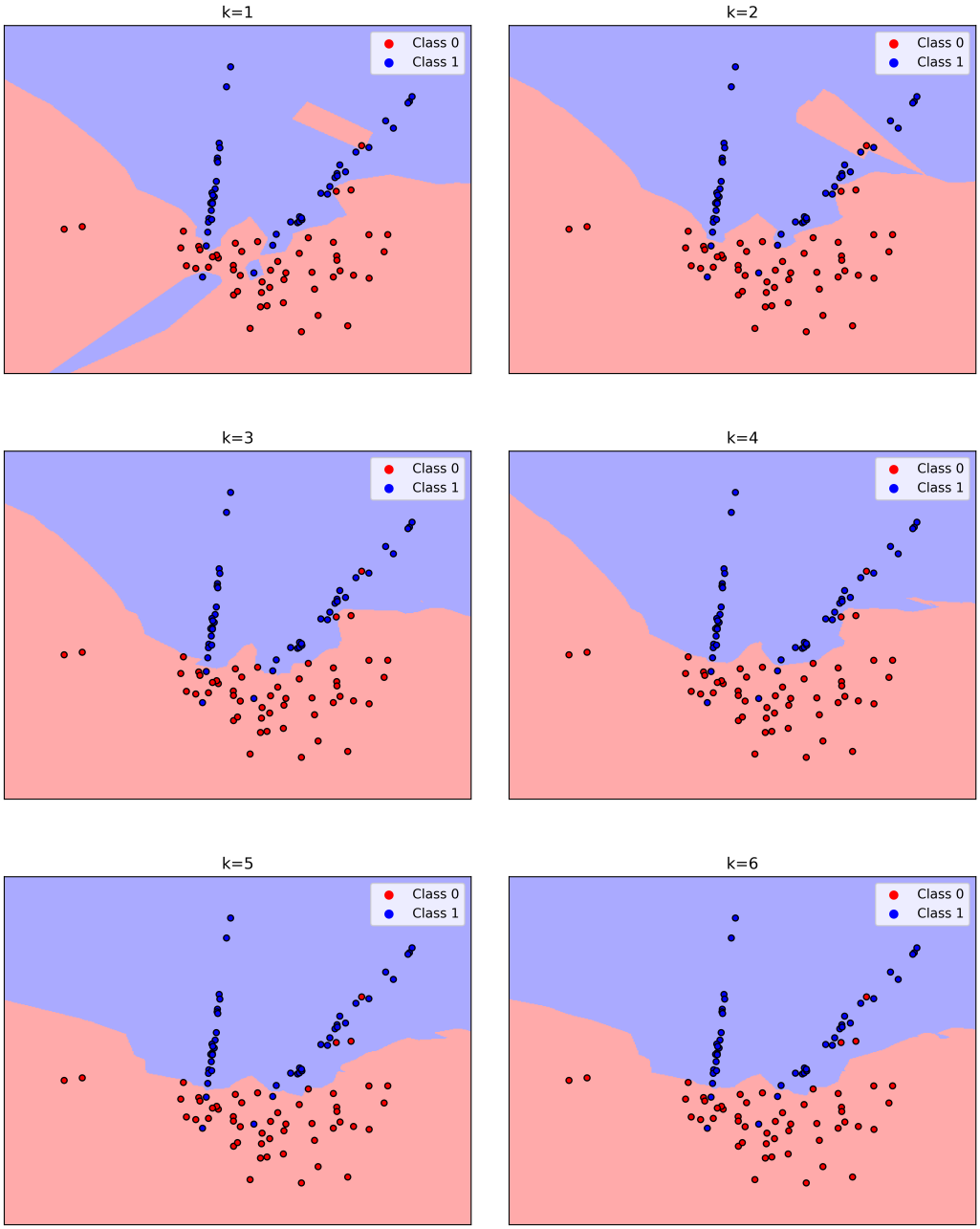
Note: Because we can define the conditional class probability for any label/class we like, we are not restricted to binary classification only. We can use the KNN to build a multi-class classifier by defining

$$h(x) = \arg\max_{c \in \mathcal{Y}} P(Y = c \mid X = x, \mathcal{D}, k) \tag{3.4}$$

Attention: In the beginning of this chapter we said that k is a hyperparameter which we have to tune/optimize. By considering the knn classifier we can identify two extreme values of k which might cause problems.
The first extreme case is $k = 1$. In this case we simply consider the output of the nearest point. On the training set we would get all points correct, since they all match each other in the training set.
The second extreme value is $k = |\mathcal{D}|$. In this case we simply predict the class of the majority.
It is obvious that both cases are not desirable. Thus, we should select a k which is "somewhere"[3] in between.

[3]simply try different values

Figure 3.1: K-nearest neighbors classifier for different values of k



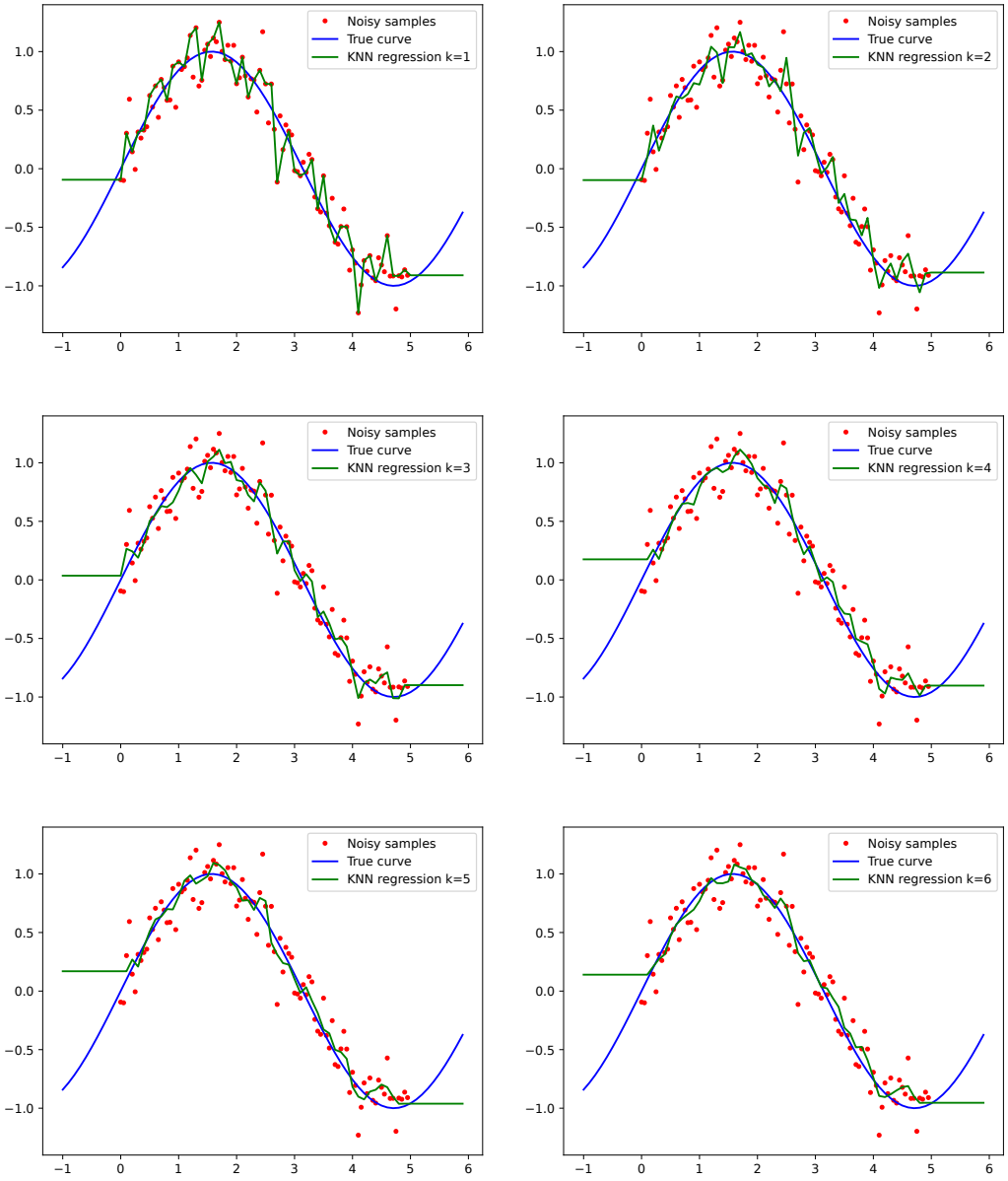
3.2. K-nearest neighbors regression model

As already mentioned, we can use the KNN model for regression too.
The approach is very similar to the one in classification, except that we do not want to count individual outcomes (classes/labels) but it makes more sense to average them.
We obtain the knn-regression model by computing the average of the k-nearest neighbors of a point. Formally

$$f(x) = \frac{1}{k} \sum_{y_j \in \mathcal{E}(x,\mathcal{D},k)} y_j \tag{3.5}$$

Note: If averaging is not satisfying, we could use some other kind of merging the k-nearest neighbors. E.g. weighted average (weights which are proportional to the distance), ...

Figure 3.2: K-nearest neighbors regression for different values of k



3.3. Parametric vs. non-parametric models

The knn model is an example of a *non-parametric model*. The property of a non-parametric model is that it does not try to compress the information from a given data set into a fixed size set of parameters, what is done by *parametric models*, but instead simply stores all data points.
Examples of parametric models are Gaussian naive Bayes, linear regression and logistic regression.

3.4. Outlook

KNN belongs to the family of *prototype-based models*. Other models of this family are *vector quantization models* (e.g. LVQ, GLVQ, GMLVQ, RSLVQ, ...).
The idea is always the same: Compute and store some prototypes. If you have to make a prediction compute the distance or similarity to these prototypes and merge their labels/outputs into a final output.

3.5. Exercises

1. Standardize a data set.
   A data set is *standardized* if each feature has zero mean and unit variance. Thus

$$\begin{aligned} \frac{1}{n}\sum_i (\vec{x}_i)_j &= \mu_j = 0 \\ \frac{1}{n}\sum_i \Big((\vec{x}_i)_j - \mu_j\Big)^2 &= \sigma^2 = 1 \quad \forall\, j \end{aligned} \tag{3.6}$$

   Show that the following transformation transforms a data set into a standardized data set.

$$x'_j = \frac{x_j - \mu_j}{\sigma} \tag{3.7}$$

Chapter 4

Linear regression

In section 1.1 we defined the regression problem and a regression function. When we did so, we did not make any assumptions on the input space $\mathcal{X}$. In this chapter we assume that $\mathcal{X}$ is a real valued vectors space, hence $\mathcal{X} = \mathbb{R}^d$. We generalize linear regression to arbitrary data types at the end of this chapter in section 4.11.

4.1. Modeling

In *linear regression* we define the prediction function $f(\vec{x})$ as

$$f(\vec{x}) = \vec{w}^\top \vec{x} - b \tag{4.1}$$

where $\vec{w} \in \mathbb{R}^d$ is the *weight vector* and b is called *bias/intercept/offset*.
Note: This is called *linear regression* because the output/prediction is a *weighted linear combination* - with weights $\vec{w}$ - of the input features.

Sometimes people do not like the $-b$ in 4.1 and write $+b$ instead of it

$$f(\vec{x}) = \vec{w}^\top \vec{x} + b \tag{4.2}$$

Note: The only difference between 4.1 and 4.2 is $-b$ vs. $+b$. Nevertheless, from a modeling perspective both are equivalent because both model a hyperplane[1] - see Fig. 4.1 for a linear regression example.

In order to be consistent with the majority of literature we will stick to 4.1 - but keep in mind that we can always use 4.2 by simply flipping the sign of the bias b.

[1] a line or plane is a special case/name for a 1d or 2d hyperplane!

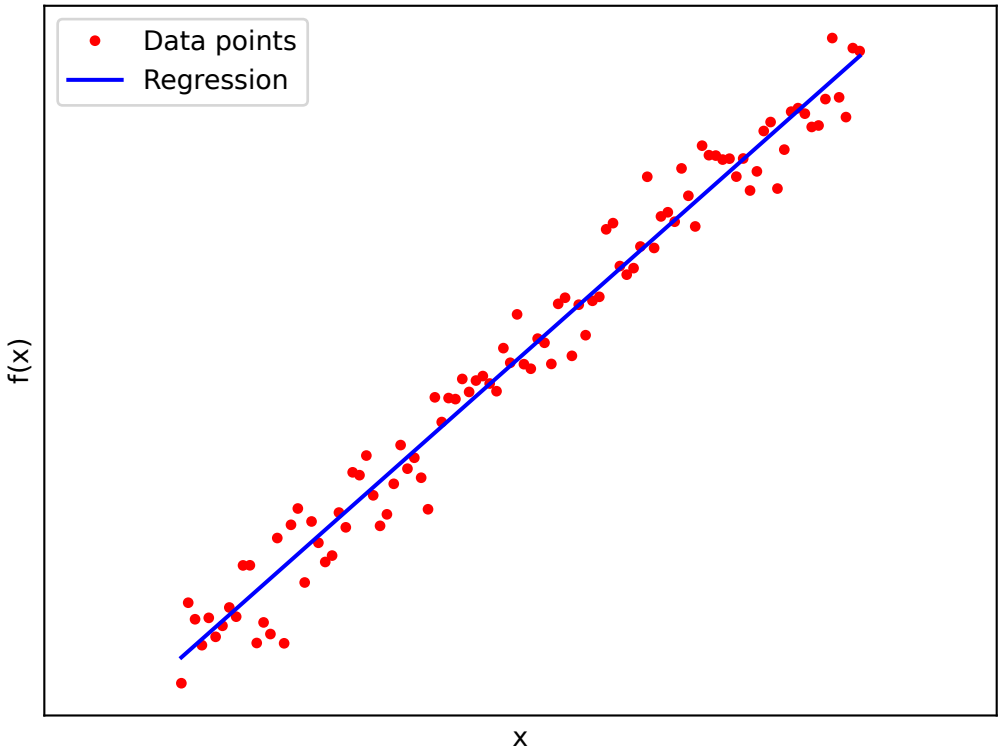


Figure 4.1: Linear regression fitted to a data set

4.1.1. Hidden bias

For the purpose of having a more compact formula, we write 4.1 as a scalar product only and without an explicit bias. We can do so by *integrating the bias b into the weight vector* $\vec{w}$. Hence, 4.1 becomes

$$f(\vec{x}) = \vec{w}_*^\top \vec{x}_* \tag{4.3}$$

where $\vec{w}_* = \begin{pmatrix} b \\ \vec{w} \end{pmatrix}$ and $\vec{x}_* = \begin{pmatrix} -1 \\ \vec{x} \end{pmatrix}$.

Note: We also need to add -1 (or $+1$ for 4.2) to the new input $\vec{x}_*$ to get a valid scalar product.

We can easily convince ourself that 4.3 and 4.1 are equivalent by simply calculating the scalar product:

$$\begin{aligned} f(\vec{x}) &= \vec{w}_*^\top \vec{x}_* \\ &= b(-1) + \vec{w}^\top \vec{x} \\ &= \vec{w}^\top \vec{x} - b \end{aligned} \tag{4.4}$$

Because the bias b is "hidden" inside the scalar product, 4.3 is called *hidden bias*.

From now on we will use the hidden bias notation only. Whenever we talk about a $\vec{w}$, we implicitly assume that $\vec{w}$ contains a hidden bias b without explicitly mentioning it again.

4.2. Cost function

As usual in parameterized models, we need a way to measure how good a given parameter - in this case $\vec{w}$ - is. We use the *Sum of Squared Errors* which is defined as[2]

$$\begin{aligned} \mathrm{SSE}_{\mathcal{D}}(\vec{w}) &= \frac{1}{2}\sum_i (y_i - f(\vec{x}_i))^2 \\ &= \frac{1}{2}\sum_i (y_i - \vec{w}^\top \vec{x}_i)^2 \end{aligned} \tag{4.5}$$

or equivalent in *matrix-vector notation*

$$\mathrm{SSE}_{\mathcal{D}}(\vec{w}) = \frac{1}{2}(\vec{y} - \mathbf{X}\vec{w})^\top (\vec{y} - \mathbf{X}\vec{w}) \tag{4.6}$$

where $\mathbf{X} = \begin{pmatrix} -1 & (\vec{x}_1)_1 & (\vec{x}_1)_2 & \cdots & (\vec{x}_1)_d \\ -1 & (\vec{x}_2)_1 & (\vec{x}_2)_2 & \cdots & (\vec{x}_2)_d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & (\vec{x}_n)_1 & (\vec{x}_n)_2 & \cdots & (\vec{x}_n)_d \end{pmatrix}$ and $\vec{y} = (y_1, y_2, \ldots, y_n)^\top$.

Note: $\mathbf{X}$ is sometimes called the *design-matrix*. Note that the first column contains -1 for the hidden bias.

A *lower SSE is better* because it implies a smaller (squared) difference between $f(\vec{x}_i)$ and y_i. That's why we have to find the $\vec{w}$ which minimizes the SSE. This can be formalized as the following optimization problem

$$\vec{w} = \underset{\vec{w} \in \mathbb{R}^d}{\arg\min}\ \mathrm{SSE}_{\mathcal{D}}(\vec{w}) \tag{4.7}$$

Note: Together 4.5 and 4.7 are also called *Linear Least Squares*.

[2] sometimes the $\frac{1}{2}$ is dropped

4.2.1. Convexity

Theorem 2 (Minimizing the SSE is a convex optimization problem)**.** *The linear least squares problem (as stated in 4.7) is a convex optimization problem.*

Proof. It is sufficient to show that the sum of squared errors (see 4.6) is a convex function.
Because 4.6 is twice differentiable, we can use the second order condition (see A.5) for showing convexity.
Thus,

$$\begin{aligned}\nabla^2_{\vec{w}}\mathrm{SSE}_{\mathcal{D}} &= \nabla^2_{\vec{w}}\frac{1}{2}(\vec{y}-\mathbf{X}\vec{w})^\top(\vec{y}-\mathbf{X}\vec{w}) \\ &= \nabla_{\vec{w}}\frac{1}{2}(\nabla_{\vec{w}}(\vec{y}-\mathbf{X}\vec{w})^\top(\vec{y}-\mathbf{X}\vec{w})) \\ &= \frac{1}{2}\nabla_{\vec{w}}\left(2\mathbf{X}^\top\mathbf{X}\vec{w}-2\mathbf{X}^\top\vec{y}\right) \\ &= \mathbf{X}^\top\mathbf{X} \\ &\succeq 0\end{aligned} \tag{4.8}$$

$\mathbf{X}^\top\mathbf{X} \succeq 0$ holds, because:

1. $\mathbf{X}^\top\mathbf{X}$ is symmetric:

$$\begin{aligned}(\mathbf{X}^\top\mathbf{X})^\top &= \mathbf{X}^\top(\mathbf{X}^\top)^\top \\ &= \mathbf{X}^\top\mathbf{X}\end{aligned} \tag{4.9}$$

2. $\mathbf{X}^\top\mathbf{X}$ is positive semi-definite:

$$\begin{aligned}\vec{v}^\top\mathbf{X}^\top\mathbf{X}\vec{v} &= (\vec{v}^\top\mathbf{X}^\top)(\mathbf{X}\vec{v}) \\ &= (\mathbf{X}\vec{v})^\top(\mathbf{X}\vec{v}) \\ &\geq 0 \quad \forall\,\vec{v}\in\mathbb{R}^d\end{aligned} \tag{4.10}$$

Because $\vec{w}\in\mathbb{R}^d$ and $\mathbb{R}^d$ is known to be a convex set, we can conclude that 4.7 is a convex optimization problem. □

4.3. Optimization

Now that we have a *parameterized model*, a *cost function* and a corresponding *optimization problem* 4.7, we are ready to search for an "optimal" parameter

$\vec{w}$.
Luckily, as stated in 2, it's a *convex optimization problem* so that we know methods for solving it. Moreover, we can choose between an *closed form* and an *iterative solution*.

4.3.1. Closed form solution

We can solve 4.7 by computing the gradient $\nabla_{\vec{w}} \mathrm{SSE}$, setting $\nabla_{\vec{w}} \mathrm{SSE} = \vec{0}$ and finally solve for $\vec{w}$.
First, we compute the gradient $\nabla_{\vec{w}} \mathrm{SSE}$

$$\begin{aligned}
\nabla_{\vec{w}}\mathrm{SSE}_{\mathcal{D}}(\vec{w}) &= \nabla_{\vec{w}} \frac{1}{2}(\vec{y} - \mathbf{X}\vec{w})^\top (\vec{y} - \mathbf{X}\vec{w}) \\
&= \frac{1}{2}\nabla_{\vec{w}} (\mathbf{X}\vec{w} - \vec{y})^\top (\mathbf{X}\vec{w} - \vec{y}) \\
&= \frac{1}{2}\nabla_{\vec{w}} (\vec{w}^\top \mathbf{X}^\top \mathbf{X}\vec{w} - \vec{w}^\top \mathbf{X}^\top \vec{y} - \vec{y}^\top \mathbf{X}\vec{w} + \vec{y}^\top \vec{y}) \\
&= \frac{1}{2}\nabla_{\vec{w}} \vec{w}^\top \mathbf{X}^\top \mathbf{X}\vec{w} - \frac{1}{2}\nabla_{\vec{w}} \vec{w}^\top \mathbf{X}^\top \vec{y} - \frac{1}{2}\nabla_{\vec{w}} \vec{y}^\top \mathbf{X}\vec{w} + \frac{1}{2}\nabla_{\vec{w}} \vec{y}^\top \vec{y} \\
&= \mathbf{X}^\top \mathbf{X}\vec{w} - \frac{1}{2}\mathbf{X}^\top \vec{y} - \frac{1}{2}\mathbf{X}^\top \vec{y} \\
&= \mathbf{X}^\top \mathbf{X}\vec{w} - \mathbf{X}^\top \vec{y}
\end{aligned} \tag{4.11}$$

Next, we set $\nabla_{\vec{w}}\mathrm{SSE}_{\mathcal{D}} = \vec{0}$ and solve for $\vec{w}$

$$\begin{aligned}
\nabla_{\vec{w}}\mathrm{SSE}_{\mathcal{D}}(\vec{w}) &= \vec{0} \\
\mathbf{X}^\top \mathbf{X}\vec{w} - \mathbf{X}^\top \vec{y} &= \vec{0} \\
\mathbf{X}^\top \mathbf{X}\vec{w} &= \mathbf{X}^\top \vec{y} \\
\mathbf{X}^\top \mathbf{X}\vec{w} &= \mathbf{X}^\top \vec{y} \\
\vec{w} &= (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \vec{y}
\end{aligned} \tag{4.12}$$

Finally, we obtain a formula for computing the optimal $\vec{w}$

$$\vec{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \vec{y} \tag{4.13}$$

Note: The expression $(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$ is called *pseudo inverse*[3] of $\mathbf{X}$.
Furthermore, the closed form solution exists if and only if $\mathbf{X}^\top \mathbf{X}$ is invertible.

[3] also called *Moore-Penrose inverse*.
In general, the pseudo inverse is a matrix - usually denoted as $\mathbf{A}^\dagger$ - with some defined properties. The pseudo inverse always exists and is unique.
In the special case that the matrix $\mathbf{X}^\top \mathbf{X}$ is invertible, the pseudo inverse is given by

4.3.2. Iterative solution

Another way to solve the convex optimization problem 4.7 is by using the *gradient descent* algorithm (see section A.3.2). Again we need $\nabla_{\vec{w}}\text{SSE}_{\mathcal{D}}$, but this time we do not use the matrix-vector notation because this method is often used in scenarios where you can not or do not want to construct the matrix $\mathbf{X}$.

$$\begin{aligned}
\nabla_{\vec{w}}\text{SSE}_{\mathcal{D}}(\vec{w}) &= \nabla_{\vec{w}}\frac{1}{2}\sum_i (y_i - f(\vec{x}_i))^2 \\
&= \frac{1}{2}\sum_i \nabla_{\vec{w}}(y_i - f(\vec{x}_i))^2 \\
&= \frac{1}{2}\sum_i 2(y_i - f(\vec{x}_i))\nabla_{\vec{w}}(y_i - f(\vec{x}_i)) \\
&= \frac{1}{2}\sum_i 2(y_i - f(\vec{x}_i))(-\nabla_{\vec{w}}\vec{w}^\top \vec{x}_i) \qquad (4.14) \\
&= \frac{1}{2}\sum_i 2(y_i - f(\vec{x}_i))(-\vec{x}_i) \\
&= -\sum_i (y_i - f(\vec{x}_i))\vec{x}_i \\
&= \sum_i f(\vec{x}_i)\vec{x}_i - y_i\vec{x}_i
\end{aligned}$$

Finally, we use the gradient descent algorithm as described in Algorithm 1 where $\nabla_{\vec{w}}\text{SSE}_{\mathcal{D}}(\vec{w})$ is computed in 4.14 and η is the learning rate (or step size).

Algorithm 1 Linear regression - Gradient descent algorithm

1: $\vec{w} = \vec{0}$ ▷ Initialize $\vec{w}$
2: **repeat**
3: $\quad \vec{w} = \vec{w} - \eta\nabla_{\vec{w}}\text{SSE}_{\mathcal{D}}(\vec{w})$ ▷ Update $\vec{w}$
4: **until** convergence

$\mathbf{X}^\dagger = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$.
In general, the pseudo inverse can be computed by using the singular value decomposition. **A provable fact about the pseudo inverse states that $\vec{w} = \mathbf{X}^\dagger\vec{y}$ is the minimizer of $\text{SSE}_{\mathcal{D}}(\vec{w})$, no matter whether $\mathbf{X}^\top\mathbf{X}$ is invertible or not!**
If you want to learn more about the pseudo inverse, a good starting point is the corresponding wikipedia page `https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse`

4.3.2.1 Optimal step size

As discussed in section A.3.2, we want to choose the set size η such that the resulting SSE is minimized. Hence, we have to solve the following optimization problem

$$\eta = \arg\min_{\eta} \mathrm{SSE}_{\mathcal{D}}(\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}) \tag{4.15}$$

Luckily, 4.15 is a convex optimization problem and a closed form solution exists. We can compute the closed form solution by computing the derivative of 4.15 with respect to η, set it equal to zero and solve for η.

We start by resolving the brackets

$$\begin{aligned}
\mathrm{SSE}_{\mathcal{D}}(\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}) &= (\mathbf{X}(\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}) - \vec{y})^{\top} (\mathbf{X}(\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}) - \vec{y}) \\
&= (\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}})^{\top} \mathbf{X}^{\top} \mathbf{X} (\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}) - (\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}) \mathbf{X}^{\top} \vec{y} - \\
&\quad \vec{y}^{\top} \mathbf{X} (\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}) + \vec{y}^{\top} \vec{y} \\
&= (\vec{w} \mathbf{X}^{\top} \mathbf{X} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X}) (\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}) - \vec{w}^{\top} \mathbf{X}^{\top} \vec{y} + \\
&\quad \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \vec{y} - \vec{y}^{\top} \mathbf{X} \vec{w} + \eta \vec{y}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} + \vec{y}^{\top} \vec{y} \\
&= \vec{w}^{\top} \mathbf{X}^{\top} \mathbf{X} \vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \vec{w} - \eta \vec{w}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} + \\
&\quad \eta^2 \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} - \vec{w}^{\top} \mathbf{X}^{\top} \vec{y} + \\
&\quad \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \vec{y} - \vec{y}^{\top} \mathbf{X} \vec{w} + \eta \vec{y}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} + \vec{y}^{\top} \vec{y}
\end{aligned}$$

Next, we compute the derivative of 4.15 with respect to η

$$\begin{aligned}
&\frac{\partial}{\partial \eta} \mathrm{SSE}_{\mathcal{D}}(\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}) \\
=&\frac{\partial}{\partial \eta} (\vec{w}^{\top} \mathbf{X}^{\top} \mathbf{X} \vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \vec{w} - \eta \vec{w}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} + \eta^2 \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} - \\
&\vec{w}^{\top} \mathbf{X}^{\top} \vec{y} + \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \vec{y} - \vec{y}^{\top} \mathbf{X} \vec{w} + \eta \vec{y}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} + \vec{y}^{\top} \vec{y}) \\
=&\nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \vec{w} - \vec{w}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} + 2\eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} + \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \vec{y} + \\
&\vec{y}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} \\
=& -2 \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \vec{w} + 2\eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} + 2 \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \vec{y}
\end{aligned} \tag{4.16}$$

Setting the derivative 4.16 equal to zero yields

$$\begin{aligned}
&\frac{\partial}{\partial \eta} \mathrm{SSE}_{\mathcal{D}}(\vec{w} - \eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}) = 0 \\
&\Leftrightarrow \\
&-2 \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \vec{w} + 2\eta \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}} + 2 \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \vec{y} = 0 \\
&\Leftrightarrow \\
&\eta = \frac{\nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \vec{w} - \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \vec{y}}{\nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}} \\
&= \frac{\nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} (\mathbf{X}^{\top} \mathbf{X} \vec{w} - \mathbf{X}^{\top} \vec{y})}{\nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}} \\
&= \frac{\nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}}{\nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}{}^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}}
\end{aligned} \tag{4.17}$$

By making use of the optimal step size 4.17, we can rewrite the update step in the gradient descent algorithm 1 for minimizing the SSE as

$$\vec{w}_{t+1} = \vec{w}_t - \frac{\nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}(\vec{w}_t)^{\top} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}(\vec{w}_t)}{\nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}(\vec{w}_t)^{\top} \mathbf{X}^{\top} \mathbf{X} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}(\vec{w}_t)} \nabla_{\vec{w}} \mathrm{SSE}_{\mathcal{D}}(\vec{w}_t) \tag{4.18}$$

4.4. Feature transformation

Linear regression is easy to use - it is fast and we can fit it easily - but it's always a linear function (hyperplane) only. Therefore it does not seem to be suitable for modeling non-linear relationships, which we can see in Fig. 4.2 (in the upper left plot we can see that, in this example, a line is obviously not the best choice).
In order to give *linear regression* the ability to *model non-linearities*, we introduce smth. called *feature transformations*.
The idea is that if we can not model the data by a hyperplane (e.g. a straight line) in its original space maybe we can *transform the data* nonlinearly - e.g. projecting it into a higher dimensional space - *such that it can be better described by a hyperplane*. We transform each data point $\vec{x}$ by a function $\phi : \mathcal{X} \mapsto \mathcal{X}'$, where $\mathcal{X}$ is the original data space and $\mathcal{X}'$ is the new data space - $\mathcal{X}$ and $\mathcal{X}'$ can be equal but usually they are not. Hence, $\vec{x}$ becomes $\phi(\vec{x})$.
We can approximate "everything" by using the "right" feature transformation.

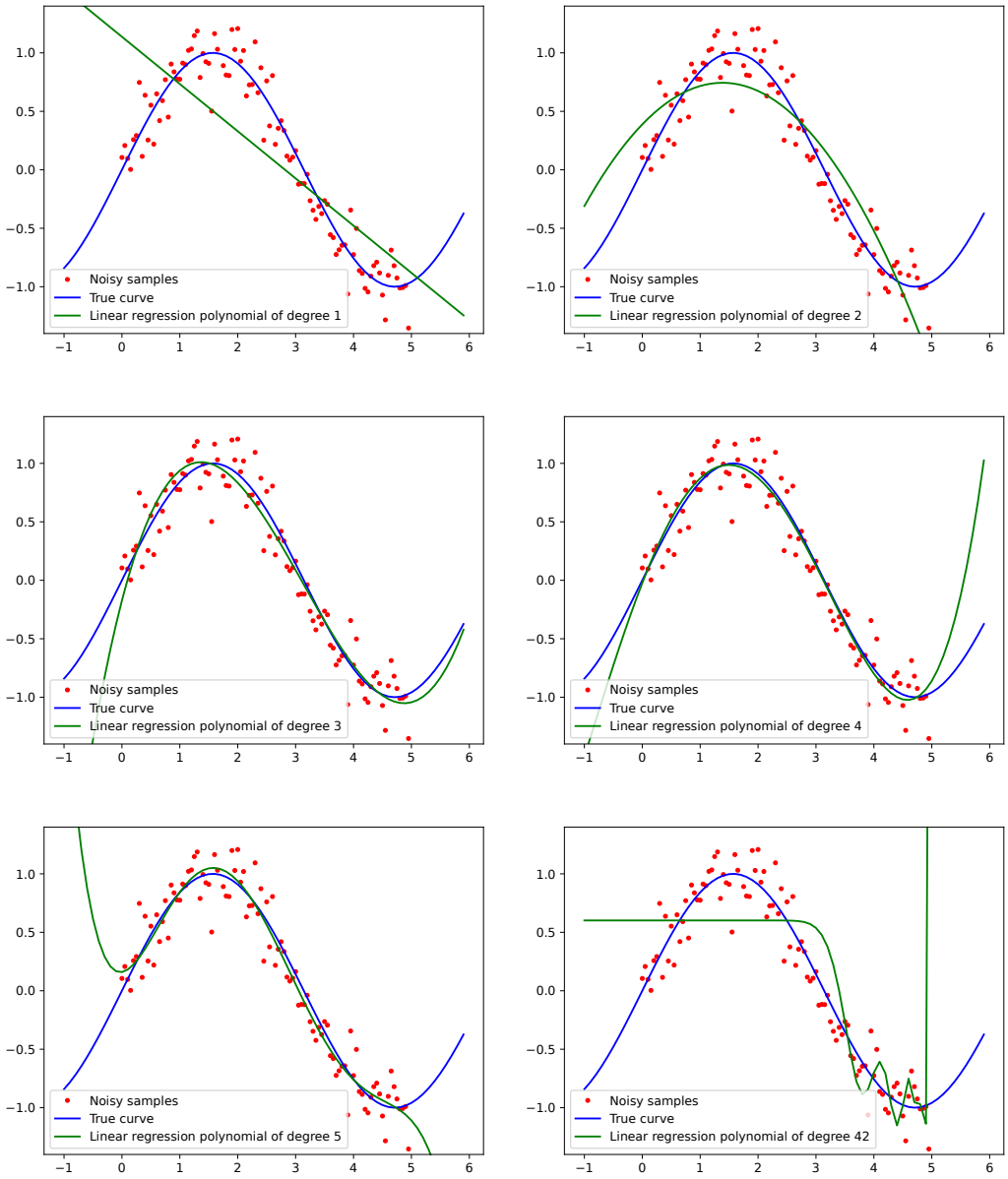
Note: Because we modified the data (all $\vec{x}$) only, the *model itself remains*

the same. The procedure of fitting and computing predictions is still the same except that we have to replace all $\vec{x}$ by $\phi(\vec{x})$.

The remaining question is, how a "typical" ϕ looks like and how to choose a suitable one.
After the good news - by using a suitable ϕ we can use linear regression for non-linear problems - now comes the bad news. *There is no unique or "the best" ϕ in general* and no "general" way of finding it. Usually one tries and evaluates different ϕ and finally pick the best one.
A *common choice for ϕ is a polynomial*. E.g. if $\vec{x} = (x_1, x_2)^\top \in \mathbb{R}^2$ and we choose the degree of the polynomial to be 2, we can write $\phi(\vec{x}) = (x_1,\, x_2,\, x_1x_2,\, x_1^2,\, x_2^2, 1)^\top$. Note that the 2 dimensional data point $\vec{x}$ is transformed into a 6 (+ 1 for the bias) dimensional data point!.
An example of linear regression with a polynomial feature transformation (with different degrees) is shown in Fig. 4.2

For the remainder of this book, we will not consider any feature transformations, but we keep in mind that we can always introduce/add a feature transformation by simply replacing all $\vec{x}$ by $\phi(\vec{x})$.

Figure 4.2: Linear regression with polynomial feature transformation



4.5. Regularization

For the sake of *stability* and in order to *avoid overfitting*[4], we usually prefer weights $\vec{w}$ with a "small" norm/length (equivalent to "small" entries).
For this reason we add a penalty for large weights to our cost function and then obtain a new SSE given by

$$\text{SSE}_{\mathcal{D}}(\vec{w}) = \frac{1}{2}\sum_i (y_i - f(\vec{x}_i))^2 + \lambda \|\vec{w}\|_2^2 \tag{4.19}$$

where $\lambda > 0$ is called *regularization strength*. With λ we can control how important we think small weights - *small weights result in a flatter*[5] *curve* - are in contrast to a small SSE (*small error vs. small weights*). See Fig. 4.3 for an illustration of different regularization strengths.
As we did before, we can rewrite the new cost function 4.19 in matrix-vector notation as

$$\text{SSE}_{\mathcal{D}}(\vec{w}) = \frac{1}{2}(\vec{y} - \mathbf{X}\vec{w})^\top (\vec{y} - \mathbf{X}\vec{w}) + \lambda \vec{w}^\top \vec{w}. \tag{4.20}$$

Note: This type of regularization is also called L2, ℓ_2 or *Tikhonov*[6] regularization. The resulting linear regression is also called *ridge regression*.

To find such a $\vec{w}$ we have solve the optimization problem 4.7 again, but this time with our new SSE (because we want to include the regularization).

4.5.0.1 Convexity

Theorem 3 (Minimizing the L2 regularized SSE is a convex optimization problem)**.** *Minimizing the regularized* SSE *(as stated in 4.19) is a convex optimization problem.*

Proof. See exercise 8. □

4.5.1. Closed form solution

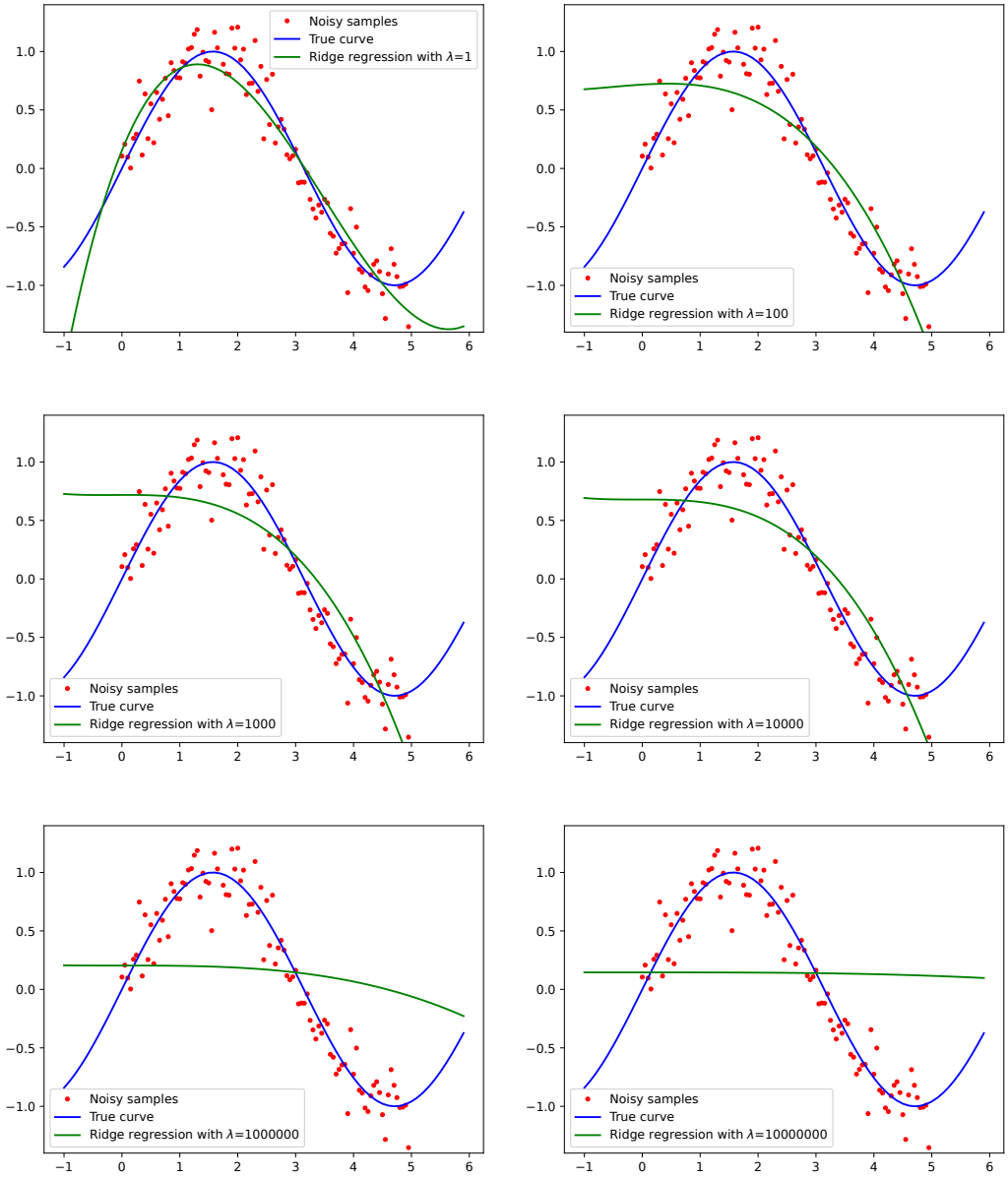
Again we have to compute the gradient $\nabla_{\vec{w}} \text{SSE}_{\mathcal{D}}(\vec{w})$, set $\nabla_{\vec{w}} \text{SSE}_{\mathcal{D}}(\vec{w}) = \vec{0}$ and finally solve for $\vec{w}$.

[4] overfitting means that we make few errors on training data and many errors on new data - see section 7.2.3

[5] flatter = lower curvature

[6] to be precise: This is just a special of case the general Tikhonov regularization - see `https://en.wikipedia.org/wiki/Tikhonov_regularization`

Figure 4.3: Linear regression with L2 regularization



The derivation is similar to the previous case without regularization and is left as an exercise (see exercise 9).
The final formula for the optimal $\vec{w}$ is given by

$$\vec{w} = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbb{I})^{-1} \mathbf{X}^\top \vec{y} \tag{4.21}$$

where $\mathbb{I}$ is the identity matrix with suitable dimension. Because of $\lambda > 0$, the matrix in 4.21 is always invertible - see exercise 2.

4.5.2. Iterative solution

Like in the previous section we need to compute $\nabla_{\vec{w}} \text{SSE}_{\mathcal{D}}(\vec{w})$. Again we do not use the matrix-vector notation and leave the detailed derivation as an exercise.
The final result is

$$\nabla_{\vec{w}} \text{SSE}_{\mathcal{D}}(\vec{w}) = -\sum_i (y_i - f(\vec{x}_i))\vec{x}_i + \lambda \vec{w} \tag{4.22}$$

Because of convexity (see theorem 3) we can use the gradient descent algorithm (see section A.3.2) for computing the solution iteratively - analogous to the previous case without regularization.

4.6. Probabilistic interpretation

4.6.1. Noisy outputs

So far we have modeled the relation between input and output by a deterministic function and minimized the resulting error. Next, we will take a probabilistic perspective on the same problem and assume that the targets have been generated by a linear model with the addition of Gaussian noise:

$$y = f(x) + \varepsilon = \vec{w}^\top \vec{x} + \varepsilon \tag{4.23}$$

where $\varepsilon \sim \mathcal{N}(\mu = 0, \sigma^2)$.
In plain English: The additive noise ε is distributed according to a normal distribution with mean 0 and variance σ^2.

Because of the random noise, we switch to a stochastic model and define pairs of random variables - one pair for each data point. The random variable for the input $X : \mathcal{X} \mapsto \mathcal{X}$ with $X(x) = x$ and a random variable for the output $Y : \mathcal{Y} \mapsto \mathcal{Y}$ with $Y(y) = y$. We assume that that the data points in

$\mathcal{D}$ are i.i.d. samples[7] from a random process described by 4.23.

By using 4.23, we can specify a probability distribution for the continuous output y given an input $\vec{x}$

$$p(Y = y \mid X = \vec{x}, \vec{w}) = \mathcal{N}(y \mid \vec{w}^\top \vec{x}, \sigma^2) \tag{4.24}$$

See Fig. 4.4 for an illustration.

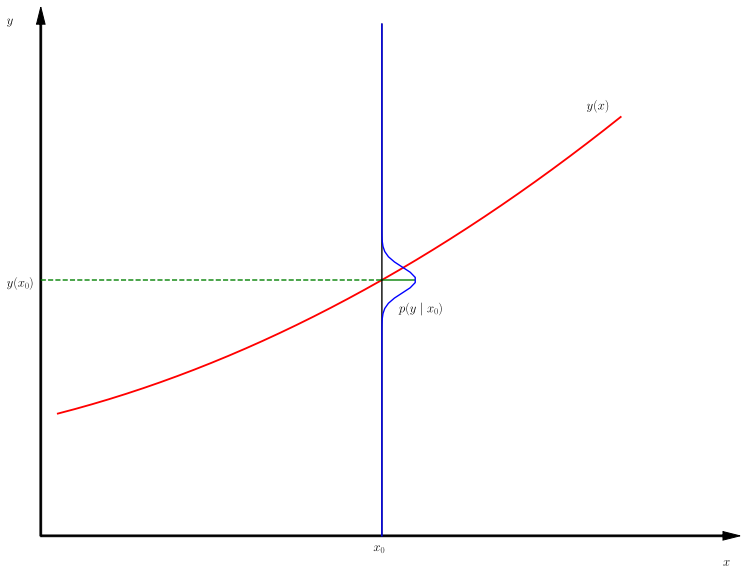


Figure 4.4: Noisy output

Together with 4.24 and under the *assumption* that the *data points are independent* we can define the *(conditional) likelihood* $\mathrm{L}_{\mathcal{D}}(\vec{w})$ (see section B.13.2) as

$$\begin{aligned} \mathrm{L}_{\mathcal{D}}(\vec{w}) &= \prod_i p(y_i \mid \vec{x}_i, \vec{w}) \\ &= \prod_i \mathcal{N}(y_i \mid \vec{w}^\top \vec{x}_i, \sigma^2) \end{aligned} \tag{4.25}$$

In order to make the likelihood numerically stable and to simplify the formula

[7] we assume that the pairs are independent of each other, not x_i and y_i!

we use the *log-likelihood* ($\mathrm{LL}_{\mathcal{D}}(\vec{w})$)

$$\begin{aligned}
\mathrm{LL}_{\mathcal{D}}(\vec{w}) &= \log\left(\prod_i p(y_i \mid \vec{x}_i,\, \vec{w})\right) \\
&= \sum_i \log\left(\mathcal{N}(y_i \mid \vec{w}^\top \vec{x}_i,\, \sigma^2)\right) \\
&= \sum_i \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) \exp\left(-\frac{(y_i - \vec{w}^\top \vec{x_i})^2}{2\sigma^2}\right) \\
&= \sum_i -\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y_i - \vec{w}^\top \vec{x_i})^2 \\
&= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_i (y_i - \vec{w}^\top \vec{x_i})^2 \\
&= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}\mathrm{SSE}_{\mathcal{D}}(\vec{w})
\end{aligned} \tag{4.26}$$

where the number of training data points is denoted by $|\mathcal{D}| = n$.

4.6.2. Maximum likelihood

Next, we use the *maximum likelihood* approach (see section B.13.2) to estimate the parameter $\vec{w}$ using the data set $\mathcal{D}$.

$$\vec{w} = \underset{\vec{w} \in \mathbb{R}^d}{\arg\max}\ \mathrm{LL}_{\mathcal{D}}(\vec{w}) \tag{4.27}$$

By looking at 4.26, we can see that the log-likelihood is equivalent (up to some constants) to the SSE. Therefore *maximizing the log-likelihood is equivalent to minimizing the SSE* - which we did in the previous sections. Hence

$$\vec{w} = \underset{\vec{w} \in \mathbb{R}^d}{\arg\max}\, \mathrm{LL}_{\mathcal{D}}(\vec{w}) = \underset{\vec{w} \in \mathbb{R}^d}{\arg\min}\, \mathrm{SSE}_{\mathcal{D}}(\vec{w}) \tag{4.28}$$

4.6.3. Maximum a posteriori

If we want to add an assumption on $\vec{w}$ to our probabilistic model, we can do so by switching to Bayesian inference and defining a prior/distribution of $\vec{w}$. We then use *maximum a posteriori* (see section B.13.3) to estimate $\vec{w}$.
As explained in section 4.5 about *regularization*, we would like to prefer small weights and therefore "arbitrarily" define

$$p(\vec{w}) = \prod_j \mathcal{N}\left((\vec{w})_j \mid 0,\, \tau^{-1}\right) \tag{4.29}$$

Then, the posterior distribution of $\vec{w}$ is given by[8]

$$p(\vec{w} \mid \mathcal{D}) \propto \mathrm{L}_{\mathcal{D}}(\vec{w})p(\vec{w}) \tag{4.30}$$

Again to make life easier, we take the logarithm of the posterior 4.30.

$$\begin{aligned}
\log\left(p(\vec{w} \mid \mathcal{D})\right) &\sim \log\Big(\mathrm{L}_{\mathcal{D}}(\vec{w})p(\vec{w})\Big) \\
&= \log\Big(\mathrm{L}_{\mathcal{D}}(\vec{w})\Big) + \log\Big(p(\vec{w})\Big) \\
&= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}\mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \log\Big(\prod_j \mathcal{N}\big((\vec{w})_j \mid 0,\, \tau^{-1}\big)\Big) \\
&= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}\mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \sum_j \log\Big(\mathcal{N}\big((\vec{w})_j \mid 0,\, \tau^{-1}\big)\Big) \\
&= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}\mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \sum_j \left(-\frac{1}{2}\log(2\pi\tau^{-1}) - \frac{1}{2\tau^{-1}}\big((\vec{w})_j - 0\big)^2\right) \\
&= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}\mathrm{SSE}_{\mathcal{D}}(\vec{w}) - \frac{d}{2}\log(2\pi\tau^{-1}) - \frac{1}{2\tau^{-1}}\sum_j (\vec{w})_j^2 \\
&= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}\mathrm{SSE}_{\mathcal{D}}(\vec{w}) - \frac{d}{2}\log(2\pi\tau^{-1}) - \frac{\tau}{2}\vec{w}^\top\vec{w} \\
&= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{\sigma^2}\mathrm{SSE}_{\mathcal{D}}(\vec{w}) - \frac{d}{2}\log(2\pi\tau^{-1}) - \frac{\tau}{2}\|\vec{w}\|_2^2 \\
&= -\frac{N}{2}\log(2\pi\sigma^2) - \frac{d}{2}\log(2\pi\tau^{-1}) - \frac{1}{\sigma^2}\Big(\mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \sigma^2\tau\|\vec{w}\|_2^2\Big)
\end{aligned} \tag{4.31}$$

By looking at 4.31, we notice that the posterior is equivalent (again up to some constants) to the SSE with L2/Tikhonov regularization. Therefore, we conclude that *MAP with Gaussian prior is equivalent to minimizing the SSE + L2 regularization* where the regularization strength λ is now $\sigma^2\tau$. Hence

$$\vec{w} = \underset{\vec{w}\in\mathbb{R}^d}{\arg\max}\ \log p(\vec{w} \mid \mathcal{D}) = \underset{\vec{w}\in\mathbb{R}^d}{\arg\min}\ \mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \lambda\|\vec{w}\|_2^2 \tag{4.32}$$

4.7. Robust regression

In reality the *data itself can contain errors*, in the sense that some data points might be wrong - e.g. due to measurement errors. Such "wrong" data points are called *outliers*.

[8]Note that we dropped the normalizing constant

In Fig. 4.6 we can see a data set which seems to follow a linear relation except some points - 3 points are far away from the other points (e.g. its neighbors). If we fit *linear regression* to this data set, we observe that the model *does not capture/model the linear relation* as a human being would easily do. The model is "pulled down" or heavily influenced by the outliers.
In plain English: Standard linear regression is sensitive to outliers.

To solve this issue one would either *remove all outliers* from the data set or *use* a *method* which is more *robust to outliers*[9].

4.7.1. Huber regression

Huber regression is an example of a regression model which is robust to outliers. Huber regression is a linear model (viz. $f(\vec{x}) = \vec{w}^\top \vec{x}$) and minimizes the following cost function

$$\sum_i L_H \left(f(\vec{x}_i) - y_i, \delta \right) \quad \text{where } \delta > 0 \tag{4.33}$$

where L_H is defined as

$$L_H(r, \delta) = \begin{cases} \frac{r^2}{2} & \text{if } |r| \leq \delta \\ \delta |r| - \frac{\delta^2}{2} & \text{otherwise} \end{cases} \tag{4.34}$$

The cost function is a mixture between L1 and L2 norm. It uses L2 for "small" residuals and L1, which is more robust to outliers because of the missing square, for "large" residuals. By varying δ we can control what the model interprets as a "small" and what as a "large" residual. See Fig. 4.5 for an illustration and comparision with other loss functions.
The cost function 4.34 can be optimized by a gradient based iterative algorithm (see section. 4.7.1.1).

In Fig. 4.6 we can see an example of Huber regression fitted to a data set containing outliers.
Warning: The example shown in Fig. 4.6 is a toy example for educational purposes. In reality it might not be that easy to identify outliers!
If we say that some points are outliers, *we introduce an assumption/bias about the data* - e.g. "We assume that the data can be described by a linear relation, consequently all data points for which this is not true must be outliers". We need to understand that this is *our assumption* and it *might*

[9] Often one would do both: remove the "obvious outliers" and use a "robust" model.

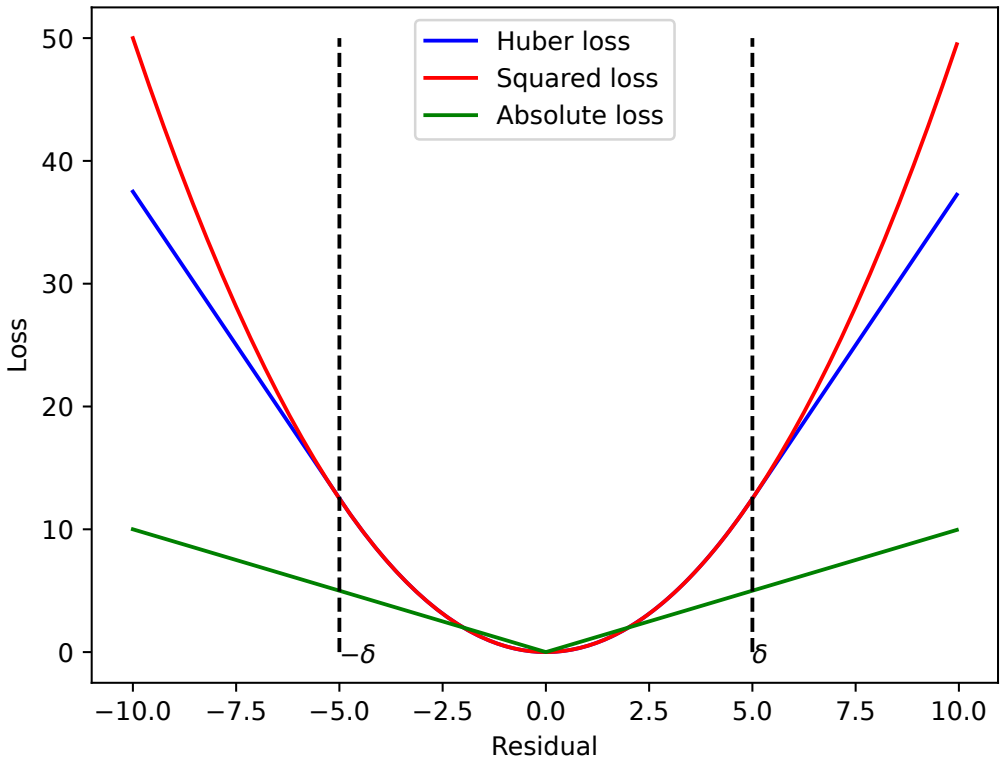


Figure 4.5: Different loss functions

be correct/valid or not. Whenever we do this, we should think very carefully about this.
If we "have" (detected) *too many outliers* in our data set, it might be either the case that the *data has a very bad quality or that our assumption is wrong. Think twice and carefully before declaring points as outliers!*

4.7.1.1 Details on Huber regression

Huber regression can be stated as the following optimization problem

$$\begin{aligned} &\min_{\vec{w} \in \mathbb{R}^d} \sum_i L_H(r_i, \delta) \\ &\text{where } r_i = \vec{w}^\top \vec{x}_i - y_i \quad \text{and } \delta > 0 \end{aligned} \tag{4.35}$$

Theorem 4 (Convexity of Huber regression)**.** *Huber regression as stated in 4.35 is a convex optimization problem.*

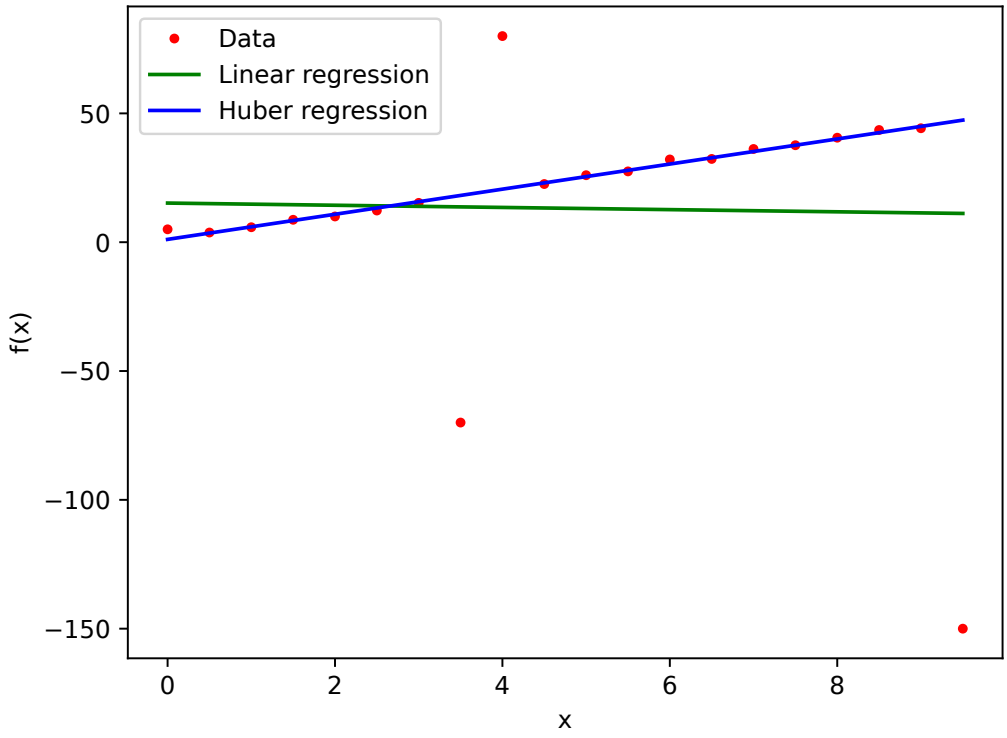


Figure 4.6: Huber vs. Linear regression on a data set with outliers

Proof. Because of $\vec{w} \in \mathbb{R}^d$ and the fact that $\mathbb{R}^d$ is a convex set, we only have to show that $\sum_i L_H(r_i, \delta)$ is a convex function in $\vec{w}$.
Because the sum of convex functions is a convex function, it is sufficient to show that $L_H(r_i, \delta)$ is a convex function in $\vec{w}$.
Because $r_i = y_i - \vec{w}^\top \vec{x}_i$ is a convex function in $\vec{w}$, we can conclude that $L_H(r_i, \delta)$ is a convex function if $L_H(r_i, \delta)$ is a non-decreasing convex function in $|r_i|$ (see 3). Obviously, $L_H(r_i, \delta)$ is non-decreasing (see Fig. 4.6 and definition 4.34), therefore we only have to prove convexity of $L_H(r_i, \delta)$ in $|r_i|$.
Because $L_H(r_i, \delta)$ is piece-wise defined function, we first have to check convexity of each piece separately and then check the derivative at the transition.

1. $f(x) = \frac{1}{2}x^2$ is a convex function because the second-order criterion is satisfied ($\frac{\partial}{\partial x^2} f(x) = 1 \geq 0$).

2. $g(x) = \delta(|x| - \frac{1}{2}\delta) = |x|\delta - \frac{1}{2}\delta^2$ is a convex function because $|x|$ is a convex function and $\delta > 0$.

3. The derivatives of f and g are $\frac{\partial}{\partial x}f(x) = x$ and $\frac{\partial}{\partial x}g(x) = \text{sign}(x)\delta$. Because the transition happens at $|x| = \delta$, we have to check the points $x_1 = \delta$ and $x_2 = -\delta$.

   (a) x_1: $\frac{\partial f}{\partial x}(x_1) = \delta$ and $\frac{\partial g}{\partial x}(x_1) = \delta$.

   (b) x_2: $\frac{\partial f}{\partial x}(x_2) = -\delta$ and $\frac{\partial g}{\partial x}(x_2) = -\delta$.

We conclude that 4.35 is a convex optimization problem. □

4.7.1.1.1 Optimization

From theorem 4, we know that huber regression is a convex optimization problem. Therefore we can use one of the iterative solvers from section A.3.

First, we note that

$$\begin{aligned} \nabla_{\vec{w}} r_i &= \nabla_{\vec{w}} \big(\vec{w}^\top \vec{x}_i - y_i\big) \\ &= \vec{x}_i \end{aligned} \tag{4.36}$$

and

$$\begin{aligned} \nabla_{\vec{w}} |r_i| &= \text{sign}(r_i) \nabla_{\vec{w}} r_i \\ &= \text{sign}(r_i) \vec{x}_i \end{aligned} \tag{4.37}$$

where we assume that $r_i \neq 0$.
Next, by making use of 4.36 and 4.37, we can derive the gradient of 4.35

$$\begin{aligned} \nabla_{\vec{w}} \sum_i L_H(r_i, \delta) &= \sum_i \nabla_{\vec{w}} L_H(r_i, \delta) \\ &= \sum_i \begin{cases} \nabla_{\vec{w}} \frac{r_i^2}{2} & \text{if } |r_i| \leq \delta \\ \nabla_{\vec{w}} \big(\delta |r_i| - \frac{\delta^2}{2}\big) & \text{if } |r_i| > \delta \end{cases} \\ &= \sum_i \begin{cases} r_i \nabla_{\vec{w}} r_i & \text{if } |r_i| \leq \delta \\ \delta \nabla_{\vec{w}} |r_i| & \text{if } |r_i| > \delta \end{cases} \\ &= \sum_i \begin{cases} r_i \vec{x}_i & \text{if } |r_i| \leq \delta \\ \delta \, \text{sign}(r_i) \vec{x}_i & \text{if } |r_i| > \delta \end{cases} \end{aligned} \tag{4.38}$$

Finally, we can use the gradient 4.38 to solve 4.35 by using a "naive" gradient descent algorithm A.3.2 or a more "sophisticated" method like L-BFGS A.3.5.2[10].

[10] this is what scikit-learn is doing!

4.7.2. Least absolute deviations

Another linear regression model, which is more robust to outliers, is *Linear absolute deviations (LAD)*[11].
Until now, we always - except in huber regession - optimized (minimized) the squared error. As we learned previously, *the squared error is sensitive to outliers* because of the square - large errors get even larger in the sense that they dominate all the other errors. One thing to do against this, is to use the *absolute error instead of the squared error*, because the absolute value does not grow so quickly in comparison to the squared value. This is exactly what *linear absolute deviations* is doing.
In plain English: Instead of the squared error we use the absolute error because it is more robust to outliers.
The new cost function is called *sum of absolute deviations* and is given by

$$\sum_i |y_i - f(\vec{x}_i)| \tag{4.39}$$

Thus, our new optimization problem is given by

$$\begin{aligned} \vec{w} &= \arg\min_{\vec{w} \in \mathbb{R}^d} \sum_i |y_i - f(\vec{x}_i)| \\ &= \arg\min_{\vec{w} \in \mathbb{R}^d} \sum_i |y_i - \vec{w}^\top \vec{x}_i| \\ &= \arg\min_{\vec{w} \in \mathbb{R}^d} \sum_i |r_i| \end{aligned} \tag{4.40}$$

where $r_i = y_i - \vec{w}^\top \vec{x}_i$ is the i-th residual.
Note: Because the residual can be negative - if our estimate is smaller than the true output - we take the absolute value of it to make sure that the summands are always positive - in the squared error we did not care about this because the square of smth. is always positive.

Another way to motivate 4.40 is to assume that the model $f(\vec{x}) = \vec{w}^\top \vec{x}$ is *correct in the mean only and is usually disturbed by some noise. This noise is distributed according to a Laplace distribution*[12]. Note that this is very similar to the case where we assumed Gaussian noise!

[11] also called *least absolute errors (LAE)*, *least absolute value (LAV)*, *least absolute residual (LAR)* or *sum of absolute deviations*
[12] see `https://en.wikipedia.org/wiki/Laplace_distribution`

Hence, we can write

$$\begin{aligned} p(y \mid \vec{x}, \vec{w}) &= \text{Lap}(y \mid \vec{w}^\top \vec{x}, b) \\ &= \frac{1}{2b} \exp\left(-\frac{1}{b}|y - \vec{w}^\top \vec{x}|\right) \\ &\propto \exp\left(-\frac{1}{b}|y - \vec{w}^\top \vec{x}|\right) \end{aligned} \tag{4.41}$$

where b is the scaling parameter (related to the variance) of the noise.
Looking at the log-likelihood, and assuming that the samples are independent of each other, reveals

$$\begin{aligned} \text{LL}_{\mathcal{D}}(\vec{w}) &= \log\left(\prod_i p(y_i \mid \vec{x}_i, \vec{w})\right) \\ &= \sum_i \log\Big(p(y_i \mid \vec{x}_i, \vec{w})\Big) \\ &\sim \sum_i \log\left(\exp\left(-\frac{1}{b}|y_i - \vec{w}^\top \vec{x}_i|\right)\right) \\ &= \sum_i -\frac{1}{b}|y_i - \vec{w}^\top \vec{x}_i| \end{aligned} \tag{4.42}$$

We observe that *maximizing the log-likelihood 4.42 is equivalent (up to a constant) to minimizing the sum of absolute deviations 4.39.*

The next step is to solve the optimization problem 4.40. Unfortunately, a direct solution is not possible, but we can *transform the problem into a linear program* [13] which we then can *solve by using linear programming (LP)*[14]. The transformation of 4.40 into a linear optimization problem requires some work - e.g. one problem is the $|\cdot|$ in the target function:
First, we write 4.40 in a slightly different way

$$\begin{aligned} &\min_{\vec{w}, r_i} \sum_i |r_i| \\ &\text{s.t.} \quad \vec{w}^\top \vec{x}_i + r_i = y_i \quad \forall_i \end{aligned} \tag{4.43}$$

In plain English: Find a $\vec{w}$ which minimizes the sum of absolute residuals r_i such that for all samples in our training set the prediction/output of our

[13] as discussed in section A.4, linear programs belong to the class of convex optimization problems and can be solved efficiently

[14] also called linear optimization

model is correct ($\vec{w}^\top \vec{x}_i = y_i$) up to the error r_i - we add the error/residual r_i to our prediction $\vec{w}^\top \vec{x}_i$ in order to remove the error and get the correct output y_i.

Next, we split up the residual r_i into the part r_i^+ for a positive error, the prediction is smaller than the true value - underestimation, and r_i^- for a negative error, the prediction is larger that the true value - overestimation. Because we would like to have $r_i^+ \geq 0$ and $r_i^- \geq 0$ (*both errors are now positive!*), r_i is written as $r_i = r_i^+ - r_i^-$, whereby the $-$ in front of r_i^- is necessary because r_i^- itself is positive. By substituting r_i in 4.43 we get

$$\begin{aligned} &\min_{\vec{w}, r_i^+, r_i^-} \sum_i |r_i^+ - r_i^-| \\ &\text{s.t.} \\ &r_i^+ \geq 0 \; r_i^- \geq 0 \\ &\vec{w}^\top \vec{x_i} + r_i^+ - r_i^- = y_i \quad \forall_i \end{aligned} \tag{4.44}$$

Because either r_i^+ or r_i^- is 0 - *either our prediction is larger or smaller than the true value, but not both at the same time*, we can simplify $|r_i^+ - r_i^-|$. Because $r_i^+ \geq 0$ and $r_i^- \geq 0$ and $r_i^+ = 0 \vee r_i^- = 0$, it holds that

$$\begin{aligned} |r_i^+ - r_i^-| &= |r_i^+| + |r_i^-| \\ &= r_i^+ + r_i^- \end{aligned} \tag{4.45}$$

By substituting 4.45 in 4.44, we obtain the following *linear optimization problem*

$$\begin{aligned} &\min_{\vec{w}, r_i^+, r_i^-} \sum_i r_i^+ + r_i^- \\ &\text{s.t.} \\ &r_i^+ \geq 0 \; r_i^- \geq 0 \\ &\vec{w}^\top \vec{x_i} + r_i^+ - r_i^- = y_i \quad \forall_i \end{aligned} \tag{4.46}$$

Finally, we can solve 4.46 (delivers us $\vec{w}$) by using our favorite linear programming algorithm (e.g. simplex algorithm, ...). Further details on how to do this, can be found in special literature on linear programming.

4.8. Sparsity regularization

In one of the previous sections we learned about regularization and introduced the L2 regularization. In L2 regularization we favor small weights

(close to zero), but what if we want some of the *feature weights being exactly zero*? Such a weight vector is called *sparse* and uses a *subset of the features* only - all features with non zero weights.
Usually L2 regularization delivers weights close to zero but not exactly zero.In this section we introduce a sparsity regularization called *L1* or ℓ_1 and the corresponding regression *LASSO*[15] for enforcing sparse weight vectors.

Like we did in L2 regularization, we add a penalty for large weights to our cost function and then obtain a new SSE given by

$$\begin{aligned} \mathrm{SSE}_{\mathcal{D}}(\vec{w}) &= \frac{1}{2}\sum_i (y_i - f(\vec{x}_i))^2 + \lambda \sum_j |(\vec{w})_j| \\ &= \frac{1}{2}\sum_i (y_i - f(\vec{x}_i))^2 + \lambda \|\vec{w}\|_1 \end{aligned} \tag{4.47}$$

where λ is called *regularization strength* and $||\vec{w}||_1 = \sum_j |(\vec{w})_j|$ is the 1-norm. By varying λ we can control how important we think small weights are in contrast to a small SSE.
Note: The only difference between L2 and L1 regularization is $\sum_j (\vec{w})_j^2$ vs. $\sum_j |(\vec{w})_j|$. We are no longer summing over the *squared values* but instead over the *absolute values* of $\vec{w}$.

When we looked at linear regression from a probabilistic point of view, we observed that ridge regression is equivalent to linear regression with gaussian noise and a gaussian prior for $\vec{w}$.
For LASSO it is the same, but instead of using a gaussian prior we have to use a *laplacian prior* (see exercise 6). Therefore

$$\begin{aligned} p_\lambda(\vec{w}) = \prod_j \mathrm{Lap}\left((\vec{w})_j \mid 0, \lambda\right) &= \prod_j \frac{1}{2\lambda} \exp\left(-\frac{|(\vec{w})_j - 0|}{\lambda}\right) \\ &= \prod_j \frac{1}{2\lambda} \exp\left(-\frac{|(\vec{w})_j|}{\lambda}\right) \end{aligned} \tag{4.48}$$

In ridge regression it was "very easy" to find a *closed form solution* - compute the gradient of the cost function, set it equal to zero and solve for $\vec{w}$. Unfortunately, this is *no longer possible* if we use the L1 instead of the L2 regularization. In general there does not exist a closed form solution and the only remaining option is to use an iterative algorithm (see section 4.8.1).

[15] details can be found in section 4.8.1

In Fig. 4.7 we can see a plot visualizing how the entries of the weight vector $\vec{w}$ are changing if the regularization strength is increased. The value for each feature (entry in the weight vector) is plotted on the y-axis and each feature is encoded by a different color. The x-axis displays the regularization strength.
As we can see, while increasing the regularization strength, many weights become zero very quickly whereas a few resists longer. We can interpret the "speed of becoming zero" as a some kind of *measurement for the importance of the feature*. This is because we ask the model to find a good regression while using a subset - the size depends on the regularization strength - of features only. Hence the model tries to find some features which describe the data well so that it can drop the other features.
We note that LASSO can be used for *feature selection*.

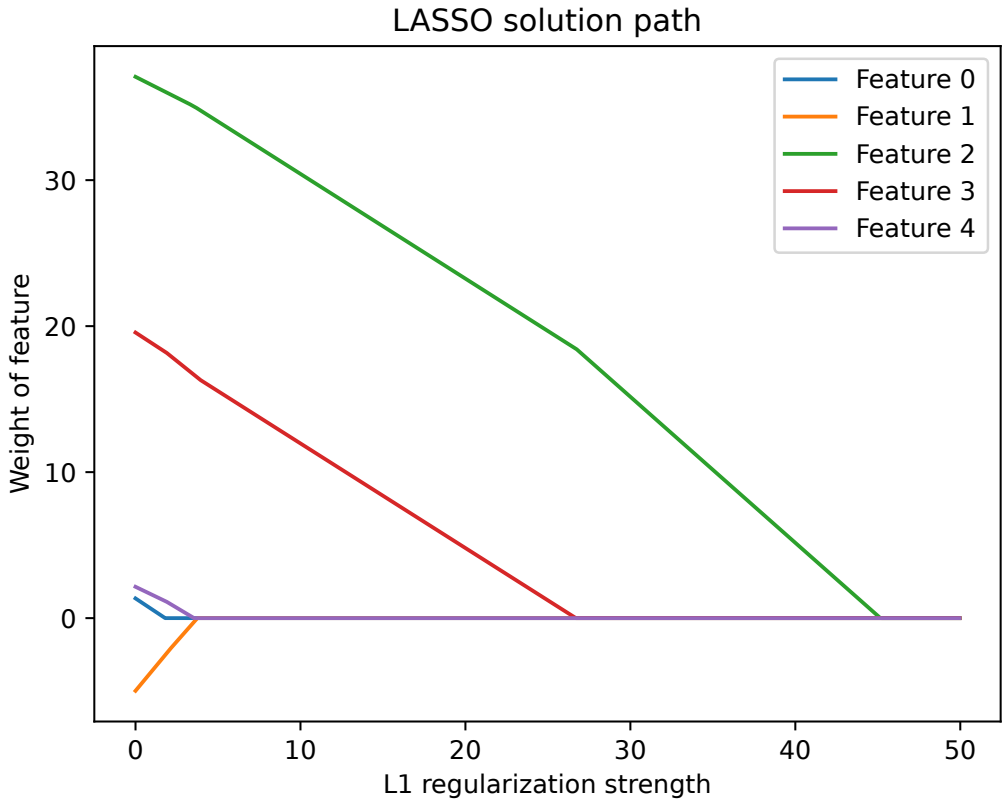


Figure 4.7: LASSO: Feature weights vs. regularization strength

4.8.1. Details on LASSO

LASSO[15][16] (Least Absolute Shrinkage and Selection Operator) is a model/procedure for delivering a sparse regression weight vector (minimizing the sum of squared errors). LASSO was originally stated as follows

$$\begin{aligned} &\min_{\vec{w} \in \mathbb{R}^d} \frac{1}{2} \|\vec{y} - \mathbf{X}\vec{w}\|_2^2 \\ &\text{s.t. } \|\vec{w}\|_1 \leq t \end{aligned} \tag{4.49}$$

Often an equivalent reformulation of 4.49 is used, where the constraint is included in the cost function by introducing a Lagrangian multiplier λ. It can be shown that for each value of t in 4.49, we can find a value λ such that 4.49 is equivalent to

$$\min_{\vec{w} \in \mathbb{R}^d} \frac{1}{2} \|\vec{y} - \mathbf{X}\vec{w}\|_2^2 + \lambda \|\vec{w}\|_1 \tag{4.50}$$

4.50 is the most common form of LASSO and an example of a *penalized regression model*.

Theorem 5 (Convexity of LASSO)**.** *LASSO as stated in 4.50 is a convex optimization problem.*

Proof. Because $\vec{w} \in \mathbb{R}^d$ and $\mathbb{R}^d$ is known to be a convex set, we only have to show that

$$\frac{1}{2} \|\vec{y} - \mathbf{X}\vec{w}\|_2^2 + \lambda \|\vec{w}\|_1 \tag{4.51}$$

is a convex function.

From theorem 2, we know that $\frac{1}{2}\|\vec{y} - \mathbf{X}\vec{w}\|_2^2$ is a convex function. Furthermore, we know from section A.3 that $\lambda\|\vec{w}\|_1$ is a convex function and the sum of two convex function is again a convex function.
Therefore we can conclude that 4.50 is a convex optimization problem. □

If the design matrix $\mathbf{X}$ is orthogonal, there exists a closed form solution to 4.50 (see section 4.8.1.1). In the general case, *coordinat descent* is a simple algorithm for solving 4.50 (see section 4.8.1.2).

[16] a good starting point for learning about LASSO is `https://statweb.stanford.edu/~tibs/lasso.html`

4.8.1.1 Special case - Orthogonal design matrix

Next, we derive the closed form solution for the special case of an orthogonal design matrix.
If the matrix $\mathbf{X}$ is orthogonal, we know that

$$\mathbf{X}^\top \mathbf{X} = \mathbb{I} \tag{4.52}$$

By making use of 4.52, we can simplify the expression in 4.50

$$\begin{aligned}\frac{1}{2}\|\vec{y} - \mathbf{X}\vec{w}\|_2^2 + \lambda\|\vec{w}\|_1 &= \frac{1}{2}(\vec{y} - \mathbf{X}\vec{w})^\top(\vec{y} - \mathbf{X}\vec{w}) + \lambda\|\vec{w}\|_1 \\ &= \frac{1}{2}\vec{y}^\top\vec{y} - \frac{1}{2}\vec{y}^\top\mathbf{X}\vec{w} - \frac{1}{2}\vec{w}^\top\mathbf{X}^\top\vec{y} + \frac{1}{2}\vec{w}^\top\mathbf{X}^\top\mathbf{X}\vec{w} + \lambda\|\vec{w}\|_1 \\ &= \frac{1}{2}\vec{y}^\top\vec{y} - \vec{y}^\top\mathbf{X}\vec{w} + \frac{1}{2}\vec{w}^\top\vec{w} + \lambda\|\vec{w}\|_1\end{aligned} \tag{4.53}$$

Substituting 4.53 into 4.50 yields

$$\begin{aligned}&\min_{\vec{w}\in\mathbb{R}^d} \frac{1}{2}\|\vec{y} - \mathbf{X}\vec{w}\|_2^2 + \lambda\|\vec{w}\|_1 \\ &\Leftrightarrow \\ &\min_{\vec{w}\in\mathbb{R}^d} \frac{1}{2}\vec{y}^\top\vec{y} - \vec{y}^\top\mathbf{X}\vec{w} + \frac{1}{2}\vec{w}^\top\vec{w} + \lambda\|\vec{w}\|_1 \\ &\Leftrightarrow \\ &\min_{\vec{w}\in\mathbb{R}^d} -\vec{y}^\top\mathbf{X}\vec{w} + \frac{1}{2}\vec{w}^\top\vec{w} + \lambda\|\vec{w}\|_1 \\ &\Leftrightarrow \\ &\min_{\vec{w}\in\mathbb{R}^d} -\vec{\alpha}\vec{w} + \frac{1}{2}\vec{w}^\top\vec{w} + \lambda\|\vec{w}\|_1\end{aligned} \tag{4.54}$$

where $\vec{\alpha} = \vec{y}^\top\mathbf{X}$. Note that $\vec{\alpha}$ is a row vector whereas $\vec{w}$ is a column vector. Rewrite all terms in 4.54 as a sum over the dimensions

$$\begin{aligned}&\min_{\vec{w}\in\mathbb{R}^d} -\sum_j(\vec{\alpha})_j(\vec{w})_j + \frac{1}{2}\sum_j(\vec{w})_j^2 + \lambda\sum_j|(\vec{w})_j| \\ &\Leftrightarrow \\ &\min_{\vec{w}\in\mathbb{R}^d} -\sum_j(\vec{\alpha})_j(\vec{w})_j + \frac{1}{2}(\vec{w})_j^2 + \lambda|(\vec{w})_j|\end{aligned} \tag{4.55}$$

Because 4.54 is minimizing a convex function, we compute the gradient of the cost function in 4.54 by differentiating 4.55 with respect to each dimension

j, and set it equal to zero.

$$\begin{aligned} &\frac{\partial}{\partial(\vec{w})_j}\left(-\sum_j(\vec{\alpha})_j(\vec{w})_j+\frac{1}{2}(\vec{w})_j^2+\lambda|(\vec{w})_j|\right)=0 \\ &\Leftrightarrow \\ &-(\vec{\alpha})_j+(\vec{w})_j+\lambda\frac{\partial}{\partial(\vec{w})_j}(|(\vec{w})_j|)=0 \end{aligned} \tag{4.56}$$

Note that $f(x)=|x|$ is not differentiable at 0. Therefore we have to take a look at the subdifferential (see section A.2.4) of $f(x)=|x|$

$$\frac{\partial|x|}{\partial x}=\begin{cases}1 & \text{if } x>0\\ -1 & \text{if } x<0\\ [-1,1] & \text{if } x=0\end{cases} \tag{4.57}$$

Notice that the subderivative at $x=0$ can be any value in $[-1,1]$. We can "arbitrarily" choose one value from this interval.
Next, we have to distinguish between two cases:

1. $(\vec{\alpha})_j>0$: In order to minimize 4.54, it must be that $(\vec{w})_j\geq 0$. Therefore we can set $\frac{\partial|(\vec{w})_j|}{\partial(\vec{w})_j}=1$.
   Substituting this into 4.56 yields

   $$\begin{aligned} &-(\vec{\alpha})_j+(\vec{w})_j+\lambda=0 \\ &\Leftrightarrow \\ &(\vec{w})_j=(\vec{\alpha})_j-\lambda \end{aligned} \tag{4.58}$$

   But because we argued that $(\vec{w})_j\geq 0$, we have to clamp the value of $(\vec{w})_j$ to zero when it becomes negative

   $$(\vec{w})_j=\max(0,(\vec{\alpha})_j-\lambda) \tag{4.59}$$

2. $(\vec{\alpha})_j\leq 0$: In order to minimize 4.54, it must be that $(\vec{w})_j\leq 0$. Therefore we can set $\frac{\partial|(\vec{w})_j|}{\partial(\vec{w})_j}=-1$
   Substituting this into 4.56 yields

   $$\begin{aligned} &-(\vec{\alpha})_j+(\vec{w})_j-\lambda=0 \\ &\Leftrightarrow \\ &(\vec{w})_j=(\vec{\alpha})_j+\lambda \end{aligned} \tag{4.60}$$

   But because we argued that $(\vec{w})_j\leq 0$, we have to clamp the value of $(\vec{w})_j$ to zero when it becomes positive

   $$(\vec{w})_j=\min(0,(\vec{\alpha})_j+\lambda) \tag{4.61}$$

We observe that both cases can be written as

$$(\vec{w})_j = \text{sign}(\alpha_j)(|\alpha_j| - \lambda)^+ \tag{4.62}$$

where[17]

$$(|\alpha_j| - \lambda)^+ = \max(0, |\alpha_j| - \lambda) \tag{4.63}$$

Finally, we have arrived at the closed form solution 4.62 of LASSO 4.50 - if the design matrix $\mathbf{X}$ is orthogonal.
By looking at the closed form solution 4.62, it becomes obvious that LASSO tends to set more entries/features in $\vec{w}$ to zero, when we increase the regularization strength λ.

4.8.1.2 General case

As already mentioned, in the general case there does not exist a closed form solution of 4.50. In this section, we derive the update rule for a coordinate descent algorithm (see section A.3.7).

First, we have to verify that our problem (4.50) is suited for the coordinate descent algorithm.
From section A.3.7 we know that the coordinate descent algorithm is suited for the following optimization problem

$$\min_{\vec{\theta} \in \mathbb{R}^d} f(\vec{\theta}) + \sum_j h_j((\vec{\theta})_j) \tag{4.64}$$

where f and h_j are convex functions. In addition, f is required to be differentiable everywhere (viz. f is a smooth function).
We can recognize LASSO 4.50 as an instance 4.64 by setting

$$\begin{aligned} \vec{\theta} &:= \vec{w} \\ f(\vec{\theta}) &:= \frac{1}{2}\|\vec{y} - \mathbf{X}\vec{\theta}\|_2^2 \\ h_j((\vec{\theta})_j) &:= \lambda|(\vec{\theta})_j| \end{aligned} \tag{4.65}$$

We conclude that the prerequisites for applying the coordinate descent algorithm are met.

[17] also called *soft threshold(ing) function*

We can rewrite 4.50 as a sum over all data points and their dimensions

$$\frac{1}{2}\sum_i \left(y_i - \sum_j (\vec{x}_i)_j (\vec{w})_j \right)^2 + \lambda \sum_j |(\vec{w})_j| \tag{4.66}$$

Next, we compute the derivative of the SSE in 4.66 with respect to th k-th component of $\vec{w}$

$$\begin{aligned}
\frac{\partial}{\partial(\vec{w})_k}\frac{1}{2}\sum_i \left(y_i - \sum_j (\vec{x}_i)_j (\vec{w})_j \right)^2 &= \frac{\partial}{\partial(\vec{w})_k}\left(\frac{1}{2}\sum_i y_i^2 - 2y_i \sum_j (\vec{x}_i)_j (\vec{w})_j + \Big(\sum_j (\vec{x}_i)_j (\vec{w})_j \Big)^2 \right) \\
&= -\sum_i \frac{\partial}{\partial(\vec{w})_k} y_i \sum_j (\vec{x}_i)_j (\vec{w})_j + \\
&\sum_j (\vec{x}_i)_j (\vec{w})_j \frac{\partial}{\partial(\vec{w})_k} \sum_j (\vec{x}_i)_j (\vec{w})_j \\
&= -\sum_i y_i (\vec{x}_i)_k + (\vec{x}_i)_k \sum_j (\vec{x}_i)_j (\vec{w})_j \\
&= -\sum_i (\vec{x}_i)_k \left(y_i - \sum_j (\vec{x}_i)_j (\vec{w})_j \right) \\
&= -\sum_i (\vec{x}_i)_k \left(y_i - \sum_{j\neq k} (\vec{x}_i)_j (\vec{w})_j \right) + \sum_i (\vec{x}_i)_k (\vec{x}_i)_k (\vec{w})_k \\
&= -r_k + \sum_i (\vec{x}_i)_k^2 (\vec{w})_k
\end{aligned} \tag{4.67}$$

where $r_k = \sum_i (\vec{x}_i)_k \left(y_i - \sum_{j\neq k} (\vec{x}_i)_j (\vec{w})_j \right)$.

Computing the derivative of the l1-penalty $\lambda|w|$ yields

$$\begin{aligned}
\frac{\partial}{\partial w}\lambda|w| &= \lambda \begin{cases} 1 & \text{if } w > 0 \\ -1 & \text{if } w < 0 \\ [-1,1] & \text{if } w = 0 \end{cases} \\
&= \begin{cases} \lambda & \text{if } w > 0 \\ -\lambda & \text{if } w < 0 \\ [-\lambda,\lambda] & \text{if } w = 0 \end{cases} \\
&= \begin{cases} \text{sign}(w)\lambda & \text{if } w \neq 0 \\ [-\lambda,\lambda] & \text{if } w = 0 \end{cases}
\end{aligned} \tag{4.68}$$

Combining 4.67 and 4.68 yields

$$\begin{aligned}
&\frac{\partial}{\partial(\vec{w})_k}\left(\frac{1}{2}\sum_i\left(y_i-\sum_j(\vec{x}_i)_j(\vec{w})_j\right)^2+\lambda\sum_j|(\vec{w})_j|\right)\\
&=-r_k+\sum_i(\vec{x}_i)_k^2(\vec{w})_k+\begin{cases}\text{sign}((\vec{w})_k)\lambda & \text{if } (\vec{w})_k\neq 0\\ [-\lambda,\lambda] & \text{if } (\vec{w})_k=0\end{cases}\\
&=\begin{cases}-r_k+\sum_i(\vec{x}_i)_k^2(\vec{w})_k+\text{sign}((\vec{w})_k)\lambda & \text{if } (\vec{w})_k\neq 0\\ [-r_k-\lambda,-r_k+\lambda] & \text{if } (\vec{w})_k=0\end{cases}
\end{aligned}\tag{4.69}$$

Setting the derivative to zero

$$0=\begin{cases}-r_k+\sum_i(\vec{x}_i)_k^2(\vec{w})_k+\text{sign}((\vec{w})_k)\lambda & \text{if } (\vec{w})_k\neq 0\\ [-r_k-\lambda,-r_k+\lambda] & \text{if } (\vec{w})_k=0\end{cases}\tag{4.70}$$

By looking at the second case in 4.70, we find that

$$\begin{aligned}
&0\in[-r_k-\lambda,-r_k+\lambda]\\
&\Leftrightarrow\\
&-\lambda\leq r_k\leq\lambda
\end{aligned}\tag{4.71}$$

Solving the first case in 4.70 for $(\vec{w})_k$ yields

$$\begin{aligned}
&-r_k+\sum_i(\vec{x}_i)_k^2(\vec{w})_k+\text{sign}((\vec{w})_k)\lambda=0\\
&\Leftrightarrow\\
&-r_k+\sum_i(\vec{x}_i)_k^2(\vec{w})_k+\text{sign}(r_k)\lambda=0\\
&\Leftrightarrow\\
&(\vec{w})_k=\frac{r_k-\text{sign}(r_k)\lambda}{\sum_i(\vec{x}_i)_k^2}
\end{aligned}\tag{4.72}$$

Because of 4.70, we know that sign $((\vec{w})_k)$ must be equal to $\text{sign}(r_k)$.

Combining 4.72 and 4.71 delivers the final formula for updating $(\vec{w})_k$

$$(\vec{w})_k=\begin{cases}0 & \text{if } -\lambda\leq r_k\leq\lambda\\ \frac{r_k-\text{sign}(r_k)\lambda}{\sum_i(\vec{x}_i)_k^2} & \text{otherwise}\end{cases}\tag{4.73}$$

We observe that we can write both cases as

$$(\vec{w})_k=\frac{\text{sign}(r_k)(|r_k|-\lambda)^+}{\sum_i(\vec{x}_i)_k^2}\tag{4.74}$$

where $(...)^+$ denotes the familiar soft thresholding function.

The final coordinate descent algorithm for solving 4.50 is described in Algorithm 2. Note that we have to recompute the value of r_k for each k. This is because the value of $\vec{w}$ is updated in each update step.

Algorithm 2 LASSO - Coordinate descent algorithm

1: $\vec{w} = \vec{0}$ ▷ Initialize $\vec{w}$
2: **repeat**
3: **for all** $(\vec{w})_k$ **do** ▷ Iterate over all dimensions of $\vec{w}$
4: $r_k = \sum_i (\vec{x}_i)_k \left(y_i - \sum_{j \neq k} (\vec{x}_i)_j (\vec{w})_j \right)$
5: $(\vec{w})_k = \frac{\text{sign}(r_k)(|r_k| - \lambda)^+}{\sum_i (\vec{x}_i)_k^2}$ ▷ Update $\vec{w}$
6: **end for**
7: **until** convergence

4.9. Elastic net

By using L2 regularization we get small weights and by using L1 regularization (LASSO) we can perform feature selection. The *problem with LASSO* is that if we have *multiple correlated features* in the data set, it will *only pick one of them but not all*. However, we would like to pick all correlated features once we decided to select one of them - this is called *grouping effect*. Selecting correlated features is beneficial because *the usage of correlated features makes the model more robust*. Furthermore, *it improves the interpretability of the model* because it gives more insight into the underlying domain/problem and the relevant features.

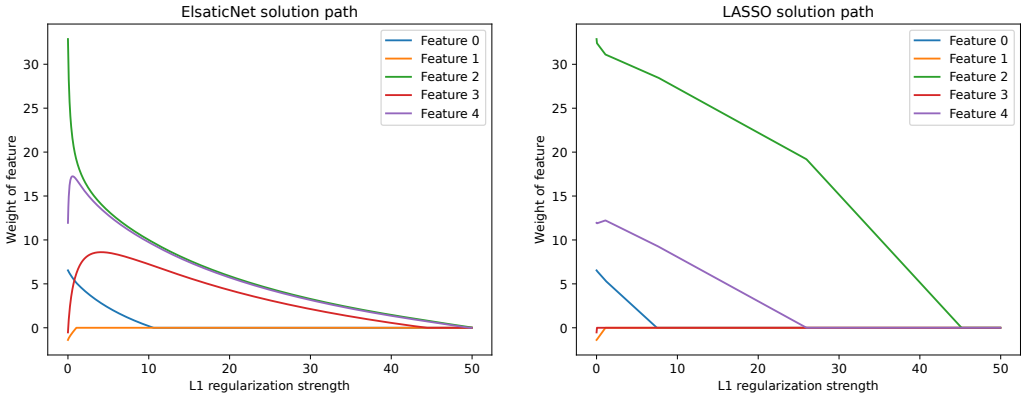
This problem is not new and people thought about this and came up with a "solution" called *Elastic net* [19][18].
Elastic net is a linear regression model (hence $f(\vec{x}) = \vec{w}^\top \vec{x}$) and the idea behind it is to *use a combination of L1 and L2 regularization*[19]. By this, the model delivers small weights and performs a feature selection according to the grouping effect - therefore it picks all correlated features instead of just picking one of them - see Fig. 4.8.

[18] more details can be found in the original paper by Zou & Hastie which is not too difficult to read

[19] together, L1 and L2 regularization is also called *elastic net penalty*

In Fig. 4.8 we applied Elastic net and LASSO to an artifical data set. Features 0 to 2 are the orginal ones, feature 3 and 4 are created from feature 2 by adding gaussian noise to it. As we can see, ElasticNet tends to select all correlated features (2, 3 and 4), whereas LASSO does not.

Figure 4.8: Grouping effect: Elastic net vs. LASSO



The optimization problem of elastic net is given by

$$\begin{aligned} \vec{w} &= \arg\min_{\vec{w} \in \mathbb{R}^d} \mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \lambda_1 \|\vec{w}\|_1 + \lambda_2 \|\vec{w}\|_2^2 \\ &= \arg\min_{\vec{w} \in \mathbb{R}^d} \mathrm{NLL}_{\mathcal{D}}(\vec{w}) + \lambda_1 \|\vec{w}\|_1 + \lambda_2 \|\vec{w}\|_2^2 \end{aligned} \tag{4.75}$$

where the first equation is the notation used in the original paper and the second one is the notation used on the lecture slides (both are equivalent to each other). The parameters λ_1 and λ_2 enable us control the regularization strength and influence of L1 and L2 regularization.
Note: By using "extreme" values for λ_1 and/or λ_2 we can recover "ordinary" linear regression ($\lambda_1 = \lambda_2 = 0$), ridge regression ($\lambda_1 = 0$) and LASSO ($\lambda_2 = 0$). Thus all of them are special cases of elastic net.

Sometimes people use an alternative (equivalent) formulation[20] of the cost function, instead of

$$\mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \lambda_1 \|\vec{w}\|_1 + \lambda_2 \|\vec{w}\|_2^2 \tag{4.76}$$

they use

$$\mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \alpha\gamma \|\vec{w}\|_1 + \alpha(1-\gamma) \|\vec{w}\|_2^2 \tag{4.77}$$

[20] e.g. scikit-learn [11] does so

where $\alpha = \lambda_1 + \lambda_2$ and $\gamma = \frac{\lambda_1}{\lambda_1+\lambda_2}$. The equivalence can be easily proven by substituting α and γ in 4.77 and after some simplifications one obtains 4.76.

Again, the *optimization problem can not be solved analytically*[21] that is why iterative algorithms are used again (see section 4.9.1).

4.9.1. Optimization

Theorem 6 (Convexity of Elastic net)**.** *ElasticNet as stated in 4.75 is a convex optimization problem.*

Proof. Because $\vec{w} \in \mathbb{R}^d$ and $\mathbb{R}^d$ is known to be a convex set, we only have to show that

$$\mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \lambda_1\|\vec{w}\|_1 + \lambda_2\|\vec{w}\|_2^2 \tag{4.78}$$

is a convex function.

From theorem 3, we know that $\mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \lambda_2\|\vec{w}\|_2^2$ is a convex function. Furthermore, we know from section A.3 that $\lambda_1\|\vec{w}\|_1$ is a convex function and the sum of two convex function is again a convex function.
Therefore we can conclude that 4.75 is a convex optimization problem. □

Next, we argue that 4.75 in general can be solved by using the coordinate descent algorithm and derive the associated update rule. Because the derivation of the update rule is very similar to the derivation we did in LASSO (see section 4.8.1.2), we will keep things superficially. If you miss some steps, go to section 4.8.1.2 - it's more or less exactly the same.

First, we have to verify that our problem (4.75) is suited for the coordinate descent algorithm.
From section A.3.7 we know that the coordinate descent algorithm is suited for the following optimization problem

$$\min_{\vec{\theta}\in\mathbb{R}^d} f(\vec{\theta}) + \sum_j h_j((\vec{\theta})_j) \tag{4.79}$$

where f and h_j are convex functions. In addition, f have to be differentiable everywhere (viz. f is a smooth function).

[21] this is true for the general case but there are some special cases where a closed form solution exists

We can transform 4.79 into ElasticNet 4.75 by setting

$$\begin{aligned}\vec{\theta} &:= \vec{w} \\ f(\vec{\theta}) &:= \frac{1}{2}\|\vec{y} - \mathbf{X}\vec{\theta}\|_2^2 + \lambda_2\|\vec{w}\|_2^2 \\ h_j((\vec{\theta})_j) &:= \lambda_1|(\vec{\theta})_j|\end{aligned} \tag{4.80}$$

We conclude that the prerequisites for applying the coordinate descent algorithm are met.

We can rewrite 4.75 as a sum over all data points and their dimensions

$$\frac{1}{2}\sum_i \left(y_i - \sum_j (\vec{x}_i)_j(\vec{w})_j\right)^2 + \lambda_1 \sum_j |(\vec{w})_j| + \lambda_2 \sum_j (\vec{w})_j^2 \tag{4.81}$$

Next, we compute the derivative of the SSE + L2-regularization in 4.81 with respect to th k-th component of $\vec{w}$

$$\frac{\partial}{\partial(\vec{w})_k}\left(\frac{1}{2}\sum_i \left(y_i - \sum_j (\vec{x}_i)_j(\vec{w})_j\right)^2 + \lambda_2 \sum_j (\vec{w})_j^2\right) = -r_k + \sum_i (\vec{x}_i)_k^2(\vec{w})_k + \lambda_2(\vec{w})_k \tag{4.82}$$

where $r_k = \sum_i (\vec{x}_i)_k \left(y_i - \sum_{j\neq k}(\vec{x}_i)_j(\vec{w})_j\right)$.

Computing the derivative of the l1-penalty $\lambda_1|w|$ yields

$$\frac{\partial}{\partial w}\lambda_1|w| = \begin{cases}\text{sign}(w)\lambda_1 & \text{if } w \neq 0 \\ [-\lambda_1, \lambda_1] & \text{if } w = 0\end{cases} \tag{4.83}$$

Combining 4.82 and 4.83 yields

$$\begin{aligned}&\frac{\partial}{\partial(\vec{w})_k}\left(\frac{1}{2}\sum_i \left(y_i - \sum_j (\vec{x}_i)_j(\vec{w})_j\right)^2 + \lambda_1 \sum_j |(\vec{w})_j| + \lambda_2 \sum_j (\vec{w})_j^2\right) \\ &= -r_k + \sum_i (\vec{x}_i)_k^2(\vec{w})_k + \lambda_2(\vec{w})_k + \begin{cases}\text{sign}((\vec{w})_k)\lambda_1 & \text{if } (\vec{w})_k \neq 0 \\ [-\lambda_1, \lambda_1] & \text{if } (\vec{w})_k = 0\end{cases} \\ &= \begin{cases}-r_k + \sum_i (\vec{x}_i)_k^2(\vec{w})_k + \lambda_2(\vec{w})_k + \text{sign}((\vec{w})_k)\lambda_1 & \text{if } (\vec{w})_k \neq 0 \\ [-r_k - \lambda_1, -r_k + \lambda_1] & \text{if } (\vec{w})_k = 0\end{cases}\end{aligned} \tag{4.84}$$

Setting the derivative 4.84 to zero

$$0 = \begin{cases}-r_k + \sum_i (\vec{x}_i)_k^2(\vec{w})_k + \lambda_2(\vec{w})_k + \text{sign}((\vec{w})_k)\lambda_1 & \text{if } (\vec{w})_k \neq 0 \\ [-r_k - \lambda_1, -r_k + \lambda_1] & \text{if } (\vec{w})_k = 0\end{cases} \tag{4.85}$$

By looking at the second case in 4.85, we find that

$$\begin{aligned} & 0 \in [-r_k - \lambda_1, -r_k + \lambda_1] \\ & \Leftrightarrow \\ & -\lambda_1 \leq r_k \leq \lambda_1 \end{aligned} \tag{4.86}$$

Solving the first case in 4.85 for $(\vec{w})_k$ yields

$$\begin{aligned} & -r_k + \sum_i (\vec{x}_i)_k^2 (\vec{w})_k + \lambda_2 (\vec{w})_k + \text{sign}((\vec{w})_k)\lambda_1 = 0 \\ & \Leftrightarrow \\ & -r_k + \sum_i (\vec{x}_i)_k^2 (\vec{w})_k + \lambda_2 (\vec{w})_k + \text{sign}(r_k)\lambda_1 = 0 \\ & \Leftrightarrow \\ & (\vec{w})_k = \frac{r_k - \text{sign}(r_k)\lambda_1}{\sum_i (\vec{x}_i)_k^2 + \lambda_2} \end{aligned} \tag{4.87}$$

Because of 4.85, we know that sign $((\vec{w})_k)$ must be equal to $\text{sign}(r_k)$.

Combining 4.87 and 4.86 delivers the final formula for updating $(\vec{w})_k$

$$(\vec{w})_k = \begin{cases} 0 & \text{if } -\lambda_1 \leq r_k \leq \lambda_1 \\ \frac{r_k - \text{sign}(r_k)\lambda_1}{\sum_i (\vec{x}_i)_k^2 + \lambda_2} & \text{otherwise} \end{cases} \tag{4.88}$$

We obeserve that both cases can be written as

$$(\vec{w})_k = \frac{\text{sign}(r_k)(|r_k| - \lambda_1)^+}{\sum_i (\vec{x}_i)_k^2 + \lambda_2} \tag{4.89}$$

where $(...)^+$ denotes the familiar soft thresholding function.

The final coordinate descent algorithm for solving 4.75 is described in Algorithm 3. Note that we have to recompute the value of r_k for each k . This is because the value of $\vec{w}$ is updated in each update step.

Algorithm 3 ElasticNet - Coordinate descent algorithm

1: $\vec{w} = \vec{0}$ ▷ Initialize $\vec{w}$
2: **repeat**
3: **for all** $(\vec{w})_k$ **do** ▷ Iterate over all dimensions of $\vec{w}$
4: $r_k = \sum_i (\vec{x}_i)_k \left(y_i - \sum_{j \neq k} (\vec{x}_i)_j (\vec{w})_j \right)$
5: $(\vec{w})_k = \frac{\text{sign}(r_k)(|r_k| - \lambda_1)^+}{\sum_i (\vec{x}_i)_k^2 + \lambda_2}$ ▷ Update $\vec{w}$
6: **end for**
7: **until** convergence

4.10. Bayesian linear regression

So far we have always used one $\vec{w}$ only. We found it by either maximum likelihood or maximum a posteriori. But what about using *all possible* $\vec{w}$? At first glance this might sound crazy but the general idea is to make predictions by using a "weighted combination of all possible weight vectors $\vec{w}$". This is useful because the prediction of the most probable model does not necessarily has to be the most probable prediction in general. Therefore, weighted averaging predictions from all possible models is beneficial. This is what *Bayesian linear regression* is all about.
In Bayesian linear regression we are no longer fitting a model - in the sense of computing a specific weight vector $\vec{w}$ - but instead computing a *predictive distribution*. This means that the output of a prediction is a distribution[22] and no longer a single value! If we have to *deliver a prediction*, we usually *choose the most likely value* under the predictive distribution.
The predictive distribution itself, in its simplest version, is given by

$$P(y \mid \vec{x}, \mathcal{D}) = \int p(y \mid \vec{x}, \vec{w}) p(\vec{w} \mid \mathcal{D}) \mathrm{d}\vec{w} \tag{4.90}$$

The easiest version of 4.90 is the one where all distributions are Gaussians and we integrate over the weight $\vec{w}$ only[23].
We can use the models from the previous sections

$$\begin{aligned} p(y \mid \vec{x}, \vec{w}) &= \mathcal{N}(y \mid \vec{w}^\top \vec{x}, \sigma^2) \\ p(\vec{w} \mid \mathcal{D}) = p(\mathcal{D} \mid \vec{w}) p(\vec{w}) &= \prod_i \mathcal{N}(y_i \mid \vec{w}^\top \vec{x}_i, \sigma^2) \prod_j \mathcal{N}((\vec{w})_j \mid 0, \lambda^2) \end{aligned} \tag{4.91}$$

[22] inference
[23] we could include additional parameters like the noise in the integration

Because the *product of Gaussian distributions is again a Gaussian distribution* and the *convolution (integral) of two Gaussian distributions is again gaussian*, the final result $p(y \mid \vec{x}, \mathcal{D})$ *is a Gaussian distribution* given by

$$p(y \mid \vec{x}, \mathcal{D}) = \mathcal{N}(y \mid \mu_{\vec{x}} = \frac{1}{\sigma^2}\vec{x}^\top \mathbf{\Sigma} \sum_i \vec{x}_i y_i, \sigma^2_{\vec{x}} = \sigma^2 + \vec{x}^\top \mathbf{\Sigma} \vec{x}) \tag{4.92}$$

where $\mathbf{\Sigma}^{-1} = \lambda \mathbb{I} + \frac{1}{\sigma^2} \sum_i \vec{x}_i \vec{x}_i^\top$.

An example of Bayesian linear regression is visualized in Fig. 4.9. The red curve visualizes the mean (most likely prediction) and the light red shaded region visualizes samples from the predictive distribution. The blue points are noisy samples from the ground truth (green curve). A more complex

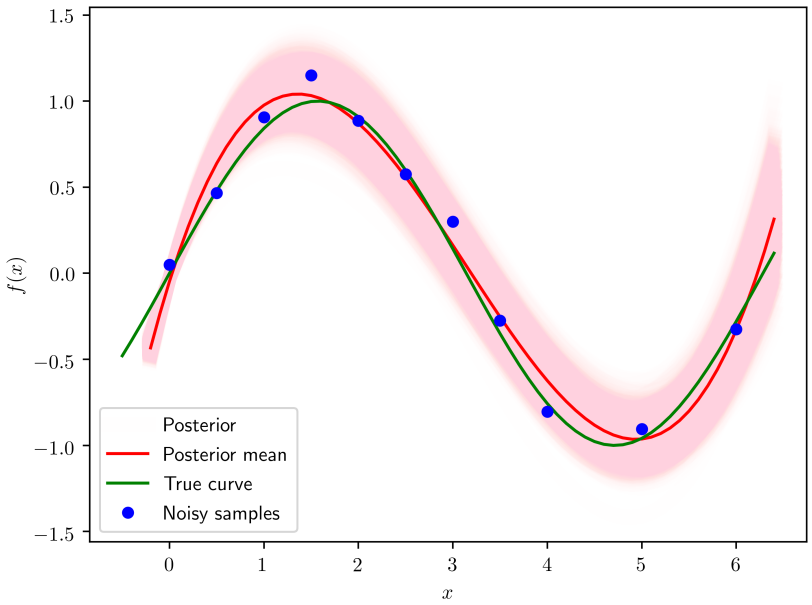


Figure 4.9: Example of Bayesian linear regression

example - including different non-Gaussian distributions as well as an integration over σ - can be found on the lecture slides.

Note: Sometimes, the predictive distribution can not be computed because the integral is intractable - this might happen if we use "not so friendly distributions" like the Gaussian distribution, consequently we have to use some kind of approximation[24].

[24] Approximate Inference

4.10.1. Conjugate priors

A prior distribution is called a *conjugate prior* to some likelihood function if the posterior has the same distribution as the prior (with other parameter values of course). Recall that the posterior distribution is computed by

$$p(\vec{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \vec{\theta})p(\vec{\theta})}{\int p(\mathcal{D} \mid \vec{\theta})p(\vec{\theta})\mathrm{d}\vec{\theta}} \tag{4.93}$$

where $\vec{\theta}$ contains the parameters of the likelihood and the term $p(\mathcal{D} \mid \vec{\theta})$ denotes the likelihood and the prior distribution over the parameters $\vec{\theta}$ is denoted by $p(\theta)$.
Conjugate priors are useful because the posterior is a closed form probability distribution. Otherwise we would have to approximate the normalizing constant (denominator of 4.93) which might be computational expensive.

A list of conjugate priors can be found in the wikipedia article on conjugate priors[25].

4.10.1.1 Example

The Beta distribution is a conjugate prior for the Bernoulli distribution. To see why, we simply have to compute 4.93 with the Bernoulli likelihood and the Beta distribution as a prior and verify that the result is a Beta distribution.
We assume that $\mathcal{D} = \{x_i\}$ where $x_i \in \{0, 1\}$.
First, we compute the numerator of 4.93

$$\begin{aligned} p(\mathcal{D} \mid \theta)p(\theta) &= \mathrm{Ber}(\mathcal{D} \mid \theta)\,\mathrm{Beta}(\theta \mid \alpha, \beta) \\ &= \theta^{\sum_i \mathbb{1}(x_i=1)}(1-\theta)^{\sum_i \mathbb{1}(x_i=0)}\,\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1} \\ &= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1+\sum_i \mathbb{1}(x_i=1)}(1-\theta)^{\beta-1+\sum_i \mathbb{1}(x_i=0)} \end{aligned} \tag{4.94}$$

where we wrote the parameter p of the Bernoulli distribution as θ and assumed that the data points are i.i.d - that is why the Bernoulli likelihood is a product of Bernoulli distributions.

[25] see `https://en.wikipedia.org/wiki/Conjugate_prior`

Before proceeding, we notice that

$$\begin{aligned}
&\int \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}\mathrm{d}\theta = 1 \\
&\Leftrightarrow \\
&\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\int \theta^{\alpha-1}(1-\theta)^{\beta-1}\mathrm{d}\theta = 1 \\
&\Leftrightarrow \\
&\int \theta^{\alpha-1}(1-\theta)^{\beta-1}\mathrm{d}\theta = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}
\end{aligned} \tag{4.95}$$

Next, we compute the denominator of 4.93 by making use of 4.95

$$\begin{aligned}
\int p(\mathcal{D} \mid \theta)p(\theta)\mathrm{d}\theta &= \int \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1+\sum_i \mathbb{1}(x_i=1)}(1-\theta)^{\beta-1+\sum_i \mathbb{1}(x_i=0)}\mathrm{d}\theta \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\int \theta^{\alpha-1+\sum_i \mathbb{1}(x_i=1)}(1-\theta)^{\beta-1+\sum_i \mathbb{1}(x_i=0)}\mathrm{d}\theta \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\frac{\Gamma(\alpha+\sum_i \mathbb{1}(x_i=1))\Gamma(\beta+\sum_i \mathbb{1}(x_i=0))}{\Gamma(\alpha+\sum_i \mathbb{1}(x_i=1)+\beta+\sum_i \mathbb{1}(x_i=0))} \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\frac{\Gamma(\alpha+\sum_i \mathbb{1}(x_i=1))\Gamma(\beta+\sum_i \mathbb{1}(x_i=0))}{\Gamma(\alpha+\beta+|\mathcal{D}|)}
\end{aligned} \tag{4.96}$$

Finally, we can plug 4.94 and 4.96 into 4.93 and obtain a Beta distribution

$$\begin{aligned}
p(\theta \mid \mathcal{D}) &= \frac{p(\mathcal{D} \mid \theta)p(\theta)}{\int p(\mathcal{D} \mid \theta)p(\theta)\mathrm{d}\theta} \\
&= \frac{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1+\sum_i \mathbb{1}(x_i=1)}(1-\theta)^{\beta-1+\sum_i \mathbb{1}(x_i=0)}}{\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\frac{\Gamma(\alpha+\sum_i \mathbb{1}(x_i=1))\Gamma(\beta+\sum_i \mathbb{1}(x_i=0))}{\Gamma(\alpha+\beta+|\mathcal{D}|)}} \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\theta^{\alpha-1+\sum_i \mathbb{1}(x_i=1)}(1-\theta)^{\beta-1+\sum_i \mathbb{1}(x_i=0)}\frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)} \\
&\quad \frac{\Gamma(\alpha+\beta+|\mathcal{D}|)}{\Gamma(\alpha+\sum_i \mathbb{1}(x_i=1))\Gamma(\beta+\sum_i \mathbb{1}(x_i=0))} \\
&= \frac{\Gamma(\alpha+\beta+|\mathcal{D}|)}{\Gamma(\alpha+\sum_i \mathbb{1}(x_i=1))\Gamma(\beta+\sum_i \mathbb{1}(x_i=0))}\theta^{\alpha-1+\sum_i \mathbb{1}(x_i=1)}(1-\theta)^{\beta-1+\sum_i \mathbb{1}(x_i=0)} \\
&= \mathrm{Beta}(\theta \mid \alpha+\sum_i \mathbb{1}(x_i=1), \beta+\sum_i \mathbb{1}(x_i=0))
\end{aligned} \tag{4.97}$$

4.11. Kernel regression

In the section about feature transformation we learned that we can use linear regression for non-linear settings by transforming the data through a function ϕ. Therefore we replace all $\vec{x}$ by $\phi(\vec{x})$ and the design matrix $\mathbf{X}$ becomes $\mathbf{\Phi}(\mathbf{X})$

$$\mathbf{\Phi}(\mathbf{X}) = \begin{pmatrix} -1 & \phi(\vec{x}_1)^\top \\ -1 & \phi(\vec{x}_2)^\top \\ \vdots & \vdots \\ -1 & \phi(\vec{x}_N)^\top \end{pmatrix} = \begin{pmatrix} -1 & \phi(\vec{x}_1)_1 & \phi(\vec{x}_1)_2 & \cdots & \phi(\vec{x}_1)_d \\ -1 & \phi(\vec{x}_2)_1 & \phi(\vec{x}_2)_2 & \cdots & \phi(\vec{x}_2)_d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & \phi(\vec{x}_n)_1 & \phi(\vec{x}_n)_2 & \cdots & \phi(\vec{x}_n)_d \end{pmatrix} \tag{4.98}$$

The closed form solution of ridge regression 4.19 with a feature transformation ϕ is given by

$$\vec{w} = (\mathbf{\Phi}(\mathbf{X})^\top \mathbf{\Phi}(\mathbf{X}) + \lambda \mathbb{I})^{-1} \mathbf{\Phi}(\mathbf{X})^\top \vec{y} \tag{4.99}$$

By using a consequence of the *matrix inversion lemma*[26]

$$(\mathbf{UCV} + \mathbf{A})^{-1}\mathbf{UC} = \mathbf{A}^{-1}\mathbf{U}(\mathbf{VA}^{-1}\mathbf{U} + \mathbf{C}^{-1})^{-1} \tag{4.100}$$

where we set

$$\begin{aligned} \mathbf{A} &= \lambda \mathbb{I} & \mathbf{C} &= \mathbb{I} \\ \mathbf{U} &= \mathbf{\Phi}(\mathbf{X})^\top & \mathbf{V} &= \mathbf{\Phi}(\mathbf{X}) \end{aligned}$$

we can rewrite the term $(\mathbf{\Phi}(\mathbf{X})^\top \mathbf{\Phi}(\mathbf{X}) + \lambda \mathbb{I})^{-1} \mathbf{\Phi}(\mathbf{X})^\top$ as

$$\begin{aligned} (\mathbf{\Phi}(\mathbf{X})^\top \mathbf{\Phi}(\mathbf{X}) + \lambda \mathbb{I})^{-1} \mathbf{\Phi}(\mathbf{X})^\top &= \frac{1}{\lambda}\mathbb{I}\mathbf{\Phi}(\mathbf{X})^\top \Big(\mathbf{\Phi}(\mathbf{X})(\frac{1}{\lambda}\mathbb{I})\mathbf{\Phi}(\mathbf{X})^\top + \mathbb{I}\Big)^{-1} \\ &= \mathbf{\Phi}(\mathbf{X})^\top \Big(\frac{1}{\lambda}\mathbb{I}\Big)\Big(\frac{1}{\lambda}\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top + \mathbb{I}\Big)^{-1} \\ &= \mathbf{\Phi}(\mathbf{X})^\top \Big(\Big(\frac{1}{\lambda}\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})\Big)^\top + \mathbb{I}\Big)\Big(\lambda\mathbb{I}\Big)\Big)^{-1} \\ &= \mathbf{\Phi}(\mathbf{X})^\top (\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top + \lambda\mathbb{I})^{-1} \end{aligned} \tag{4.101}$$

By making use of 4.101, we can rewrite 4.99 as

$$\vec{w} = \mathbf{\Phi}(\mathbf{X})^\top (\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top + \lambda\mathbb{I})^{-1}\vec{y} \tag{4.102}$$

[26] also called Woodbury matrix identity - see `https://en.wikipedia.org/wiki/Woodbury_matrix_identity`

and furthermore as

$$\begin{aligned} \vec{w} &= \mathbf{\Phi}(\mathbf{X})^\top \vec{\alpha} \\ &= \sum_i (\vec{\alpha})_i \phi(\vec{x}_i) \end{aligned} \tag{4.103}$$

where $\vec{\alpha} = (\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top + \lambda\mathbb{I})^{-1}\vec{y}$.
If we plug 4.103 into the prediction formula $f(\vec{x}) = \vec{w}^\top\phi(\vec{x})$, we find that

$$\begin{aligned} f(x) &= \vec{w}^\top \phi(\vec{x}) \\ &= \vec{\alpha}^\top \mathbf{\Phi}(\mathbf{X})\phi(\vec{x}) \\ &= \sum_i (\vec{\alpha})_i \phi(\vec{x}_i)^\top \phi(\vec{x}) \end{aligned} \tag{4.104}$$

If we now look at 4.104, we notice that $\vec{w}$ has disappeared and all occurrences of $\phi(\vec{x})$ have the form of $\phi(\vec{x})^\top\phi(\vec{x}')$.
The formulation where $\vec{x}$ only occurs in the expression $\phi(\vec{x})^\top\phi(\vec{x}')$ is called *dual form* and the original one is called *primal form*.

4.11.1. Dual form of ridge regression

The way we took to get the dual form of the solution 4.21 is not the only one. Another way is to convert the primal form of ridge regression 4.19 into its dual form and then solve the resulting dual problem. As we will see, the final result is the same, just the way of obtaining it is different.

The primal problem of ridge regression 4.21 can be stated as

$$\begin{aligned} &\min_{\substack{\vec{r}\in\mathbb{R}^n, \\ \vec{w}\in\mathbb{R}^d}} \vec{r}^\top\vec{r} \\ &\text{s.t.} \\ &\vec{y} = \mathbf{\Phi}(\mathbf{X})\vec{w} + \vec{r} \\ &\vec{w}^\top\vec{w} \leq C \end{aligned} \tag{4.105}$$

However, is not that convenient to work with 4.105 directly. Instead of modelling the regularization of $\vec{w}$ as an inequality constraint, we add it to the objective and introduce a Lagrangian multiplier λ. We can do this, because for every C there exists a λ such that 4.105 and 4.106 are equivalent[27].

$$\begin{aligned} &\min_{\substack{\vec{r}\in\mathbb{R}^n, \\ \vec{w}\in\mathbb{R}^d}} \vec{r}^\top\vec{r} + \lambda\vec{w}^\top\vec{w} \\ &\text{s.t.} \quad \vec{y} = \mathbf{\Phi}(\mathbf{X})\vec{w} + \vec{r} \end{aligned} \tag{4.106}$$

[27] we used a similar argument when working on LASSO

We obtain the Lagrangian function $\mathcal{L}$ of 4.106 by introducing the Lagrange multiplier $\vec{\mu}$

$$\min_{\substack{\vec{r}\in\mathbb{R}^n,\\ \vec{w}\in\mathbb{R}^d,\\ \vec{\mu}\in\mathbb{R}^n}} \mathcal{L}(\vec{r},\vec{w},\vec{\mu}) = \vec{r}^\top\vec{r} + \lambda\vec{w}^\top\vec{w} + \vec{\mu}^\top(\vec{y} - \mathbf{\Phi}(\mathbf{X})\vec{w} - \vec{r}) \tag{4.107}$$

The optimality conditions of 4.107 reveal

$$\begin{aligned} &\nabla_{\vec{w}}\mathcal{L} = \vec{0} \\ &\Leftrightarrow -\mathbf{\Phi}(\mathbf{X})^\top\vec{\mu} + 2\lambda\vec{w} = \vec{0} \\ &\Leftrightarrow \vec{w} = \frac{1}{2\lambda}\mathbf{\Phi}(\mathbf{X})^\top\vec{\mu} \end{aligned} \tag{4.108}$$

$$\begin{aligned} &\nabla_{\vec{r}}\mathcal{L} = \vec{0} \\ &\Leftrightarrow 2\vec{r} - \vec{\mu} = \vec{0} \\ &\Leftrightarrow \vec{r} = \frac{1}{2}\vec{\mu} \end{aligned} \tag{4.109}$$

$$\begin{aligned} &\nabla_{\mu}\mathcal{L} = \vec{0} \\ &\Leftrightarrow \vec{y} - \mathbf{\Phi}(\mathbf{X})\vec{w} - \vec{r} = 0 \\ &\Leftrightarrow \vec{r} = \vec{y} - \mathbf{\Phi}(\mathbf{X})\vec{w} \end{aligned} \tag{4.110}$$

We can minimize 4.107 with respect to $\vec{r}$ and $\vec{w}$ by substituting 4.108 and 4.109

$$\begin{aligned} \min_{\substack{\vec{r}\in\mathbb{R}^n,\\ \vec{w}\in\mathbb{R}^d}} \mathcal{L}(\vec{r},\vec{w},\vec{\mu}) &= \frac{1}{4}\vec{\mu}^\top\vec{\mu} + \frac{1}{4\lambda}\vec{\mu}^\top\mathbf{X}\mathbf{\Phi}(\mathbf{X})^\top\vec{\mu} + \vec{\mu}^\top\vec{y} - \frac{1}{2\lambda}\vec{\mu}^\top\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top\vec{\mu} - \frac{1}{2}\vec{\mu}^\top\vec{\mu} \\ &= -\frac{1}{4}\vec{\mu}^\top\vec{\mu} - \frac{1}{4\lambda}\vec{\mu}^\top\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top\vec{\mu} + \vec{\mu}^\top\vec{y} \end{aligned} \tag{4.111}$$

In order to simply 4.111, we introduce a new variable $\vec{\alpha}$

$$\begin{aligned} \vec{\alpha} &= \frac{1}{2\lambda}\vec{\mu} \\ \Leftrightarrow \vec{\mu} &= 2\lambda\vec{\alpha} \end{aligned} \tag{4.112}$$

By making use of 4.111 and 4.112 we can write down the Lagrange dual $\mathcal{L}_D$ of 4.107

$$\begin{aligned} \mathcal{L}_D(\alpha) &= \min_{\substack{\vec{r}\in\mathbb{R}^n,\\ \vec{w}\in\mathbb{R}^d}} \mathcal{L}(\vec{r},\vec{w},\vec{\mu}) \\ &= -\lambda^2\vec{\alpha}^\top\vec{\alpha} - \lambda\vec{\alpha}^\top\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top\vec{\alpha} + 2\lambda\vec{\alpha}^\top\vec{y} \end{aligned} \tag{4.113}$$

Because strong duality holds for 4.106 (because of Slater's condition - see A.83), we know that optimizing the primal 4.106 is equivalent to maximizing the dual 4.113. Thus

$$\begin{aligned} &\min_{\substack{\vec{r}\in\mathbb{R}^n,\\ \vec{w}\in\mathbb{R}^d}} \vec{r}^\top\vec{r} + \lambda\vec{w}^\top\vec{w} \qquad \text{s.t.} \quad \vec{y} = \mathbf{\Phi}(\mathbf{X})\vec{w} + \vec{r} \\ &\Leftrightarrow \max_{\vec{\alpha}\in\mathbb{R}^N} \mathcal{L}_D(\vec{\alpha}) \end{aligned} \tag{4.114}$$

The optimality conditions of the dual 4.113 yields

$$\begin{aligned} &\nabla_{\vec{\alpha}}\mathcal{L}_D = \vec{0} \\ &\Leftrightarrow -\lambda^2\vec{\alpha} - 2\lambda\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top\vec{\alpha} + 2\lambda\vec{y} = \vec{0} \\ &\Leftrightarrow 2\lambda\vec{y} - 2\lambda(\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top + \lambda\mathbb{I})\vec{\alpha} = \vec{0} \\ &\Leftrightarrow \vec{\alpha} = (\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top + \lambda\mathbb{I})^{-1}\vec{y} \end{aligned} \tag{4.115}$$

By making use of 4.108,4.112 and 4.115, we can find the final formula for computing $\vec{w}$.

$$\begin{aligned} \vec{w} &= \frac{1}{2\lambda}\mathbf{\Phi}(\mathbf{X})^\top\vec{\mu} \\ &= \mathbf{\Phi}(\mathbf{X})^\top\vec{\alpha} \\ &= \mathbf{\Phi}(\mathbf{X})^\top(\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top + \lambda\mathbb{I})^{-1}\vec{y} \end{aligned} \tag{4.116}$$

Note that the formula for computing $\vec{w}$ 4.116 is exactly the same as 4.102 from the previous section.

4.11.2. Kernels

A mapping $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is called *kernel* if there exists a mapping (also called *feature map*) $\phi : \mathcal{X} \mapsto \mathcal{H}$, where $\mathcal{H}$ is an Hilbert space[28], such that

$$\begin{aligned} k(x, x') &= \phi(x)^\top\phi(x') \\ &= \langle\phi(x), \phi(x')\rangle \end{aligned} \tag{4.117}$$

where $\langle\cdot,\cdot\rangle$ is just a different way of writing the scalar product.
Applying this definition to our design matrix $\mathbf{\Phi}(\mathbf{X})$ produces the *Kernel/-Gram matrix*

$$\mathbf{K} = \mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top \tag{4.118}$$

where $\mathbf{K}_{i,j} = \phi(\vec{x}_i)^\top\phi(\vec{x}_j) = k(\vec{x}_i, \vec{x}_j)$.
In plain English: A kernel measures the "similarity" between two given items.

[28] see `https://en.wikipedia.org/wiki/Hilbert_space`

It does so by computing the scalar product of them in a transformed feature space. Recall that the scalar product is related to the angle and distance between two vectors.

By using the definition of a kernel, we can replace all $\phi(\vec{x})^\top \phi(\vec{x}')$ by $k(\vec{x}, \vec{x}')$ and $\mathbf{\Phi}(\mathbf{X})\mathbf{\Phi}(\mathbf{X})^\top$ by $\mathbf{K}$. If we do so we get the following equations

$$\vec{w} = \mathbf{\Phi}(\mathbf{X})^\top (\mathbf{K} + \lambda \mathbb{I})^{-1} \vec{y} \tag{4.119}$$

$$\vec{\alpha} = (\mathbf{K} + \lambda \mathbb{I})^{-1} \vec{y} \tag{4.120}$$

$$f(\vec{x}) = \sum_i (\vec{\alpha})_i k(\vec{x_i}, \vec{x}) \tag{4.121}$$

This replacement is called *kernel trick*. The process of converting the primal form into the dual form and applying th kernel trick is called *kernelization*. The final result (4.119,4.120,4.121) is called *kernel ridge regression* and is *equivalent to ridge regression with feature transformation*.
Note: We can do the same on *linear regression with feature transformation*, the corresponding result is called *kernel linear regression*.

At this point we might wonder *why we are doing all these stuff*, because we started with smth. (ridge regression & feature transformation) and arrived at smth. equivalent (kernel ridge regression). From a modeling perspective there is no difference, we expressed exactly the same thing in just two different ways. The *benefits of kernelized regression* will become clear when we talk a little bit more about *kernels and their benefits*.

Some common kernels on vectors are

1. The linear kernel
$$k(\vec{x}, \vec{x}') = \vec{x}^\top \vec{x}' \tag{4.122}$$
2. The polynomial kernel (with degree d)
$$k(\vec{x}, \vec{x}') = (\vec{x}^\top \vec{x}' + 1)^d \tag{4.123}$$
3. The rbf kernel[29]
$$k(\vec{x}, \vec{x}') = \exp\left(-\frac{||\vec{x} - \vec{x}'||^2}{\gamma}\right) \tag{4.124}$$

[29] very similar to the gaussian kernel

4. The exp-sine-squared kernel[30]

$$k(\vec{x}, \vec{x}') = \exp\left(\frac{-2\sin(\pi\lambda||\vec{x} - \vec{x'}||)}{l}\right)^2 \tag{4.125}$$

By definition there exists a ϕ for all kernels.

4.11.2.1 Example

E.g. the ϕ of the polynomial kernel is the same ϕ which we used in the polynomial feature transformation.
To make this clear, we look at the polynomial kernel with degree 2 and $\mathcal{X} = \mathbb{R}^2$. We can rewrite the degree 2 polynomial kernel 4.123 as

$$\begin{aligned}
k(\vec{x}, \vec{y}) &= (\vec{x}^\top \vec{y} + 1)^2 \\
&= (\vec{x}^\top \vec{y})^2 + 2\vec{x}^\top \vec{y} + 1^2 \\
&= (x_1 y_1 + x_2 y_2)^2 + 2(x_1 y_1 + x_2 y_2) + 1 \\
&= x_1^2 y_1^2 + 2x_1 y_1 x_2 y_2 + x_2^2 y_2^2 + 2x_1 y_1 + 2x_2 y_2 + 1 \\
&= x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + x_2^2 y_2^2 + 2x_1 y_1 + 2x_2 y_2 + 1 \\
&= x_1^2 y_1^2 + \sqrt{2}x_1 x_2 \sqrt{2} y_1 y_2 + x_2^2 y_2^2 + \sqrt{2}x_1 \sqrt{2} y_1 + \sqrt{2}x_2 \sqrt{2} y_2 + \sqrt{1}\sqrt{1} \\
&= (x_1^2, \sqrt{2}x_1 x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{1})(y_1^2, \sqrt{2}y_1 y_2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{1})^\top \\
&= \phi(\vec{x})^\top \phi(\vec{y}) \quad \text{where } \phi(\vec{a}) = (a_1^2, \sqrt{2}a_1 a_2, a_2^2, \sqrt{2}a_1, \sqrt{2}a_2, 1)^\top
\end{aligned} \tag{4.126}$$

We can recognize the ϕ from the polynomial feature transformation.

4.11.2.2 Discussion

Kernels like the rbf kernel are "interesting" because if we try to compute/find the ϕ like we did with the polynomial kernel, we will find that the corresponding ϕ *of the rbf-kernel is infinitely large - it maps into a infinitely dimensional feature space.*

By using a kernel we implicitly use a feature transformation without explicitly computing it. This is great because we do not have to deal with problems like that the ϕ might be high dimensional and/or be complicated to compute or it might be even impossible to compute if it is infinitely large like in the rbf kernel.

[30] useful for periodic functions

So far we always assumed that the *data is represented as a vector* because all methods we discussed so far are working on vectors only. *By using kernels this is no longer necessary.* Because in kernelized methods the data points are only occurring in the form of $k(x, x')$, *we can use data from an arbitrary domain as long as we are able to compute a kernel in this domain.* Indeed there exist kernels for graph data, text data, time series, etc. A kernel for non-vectorial data is called *structure kernel.*

Unfortunately, the kernel approach has some *disadvantages*, too. The main problem with the kernel approach is that it *scales very badly to large data sets.* Constructing and inverting the kernel matrix might be computational expensive and if we want to make a prediction for a query point we have to compute the kernel of the query point and each data point in the training set - which we need to store somewhere.
To deal with this problem we can use an approximation method like the *Nyström approximation method.* A detailed discussion of these methods is not the topic of this book and is usually a topic of an advanced lecture of machine learning. However, if you are interested in these things you can find a lot of information/material about it on the internet.

4.11.2.3 More details on kernels

A *Mercer kernel* (Mercer's Theorem[31]) is a real valued function k that satisfies the following condition

$$\iint g(x)k(x,y)g(y)\mathrm{d}x\mathrm{d}y \geq 0 \quad \forall\, g \in \left\{ g : \mathbb{R} \mapsto \mathbb{R} \text{ where } \int_{-\infty}^{\infty} |g(x)|^2 \mathrm{d}x < \infty \right\} \tag{4.127}$$

Note: A Mercer kernel implies a feature mapping into a Hilbert space.

Because we are working with data points only, we can not use the continuous version 4.127 and instead have to switch to the discrete version of 4.127 which is given as

$$\begin{aligned} &\mathbf{K} \succeq 0 \\ &\text{where } (\mathbf{K})_{i,j} = k(x_i, y_j) \end{aligned} \tag{4.128}$$

In plain English: A function k is a valid (Mercer) kernel if and only if the kernel matrix is symmetric positive-definite.

[31] see `https://en.wikipedia.org/wiki/Mercer%27s_theorem`

Recall that

$$\begin{aligned} &\mathbf{K} \succeq 0 \\ &\Leftrightarrow \\ &\vec{g}^\top \mathbf{K} \vec{g} = \sum_{i,j} (\vec{g})_i (\mathbf{K})_{i,j} (\vec{g})_j \geq 0 \quad \forall \, \vec{g} \in \mathbb{R}^N \end{aligned} \tag{4.129}$$

4.12. Outlook

In this section we learned about linear regression. Linear regression computes the prediction as a weighted linear combination of the input features. We learned about L1/LASSO and L2 regularization, as well as elastic net and robust regression. We observed that L1/LASSO performs feature selection and that robust regression is more robust to outliers in the training data. Next, we turned linear regression into non-linear regression by explicitly applying feature transformations or implicitly using kernels.
In addition, we noticed that linear regression can interpreted as maximum likelihhood under the assumption that the training data have been perturbed by i.i.d. gaussian noise. In this setting, L1/L2 regularization can be interpreted as maximum a posteriori with a laplacian/gaussian prior centered at zero.

Because linear regression is a complex and large topic, we only covered the basics of it. If you want to learn more, we recommend to take a look at [7], [4] and [5]. Furthermore, in chapter 5, we talk about logistic regression which can be interpreted as linear regression for binary classification.

4.13. Exercises

1. Prove that, if the design matrix $\mathbf{X}$ is orthogonal, the linear least squares estimator 4.13 can be written as $\vec{w} = \mathbf{X}^\top \vec{y}$.

2. Prove that the matrix $\mathbf{A} = \mathbf{X}^\top \mathbf{X} + \lambda \mathbb{I}$ for $\lambda > 0, \mathbf{X} \in \mathbb{R}^{n \times m}$ is always invertible.

3. High dimensional regression:
   Prove that the matrix $\mathbf{X}^\top \mathbf{X}$ is *not invertible* if $\mathbf{X} \in \mathbb{R}^{n \times d}$ where $d > n$.

4. Assume that the responses y_i and data points $\vec{x}_i$ are centered. Hence
   $$\bar{y} = \frac{1}{n} \sum_i y_i = 0 \quad \text{and } \bar{x} = \frac{1}{n} \sum_i \vec{x}_i = \vec{0} \tag{4.130}$$
   Prove that the minimizing bias/intercept b of the SSE 4.7 is equal to 0.

5. Univariate linear regression:
   Univariate linear regression denotes one dimensional linear regression, where $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \mathbb{R}$. Thus, we have one predictor variable X and one response variable Y.
   By using the two random variables X and Y for the inputs and outputs, we can write the *linear least squares problem* 4.7 as
   $$\min_{b,w \in \mathbb{R}} \mathbb{E}[(Y - b - wX)^2] \tag{4.131}$$
   where we modeled the bias/intercept b explicitly. Because we are in 1d, the weight vector is a scaler and denoted by w.

   Prove that the solution of 4.131 is given by
   $$\begin{aligned} b &= \mathbb{E}[Y] \\ w &= \frac{\mathrm{Cov}(X, Y)}{\mathrm{Var}(X)} \end{aligned} \tag{4.132}$$
   if $\mathbb{E}[X] = 0$ holds.

6. LASSO from a probabilistic point of view:
   Prove that LASSO is equivalent to using a laplacian prior. Thus
   $$\arg\min_{\vec{w} \in \mathbb{R}^d} \mathrm{SSE}_{\mathcal{D}}(\vec{w}) + \frac{1}{\lambda} \|\vec{w}\|_1 = \arg\max_{\vec{w} \in \mathbb{R}^d} \mathrm{L}_{\mathcal{D}}(\vec{w}) p(\vec{w}) \tag{4.133}$$
   where $\mathrm{L}_{\mathcal{D}}(\vec{w}) = \prod_i \mathcal{N}(y_i \mid \vec{w}^\top \vec{x}_i, \sigma^2)$ and $p(\vec{w}) = \prod_j \mathrm{Lap}\left((\vec{w})_j \mid 0, \lambda\right)$.

7. Weighted linear regression:
   So far we always assumed that all data points are equally likely and important. But sometimes it might be beneficial to focus on some points more than on others. Hence, we are looking for a mechanism to penalize the errors of points differently. We can do so by introducing a weight $d_i \geq 0$ for each point telling us how "important" a specific point is - how much effort should we spend on this sample to get it right?
   By applying this idea to the SSE, we get the so called *Weighted Squared Error* defined as
   $$\text{WSSE} = \sum_i d_i (y_i - \hat{y}_i)^2 \tag{4.134}$$
   where $\hat{y}_i = \vec{w}^\top \vec{x}_i$.
   (a) Compute the gradient of WSSE with respect to $\vec{w}$.
   (b) Rewrite the gradient of WSSE in matrix-vector notation.
   (c) Find a closed form solution for minimizing the WSSE (use the formula from 7b).
   (d) Add L2 regularization to WSSE, i.e. $\text{WSSE}' = \text{WSSE} + \lambda \|\vec{w}\|^2$. Redo tasks 7a, 7b and 7c for WSSE$'$.

8. Prove theorem 3. Do *not* use the second order condition.

9. Prove that the L2 regularized linear least squares estimator is given by 4.21.

10. LASSO on standardized data:
    Show that the update formula for the k-th entry given by 4.74 can be simplified to
    $$(\vec{w})_k = \text{sign}(r_k)(|r_k| - \lambda)^+ \tag{4.135}$$
    if the data set have been standardized (viz. each feature has zero mean and unit variance).

11. Let k_1 and k_2 be valid kernels (see section 4.11.2.3). Prove that the following functions are also valid kernels
    (a) $k_3 = k_1 + k_2$
    (b) $k_3 = \alpha\, k_1$ where $\alpha > 0$
    (c) $k_3 = k_1 \cdot k_2$

Chapter 5

Logistic regression

Logistic regression is a model for classification (see section 1.2). It is *linear regression* applied to a *binary response variable* - in linear regression the response variable was $\mathbb{R}$ which is continuous.
However, in this chapter we discuss a derivation of logistic regression which is motivated by building a classifier that approximates the optimum Bayes classifier (see chapter 2). A more general derivation of logistic regression which is more directly related to linear regression is mentioned in the outlook of this chapter.

5.1. Modeling

We model logistic regression as a particular instance of the Bayes classifier (see chapter 2). Recall from the definition of the Bayes classifier 2.2, that we need to define/approximate $P(Y = 1 \mid \vec{x})$. If we would know the true conditional distribution we would use it and obtain the optimum Bayes classifier as discussed in section 2.1.

In logistic regression we model the conditional probability $P(Y = 1 \mid \vec{x})$ with the *Bernoulli distribution*.
Thus, we assume that for all data points $(\vec{x}, y)$, $\vec{x} \in \mathcal{X}, y \in \mathcal{Y}$ and $P(Y = 1 \mid \vec{x}) = \text{Ber}(Y = 1 \mid p)$ holds or at least that is is an appropriate assumption[1]. Furthermore, we assume that all *data points are independent* from each other[2]. This means that no data point affects any other data point. Be-

[1] **Note:** This is a *crucial assumption*. If this assumption is wrong it might break the model!

[2] **Note:** This is another *crucial assumption*! There exists cases where the independence assumption does not hold - e.g. time series where one point in time is affected by the

cause of this *independence assumption* we do not have to work on the joint probability of all points but we can directly work on the individual probabilities of each data point. This will *make the math a lot easier.*
In plain English: We assume that all data points are *independent identically distributed (i.i.d)* according to the Bernoulli distribution..

Furthermore, we assume that the parameter p of the Bernoulli distribution is not fixed but rather depends on $\vec{x}$. Hence, p can be different for different $\vec{x}$. We write $P(y = 1 \mid \vec{x}) = \text{Ber}(y = 1 \mid p_{\vec{x}})$.
In *logistic regression* we estimate $p_{\vec{x}}$ by using *linear regression*

$$p_{\vec{x}} = \text{sgd}(\vec{w}^\top \vec{x}) \tag{5.1}$$

where sgd is the sigmoid function 5.2, which is a special case of the logistic function, and $\vec{w} \in \mathbb{R}^d$ is the weight vector in the linear regression.

$$\text{sgd}(u) = \frac{1}{1 + \exp(-u)} \tag{5.2}$$

Note: We apply the sigmoid function to make sure that the result of $\vec{w}^\top \vec{x}$ is a valid probability.
This is the reason why the model is called *logistic regression.* We use linear *regression* and a *logistic* function, namely the sigmoid function, to estimate $p_{\vec{x}}$.
Since $p_{\vec{x}}$ is the probability for $y = 1$ given $\vec{x}$, we can write

$$P(Y = 1 \mid \vec{x}, \vec{w}) = p_{\vec{x}} = \text{sgd}(\vec{w}^\top \vec{x}) \tag{5.3}$$

With the previous definitions & assumptions we can estimate the classification rule of the Bayes classifier 2.2 as

$$h(\vec{x}) = \begin{cases} 1 & \text{if } P(Y = 1 \mid \vec{x}, \vec{w}) = \text{sgd}(\vec{w}^\top \vec{x}) > t \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

The next issue is to *find* $\vec{w}$. We use the *maximum likelihood approach* (see section B.13.2) to estimate $\vec{w}$.
Remember that the likelihood enables us to measure how good a given model parameter θ is. In the case of logistic regression the model parameter is $\vec{w}$

previous one.

(θ *becomes* $\vec{w}$). Therefore, we can write down the *(conditional) likelihood* (see B.117) for logistic regression as

$$\begin{aligned} \mathrm{L}_{\mathcal{D}}(\vec{w}) &= \prod_i P(y_i \mid \vec{x}_i, \vec{w}) \\ &= \prod_i \mathrm{Ber}(Y = y_i \mid p_{\vec{x}_i}) \\ &= \prod_i p_{\vec{x}_i}^{y_i} \left(1 - p_{\vec{x}_i}\right)^{1-y_i} \end{aligned} \tag{5.5}$$

where we assume that $\vec{x}_i$ is always known - remember that our goal is to build a model which takes $\vec{x}$ as an input and maps it to an output y - so that we can treat $\vec{x}_i$ as a parameter and we are left with the conditional distribution $P(y_i \mid \vec{x}_i, \vec{w})$ only.
Note that the term $p_{\vec{x}_i}^{y_i}(1 - p_{\vec{x}_i})^{1-y_i}$ is exactly $p_{\vec{x}_i}$ if $y_i = 1$ and exactly $1 - p_{\vec{x}_i}$ if $y_i = 0$.
Next, we plug 5.5 into the definition of the negative-log-likelihood B.120.

$$\begin{aligned} \mathrm{NLL}_{\mathcal{D}}(\vec{w}) &= -\log\Big(\mathrm{L}_{\mathcal{D}}(\vec{w})\Big) \\ &= -\log\left(\prod_i p_{\vec{x}_i}^{y_i}(1 - p_{\vec{x}_i})^{1-y_i}\right) \\ &= -\sum_i \log\left(p_{\vec{x}_i}^{y_i}(1 - p_{\vec{x}_i})^{1-y_i}\right) \\ &= -\sum_i y_i \log\left(p_{\vec{x}_i}\right) + (1 - y_i)\log\left(1 - p_{\vec{x}_i}\right) \\ &= -\sum_i y_i \log\left(\mathrm{sgd}(\vec{w}^\top \vec{x}_i)\right) + (1 - y_i)\log\left(1 - \mathrm{sgd}(\vec{w}^\top \vec{x}_i)\right) \end{aligned} \tag{5.6}$$

In B.124, we defined the finding of the parameter θ as an optimization problem. If we plug 5.6 into B.124, we obtain the *maximum likelihood estimator* (see section B.13.2) for estimating $\vec{w}$.

$$\begin{aligned} \vec{w} &= \arg\min_{\vec{w} \in \mathbb{R}^d}\Big(\mathrm{NLL}_{\mathcal{D}}(\vec{w})\Big) \\ &= \arg\min_{\vec{w} \in \mathbb{R}^d}\left(-\sum_i y_i \log\left(\mathrm{sgd}(\vec{w}^\top \vec{x}_i)\right) + (1 - y_i)\log\left(1 - \mathrm{sgd}(\vec{w}^\top \vec{x}_i)\right)\right) \end{aligned} \tag{5.7}$$

Note: 5.6 is also called *cross entropy* - see section 5.1.1 for details on cross entropy and its relation to information theory.

5.1.1. Cross entropy and information theory

From section B.10 we know that the cross entropy of two discrete probability distributions P_Y and $P_{\hat{Y}}$ is given by

$$\mathrm{H}(P_Y, P_{\hat{Y}}) = -\sum_i P_Y(x_i) \log\Big(P_{\hat{Y}}(x_i)\Big) = \mathrm{H}(P_Y) + \mathrm{D}_{\mathrm{KL}}(P_Y \| P_{\hat{Y}}) \tag{5.8}$$

We observe that 5.8 is equivalent to the negative log-likelihood 5.6 if we set

$$\begin{aligned} &P_Y(x_i) = y_i \\ &P_{\hat{Y}}(x_i) = \mathrm{sgd}(\vec{w}^\top \vec{x}_i) \\ &x_i \in \mathbb{R}^d,\ y_i \in \{0,1\} \quad \forall\, i \end{aligned} \tag{5.9}$$

Because the labels y_i of the data are fixed, the entropy of the label distribution $\mathrm{H}(P_Y)$ is a constant and can be ignored when minimizing the cross entropy. Thus

$$\arg\min_{\vec{w}\in\mathbb{R}^d} \mathrm{NLL}_{\mathcal{D}}(\vec{w}) \Leftrightarrow \arg\min_{\vec{w}\in\mathbb{R}^d} \mathrm{H}(P_Y, P_{\hat{Y}}) \Leftrightarrow \arg\min_{\vec{w}\in\mathbb{R}^d} \mathrm{D}_{\mathrm{KL}}(P_Y, P_{\hat{Y}}) \tag{5.10}$$

where $P_{\hat{Y}}$ depends on $\vec{w}$.

Therefore, we can interpret minimizing the negative log-likelihood as minimizing the Kullback-Leibler divergence[3] of the true labels and the predictions of logistic regression.
In plain English: We assume that there exists a probability distribution which assigns either 1 or 0 to each possible data point $\vec{x} \in \mathbb{R}^d$. We do not know this distribution but we observed some samples $\mathcal{D} = \{(\vec{x}_i, y_i)\}$ from it. We use these samples $\mathcal{D}$ to construct an approximation P_Y of the true distribution. We then try to build a model/distribution $P_{\hat{Y}}$ which approximates/reproduces the distribution P_Y by assigning a probability $\mathrm{sgd}(\vec{w}^\top \vec{x}_i)$ to each observed data point $\vec{x}_i \in \mathbb{R}^d$. Our goal is to make $P_{\hat{Y}}$ as similar as possible to P_Y, where we measure the "distance"/"similarity" between the two distributions with the Kullback-Leibler divergence.

5.1.2. Convexity

Theorem 7 (The loss function in logistic regression is a convex function)**.** *Minimizing the loss function in logistic regression (as stated in 5.7) is a convex optimization problem.*

[3] recall that the Kullback-Leibler divergence is a measure for the distance/dissimilarity of two probability distributions, although it is not a proper metric!

Proof. From section A.2.2 we know that the sum of convex functions is again convex. Therefore, it is sufficient to show convexity of the two functions

$$\begin{gathered} -y_i \log\left(\mathrm{sgd}(\vec{w}^\top \vec{x}_i)\right) \\ \text{and} \\ -(1-y_i)\log\left(1-\mathrm{sgd}(\vec{w}^\top \vec{x}_i)\right) \end{gathered} \tag{5.11}$$

Because of

$$y_i,\ (1-y_i) \in \{0,1\}\ \forall\, i \tag{5.12}$$

we only have to show convexity of

$$\begin{gathered} -\log\left(\mathrm{sgd}(\vec{w}^\top \vec{x}_i)\right) \\ \text{and} \\ -\log\left(1-\mathrm{sgd}(\vec{w}^\top \vec{x}_i)\right) \end{gathered} \tag{5.13}$$

Next, we simplify the two terms:

$$\begin{aligned} -\log\left(1-\mathrm{sgd}(\vec{w}^\top \vec{x}_i)\right) &= -\log\left(1-\frac{1}{1+\exp(-\vec{w}^\top \vec{x}_i)}\right) \\ &= -\log\left(\frac{\exp(-\vec{w}^\top \vec{x}_i)}{1+\exp(-\vec{w}^\top \vec{x}_i)}\right) \\ &= -\log\left(\exp(-\vec{w}^\top \vec{x}_i)\right) + \log\left(1+\exp(-\vec{w}^\top \vec{x}_i)\right) \\ &= \vec{w}^\top \vec{x}_i + \log\left(1+\exp(-\vec{w}^\top \vec{x}_i)\right) \end{aligned} \tag{5.14}$$

$$\begin{aligned} -\log\left(\mathrm{sgd}(\vec{w}^\top \vec{x}_i)\right) &= -\log\left(\frac{1}{1+\exp(\vec{w}^\top \vec{x}_i)}\right) \\ &= -\log(1) + \log\left(1+\exp(-\vec{w}^\top \vec{x}_i)\right) \\ &= \log\left(1+\exp(-\vec{w}^\top \vec{x}_i)\right) \end{aligned} \tag{5.15}$$

From A.2.3, we know that

$$-\log(x) \quad \forall\, x > 0 \tag{5.16}$$

is a non decreasing convex function and

$$-\vec{w}^\top \vec{x} \quad \forall\, \vec{w}, \vec{x} \in \mathbb{R}^d \tag{5.17}$$

is a convex function in $\vec{w}$.
Because the sum of convex functions is again convex, it is sufficient to show that

$$-\log\left(1+\exp(-\vec{w}^\top \vec{x}_i)\right) \tag{5.18}$$

is a convex function in $\vec{w}$.
Because of 5.16 and the chain rule for non decreasing convex functions, we only have to prove that

$$1 + \exp(-\vec{w}^\top \vec{x}_i) \tag{5.19}$$

is a convex function in $\vec{w}$, since it always holds that $1 + \exp(-\vec{w}^\top \vec{x}_i) > 0$.
Because the sum of convex functions is a convex function and because of A.2.3, we know that 5.19 is indeed a convex function.

We conclude that $-\log\big(\text{sgd}(\vec{w}^\top \vec{x}_i)\big)$ and $-\log\big(1 - \text{sgd}(\vec{w}^\top \vec{x}_i)\big)$ are convex functions.

Because $\vec{w} \in \mathbb{R}^d$ and $\mathbb{R}^d$ is known to be a convex set, we conclude that 5.7 is a convex optimization problem. □

5.2. Optimization

The next step in computing $\vec{w}$ is to solve the optimization problem 5.7.
In the previous section we proved that 5.7 is a convex optimization problem, which allows us to use our toolkit for tackling convex optimization problems (see section A.3).

However, the analytical way of finding the minimum of $\text{NLL}_\mathcal{D}(\vec{w})$ by computing the first derivative $\nabla_{\vec{w}}\text{NLL}_\mathcal{D}(\vec{w})$ setting it to 0 and solve for $\vec{w}$ (e.g. $\nabla_{\vec{w}}\text{NLL}_\mathcal{D}(\vec{w}) = 0 \Leftrightarrow \vec{w} = \dots$) is not possible in logistic regression.
In plain English: There is no closed form solution in logistic regression!
Instead we have to use an iterative algorithm to solve 5.7. We use the *gradient descent* algorithm[4](see section A.3.2), which iteratively computes the optimum of a function by *using the gradient of the function*. Therefore, we need to compute the gradient of 5.6 with respect to $\vec{w}$.

$$\begin{aligned}
\nabla_{\vec{w}}\text{NLL}_\mathcal{D}(\vec{w}) &= \nabla_{\vec{w}} \left(-\sum_i y_i \log\big(\text{sgd}(\vec{w}^\top \vec{x}_i)\big) + (1 - y_i) \log\big(1 - \text{sgd}(\vec{w}^\top \vec{x}_i)\big) \right) \\
&= -\sum_i y_i \nabla_{\vec{w}} \log\big(\text{sgd}(\vec{w}^\top \vec{x}_i)\big) + (1 - y_i) \nabla_{\vec{w}} \log\big(1 - \text{sgd}(\vec{w}^\top \vec{x}_i)\big) \\
&= -\sum_i y_i \frac{1}{\text{sgd}(\vec{w}^\top \vec{x}_i)} \nabla_{\vec{w}} \text{sgd}(\vec{w}^\top \vec{x}_i) + (1 - y_i) \frac{1}{1 - \text{sgd}(\vec{w}^\top \vec{x}_i)}
\end{aligned}$$

[4]Note that this is not the only algorithm we could use.
There exist other and more sophisticated algorithms for solving this kind of problem - see section 5.2.1

$$
\begin{aligned}
&\nabla_{\vec{w}} \left(1 - \operatorname{sgd}(\vec{w}^\top \vec{x}_i)\right) \\
&= -\sum_i y_i \frac{1}{\operatorname{sgd}(\vec{w}^\top \vec{x}_i)} \operatorname{sgd}(\vec{w}^\top \vec{x}_i) \left(1 - \operatorname{sgd}(\vec{w}^\top \vec{x}_i)\right) \nabla_{\vec{w}} \vec{w}^\top \vec{x}_i + \\
&\quad (1 - y_i) \frac{1}{1 - \operatorname{sgd}(\vec{w}^\top \vec{x}_i)} \left(-\operatorname{sgd}(\vec{w}^\top \vec{x}_i)\right) \left(1 - \operatorname{sgd}(\vec{w}^\top \vec{x}_i)\right) \nabla_{\vec{w}} \vec{w}^\top \vec{x}_i \\
&= -\sum_i y_i \frac{\operatorname{sgd}(\vec{w}^\top \vec{x}_i) \left(1 - \operatorname{sgd}(\vec{w}^\top \vec{x}_i)\right) \vec{x}_i}{\operatorname{sgd}(\vec{w}^\top \vec{x}_i)} + (1 - y_i) \\
&\quad \frac{\left(-\operatorname{sgd}(\vec{w}^\top \vec{x}_i)\right) \left(1 - \operatorname{sgd}(\vec{w}^\top \vec{x}_i)\right) \vec{x}_i}{1 - \operatorname{sgd}(\vec{w}^\top \vec{x}_i)} \\
&= -\sum_i y_i \left(1 - \operatorname{sgd}(\vec{w}^\top \vec{x}_i)\right) \vec{x}_i + (1 - y_i) \left(-\operatorname{sgd}(\vec{w}^\top \vec{x}_i)\right) \vec{x}_i \\
&= -\sum_i y_i \vec{x}_i - y_i \operatorname{sgd}(\vec{w}^\top \vec{x}_i) \vec{x}_i - \operatorname{sgd}(\vec{w}^\top \vec{x}_i) \vec{x}_i + y_i \operatorname{sgd}(\vec{w}^\top \vec{x}_i) \vec{x}_i \\
&= -\sum_i y_i \vec{x}_i - \operatorname{sgd}(\vec{w}^\top \vec{x}_i) \vec{x}_i \\
&= \sum_i -y_i \vec{x}_i + \operatorname{sgd}(\vec{w}^\top \vec{x}_i) \vec{x}_i \\
&= \sum_i \operatorname{sgd}(\vec{w}^\top \vec{x}_i) \vec{x}_i - y_i \vec{x}_i \\
&= \sum_i \left(\operatorname{sgd}(\vec{w}^\top \vec{x}_i) - y_i\right) \vec{x}_i
\end{aligned} \tag{5.20}
$$

where we made use of $\nabla_u \operatorname{sgd}(u) = \operatorname{sgd}(u)\,(1 - \operatorname{sgd}(u))$.
We can rewrite 5.20 in matrix-vector notation as

$$
\begin{aligned}
\nabla_{\vec{w}} \mathrm{NLL}_{\mathcal{D}}(\vec{w}) &= \sum_i \left(\operatorname{sgd}(\vec{w}^\top \vec{x}_i) - y_i\right) \vec{x}_i \\
&= \mathbf{X}^\top (\vec{\sigma} - \vec{y}) \\
&= \mathbf{X}^\top \vec{\sigma} - \mathbf{X}^\top \vec{y}
\end{aligned} \tag{5.21}
$$

where $(\vec{\sigma})_i = \operatorname{sgd}(\vec{w}^\top \vec{x}_i)$, $\mathbf{X} = \begin{pmatrix} (\vec{x}_1)_1 & (\vec{x}_1)_2 & \cdots & (\vec{x}_1)_d \\ (\vec{x}_2)_1 & (\vec{x}_2)_2 & \cdots & (\vec{x}_2)_d \\ \vdots & \vdots & \ddots & \vdots \\ (\vec{x}_n)_1 & (\vec{x}_n)_2 & \cdots & (\vec{x}_n)_d \end{pmatrix}$[5] and $\vec{y} = (y_1, y_2, \ldots, y_n)^\top$.
Note that the expression $\mathbf{X}^\top \vec{y}$ is a constant, which can/should be precomputed because it does not depend on $\vec{w}$.

[5] note that we could add a column of 1s for the hidden bias

The gradient descent algorithm for iteratively computing $\vec{w}$ is described in Algorithm 4. As discussed in section A.3.2, the algorithm needs the *gradient*,

Algorithm 4 Logistic regression - Gradient descent algorithm

1: $\vec{w} = \vec{0}$ ▷ Initialize $\vec{w}$
2: **repeat**
3: $\quad \vec{w} = \vec{w} - \eta \nabla_{\vec{w}} \mathrm{NLL}_{\mathcal{D}}(\vec{w})$ ▷ Update $\vec{w}$
4: **until** convergence

the *learning rate* η and a *convergence criterion*.

5.2.1. 2. Order methods

First, we compute the second derivative (the hessian) of 5.6

$$\begin{aligned}
\left(\boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{w}}}\mathbf{NLL}_{\boldsymbol{\mathcal{D}}}\right)_{k,j} &= \frac{\partial^2 \mathrm{NLL}_{\mathcal{D}}}{\partial(\vec{w})_k \, \partial(\vec{w})_j} \\
&= \frac{\partial}{\partial(\vec{w})_j} \sum_i \mathrm{sgd}(\vec{w}^\top \vec{x}_i)(\vec{x}_i)_k - y_i(\vec{x}_i)_k \\
&= \sum_i \frac{\partial \, \mathrm{sgd}(\vec{w}^\top \vec{x}_i)}{\partial \vec{w}^\top \vec{x}_i} \frac{\partial \vec{w}^\top \vec{x}_i}{\partial(\vec{w})_j} (\vec{x}_i)_k \\
&= \sum_i \mathrm{sgd}(\vec{w}^\top \vec{x}_i)(1 - \mathrm{sgd}(\vec{w}^\top \vec{x}_i))(\vec{x}_i)_j (\vec{x}_i)_k
\end{aligned} \tag{5.22}$$

Furthermore, we can rewrite 5.22 in matrix-vector notation as

$$\boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{w}}}\mathbf{NLL}_{\boldsymbol{\mathcal{D}}} = \mathbf{X}^\top \boldsymbol{\Sigma} \mathbf{X} \tag{5.23}$$

where $\boldsymbol{\Sigma} = \mathrm{diag}\Big(\mathrm{sgd}(\vec{w}^\top \vec{x}_i)(1 - \mathrm{sgd}(\vec{w}^\top \vec{x}_i))\Big)$.

We can then use Newton's method A.3.4 to iteratively compute the best $\vec{w}$. Newton's method for logistic regression is described in Algorithm 5, where again η is the step size.

Note: Usually, Newton's method is *not used* in practice because constructing and inverting the hessian might be computational expensive or impossible (e.g. the hessian might not be invertible). Instead of Newton's method, Quasi-Newton methods like L-BFGS A.3.5.2 or more recent algorithms like SAGA are used.

Algorithm 5 Logistic regression - Newton's method

1: $\vec{w} = \vec{0}$ ▷ Initialize $\vec{w}$
2: **repeat**
3: $\quad \vec{w} = \vec{w} - \eta(\mathbf{X}^\top \mathbf{\Sigma} \mathbf{X})^{-1} \nabla_{\vec{w}} \mathrm{NLL}_{\mathcal{D}}(\vec{w})$ ▷ Update $\vec{w}$
4: **until** convergence

5.3. Separating hyperplane

Because *logistic regression is a linear classifier*, the *points are separated by a hyperplane*. For instance in Fig. 5.1 logistic regression is applied to a simple 2D data set. As we can see, we get a linear decision boundary - the data points are separated by a line[6]. In Fig. 5.1 the probabilities are visualized, too. Recall that the formula for classification 5.4 is given by

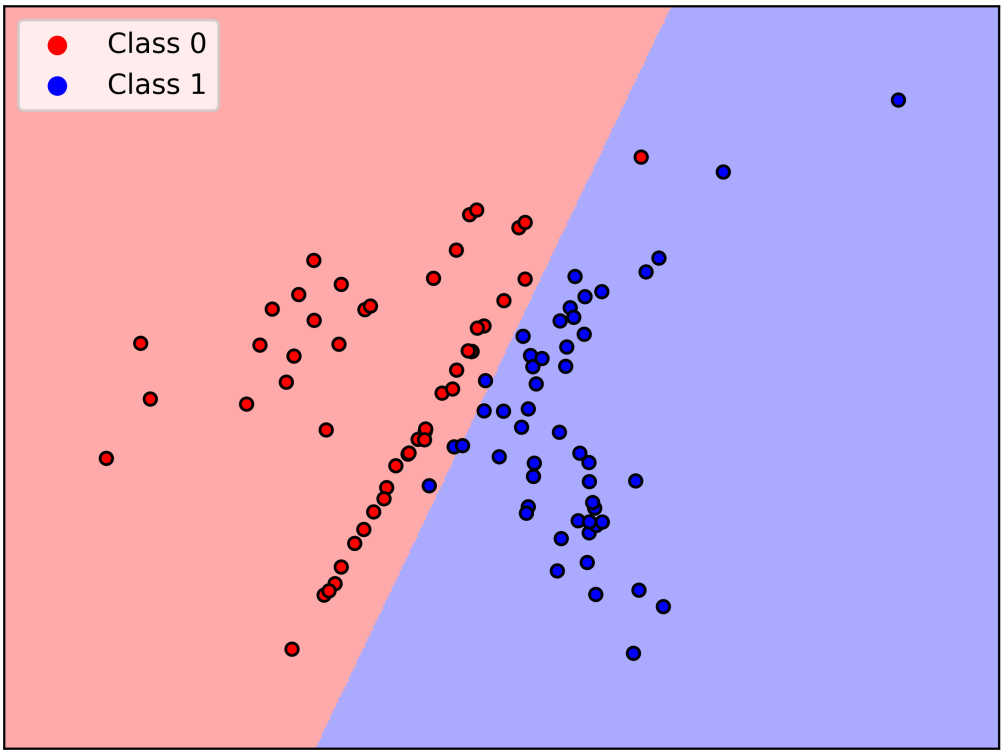


Figure 5.1: Fitted logistic regression model

[6] a line is the 2D case of a hyperplane

$$h(\vec{x}) = \begin{cases} 1 & \text{if } P(Y = 1 \mid \vec{x}, \vec{w}) = \text{sgd}(\vec{w}^\top \vec{x}) > t \\ 0 & \text{otherwise} \end{cases} \tag{5.24}$$

If we set the discrimination threshold $t = 0.5$ we classify every $\vec{x}$ for which $\vec{w}^\top \vec{x} > 0$ as class 1 and otherwise ($\vec{w}^\top \vec{x} \leq 0$) as class 0. This is the case because the sigmoid function (see Fig. 5.2) intersects the y axis at 0.5.
If we set $t = 0.5$ and *do not care about probabilities/confidences* but about the classification result (0 or 1) only, we can rewrite the classification rule 5.4 as

$$h(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w}^\top \vec{x} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.25}$$

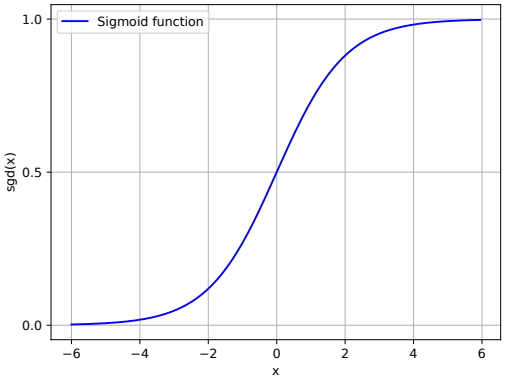


Figure 5.2: Sigmoid function

In 5.25 the decision boundary is given by the hyperplane orthogonal to $\vec{w}$. See Fig. 5.3 for a short outline on the mathematics of a hyperplane.

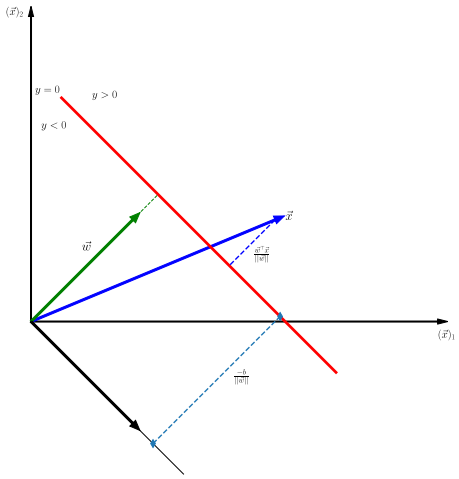


Figure 5.3: Classification with a hyperplane

5.4. Feature transformation, regularization & kernelization

Because logistic regression is an instance of a *general linear regression model*, all the extensions to linear regression like feature transformation, regularization (L1 and L2), kernelization and even the Bayesian version of linear regression can be carried over to logistic regression.

As discussed in section 4.4, feature transformation can be used on all models by simply replacing all x by $\phi(x)$ where ϕ is some feature transformation. We can think of it that we simply apply the feature transformation to the input before feeding it to any model.

Like in linear regression, we can add a regularization term (see sections 4.5 and 4.8) to the cost function of logistic regression 5.7.
In the case of L2 regularization, the cost function 5.7 becomes

$$\vec{w} = \underset{\vec{w} \in \mathbb{R}^d}{\arg\min}\ \mathrm{NLL}_{\mathcal{D}}(\vec{w}) + \lambda \|\vec{w}\|_2^2 \tag{5.26}$$

where $\lambda > 0$ is the regularization strength.
Analogous for the L1 regularization (results in a sparse weight vector $\vec{w}$)

$$\vec{w} = \underset{\vec{w} \in \mathbb{R}^d}{\arg\min}\ \mathrm{NLL}_{\mathcal{D}}(\vec{w}) + \lambda \sum_j |(\vec{w})_j| \tag{5.27}$$

From the *representer theorem*[7] we know that the weight vector $\vec{w}$[8] can be written as a linear combinations of the inputs $\vec{x}_i$. Thus, $\vec{w} = \sum_i \alpha_i \vec{x}_i$ for some specific α_i.
By making use of this linear combination and applying a feature transformation ϕ to the $\vec{x}_i$, we can obtain a kernelized version of logistic regression which is called *kernelized logistic regression* (see exercise 4).

The Bayesian version of logistic regression is called *Bayesian logistic regression*, but in contrast to Bayesian linear regression it is always intractable to solve exactly - recall that Bayesian linear regression was tractable if we used the "right" distributions. See [2] for more information about Bayesian logistic regression including some useful approximation methods.

5.5. Outlook

Students with a background in statistics might already have heard about logistic regression as an instance of the *GLM (Generalized Linear Model)*. In logistic regression we use the Bernoulli distribution because it is a natural choice for modeling binary variables. However, if we have variables with different constraints (e.g. positive numbers, integers, ...) we have to choose a different distribution. For instance if our response variable is *count data*, we might model it by using the *poisson distribution* which gives rise to the *poisson regression* (see exercise 6). The derivation is exactly the same as in logistic regression and should be accessible to you if you understood the material of this chapter.

[7] e.g. see `http://cs229.stanford.edu/extra-notes/representer-function.pdf`

[8] in linear regression as well as in logistic regression and many other models which have a special kind of loss function

5.6. Exercises

1. Compute the gradient of logistic regression with L2 regularization.

$$\nabla_{\vec{w}} \left(\mathrm{NLL}_{\mathcal{D}}(\vec{w}) + \lambda \|\vec{w}\|_2^2 \right) = ... \tag{5.28}$$

   where $\lambda > 0$ is the regularization strength.

2. Let $\vec{w}$ and b be a fitted logistic regression model for the 2 dimensional space ($\mathbb{R}^2$). From section 5.3 we know that logistic regression results in a linear separation - meaning that the data points are separated by a hyper plane. Rewrite the decision boundary for the given model as a function $f(x) = \cdots$ where $x \in \mathbb{R}$, such that the decision boundary/line (2d version of hyperplane) is plotted by plotting $f(x)$.
   Hint: For all points $\vec{x} \in \mathbb{R}^2$ on the decision boundary, it holds that $\vec{w}^\top \vec{x} + b = 0$.

3. Prove that the following cost function of logistic regression is equivalent to the one defined in 5.7.

$$\sum_i \log \left(1 + \exp(-y_i \vec{w}^\top \vec{x}_i) \right) \tag{5.29}$$

   where $y_i \in Y = \{-1, 1\}$.
   Note: The label 0 is replaced by -1.
   Hint: $\mathrm{sgd}(u) = \frac{1}{1+\exp(-u)} = \frac{\exp(u)}{1+\exp(u)}$

4. Derive a kernelized version of logistic regression.
   Hint: Rewrite the weight vector $\vec{w}$ as a linear combination of the training points and apply a feature transformation to all $\vec{x}$

5. Prove that the likelihood function in logistic regression

$$\mathrm{L}_{\mathcal{D}}(\vec{w}) = \prod_i \mathrm{sgd}(\vec{w}^\top \vec{x}_i)^{y_i} (1 - \mathrm{sgd}(\vec{w}^\top \vec{x}_i))^{1-y_i} \tag{5.30}$$

   is *not a convex function* in $\vec{w}$.

6. Poisson regression
   The *Poisson distribution*[9] is given by

$$P(y \mid \lambda) = \frac{\lambda^y \exp(-\lambda)}{y!} \tag{5.31}$$

[9] see `https://en.wikipedia.org/wiki/Poisson_distribution`

where $y \in \mathbb{N}$ and $\lambda \in \mathbb{R}_+$.
The Poisson distribution is characterized by the positive scalar λ and is useful for modeling situations where the result/outcome are natural numbers.

Poisson regression uses the Poisson distribution for modeling a *regression on the natural numbers* $\mathbb{N}$ - Recall that we used the Normal distribution for modeling a regression on the real numbers $\mathbb{R}$, and the Bernoulli distribution for modeling a regression on the numbers $\{0, 1\}$.
In plain English: We use the poisson regression for computing a regression $f : \mathcal{X} \mapsto \mathcal{Y}$ where $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \mathbb{N}$.

Similar to logistic regression, we estimate the parameter λ by linear regression. Because λ can not be negative, we use the exp function as a link function. Thus

$$\lambda = \exp(\vec{w}^\top \vec{x}) \tag{5.32}$$

Write down the log-likelihood of poisson regression and compute its gradient with respect to $\vec{w}$.

Chapter 6

Tree based models

6.1. Decision trees

6.1.1. Model

A *decision tree* (also called *classification tree*) is a classifier which partitions the data space into a set of non-overlapping rectangles $\{R_j\}$ and a class $c_j \in \mathcal{Y}$ is assigned to each rectangle R_j. When we have to make a prediction for a new input x, we compute the rectangle R_j in which the input x is located and output the corresponding class c_j. See Fig. 6.1.1 for an illustration.

Figure 6.1: Decision tree splits the data space into non-overlapping rectangles and assigns a prediction to each rectangle

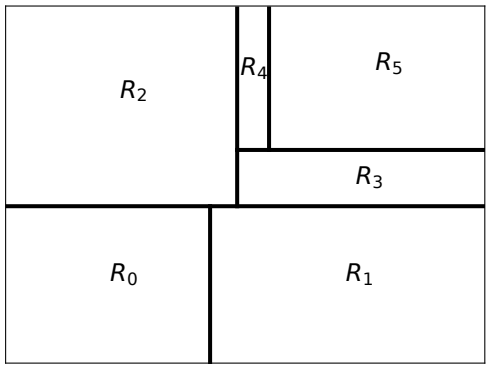


Figure 6.2: Data space is split into non-overlapping rectangles

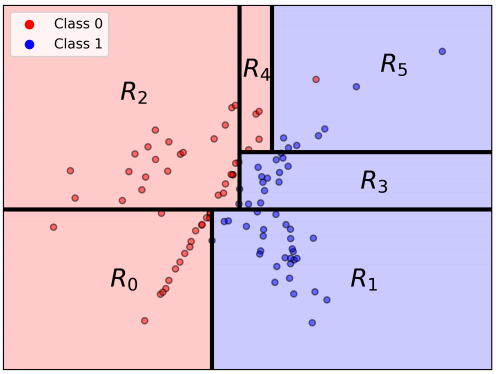


Figure 6.3: A prediction is assigned to each rectangle

We can write down the classification rule of the decision tree classifier as

$$h(x) = \sum_j c_j \mathbb{1}(x \in R_j) \tag{6.1}$$

where j denotes the j-th rectangle R_j.

We can interpret a decision tree as a *binary tree* if we interpret the construction/definition of the rectangles as a binary tree: Starting from the root node, we recursively splitting the data space into non-overlapping rectangles by *splitting a variable*. The term "splitting a variable" means that we set a threshold and put all data points that are below this threshold into the left subtree and everything that is greater or equal to the threshold into the right subtree. Every path from the root node to a leaf specifies a rectangle R_j.
See Fig. 6.4 for an illustration.

6.1.2. Fitting

Unfortunately, fitting/contructing an optimal decision tree is hard - the number of possible trees is exponential and we do not know any efficient strategy for searching in the set of possible trees. Therefore, we use a heuristic for constructing a decision tree.

There exist many different heuristics/algorithms for constructing decision trees. However, most (or even all) algorithms use a greedy strategy for growing the tree. That is, we start at the root (no partition at all) and repeatedly constructing child nodes by splitting a variables until some kind of stopping criterion is fulfilled. A common stopping criterion is the depth of the tree. To determine the variable to split on next, we compute for each variable the split and select the variable (including the corresponding split) which minimizes an impurity measurement like entropy or gini impurity - in practice it usually does not matter which impurity measurement is used, although people sometimes prefer gini impurity over entropy because it is simpler to compute (there is no log in gini impurity).

The *entropy* B.95 of a discrete probability distribution Y is defined as

$$\mathrm{H}(Y) = -\sum_j p_j \log(p_j) \tag{6.2}$$

where p_j denotes the frequency of the j-th item and approximates $P(Y = j)$.

The *gini impurity* of a discrete probability distribution is defined as

$$\text{Gini}(Y) = 1 - \sum_j p_j \tag{6.3}$$

where p_j denotes the frequency of the j-th item and approximates $P(Y = j)$.

Gini impurity and entropy a plotted in Fig. 6.7. Note that they both look very similar and have the same maximum and minimum.

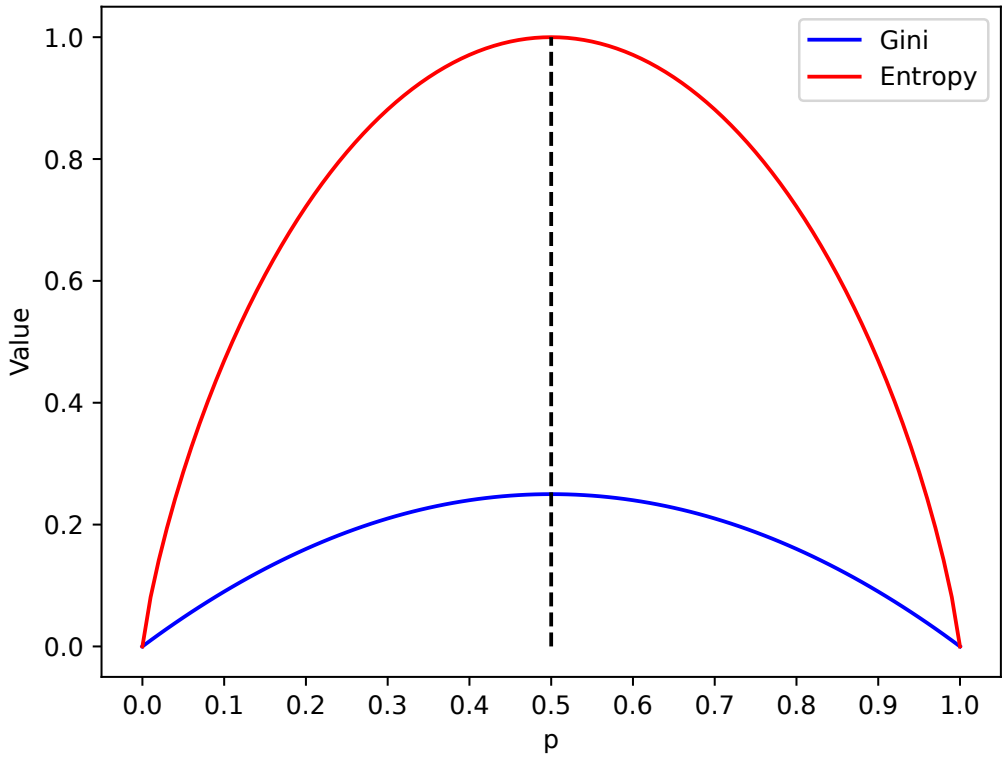


Figure 6.7: Gini impurity vs. Entropy

The pseudocode for greedily fitting a decision tree is described in Algorithm 6.

Algorithm 6 Greedily building a decision tree

Input: Data points $\mathcal{D} = \{(x_i, y_i)\}$ and a stopping criterion (e.g. maximum depth)

Output: Decision tree with root node rootNode

```
rootNode = {left : None, right : None}                    ▷ Initialize root node
splitNode(rootNode, D)                                    ▷ Build the tree.

function SPLITNODE(node, D_node)
    if stoppingCriterion(node, D_node) then
        node.prediction = majorityVote(D_node)            ▷ Assign a prediction to the leaf.
    else
        findMinImpuritySplit(D_node)                      ▷ Find feature to split on.
        node.left, node.right, D_left, D_right = split(node, D)   ▷ Split on this feature.
        splitNode(node.left, D_left)                      ▷ Recursively split the children.
        splitNode(node.right, D_right)
    end if
end function
```

There are many extensions to this procedure of greedily growing the tree. A very common extension is *pruning*. In pruning we first greedily grow the tree and afterwards remove some branches to reduce the comlexity of the final tree - useful for generalization. There are many different strategies for selecting "not so important" branches which we then remove.

It turns out that decision trees are very sensitive to its depth. If the tree is to deep, we overfit the training data while we underfit it if it is not deep enough - see Fig. 6.9 and 6.1.2 for examples.

Figure 6.4: Decision tree applied to a data set

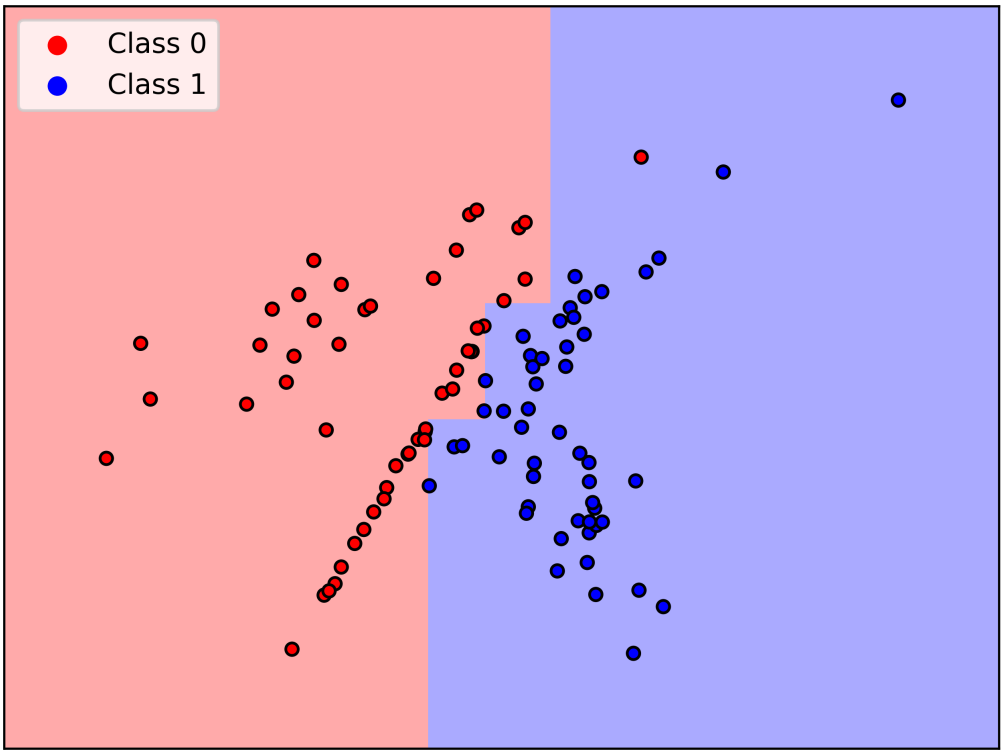


Figure 6.5: Decision boundary of the decision tree

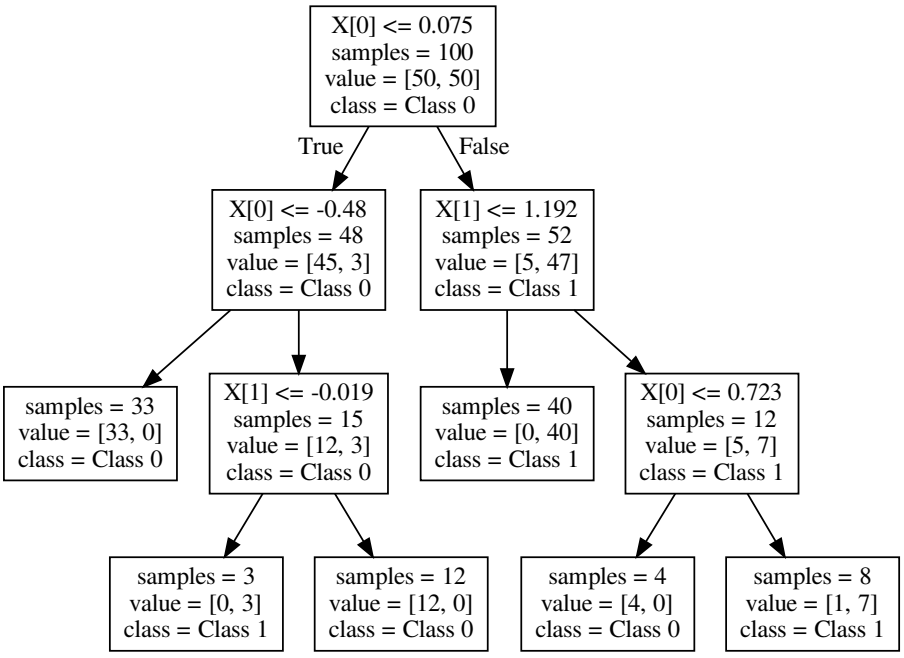


Figure 6.6: Decision tree as a binary tree

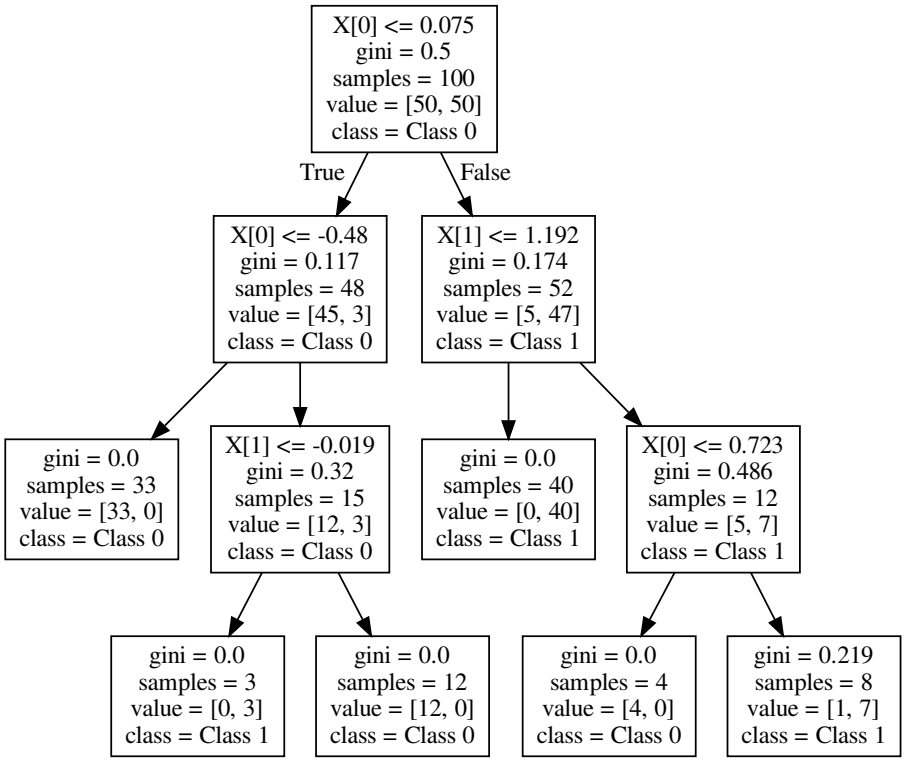


Figure 6.8: Decision tree - binary tree with gini

Figure 6.9: Decision tree - underfitting ⇒ overfitting

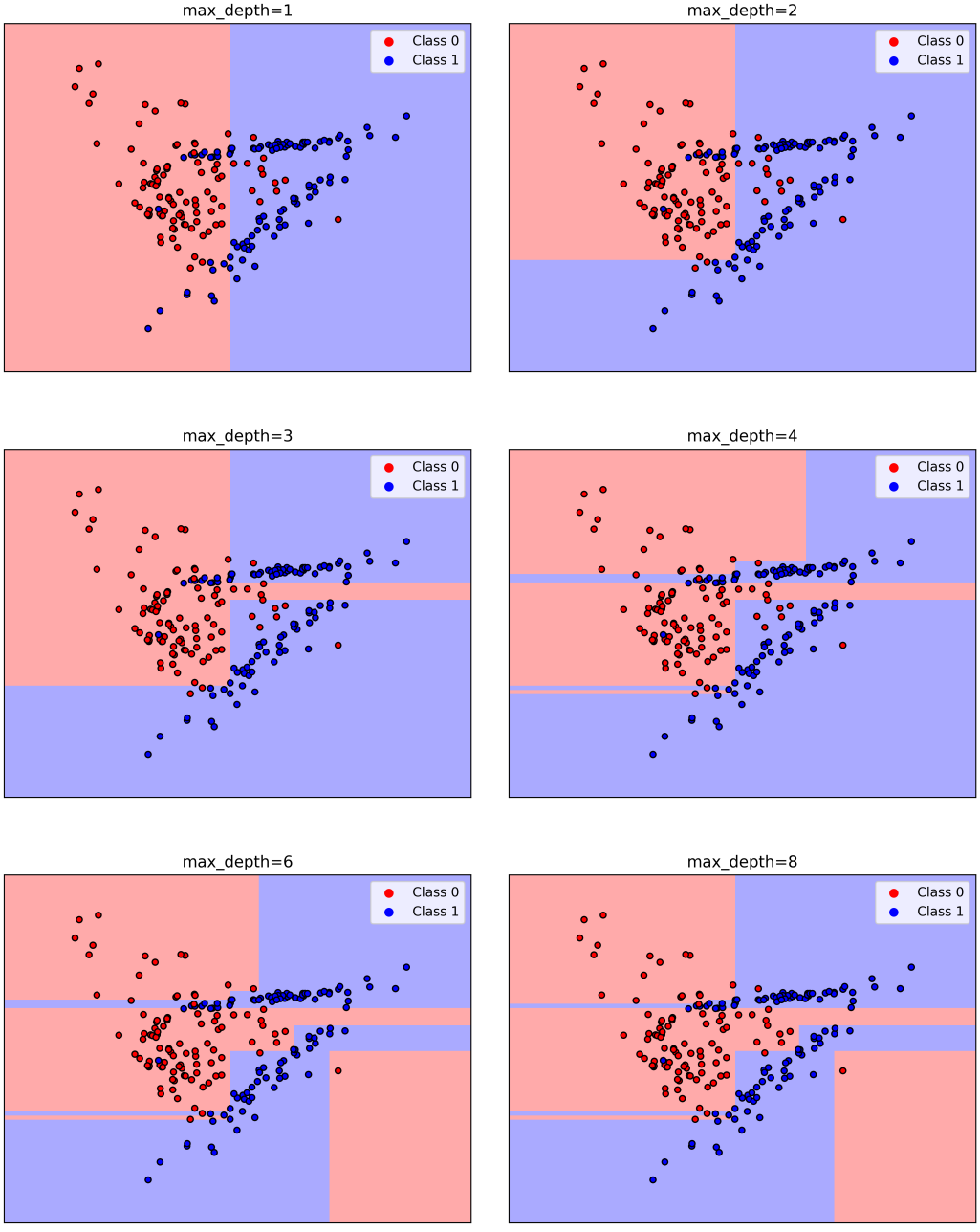


Figure 6.10: Underfitting: max_depth = 1

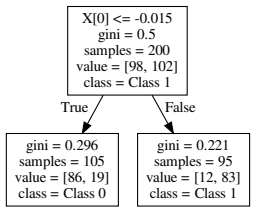


Figure 6.11: max_depth = 4

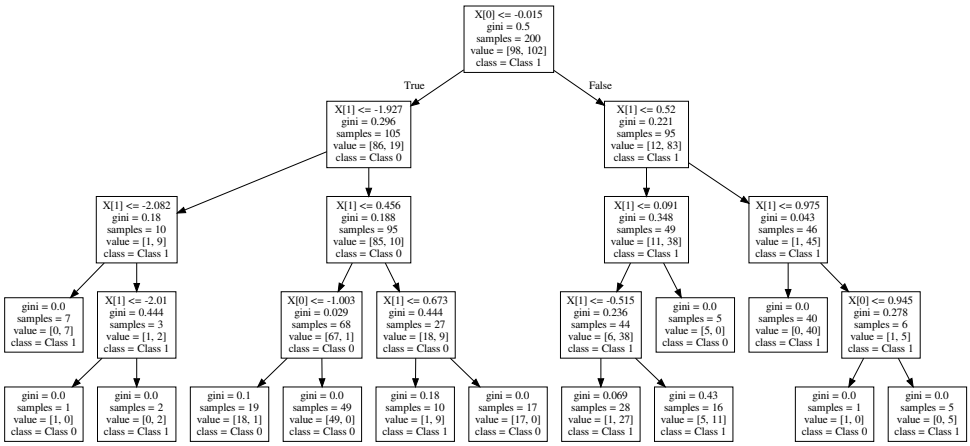
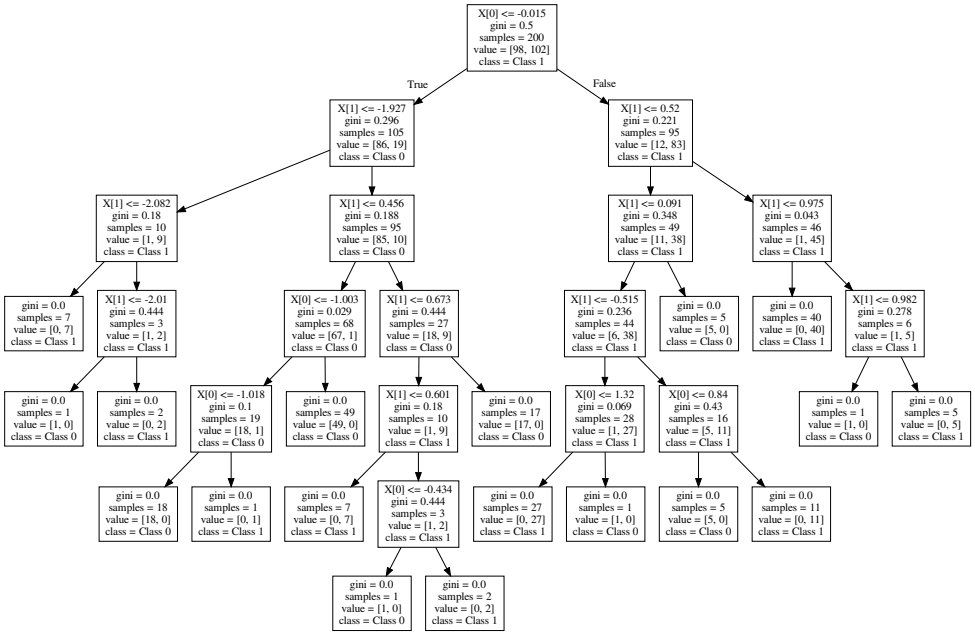


Figure 6.12: Overfitting: max_depth = 8



6.2. Regression trees

A *regression tree* is the analogon of the decision tree for regression.
Like in the decision tree, the data space is partitioned into non-overlapping rectangles which can be visualized/interpreted as a binary tree - like in the decision tree, each node splits a variable. Similar to the decision tree, we assign a constant output/prediction to each rectangle/leaf which results in a step function for regression.
See Fig. 6.13 for an illustration of a regression tree.

6.2.1. Fitting

Fitting a regression tree is more or less the same as fitting a decision tree. The only difference is the used impurity measuremnt and the way a prediction is assigned to each leaf/rectangle.
Entropy and gini impurity are used in decision trees, whereas the mean squared error is often used as an "impurity measurement" in regression trees. In regression trees we can compute the prediction of a leaf by avergaing the output of all training samples assiged to this leaf.

An example of a regression tree, with the mean squared error as in impurity measurement, is shown in Fig. 6.15.

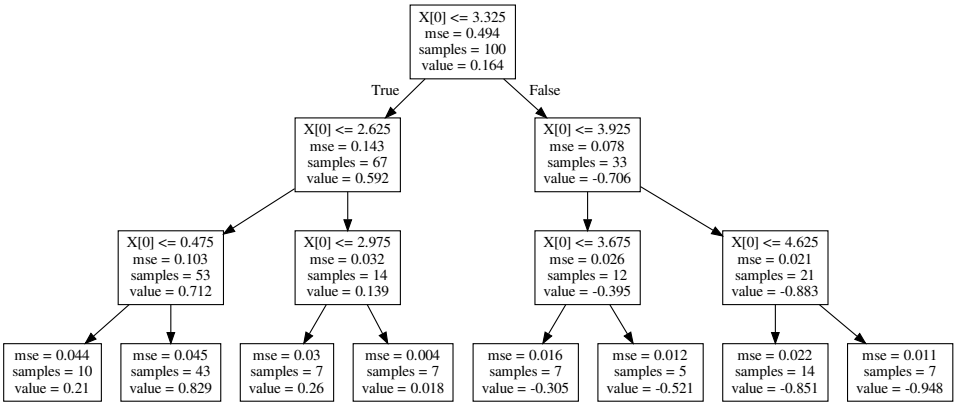


Figure 6.15: Regression tree - binary tree with MSE

Figure 6.13: Regression tree applied to a data set

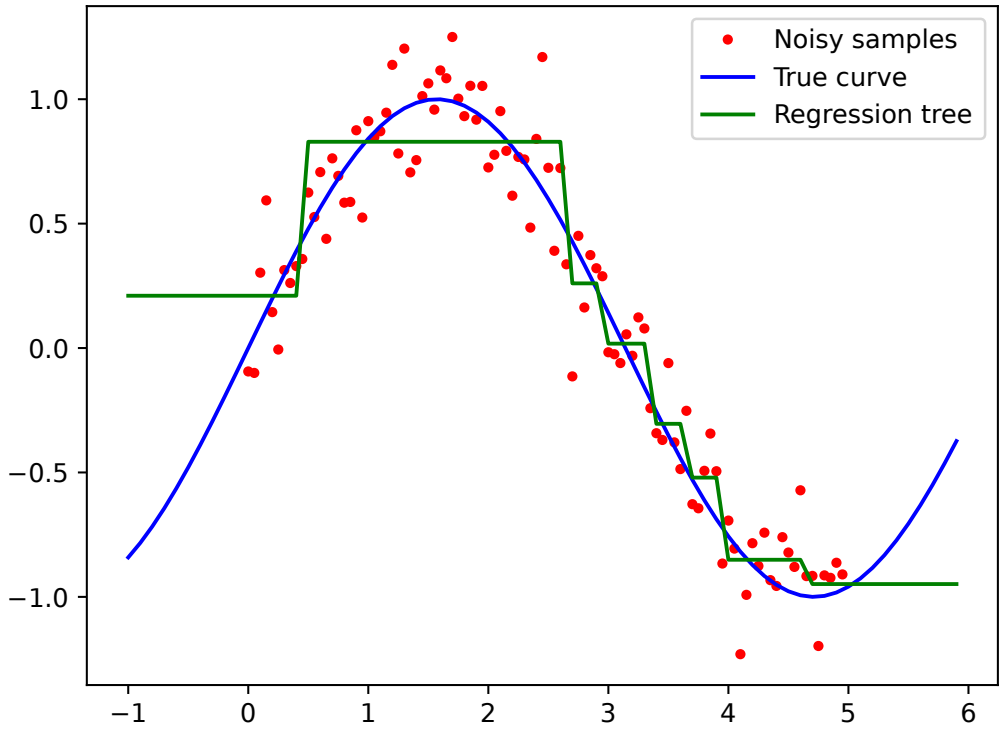
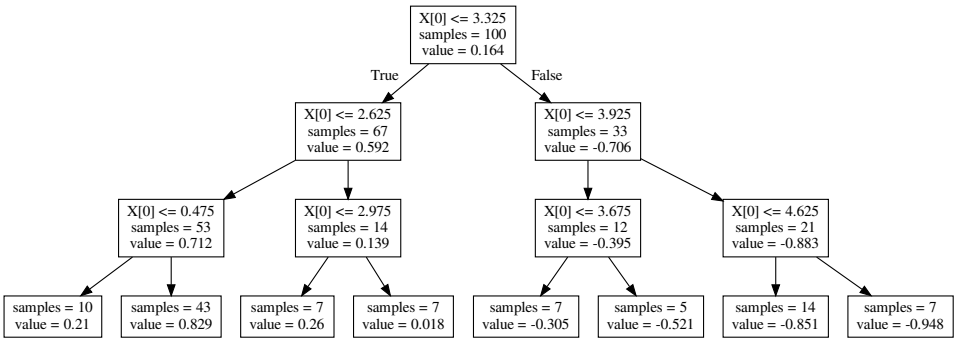


Figure 6.14: Regression tree as a binary tree



Similar to decision trees, a regression tree is very sensitive to its depth. If the tree is to deep, we obtain overfitting while we obtain underfitting if it is not deep enough - see Fig. 6.16 and 6.2.1 for examples.

Figure 6.16: Regression tree - underfitting ⇒ overfitting

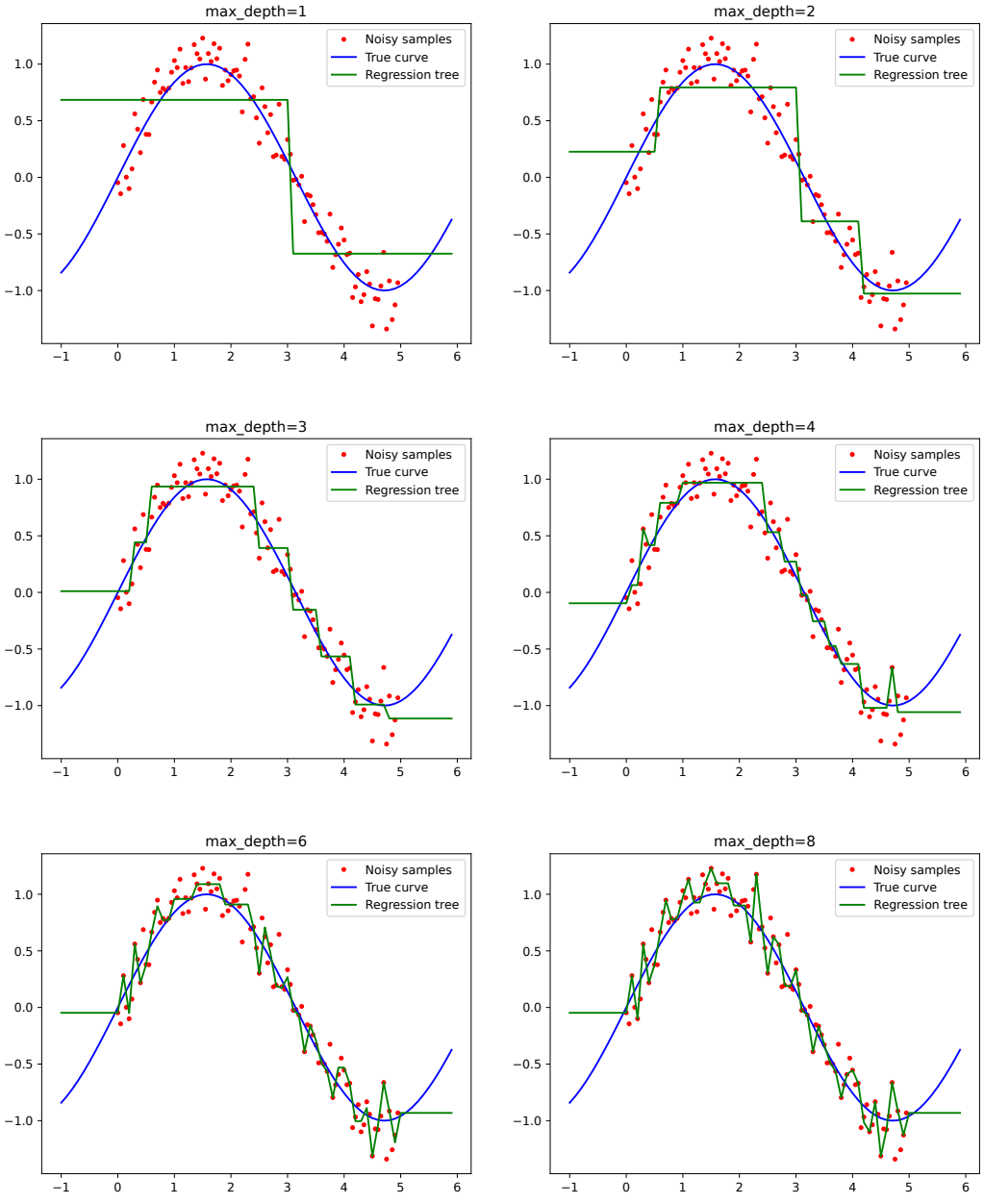


Figure 6.17: Underfitting: max_depth = 1

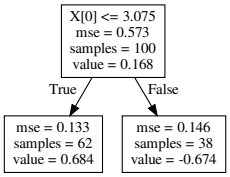


Figure 6.18: max_depth = 4

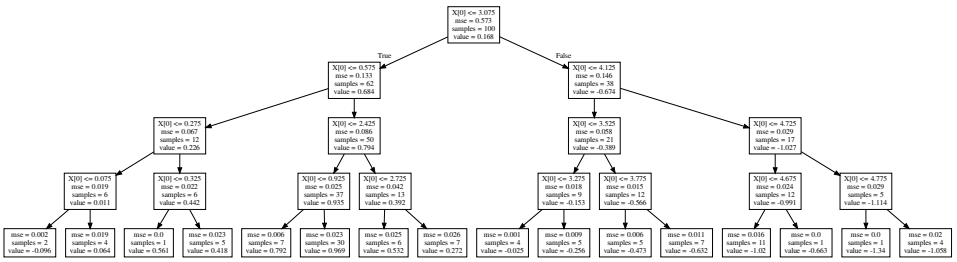
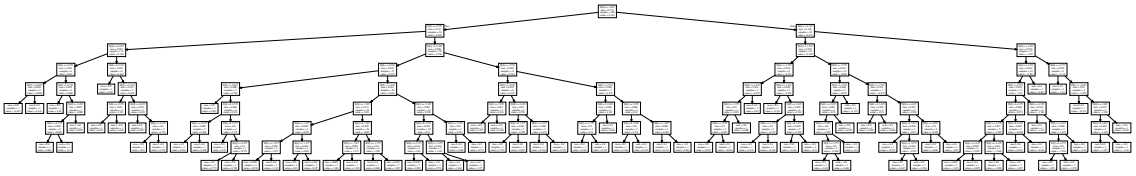


Figure 6.19: Overfitting: max_depth = 8



6.3. Random forest

The major problem with *decision/regression trees* is their variance. A decision tree can fit any decision boundary arbitrarily well but it is very sensitive to changes in the data. A small change in the data can lead to a completely different tree. By increasing the complexity of a tree (e.g. by increasing its maximum depth), we increase the variance too. However, we need complex (e.g. deep) trees in order to model complicated relationships. Therefore, we are looking for a way to use complex/deep trees while reducing the variance - this is what a random forest can do.

A *random forest* is an *ensemble* of decision trees - if have a regression problem, we simply replace the decision trees by regression trees.

Fitting a random forest is equivalent to fitting a sequence of decision trees. But instead of always using the same data set, we use for each tree a different bootstrapped data set (see section B.12). Furthermore, we only consider a random subset of features when computing the next split in a tree. By bootstrapping the data and considering a random subset of the features at each split only, we reduce the variance of the final model.
The idea of fitting a sequence of models on bootstrapped data sets is also called *bootstrap aggregation* (short *bagging*). The idea of selecting a random subset of features is also called *feature bagging* or *random subspace method*.

The prediction of the random forest model is obtained by combining the output of the trees in the forest.
If we use a random forest for classification, one way to combine the predictions of the decision trees is to do a majority voting - select the class which have been predicted the most.

$$f(x) = \arg\max_{c \in \mathcal{Y}} \sum_j \mathbb{1}(f_j(x) = c) \tag{6.4}$$

where f_j denotes the j-th decision tree.
An alternative to majority voting is to output a probability for each class, where we could compute the probabilities based on the probabilities of the decision trees or as the ratio of number of trees that predicted a specific class to total number of trees in the forest.

$$p(Y = c \mid x) = \frac{\sum_j \mathbb{1}(f_j(x) = c)}{b} \tag{6.5}$$

where b denotes the number of trees in the forest.
If we use a random forest for regression, we could simply average the predic-

tion of the regression trees.

$$f(x) = \frac{1}{b} \sum_{j} f_j(x) \tag{6.6}$$

where f_j denotes the j-th regression tree.

A comparison of a random forest classifier and a single decision tree on a toy data set is shown in Fig. 6.21. The single decision tree and the random forest have the same maximum depth of 8. However, we can observe that the decision boundary of the random forest (consists of 200 decision trees) appears to be much more robust and plausible than the decision boundary of the single decision tree.
Likewise, a comparison of a random forest regression and a regression tree on a toy data set is shown in Fig. 6.23. The single regression tree and the random forest have the same maximum depth of 6. However, similar to the random forest classifier, we can observe that the regression curve of the random forest (consists of 100 regression trees) appears to be much more robust and plausible than the regression curve of the single regression tree.

6.3.1. Feature relevance

Random forest can be used for determining the relevance of features. In the next two subsections we will take a look at two different methods for determining relevant features. However, note that different methods define the term "relevant features" differently. Therefore, it can happen that different methods declare different features to be relevant.

6.3.1.1 Permutation importance

Permutation importance - also called *Mean Decrease in Accuracy (MDA)* - measure the importance of a particular feature by computing the difference in prediction between an original out-of-bag data set and the same out-of-bag data set where this feature is permuted among all other data points in this set.

In order to determine the importance of the j-th feature, we determine the importance of that feature in each tree. To do so, we collect all *out-of-bag samples* for a particular tree[1] and use this tree to compute a prediction and the corresponding error (e.g. accuracy) of these out-of-bag samples. Then,

[1] the set of samples from the original training set that are not contained in the bootstrapped data set for this particular tree

we randomly permute the j-th feature among the out-of-bag samples and compute the predictions and error again. By permuting the feature values, we destroy the correlation of the feature to the output and all other features - you can think of it like putting "nonesense" into this feature. Finally, we compute the relevance of the j-th feature as the difference of the original error and the error on the set where the feature is permuted. We average this difference over alle trees and normalize it by dividing with the standard deviation.

The pseudocode of this procedure is described in algorithm 7.

Algorithm 7 Permutation importance/Mean decrease in accuracy (MDA)

for all features j **do** ▷ Compute relevance of each features
  for all trees T_i **do** ▷ Average over all trees in the forest
    $\mathcal{D}_{\text{oob}} = \text{oob}(\mathcal{D}, T_i)$ ▷ Out-of-bag data set for the current tree
    $\text{score}_i = \text{error}(\mathcal{D}_{\text{oob}}, T_i)$ ▷ Compute out-of-bag error
    $\mathcal{D}^{\pi}_{\text{oob}} = \text{permuteFeature}(\mathcal{D}_{\text{oob}}, j)$ ▷ Permute feature j
    $\text{score}^{\pi}_i = \text{error}(\mathcal{D}^{\pi}_{\text{oob}}, T_i)$ ▷ Compute out-of-bag error with permuted feature
    $\text{score}^i_j = |\text{score}_i - \text{score}^{\pi}_i|$ ▷ Compute difference in errors
  end for
  $\text{relevance}_j = \frac{\frac{1}{b}\sum_i \text{score}^i_j}{\sqrt{\sum_i (\text{score}^i_j - \overline{\text{score}_j})^2}}$ ▷ Compute the relevance of the j-th feature
end for

6.3.1.2 Mean decrease in impurity

The *mean decrease in impurity (MDI)* method[2] measure the relevance/importance of the j-th feature by summing the decrease in impurity of each node that splits on the j-th feature weighted by the proportion of samples that are split[3]. This value is averaged over all trees in the random forest.

[2] as implemented in scikit-learn [11]

[3] measured on the training data set - approximates the probability of reaching this particular node/split

Figure 6.20: Random forest classification vs. Decision tree

Figure 6.21: Decision tree

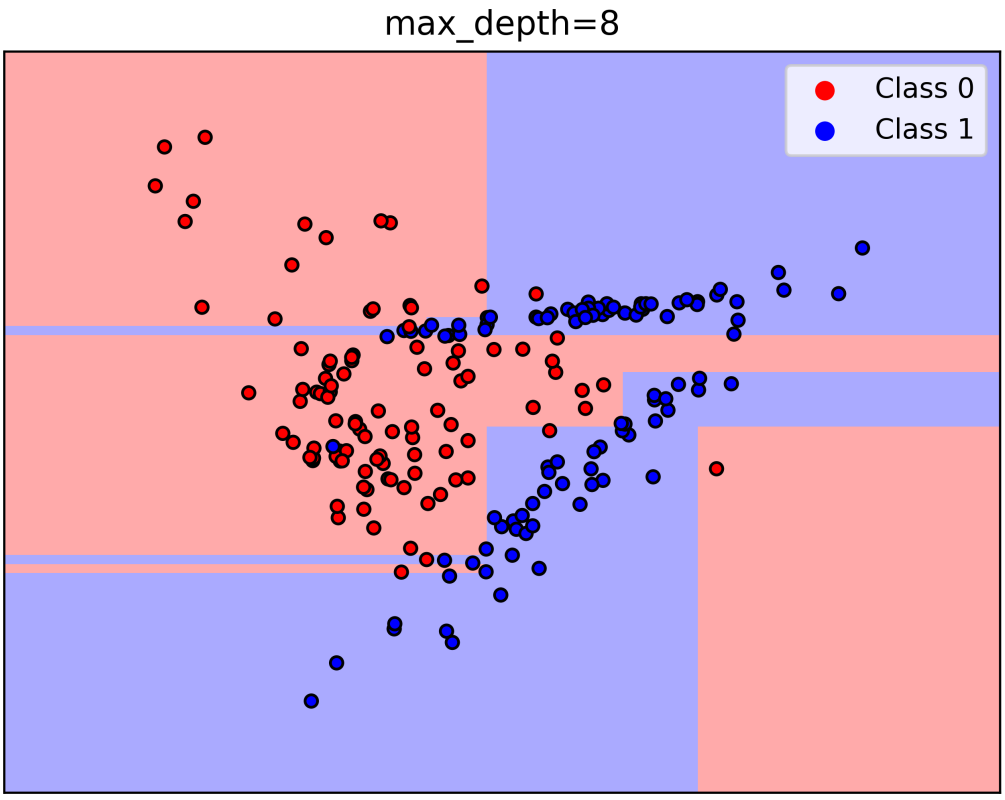


Figure 6.22: Random forest classification

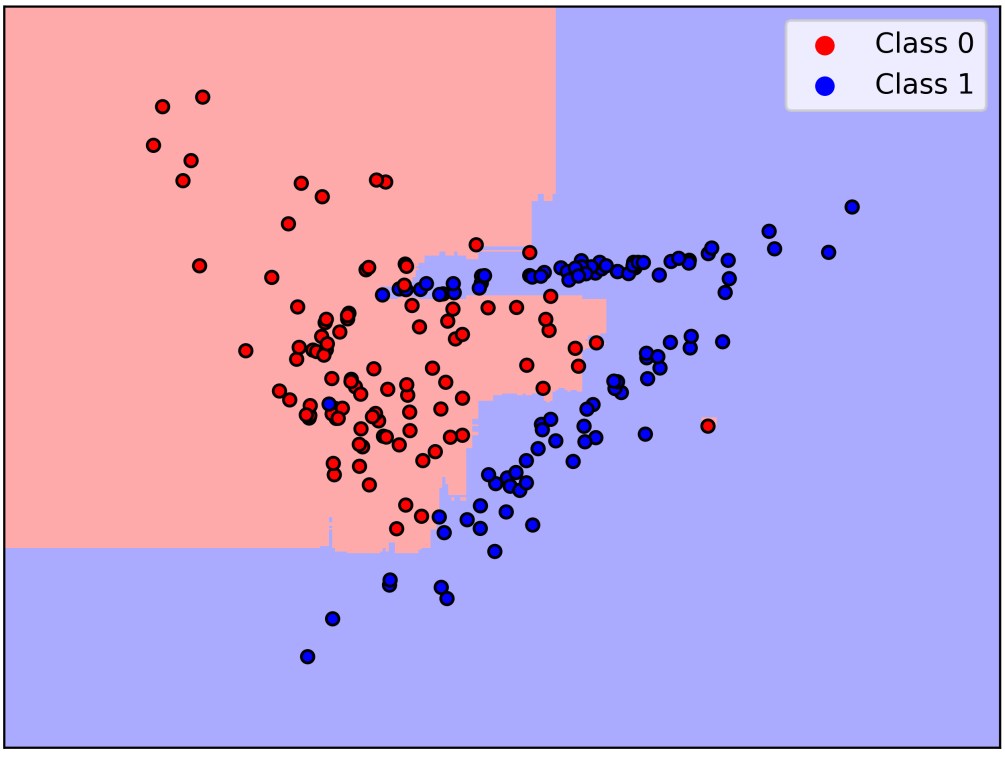


Figure 6.23: Random forest regression vs. Regression tree

Figure 6.24: Regression tree

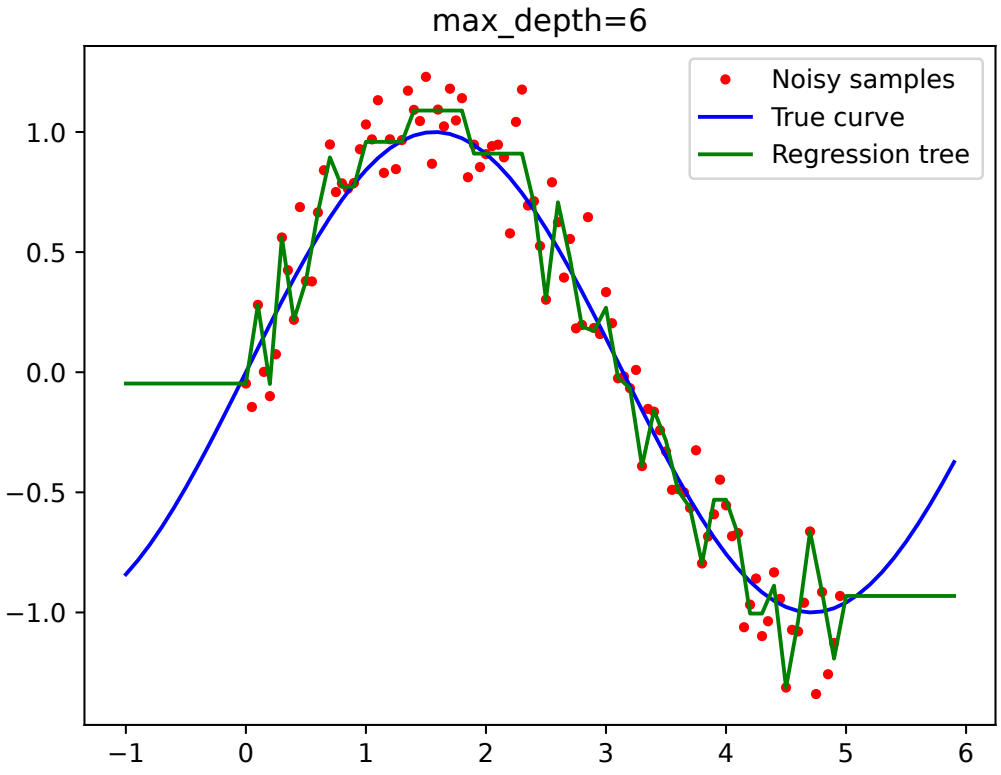
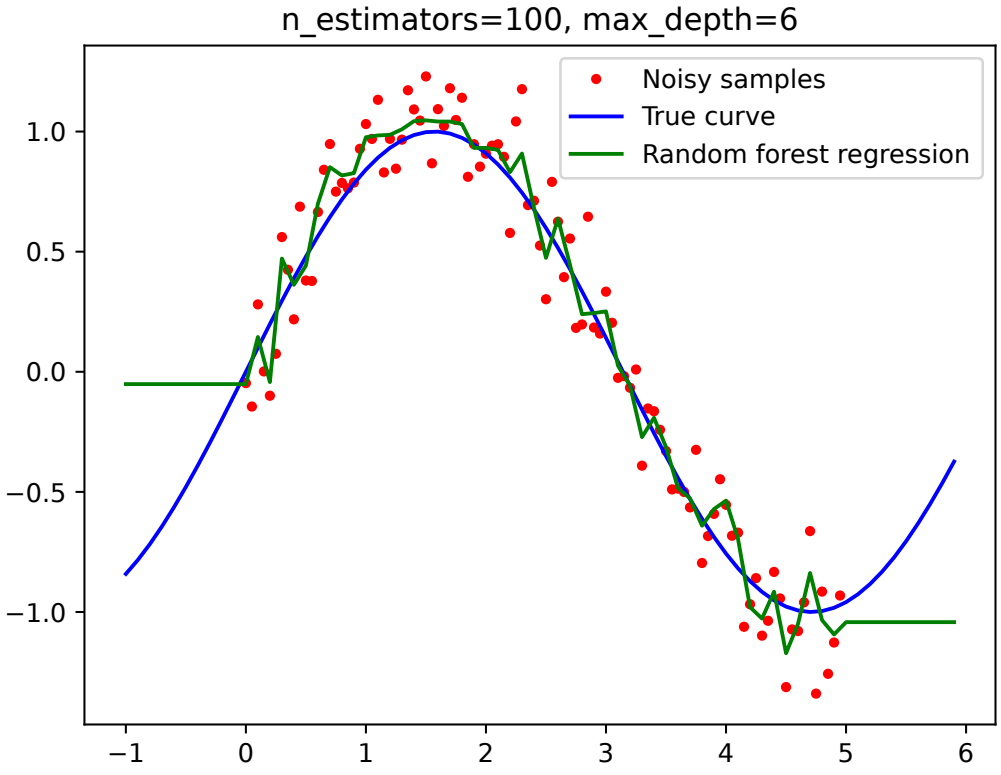


Figure 6.25: Random forest regression



6.4. Outlook

In this chapter we learned about tree based models. In particular, we learned about decision/regression trees and random forest as an esemble of decision/regression trees.
A decision/regression tree partitions the space into non-overlapping rectangles and assigns a prediction to each rectangle. We noticed that fitting/-constructing a decision/regression tree is hard and that is why some kind of heuristic is used - usually, we greedily grow a tree by recursively splitting on features and after that is done, applying some kind of pruning to reduce the complexity of the tree.
Because a single decision/regression tree is very sensitive to the training data - which can cause overfitting, we introduced an ensemble of trees (called random forest) that combines many trees to get a more robust model that is less sensitive to noisy training samples.

There are many more tree models like GBRT (Gradient Boosting Regression Trees) and AdaBoost. If you want to know more, a good starting point might be [16].

6.5. Exercises

Chapter 7

Evaluation

7.1. Metrics

When building and fitting models, one major task is to compute the quality of a model (e.g. determine how good a given model is). Usually we assess the quality of a model by computing smth. called a *metric*. An evaluation metric[1] computes (often) a score which gives us a hint on how "good" a model is. "Good" is smth. we need and actually will define. However, there is no universal best measurement/metric because each metric focus on slightly different aspects. Depending on the particular application we are working on, we have to select a suitable metric.
In this chapter we look at a couple of metrics. Both for regression and classification.

7.1.1. Regression

So far, we always used the sum of squared errors or the the sum of absolute deviations for measuring and optimizing our regression models.
However, there exist many different "measurement scores" for judging the quality of a given regression model for a given data set. In the following we give a short but not complete overview of the "most important" ones.

[1] not to be confused with a distance metric

7.1.1.1 SSE/RSS/SSR

The *Sum of Squared Errors* (SSE), *Residual Sum of Squares* (RSS) and *Sum of Squared Residuals* (SSR) are all the same and defined as

$$\sum_i (y_i - f(\vec{x}_i))^2 \tag{7.1}$$

where the expression $y_i - f(\vec{x}_i)$ is called *residual*.

7.1.1.2 MSE

The *Mean Squared Error* (MSE) is just the SSE with an additional regularization $\frac{1}{n}$

$$\text{MSE} = \frac{1}{n} \sum_i (y_i - f(\vec{x}_i))^2 \tag{7.2}$$

7.1.1.3 RMSE

The *Root Mean Squared Error* (RMSE) is just the square root of the MSE

$$\text{RMSE} = \sqrt{\text{MSE}} \tag{7.3}$$

7.1.1.4 RMSLE

The *Root Mean Squared Logarithmic Error* (RMSLE) is a metric which allows us to *penalize over or under estimates*.

$$\text{RMSLE} = \sqrt{\sum_i (\log(y_i) - \log(f(\vec{x}_i)))^2} \tag{7.4}$$

In 7.4 we penalize under estimates because $\log(y_i) - \log(f(\vec{x}_i)) = \log(\frac{y_i}{f(\vec{x}_i)})$ becomes larger if $f(\vec{x}_i)$ is less than y_i.
We can penalize over estimates by simply exchanging y_i and $f(\vec{x}_i)$ in 7.4.

Attention: This formula only makes sense if $f(\vec{x}_i)$ and y_i are *always positive*[2].

[2]The logarithm of a negative number is a complex number which makes things difficult...

7.1.1.5 R^2 score

The R^2 score tells us how much better than a simple *mean predictor* (thus $f(x) = \overline{y}$) a given regression model is.
In plain English: The R^2 score tells us how much (in percentage) the error reduces if we use the given regression model instead of a mean predictor.

The general formula for the R^2 score (sometimes also called *coefficient of determination*) is given by

$$R^2 = 1 - \frac{\sum_i (f(\vec{x}_i) - y_i)^2}{\sum_i (y_i - \overline{y})^2} \tag{7.5}$$

where $\overline{y}$ is the mean of the given targets (i.e. $\overline{y} = \frac{1}{n} \sum_i y_i$).

If the regression function is calculated by minimizing the SSE, we can use an equivalent formula (equivalent to 7.5) for the R^2 score (in this special case also called *explained variance*) which is given by

$$R^2 = \frac{\sum_i (f(\vec{x}_i) - \overline{y})^2}{\sum_i (y_i - \overline{y})^2} \tag{7.6}$$

Note: The value of 7.6 is always in [0, 1] and can not be negative! But in the general case the R^2 can be negative which means that the regression function performs worse than the mean predictor.

In Fig. 7.1 we can see an example of a positive and negative R^2 score. As we can see the mean predictor $f(\vec{x}) = \overline{y}$ would be a much better choice than the blue regression curve which has a negative R^2 score.

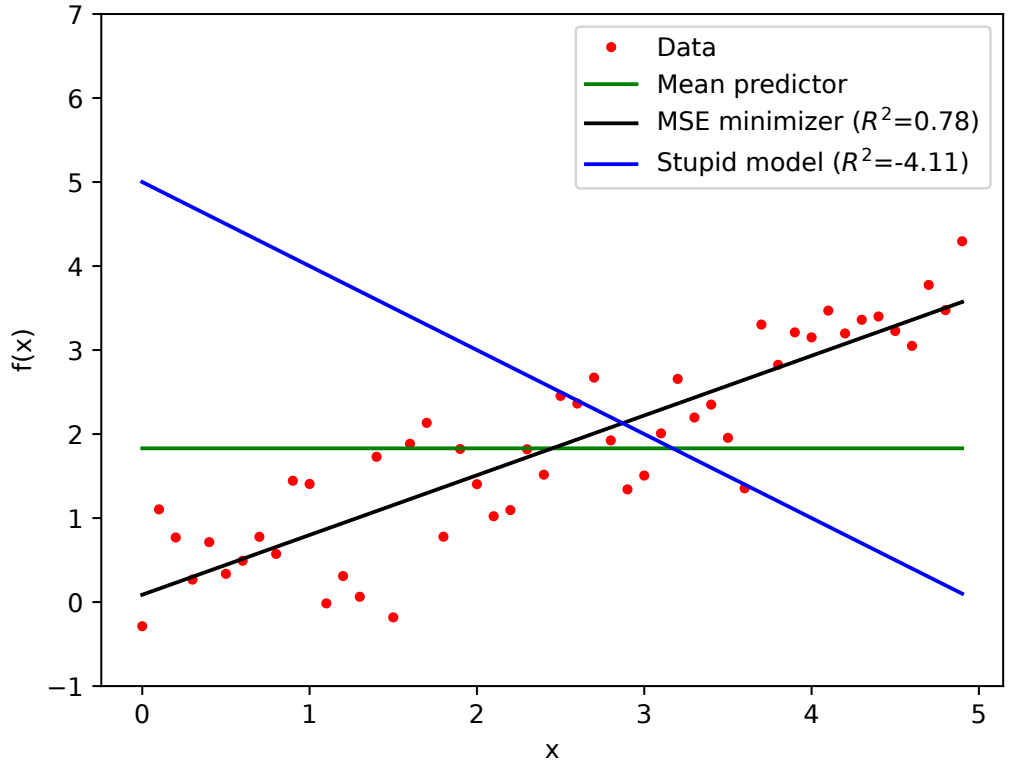


Figure 7.1: Illustration of different R^2 scores

7.1.2. Classification

Next, we look at some metrics - ignoring the likelihood and risk which we already discussed in previous sections - for assessing the quality of a classifier.

Before we can discuss a couple of classification metrics, we first have to define some terms.

1. P: Number of "positive" labels
   Formally: $\mathrm{P} = \left|\{i \mid (x_i, y_i) \in \mathcal{D}, y_i = 1\}\right|$
2. N: Number of "negative" labels
   Formally: $\mathrm{N} = \left|\{i \mid (x_i, y_i) \in \mathcal{D}, y_i = 0\}\right|$
3. TP: Number of *true positives*
   Formally: $\mathrm{TP} = \left|\{i \mid (x_i, y_i) \in \mathcal{D}, h(x_i) = y_i = 1\}\right|$

4. FP: Number of *false positives*
   Formally: $\mathrm{FP} = \left|\{i \mid (x_i, y_i) \in \mathcal{D}, h(x_i) = 1 \neq y_i\}\right|$
5. TN: Number of *true negatives*
   Formally: $\mathrm{TN} = \left|\{i \mid (x_i, y_i) \in \mathcal{D}, h(x_i) = y_i = 0\}\right|$
6. FN: Number of *false negatives*
   Formally: $\mathrm{FN} = \left|\{i \mid (x_i, y_i) \in \mathcal{D}, h(x_i) = 0 \neq y_i\}\right|$

Note: In statistics FP and FN are also called *type 1 error* and *type 2 error*.

7.1.2.1 Confusion matrix

The *confusion matrix* (also called *contingency table*) is defined as

Table 7.1: Confusion matrix

		Truth	
		1	0
Pred	1	TP	FP
	0	FN	TN

7.1.2.2 Accuracy

A natural measurement for assessing the quality of a classifier is the *accuracy* which is defined as

$$\frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}} \tag{7.7}$$

The accuracy computes the ratio of correct predictions to the total number of predictions. At a first glance this might seem to be a good choice, however, there is one major problem with this metric - *imbalanced data*. The term *imbalanced* data usually referes to the setting (data set) where one label occurs much more often than the other one. For instance consider a data set where 90 data points are labeled as 1 and 10 data points are labeled as 0. Furthermore, assume that we use a very stupid classifier which always predict 1 (thus $h(x) = 1$). When computing the accuracy of this classifier, by using 7.7, we get a score of 0.9 which does not look that bad. Hence, a high accuracy for imbalanced data sets does not necessarily indicate a model

that captures much of the inherent structure of the data.
Therefore, the accuracy is not always a reliable measurement of the performance of a classifier.

7.1.2.3 Precision & Recall

The *precision* is defined as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7.8}$$

and measures how "precise" our classifier is (which explains its name). The precision tells us how many of the items our classifier labeled with a 1 have indeed the label 1.
Hint: The denominator is the sum of the first row in the confusion matrix.

The *recall* (also called *true positive rate*, *hitrate* or *sensitivity*) is defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{7.9}$$

and measures how many of the "relevant" items we got. The term *relevant* refers to data points having the label 1.
Hint: The denominator is the sum of the first column in the confusion matrix.

We can imagine that it is sometimes better to have a high precision (e.g. spam classification) and sometimes a having a high recall is important (e.g. testing for diseases). Of course the best is to have a high precision and recall, which leads us the the next metric.

7.1.2.4 F1 score

The *F1 score* is defined as

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} = 2 \cdot \frac{\frac{\text{TP}}{\text{TP}+\text{FP}} \cdot \frac{\text{TP}}{\text{TP}+\text{FN}}}{\frac{\text{TP}}{\text{TP}+\text{FP}} + \frac{\text{TP}}{\text{TP}+\text{FN}}} \tag{7.10}$$

and combines *precision* and *recall* by computing the *harmonic mean* of them. This metrics might be a good choice if you want to optimize both, precision and recall.

7.1.2.5 False positive rate

The *false positive rate* is defined as

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{N}} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{7.11}$$

and measures *how many of the negative points where incorrectly labeled as positives*. The best value is 0 (no negative point have been classified as positive one) and the worst value is 1 (all negative points have been classified as positives).
Hint: The denominator is the sum of the second column in the confusion matrix.

7.1.2.6 True negative rate

The *true negative rate* (also called *specificity*) is defined as

$$\mathrm{TNR} = \frac{\mathrm{TN}}{\mathrm{N}} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} \tag{7.12}$$

and measures *how many of the negative points where correctly labeled as negative*. The best value is 1 (all negative points where recognized as negative points) and the worst value is 0 (all negative points where mislabeled as positives).
Hint: The denominator is the sum of the second column in the confusion matrix.

7.1.2.7 Receiver operating characteristic

From the previous sections we know the *false positive rate* and the *true positive rate* (also called recall). It might happen, that while optimizing one of them the other becomes worse. Hence, it would be nice if we had a way of combing them into a single measurement (like we did in the F1 score).
The *receiver operating characteristic* (short *roc*) measure how well a classifier is maximizing the true positive rate and minimizing the false positive rate. It does so by varying the discrimination threshold and computing for each threshold the true positive rate and false positive rate.
We start with the two most extreme cases:
First, consider $t = 0$ which means we always predict class 1, if the probability for class 1 is greater than 0. As a consequence we find that TPR = 1 and FPR = 1. This point (pair of TPR and FPR) would occur in the upper right corner of the roc-curve-plot.

The second case is $t = 1$ which implies that we always predict class 0 (again, no matter what the input is, because no probability can be greater than 1). It follows that TPR = 0 and FPR = 0. This point would be located in the lower left corner of the roc-curve-plot.
By computing and plotting the points for the remaining thresholds we obtain the roc-curve. The roc-curve is a monotonically increasing function which starts in 0 (first extreme case) and goes to 1 (second extreme case).
See Fig.7.2 for an illustration of two roc curves (each curve belongs to a different classifier). The higher/quicker the "slope" of the curve is the better the

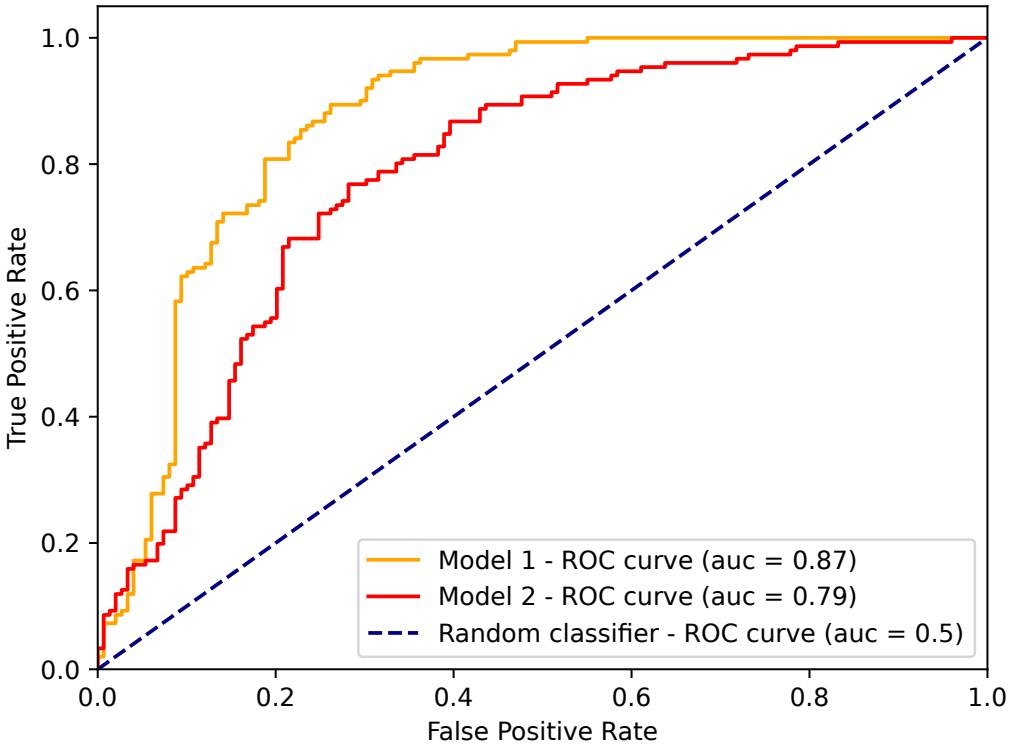


Figure 7.2: Illustration of different roc curves

model performs. The perfect classifier "jumps" to 1 immediately in the beginning, since it always predicts the right class until the threshold becomes 1.

By looking at roc-curvs of multiple classifiers we, as a human being, can easily tell which classifier "is better". Recall that the perfect classifier jumps to TPR = 1 right in the beginning and stays there, no matter what threshold we choose. However, looking at a plot can be problematic if we want

to do some automatic evaluation. An alternative to "looking at plots" is to compute the *area under the curve (AUC)* of the roc-curve. As the name suggests, the area under the curve is simply the area under the roc curve. The perfect classifier has an AUC of 1, the random classifier an AUC of 0.5. Thus, a higher AUC is better - 0.5 is the "random" classifier which is often considered as stupid baseline which we have to outperform.

7.2. How to estimate scores

Now that we know a lot of different metrics for evaluating a given model, the remaining question is on what data should we compute it. It is obvious that using the same data set, which we already used for training the model, might not be the best idea[3]. Indeed, we will not get an unbiased estimate of the score by using the same data set. Instead, we should use a new/different data set called *test set*.

7.2.1. Train - Test split

Before we start training/fitting a model we randomly split the given data set into two parts. A training set and a test set. See Fig. 7.3 for an illustraition. As the name suggests, we use the training set to fit our model (i.e. learning/-computing/estimating all parameters) and the test set is used for computing the evaluation metric afterwards.

Note: Sometimes the data set is split into three parts: training, test and validation set, where the validation set might be used for parameter tuning.

[3]because of overfitting 7.2.3

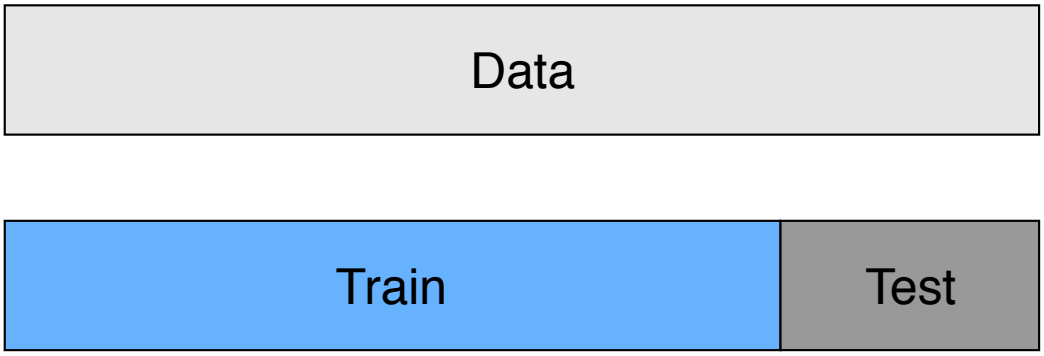


Figure 7.3: Illustration of a train-test split

This procedure is perfectly valid but when we only have a "small" data set available we run into problems. If our data set is small, our test set will be even smaller. A small test set is problematic because it might no longer be "representative", thus our computed scores are still unbiased but more or less useless because the test set is to small - we either might be lucky and having all "easy" points in our test set or the other way around or a mixture of both. We can not increase the size of the test set because this would decrease the size of the training set - training on a smaller training set is more difficult and might even be impossible if the training set is no longer "representative". If gathering more data is not possible we could/should switch to smth. called *cross validation*.

7.2.2. Cross validation

Cross validation is a procedure/strategy for splitting the data into train and test sets. In contrast to the train-test split strategy from sec. 7.2.1 it splits the data into *multiple* train-test splits.
The procedure of cross validation is illustrated in Fig. 7.4.

7.2.3. Overfitting & Underfitting

The scenario, where our model has a very good score (e.g. low error) on training set, but a very bad score (e.g. high error) on the test set, is called *overfitting*. Usually overfitting indicates that our model is too complex. Viz., it has too many degrees of freedom and is able to capture the noise in the training set. To avoid overfitting, we should switch to a less complex model and/or add more training data (if possible).

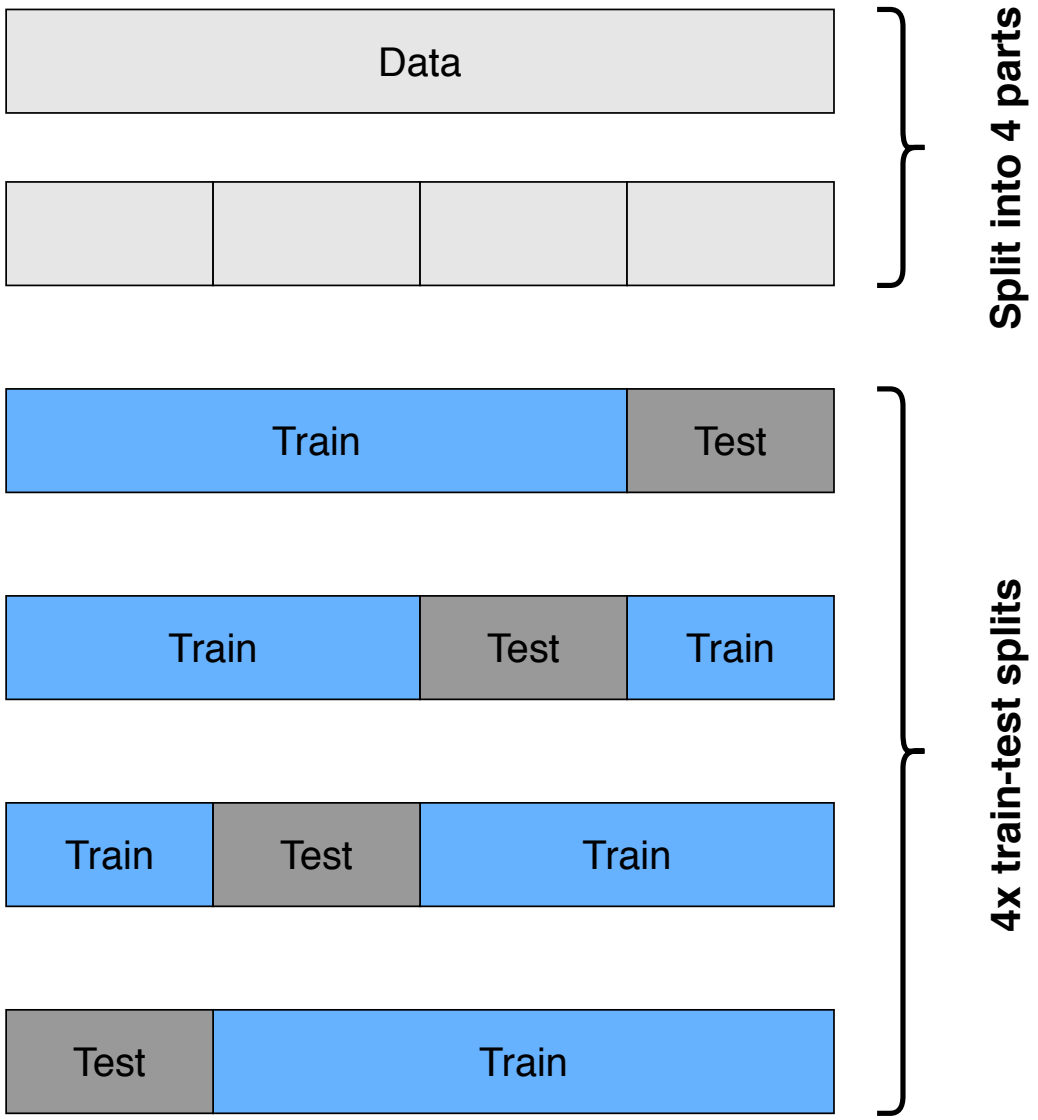


Figure 7.4: Illustration of 4-fold cross validation

The other scenario, where our models has a very bad score on the training and test set, is called *underfitting*. In this case our model is either not complex enough (e.g. too few degrees of freedom) to capture the patterns in the data set or our training set is not large enough. Therefore we should use a model with more degrees of freedom.

7.3. Model selection

Occams razor[4] is an informal principle which states that we should prefer the simple theory/hypothesis over a complicated theory if they both explain something of our interest equally well.

In machine learning, the term "simple model" usually referes to models with low complexity - e.g. low number of parameters or degrees of freedom. That is, if we have two models with an equal test score, we should select the model with the lowest complexity.
For instance, consider that we have a polynomial regression model with degree 3 and another polynomial regression model with degree 42. If both models have roughly the same test score, we should select the polynomial regression model with degree 3 because it is simpler in the sense that it has less parameters.

7.4. Feature selection

The number of possible subsets of features grows exponential in the number of features - if we have d features, there are 2^d possible feature selections (there are exactly two possibilities per feature: either the feature is selected or not). Because testing all 2^d possibilities is infeasible, we use some kind of heuristic for finding a good subset of features. The problem of finding the best subset of features is also called *best feature subset selection problem*.
We distinguish between three different types of feature selection methods:

1. *Wrapper methods* select features based on a model. For instance, removing or adding a feature and checking whether the model performance increases.

2. *Filter methods* select features without using a model. That is, features are selected before putting them into any model.

3. *Embedded methods* are models with a build-in mechanism for feature selection (e.g. lasso).

7.4.1. Wrapper methods

We can use random forests for feature selection (see section 6.3.1). In particular permutation importance 6.3.1.1 and mean decrease in impurity 6.3.1.2.

[4]see `https://en.wikipedia.org/wiki/Occam's_razor`

7.4.2. Filter methods

The *f-score* measures how well two sets of real numbers are separated. In the context of classification, we can interpret this as the discriminative power of a particular feature. The f-score of the j-th feature is defined as

$$F_j = \frac{\left(\overline{(\vec{x})_j}^{+} - \overline{(\vec{x})_j}\right)^2 + \left(\overline{(\vec{x})_j}^{-} - \overline{(\vec{x})_j}\right)^2}{\frac{1}{n_+ - 1}\sum_i \left((\vec{x}_i)_j^{+} - \overline{(\vec{x})_j}^{+}\right)^2 + \frac{1}{n_- - 1}\sum_i \left((\vec{x}_i)_j^{-} - \overline{(\vec{x})_j}^{-}\right)^2} \tag{7.13}$$

where $\overline{(\vec{x})_j}^{+}$ denotes the mean of the j-th feature of all samples from the first class and $\overline{(\vec{x})_j}^{-}$ denotes the mean of the j-th feature of all samples from the second class, n_+ and n_- denote the number of samples in the first and second class and $\overline{(\vec{x})_j}$ denotes the mean of the j-th feature over all samples (from both classes).
A large f-score means that the two sets are more apart from each other and thus can be be better separated.
In case of regression, we can use *Pearson's correlation coefficient*[5] (sample estimate of the correlation B.53) to compute the linear correlation of a particular feature and the output.

Another way for testing the correlation or dependency of a feature with the output is to use mutal information B.10.3. In contrast to f-score or Pearson's correlation coefficient , mutal information is not limited to linear correlations. However, mutal information can be difficult to estimate if we have have a small data set of high dimensional data points.

7.4.3. Embedded methods

Lasso 4.8.1 is a regularization technique that leads to sparse weight vectors. The term sparse means that "many" entries in the weight vector are equal to zero. All features having a zero entry in the weight vector a not used for computing a prediction. Thus, lasso selects a subset of features - lasso can be viewed as a convex relaxation of the best feature subset selection problem. We can control the number of selcted features by varying the regularization strength. If we want to include the grouping effect, we should combine lasso with L2-regularization which yields the elasticnet-penalty 4.9.1.

[5] see `https://en.wikipedia.org/wiki/Pearson_correlation_coefficient`

7.5. Exercises

Chapter 8

Dimensionality reduction

Dimensionality reduction is all about computing a low dimensional representation of a data set while maintaining as much "information/structure" as possible.
Dimensionality reduction might be applied to reduce the dimension of a data set to make computation more efficient, stable or feasible - e.g. we usually can not plot data with more than 3 dimensions.

8.1. PCA

Principle component analysis (*PCA*) computes a p dimensional representation of a given data set such that the transformed features are linearly uncorrelated and the squared reconstruction error is minimized.

We assume that the *centered data*[1] are ordered row wise in a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, we then define the PCA problem for computing a p dimensional approximation of the data $\mathbf{X}$ as

$$\min_{\substack{\text{rank}((\mathbf{XP})\mathbf{P}^\top)=p, \\ \mathbf{P}^\top\mathbf{P}=\mathbb{I}}} \|\mathbf{X} - (\mathbf{XP})\mathbf{P}^\top\|_F^2 \tag{8.1}$$

where we need to find the unknown projection matrix $\mathbf{P}$.
Note: The PCA problem as stated in 8.1 is a *non-convex* optimization problem.

The PCA can be derived in two different ways, yielding the same result. In the next two sections, we take a closer look at each derivation.

[1] all columns have zero mean

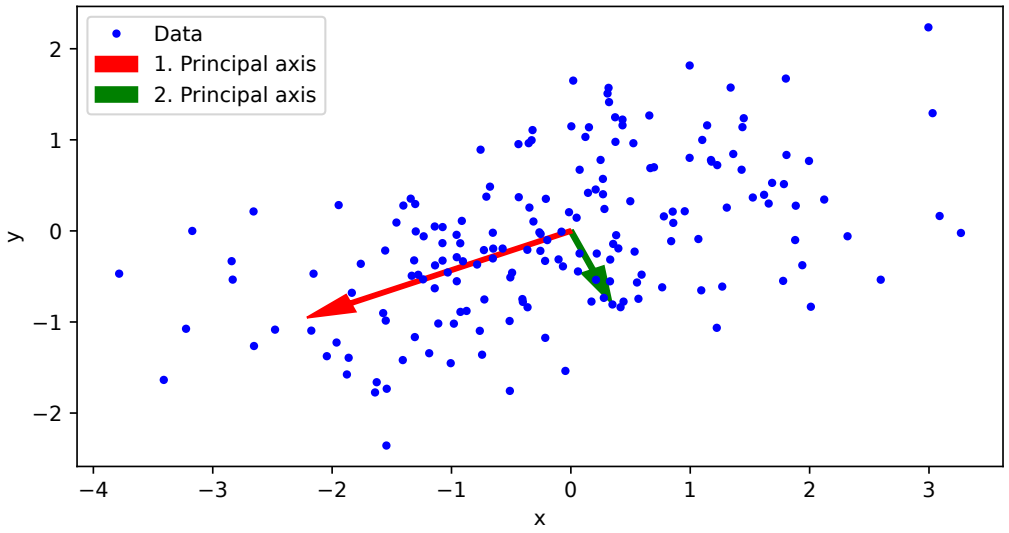


Figure 8.1: Principal components of a 2d data set

8.1.1. Derivation - Reconstruction error

We assume that the data is given as a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$. Furthermore we assume that the data has zero mean (viz. $\frac{1}{n}\sum_i \vec{x}_i = \vec{0}$).

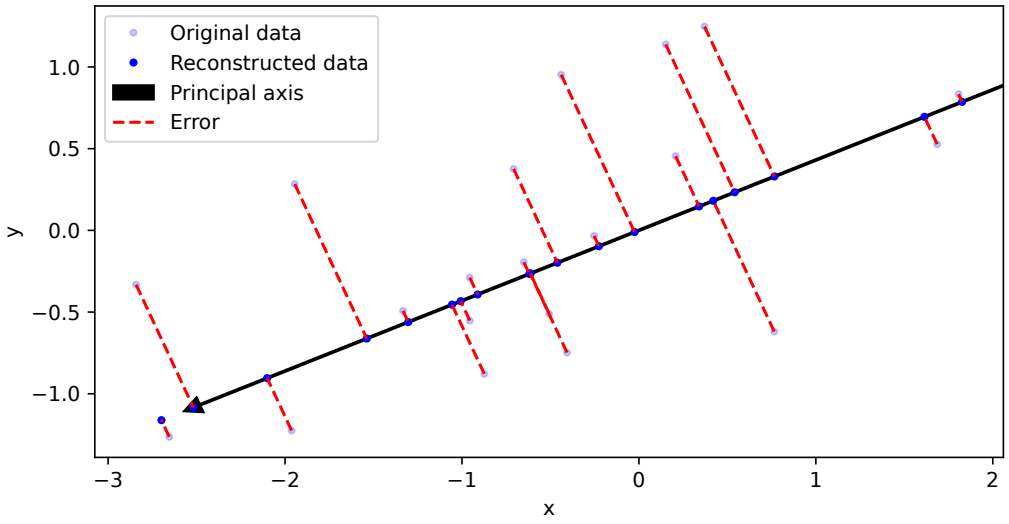


Figure 8.2: PCA - Reconstruction error

First, we consider the special case with one axis only - projecting the data onto a line. We want to select an axis (normalized weight vector) such that the reconstruction error is as small as possible (see Fig. 8.2 for an illustration). Hence, we are trying to solve the following optimization problem

$$\begin{aligned} \min_{\vec{w}_1 \in \mathbb{R}^d} \frac{1}{n} \sum_i \|\vec{x}_i - (\vec{w}_1^\top \vec{x}_i)\vec{w}_1\|_2^2 \\ \text{s.t. } \vec{w}_1^\top \vec{w}_1 = 1 \end{aligned} \tag{8.2}$$

In plain English: Find a line (weight vector $\vec{w}_1$), such that projecting a point $\vec{x}_i$ onto the line and back again ($(\vec{w}_1^\top \vec{x}_i)\vec{w}_1$) is as close as possible to the original point.

Next, we take a look at the reconstruction error $\|\vec{x}_i - (\vec{w}_1^\top \vec{x}_i)\vec{w}_1\|_2^2$ and simplify it

$$\begin{aligned} \|\vec{x}_i - (\vec{w}_1^\top \vec{x}_i)\vec{w}_1\|_2^2 &= (\vec{x}_i - (\vec{w}_1^\top \vec{x}_i)\vec{w}_1)^\top (\vec{x}_i - (\vec{w}_1^\top \vec{x}_i)\vec{w}_1) \\ &= \vec{x}_i^\top \vec{x}_i - \vec{x}_i^\top (\vec{w}_1^\top \vec{x}_i)\vec{w}_1 - \vec{w}_1^\top (\vec{x}_i^\top \vec{w}_1)\vec{x}_i + \vec{w}_1^\top (\vec{x}_i^\top \vec{w}_1)(\vec{w}_1^\top \vec{x}_i)\vec{w}_1 \\ &= \vec{x}_i^\top \vec{x}_i - 2\left(\vec{x}_i^\top (\vec{w}_1^\top \vec{x}_i)\vec{w}_1\right) + (\vec{x}_i^\top \vec{w}_1)(\vec{w}_1^\top \vec{x}_i) \\ &= \vec{x}_i^\top \vec{x}_i - 2(\vec{w}_1^\top \vec{x}_i)^2 + (\vec{w}_1^\top \vec{x}_i)^2 \\ &= \vec{x}_i^\top \vec{x}_i - (\vec{w}_1^\top \vec{x}_i)^2 \end{aligned} \tag{8.3}$$

By making use of 8.3 we can rewrite 8.2 as

$$\begin{aligned} &\min_{\vec{w}_1 \in \mathbb{R}^d} \frac{1}{n} \sum_i ||\vec{x}_i - (\vec{w}_1^\top \vec{x}_i)\vec{w}_1||_2^2 \quad \text{s.t. } \vec{w}_1^\top \vec{w}_1 = 1 \\ \Leftrightarrow &\min_{\vec{w}_1 \in \mathbb{R}^d} \frac{1}{n} \sum_i \vec{x}_i^\top \vec{x}_i - (\vec{w}_1^\top \vec{x}_i)^2 \quad \text{s.t. } \vec{w}_1^\top \vec{w}_1 = 1 \\ \Leftrightarrow &\max_{\vec{w}_1 \in \mathbb{R}^d} \frac{1}{n} \sum_i (\vec{w}_1^\top \vec{x}_i)^2 \quad \text{s.t. } \vec{w}_1^\top \vec{w}_1 = 1 \end{aligned} \tag{8.4}$$

From probability theory (see chapter B) we know that the following is true

$$\mathbb{E}[X^2] = \mathbb{E}[X]^2 + \text{Var}[X] \tag{8.5}$$

If X has zero mean, 8.5 can be rewritten as

$$\mathbb{E}[X^2] = \text{Var}[X] \tag{8.6}$$

The same (8.5, 8.6) is true for the empirical version [2] of 8.5 and 8.6.

Because we assumed that the data has zero mean, it holds that

$$\begin{aligned} \frac{1}{n}\sum_i \vec{w}_1^\top \vec{x}_i &= \vec{w}_1^\top \frac{1}{n}\sum_i \vec{x}_i \\ &= 0 \end{aligned} \tag{8.7}$$

Because of 8.7, we can make use of 8.6 and observe that 8.4 is equivalent to maximizing the variance of the projection. Thus, with a slight abuse of notation, we can rewrite 8.2 as

$$\max_{\vec{w}_1 \in \mathbb{R}^d} \text{Var}[\vec{w}_1^\top \vec{x}_i] \quad \text{s.t. } \vec{w}_1^\top \vec{w}_1 = 1 \tag{8.8}$$

Next, we rewrite the function from 8.4 in matrix vector notation

$$\begin{aligned} \frac{1}{n}\sum_i (\vec{w}_1^\top \vec{x}_i)^2 &= \frac{1}{n}(\mathbf{X}\vec{w}_1)^\top(\mathbf{X}\vec{w}_1) \\ &= \frac{1}{n}\vec{w}_1^\top \mathbf{X}^\top \mathbf{X}\vec{w}_1 \\ &= \vec{w}_1^\top (\frac{1}{n}\mathbf{X}^\top \mathbf{X})\vec{w}_1 \\ &= \vec{w}_1^\top \mathbf{S}_\mathbf{X}\vec{w}_1 \end{aligned} \tag{8.9}$$

where $\mathbf{S}_\mathbf{X} = \frac{1}{n}\mathbf{X}^\top\mathbf{X}$ is the empirical covariance matrix of $\mathbf{X}$. Note that the matrix $\mathbf{S}_\mathbf{X} \in \mathbb{R}^{d\times d}$ is symmetric.
Finally, we arrive at the final optimization problem

$$\max_{\vec{w}_1 \in \mathbb{R}^d} \vec{w}_1^\top \mathbf{S}_\mathbf{X}\vec{w}_1 \quad \text{s.t. } \vec{w}_1^\top \vec{w}_1 = 1 \tag{8.10}$$

We solve 8.10 by introducing a Lagrange multiplier λ for the constraint $\vec{w}_1^\top \vec{w}_1 = 1$.

$$\max_{\vec{w}_1 \in \mathbb{R}^d, \lambda \in \mathbb{R}} \mathcal{L}(\vec{w}_1, \lambda) = \vec{w}_1^\top \mathbf{S}_\mathbf{X}\vec{w}_1 - \lambda(\vec{w}_1^\top \vec{w}_1 - 1) \tag{8.11}$$

From the optimality conditions we find that

$$\begin{aligned} &\frac{\partial \mathcal{L}}{\partial \lambda} = 0 \\ &\Leftrightarrow -\vec{w}_1^\top \vec{w}_1 + 1 = 0 \\ &\Leftrightarrow \vec{w}_1^\top \vec{w}_1 = 1 \end{aligned} \tag{8.12}$$

[2] the expectation is replaced by an average - see law of large numbers B.8

and

$$\begin{aligned} &\nabla_{\vec{w}_1}\mathcal{L} = \vec{0} \\ &\Leftrightarrow 2\mathbf{S_X}\vec{w}_1 - 2\lambda\vec{w}_1 = \vec{0} \\ &\Leftrightarrow \mathbf{S_X}\vec{w}_1 = \lambda\vec{w}_1 \end{aligned} \tag{8.13}$$

We observe that 8.13 is the eigen-vector/value equation for the matrix $\mathbf{S_X}$. Therefore we have to choose an eigenvector of $\mathbf{S_X}$ with unit length (because of 8.12). Luckily, linear algebra tells us that all eigenvectors of a real-symmetric matrix (like $\mathbf{S_X}$) are orthonormal. But which one should we pick?
We want to pick the eigenvector $\vec{w}_1$ which maximizes $\vec{w}_1^\top \mathbf{S_X}\vec{w}_1$. By making use of 8.13 we find that

$$\begin{aligned} \vec{w}_1^\top \mathbf{S_X}\vec{w}_1 &= \vec{w}_1^\top(\lambda\vec{w}_1) \\ &= \lambda\vec{w}_1^\top\vec{w}_1 \\ &= \lambda \end{aligned} \tag{8.14}$$

Therefore we should select the eigenvector $\vec{w}_1$ with the largest eigenvalue λ. Furthermore we notice that the variance, remember that our goal was to maximize the variance - keep as much "information" as possible, of the projection is equal to the eigenvalue of the used eigenvector. Thus

$$\mathrm{Var}[\vec{w}_1^\top\vec{x_i}] = \lambda \tag{8.15}$$

Now that we know how to compute the first principal axis, we want to compute the second axis too.
As with the first axis, we want to find a projection, such that the reconstruction error is minimized. In addition, we also want this second axis to be orthogonal to the first axis. If we do the same derivation like we did in first case, we obtain the following optimization problem

$$\max_{\vec{w}_2 \in \mathbb{R}^d} \vec{w}_2^\top \mathbf{S_X}\vec{w}_2 \quad \text{s.t. } \vec{w}_2^\top\vec{w}_2 = 1 \ , \ \vec{w}_2^\top\vec{w}_1 = 0 \tag{8.16}$$

Note the similarity with 8.10. Next, we introduce Lagrange multipliers λ_1 and λ_2 for the constraints and obtain

$$\max_{\substack{\vec{w}_2 \in \mathbb{R}^d, \\ \lambda_1,\lambda_2 \in \mathbb{R}}} \mathcal{L}(\vec{w}_2, \lambda_1, \lambda_2) = \vec{w}_2^\top \mathbf{S_X}\vec{w}_2 - \lambda_1(\vec{w}_2^\top\vec{w}_2 - 1) - \lambda_2\vec{w}_2^\top\vec{w}_1 \tag{8.17}$$

The optimiality conditions yield

$$\begin{aligned} &\frac{\partial\mathcal{L}}{\partial\lambda_1} = 0 \\ &\Leftrightarrow \vec{w}_2^\top\vec{w}_2 = 1 \end{aligned} \tag{8.18}$$

$$\begin{aligned} &\frac{\partial \mathcal{L}}{\partial \lambda_2} = 0 \\ &\Leftrightarrow \vec{w}_2^\top \vec{w}_1 = 0 \end{aligned} \tag{8.19}$$

and

$$\begin{aligned} &\nabla_{\vec{w}_2} \mathcal{L} = \vec{0} \\ &\Leftrightarrow 2\mathbf{S_X}\vec{w}_2 - 2\lambda_1 \vec{w}_2 - \lambda_2 \vec{w}_1 = \vec{0} \\ &\Leftrightarrow \mathbf{S_X}\vec{w}_2 = \lambda_1 \vec{w}_2 + \frac{\lambda_2}{2}\vec{w}_1 \end{aligned} \tag{8.20}$$

Note that 8.20 looks like an eigenvector/value equation with the additional term $\frac{\lambda_2}{2}\vec{w}_1$. Next, we prove that it must be the case that $\lambda_2 = 0$.
Claim: The λ_2 in 8.20 must be equal to 0.
Proof: We multiply 8.20 by $\vec{w}_1^\top$ and simplify the resulting expression

$$\begin{aligned} \vec{w}_1^\top \mathbf{S_X}\vec{w}_2 - \lambda_1 \vec{w}_1^\top \vec{w}_2 - \frac{\lambda_2}{2}\vec{w}_1^\top \vec{w}_1 &= 0 \\ (\mathbf{S_X}\vec{w}_1)^\top \vec{w}_2 - \frac{\lambda_2}{2} &= 0 \\ \lambda_1 \vec{w}_1^\top \vec{w}_2 - \frac{\lambda_2}{2} &= 0 \\ -\frac{\lambda_2}{2} &= 0 \end{aligned} \tag{8.21}$$

The only value of λ_2, for which 8.21 is true, is $\lambda_2 = 0$.
Therefore 8.20 becomes

$$\mathbf{S_X}\vec{w}_2 = \lambda_1 \vec{w}_2 \tag{8.22}$$

Hence, we have to select the eigenvector $\vec{w}_2$ with the snd-largest eigenvalue - we can not take the one with the largest eigenvalue, because this is already $\vec{w}_1$ and clearly $\vec{w}_1$ is not orthogonal to itself.

Finally, we can generalize the previous cases to compute the k-th principal axis. The optimization problem for the k-th axis can be stated as follows

$$\max_{\substack{\vec{w}_k \in \mathbb{R}^d, \\ \lambda_1,\ldots,\lambda_k \in \mathbb{R}}} \mathcal{L}(\vec{w}_k, \lambda_1, ..., \lambda_k) = \vec{w}_k^\top \mathbf{S_X}\vec{w}_k - \lambda_1(\vec{w}_k^\top \vec{w}_k - 1) - \sum_{i=2}^{k-1} \lambda_i \vec{w}_k^\top \vec{w}_i \tag{8.23}$$

The optimality conditions yield

$$\begin{aligned} &\frac{\partial \mathcal{L}}{\partial \lambda_1} = 0 \\ &\Leftrightarrow \vec{w}_k^\top \vec{w}_k = 1 \end{aligned} \tag{8.24}$$

$$\begin{aligned} &\frac{\partial \mathcal{L}}{\partial \lambda_i} = 0 \\ &\Leftrightarrow \vec{w}_k^\top \vec{w}_i = 0 \end{aligned} \tag{8.25}$$

and

$$\begin{aligned} &\nabla_{\vec{w}_k} \mathcal{L} = \vec{0} \\ &\Leftrightarrow 2\mathbf{S_X}\vec{w}_k - 2\lambda_1 \vec{w}_k - \sum_{i=2}^{k-1} \lambda_i \vec{w}_i = \vec{0} \\ &\Leftrightarrow \mathbf{S_X}\vec{w}_k = \lambda_1 \vec{w}_k + \sum_{i=2}^{k-1} \frac{\lambda_i}{2} \vec{w}_i \end{aligned} \tag{8.26}$$

By using the same argument which we used for the snd-axis, we can show that $\lambda_2 = ... = \lambda_k = 0$ - we multiply 8.26 by $\vec{w}_1^\top$ and observe that $\lambda_2 = 0$, we multiply 8.26 by $\vec{w}_2$ and observe that $\lambda_3 = 0$, ...
We conclude that the k-th principal component is the eigenvector of $\mathbf{S_X}$ with the k-th largest eigenvalue.

When computing a dimensionality reduction, we do not use all d eigenvectors but only $p < d$ with the largest eigenvalues. Because we know that the variance is equal to the eigenvalue, we can compute the amount/fraction of captured variance as follows

$$\frac{\sum_i^p \lambda_i}{\sum_i^d \lambda_i} \tag{8.27}$$

where we made use of the linearity of the variance when the variables (axis) are uncorrelated, which is true in our case, because of the orthogonality constraints.

We can construct the projection matrix $\mathbf{P}$ in 8.1 by horizontally concatenating the principal axis $\vec{w}_j$. Thus

$$\mathbf{P} = (\vec{w}_1, \cdots, \vec{w}_p) \tag{8.28}$$

8.1.2. Derivation - Diagonal covariance matrix

Our goal is to find a projection matrix $\mathbf{P}$ such that the (empirical) covariance matrix of the transformed data is a diagonal matrix - we want the features of the transformed data to be uncorrelated. We require that all basis vectors

(columns of $\mathbf{P}$) are orthonormal.
The transformed data $\mathbf{X}'$ is denoted by

$$\mathbf{X}' = \mathbf{X}\mathbf{P} \tag{8.29}$$

The (empirical) covariance matrix of the transformed data $\mathbf{S}_{\mathbf{X}'}$ is denoted by

$$\mathbf{S}_{\mathbf{X}'} = \frac{1}{n}\mathbf{X}'^{\top}\mathbf{X}' \tag{8.30}$$

where sometimes $\frac{1}{n}$ is replaced by $\frac{1}{n-1}$, for the purpose of getting an unbiased estimator of the covariance.
Recall that our goal is to have a diagonal covariance matrix for the transformed data

$$\mathbf{S}_{\mathbf{X}'} = \mathrm{diag}(\alpha_i) \tag{8.31}$$

Because the projection matrix $\mathbf{P}$ is required to be orthonormal, we know that

$$\mathbf{P}^{\top}\mathbf{P} = \mathbf{P}\mathbf{P}^{\top} = \mathbb{I} \text{ and } \|\vec{p}_i\|_2 = 1 \; \forall \, i \tag{8.32}$$

Working on 8.30 yields

$$\begin{aligned} \mathbf{S}_{\mathbf{X}'} &= \frac{1}{n}\mathbf{X}'^{\top}\mathbf{X}' \\ &= \frac{1}{n}(\mathbf{X}\mathbf{P})^{\top}\mathbf{X}\mathbf{P} \\ &= \frac{1}{n}\mathbf{P}^{\top}\mathbf{X}^{\top}\mathbf{X}\mathbf{P} \\ &= \frac{1}{n}\mathbf{P}^{\top}\tilde{\mathbf{X}}\mathbf{P} \end{aligned} \tag{8.33}$$

where $\tilde{\mathbf{X}} = \mathbf{X}^{\top}\mathbf{X}$. Note that $\tilde{\mathbf{X}}$ is a real symmetric matrix.

From linear algebra[3] we know that a real symmetric matrix can be diagonalized by

$$\tilde{\mathbf{X}} = \mathbf{P}\,\mathrm{diag}(\lambda_i)\mathbf{P}^{\top} \tag{8.34}$$

where the columns of $\mathbf{P}$ are the eigenvectors of $\tilde{\mathbf{X}}$ and λ_i are the corresponding eigenvalues. Note that the eigenvectors of a real symmetric matrix are orthonormal - they are orthogonal to each other and have unit length. Therefore our definition/choice of $\mathbf{P}$ satisfies 8.32.

[3] see `https://en.wikipedia.org/wiki/Diagonalizable_matrix`

Next, we verify that the covariance matrix of the transformed data is a diagonal matrix.

$$\begin{aligned} \mathbf{S}_{\mathbf{X}'} &= \frac{1}{N}\mathbf{P}^\top \tilde{\mathbf{X}}\mathbf{P} \\ &= \frac{1}{N}\mathbf{P}^\top \mathbf{P}\,\mathrm{diag}(\lambda_i)\mathbf{P}^\top \mathbf{P} \\ &= \frac{1}{N}\,\mathrm{diag}(\lambda_i) \end{aligned} \tag{8.35}$$

We conclude that the projection matrix with the basis vectors equal to the eigenvectors of the (empirical) covariance matrix, results in a diagonal covariance matrix of the transformed data.
Because the diagonal entries of the covariance matrix are describing the variance captured by the corresponding basis vector, and are equal to the scaled (by $\frac{1}{n}$) eigenvalues, we simply have to select the p eigenvectors of the p largest eigenvalues if we have to reduce the dimension from d to p.

8.2. Kernelized PCA

PCA can do linear dimensionality reduction only. In order to be able to do non-linear dimensionality reductions, we extend PCA with kernels and obtain *kernelized PCA* (also called *kernel PCA - KPCA*) [13].
The derivation of kernelized PCA is analog to PCA except that we replace all x_i by $\phi(x_i)$, where ϕ denotes the feature mapping of the kernel k, and all dot products $\phi(x_i)^\top \phi(x_j)$ by $k(x_i, x_j)$. After rearranging some terms[4], we obtain the following eigenvector/value problem

$$\mathbf{K}\vec{\alpha}_k = \lambda_k n\,\vec{\alpha}_k \tag{8.36}$$

where $\mathbf{K}$ denotes the Gram matrix of the training data (we assume that we have n data points) and $\vec{\alpha}_k \in \mathbb{R}^n$ is the coefficient vector of the k-th principal component.
We can compute the projection of a data point x on k-th principal component as

$$\sum_i (\vec{\alpha}_k)_i\, k(x, x_i) \tag{8.37}$$

We can use kernelized PCA on any domain we like, as long as we can define a meaningful kernel in this domain. Furthermore, the dimensionality reduction always yields real-valued vectors, no matter what the original data domain is.

[4] see [13]

8.3. Outlook

In this chapter we learned about dimensionality reduction, in particular about PCA (Principle Component Analysis).
PCA finds orthogonal axes with maximum variance along them. Although PCA is a non-convex problem, we were able to show that it is equivalent to computing the eigenvectors and values of a matrix. Next, we briefly discussed kernelized PCA which allows PCA to find non-linear relations by using kernels.

Besides PCA, there exist many other dimensionality reduction techniques like autoencoders, NMF[5], t-SNE[6], ICA[7] and UMAP[8].

[5]Non-negative Matrix Factorization
[6]t-distributed stochastic neighbor embedding
[7]Independent Component Analysis
[8]Uniform Manifold Approximation and Projection

8.4. Exercises

1. Covariance measures the linear relationship only.
   We assume that we have two random variables X and $Y = X^2$ where $X \sim \mathcal{U}(\delta, -\delta)$. Compute the covariance $\mathrm{Cov}(X, Y)$ between X and Y.

Chapter 9

Clustering

Clustering is all about finding groups (also called *clusters*) of "similar" items. The requirement for a "good" partition into clusters is often described by the *Good Clustering Principle*[6]: "Every pair of points from the same cluster should be closer to each other than any pair of points from different clusters".
Issues in clustering include the measurement of proximity and the shape and number of clusters to compute.

In this chapter we talk about three different models/algorithms for computing clusters - including different types of clusters.

9.1. K-means

The *k-means clustering* problem is given by

$$\min_{\{c_i\},\{\vec{c}_k\}} \sum_k \sum_{i:c_i=k} \|\vec{x_i} - \vec{c}_k\|_2^2 \tag{9.1}$$

The assignment of a data point $\vec{x}_i$ to a cluster is stored in c_i. The center of the k-th cluster is denoted by $\vec{c}_k$. The number of clusters is a hyperparameter and denoted by k.
Note: Because of 9.1, all clusters will be shaped like a *Voronoi cell*[1].

The *k-means algorithm* (also called *Lloyd algorithm*) computes a approximately solution of 9.1 and is described in Algorithm 8.

Note: The k-means problem 9.1 is known to NP-hard (if $k > 1$).

[1] `https://en.wikipedia.org/wiki/Voronoi_diagram`

The algorithm 8 computes a local optimum only. We might want to run the algorithm multiple times and select the best result.

Algorithm 8 K-means clustering algorithm

Input: Data points $\mathcal{D} = \{\vec{x}_i\}$, Number of clusters k
Output: Cluster centers $\{\vec{c}_k\}$ and cluster assignments $\{c_i\}$

1: Init $\vec{c}_k$ (e.g. random, samples from $\mathcal{D}$, ...)
2: **while** Assignments c_i are changing **do**
3: $c_i = \underset{k \in \{1,...,k\}}{\arg\min} \|\vec{x}_i - \vec{c}_k\|_2^2 \quad \forall\, i \in \{1, ..., |\mathcal{D}|\}$ ▷ Assign points to clusters
4: $n_k = \sum_i \mathbb{1}(c_i = k) \quad \forall\, k \in \{1, ..., k\}$ ▷ Count points per cluster
5: $\vec{c}_k = \frac{1}{n_k} \sum_i \mathbb{1}(c_i = k)\vec{x}_i \quad \forall\, k \in \{1, ..., k\}$ ▷ Recompute cluster centers
6: **end while**

In plain English: In the beginning, we initialize[2] the centers $\vec{c}_k$ of the k clusters. We could place them randomly in space or randomly select samples from our data set $\mathcal{X}$ and use them as initial cluster centers.
Next, we repeatedly assign each data point x_i to its nearest cluster by updating c_i. Then we recompute the cluster centers by computing the mean of all points belonging to a cluster (we compute the *center of gravity*). We repeat this procedure until the cluster assignments c_i (c_i stores the cluster id to which the data point x_i is assigned) do not change anymore.

Fig. 9.1 illustrates the resulting clustering of k-means for two different data sets. For each data set we asked k-means to find 3 clusters.

9.1.1. K-means++

Instead of initializing the centers randomly (e.g. by selecting random samples from the data set), the *k-means++* algorithm [1] proposes[3] a "smarter" way of selecting the initial cluster centers before running the k-means algorithm. The algorithm for computing the initial cluster centers is described in Algorithm 9.
In plain English: First, we select one data point from the data set $\mathcal{D}$ uniformly at random and set it as the first cluster center $\vec{c}_1$. Then, we compute the distance d_i of each point $\vec{x}_i \in \mathcal{D}$ to this center $\vec{c}_1$ and select the next cluster center by randomly selecting a point from the data set, where the

[2] e.g. k-means++ see section 9.1.1

[3] k-means++ is often used as the default method for initializing the cluster centers - e.g. scikit-learn [11] does so

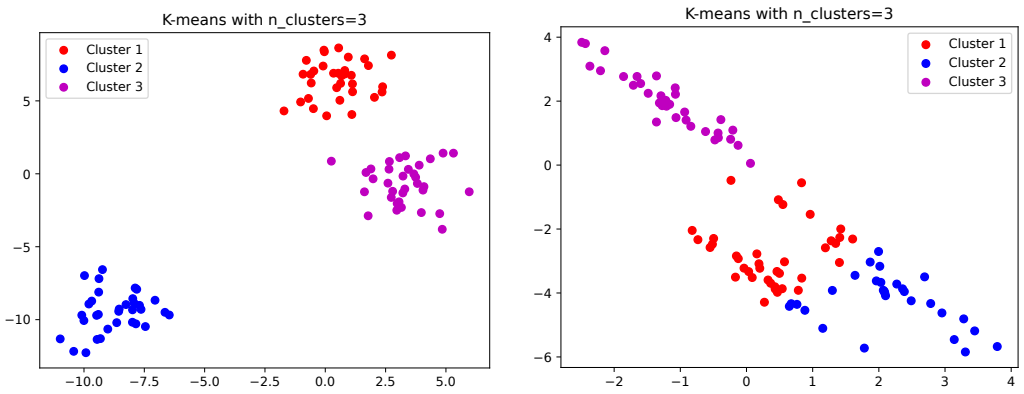


Figure 9.1: k-means fitted to two different data sets

Algorithm 9 K-means++ algorithm

Input: Data points $\mathcal{D} = \{\vec{x}_i\}$, Number of clusters k
Output: Initial cluster centers $\{\vec{c}_k\}$

1: $\vec{c}_1 = \vec{x}_j \quad$ where $j \sim \mathcal{U}(1, |\mathcal{D}|)$ ▷ Select first cluster center
2: $d_i = \|\vec{x}_i - \vec{c}_1\|_2^2 \quad \forall\, i \in \{1, ..., |\mathcal{D}|\}$ ▷ Compute distances
3: $p_i = \frac{d_i}{\sum_i d_i} \quad \forall\, i \in \{1, ..., |\mathcal{D}|\}$ ▷ Compute probabilities
4: **for** $t := 2$ to k **do**
5: $\quad \vec{c}_t = \vec{x}_j \quad$ where $j \sim \mathcal{P}(\{p_i\})$ ▷ Select next cluster center
6: $\quad d_i = \min_{s \in \{1,...,t\}} \|\vec{x}_i - \vec{c}_s\|_2^2 \quad \forall\, i \in \{1, ..., |\mathcal{D}|\}$ ▷ Update distances
7: $\quad p_i = \frac{d_i}{\sum_i d_i} \quad \forall\, i \in \{1, ..., |\mathcal{D}|\}$ ▷ Update probabilities
8: **end for**

probability p_i for $\vec{x}_i$ is $p_i = \frac{d_i}{\sum_i d_i}$ - thus, distant points are more likely to be selected. We then recompute the distance d_i for each point $\vec{x}_i$ to be the distance to the nearest cluster center. Then, we select the next cluster center by randomly selecting a data point from the data set according to the new probabilities p_i. We repeat this procedure until all cluster centers have been chosen.

9.1.2. Voronoi tessellation

Because the k-means algorithm assigns each data point to the nearest center, the data space is partitioned into convex polygons if we use the euclidean distance. Each of this convex polygons defines one cluster.

Given a set of center points $\mathcal{C} = \{\vec{c}_k\}$ with $\vec{c}_k \in \mathbb{R}^d$, the assignment of all points in the space $\mathbb{R}^d$ to the nearest center point is called *Voronoi tessellation*. The Voronoi tessellation partitions the space $\mathbb{R}^d$ into $|\mathcal{C}|$ regions (also called *Voronoi cells*) where each region $\mathcal{R}_j$ is defined as

$$\mathcal{R}_j = \left\{ \vec{x} \in \mathbb{R}^d \mid j = \underset{k \in \{1,\ldots,|\mathcal{C}|\}}{\arg\min}\ d(\vec{x}, \vec{c}_k) \right\} \tag{9.2}$$

where the function d computes the distance between two points. If d is the euclidean distance, the $\mathcal{R}_j$ are convex regions and the boundaries can be interpreted as convex polygons.
See Fig. 9.2 for an illustration of a Voronoi tessellation of a k-means clustering. As we can see, each cluster is a convex polygon and the entire data space is partitioned into convex polygons.

Note: The clusters computed by the k-means algorithm are always convex polygons (if we use the euclidean distance). If we assume that our clusters are not necessarily convex, we should not use the k-means algorithm!

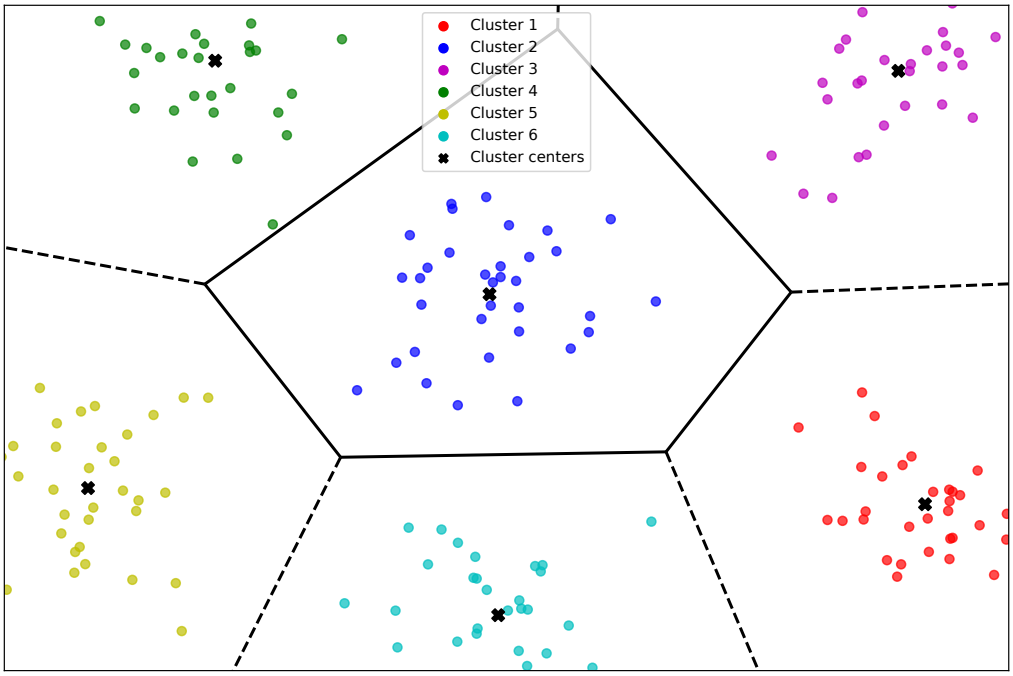


Figure 9.2: Voronoi tessellation of a k-means clustering (data set with 6 clusters)

9.2. Agglomerative Clustering

Agglomerative clustering is an instance of *hierarchical clustering*. The idea behind hierarchical clustering is to organize the data as a *binary tree* (also called *dendogram*) where each level of the tree corresponds to a particular clustering and the data points itself are stored in the leafs of the tree. Therefore we do not need to specify the number of clusters in advance like we did in the k-means algorithm from the previous section.
Hierarchical clustering is usually divided into two different groups. *Agglomerative clustering* where the tree is build from bottom to top (bottom-up) and *divisive clustering* where the tree is computed from top to down. In this section we only talk about agglomerative clustering.

The agglomerative clustering algorithm does not work directly on the data points $\mathcal{D} = \{x_i\}$ but instead on a *dissimilarity matrix* $\mathbf{D}$

$$\begin{aligned} &\mathbf{D} \in \mathbb{R}_+^{n \times n} \\ &d_{i,j} = d_{j,i} = d(x_i, x_j) \end{aligned} \tag{9.3}$$

where $d(x_i, x_j)$ measures the dissimilarity or "distance" between two data points x_i and x_j. Because we do not work on x_i and x_j directly, we do not make any assumptions on their domain. We could use any domain as long as we are able to compute/define a "meaningful" dissimilarity/distance measure on it. Note that this is very similar to the idea of kernels.
Next, we define a couple of different ways for measuring the distance between two clusters $\mathcal{C}_j$ and $\mathcal{C}_k$ (a cluster is a set of points):

1. *Single linkage* defined as

$$d_{SL}(\mathcal{C}_j, \mathcal{C}_k) = \min_{\substack{j \in \mathcal{C}_j, \\ k \in \mathcal{C}_k}} d_{j,k} \tag{9.4}$$

2. *Complete linkage* defined as

$$d_{CL}(\mathcal{C}_j, \mathcal{C}_k) = \max_{\substack{j \in \mathcal{C}_j, \\ k \in \mathcal{C}_k}} d_{j,k} \tag{9.5}$$

3. *Average linkage* defined as

$$d_{AL}(\mathcal{C}_j, \mathcal{C}_k) = \frac{1}{n_{\mathcal{C}_j} n_{\mathcal{C}_k}} \sum_{j \in \mathcal{C}_j} \sum_{k \in \mathcal{C}_k} d_{j,k} \tag{9.6}$$

   where $n_{\mathcal{C}_j} = |\mathcal{C}_j|$ and $n_{\mathcal{C}_k} = |\mathcal{C}_k|$.

4. *Ward's linkage* defined as

$$d_{WL}(\mathcal{C}_j, \mathcal{C}_k) = \sum_{x \in \mathcal{C}_j \cup \mathcal{C}_k} \|x - \mu_{\mathcal{C}_j \cup \mathcal{C}_k}\|_2^2 - \sum_{x \in \mathcal{C}_j} \|x - \mu_{\mathcal{C}_j}\|_2^2 - \sum_{x \in \mathcal{C}_k} \|x - \mu_{\mathcal{C}_k}\|_2^2 \tag{9.7}$$

   where $\mu_{\mathcal{C}} = \frac{1}{n_{\mathcal{C}}} \sum_{x \in \mathcal{C}} x$ denotes the center of cluster $\mathcal{C}$. The size (number of elements) of a cluster C is denoted by $n_{\mathcal{C}} = |\mathcal{C}|$.

   Attention: Ward's method can only be used if we can average data points! Thus, we need more information/capabilities than just the dissimilarity matrix.

Finally, the agglomerative clustering algorithm is described in Algorithm 10.

In plain English: We start by putting each data point $x_i \in \mathcal{D}$ in its own cluster $\mathcal{C}_i$. We then repeatedly merge two clusters into a new cluster until all data points are in one cluster. We merge the two clusters which are closest to each other, where we determine proximity by using one of the previously

Algorithm 10 Agglomerative clustering algorithm

Input: Dissimilarity matrix $\mathbf{D}$, d measures the distance between two clusters
Output: Hierarchical clustering (e.g. dendogram)

1: Put each data point into its own cluster $\mathcal{C}_i$. Set of all clusters $\mathcal{C} = \{\mathcal{C}_i\}$
2: **while** $|\mathcal{C}| > 1$ **do**
3: $\quad \mathcal{C}_j, \mathcal{C}_k = \min_{\substack{\mathcal{C}_j \in \mathcal{C}, \\ \mathcal{C}_k \in \mathcal{C}}} d(\mathcal{C}_j, \mathcal{C}_k)$ ▷ Find two closest clusters
4: $\quad \mathcal{C} = \mathcal{C} \setminus \{\mathcal{C}_j, \mathcal{C}_k\} \cup \{\mathcal{C}_j \cup \mathcal{C}_k\}$ ▷ Merge the two clusters
5: **end while**

defined metrics (single-, complete- or average linkage). Each merging step is memorized, such that we can reconstruct the hierarchy afterwards when computing the dendogram.
Because we have to recompute the distances between the clusters in each iteration, a naive implementation has cubic complexity. However, there are clever tricks for efficiently recomputing cluster distances by making use of previously computed distances - e.g. Lance-Williams formula[4] for Ward's linkage.

In Fig. 9.5 and Fig. 9.4, we can observe that different *linkage criterions* might result in different clusterings.

Note: The dendogram contains more information than a just a hierarchical ordering. The difference in heights (y axis) of two nodes/clusters indicates the dissimilarity of these two clusters. Therefore, the height can/should be considered when selecting the number of clusters.

[4] see `https://en.wikipedia.org/wiki/Ward%27s_method`

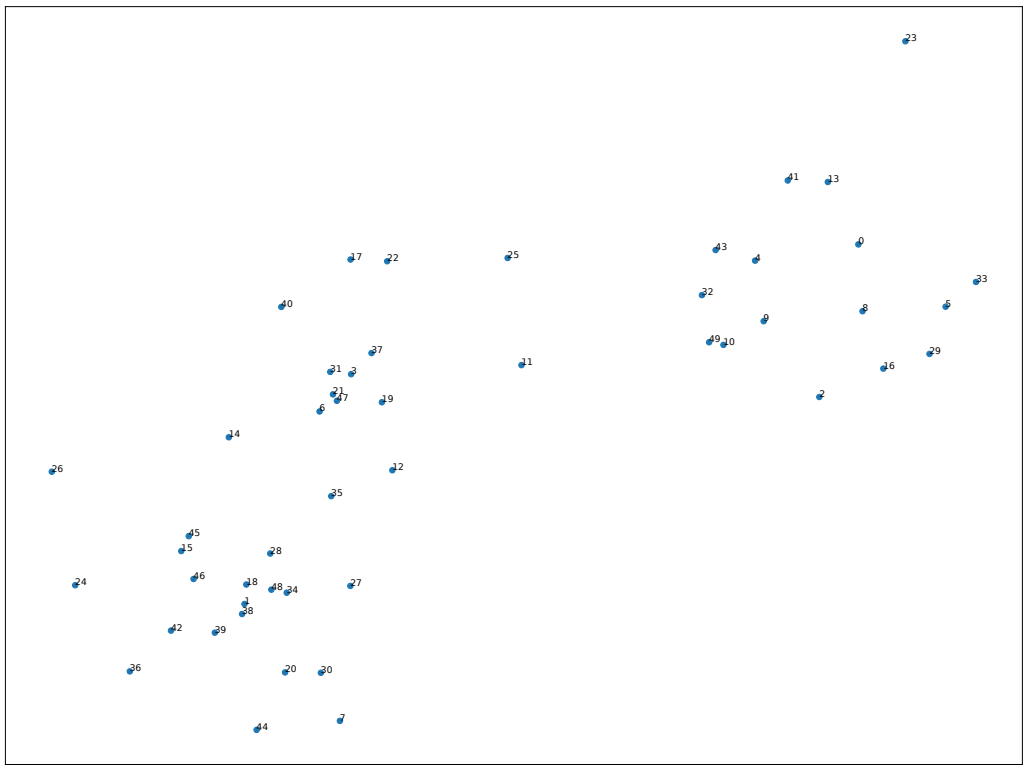


Figure 9.3: Data set for agglomerative clustering (Fig. 9.4, 9.6, 9.7)

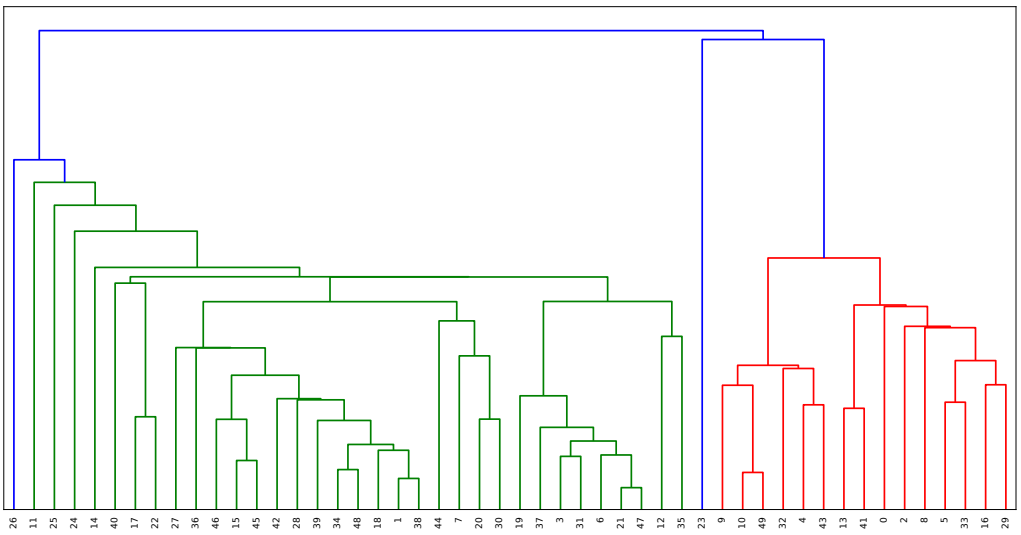


Figure 9.4: Single linkage

Figure 9.5: Agglomerative clustering (*euclidean distance*) of data set Fig. 9.3

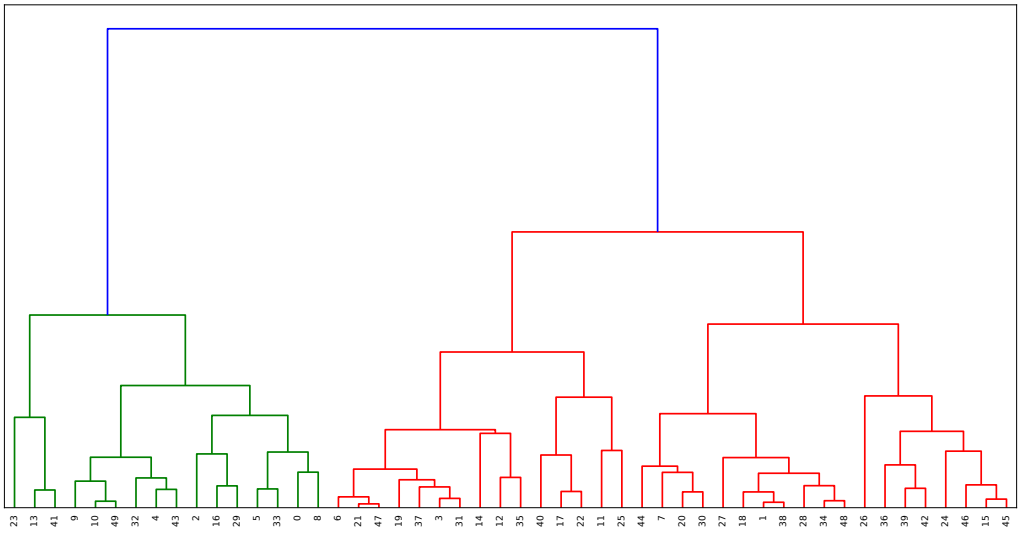


Figure 9.6: Complete linkage

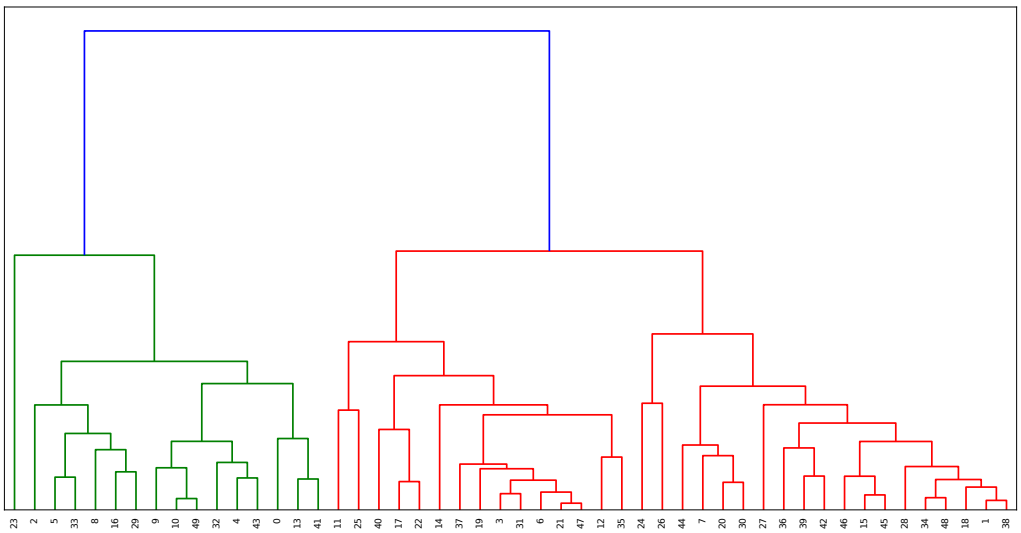


Figure 9.7: Average linkage

9.3. DBSCAN

Density based spatial clustering of applications with noise (*DBSCAN*) belongs to the family of *density based clustering* methods.
Before we can discuss the algorithm, we need a couple of definitions:

1. The *epsilon neighborhood* of a point x is defined as

$$\mathcal{E}(x, \mathcal{D}, \epsilon) = \{x_j \in \mathcal{D} \mid d(x_j, x) \leq \epsilon\} \tag{9.8}$$

   where d is a function for measuring the "distance" between two points[5].

2. The set of *core points* is defined as

$$\mathcal{C}(\mathcal{D}) = \{x_i \in \mathcal{D} \mid \ |\mathcal{E}(x_i, \mathcal{D}, \epsilon)| \geq M\} \tag{9.9}$$

   where M is some fixed integer.

3. The undirected *core graph* is defined as

$$G_{\text{core}} = (\mathcal{V}_{\text{core}}, \mathbf{E}_{\text{core}}) \tag{9.10}$$

   where $\mathcal{V}_{\text{core}} = \mathcal{C}(\mathcal{D})$ and the edge matrix[6]

$$(\mathbf{E}_{\text{core}})_{i,j} = \begin{cases} 1 & \text{if } x_i \in \mathcal{E}(x_j, \mathcal{C}, \epsilon) \\ 0 & \text{otherwise} \end{cases} \tag{9.11}$$

The DBSCAN algorithm is described in Algorithm 11.

Algorithm 11 DBSCAN algorithm

Input: Data set $\mathcal{D} = \{x_i\}$, Parameter ϵ and M
Output: Clustering

1: Compute for each point $x_i \in \mathcal{D}$ the epsilon neighborhood $\mathcal{E}(x_i, \mathcal{D}, \epsilon)$.
2: Compute the set of *core points* $\mathcal{C}(\mathcal{D})$.
3: Compute the the undirected *core graph* G_{core}.
4: Each *connected component* G_i of G_{core} becomes a cluster c_i.
5: Assign each remaining data point $x_j \in \mathcal{D} \setminus \mathcal{C}(\mathcal{D})$ to the nearest cluster c_i, determined by the nearest core point $x_i \in \mathcal{C}(\mathcal{D})$, if $d(x_j, x_i) \leq \epsilon$.
6: All unassigned points are declared to be noise.

[5] **Note:** We are not limited to vectorial data, as long as we can define a meaningful distance measurement on the domain - this is very similar to the kernel approach!
[6] this is not a practical implementation!

In plain English: First, we compute for each point the set of points which are ϵ-near. Next, we select all points which have at least M other points in their ϵ-neighborhood and call them *core points*.
We then construct an undirected graph from these core points, where the vertices are the core points and two points $x_i, x_j \in \mathcal{C}(\mathcal{D})$ are connected by an edge if the point x_i is in the ϵ-neighborhood of the point x_j. Each connected component G_i of this graph defines/creates a cluster c_i.
Finally, we assign all remaining points (non core points) $x_j \in \mathcal{D} \setminus \mathcal{C}(\mathcal{D})$ to the nearest cluster if the distance to this nearest cluster is smaller or equal to ϵ. The nearest cluster is determined by the core point with minimum distance to x_j. Points with a distance larger than ϵ are not assigned to any clusters and are declared to be noise.

Fig. 9.8 illustrates the DBSCAN clustering for two different data sets. Note that the algorithm automatically determines the number of clusters.

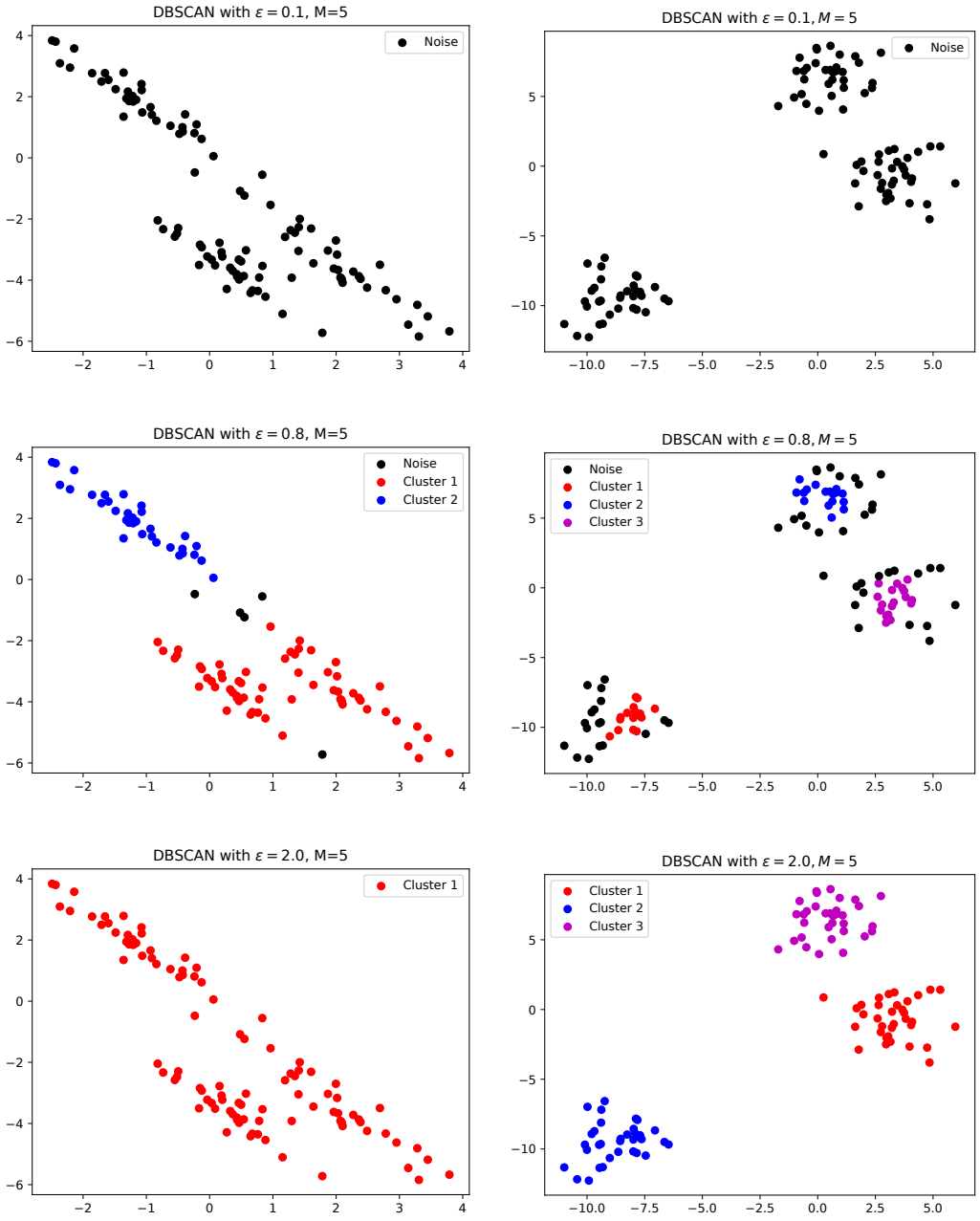


Figure 9.8: DBSCAN with different ϵ fitted to two different data sets

9.4. Spectral clustering

Spectral clustering (e.g. see [17] for an excellent tutorial) is a clustering method that works on a similarity matrix instead of the data directly.
Spectral clustering first constructs a graph from the similarity matrix and then uses the eigenvectors of the graph Laplacian matrix to perform a dimensionality reduction of the data. Then, we apply some other clustering algorithm (e.g. k-means 9.1) to this new data set.

The spectral clustering algorithm is described in Algorithm 12.

Algorithm 12 Spectral clustering algorithm

Input: Similarity matrix $\mathbf{A}$, Number of clusters k, Number of eigenvectors s

Output: Clustering

1: $\mathbf{D} = \text{diag}(d_i)$ where $d_i = \sum_j (\mathbf{A})_{i,j}$ ▷ Compute the degree matrix
2: $\mathbf{L} = \mathbf{D} - \mathbf{A}$ ▷ Compute the graph Laplacian matrix
3: Compute all eigenvectors $\vec{v}_k$ and eigenvalues λ_k of $\mathbf{L}$
4: $\mathbf{X} = \begin{pmatrix} | & & | \\ \vec{v}_1 & \cdots & \vec{v}_s \\ | & & | \end{pmatrix}$ ▷ Choose the s largest eigenvectors
5: $\{c_i\} = \text{k-means}(\mathbf{X}, k)$ ▷ Interpret the rows of $\mathbf{X}$ as data points and cluster them by using the k-means algorithm
6: Return the cluster assignments $\{c_i\}$ as a clustering of the original data

See Fig.9.9 for an illustration of spectral clustering applied to a toy example. The similarity matrix is computed by using the ϵ-neighborhood with $\epsilon = 0.008$ - two points are connected if the euclidean distance between them is less or equal to ϵ.
Note that the example in Fig. 9.9 is a toy example only and is constructed to visualize the spectral clustering method. In reality we might need more than two eigenvectors and choosing the right hyper parameter might not be that easy!

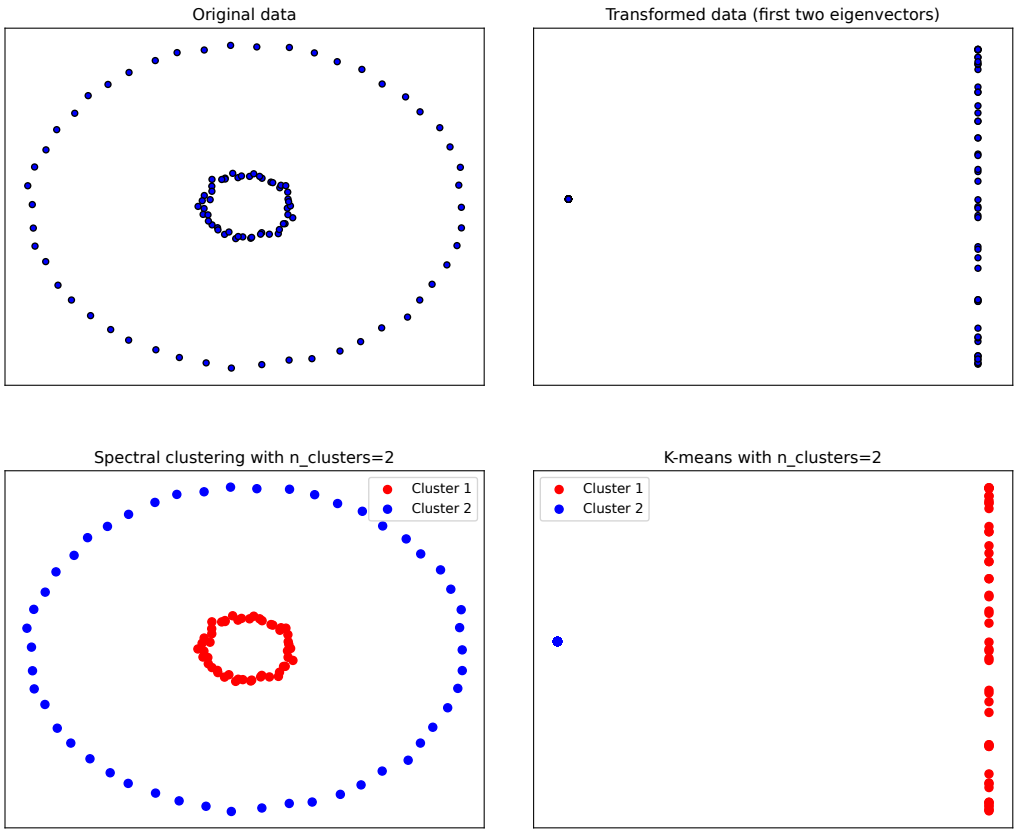


Figure 9.9: Spectral clustering applied to a toy example.
The original data is plotted in the upper left plot. The transformed data (by using the first two eigenvectors of the graph laplacian) is plotted in the upper right plot.
The transformed data is clustered with the k-means algorithm (k=2) and the result is plotted in the lower right plot. Transferring the cluster labels to the original data yields to the clustering of the original data, which is plotted in the lower left plot.

9.5. Gaussian mixture model

The *Gaussian mixture model* is a model for *density estimation*. That is we want to learn/estimate the underlying density function of a given data set $\mathcal{D} = \{\vec{x}_i\}$ where $\vec{x}_i \in \mathbb{R}^d$.

We assume that the density is a mixture of Gaussian distributions[7]. We assume that the mixture is linear, hence it is a weighted sum of Gaussian densities.
We define the probability density of a data point $\vec{x}_i$ as

$$p(\vec{x}_i \mid \{\vec{\mu}_k\}, \{\mathbf{\Sigma}_k\}, \{\pi_k\}) = \sum_k \pi_k \mathcal{N}(\vec{x}_i \mid \vec{\mu}_k, \mathbf{\Sigma}_k) \tag{9.12}$$

where π_k are the mixture coefficients and act as a prior of the individual Gaussians.
Therefore, we find that the likelihood (see section B.13.2) for a given data set $\mathcal{D}$ is

$$\begin{aligned} p(\mathcal{D} \mid \{\vec{\mu}_k\}, \{\mathbf{\Sigma}_k\}, \{\pi_k\}) &= \prod_i p(\vec{x}_i \mid \{\vec{\mu}_k\}, \{\mathbf{\Sigma}_k\}, \{\pi_k\}) \\ &= \prod_i \sum_k \pi_k \mathcal{N}(\vec{x}_i \mid \vec{\mu}_k, \mathbf{\Sigma}_k) \end{aligned} \tag{9.13}$$

where we assume that all data points $\vec{x}_i \in \mathcal{D}$ are i.i.d.

The final optimization problem for estimating the parameters of the Gaussian mixture model is to maximize the log-likelihood and is stated in 9.14.

$$\begin{aligned} &\underset{\{\vec{\mu}_k\},\{\mathbf{\Sigma}_k\},\{\pi_k\}}{\arg\max} \sum_i \log\left(\sum_k \pi_k \mathcal{N}(\vec{x}_i \mid \vec{\mu}_k, \mathbf{\Sigma}_k)\right) \\ &\text{s.t.} \\ &0 \leq \pi_k \leq 1 \quad \text{and} \quad \sum_k \pi_k = 1 \end{aligned} \tag{9.14}$$

Unfortunately, 9.14 in general is a difficult optimization problem - in particular it is non-convex. There is no algorithm for solving it exactly. We compute a (local) optimum of 9.14 by using the Expectation-Maximization algorithm. The algorithm is described in Algorithm 13.
A detailed discussion of the general Expectation-Maximization algorithm is the topic of section 9.5.1.
In plain English: First, we compute for each Gaussian and each data point the probability $p_{k|i}$ of the i-th data point under the k-th Gaussian. We can interpret $p_{k|i}$ as the probability that data point $\vec{x}_i$ belongs (is assigned) to the k-th Gaussian. Then, we recompute the parameters $\vec{\mu}_k$, $\mathbf{\Sigma}_k$ and π_k to

[7] we can approximate any density arbitrarily well by using enough Gaussian distributions

Algorithm 13 Gaussian mixture model: EM algorithm

Input: Data points $\mathcal{D} = \{\vec{x}_i\}$, Number of Gaussians k
Output: Centers $\{\vec{\mu}_k\}$, covariance matrices $\{\mathbf{\Sigma}_k\}$ and probabilities $\{\pi_k\}$

1: Initialize parameters $\vec{\mu}_k, \mathbf{\Sigma}_k, \pi_k$
2: **repeat**
3: $p_{k|i} = \frac{\pi_k \mathcal{N}(\vec{x}_i|\vec{\mu}_k, \mathbf{\Sigma}_k)}{\sum_k \pi_k \mathcal{N}(\vec{x}_i|\vec{\mu}_k, \mathbf{\Sigma}_k)}$ ▷ Responsibility of cluster k for the i-th data point
4: $n_k = \sum_i p_{k|i}$ ▷ Estimate parameters
5: $\vec{\mu}_k = \frac{1}{n_k} \sum_i p_{k|i}\, \vec{x}_i$
6: $\mathbf{\Sigma}_k = \frac{1}{n_k} \sum_i p_{k|i} (\vec{x}_i - \vec{\mu}_k)(\vec{x}_i - \vec{\mu}_k)^\top$
7: $\pi_k = \frac{n_k}{|\mathcal{D}|}$
8: **until** convergence

maximize the likelihood of the data points $\vec{x}_i$ under the assignments $p_{k|i}$ - note that this is very similar to the k-means procedure, but instead of a hard assignment we use the soft assignment $p_{k|i}$.
We repeat this procedure until convergence. A convergence criterion might be the maximum number of iterations, the amount of change in the parameters or the log-likelihood value.

Because we typically use more than one Gaussian, we can model much more complicated densities and are no longer restricted to elliptic shapes. See Fig. 9.10 for an example of a Gaussian mixture model fitted to a data set.

9.5.1. Details on the EM-algorithm

In this section we introduce the *expectation-maximization algorithm* (short *EM-algorithm*) from a very theoretical point of view. In subsection 9.5.1.1, we derive the EM-algorithm for a gaussian mixture model.

We assume that we have two random variables X and Z. However, we only observed instances of X wheras we never observe instances of Z. Because of this, we call Z a *latent random variable*. Furthermore, we assume that X depends on Z. That is, the probability distribution of X depends on Z. If we assume that Z *is discrete*, we can write the parameterized (conditional) likelihood of X as

$$p_X(x \mid \vec{\theta}) = \sum_z p_{X,Z}(x, z \mid \vec{\theta}) \tag{9.15}$$

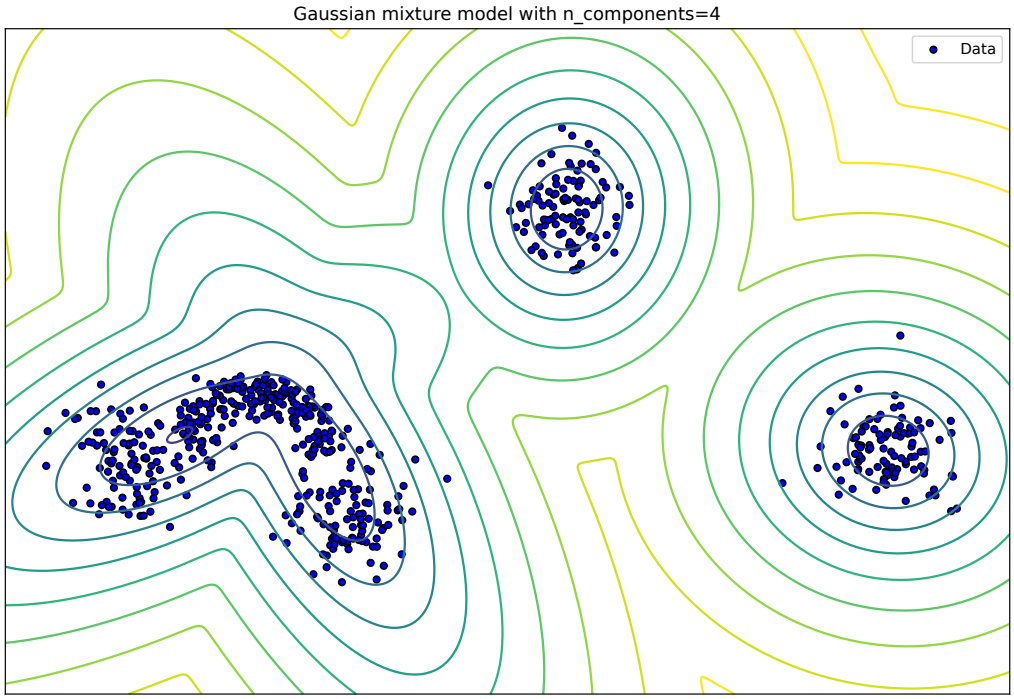


Figure 9.10: Gaussian mixture model fitted to a data set

where $\vec{\theta}$ denotes the parameters of the distribution.
We claim, that we can write the (conditional) log-likelihood as

$$\log\left(p_X(x \mid \vec{\theta})\right) = \Psi(q_Z, \vec{\theta}) + \mathrm{D}_{\mathrm{KL}}(q_Z \,\|\, p_{Z|X}) \tag{9.16}$$

where we define

$$\Psi(q_z, \vec{\theta}) = \sum_z q_Z(z) \log\left(\frac{p_{X,Z}(x, z \mid \vec{\theta})}{q_Z(z)}\right) \tag{9.17}$$

We can prove 9.16 by applying a bunch of simple algebraic manipulations.

$$\begin{aligned}
\log\left(p_X(x \mid \vec{\theta})\right) &= \Psi(q_z, \vec{\theta}) + \mathrm{D}_{\mathrm{KL}}(q_Z \,\|\, p_{Z|X}) \\
&= \sum_z q_Z(z) \log\left(\frac{p_{X,Z}(x, z \mid \vec{\theta})}{q_Z(z)}\right) - \sum_z q_Z(z) \log\left(\frac{p_{Z|X}(z \mid x, \vec{\theta})}{q_Z(z)}\right) \\
&= \sum_z q_Z(z)\Big(\log\left(p_{X,Z}(x, z \mid \vec{\theta})\right) - \log\left(q_Z(z)\right)\Big) -
\end{aligned}$$

$$
\begin{aligned}
&\sum_z q_Z(z)\Big(\log\Big(p_{Z|X}(z \mid x, \vec{\theta})\Big) - \log\big(q_Z(z)\big)\Big) \\
&= \sum_z q_Z(z)\log\Big(p_{X,Z}(x, z \mid \vec{\theta})\Big) - \sum_z q_Z(z)\log\big(q_Z(z)\big) - \\
&\quad \sum_z q_Z(z)\log\Big(p_{Z|X}(z \mid x, \vec{\theta})\Big) + \sum_z q_Z(z)\log\big(q_Z(z)\big) \\
&= \sum_z q_Z(z)\Big(\log\Big(p_{X,Z}(x, z \mid \vec{\theta})\Big) - \log\Big(p_{Z|X}(z \mid x, \vec{\theta})\Big)\Big) \\
&= \sum_z q_Z(z)\log\left(\frac{p_{X,Z}(x, z \mid \vec{\theta})}{p_{Z|X}(z \mid x, \vec{\theta})}\right) \\
&= \sum_z q_Z(z)\log\left(\frac{p_{Z|X}(z \mid x, \vec{\theta})p_X(x \mid \vec{\theta})}{p_{Z|X}(z \mid x, \vec{\theta})}\right) \\
&= \sum_z q_Z(z)\log\Big(p_X(x \mid \vec{\theta})\Big) \\
&= \log\Big(p_X(x \mid \vec{\theta})\Big)\sum_z q_Z(z) \\
&= \log\Big(p_X(x \mid \vec{\theta})\Big)
\end{aligned}
$$

Because $\mathrm{D}_{\mathrm{KL}}(q_Z \,\|\, p_{Z|X}) \geq 0$ it holds that

$$
\sum_z q_Z(z)\log\left(\frac{p_{X,Z}(x, z \mid \vec{\theta})}{q_Z(z)}\right) \leq \log\Big(p_X(x \mid \vec{\theta})\Big) \tag{9.18}
$$

Equality holds, if and only if $\mathrm{D}_{\mathrm{KL}}(q_Z \,\|\, p_{Z|X}) = 0$. This can be achieved by setting

$$
q_Z(z) = p_{Z|X}(z \mid x, \vec{\theta}_t) \tag{9.19}
$$

where we fixed the parameter $\vec{\theta} = \vec{\theta}_t$ with our current estimate $\vec{\theta}_t$.
It follows that

$$\begin{aligned}
\log\left(p_X(x \mid \vec{\theta})\right) &= \Psi\left(p_{Z|X}(z \mid x, \vec{\theta}_t), \vec{\theta}\right) \\
&= \sum_z p_{Z|X}(z \mid x, \vec{\theta}_t) \log\left(\frac{p_{X,Z}(x, z \mid \vec{\theta})}{p_{Z|X}(z \mid x, \vec{\theta})}\right) \\
&= \sum_z p_{Z|X}(z \mid x, \vec{\theta}_t) \log\left(p_{X,Z}(x, z \mid \vec{\theta})\right) - \\
&\quad \sum_z p_{Z|X}(z \mid x, \vec{\theta}_t) \log\left(p_{Z|X}(z \mid x, \vec{\theta}_t)\right) \\
&= \mathcal{Q}_{\vec{\theta}_t}(\vec{\theta}) + \mathrm{H}\left(p_{Z|X}(z \mid x, \vec{\theta}_t)\right)
\end{aligned} \tag{9.20}$$

where we defined $\mathcal{Q}(\vec{\theta}, \vec{\theta}_t) = \sum_z p_{Z|X}(z \mid x, \vec{\theta}_t) \log\left(p_{X,Z}(x, z \mid \vec{\theta})\right)$ and the constant (does not depend of $\vec{\theta}$) entropy of $p_{Z|X}(z \mid x, \vec{\theta}_t)$.

$$\mathrm{H}\left(p_{Z|X}(z \mid x, \vec{\theta}_t)\right) = -\sum_z p_{Z|X}(z \mid x, \vec{\theta}_t) \log\left(p_{Z|X}(z \mid x, \vec{\theta}_t)\right) \tag{9.21}$$

Note that we can interpret $\mathcal{Q}_{\vec{\theta}_t}(\vec{\theta})$ as the conditional expectation of the log-likelihood $\log\left(p_{X,Z}(x, z \mid \vec{\theta})\right)$ over $Z \mid X, \vec{\theta}_t$.

$$\mathcal{Q}_{\vec{\theta}_t}(\vec{\theta}) = \mathbb{E}_{Z|X,\vec{\theta}_t}\left[\log\left(p_{X,Z}(x, z \mid \vec{\theta})\right)\right] \tag{9.22}$$

Now, that we found the optimal distribution q_Z, the next step is to compute new optimal parameters $\vec{\theta}_{t+1}$. Because we used the current estimate $\vec{\theta}_t$ for constructing the optimal q_Z, we have to maximize 9.20 over $\vec{\theta}$.

$$\begin{aligned}
\vec{\theta}_{t+1} &= \arg\max_{\vec{\theta} \in \Theta} \mathcal{Q}_{\vec{\theta}_t}(\vec{\theta}) + \mathrm{H}\left(p_{Z|X}(z \mid x, \vec{\theta}_t)\right) \\
&= \arg\max_{\vec{\theta} \in \Theta} \mathcal{Q}_{\vec{\theta}_t}(\vec{\theta})
\end{aligned} \tag{9.23}$$

Because the Kullback-Leibler divergence $\mathrm{D}_{\mathrm{KL}}(q_Z \,\|\, p_{Z|X})$ is always greater or equal to zero, we know that

$$\Psi(q_z, \vec{\theta}_{t+1}) \leq \log\left(p_X(x \mid \vec{\theta}_{t+1})\right) \tag{9.24}$$

If the Kullback-Leibler divergence is zero - equivalent that the estimate of the latent variables do not change, we found an optimum and return $\vec{\theta}_{t+1}$ as

the final estimate. It can be shown that the likelihood never decreases.

The EM-algorithm is simply repeating the procedure of computing q_Z based on the current estimate $\vec{\theta}_t$ and then computing a new estimate $\vec{\theta}_{t+1}$ of $\vec{\theta}$ by maximizing $\mathcal{Q}_{\vec{\theta}_t}(\vec{\theta})$. In this way we maximize the log-likelihood $\log\left(p_X(x \mid \vec{\theta})\right)$.

The final EM-algorithm is described in Algorithm 14.
Note: The EM-algorithm computes a *local optimum only*.

Algorithm 14 Expectation-Maximization algorithm

Input: Parameterized joint and marginal distributions $p(X, Z \mid \vec{\theta})$, $p(Z \mid X, \vec{\theta})$ and $p(X \mid \vec{\theta})$ of observed variables X and latent variables Z
Output: Estimated parameters $\vec{\theta}$ and latent variables z_i

1: Initialize parameters $\vec{\theta}$
2: **repeat**
3: Compute $q_Z(z) = p(Z \mid X, \vec{\theta})$ ▷ E-Step
4: $\vec{\theta} = \underset{\vec{\theta} \in \Theta}{\arg\max} \sum_z q_Z(z) \log\left(p(X, Z \mid \vec{\theta})\right)$ ▷ M-Step
5: **until** convergence

9.5.1.1 EM-algorithm for Gaussian mixture model

In this section we derive the expectation-maximization algorithm for a gaussian mixture model.

We assume, that we have a data set of real-valued vectors $\mathcal{D} = \{\vec{x}_i\}$ where $\vec{x}_i \in \mathbb{R}^d$. Furthermore, we assume that each $\vec{x}_i$ is associated with an unobserved/unknown latent vector $\vec{z}_i \in \{0, 1\}^k$ where we assume that exactly one entry in $\vec{z}_i$ is equal to one and all other entries are equal to zero - thus, $\vec{z}_i$ can be interpreted as a "one hot encoding".
In a gaussian mixture model, we have a set of k gaussian distributions and we assume that each data point is generated/sampled from one of these distributions. The specific/responsible distribution of a data point is determined by the entry with a one in the data points latent vector.

Because the latent vectors are unobserved, we treat them as random variables - to be more precise, we treat them as discrete random vectors.
We define the probability that the k-th entry of a latent vector $\vec{z}$ is equal to

one as

$$p\Big((\vec{z})_k = 1\Big) = \pi_k \tag{9.25}$$

Because we assumed that exactly one entry in $\vec{z}$ is equal to one and all other entries are equal to zero, we can write the probability of a latent vector $\vec{z}$ as

$$p(\vec{z}) = \prod_k \pi_k^{(\vec{z})_k} \tag{9.26}$$

From the properties of a probability distribution B.23, we know that

$$\sum_k p\Big((\vec{z})_k = 1\Big) = \sum_k \pi_k = 1 \tag{9.27}$$

The probability density of a data point $\vec{x}$ under the k-th gaussian distribution is given as

$$p_{\vec{\theta}}(\vec{x} \mid (\vec{z})_k = 1) = \mathcal{N}(\vec{x} \mid \vec{\mu}_k, \mathbf{\Sigma}_k) \tag{9.28}$$

where we somehow encoded all parameters like $\{\pi_k\}$, $\{\mu_k\}$ and $\{\mathbf{\Sigma}_k^{-1}\}$ in $\vec{\theta}$.

Because we assumed that a data point is generated by exactly one distribution, we can write the conditional probability density of a point $\vec{x}$ as

$$p_{\vec{\theta}}(\vec{x} \mid \vec{z}) = \prod_k \mathcal{N}(\vec{x} \mid \vec{\mu}_k, \mathbf{\Sigma}_k)^{(\vec{z})_k} \tag{9.29}$$

The marginal probability density of a data point $\vec{x}$ can be written as

$$\begin{aligned} p_{\vec{\theta}}(\vec{x}) &= \sum_k p_{\vec{\theta}}(\vec{x} \mid (\vec{z})_k = 1) p\Big((\vec{z})_k = 1\Big) \\ &= \sum_k \pi_k \mathcal{N}(\vec{x} \mid \mu_k, \mathbf{\Sigma}_k) \end{aligned} \tag{9.30}$$

By applying the log to the conditional probability density of a data point $\vec{x}$, we obtain

$$\log\Big(p_\theta(\vec{x} \mid (\vec{z})_k = 1)\Big) = -\frac{1}{2}\log\Big((2\pi)^d \det(\mathbf{\Sigma}_k)\Big) - \frac{1}{2}(\vec{x} - \vec{\mu}_k)\mathbf{\Sigma}_k^{-1}(\vec{x} - \vec{\mu}_k) \tag{9.31}$$

Because of Baye's theorem, we know that we can write the conditional probability that the k-th entry of the latent vector $\vec{z}$ is equal to one after observing the associated data point $\vec{x}$ as

$$p_{\vec{\theta}}\Big((\vec{z})_k = 1 \mid \vec{x}\Big) = \frac{p_{\vec{\theta}}(\vec{x} \mid (\vec{z})_k = 1) p\Big((\vec{z})_k = 1\Big)}{p_{\vec{\theta}}(\vec{x})} \tag{9.32}$$

where the numerator is equal to the joint distribution of the data point $\vec{x}$ and the k-th entry of the latent vector $\vec{z}$ being equal to one. Thus

$$p_{\vec{\theta}}(\vec{x}, (\vec{z})_k = 1) = p_{\vec{\theta}}(\vec{x} \mid (\vec{z})_k = 1)p\Big((\vec{z})_k = 1\Big) \tag{9.33}$$

So far, we considered one data point only. Next, we define and derive all necessary equations for the set of data points $\mathcal{D}$.
We assume that the prior probability of a specific distribution in the mixture model is independent of the data point.

$$p\Big((\vec{z}_i)_k = 1\Big) = \pi_k \quad \forall\, i \tag{9.34}$$

Next, we assume that all data points $\vec{x}_i \in \mathcal{D}$ are i.i.d. Therefore, the likelihood of the data points and their associated latent vectors can be written as

$$\begin{aligned}
\mathrm{L}_{\mathcal{D}}(\vec{\theta}) &= \prod_i p_{\vec{\theta}}(\vec{x}_i, \vec{z}_i) \\
&= \prod_i p_{\vec{\theta}}(\vec{x}_i \mid \vec{z}_i)p(\vec{z}_i) \\
&= \prod_i \prod_k \Big(p_{\vec{\theta}}(\vec{x}_i \mid (\vec{z}_i)_k = 1)p\big((\vec{z}_i)_k = 1\big)\Big)^{(\vec{z}_i)_k} \\
&= \prod_i \prod_k \Big(\pi_k \mathcal{N}(\vec{x}_i \mid \mu_k, \mathbf{\Sigma}_k)\Big)^{(\vec{z}_i)_k}
\end{aligned} \tag{9.35}$$

By applying the log to 9.35, we obtain the log-likelihood

$$\begin{aligned}
\mathrm{LL}_{\mathcal{D}}(\vec{\theta}) &= \log\Big(\mathrm{L}_{\mathcal{D}}(\vec{\theta})\Big) \\
&= \log\left(\prod_i \prod_k \Big(\pi_k \mathcal{N}(\vec{x}_i \mid \mu_k, \mathbf{\Sigma}_k)\Big)^{(\vec{z}_i)_k}\right) \\
&= \sum_i \log\left(\prod_k \Big(\pi_k \mathcal{N}(\vec{x}_i \mid \mu_k, \mathbf{\Sigma}_k)\Big)^{(\vec{z}_i)_k}\right)
\end{aligned} \tag{9.36}$$

By applying 9.32 to the i-th data point $\vec{x}_i$, we obtain

$$\begin{aligned}
p_{k|i} &= p\Big((\vec{z}_i)_k = 1 \mid \vec{x}_i\Big) \\
&= \frac{p_{\vec{\theta}}(\vec{x}_i \mid (\vec{z})_k = 1)p\Big((\vec{z})_k = 1\Big)}{\sum_j p_{\vec{\theta}}(\vec{x}_i \mid (\vec{z})_j = 1)p\Big((\vec{z})_j = 1\Big)} \\
&= \frac{\pi_k \mathcal{N}(\vec{x}_i \mid \mu_k, \mathbf{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\vec{x}_i \mid \mu_j, \mathbf{\Sigma}_j)}
\end{aligned} \tag{9.37}$$

Note that this is exactly the formula we used in the E-step of algorithm 13 where we estimate the responsibility of the k-th gaussian for the i-th data point.

By plugging the log-likelihood 9.36 into the definition of the $\mathcal{Q}$ function 9.22, we obtain

$$\begin{aligned}
\mathcal{Q}_{\vec{\theta}_t}(\vec{\theta}) &= \mathbb{E}_{Z|X,\vec{\theta}_t}\Big[\mathrm{LL}_{\mathcal{D}}(\vec{\theta})\Big] \\
&= \mathbb{E}_{Z|X,\vec{\theta}_t}\Big[\sum_i \log\left(\prod_k \Big(\pi_k \mathcal{N}(\vec{x}_i \mid \mu_k, \mathbf{\Sigma}_k)\Big)^{(\vec{z}_i)_k}\right)\Big] \\
&= \sum_i \mathbb{E}_{Z|X,\vec{\theta}_t}\Big[\log\left(\prod_k \Big(\pi_k \mathcal{N}(\vec{x}_i \mid \mu_k, \mathbf{\Sigma}_k)\Big)^{(\vec{z}_i)_k}\right)\Big] \\
&= \sum_i \sum_k p_{k|i} \log\Big(\pi_k \mathcal{N}(\vec{x}_i \mid \mu_k, \mathbf{\Sigma}_k)\Big) \\
&= \sum_i \sum_k p_{k|i} \log(\pi_k) + p_{k|i} \log\Big(\mathcal{N}(\vec{x}_i \mid \mu_k, \mathbf{\Sigma}_k)\Big) \\
&= \sum_i \sum_k p_{k|i} \log(\pi_k) - p_{k|i}\frac{1}{2}\log\Big((2\pi)^d \det(\mathbf{\Sigma}_k)\Big) - p_{k|i}\frac{1}{2}(\vec{x}-\vec{\mu}_k)\mathbf{\Sigma}_k^{-1}(\vec{x}-\vec{\mu}_k)
\end{aligned} \tag{9.38}$$

The M-step of alogrithm 13 can be obtained by maximizing the $\mathcal{Q}$ function 9.38 over all parameters - all parameters $\{\pi_k\}$, $\{\mu_k\}$ and $\{\mathbf{\Sigma}_k^{-1}\}$ are somehow encoded in $\vec{\theta}$.

$$\vec{\theta}_{t+1} = \arg\max_{\vec{\theta}\in\Theta} \mathcal{Q}_{\vec{\theta}_t}(\vec{\theta}) \tag{9.39}$$

Because the $\mathcal{Q}$ function 9.38 is concave in all its parameters, the first order optimality condition is necessary and sufficient.

First, we maximize over π_k. Because of the constraint 9.27, we have to use the method of lagrangian multipliers. We construct the Lagrangian

$$\mathcal{L}(\pi_k, \lambda) = \sum_i \sum_k p_{i,k} \log(\pi_k) - \lambda\left(\sum_j \pi_j - 1\right) \tag{9.40}$$

where λ is the lagrangian multiplier for the constraint 9.27.
The derivative of the Lagrangian 9.40 with respect to π_k is given by

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = \sum_i \frac{p_{k|i}}{\pi_k} - \lambda \tag{9.41}$$

Setting the derivative 9.41 equal to zero yields

$$\begin{aligned}
&\frac{\partial \mathcal{L}}{\partial \pi_k} = \sum_i \frac{p_{i,k}}{\pi_k} - \lambda = 0 \\
&\Leftrightarrow \sum_i \frac{p_{k|i}}{\pi_k} = \lambda \\
&\Leftrightarrow \sum_i p_{k|i} = \lambda \pi_k \\
&\Leftrightarrow \sum_i \sum_k p_{k|i} = \lambda \sum_k \pi_k \\
&\Leftrightarrow \sum_i 1 = \lambda \\
&\Leftrightarrow n = \lambda
\end{aligned} \tag{9.42}$$

Replacing the lagrangian multiplier λ in the derivative 9.41 yields

$$\begin{aligned}
&\sum_i p_{i,k}\pi_k - n = 0 \\
&\Leftrightarrow \pi_k = \frac{\sum_i p_{k|i}}{n}
\end{aligned} \tag{9.43}$$

Note that this is exactly the formula for updating π_k in the M-step of algorithm 13.

Next, we we maximize over μ_k. Computing the gradient of the $\mathcal{Q}$ function 9.38 with respect to μ_k yields

$$\begin{aligned}
\nabla_{\vec{\mu}_k} \mathcal{Q}_{\vec{\theta}_t}(\vec{\theta}) &= \nabla_{\vec{\mu}_k} \left(\sum_i -p_{k|i} \frac{1}{2} \log\left((2\pi)^d \det(\mathbf{\Sigma})\right) - p_{k|i} \frac{1}{2} (\vec{x}_i - \vec{\mu}_k) \mathbf{\Sigma}_k^{-1} (\vec{x}_i - \vec{\mu}_k) \right) \\
&= \nabla_{\vec{\mu}_k} \sum_i -p_{k|i} \frac{1}{2} (\vec{x}_i - \vec{\mu}_k) \mathbf{\Sigma}_k^{-1} (\vec{x}_i - \vec{\mu}_k) \\
&= \sum_i p_{k|i} \mathbf{\Sigma}_k^{-1} (\vec{x}_i - \vec{\mu}_k)
\end{aligned} \tag{9.44}$$

Setting the gradient 9.44 equal to zero yields

$$\begin{aligned}
&\nabla_{\vec{\mu}_k} \mathcal{Q}_{\vec{\theta}_t}(\vec{\theta}) = \sum_i p_{k|i} \boldsymbol{\Sigma}_k^{-1}(\vec{x}_i - \vec{\mu}_k) = \vec{0} \\
&\Leftrightarrow \sum_i p_{k|i} \boldsymbol{\Sigma}_k^{-1} \vec{x}_i - \sum_i p_{k|i} \boldsymbol{\Sigma}_k^{-1} \vec{\mu}_k = \vec{0} \\
&\Leftrightarrow \boldsymbol{\Sigma}_k^{-1} \sum_i p_{k|i} \vec{x}_i = \boldsymbol{\Sigma}_k^{-1} \sum_i p_{k|i} \vec{\mu}_k \\
&\Leftrightarrow \vec{\mu}_k = \frac{\sum_i p_{k|i}\, \vec{x}_i}{\sum_i p_{k|i}}
\end{aligned} \tag{9.45}$$

Note that this exactly the formula for updating μ_k in the M-step of algorithm 13.

Finally, we maximize over $\boldsymbol{\Sigma}_k^{-1}$. Computing the derivative of the $\mathcal{Q}$ function 9.38 with respect to $\boldsymbol{\Sigma}_k^{-1}$ yields

$$\begin{aligned}
\nabla_{\boldsymbol{\Sigma}_k^{-1}} \mathcal{Q}_{\vec{\theta}_t}(\vec{\theta}) &= \nabla_{\boldsymbol{\Sigma}_k^{-1}} \left(\sum_i -p_{k|i} \frac{1}{2} \log\Big((2\pi)^d \det(\boldsymbol{\Sigma}_k)\Big) - p_{k|i} \frac{1}{2} (\vec{x}_i - \vec{\mu}_k) \boldsymbol{\Sigma}_k^{-1} (\vec{x}_i - \vec{\mu}_k) \right) \\
&= \nabla_{\boldsymbol{\Sigma}_k^{-1}} \left(\sum_i -p_{k|i} \frac{1}{2} \log\Big(\frac{(2\pi)^d}{\det(\boldsymbol{\Sigma}_k^{-1})}\Big) - p_{k|i} \frac{1}{2} (\vec{x}_i - \vec{\mu}_k) \boldsymbol{\Sigma}_k^{-1} (\vec{x}_i - \vec{\mu}_k) \right) \\
&= \nabla_{\boldsymbol{\Sigma}_k^{-1}} \left(\sum_i p_{k|i} \frac{1}{2} \log\Big(\det(\boldsymbol{\Sigma}_k^{-1})\Big) - p_{k|i} \frac{d}{2} \log(2\pi) - \right. \\
&\qquad \left. p_{k|i} \frac{1}{2} (\vec{x}_i - \vec{\mu}_k) \boldsymbol{\Sigma}_k^{-1} (\vec{x}_i - \vec{\mu}_k) \right) \\
&= \sum_i p_{k|i} \frac{1}{2} \nabla_{\boldsymbol{\Sigma}_k^{-1}} \log\Big(\det(\boldsymbol{\Sigma}_k^{-1})\Big) - p_{k|i} \frac{1}{2} \nabla_{\boldsymbol{\Sigma}_k^{-1}} (\vec{x}_i - \vec{\mu}_k) \boldsymbol{\Sigma}_k^{-1} (\vec{x}_i - \vec{\mu}_k) \\
&= \frac{1}{2} \boldsymbol{\Sigma}_k \sum_i p_{k|i} - \frac{1}{2} \sum_i p_{k|i} (\vec{x}_i - \vec{\mu}_k)(\vec{x}_i - \vec{\mu}_k)^\top
\end{aligned} \tag{9.46}$$

where we made use of the facts

$$\nabla_{\boldsymbol{\Sigma}^{-1}} \log\big(\det(\boldsymbol{\Sigma}^{-1})\big) = \boldsymbol{\Sigma} \tag{9.47}$$

$$\det(\mathbf{A})^{-1} = \det(\mathbf{A}^{-1}) \tag{9.48}$$

$$\nabla_{\mathbf{X}} \vec{a}^\top \mathbf{X} \vec{b} = \vec{a}\vec{b}^\top \tag{9.49}$$

Setting the derivative 9.46 equal to zero yields

$$\begin{aligned}
&\nabla_{\mathbf{\Sigma}_k^{-1}} \mathcal{Q}_{\vec{\theta}_t}(\vec{\theta}) = \frac{1}{2}\mathbf{\Sigma}_k \sum_i p_{k|i} - \frac{1}{2}\sum_i p_{k|i}(\vec{x}_i - \vec{\mu}_k)(\vec{x}_i - \vec{\mu}_k)^\top = \mathbf{0} \\
&\Leftrightarrow \mathbf{\Sigma}_k \sum_i p_{k|i} = \sum_i p_{k|i}(\vec{x}_i - \vec{\mu}_k)(\vec{x}_i - \vec{\mu}_k)^\top \\
&\Leftrightarrow \mathbf{\Sigma}_k = \frac{\sum_i p_{k|i}(\vec{x}_i - \vec{\mu}_k)(\vec{x}_i - \vec{\mu}_k)^\top}{\sum_i p_{k|i}}
\end{aligned} \tag{9.50}$$

Note that this exactly the formula for updating $\mathbf{\Sigma}_k$ in the M-step of algorithm 13.

9.6. Outlook

In this chapter we learned about clustering. Clustering aims to find groups of "similar" items in a set of items, whereas different defintions of the term "similar" lead to different clusterings.
We discussed various methods for computing a clustering of a given data set. We talked about vector quantization methods like k-means, agglomerative clustering as an instance of hierarchical clustering and special methods like spectral clustering. Finally, we discussed DBSCAN and Gaussian mixture models as examples of density based clusterings. In the setting of Gaussian mixture models, we also discussed the EM-algorithm as a general technique for optinizing latent variable models.

9.7. Exercises

1. Prove that the *center of gravity*, given by

$$\vec{c}_k = \frac{1}{n} \sum_i \vec{x}_i \tag{9.51}$$

is the optimal solution to the k-means clustering problem 9.1 if $k = 1$.

Appendices

Appendix A

Convex optimization

In this chapter we take a look at a special class of optimization problems and how to solve them.
We do not discuss *non-convex optimization problems* because we will not encounter any of those in these notes. However, note that non-convex optimization problems may occur in different settings (e.g. in *Deep Learning* you are almost always confronted with non-convex optimization problems). Although non-convex problems are usually much harder than convex problems, practitioners often apply tools from convex optimization (e.g. an iterative solver like gradient descent A.3.2) to get an approximate solution (often this works surprisingly well).

A.1. Convex set

A *convex set* is a subset of an *affine space*[1] that is closed under *convex combination*. The subset from the euclidean space $\mathcal{C} \subseteq \mathbb{R}^d$ is called a *convex region*, if it is closed under convex combination.
Closedness under convex combination can be expressed as follows

$$\alpha\vec{x} + (1-\alpha)\vec{y} \in \mathcal{C} \quad \forall\, \vec{x}, \vec{y} \in \mathcal{C} \;\; \alpha \in [0,1] \tag{A.1}$$

Obviously, $\mathcal{C} = \mathbb{R}^d$ is a convex set/region.

In plain English: A set is called a convex set, if and only if the straight line connecting any pair of points lies inside the set.
See Fig. A.1 for an example of a convex and a non-convex set.

[1] Note that the Euclidean space $\mathbb{R}^d$ is an affine space.

Figure A.1: Convex vs. non-convex set

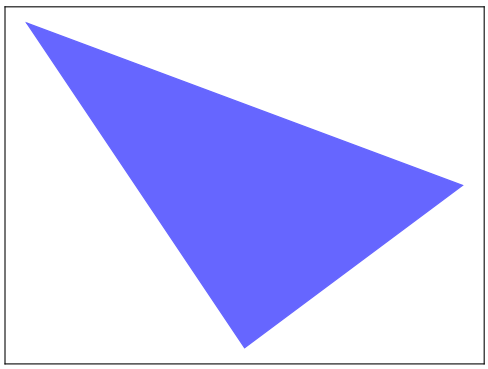


Figure A.2: Convex set

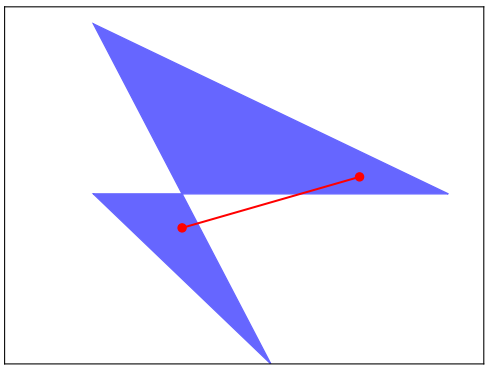


Figure A.3: Non-convex set

A.2. Convex functions

A function $f : \mathbb{R}^d \mapsto \mathbb{R}$ is said to be a *convex function* if and only if

$$f(\alpha\vec{x} + (1-\alpha)\vec{y}) \leq \alpha f(\vec{x}) + (1-\alpha)f(\vec{y}) \quad \forall\, \vec{x}, \vec{y} \in \mathbb{R}^d \;\; \alpha \in [0,1] \qquad \text{(A.2)}$$

where we assume that the domain of f is $\mathbb{R}^d$. More general one would write $\vec{x}, \vec{y} \in \mathcal{D}(f)$ to make sure that the values are always in the domain of f.

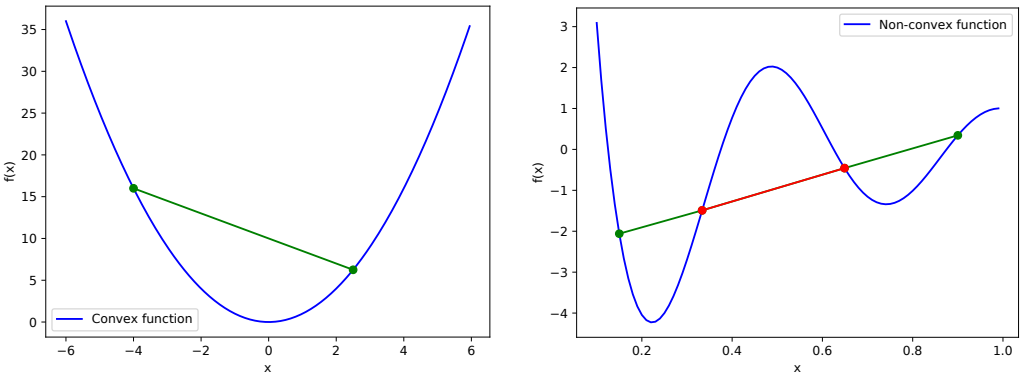
In plain English: A function is convex, if and only if the line, connecting any pair of points on the function graph, is always "above" or greater than the function values in between those two points. See Fig. A.4 for an example of a convex and a non-convex function.
A convex function f is called *strictly convex*, if the less or equal in A.2 is strictly less $\forall \vec{x} \neq \vec{y}$. Hence

$$f(\alpha\vec{x}+(1-\alpha)\vec{y}) < \alpha f(\vec{x})+(1-\alpha)f(\vec{y}) \quad \forall\, \vec{x}, \vec{y} \in \mathbb{R}^d,\; \vec{x} \neq \vec{y} \;\; \alpha \in [0,1] \;\; \text{(A.3)}$$

A function f is called *concave* if $-f$ is convex (see Fig. A.5 for an example). Simultaneously, a function f is called *strictly concave* if $-f$ is strictly convex.

To prove that a function is convex we have different options:

Figure A.4: Convex vs. non-convex function



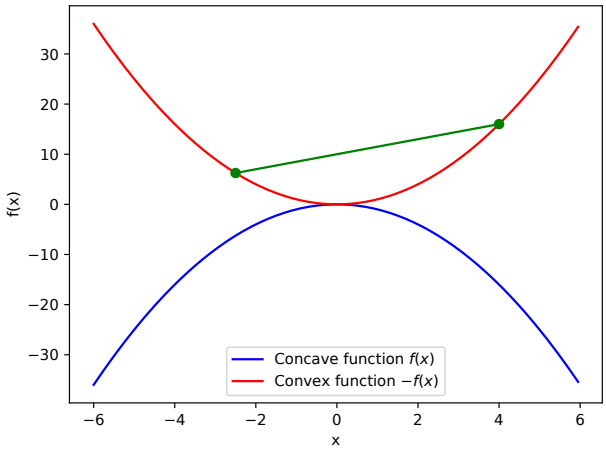


Figure A.5: A concave function

1. Prove that A.2 holds.
   This can be difficult. If the function is differentiable, we might want to try one of the next options first.

2. Show that the *first order condition* holds.
   The condition is given by

$$f(\vec{y}) \geq \nabla_{\vec{x}} f(\vec{x})^\top (\vec{y} - \vec{x}) \quad \forall\, \vec{x}, \vec{y} \in \mathcal{D}(f) \tag{A.4}$$

   where we assume the the function f is at least once differentiable.

   In plain English: Show that the function curve is always above the tangent at any point (see Fig. A.6).

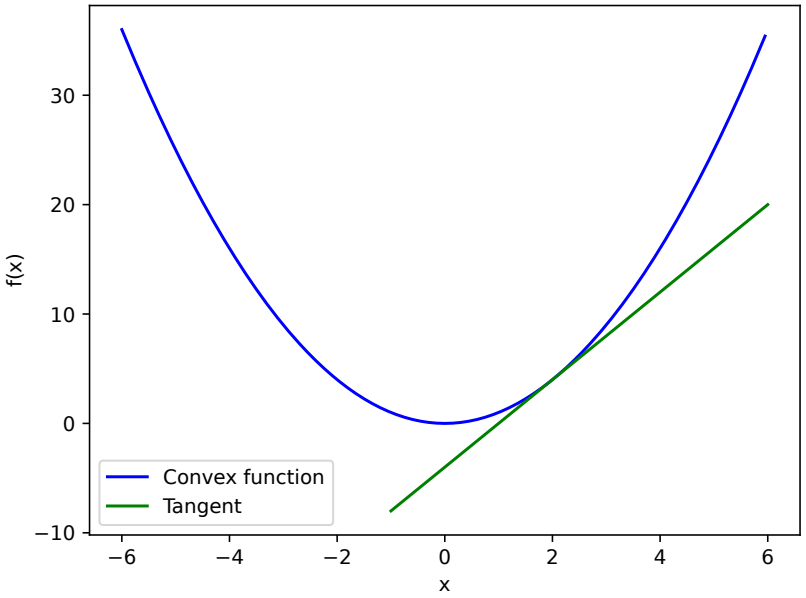


Figure A.6: Convex function: First order condition

3. Show that the *second order condition* holds.
   The condition is given by

$$\nabla^2_{\vec{x}} \boldsymbol{f}(\vec{x}) \succeq 0 \quad \forall\, \vec{x} \in \mathcal{D}(f) \tag{A.5}$$

   where we assume that the function f is at least twice differentiable.

   In plain English: Show that the second derivative is always non-negative. In the multivariate case, the second derivative - the Hessian - is required to be positive semidefinite.

Note that each of the three conditions alone is *necessary and sufficient*.

An alternative to the previous options is to decompose a given function into known convex functions and combine them by using convexity preserving operations (see section A.2.2).

The nice thing about convex functions is that *a local optimum is a global optimum, too*. This means that, once we found a local optimum, we can be sure that this one is the global optimum. Hence, when optimizing (minimizing) a convex function we do not need to worry about (sub optimal) local minima and can use any method which results in a local optimum. Furthermore, for strictly convex functions the optimum is unique.

A.2.1. Local vs. global optimum

We call a point $\vec{x}_0$ a *local optimum/stationary point* of a (differentiable) function f, if the gradient of f at $\vec{x}_0$ is equal to zero.

$$\nabla_{\vec{x}} f(\vec{x}_0) = \vec{0} \tag{A.6}$$

In order to distinguish between maxima and minima, we have to check the Hessian (second derivative) at $\vec{x}_0$, too.
As we can see from the right figure in Fig. A.4, if the Hessian of a local optimum $\vec{x}_0$ is *positiv definite*, we know that $\vec{x}_0$ is a local minimum.
Similar, if the Hessian of a local optimum $\vec{x}_0$ is *negativ definite*, we know that $\vec{x}_0$ is a local maximum.
If the Hessian at $\vec{x}_0$ is *indefinite*, then $\vec{x}_0$ is neither a minimum nor a maximum but a saddle point.

We call a point $\vec{x}_0$ a *global minimum* of a function f, if there is not other point $\vec{x} \neq \vec{x}_0$ whose function value is *less than* the function value of $\vec{x}_0$.
Similar, we call a point $\vec{x}_0$ a *global maximum* of a function f, if there is not other point $\vec{x} \neq \vec{x}_0$ whose function value is *greater than* the function value of $\vec{x}_0$.
Obviously, the gradient of the function f, for each local and global optimium $\vec{x}_0$, must be equal to zero. However, as we can see in the right figure of Fig. A.4, a zero gradient itself is not sufficient for global optimality. There might be other points with a zero gradient whose function value are either smaller or greater. Furthermore, the limit of the border might go to plus or minus infinity.
Because of this, in general it is very difficult to find a global optimum of a function. Lucklily, finding global optima is simple if we are working with convex/concave functions.
Because of the second order condition A.5[2] of convex functions, we know that *a local minimum of a convex function is guaranteed to be global minimum* too. Therefore, the first order condition of optimality A.6 is a necessary and sufficient optimiality criterion for all convex/concave functions.

Note: Instead of the gradient, we can use its generalization the subgradient (see section A.2.4).

[2]because a convex/concave function does not have any saddle points, it is sufficient to have semidefiniteness instead of definiteness

A.2.2. Convexity preserving operations

Let f, k and g be convex functions over the domain $\mathbb{R}^d$ (other domains are possible, for simplicity we stick to $\mathbb{R}^d$) and $\lambda \in \mathbb{R}_+$. Furthermore, we assume that k is non-decreasing.
Then, the following functions are convex[3]

1. $z(\vec{x}) = f(\vec{x}) + g(\vec{x})$
2. $z(\vec{x}) = \lambda f(\vec{x})$
3. $z(\vec{x}) = k(f(\vec{x}))$

A.2.3. Examples

Some examples[4] of convex functions:

1. $f(x) = x^{2a}$ for $a \in \mathbb{N}$
2. $f(\vec{x}) = \vec{a}^\top \vec{x} + b$ for $\vec{x}, \vec{a} \in \mathbb{R}^d$ and $b \in \mathbb{R}$
3. $f(x) = \exp(ax)$ for $a, x \in \mathbb{R}$
4. $f(x) = -\ln(x)$ for $x > 0$
5. $f(x) = -x\ln(x)$ for $x > 0$
6. $f(x) = |x|^a$ for $a \geq 1$
7. $f(x) = x^a$ for $x > 0$ and $a \geq 1 \vee a \leq 0$
8. $f(\vec{x}) = \vec{x}^\top \mathbf{A}\vec{x} + \vec{c}^\top \vec{x}$ for $\mathbf{A} \succeq 0$ and $\vec{x}, \vec{c} \in \mathbb{R}^d$

A.2.4. Subdifferential

The *subdifferential*[5] (also called *subderivative* or *subgradient*) is a generalization of the derivative of a convex function.
We define the subdifferential at $\vec{x}_0$ of a convex function f as the set[6]

$$\partial f(\vec{x}_0) = \{\vec{s} \mid f(\vec{x}) \geq f(\vec{x}_0) + \vec{s}^\top(\vec{x}_0 - \vec{x}) \;\; \forall \vec{x} \in \mathbb{R}^d\} \tag{A.7}$$

[3] proofs are omitted because they can be found in every textbook on convex analysis.
[4] a good exercise is to prove convexity of these functions
[5] see `https://en.wikipedia.org/wiki/Subderivative`
[6] Sometimes $\partial f(\vec{x}_0)$ is rewritten as $\frac{\partial}{\partial \vec{x}} f(\vec{x}_0)$, to stress that we take the derivative with respect to x.

where we call each $\vec{s} \in \partial f(\vec{x}_0)$ subgradient of f at $\vec{x}_0$.

We say that a convex function f is *differentiable* at $\vec{x}_0$, if $|\partial f(\vec{x}_0)| = 1$. In this case the only element in $\partial f(\vec{x}_0)$ is the "standard" gradient $\nabla_{\vec{x}} f(\vec{x}_0)$. If $|\partial f(\vec{x}_0)| > 1$, we say that the convex function f is only *subdifferentiable* at $\vec{x}_0$.

Note: Every convex function is at least subdifferentiable for all $\vec{x}_0$.

A.2.4.1 Example

We consider the absolute value function, which we know from A.2.3 is a convex function.

$$f(x) = |x| \tag{A.8}$$

The subdifferential of A.8 is given by

$$\partial f(x) = \begin{cases} \{1\} & \text{if } x > 0 \\ \{-1\} & \text{if } x < 0 \\ [-1, 1] & \text{if } x = 0 \end{cases} \tag{A.9}$$

The subdifferential A.9 is visualized in Fig. A.7

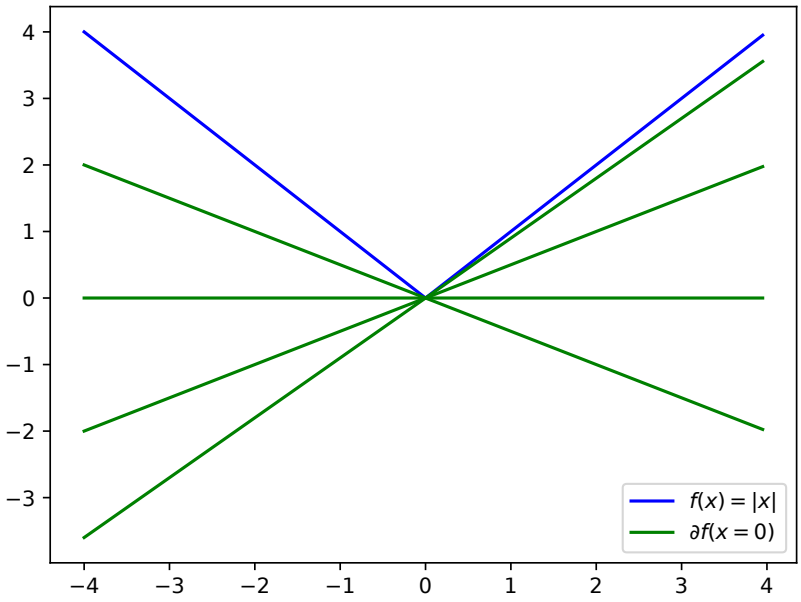


Figure A.7: Subgradients of $f(x) = |x|$ at $x = 0$

A.3. Convex optimization

Let f be a convex function (not necessarily differentiable), then the following is called an *convex optimization problem*

$$\begin{aligned} &\min_{\vec{x} \in \mathcal{X}} f(\vec{x}) \\ &\text{where } \mathcal{X} \subseteq \mathbb{R}^d \text{ is a } \textit{convex set} \text{ of feasible solutions} \end{aligned} \tag{A.10}$$

We call A.10 an *unconstrained convex optimization problem* if $\mathcal{X} = \mathbb{R}^d$.

In the next sections we will discuss how to solve problems like A.10.

A.3.1. Closed form solution

If we consider $\mathcal{X} = \mathbb{R}^d$, we know that for any optimal point $x_* \in \mathbb{R}^d$ the gradient of f at x_* must be equal to zero (this also called *the necessary condition*). Because f is a convex function, this necessary condition is a sufficient condition, too - recall that the sufficient condition says that the second derivative of f at x_* must be positive, which is one of the criterions/definitions for a function being a convex function (see the second order condition A.5).
Therefore, the set of minimizers is given by

$$\{x_* \in \mathbb{R}^d \mid \nabla_x f(x_*) = 0\} \tag{A.11}$$

If we consider the subdifferential as a generalization of the gradient for non-differentiable convex functions, the set of minimizers is given by

$$\{x_* \in \mathbb{R}^d \mid 0 \in \partial f(x_*)\} \tag{A.12}$$

Sometimes we can solve $\nabla_x f(x_*) = 0$ (or $0 \in \partial f(x_*)$) for x_* analytically (like in the next example A.3.1.1) and sometimes we can not. If we can not obtain a closed form solution for x_*, we have to use an iterative algorithm for computing x_* (e.g. the gradient descent algorithm - see section A.3.2).

For an arbitrary convex set $\mathcal{X} \subseteq \mathbb{R}^d$, we have the following optimality conditions

$$\{x_* \in \mathcal{X} \mid \nabla_x f(x_*)(y - x_*) \geq 0 \ \ \forall\, y \in \mathcal{X}\} \tag{A.13}$$

for a differentiable convex function f, and

$$\Big\{x_* \in \mathbb{R}^d \mid 0 \in \partial\big(f(x_*) + \mathbb{1}(x_* \in \mathcal{X})\big)\Big\} \tag{A.14}$$

for a subdifferentiable convex function f, where the subdifferential of $\mathbb{1}(x_* \in \mathcal{X})$ is the *normal cone* of $\mathcal{X}$ at x_* (see [3] for details).
Alternatively, we can write A.14 as

$$\Big\{ x_* \in \mathcal{X} \mid 0 \in \partial\big(f(x_*)\big) \Big\} \tag{A.15}$$

A.3.1.1 Example

Suppose that we want to solve the following optimization problem

$$x = \underset{x \in [0,1]}{\arg\max}\; x^6(1-x)^4 \tag{A.16}$$

First, we note that A.16 is a non-convex optimization problem. The domain $[0, 1]$ is a convex set, but the function $x^6(1-x)^4$ is not convex over the set $[0, 1]$.
However, we can transform A.16 such that it becomes a convex optimization problem while maintaining the location of the optimum. We can do so by applying the logarithm to the function and multiplying the result by -1. Hence, A.16 can be rewritten as

$$x = \underset{x \in [0,1]}{\arg\min}\; -\log\big(x^6(1-x)^4\big) \tag{A.17}$$

Because the log of 0 is undefined, we can no longer have $x = 0$ or $x = 1$. That is, we have reduced the set of possible solutions from $[0, 1]$ to $(0, 1)$. However, by looking at the left figure in Fig. A.3.1.1 we argue that neither $x = 0$ nor $x = 1$ leads to maximum. In practice, we often know/assume that the boundary/extreme values of the domain are no solution.
Note that A.17 is a convex optimization problem and is equivalent to A.16 (see exercise 7). The difference between A.16 and A.17 is illustrated in Fig. A.3.1.1.
Note: This "trick" of transforming a non-convex function into a convex function, by applying the logarithm to the function and then multiplying by -1, is sometimes called the *log-trick* and we will often make use of it in these notes.

Because A.17 is equivalent to A.16 and because it is a convex optimization problem, we solve A.17 instead of A.16.
First, we compute the derivative with respect to x.

$$\begin{aligned} \frac{\partial}{\partial x}\big(-\log\big(x^6(1-x)^4\big)\big) &= \frac{\partial}{\partial x}\big(-6\log(x) - 4\log(1-x)\big) \\ &= -\frac{6}{x} + \frac{4}{1-x} \end{aligned} \tag{A.18}$$

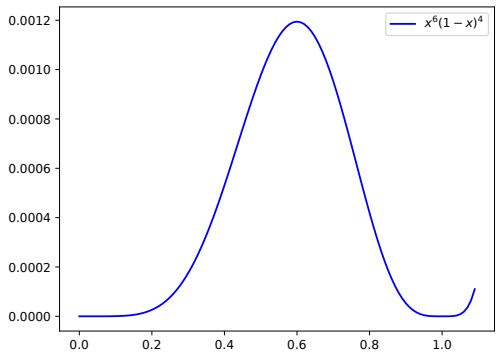


Figure A.8: Original function

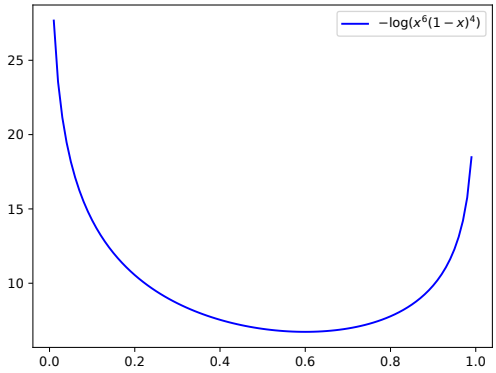


Figure A.9: Transformed function

Next, we set the derivative equal to zero and solve for x.

$$\begin{aligned} -\frac{6}{x} + \frac{4}{1-x} &= 0 \\ \frac{6}{x} &= \frac{4}{1-x} \\ \frac{x}{6} &= \frac{1-x}{4} \\ \frac{4}{6}x &= 1-x \\ \frac{10}{6}x &= 1 \\ x &= \frac{6}{10} = 0.6 \end{aligned} \tag{A.19}$$

Recall that we required $x \in [0, 1]$. Because $0.6 \in [0, 1]$ we found the optimum in our set of feasible solutions. Note that this might not always work. In this case we are lucky because the global optimum of f lies inside our set of feasible solutions $\mathcal{X}$. In general, this might not be case and we would have to use different techniques for restricting x to the set $\mathcal{X}$ - e.g. we could describe the constraint $x \in \mathcal{X}$ by a set of functions and then use Lagrange multipliers (see section. A.6).

A.3.2. Gradient descent

A very simple and famous *iterative algorithm* for solving an unconstrained convex optimization problem (that is minimizing a convex function) is the *gradient descent* algorithm.

The algorithm is described in Algorithm 15

Algorithm 15 Gradient descent

1: $\vec{x} = \vec{0}$ ▷ Initialize $\vec{x}$
2: **repeat**
3: $\quad \vec{x} = \vec{x} - \eta \nabla_{\vec{x}} f(\vec{x})$ ▷ Update $\vec{x}$
4: **until** convergence

where η is called *step size* - in machine learning also called *learning rate*. It can be shown (e.g. see [3]), that by choosing the "correct" value for η the gradient descent algorithm 15 converges to the optimal solution.
Finding the optimal value of η is another optimization problem (also called *line search*)

$$\eta = \arg\min_{\eta} f(\vec{x} - \eta \nabla_{\vec{x}} f(\vec{x})) \tag{A.20}$$

Sometimes A.20 can be solved analytically, but sometimes it is not solvable efficiently. In such cases, we can perform iterative optimization techniques to find the best η, i.e. trying a few values and taking the best one.
When trying different values for η is too inefficient - evaluation of the objective function might be very time consuming like in deep learning -, we simply select a fixed value for η, typically $\eta < 1$. Sometimes additional heuristics, like decreasing η over time, are used to get closer to the optimum. Although that might look very "imprecise", these heuristics often work surprisingly well in practice.
In plain English: If you can do a line search (either solve A.20 exactly or find a "good approximation"), do it. If you can not, select a $\eta < 1$[7] and adjust it over time (e.g. if your objective function stops decreasing).
Techniques for finding the optimal step size are extensively discussed in section A.3.6.

The second ingredient of the gradient descent algorithm 15 is the *convergence criterion*. The convergence criterion is used to determine whether we should stop the algorithm or not. We know that $\nabla_{\vec{x}} f(\vec{x}) = 0$ holds if we found the $\vec{x}$ which minimizes $f(\vec{x})$. Thus a suitable stopping criterion might be $\|\nabla_{\vec{x}} f(\vec{x})\| < \epsilon$ (note that $= 0$ might never be achieved due to numerical issues like fixed size representation of numbers).
An additional stopping criterion is to stop when a maximum number of iterations is reached.

[7] make sure that your objective is scaled/preprocessed properly

Note: In this description of the gradient descent algorithm, we assumed that f is differentiable. We can easily extend the gradient descent algorithm to subdifferentiable functions by replacing all occurrences of the gradient $\nabla_{\vec{x}} f(\vec{x})$ by any element $\vec{d} \in \partial f(\vec{x})$. In this case the gradient descent algorithm is also called *subgradient algorithm.*

Remark: There are more sophisticated solvers[8] like Newton methods or quasi-newton methods. These solvers are more "complicated" but also more robust in the sense that they converge faster no matter what the objective function looks like, which is not true for the gradient descent algorithm.
Some of these "more sophisticated methods" are discussed in section A.3.4.
In general, you can find more information about these and other methods as well as mathematical analyses in special literature on optimization like [3],[9].

Beside the discussion of iterative solvers, one should mention that sometimes an analytical solution exists (see section A.3.1). If using/computing the analytical solution is possible (e.g. we have enough time and resources to compute it, it is numerically stable, ...), there is no excuse for not doing so.

A.3.2.1 Example

We want to solve the following convex optimization problem by using the gradient descent algorithm 15.

$$\min_{\vec{w} \in \mathbb{R}^2} \frac{1}{2} \|\mathbf{X}\vec{w} - \vec{y}\|_2^2 \quad \text{where } \mathbf{X} \in \mathbb{R}^{100 \times 2},\ \vec{y} \in \mathbb{R}^{100} \tag{A.21}$$

The gradient of A.21 with respect to $\vec{w}$ is given by

$$\nabla_{\vec{w}} = \mathbf{X}^\top \mathbf{X}\vec{w} - \mathbf{X}\vec{y} \tag{A.22}$$

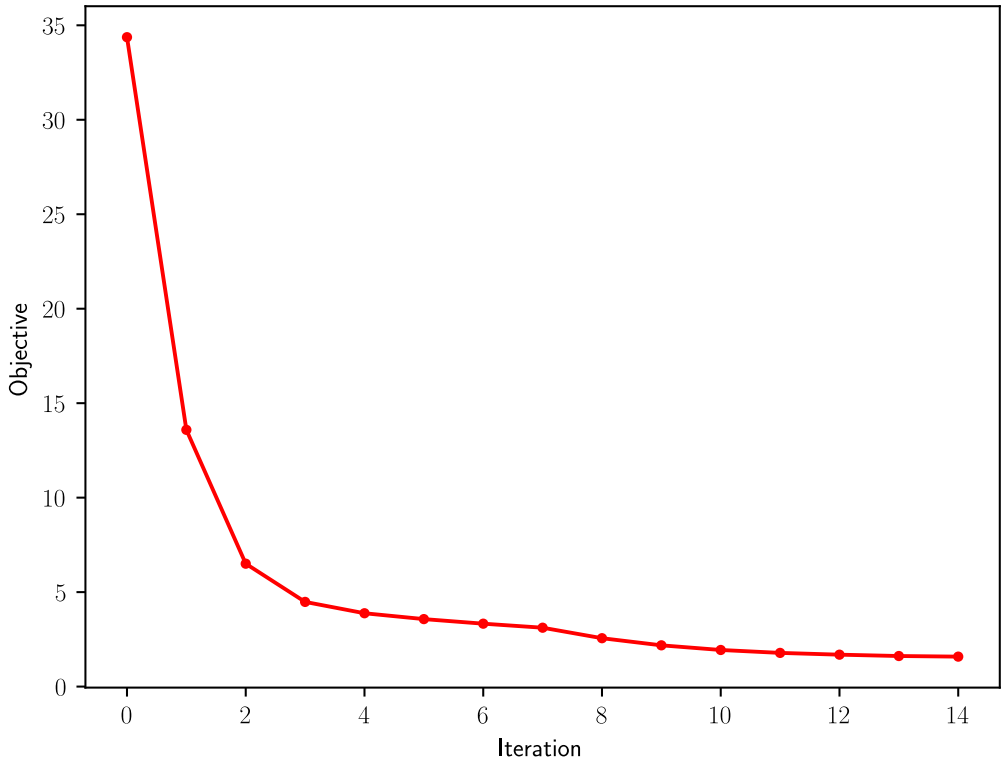
We start at $\vec{w} = (5, 2)^\top$ with the step size $\eta = 0.01$ and compute 15 iterations of the gradient descent algorithm. After 5 iterations we set $\eta = 0.03$ and after 10 iterations we set $\eta = 0.04$.
In Fig. A.10 we can see how the gradient descent algorithm behaves.
In the first plot, we can see how the value of the objective function is changing over time. As we can see, the value decreases over time which means that we get closer to the optimum (minimum of the objective). We observe that the value descreases very rapidly in the beginning whereas the changes

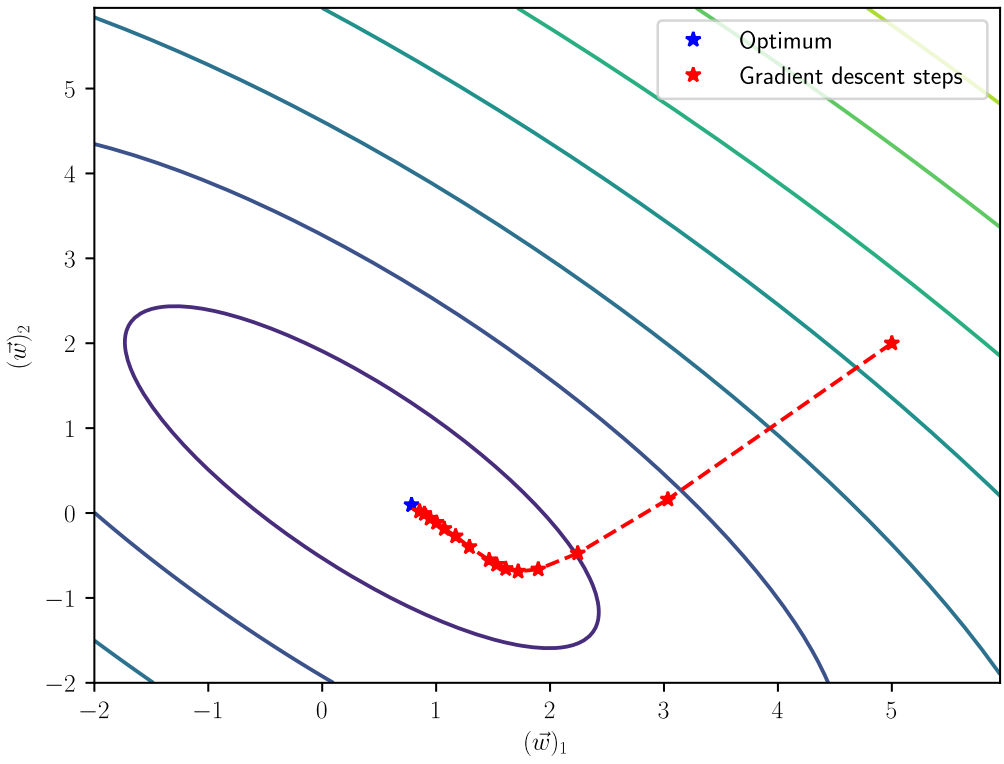
[8] The gradient descent algorithm is the simplest thing you can do, but it is not always the best thing you could do!

become smaller later in time. An explanation of this behaviour can be found in the second plot (see below for a discussion).
The second plot shows the contours of the objective function as well as the intermediate estimates of $\vec{w}$. We can see that the the estimates of $\vec{w}$ get closer and closer to the optimum. In the beginning we make "huge" steps, whereas the steps become smaller near the optimum. This is because the gradient becomes smaller (closer to zero) when we are close to the optimum - this is why we should use an adaptive step size η.

Remark: This is a toy example only. In practice we should use a line search for computing the optimal step size η as well as a preprocessing/reformulation of A.21. Or we could use a completly different (better suited) algorithm for solving A.21.

Figure A.10: Gradient descent - Example





A.3.3. Intuition behind gradient descent

We assume that we want to minimize a given function $f : \mathbb{R}^d \mapsto \mathbb{R}$.

$$\vec{x} = \arg\min_{\vec{x} \in \mathbb{R}^d} f(\vec{x}) \tag{A.23}$$

Furthermore, we assume that we do not know how to solve A.23 because the shape of f might be too complicated or some other reasons prevent us from working with f directly.
Instead of minimizing f, we could try to minimize a function which is similar to f but having a much nicer/easier shape.
From Taylor's theorem we know that we can approximate our function f around a given point $\vec{x}_t$ arbitrarily well (if f is infinitely often differentiable). The simplest approximation of a function around a point is the first order Taylor approximation.
The *first order Taylor approximation* of our function f around $\vec{x}_t$ is given by

$$f(\vec{x}) \approx f(\vec{x}_t) + \nabla_{\vec{x}} f(\vec{x}_t)^\top (\vec{x} - \vec{x}_t) \tag{A.24}$$

Next, we fix a $\vec{x}_t \in \mathbb{R}^d$ and replace $f(\vec{x})$ in A.23 by the first order Taylor approximation around $\vec{x}_t$. By adding the additional term $\frac{1}{2\eta}\|\vec{x} - \vec{x}_t\|_2^2$, we obtain the following optimization problem

$$\vec{x}_{t+1} = \arg\min_{\vec{x} \in \mathbb{R}^d} f(\vec{x}_t) + \nabla_{\vec{x}} f(\vec{x}_t)^\top (\vec{x} - \vec{x}_t) + \frac{1}{2\eta}\|\vec{x} - \vec{x}_t\|_2^2 \tag{A.25}$$

In plain English: Find a $\vec{x} \in \mathbb{R}^d$ which minimizes the first order Taylor approximation of f around $\vec{x}_t$, subject to the constraint that the distance between $\vec{x}_t$ and $\vec{x}$ is not "too large" (modeled by the additive term $\frac{1}{2\eta}\|\vec{x} - \vec{x}_t\|_2^2$). The trade-off between minimization of f and minimization of the distance between $\vec{x}_t$ and $\vec{x}$ is adjusted by the parameter η.
Note that we need this distance constraint, because otherwise we would minimize a line/plane only. The minimum of a line/plane lies somewhere in infinity, which is not what we want. Furthermore, we could motivate the distance constraint by observing that the first order Taylor approximation around a point $\vec{x}_t$ becomes more and more imprecise the farther we are away from $\vec{x}_t$. Therefore we should not go too far away from $\vec{x}_t$.

We notice that A.25 is a convex optimization problem and compute its closed form solution by computing the gradient of the objective, setting it equal to zero and finally solve for $\vec{x}$.

First, we compute the gradient.

$$\begin{aligned}
&\nabla_{\vec{x}} \left(f(\vec{x}_t) + \nabla_{\vec{x}} f(\vec{x}_t)^\top (\vec{x} - \vec{x}_t) + \frac{1}{2\eta} \|\vec{x} - \vec{x}_t\|_2^2 \right) \\
&= \nabla_{\vec{x}} f(\vec{x}_t) + \nabla_{\vec{x}} \left(\nabla_{\vec{x}} f(\vec{x}_t)^\top \vec{x} \right) - \nabla_{\vec{x}} \left(\nabla_{\vec{x}} f(\vec{x}_t)^\top \vec{x}_t \right) + \nabla_{\vec{x}} \frac{1}{2\eta} (\vec{x} - \vec{x}_t)^\top (\vec{x} - \vec{x}_t) \\
&= \nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{\eta} (\vec{x} - \vec{x}_t)
\end{aligned} \tag{A.26}$$

Next, setting the gradient A.26 equal to zero yields

$$\begin{aligned}
&\nabla_{\vec{x}} f(\vec{x}_t) + \frac{1}{\eta} (\vec{x} - \vec{x}_t) = \vec{0} \\
&\Leftrightarrow \\
&\vec{x} = \vec{x}_t - \eta \nabla_{\vec{x}} f(\vec{x}_t)
\end{aligned} \tag{A.27}$$

Thus, the solution to A.25 is given by

$$\vec{x}_{t+1} = \vec{x}_t - \eta \nabla_{\vec{x}} f(\vec{x}_t) \tag{A.28}$$

By comparing A.28 and the gradient descent algorithm 15, we observe that A.28 is equivalent to one update step in Algorithm 15. This observation enables us to understand the idea/principle behind the gradient descent algorithm.

In plain English: The gradient descent algorithm starts at some "arbitrary" point, computes the first order Taylor approximation around this point and solves A.25. The solution of A.25 becomes the new starting point for the next iteration, where we again compute the first order Taylor approximation around the new point (solution of the previous iteration) and solve A.25. This procedure is repeated until we converged to the minimum or some other convergence criterion is fulfilled.
The step size η acts as a regularization of the solution, in the sense that it penalizes large deviations from the previous point. This is beneficial, because the first order Taylor approximation around a particular point is only a good approximation near this point.

The idea of solving a local approximation of a function and restricting the solution to this local area, is exactly the idea behind *trust region methods*[9].

[9] see [9] for more information

A.3.4. Newton's method

The *second order Taylor approximation* of f around $\vec{x}_t$ is given by

$$f(\vec{x}) \approx f(\vec{x}_t) + \nabla_{\vec{x}} f(\vec{x}_t)^\top (\vec{x} - \vec{x}_t) + \frac{1}{2}(\vec{x} - \vec{x}_t)^\top \boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t)(\vec{x} - \vec{x}_t) \quad \text{(A.29)}$$

Analogous to the previous section, we can replace $f(\vec{x})$ in A.23 by its second order Taylor approximation A.29 and obtain the following optimization problem

$$\vec{x}_{t+1} = \underset{\vec{x} \in \mathbb{R}^d}{\arg\min}\ f(\vec{x}_t) + \nabla_{\vec{x}} f(\vec{x}_t)^\top (\vec{x} - \vec{x}_t) + \frac{1}{2}(\vec{x} - \vec{x}_t)^\top \boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t)(\vec{x} - \vec{x}_t) + \frac{1}{2\lambda}\|\vec{x} - \vec{x}_t\|_2^2 \quad \text{(A.30)}$$

where we added the "distance regularization" $\frac{1}{2\lambda}\|\vec{x} - \vec{x}_t\|_2^2$ like we did in the previous section.

First, we compute the gradient of A.30

$$\begin{aligned}
&\nabla_{\vec{x}} \left(f(\vec{x}_t) + \nabla_{\vec{x}} f(\vec{x}_t)^\top (\vec{x} - \vec{x}_t) + \frac{1}{2}(\vec{x} - \vec{x}_t)^\top \boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t)(\vec{x} - \vec{x}_t) + \frac{1}{2\lambda}\|\vec{x} - \vec{x}_t\|_2^2 \right) \\
&= \nabla_{\vec{x}} \Bigg(f(\vec{x}_t) + \nabla_{\vec{x}} f(\vec{x}_t)^\top \vec{x} - \nabla_{\vec{x}} f(\vec{x}_t)^\top \vec{x}_t + \frac{1}{2}\vec{x}^\top \boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t)\vec{x} - \frac{1}{2}\vec{x}^\top \boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t) - \\
&\quad \frac{1}{2}\vec{x}_t \boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t)\vec{x} + \vec{x}_t^\top \boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t)\vec{x}_t + \frac{1}{2\lambda}(\vec{x} - \vec{x}_t)^\top (\vec{x} - \vec{x}_t) \Bigg) \\
&= \nabla_{\vec{x}} f(\vec{x}_t) + \boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t)\vec{x} - \boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t)\vec{x}_t + \frac{1}{\lambda}(\vec{x} - \vec{x}_t) \\
&= \nabla_{\vec{x}} f(\vec{x}_t) + \boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t)(\vec{x} - \vec{x}_t) + \frac{1}{\lambda}(\vec{x} - \vec{x}_t) \\
&= \nabla_{\vec{x}} f(\vec{x}_t) + (\boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t) + \frac{1}{\lambda}\mathbb{I})(\vec{x} - \vec{x}_t)
\end{aligned} \quad \text{(A.31)}$$

Next, setting the gradient A.31 equal to zero yields

$$\begin{aligned}
&\nabla_{\vec{x}} f(\vec{x}_t) + (\boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t) + \frac{1}{\lambda}\mathbb{I})(\vec{x} - \vec{x}_t) = \vec{0} \\
&\Leftrightarrow \\
&\vec{x} = \vec{x}_t - (\boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t) + \frac{1}{\lambda}\mathbb{I})^{-1} \nabla_{\vec{x}} f(\vec{x}_t)
\end{aligned} \quad \text{(A.32)}$$

Thus, the solution to A.30 is given by

$$\vec{x}_{t+1} = \vec{x}_t - (\boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}} \boldsymbol{f}(\vec{x}_t) + \frac{1}{\lambda}\mathbb{I})^{-1} \nabla_{\vec{x}} f(\vec{x}_t) \quad \text{(A.33)}$$

We can derive a very similar result if we ignore the additional term $\frac{1}{2\lambda}\|\vec{x}-\vec{x}_t\|_2^2$. By doing so, we obtain the new optimization problem

$$\vec{x}_{t+1} = \underset{\vec{x}\in\mathbb{R}^d}{\arg\min}\ f(\vec{x}_t)+\nabla_{\vec{x}} f(\vec{x}_t)^\top(\vec{x}-\vec{x}_t)+\frac{1}{2}(\vec{x}-\vec{x}_t)^\top \boldsymbol{\nabla^2_{\vec{x}} f}(\vec{x}_t)(\vec{x}-\vec{x}_t) \quad \text{(A.34)}$$

Computing the gradient of A.34, and setting it to zero yields

$$\begin{aligned} &\nabla_{\vec{x}} f(\vec{x}_t) + (\boldsymbol{\nabla^2_{\vec{x}} f}(\vec{x}_t))(\vec{x}-\vec{x}_t) = \vec{0} \\ &\Leftrightarrow \\ &\vec{x} = \vec{x}_t - \boldsymbol{\nabla^2_{\vec{x}} f}(\vec{x}_t)^{-1}\nabla_{\vec{x}} f(\vec{x}_t) \end{aligned} \quad \text{(A.35)}$$

Thus, the solution to A.34 is given by

$$\vec{x}_{t+1} = \vec{x}_t - \boldsymbol{\nabla^2_{\vec{x}} f}(\vec{x}_t)^{-1}\nabla_{\vec{x}} f(\vec{x}_t) \quad \text{(A.36)}$$

A.36 is the "usual"/"common" update step in Newton's method.
The pseudocode of Newtons's method is given in algorithm 16

Algorithm 16 Newton's method

1: $\vec{x} = \vec{0}$ ▷ Initialize $\vec{x}$
2: **repeat**
3: $\quad \vec{x} = \vec{x} - \eta\boldsymbol{\nabla^2_{\vec{x}} f}(\vec{x}_t)^{-1}\nabla_{\vec{x}} f(\vec{x})$ ▷ Update $\vec{x}$
4: **until** convergence

In plain English: Newtons's method starts at some "arbitrary" point, compute the second order Taylor approximation around this point and solves A.30. The solution of A.30 becomes the new starting point for the next iteration, where we again compute the second order Taylor approximation around the new point (solution of the previous iteration) and solve A.30. This procedure is repeated until we converged to the minimum or some other convergence criterion is fulfilled.

However, there is at least one problem with Newton's method. The Hessian $\nabla^2_{\vec{x}}\mathbf{f}$ in A.36 might not always be invertible. If it happens, that the Hessian is not invertible, we can not compute the update step in algorithm 16.
If the Hessian is not invertible, we could switch from A.36 to A.33. It can be shown, that if $\frac{1}{\lambda}$ is large enough [9], the matrix $(\boldsymbol{\nabla^2_{\vec{x}} f}(\vec{x}_t) + \frac{1}{\lambda}\mathbb{I})$ is always invertible[10]. In a very naive algorithm, we could choose a "very large" $\frac{1}{\lambda}$ and

[10] if f is a convex function, every $\lambda > 0$ is sufficient to make the matrix $(\boldsymbol{\nabla^2_{\vec{x}} f}(\vec{x}_t) + \frac{1}{\lambda}\mathbb{I})$ invertible

hope that it is large enough to make the matrix invertible.
An alternative to A.33 would be to switch to methods where the Hessian is approximated by an invertible matrix. Quasi-Newton methods, as discussed in the next section, are such kind of methods.

A.3.5. Quasi-Newton methods

Newton's method converges much faster than gradient descent. However, it suffers form two major disadvantages. First, the Hessian $\boldsymbol{\nabla^2_{\vec{x}} f}$ might not be invertible, second the Hessian is a square matrix where the dimensions is equal to the number of parameters of the problem (equal to the dimensions of $\vec{x}$) - thus, the Hessian grows quadratically with the number of parameters. Instead of working with the Hessian, *Quasi-Newton methods* compute an approximation of the Hessian or its inverse based on gradients.

Assume that we have a function $f(\vec{x})$ and evaluated the gradient $\nabla_{\vec{x}} f$ at two different points $\vec{x}_{t-1}$ and $\vec{x}_t$.
We compute the second order Taylor approximation of f around $\vec{x}_t$ as

$$f_{\text{approx}}(\vec{x}) = f(\vec{x}_t) + \nabla_{\vec{x}} f(\vec{x}_t)(\vec{x} - \vec{x}_t) + \frac{1}{2}(\vec{x} - \vec{x}_t)^\top \mathbf{H}_t(\vec{x} - \vec{x}_t) \tag{A.37}$$

where $\mathbf{H}_t$ is an approximation of the Hessian $\boldsymbol{\nabla^2_{\vec{x}} f}(\vec{x}_t)$ at $\vec{x}_t$.
The gradient of f_{approx} A.37 with respect to $\vec{x}$ is given by

$$\nabla_{\vec{x}} f_{\text{approx}}(\vec{x}) = \nabla_{\vec{x}} f(\vec{x}_t) + \mathbf{H}_t(\vec{x} - \vec{x}_t) \tag{A.38}$$

We require that the gradients of the approximation f_{approx} are the same as the true gradients at $\vec{x}_{t-1}$ and $\vec{x}_t$. Thus

$$\begin{aligned} \nabla_{\vec{x}} f_{\text{approx}}(\vec{x}_{t-1}) &= \nabla_{\vec{x}} f(\vec{x}_{t-1}) \\ \nabla_{\vec{x}} f_{\text{approx}}(\vec{x}_t) &= \nabla_{\vec{x}} f(\vec{x}_t) \end{aligned} \tag{A.39}$$

Note that $\nabla_{\vec{x}} f_{\text{approx}}(\vec{x}_t) = \nabla_{\vec{x}} f(\vec{x}_t)$ holds by construction of f_{approx}. Plugging $\nabla_{\vec{x}} f_{\text{approx}}(\vec{x}_{t-1}) = \nabla_{\vec{x}} f(\vec{x}_{t-1})$ into A.38 yields

$$\nabla_{\vec{x}} f_{\text{approx}}(\vec{x}_{t-1}) = \nabla_{\vec{x}} f(\vec{x}_t) + \mathbf{H}_t(\vec{x}_{t-1} - \vec{x}_t) = \nabla_{\vec{x}} f(\vec{x}_{t-1}) \tag{A.40}$$

Rearranging A.40 yields

$$\begin{aligned} &\nabla_{\vec{x}} f(\vec{x}_t) + \mathbf{H}_t(\vec{x}_{t-1} - \vec{x}_t) = \nabla_{\vec{x}} f(\vec{x}_{t-1}) \\ &\Leftrightarrow \mathbf{H}_t(\vec{x}_{t-1} - \vec{x}_t) = \nabla_{\vec{x}} f(\vec{x}_{t-1}) - \nabla_{\vec{x}} f(\vec{x}_t) \\ &\Leftrightarrow \mathbf{H}_t(\vec{x}_t - \vec{x}_{t-1}) = \nabla_{\vec{x}} f(\vec{x}_t) - \nabla_{\vec{x}} f(\vec{x}_{t-1}) \end{aligned} \tag{A.41}$$

We obtain the *secant condition*, which is given by

$$\mathbf{H}_t \vec{s}_t = \vec{y}_t \tag{A.42}$$

where $\vec{s}_t = (\vec{x}_t - \vec{x}_{t-1})$ and $\vec{y}_t = \nabla_{\vec{x}} f(\vec{x}_t) - \nabla_{\vec{x}} f(\vec{x}_{t-1})$.
The secant condition makes sure that the approximation $\mathbf{H}_t$ "behaves like a real Hessian".

Besides the secant condition, we want the approximation $\mathbf{H}_t$ to be close to the previous approximation of the Hessian $\mathbf{H}_{t-1}$. Furthermore, we want the approximation of the Hessian to be symmetric.
Finally, we obtain the following optimization problem for computing an approximation of the Hessian

$$\begin{aligned} &\mathbf{H}_t = \arg\min_{\mathbf{H}} \|\mathbf{H} - \mathbf{H}_{t-1}\| \\ &\text{s.t.} \\ &\mathbf{H}^\top = \mathbf{H} \quad \text{and} \quad \mathbf{H}(\vec{x}_t - \vec{x}_{t-1}) = \nabla \vec{x} f(\vec{x}_t) - \nabla \vec{x} f(\vec{x}_{t-1}) \end{aligned} \tag{A.43}$$

where different matrix norms $\|\cdot\|$ results in different methods. In the next section we discuss the BFGS method.

A.3.5.1 BFGS

The *Broyden-Fletcher-Goldfarb-Shanno* (short: *BFGS*) method is a popular quasi-newton method.
The BFGS method computes the approximation $\mathbf{H}_{t+1}$ by adding a rank two matrix to the previous approximation $\mathbf{H}_t$

$$\mathbf{H}_t = \mathbf{H}_{t-1} + \alpha \vec{u}\vec{u}^\top + \beta \vec{v}\vec{v}^\top \tag{A.44}$$

Note that if $\mathbf{H}_{t-1}$ is symmetric, so is $\mathbf{H}_t$.
Applying the secant condition A.42 to A.44 yields

$$\begin{aligned} \mathbf{H}_t \vec{s}_t &= \vec{y}_t \\ \left(\mathbf{H}_{t-1} + \alpha \vec{u}\vec{u}^\top + \beta \vec{v}\vec{v}^\top\right) \vec{s}_t &= \vec{y}_t \\ \alpha \vec{u}\vec{u}^\top \vec{s}_t + \beta \vec{v}\vec{v}^\top \vec{s}_t &= \vec{y}_t - \mathbf{H}_{t-1}\vec{s}_t \end{aligned} \tag{A.45}$$

If we "arbitrarily" set

$$\begin{aligned} \vec{u} &= \vec{y}_t \\ \vec{v} &= \mathbf{H}_{t-1}\vec{s}_t \end{aligned} \tag{A.46}$$

we obtain

$$\begin{aligned}\alpha\vec{y}_t\vec{y}_t^\top\vec{s}_t + \beta\mathbf{H}_{t-1}\vec{s}_t\left(\mathbf{H}_{t-1}\vec{s}_t\right)^\top\vec{s}_t &= \vec{y}_t - \mathbf{H}_{t-1}\vec{s}_t \\ \alpha\vec{y}_t\vec{y}_t^\top\vec{s}_t + \beta\mathbf{H}_{t-1}\vec{s}_t\vec{s}_t^\top\mathbf{H}_{t-1}^\top\vec{s}_t &= \vec{y}_t - \mathbf{H}_{t-1}\vec{s}_t\end{aligned} \tag{A.47}$$

When solving A.47 for α and β, we find that

$$\begin{aligned}\alpha &= \frac{1}{\vec{y}_t^\top\vec{s}_t} \\ \beta &= -\frac{1}{\vec{s}_t^\top\mathbf{H}_{t-1}\vec{s}_t}\end{aligned} \tag{A.48}$$

By plugging everything back into A.44, we obtain the BFGS update of the approximation of the Hessian

$$\begin{aligned}&\mathbf{H}_t = \mathbf{H}_{t-1} + \frac{\vec{y}_t\vec{y}_t^\top}{\vec{y}_t^\top\vec{s}_t} - \frac{\mathbf{H}_{t-1}\vec{s}_t\vec{s}_t^\top\mathbf{H}_{t-1}}{\vec{s}_t^\top\mathbf{H}_{t-1}\vec{s}} \\ &\text{where} \\ &\vec{s}_t = \vec{x}_{t+1} - \vec{x}_t \qquad \vec{y}_t = \nabla_{\vec{x}}f(\vec{x}_{t+1}) - \nabla_{\vec{x}}f(\vec{x}_t) \\ &\mathbf{H}_0 = \mathbb{I}\end{aligned} \tag{A.49}$$

where in practice the initialization $\mathbf{H}_0 = \mathbb{I}$ is sometimes replaced by a "smarter initialization".

A.3.5.1.1 Updating the inverse

In A.49 we compute an approximation of the Hessian. However, we need its inverse $\mathbf{H}_t^{-1}$ for computing a step in Newtons's method. We could invert $\mathbf{H}_t$ manually but this might be computational expensive. A better approach would be to update and maintain the inverse $\mathbf{H}_t^{-1}$ instead of $\mathbf{H}_t$. We can do so by using the Woodbury matrix identity[11] which is given by

$$(\mathbf{A}+\mathbf{UDV})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}(\mathbf{D}^{-1} + \mathbf{VA}^{-1}\mathbf{U})^{-1}\mathbf{VA}^{-1} \tag{A.50}$$

We can recover A.49 by setting

$$\begin{aligned}\mathbf{A} &= \mathbf{H}_t \\ \mathbf{U} &= \begin{pmatrix}\mathbf{H}_{t-1}\vec{s}_t & \vec{y}_t\end{pmatrix} \\ \mathbf{V} &= \begin{pmatrix}\vec{s}_t^\top\mathbf{H}_{t-1} \\ \vec{y}_t^\top\end{pmatrix} \\ \mathbf{D} &= \begin{pmatrix}-\frac{1}{\vec{s}_t\mathbf{H}_{t-1}\vec{s}_t} & 0 \\ 0 & \frac{1}{\vec{y}_t^\top\vec{s}_t}\end{pmatrix}\end{aligned} \tag{A.51}$$

[11] see https://en.m.wikipedia.org/wiki/Woodbury_matrix_identity

The inverse of $\mathbf{D}$ is given by

$$\mathbf{D}^{-1} = \begin{pmatrix} -\vec{s}_t^\top \mathbf{H}_{t-1}\vec{s}_t & 0 \\ 0 & \vec{y}_t^\top \vec{s}_t \end{pmatrix} \tag{A.52}$$

Plugging A.51 and A.52 into A.50 yields

$$\begin{aligned}
\mathbf{H}_t^{-1} &= \left(\mathbf{H}_{t-1} + \begin{pmatrix}\mathbf{H}_{t-1}\vec{s}_t & \vec{y}_t\end{pmatrix} \begin{pmatrix} -\frac{1}{\vec{s}_t\mathbf{H}_{t-1}\vec{s}_t} & 0 \\ 0 & \frac{1}{\vec{y}_t^\top \vec{s}_t}\end{pmatrix} \begin{pmatrix}\vec{s}_t^\top \mathbf{H}_{t-1} \\ \vec{y}_t^\top\end{pmatrix}\right)^{-1} \\
&= \mathbf{H}_{t-1}^{-1} - \mathbf{H}_{t-1}^{-1}\begin{pmatrix}\mathbf{H}_{t-1}\vec{s}_t & \vec{y}_t\end{pmatrix} \\
&\quad \left(\begin{pmatrix} -\vec{s}_t^\top \mathbf{H}_{t-1}\vec{s}_t & 0 \\ 0 & \vec{y}_t^\top \vec{s}_t\end{pmatrix} + \begin{pmatrix}\vec{s}_t^\top \mathbf{H}_{t-1} \\ \vec{y}_t^\top\end{pmatrix}\mathbf{H}_{t-1}^{-1}\begin{pmatrix}\mathbf{H}_{t-1}\vec{s}_t & \vec{y}_t\end{pmatrix}\right)^{-1} \begin{pmatrix}\vec{s}_t^\top \mathbf{H}_{t-1} \\ \vec{y}_t^\top\end{pmatrix}\mathbf{H}_{t-1}^{-1} \\
&= \mathbf{H}_{t-1}^{-1} - \mathbf{H}_{t-1}^{-1}\begin{pmatrix}\mathbf{H}_{t-1}\vec{s}_t & \vec{y}_t\end{pmatrix}\left(\begin{pmatrix} -\vec{s}_t^\top \mathbf{H}_{t-1}\vec{s}_t & 0 \\ 0 & \vec{y}_t^\top \vec{s}_t\end{pmatrix} + \begin{pmatrix}\vec{s}_t^\top \mathbf{H}_{t-1}\vec{s}_t & \vec{s}_t^\top \vec{y}_t \\ \vec{y}_t^\top \vec{s}_t & \vec{y}_t^\top \mathbf{H}_{t-1}^{-1}\vec{y}_t\end{pmatrix}\right)^{-1} \\
&\quad \begin{pmatrix}\vec{s}_t^\top \mathbf{H}_{t-1} \\ \vec{y}_t^\top\end{pmatrix}\mathbf{H}_{t-1}^{-1} \\
&= \mathbf{H}_{t-1}^{-1} - \mathbf{H}_{t-1}^{-1}\begin{pmatrix}\mathbf{H}_{t-1}\vec{s}_t & \vec{y}_t\end{pmatrix}\begin{pmatrix} 0 & \vec{s}_t^\top \vec{y}_t \\ \vec{y}_t^\top \vec{s}_t & \vec{y}_t^\top \vec{s}_t + \vec{y}_t^\top \mathbf{H}_{t-1}\vec{y}_t\end{pmatrix}^{-1}\begin{pmatrix}\vec{s}_t^\top \mathbf{H}_{t-1} \\ \vec{y}_t^\top\end{pmatrix}\mathbf{H}_{t-1}^{-1} \\
&= \mathbf{H}_{t-1}^{-1} - \mathbf{H}_{t-1}^{-1}\begin{pmatrix}\mathbf{H}_{t-1}\vec{s}_t & \vec{y}_t\end{pmatrix}\begin{pmatrix} -\frac{\vec{y}_t^\top \vec{s}_t + \vec{y}_t^\top \mathbf{H}_{t-1}^{-1}\vec{y}_t}{(\vec{s}_t^\top \vec{y}_t)^2} & \frac{1}{\vec{s}_t^\top \vec{y}_t} \\ \frac{1}{\vec{s}_t^\top \vec{y}_t} & 0\end{pmatrix}\begin{pmatrix}\vec{s}_t^\top \mathbf{H}_{t-1} \\ \vec{y}_t^\top\end{pmatrix}\mathbf{H}_{t-1}^{-1} \\
&= \mathbf{H}_{t-1}^{-1} - \begin{pmatrix}\vec{s}_t & \mathbf{H}_{t-1}^{-1}\vec{y}_t\end{pmatrix}\begin{pmatrix} -\frac{\vec{y}_t^\top \vec{s}_t + \vec{y}_t^\top \mathbf{H}_{t-1}^{-1}\vec{y}_t}{(\vec{s}_t^\top \vec{y}_t)^2} & \frac{1}{\vec{s}_t^\top \vec{y}_t} \\ \frac{1}{\vec{s}_t^\top \vec{y}_t} & 0\end{pmatrix}\begin{pmatrix}\vec{s}_t^\top \\ \vec{y}_t^\top \mathbf{H}_{t-1}^{-1}\end{pmatrix} \\
&= \mathbf{H}_{t-1}^{-1} - \begin{pmatrix}\vec{s}_t & \mathbf{H}_{t-1}^{-1}\vec{y}_t\end{pmatrix}\begin{pmatrix} -\frac{\vec{y}_t^\top \vec{s}_t \vec{s}_t^\top + \vec{y}_t^\top \mathbf{H}_{t-1}^{-1}\vec{y}_t \vec{s}_t^\top}{(\vec{s}_t^\top \vec{y}_t)^2} + \frac{\vec{y}_t^\top \mathbf{H}_{t-1}^{-1}}{\vec{s}_t^\top \vec{y}_t} \\ \frac{\vec{s}_t^\top}{\vec{s}_t^\top \vec{y}_t}\end{pmatrix} \\
&= \mathbf{H}_{t-1}^{-1} + \frac{\vec{s}_t\vec{s}_t^\top}{\vec{s}_t^\top \vec{y}_t} + \frac{\vec{s}_t\vec{y}_t^\top \mathbf{H}_{t-1}^{-1}\vec{y}\vec{s}_t^\top}{(\vec{s}_t^\top \vec{y}_t)^2} - \frac{\vec{s}_t\vec{y}_t^\top \mathbf{H}_{t-1}^{-1}}{\vec{s}_t^\top \vec{y}_t} - \frac{\mathbf{H}_{t-1}^{-1}\vec{y}_t\vec{s}_t^\top}{\vec{s}_t^\top \vec{y}_t} \\
&= \mathbf{H}_{t-1}^{-1} - \frac{\vec{s}_t\vec{y}_t^\top \mathbf{H}_{t-1}^{-1} + \mathbf{H}_{t-1}^{-1}\vec{y}_t\vec{s}_t^\top}{\vec{y}_t^\top \vec{s}_t} + \frac{\vec{s}_t\vec{s}_t^\top}{\vec{y}_t^\top \vec{s}_t}\Big(\mathbb{I} + \frac{\vec{y}_t^\top \mathbf{H}_{t-1}^{-1}\vec{y}_t}{\vec{y}_t^\top \vec{s}_t}\Big) \\
&= \Big(\mathbb{I} - \frac{\vec{s}_t\vec{y}_t^\top}{\vec{y}_t^\top \vec{s}_t}\Big)\mathbf{H}_{t-1}^{-1}\Big(\mathbb{I} - \frac{\vec{y}_t\vec{s}_t^\top}{\vec{y}_t^\top \vec{s}_t}\Big) + \frac{\vec{s}_t\vec{s}_t^\top}{\vec{y}_t^\top \vec{s}_t}
\end{aligned} \tag{A.53}$$

Finally, we obtain the bfgs update for updating the inverse approximation of the Hessian.

$$\begin{aligned}\mathbf{H}_t^{-1} &= \Big(\mathbb{I} - \frac{\vec{s}_t\vec{y}_t^\top}{\vec{y}_t^\top\vec{s}_t}\Big)\mathbf{H}_{t-1}^{-1}\Big(\mathbb{I} - \frac{\vec{y}_t\vec{s}_t^\top}{\vec{y}_t^\top\vec{s}_t}\Big) + \frac{\vec{s}_t\vec{s}_t^\top}{\vec{y}_t^\top\vec{s}_t} \\ &= \Big(\mathbb{I} - \rho_t\vec{s}_t\vec{y}_t^\top\Big)\mathbf{H}_{t-1}^{-1}\Big(\mathbb{I} - \rho_t\vec{y}_t\vec{s}_t^\top\Big) + \rho_t\vec{s}_t\vec{s}_t^\top \\ &= \mathbf{V}_t^\top\mathbf{H}_{t-1}^{-1}\mathbf{V}_t + \rho_t\vec{s}_t\vec{s}_t^\top \\ &\text{where} \\ \vec{s}_t &= \vec{x}_t - \vec{x}_{t-1} \qquad \vec{y}_t = \nabla_{\vec{x}}f(\vec{x}_t) - \nabla_{\vec{x}}f(\vec{x}_{t-1}) \quad \rho_t = \frac{1}{\vec{y}_t^\top\vec{s}_t} \\ \mathbf{V}_t &= \mathbb{I} - \rho_t\vec{y}_t\vec{s}_t^\top \quad \mathbf{H}_0^{-1} = \mathbb{I}\end{aligned} \tag{A.54}$$

A.3.5.2 L-BFGS

The *Limited-memory BFGS* (*L-BFGS*) algorithm computes the direction $\vec{d} = \mathbf{H}_t^{-1}\nabla_{\vec{x}}f(\vec{x}_t)$ without explicitly computing $\mathbf{H}_t^{-1}$. It does so by using only the last m values of $\vec{s}_i$, $\vec{y}_i$ and ρ_i for constructing the approximation of the inverted Hessian $\mathbf{H}_t^{-1}$ (see A.54). Because of this, we can derive an iterative algorithm (the L-BFGS algorithm) for computing $\vec{d}$.
This algorithm is very memory efficient because we do not have to compute and store the inverted Hessian. We only need to store the last m values of $\vec{s}_i$, $\vec{y}_i$ and ρ_i. The quantity $\vec{d} = \mathbf{H}_t^{-1}\nabla_{\vec{x}}f(\vec{x}_t)$ is obtained by computing a bunch of scalar products and vector additions.

The L-BFGS algorithm is described in Algorithm 17[12].

A.3.6. Choosing the step length

We assume that we already found a step direction $\vec{d}$ for updating our current estimate $\vec{x}$ for minimizing the function f. Finding the optimal step length η can be formulated as the following optimization problem

$$\eta = \arg\min_{\eta} f(\vec{x} + \eta\vec{d})) \tag{A.55}$$

The step direction $\vec{d}$ differs from algorithm to algorithm. In the gradient descent algorithm $\vec{d} = -\nabla_{\vec{x}}f(\vec{x})$, in Newton's method $\vec{d} = -\boldsymbol{\nabla}^{\mathbf{2}}_{\vec{\boldsymbol{x}}}\boldsymbol{f}(\vec{x})^{-1}\nabla_{\vec{x}}f(\vec{x})$

[12] taken & modified from [9]

Algorithm 17 L-BFGS algorithm

$\vec{q} = \nabla_{\vec{x}} f(\vec{x}_t)$
for $i = t-1, ..., t-m$ **do**
    $\alpha_i = \rho_i \vec{s}_i^\top \vec{q}$
    $\vec{q} = \vec{q} - \alpha_i \vec{y}_i$
end for
$\vec{d} = \mathbf{H}_0^{-1} \vec{q}$
for $i = t-m, ..., t-1$ **do**
    $\vec{d} = \vec{d} + \vec{s}_i(\alpha_i - \rho_i \vec{y}_i^\top \vec{d})$
end for ▷ Now: $\vec{d} = \mathbf{H}_t^{-1} \nabla_{\vec{x}} f(\vec{x}_t)$

where $\boldsymbol{\nabla^2_{\vec{x}} f}(\vec{x})^{-1}$ is just an approximation in Quasi-Newton methods.

In the ideal case we can solve A.55 analytically. If we can not do so, we use a *inexact line search*. In an inexact line search, we simply try many possible values of η and select the best one. It is called inexact because we might not find the best step size η. However, a "good" value of η is often sufficient to make reasonable progress.

A set of constraints on the step length η for guaranteeing sufficient progress is given by the *Wolfe conditions*[13]

$$f(\vec{x} + \eta\vec{d}) \leq f(\vec{x}) + c_1 \eta \vec{d}^\top \nabla_{\vec{x}} f(\vec{x}) \tag{A.56}$$

$$-\vec{d}^\top \nabla \vec{x} f(\vec{x} + \eta\vec{d}) \leq -c_2 \vec{d}^\top \nabla_{\vec{x}} f(\vec{x}) \tag{A.57}$$

where c_1 and c_2 are chosen such that $0 < c_1 < c_2 < 1$. c_1 and c_2 are hyperparameters which define acceptable step length.
Note that A.56 is also called *Armijo condition* and A.57 is also called *curvature condition*.

The *backtracking line search algorithm* is an algorithm for computing an acceptable step length (acceptable according to the Wolfe conditions) by starting with a step length (e.g. $\eta = 1.0$) and then repeatedly reducing the step length until Armijo's condition A.56 is satisfied. Because of this procedure of reducing the step length, we are guaranteed to satisfy A.57 too, although we do not use it as an termination criterion in the algorithm.
The backtracking line search algorithm is described in algorithm 18[14]

[13] see `https://en.wikipedia.org/wiki/Wolfe_conditions`
[14] see [9]

Algorithm 18 Backtracking line search

$\eta = \eta_0$ ▷ Inital step size (usually $\eta_0 = 1.0$)
$c \in (0, 1)$ ▷ c_1 in A.56
$\rho \in (0, 1)$ ▷ Shrinkage factor
repeat
    $\eta = \rho\eta$ ▷ Reduce step size η
until $f(\vec{x} + \eta\vec{d}) \leq f(\vec{x}) + c\eta\vec{d}^\top \nabla_{\vec{x}} f(\vec{x})$

A.3.7. Coordinate descent

We consider a convex optimization problem of the following form

$$\min_{\vec{\theta} \in \mathbb{R}^d} f(\vec{\theta}) + \sum_j h_j((\vec{\theta})_j) \tag{A.58}$$

where f and h_j are convex functions. Furthermore, we assume that f is differentiable everywhere (viz. f is a smooth function).
Then, the *coordinate descent algorithm* is guaranteed to solve A.58.

The coordinate descent algorithm starts with an initial "guess" of $\vec{\theta}$ and then repeatedly iterates over all dimensions of $\vec{\theta}$ and updates each coordinate/dimension of $\vec{\theta}$ separately by computing

$$(\vec{\theta})_i^{t+1} = \arg\min_{\theta_i \in \mathbb{R}} f(..., (\vec{\theta})_{i-1}^{t+1}, \theta_i, (\vec{\theta})_{i+1}^{t}, ...) \tag{A.59}$$

Note that we always make use of the currently best known values of $\vec{\theta}$. We do not wait until an iteration is over, but instead we update the values of $\vec{\theta}$ immediately.
The final algorithm is described in Algorithm 19

Algorithm 19 Coordinate descent algorithm

$\vec{\theta} = ...$ ▷ Initialize $\vec{\theta}$
repeat
    for all $(\vec{\theta})_i$ **do** ▷ Iterate over all dimensions of $\vec{\theta}$
        $(\vec{\theta})_i = \arg\min_{\theta_i \in \mathbb{R}} f(..., \theta_i, ...)$ ▷ Update $\vec{\theta}$
    end for
until convergence

A.4. Linear programming

A *linear program (LP)* is an instance of a *constrained convex optimization problem* and defined as

$$\begin{aligned} &\min_{\vec{x} \in \mathbb{R}^d} \vec{c}^\top \vec{x} \\ &\text{s.t.} \\ &\mathbf{A}\vec{x} = \vec{b} \\ &\mathbf{G}\vec{x} \geq \vec{0} \end{aligned} \tag{A.60}$$

where the vector $\vec{c} \in \mathbb{R}^d$ specifies the optimization objective, the matrix $\mathbf{A} \in \mathbb{R}^{m \times d}$ and the vector $\vec{b} \in \mathbb{R}$ specify the m equality constraints and the matrix $\mathbf{G} \in \mathbb{R}^{k \times d}$ specifies the k inequality constraints[15].
In plain English: A linear program looks for a solution $\vec{x}$, such that the linear combination $\vec{c}^\top \vec{x}$ is minimized and (if present) all equality and inequality constraints are fulfilled. Note that all constrains must be specified as a linear combination of $\vec{x}$.

There exist a bunch of algorithms (e.g. simplex algorithm or interior point method) for solving a linear program. However, a detailed discussion of those algorithms is behind he scope of this text. We refer the interested reader to specific material on linear programming.
For us it is sufficient to keep in mind that there are algorithms for solving linear programs efficiently.

A.4.1. Example

Suppose that we want to solve the following *maximization problem*

$$\begin{aligned} &\max_{x_1, x_2 \in \mathbb{R}} x_1 + 2x_2 \\ &\text{s.t. } x_1 \geq 0 \quad x_2 \geq 0 \\ &x_1 + x_2 = 42 \end{aligned} \tag{A.61}$$

[15] note that we could convert all equality constraints into inequality contraints - we would need 2 inequality constraints for each equality constraint

We can transform A.61 into a linear program by setting

$$\begin{aligned}
\mathbf{A} &= \begin{pmatrix} 1 & 1 \end{pmatrix} \\
\mathbf{G} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
\vec{c} &= (-1, -2)^\top \\
\vec{b} &= (42)^\top \\
\vec{x} &= (x_1, x_2)^\top
\end{aligned} \tag{A.62}$$

A.5. Quadratic programming

A *quadratic program (QP)* is an instance of a *constrained convex optimization problem* and defined as

$$\begin{aligned}
&\min_{\vec{x} \in \mathbb{R}^d} \frac{1}{2} \vec{x}^\top \mathbf{Q} \vec{x} + \vec{c}^\top \vec{x} \\
&\text{s.t.} \\
&\mathbf{A}\vec{x} \leq \vec{b}
\end{aligned} \tag{A.63}$$

where $\mathbf{Q} \in \mathbb{R}^{d \times d}$ is a symmetric matrix. If $\mathbf{Q}$ is a symmetric postive semidefinite matrix, A.63 is called a *convex quadratic program* because $\mathbf{Q}$ is the Hessian of $\frac{1}{2}\vec{x}^\top \mathbf{Q}\vec{x} + \vec{c}^\top \vec{x}$.

Like in linear programming, there exist a bunch of algorithms for solving a quadratic program efficiently.

A.5.1. Eample

Suppose that we want to solve the following *minimization problem*

$$\begin{aligned}
&\min_{x_1, x_2 \in \mathbb{R}} 2x_1^2 + 2x_1 x_2 + x_1 \\
&\text{s.t. } x_1 \geq 0 \quad x_2 \geq 0 \\
&x_1 + x_2 = 1
\end{aligned} \tag{A.64}$$

We can transform A.64 into a quadratic program by setting

$$\begin{aligned}
\mathbf{Q} &= \begin{pmatrix} 4 & 2 \\ 2 & 0 \end{pmatrix} \\
\mathbf{A} &= \begin{pmatrix} -1 & 0 \\ 0 & -1 \\ -1 & -1 \\ 1 & 1 \end{pmatrix} \\
\vec{c} &= (1, 0)^\top \\
\vec{b} &= (0, 0, -1, 1)^\top \\
\vec{x} &= (x_1, x_2)^\top
\end{aligned} \tag{A.65}$$

where we wrote the equality constraint $x_1 + x_2 = 1$ as two inequality constraints $x_1 + x_2 \leq 1$ and $x_1 + x_2 \geq 1$.

A.6. Lagrangian duality

We consider the following optimization problem

$$\begin{aligned}
&\min_{\vec{x} \in \mathbb{R}^d} f(\vec{x}) \\
&\text{s.t.} \\
&g_i(\vec{x}) \leq 0 \; \forall \, i \in \{1, ..., m\} \\
&h_j(\vec{x}) = 0 \; \forall \, j \in \{1, ..., k\}
\end{aligned} \tag{A.66}$$

where $f : \mathbb{R}^d \mapsto \mathbb{R}$, $g_i : \mathbb{R}^d \mapsto \mathbb{R}$ and $h_j : \mathbb{R}^d \mapsto \mathbb{R}$ are arbitrary function - *not necessarily convex*.
Furthermore, we define $\vec{x}_*$ to be the minimizer of A.66 and $p_* = f(\vec{x}_*)$.

Next, we integrate the constraints into the objective and obtain the equivalent optimization problem.

$$\min_{\vec{x} \in \mathbb{R}^d} f(\vec{x}) + \sum_i I_-(g_i(\vec{x})) + \sum_j I_0(h_j(\vec{x})) \tag{A.67}$$

where

$$I_-(u) = \begin{cases} 0 & \text{if } u \leq 0 \\ \infty & \text{otherwise} \end{cases} \tag{A.68}$$

$$I_0(u) = \begin{cases} 0 & \text{if } u = 0 \\ \infty & \text{otherwise} \end{cases} \tag{A.69}$$

The optimization problem A.66 or A.67 (they are equivalent to each other) are called *primal problem*.

Because $I_-(g_i(\vec{x}))$ and $I_0(h_j(\vec{x}))$ are not that nice to work with - they are not steady and either zero or infinity, we approximate them by "more friendly" lower bounds.
We introduce the following lower bounds

$$I_-(g_i(\vec{x})) \geq (\vec{\lambda}_1)_i g_i(\vec{x}) \quad \text{where } (\vec{\lambda}_1)_i \in \mathbb{R}_+ \tag{A.70}$$

$$I_0(h_j(\vec{x})) \geq (\vec{\lambda}_2)_j h_j(\vec{x}) \quad \text{where } (\vec{\lambda}_2)_j \in \mathbb{R} \tag{A.71}$$

See Fig. A.11 for an illustration.

Figure A.11: Lagrange multiplier

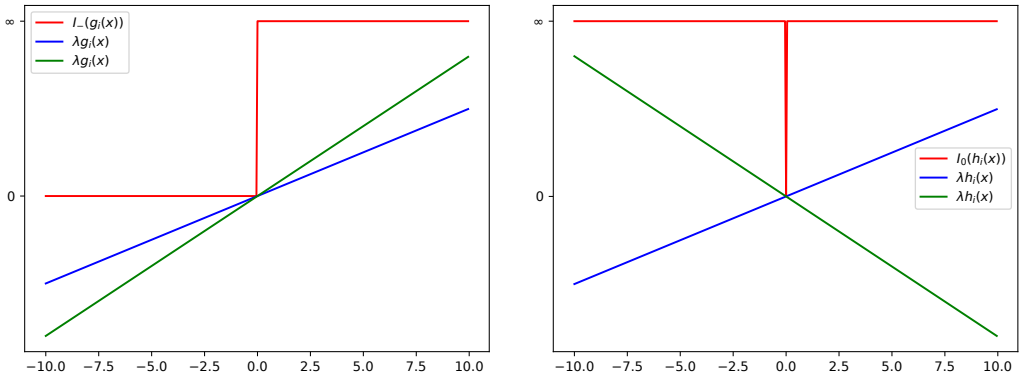


Figure A.12: Inequality constraint

Figure A.13: Equality constraint

By using the lower bounds A.70 and A.71, we define the *Lagrangian function* $\mathcal{L}$ of A.66 as

$$\mathcal{L}(\vec{x}, \vec{\lambda}_1, \vec{\lambda}_2) = f(x) + \sum_i (\vec{\lambda}_1)_i g_i(\vec{x}) + \sum_j (\vec{\lambda}_2)_j h_j(\vec{x}) \tag{A.72}$$

where $\vec{\lambda}_1$ and $\vec{\lambda}_2$ are called *Lagrange multipliers* or *dual variables*.

Because we replaced $I_-(g_i(\vec{x}))$ and $I_0(h_j(\vec{x}))$ by lower bounds, we know that

$$f(\vec{x}) + \sum_i I_-(g_i(\vec{x})) + \sum_j I_0(h_j(\vec{x})) \geq f(x) + \sum_i (\vec{\lambda}_1)_i g_i(\vec{x}) + \sum_j (\vec{\lambda}_2)_j h_j(\vec{x}) = \mathcal{L}(\vec{x}, \vec{\lambda}_1, \vec{\lambda}_2) \tag{A.73}$$

We can recover A.67 by maximizing A.72 over $\vec{\lambda}_1$ and $\vec{\lambda}_2$. If a constraint i is satisfied, we set the corresponding $(\vec{\lambda})_i$ to 0 and $(\vec{\lambda})_i \to \infty$ otherwise. Therefore, with a slight abuse of notation[16], we can write

$$f(\vec{x}) + \sum_i I_-(g_i(\vec{x})) + \sum_j I_0(h_j(\vec{x})) = \max_{\substack{\vec{\lambda}_1 \in \mathbb{R}^m_+, \\ \vec{\lambda}_2 \in \mathbb{R}^k}} \mathcal{L}(\vec{x}, \vec{\lambda}_1, \vec{\lambda}_2) \tag{A.74}$$

Because of A.74, we can rewrite A.67 as

$$\min_{\vec{x} \in \mathbb{R}^d} f(\vec{x}) + \sum_i I_-(g_i(\vec{x})) + \sum_j I_0(h_j(\vec{x})) = \min_{\vec{x} \in \mathbb{R}^d} \max_{\substack{\vec{\lambda}_1 \in \mathbb{R}^m_+, \\ \vec{\lambda}_2 \in \mathbb{R}^k}} \mathcal{L}(\vec{x}, \vec{\lambda}_1, \vec{\lambda}_2) \tag{A.75}$$

We define the *Lagrange dual function* $\mathcal{L}_D(\vec{\lambda}_1, \vec{\lambda}_2)$ of A.72 as

$$\mathcal{L}_D(\vec{\lambda}_1, \vec{\lambda}_2) = \min_{\vec{x} \in \mathbb{R}^d} \mathcal{L}(\vec{x}, \vec{\lambda}_1, \vec{\lambda}_2) \tag{A.76}$$

By construction we have that

$$\begin{aligned} f(\vec{x}) + \sum_i I_-(g_i(\vec{x})) + \sum_j I_0(h_j(\vec{x})) &\geq \mathcal{L}(\vec{x}, \vec{\lambda}_1, \vec{\lambda}_2) \\ &\geq \min_{\vec{x} \in \mathbb{R}^d} \mathcal{L}(\vec{x}, \vec{\lambda}_1, \vec{\lambda}_2) \\ &= \mathcal{L}_D(\vec{\lambda}_1, \vec{\lambda}_2) \end{aligned} \tag{A.77}$$

Because of A.77, we know that the dual is always a lower bound of the optimal value p_* from A.66.

$$\mathcal{L}_D(\vec{\lambda}_1, \vec{\lambda}_2) \leq p_* \tag{A.78}$$

Note: The Lagrange dual function A.78 is always a *concave function* (no matter what the functions f,g_i or h_i from the primal A.66 are).

[16] Actually, our notation is not quite right. We should write sup instead of max and inf instead of min. However, in order to not confuse mathematical less skilled students, we stick to max and min. We express our apologies to all mathematicians reading these notes.

Next, we define the *dual problem* as

$$\max_{\substack{\vec{\lambda}_1 \in \mathbb{R}^m_+, \\ \vec{\lambda}_2 \in \mathbb{R}^k}} \mathcal{L}_D(\vec{\lambda}_1, \vec{\lambda}_2) \tag{A.79}$$

Let $\vec{\lambda}_{1_*}$ and $\vec{\lambda}_{2_*}$ be the maximizers of A.79 and $d_* = \mathcal{L}_D(\vec{\lambda}_{1_*}, \vec{\lambda}_{2_*})$. Because of A.78 we always have

$$d_* \leq p_* \tag{A.80}$$

The value $p_* - d_*$ is called *duality gap*.

Because of A.80, we always have *weak duality* ($d_* \leq p_*$). Thus

$$\min_{\vec{x} \in \mathbb{R}^d} f(\vec{x}) + \sum_i I_-(g_i(\vec{x})) + \sum_j I_0(h_j(\vec{x})) \geq \max_{\substack{\vec{\lambda}_1 \in \mathbb{R}^m_+, \\ \vec{\lambda}_2 \in \mathbb{R}^k}} \mathcal{L}_D(\vec{\lambda}_1, \vec{\lambda}_2) \tag{A.81}$$

We say that *strong duality* holds, if $p_* - d_* = 0$. Thus[17]

$$\begin{aligned} \min_{\vec{x} \in \mathbb{R}^d} f(\vec{x}) + \sum_i I_-(g_i(\vec{x})) + \sum_j I_0(h_j(\vec{x})) &= \max_{\substack{\vec{\lambda}_1 \in \mathbb{R}^m_+, \\ \vec{\lambda}_2 \in \mathbb{R}^k}} \mathcal{L}_D(\vec{\lambda}_1, \vec{\lambda}_2) \\ &= \max_{\substack{\vec{\lambda}_1 \in \mathbb{R}^m_+, \vec{x} \in \mathbb{R}^d \\ \vec{\lambda}_2 \in \mathbb{R}^k}} \min \mathcal{L}(\vec{x}, \vec{\lambda}_1, \vec{\lambda}_2) \\ &= \min_{\vec{x} \in \mathbb{R}^d} \max_{\substack{\vec{\lambda}_1 \in \mathbb{R}^m_+, \\ \vec{\lambda}_2 \in \mathbb{R}^k}} \mathcal{L}(\vec{x}, \vec{\lambda}_1, \vec{\lambda}_2) \end{aligned} \tag{A.82}$$

In case of strong duality, solving the dual problem A.79 is equivalent to solving the primal problem A.66.

Because of weak duality, maximizing the dual $\mathcal{L}_D$ gives us a (non-trivial) lower bound of the primal solution. In the ideal case, we have strong duality and instead of solving the primal problem we can maximize the dual and obtain the optimal solution to the primal. Unfortunately, strong duality does not always hold.

There exist several *constraint qualifications*[18] (also called *regularity conditions*) defining conditions for strong duality. A very famous (and for us

[17] Note: Because of strong duality we can swap the min-max in A.75

[18] Note that these conditions are sufficient but not necessary

useful) constraint qualification is *Slater's condition*.
We consider the following optimization problem

$$\begin{aligned} &\min_{\vec{x} \in \mathbb{R}^d} f(\vec{x}) \\ &\text{s.t.} \\ &g_i(\vec{x}) \leq 0 \; \forall \, i \in \{1, ..., m\} \\ &\mathbf{A}\vec{x} = \vec{b} \end{aligned} \tag{A.83}$$

where $f : \mathbb{R}^d \mapsto \mathbb{R}$ and $g_i : \mathbb{R}^d \mapsto \mathbb{R}$ are convex functions[19]. Slater's condition requires that there exists a feasible point $\vec{x}_i$ such that $g_i(\vec{x}_i) < 0 \; \forall i$. If Slater's condition is fulfilled, strong duality is guaranteed.

A.6.1. Optimality conditions

If strong duality holds, we can define a couple of *necessary conditions for optimality*. That is, any pair of optimal dual and primal points $(\vec{x}_*, \vec{\lambda}_{1_*}, \vec{\lambda}_{2_*})$ must satisfy these conditions.

The gradient[20] of the Lagrange function with respect to the parameter

$$\nabla_{\vec{x}} \mathcal{L}(\vec{x}_*, \vec{\lambda}_{1_*}, \vec{\lambda}_{2_*}) = \nabla_{\vec{x}} f(\vec{x}_*) + \sum_i (\vec{\lambda}_{1_*})_i \nabla_{\vec{x}} g_i(\vec{x}_*) + \sum_j (\vec{\lambda}_{2_*})_j \nabla_{\vec{x}} h_j(\vec{x}_*) \tag{A.84}$$

must be equal to zero. Therefore

$$\begin{aligned} &\nabla_{\vec{x}} \mathcal{L}(\vec{x}_*, \vec{\lambda}_{1_*}, \vec{\lambda}_{2_*}) = \vec{0} \\ &\Leftrightarrow \\ &\nabla_{\vec{x}} f(\vec{x}_*) + \sum_i (\vec{\lambda}_{1_*})_i \nabla_{\vec{x}} g_i(\vec{x}_*) + \sum_j (\vec{\lambda}_{2_*})_j \nabla_{\vec{x}} h_j(\vec{x}_*) = \vec{0} \\ &\Leftrightarrow \\ &\nabla_{\vec{x}} f(\vec{x}_*) = -\sum_i (\vec{\lambda}_{1_*})_i \nabla_{\vec{x}} g_i(\vec{x}_*) - \sum_j (\vec{\lambda}_{2_*})_j \nabla_{\vec{x}} h_j(\vec{x}_*) \end{aligned} \tag{A.85}$$

When talking about optimality, we implicitly assume feasibility. Thus, all constraints must be satisfied.
Note that we can recover the original constraints by computing the derivative

[19] Note that the equality constraints h_j are replaced by an affine function
[20] we assume that the functions are differentiable

of the Lagrange function with respect to the Lagrange multipliers.

$$\begin{aligned} &\nabla_{\vec{\lambda}_2} \mathcal{L}(\vec{x}_*, \vec{\lambda}_{1_*}, \vec{\lambda}_{2_*}) = 0 \\ &\Leftrightarrow \\ &\sum_j h_j(\vec{x}_*) = 0 \end{aligned} \tag{A.86}$$

If strong duality holds, we know that

$$\sum_i (\vec{\lambda}_{1_*})_i \; g_i(\vec{x}_*) = 0 \tag{A.87}$$

Because $(\vec{\lambda}_{1_*})_i \geq 0 \; \forall \, i$, we can conclude that

$$\begin{aligned} &(\vec{\lambda}_{1_*})_i \; g_i(\vec{x}_*) = 0 \quad \forall \, i \in \{1, ..., m\} \\ &\Leftrightarrow \\ &(\vec{\lambda}_{1_*})_i > 0 \implies g_i(\vec{x}_*) = 0 \\ &\Leftrightarrow \\ &g_i(\vec{x}_*) < 0 \implies (\vec{\lambda}_{1_*})_i = 0 \end{aligned} \tag{A.88}$$

A.88 is also called *complementary slackness*.

By making use of the previously discussed conditions, we can define the *KKT-conditions* (Karush-Kuhn-Tucker conditions) as

1. $\nabla_{\vec{x}} \mathcal{L}(\vec{x}_*, \vec{\lambda}_{1_*}, \vec{\lambda}_{2_*}) = \vec{0}$
2. $h_j(\vec{x}_*) = 0 \quad \forall \, j \in \{1, ..., k\}$
3. $g_i(\vec{x}_*) \leq 0 \quad \forall \, i \in \{1, ..., m\}$
4. $(\vec{\lambda}_{1_*})_i > 0 \quad \forall \, i \in \{1, ..., m\}$
5. $(\vec{\lambda}_{1_*})_i \; g_i(\vec{x}_*) = 0 \quad \forall \, i \in \{1, ..., m\}$

The KKT-conditions A.6.1 are *necessary conditions* for optimality of any pair of optimal primal and dual points $(\vec{x}_*, \vec{\lambda}_{1_*}, \vec{\lambda}_{2_*})$.
If the primal is a *convex optimization problem*, the KTT-conditions A.6.1 are *sufficient conditions*, too.

A.6.2. Example

Suppose that we want to solve the following *minimization problem*

$$\begin{aligned} &\min_{x_1, x_2 \in \mathbb{R}} \; x_1^2 + x_2^2 \\ &\text{s.t. } 2x_1 + 2x_2 \geq 4 \end{aligned} \tag{A.89}$$

The Lagrange function of A.89 is given by

$$\mathcal{L}(x_1, x_2, \lambda) = x_1^2 + x_2^2 + \lambda\,(4 - 2x_1 - 2x_2) \tag{A.90}$$

The optimality conditions yield

$$\begin{aligned} &\frac{\partial \mathcal{L}}{\partial x_1} = 0 \\ &\Leftrightarrow 2x_1 - 2\lambda = 0 \\ &\Leftrightarrow \lambda = x_1 \end{aligned} \tag{A.91}$$

and

$$\begin{aligned} &\frac{\partial \mathcal{L}}{\partial x_2} = 0 \\ &\Leftrightarrow 2x_2 - 2\lambda = 0 \\ &\Leftrightarrow \lambda = x_2 \end{aligned} \tag{A.92}$$

Now, we know that $x_1 = x_2$. Substituting this into the original problem yields

$$\begin{aligned} &\min_{x_1 \in \mathbb{R}} \; 2x_1^2 \\ &\text{s.t. } 4x_1 \geq 4 \end{aligned} \tag{A.93}$$

From the constraint we find that $x_1 \geq 1$. Because the objective $2x_1^2$ is a non-decreasing function, we can conclude that the solution to our minimization problem is given by $x_1 = x_2 = 1$.

Note: In this toy example we were able to derive a closed form solution by looking at the optimality conditions, substituting them back into the original problem and "observed" the solution by making a simple argument about the constraint and objective. We did not even use duality.
Many problems are much more complicated and can not be solved that easily. It might be the case that neither the primal nor the dual has a closed form solution. Fruthermore, we might not always have strong duality.

A.7. Outlook

In this section we looked at convex functions and convex optimization problems. There is much more to say about convex optimization than we covered in the last sections - we omitted many/all proofs and derivations. However, the presented material will be sufficient for understanding and solving all optimization problems which we will encounter in the the rest of these notes. If you want to know more about convex optimization, a good starting point is the excellent textbook [3] about convex optimization and the coressponding (online) lecture "Convex optimization" I and II by Prof. Boyd (Stanford university).
Another great resource is [9] and the online lecture "Convex optimization" by Prof. Ryan Tibshirani[21].

[21] see `http://www.stat.cmu.edu/~ryantibs/`

A.8. Exercises

1. Prove that the following functions are convex.
   (a) $f(x) = 2x^4 - 42 \quad \text{with } x \in \mathbb{R}$
   (b) $f(x) = \|x\|_p \quad \text{with } x \in \mathbb{R},\ p \in \mathbb{N}$
   (c) $f(\vec{x}) = \mathbf{A}\vec{x} + \vec{b} \quad \text{with } \vec{x}, \vec{b} \in \mathbb{R}^d,\ \mathbf{A} \in \mathbb{R}^{d \times d}$
   (d) $f(x) = c \quad \text{with } c \in \mathbb{R}$
2. Show that, if $f : \mathbb{R} \mapsto \mathbb{R}$ and $g : \mathbb{R} \mapsto \mathbb{R}$ are convex functions so is $h(x) = f(x) + g(x)$.
3. Show that the following sets are convex.
   (a) $\mathbb{R}_+$
   (b) $[0, 1]$
   (c) $\{\mathbf{A} \mid \mathbf{A}^\top = \mathbf{A}, \quad \mathbf{A} \in \mathbb{R}^{d \times d}\}$
4. Prove that the following sets are *not* convex.
   (a) $\{\mathbf{A} \mid \text{rank}(\mathbf{A}) = k, \quad \mathbf{A} \in \mathbb{R}^{d \times d}\}$
5. Prove that the following functions are *concave*.
   (a) $f(x) = -(x)^2 \quad \text{with } x \in \mathbb{R}$
   (b) $f(\vec{x}) = \mathbf{A}\vec{x} + \vec{b} \quad \text{with } \vec{x}, \vec{b} \in \mathbb{R}^d,\ \mathbf{A} \in \mathbb{R}^{d \times d}$
6. Show that the following functions *are not* convex.
   (a) $f(x) = x^3 + 1 \quad \text{with } x \in \mathbb{R}$
   (b) $f(x) = \frac{1}{1+\exp(-2x)} \quad \text{with } x \in \mathbb{R}$
7. The log-trick.
   (a) Prove that $f(x) = x^6(1-x)^4$ is *not* a convex function $\forall\, x \in [0, 1]$.
   (b) Prove that $f(x) = -\log\left(x^6(1-x)^4\right)$ is a convex function $\forall\, x \in [0, 1]$.
   (c) Prove that

$$\underset{x \in [0,1]}{\arg\min} \ -\log\left(x^6(1-x)^4\right) = \underset{x \in [0,1]}{\arg\max}\ x^6(1-x)^4 \tag{A.94}$$

8. Solve the following optimization problem and prove that your solution is globally optimal.

$$\arg\min_{\vec{x} \in \mathbb{R}^2} \sum_i \|\vec{d_i} - \vec{x}\|_2^2 \tag{A.95}$$

where the set $\{\vec{d_i}\}$, $\vec{d_i} \in \mathbb{R}^2$, is given.

9. Argue why or why not, coordinate descent is suited for solving the following optimization problems

   (a) $\min_{\vec{x} \in \mathbb{R}^d} |\vec{x}^\top \vec{x} - 42|$.

   (b) $\min_{\vec{x} \in \mathbb{R}^d} \max(0, \vec{x}^\top \vec{a}) + \sum_j |(\vec{x})_j - 42|$ where $\vec{a} \in \mathbb{R}^d$.

   (c) $\min_{\vec{x} \in \mathbb{R}^d} ||\mathbf{A}\vec{x} - \vec{y}|| + \|\vec{x}\|_2^2$ where $\mathbf{A} \in \mathbb{R}^{n \times d}, \vec{y} \in \mathbb{R}^n$.

Appendix B

Probability theory & Statistical inference

This chapter gives a brief review of the most important concepts in probability theory and statistical inference.

B.1. Basic probability

A *random experiment* is a "process" which can be repeated infinitely often under the same conditions and its result/outcome can not be predicted for sure.
We denote the set of possible outcomes of a random experiment as Ω (also called *sample space*).

An event $\mathcal{A}$ is a subset of the sample space Ω

$$\mathcal{A} \subseteq \Omega \tag{B.1}$$

The complement of an event $\mathcal{A}$ is defined as

$$\bar{\mathcal{A}} = \Omega \setminus \mathcal{A} \tag{B.2}$$

A *probability*[1] P is a mapping from events to real numbers

$$P : \mathcal{A} \subseteq \Omega \mapsto \mathbb{R} \tag{B.3}$$

[1] according to [18]

where the mapping P satisfies

$$0 \leq P(\mathcal{A}) \leq 1 \quad \forall\, \mathcal{A} \subseteq \Omega \tag{B.4}$$

$$\begin{aligned} P(\Omega) &= 1 \\ P(\emptyset) &= 0 \end{aligned} \tag{B.5}$$

and

$$\begin{aligned} &P\left(\bigcup_i \mathcal{B}_i\right) = \sum_i P(\mathcal{B}_i) \\ &\text{where} \\ &\mathcal{B}_i \cap \mathcal{B}_j = \emptyset \;\forall\, i, j \end{aligned} \tag{B.6}$$

We interpret $P(\omega)$, $\omega \in \Omega$, as the probability that the outcome of the random experiment is ω. Furthermore, we interpret $P(\mathcal{A})$, $\mathcal{A} \subseteq \Omega$, as the probability that the outcome of the random experiment is contained in the set $\mathcal{A}$.

If Ω is finite and each $\omega \in \Omega$ is equally likely, we define the probability of an event $\mathcal{A}$ as

$$P(\mathcal{A}) = \frac{|\mathcal{A}|}{|\Omega|} \tag{B.7}$$

Note that a random experiment like B.7 is also called *Laplacian experiment*.

The probability of the complement of an event can be expressed as

$$P(\bar{\mathcal{A}}) = 1 - P(\mathcal{A}) \tag{B.8}$$

We formalize the statements "probability of events $\mathcal{A}$ *and* $\mathcal{B}$" as

$$P(\mathcal{A} \text{ and } \mathcal{B}) = P(\mathcal{A}, \mathcal{B}) = P(\mathcal{A} \cap \mathcal{B}) \tag{B.9}$$

and "probability of events $\mathcal{A}$ *or* $\mathcal{B}$" as

$$P(\mathcal{A} \text{ or } \mathcal{B}) = P(\mathcal{A} \cup \mathcal{B}) \tag{B.10}$$

The probability of the union of events $\mathcal{A}$ and $\mathcal{B}$ is given by

$$P(\mathcal{A} \cup \mathcal{B}) = P(\mathcal{A}) + P(\mathcal{B}) - P(\mathcal{A} \cap \mathcal{B}) \tag{B.11}$$

B.1.1. Conditional probabilities

The *conditional probability* of an event $\mathcal{A}$ given an event $\mathcal{B}$ is defined as

$$P(\mathcal{A} \mid \mathcal{B}) = \frac{P(\mathcal{A} \cap \mathcal{B})}{P(\mathcal{B})} \tag{B.12}$$

We interpret $P(\mathcal{A} \mid \mathcal{B})$ as the probability of the event $\mathcal{A}$ where we only consider possibilities where event $\mathcal{B}$ occurs.
By rearranging the terms in B.12, we obtain the *chain rule* for factorizing the joint probability

$$\begin{aligned} P(\mathcal{A}, \mathcal{B}) &= P(\mathcal{A} \cap \mathcal{B}) \\ &= P(\mathcal{A} \mid \mathcal{B}) P(\mathcal{B}) \\ &= P(\mathcal{B} \mid \mathcal{A}) P(\mathcal{A}) \end{aligned} \tag{B.13}$$

Substituting B.13 back into B.12 yields a formula known as *Bayes' rule*

$$\begin{aligned} P(\mathcal{A} \mid \mathcal{B}) &= \frac{P(\mathcal{A} \cap \mathcal{B})}{P(\mathcal{B})} \\ &= \frac{P(\mathcal{B} \mid \mathcal{A}) P(\mathcal{A})}{P(\mathcal{B})} \end{aligned} \tag{B.14}$$

The denominator of B.14 could be computed by using the *law of total probability* which is given by

$$\begin{aligned} &P(\mathcal{A}) = \sum_i P(\mathcal{A} \mid \mathcal{B}_i) P(\mathcal{B}_i) \\ &\text{where} \\ &\bigcup_i \mathcal{B}_i = \Omega \\ &\mathcal{B}_i \cap \mathcal{B}_j = \emptyset \; \forall \, i, j \end{aligned} \tag{B.15}$$

B.1.2. Independence

We say that two events $\mathcal{A}$ and $\mathcal{B}$ are called *independent* if and only if

$$P(\mathcal{A} \cap \mathcal{B}) = P(\mathcal{A}) P(\mathcal{B}) \tag{B.16}$$

Plugging B.16 into B.12 yields

$$P(\mathcal{A} \mid \mathcal{B}) = P(\mathcal{A}) \tag{B.17}$$

Thus, knowing that the outcome is contained in event $\mathcal{B}$, does not tell us anything about the probability of the event $\mathcal{A}$.

B.2. Random variable

A *random variable* is a function that maps outcomes of an random experiment to real numbers

$$X : \Omega \mapsto \mathcal{R} \subseteq \mathbb{R} \tag{B.18}$$

We can think of it as some kind of "measure/summary" of the result/outcome of a random experiment.

We distinguish between *discrete* and *continuous random variables*. We call a random variable X discrete, if the image of X is discrete, and we call the variable continuous if the image is continuous.

A *multivariate random variable* (also called *random vector*) is very similar to a random variable but instead of mapping outcomes to real numbers, events are mapped to real-valued vectors

$$X : \Omega \mapsto \mathcal{R} \subseteq \mathbb{R}^d \tag{B.19}$$

A very simple, but in machine learning often used, random variable is the identity function.

$$\begin{aligned} &X : \Omega \mapsto \Omega \\ &X(x) = x \quad \forall\, x \in \Omega \end{aligned} \tag{B.20}$$

B.2.1. Algebraic operations

Let X and Y be two random variables with the same domain and $c \in \mathbb{R}$ a scalar. We define the following algebraic operations for constructing a new random variable Z:

1. $Z = X + Y$ with $Z(z) = X(z) + Y(z)$.
2. $Z = X \cdot Y$ with $Z(z) = X(z) \cdot Y(z)$.
3. $Z = c \cdot X$ with $Z(z) = c \cdot Y(z)$.

B.2.2. Cumulative distribution function

The *cumulative distribution function* $F_X(x)$, where $F_X : \mathbb{R} \mapsto [0, 1]$, of some random variable X computes the probability that the random variable takes on a value less or equal than x.

$$F_X(x) = P(X \leq x) = P(\{\omega \mid X(\omega) \leq x\}) \tag{B.21}$$

B.3. Probability distributions

Like we did with events, we want to assign a probability or likelihood[2] to the values of a random variable. Thus, we want to know the probability/likelihood that a random variable takes on a particular value. We can do so by defining a *probability distribution* for a random variable.

In the next two sections we introduce common probability distributions for discrete and continuous random variables - we have to distinguish between probability distributions for discrete random variables and probability distributions for continuous random variables.

B.3.1. Discrete distributions

The probability distribution of a discrete random variable is fully specified by a *probability mass function* (short *pmf*). A probability mass functions f_X maps the image of a random variable X to the probability that the random variable takes on a specific value.

$$\begin{aligned} &f_X : \mathrm{Img}(X) \mapsto [0,1] \\ &f_X(x) = P(X = x) = P\big(\{\omega \mid X(\omega) = x\}\big) \end{aligned} \tag{B.22}$$

Furthermore, we require that the sum over the probabilities of all possible values is equal to one

$$\sum_{x \in \mathrm{Img}(X)} f_X(x) = 1 \tag{B.23}$$

The cumulative distribution function is given by

$$F_X(x) = P(X \leq x) = \sum_{i : x_i \leq x} f_X(x_i) \tag{B.24}$$

Because of $f_X(x) = P(X = x)$, we can work with probability mass functions like we did with events.

[2] we will define the term *likelihood* when talking about continuous probability distributions in section B.3.2

B.3.1.1 Bernoulli distribution

The probability mass function of the *Bernoulli distribution* is defined as

$$\text{Ber}(x \mid p) = p^x(1-p)^{1-x} \tag{B.25}$$

where $p \in [0, 1]$ and $x \in \{0, 1\}$.

Note: The Bernoulli distribution is suited for binary random variables - random variables that can take on two different values only.

B.3.1.2 Binomial distribution

The probability mass function of the *Binomial distribution* is defined as

$$\text{Bin}(x \mid n, p) = \binom{n}{x} p^x(1-p)^{n-p} \tag{B.26}$$

where $\binom{n}{x} = \frac{n!}{x!(n-x)!}$.

Note: The Binomial distribution describes the probability of x successes in a sequence of n independent Bernoulli experiments where p is always the same probability parameter in each Bernoulli experiment.

B.3.1.3 Poisson distribution

The probability mass function of the *Poisson distribution* is defined as

$$f(x \mid \lambda) = \frac{\lambda^x \exp(-x)}{x!} \tag{B.27}$$

where $x \in \mathbb{N}$ and $\lambda > 0$.

Note: The Poisson distribution is suited for random variables whose image is the natural numbers.

B.3.2. Continuous distributions

The probability distribution of a continuous random variable is fully specified by a *probability density function* (short *pdf*). A probability density function p_X maps the image of a random variable X to the likelihood[3].

$$p_X : \text{Img}(X) \mapsto \mathbb{R}_+ \tag{B.28}$$

[3]Note that, from a mathematical point of view, likelihood and probability are not always the same - although they are often used interchangeably!

The probability that the random variable takes on a value from the interval $[a, b]$ is defined as

$$P(a \leq X = x \leq b) = \int_a^b p_X(x)\mathrm{d}x \tag{B.29}$$

Note that the probability that the random variable takes on a specific value is zero!

$$P(X = x) = 0 \tag{B.30}$$

Similar to the discrete distributions, we require that the probability density function sums up to one

$$\int p_X(x)\mathrm{d}x = 1 \tag{B.31}$$

The cumulative distribution function is given by

$$F_X(x) = P(X \leq x) = \int_{-\infty}^{x} p_X(x)\mathrm{d}x \tag{B.32}$$

The conditional probability density function of two continuous random variables X and Y is given by

$$p_{X|Y=y}(x) = \frac{p_{X,Y}(x, y)}{p_Y(y)} \tag{B.33}$$

where $p_{X,Y} : \mathrm{Img}(X) \times \mathrm{Img}(Y) \mapsto \mathbb{R}_+$ denotes the joint probability density of X and Y. Note that this is the density version of Bayes' rule B.14.
By factoring the joint density $p_{X,Y}$, we can rewrite B.33 as

$$p_{X|Y=y}(x) = \frac{p_{Y,X=x}(y)p_X(x)}{p_Y(y)} \tag{B.34}$$

B.3.2.1 Normal distribution

The *Normal* distribution (also called *Gaussian distribution*) comes in an univariate and a multivariate version. The univariate case is for one dimensional continuous random variables and the multivariate version is for continuous random vectors.

B.3.2.1.1 Univariate The probability density function of the *univariate Normal distribution* is defined as

$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{B.35}$$

where $x, \mu \in \mathbb{R}$ and $\sigma^2 > 0$.
See Fig. B.1 for a plot of univariate Normal distributions with different values of σ^2.

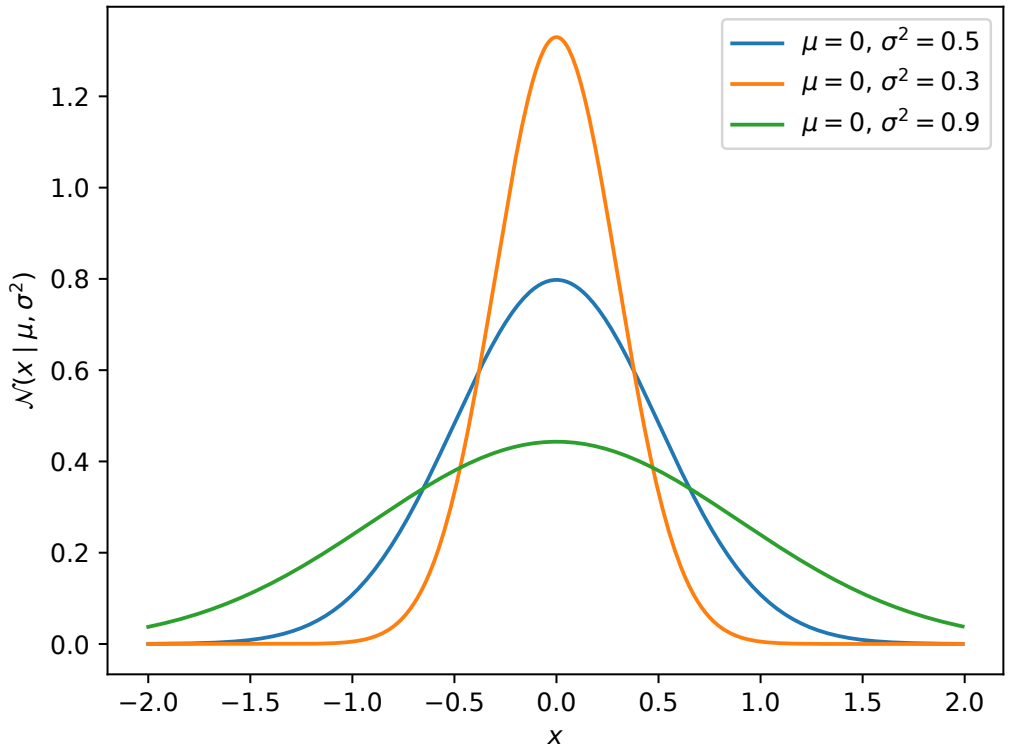


Figure B.1: Different Normal distributions

B.3.2.1.2 Multivariate The probability density function of the *multivariate Normal distribution* is defined as

$$\mathcal{N}(\vec{x} \mid \vec{\mu}, \mathbf{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma})}} \exp\left(-\frac{1}{2}(\vec{x}-\vec{\mu})^\top \mathbf{\Sigma}^{-1}(\vec{x}-\vec{\mu})\right) \tag{B.36}$$

where $\vec{x}, \vec{\mu} \in \mathbb{R}^d$ and $\mathbf{\Sigma} \in \mathbb{R}^{d \times d}$.
See Fig. B.3.2.1.2 for a plot of a 2d Normal distribution with $\vec{\mu} = (0, 0)^\top$ and $\mathbf{\Sigma} = \begin{pmatrix} 1.0 & 0.3 \\ 0.3 & 0.5 \end{pmatrix}$.

Figure B.2: Multivariate Normal distribution

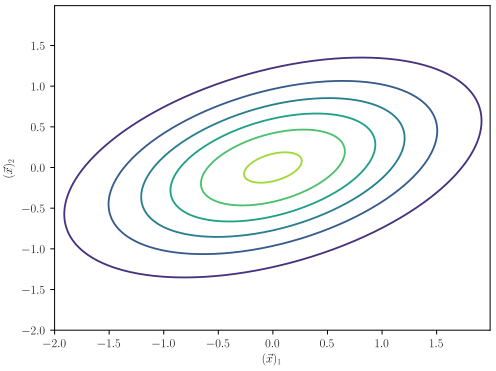


Figure B.3: Level curve of the density of a 2d Normal distribution

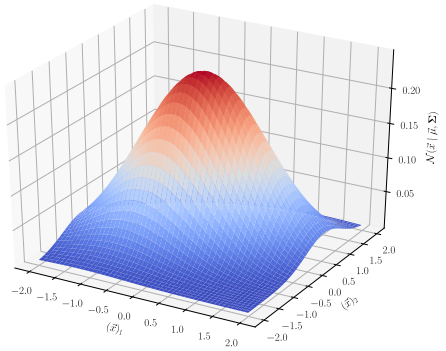


Figure B.4: Density of a 2d Normal distribution

B.3.2.2 Laplace distribution

The probability density function of the *Laplace distribution* is defined as

$$\text{Lap}(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right) \tag{B.37}$$

where $x, \mu \in \mathbb{R}$ and $b > 0$.
See Fig. B.5 for a plot of Laplace distributions with different values of b.

B.3.2.3 Beta distribution

The probability density function of the *Beta distribution* is defined as

$$\text{Beta}(x \mid \alpha, \beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha + \beta)} x^{\alpha - 1}(1 - x)^{\beta - 1} \tag{B.38}$$

where $x \in [0, 1]$ and $\alpha, \beta > 0$.
The gamma function[4] is defined as

$$\Gamma(a) = \begin{cases} (a - 1)! & \text{if } a \in \mathbb{N} \\ \int_0^\infty x^{a-1} \exp(-x) \mathrm{d}x & \text{otherwise} \end{cases} \tag{B.39}$$

[4] a generalization/interpolation of the factorial function

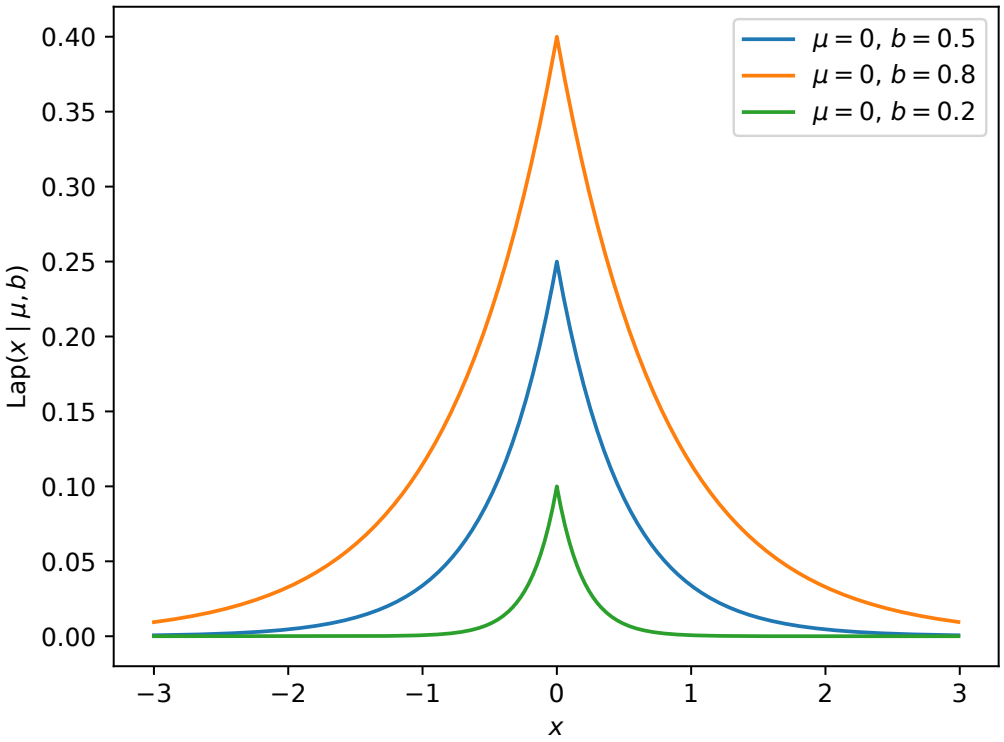


Figure B.5: Different Laplace distributions

Note: The Beta distribution is suited for random variables whose image is the interval $[0, 1]$.

B.3.2.4 Exponential distribution

The probability density function of the *Exponential distribution* is defined as

$$p(x \mid \lambda) = \lambda \exp(-\lambda x) \tag{B.40}$$

where $x \in \mathbb{R}_+$ and $\lambda > 0$.

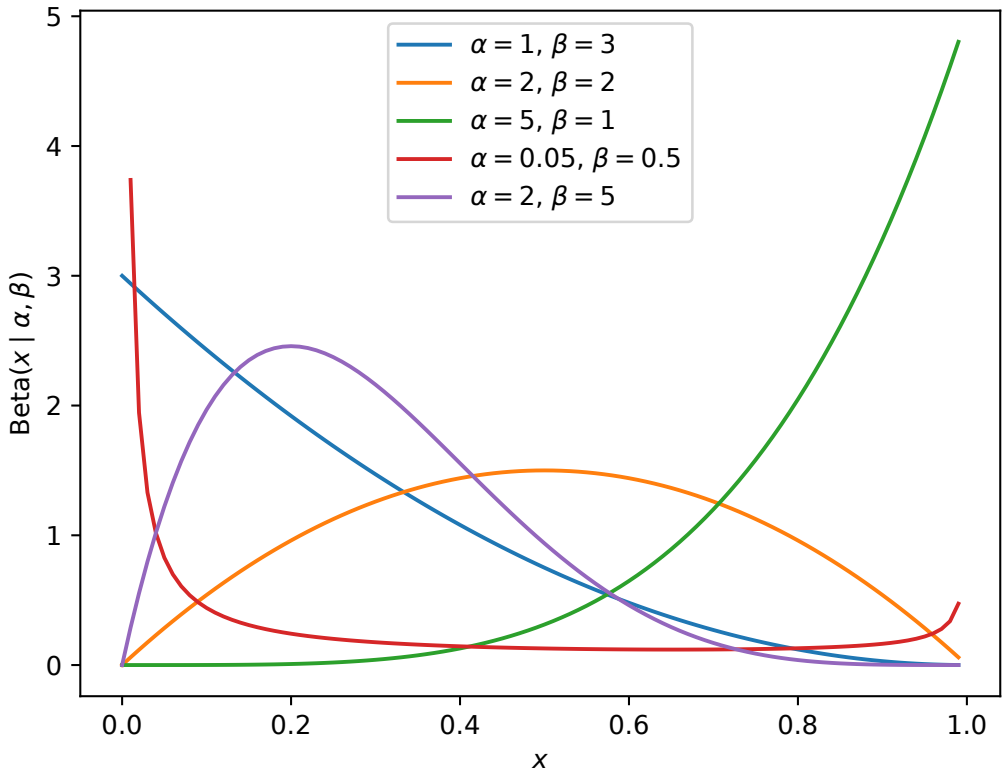


Figure B.6: Different Beta distributions

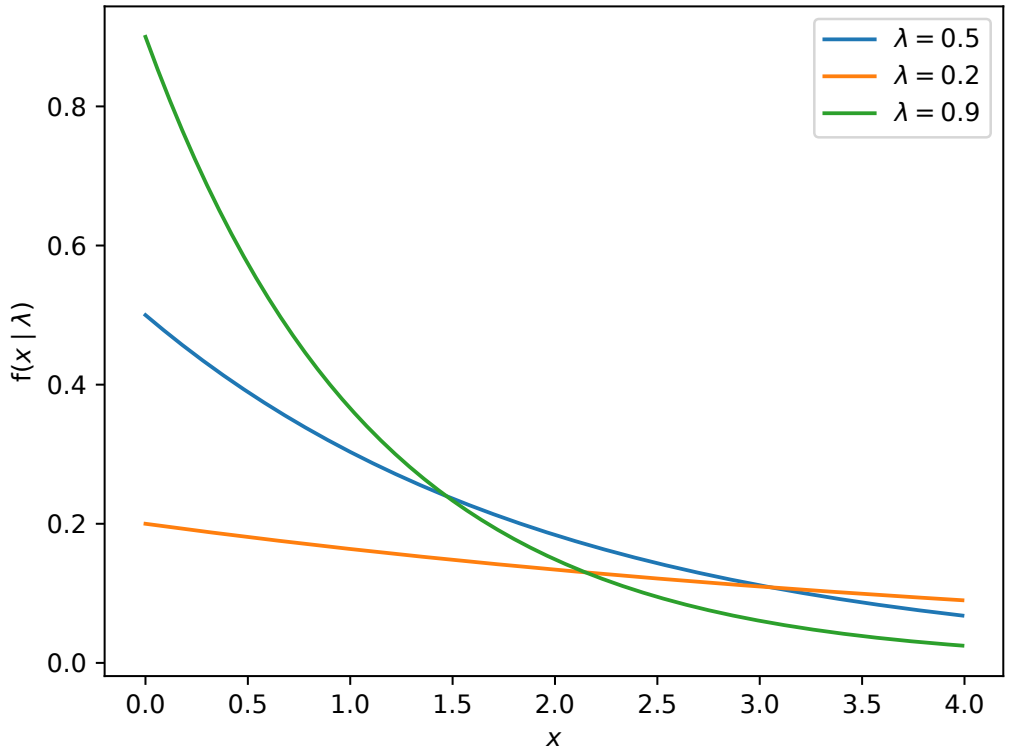


Figure B.7: Different Exponential distributions

B.4. Expectation

B.4.1. Expected value

The *expected value* of a discrete random variable is defined as

$$\mathbb{E}[X] = \sum_{x_i \in \text{Img}(X)} x_i p_i \tag{B.41}$$

where $p_i = P(X = x_i)$ and P is the corresponding probability mass function of X.
Similar, we define the expected value of a continuous random variable as

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x p_X(x) \mathrm{d}x \tag{B.42}$$

where $p_X(x)$ is the corresponding probability density function of X.

We can interpret the expected value $\mathbb{E}[X]$[5] as the "mean/average" of a probability distribution.
Other common notations for $\mathbb{E}[X]$ are μ or μ_X.

It follows that:

1. The expectation of a constant $c \in \mathbb{R}$ is the constant itself.

$$\mathbb{E}[c] = c \tag{B.43}$$

2. The expectation of the indicator function that an element x is in an event $\mathcal{A}$, is the probability of event $\mathcal{A}$.

$$\mathbb{E}\big[\mathbb{1}(x \in \mathcal{A})\big] = P(\mathcal{A}) \tag{B.44}$$

3. Because the expectation is linear, we can write the expectation of the sum of two random variables X and Y as

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y] \tag{B.45}$$

4. For any $\alpha, \beta \in \mathbb{R}$ we have that

$$\mathbb{E}[\alpha X + \beta] = \alpha \mathbb{E}[X] + \beta \tag{B.46}$$

[5] for the rest of this chapter, we assume that the expected value of a random variable exists and is finite - otherwise we do not work with it!

B.4.2. Variance

We define the *variance* of a random variable X as the expected difference of the random variable from its expected value.

$$\begin{aligned} \mathrm{Var}[X] &= \mathbb{E}[X - \mathbb{E}[X]] \\ &= \mathbb{E}[X^2] - \mathbb{E}[X]^2 \end{aligned} \tag{B.47}$$

We can interpret the variance as the "spread" of of a probability distribution. Sometimes the variance of a random variable X is denoted by σ_X^2.

The *standard deviation* σ_X of a random variable X is defined as the square root of the variance

$$\sigma_X = \sqrt{\mathrm{Var}[X]} \tag{B.48}$$

The variance of a random variable is invariant to translation. Thus, for any $\alpha \in \mathbb{R}$

$$\mathrm{Var}[X + \alpha] = \mathrm{Var}[X] \tag{B.49}$$

Scaling a random variables by a factor of $\alpha \in \mathbb{R}$ results in

$$\mathrm{Var}[\alpha X] = \alpha^2 \,\mathrm{Var}[X] \tag{B.50}$$

B.4.3. Covariance

The *covariance* of two random variables X and Y is defined as

$$\begin{aligned} \mathrm{Cov}[X, Y] &= \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] \\ &= \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] \end{aligned} \tag{B.51}$$

and measure the *linear relationship* between the two random variables.
The covariance of a random variables with itself is equal to the variance of the random variable

$$\mathrm{Cov}[X, X] = \mathrm{Var}[X] \tag{B.52}$$

The *correlation* of two random variables X and Y is defined as the normalized covariance

$$\mathrm{Corr}[X, Y] = \frac{\mathrm{Cov}[X, Y]}{\sigma_X \sigma_Y} \tag{B.53}$$

The variance of the sum of two random variables X and Y decomposes into

$$\mathrm{Var}[X + Y] = \mathrm{Var}[X] + \mathrm{Var}[Y] + 2\,\mathrm{Cov}[X, Y] \tag{B.54}$$

B.4.4. Transformation

Let $g : \mathbb{R} \mapsto \mathbb{R}$ be any function and X be a discrete random variable. The expected value of the the transformed random variable $g(X)$ is given by

$$\mathbb{E}[g(X)] = \sum_i g(x_i)P(x_i) \tag{B.55}$$

If X is a continuous random variable

$$\mathbb{E}[g(X)] = \int g(x)p_X(x)\mathrm{d}x \tag{B.56}$$

B.4.5. Conditional expectation

We can condition a random variable X on an event $\mathcal{B}$ by restricting the values that X can take on to be in the set $\mathcal{B}$. Therefore, the conditional expected value of X conditioned on the event $\mathcal{B}$ is given by

$$\mathbb{E}[X \mid \mathcal{B}] = \sum_{x \in \mathcal{B}} x\, f_{X|\mathcal{B}}(x) \tag{B.57}$$

where X is a discrete random variable. If X is a continuous random variable, we have

$$\mathbb{E}[X \mid \mathcal{B}] = \int_{x \in \mathcal{B}} x\, p_{X|\mathcal{B}}(x)\mathrm{d}x \tag{B.58}$$

where $p_{X|\mathcal{B}}$ denotes the probability density function of X conditioned on $\mathcal{B}$.

Similar to the law of total probability for events, we have a similar rule for random variables conditioned on events

$$\begin{aligned} &\mathbb{E}[X] = \sum_i \mathbb{E}[X \mid \mathcal{B}_i]P(\mathcal{B}_i) \\ &\text{where} \\ &\bigcup_i \mathcal{B}_i = \Omega \\ &\mathcal{B}_i \cap \mathcal{B}_j = \emptyset \; \forall \, i, j \end{aligned} \tag{B.59}$$

For two random variables X and Y the *law of total expectation*[6] is given by

$$\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X \mid Y]] \tag{B.60}$$

[6] also called *tower rule* or *law of iterated expectation*

B.5. Independence

Similar to events, we can define *independence* of random variables.
Two discrete random variables X and Y are called independent if and only if

$$P(X = x, Y = y) = P(X = x)P(Y = y) \tag{B.61}$$

Equivalently

$$P(X = x \mid Y = y) = P(X = x) \tag{B.62}$$

Two continuous random variables X and Y are called independent if and only if

$$p_{X,Y}(x, y) = p_X(x)p_Y(y) \tag{B.63}$$

Equivalently

$$p_{X|Y}(x, y) = p_X(x) \tag{B.64}$$

It follows, that we can write the cumulative distribution function of two independent random variables X and Y as

$$F_{X,Y}(x, y) = P(X \leq x, Y \leq y) = P(X \leq x)P(Y \leq y) = F_X(x)F_Y(y) \tag{B.65}$$

where X and Y are independent random variables.

Furthermore, if two random variables X and Y are independent, it holds that

$$\text{Cov}[X, Y] = 0 \tag{B.66}$$

Note that the reverse is not true - if the covariance of two random variables is zero, they are not necessarily independent.

We say that a set of random variables $\{X_1, ..., X_n\}$ are *i.i.d.* if the random variables are *independent, identically distributed* - they are pair-wise independent and have the same probability distribution.

B.6. Moments

In probability theory, *moments* measure the "shape" of a probability distribution. A probability distribution is completely defined by its moments (see section B.6.1).

The *k-th moment* of a random variable X, which is associated with a probability distribution, is defined as

$$m_k = \mathbb{E}[X^k] \tag{B.67}$$

The *k-th* central moment of a random variable X is defined as

$$\mathbb{E}[(X - \mathbb{E}[X])^k] \tag{B.68}$$

The *k-th standardized moment* of a random variable X is defined as

$$\frac{\mathbb{E}[(X - \mathbb{E}[X])^k]}{\sqrt{\mathbb{E}[(X - \mathbb{E}[X])^2]}^k} \tag{B.69}$$

Moments are used to compute some characteristics of a probability distribution.
The first moment is equal to the mean of the distribution.

$$\mu_X = \mathbb{E}[X] = \mathbb{E}[X^1] \tag{B.70}$$

The second central moment is equal to the variance of the distribution.

$$\mathrm{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2] \tag{B.71}$$

The third standardized moment is used as a measure of *skewness*[7] of a probability distribution. The skewness measures the asymmetry of a probability distribution.

$$\mathrm{Skewness}_X = \frac{\mathbb{E}[(X - \mathbb{E}[X])^3]}{\mathbb{E}[(X - \mathbb{E}[X])^2]^{\frac{3}{2}}} \tag{B.72}$$

The fourth standardized moment is used a measure of *kurtosis*[8] of a probability distribution. The kurtosis measures the "shape of the tails" of a probability distribution.

$$\mathrm{Kurtosis}_X = \frac{\mathbb{E}[(X - \mathbb{E}[X])^4]}{\mathbb{E}[(X - \mathbb{E}[X])^2]^2} \tag{B.73}$$

B.6.1. Moment-generating function

For a non-negative random variable X, we define the *moment-generating function* (short: *mgf*) as

$$M_X(t) = \mathbb{E}[\exp(tX)] \quad t \in \mathbb{R} \tag{B.74}$$

[7] see `https://en.wikipedia.org/wiki/Skewness`
[8] see `https://en.wikipedia.org/wiki/Kurtosis`

Note: The moment-generating function does not not always exist[9].

The k-th derivative of the moment-generating function M_X evaluated at zero gives us the k-th moment of the distribution of X.

$$\frac{\mathrm{d}^k M_X}{\mathrm{d}t^k}(0) = \mathbb{E}[X^k] \tag{B.75}$$

Note: Knowing the moment-generating function of a random variable is equivalent to knowing its probability distribution.

B.6.1.1 Example

The moment-generating function of a random variable X following normal distribution with mean μ and variance σ^2 is given by

$$M_X(t) = \exp(t\mu + \frac{\sigma^2 t^2}{2}) \tag{B.76}$$

We can derive the k-th moments by computing the k-th derivative of B.76 and evaluate it at $t = 0$. We obtain the first two moments of the normal distribution.

$$\begin{aligned} m_1 = \mathbb{E}[X^1] &= \frac{\mathrm{d}^1 M_X}{\mathrm{d}t^1}(0) = \mu \\ m_2 = \mathbb{E}[X^2] &= \frac{\mathrm{d}^2 M_X}{\mathrm{d}t^2}(0) = \mu^2 + \sigma^2 \end{aligned} \tag{B.77}$$

A comprehensive list of probability densities and their corresponding moment-generating functions can be found on wikipedia[10].

B.7. Upper bounds

B.7.1. Jensen's inequality

For any convex function $g : \mathbb{R} \mapsto \mathbb{R}$ and random variable X, *Jensen's inequality* states that

$$g(\mathbb{E}[X]) \leq \mathbb{E}[g(X)] \tag{B.78}$$

[9] The expectation $\mathbb{E}[\exp(tX)]$ might not exist for all non-negative random variables.

[10] `https://en.wikipedia.org/wiki/Moment-generating_function`

B.7.2. Chebyshev's inequality

For any random variable X with expected value $\mathbb{E}[X] = \mu$ and variance $\mathrm{Var}[X] = \sigma^2$, *Chebyshev's inequality* states that

$$P(|X - \mu| \geq t\sigma) \leq \frac{1}{t^2} \quad \forall\, t > 0 \tag{B.79}$$

In plain English: The probability that the absolute difference between a random variable and its mean is greater than t standard deviations can be bounded above by $\frac{1}{t^2}$.

B.7.3. Markov's inequality

For any non-negative random variable X, *Markov's inequality* states that

$$P(X > t) \leq \frac{\mathbb{E}[X]}{t} \quad \forall\, t > 0 \tag{B.80}$$

B.7.4. Chernoff bound

For any non-negative random variable X, the *Chernoff bound* states that

$$P(X > t) \leq \frac{M_X(t)}{\exp(ta)} \tag{B.81}$$

where $M_X(t) = \mathbb{E}[\exp(tX)]$ is the moment-generating function of X.

B.7.5. Hoeffding's inequality

For any sequence of n i.i.d.[11] and bounded random variables $X_i \in [a, b]$ with expected value $\mathbb{E}[X_i] = \mu$, *Hoeffding's inequality* states that

$$P(|\bar{X}_n - \mu| \geq \epsilon) \leq 2\exp\left(-\frac{2n\epsilon^2}{(b-a)^2}\right) \quad \forall\, \epsilon > 0$$

where

$$\bar{X}_n = \frac{1}{n}\sum_i X_i \tag{B.82}$$

In plain English: The probability that the mean of n i.i.d. bounded random variables deviates more than epsilon from the distribution's mean, can be bounded above by an exponential function in $-n$ - goes exponentially to zero. Note that this bound is much sharper than the bound from Markov's inequality.

[11] there exists a slightly more general version of Hoeffding's inequality that only needs independence but not the assumption of identical distribution

B.7.6. Cauchy–Schwarz inequality

For any pair of random variables X and Y, the *Cauchy-Schwarz inequality* states that

$$|\mathbb{E}[XY]| \leq \sqrt{\mathbb{E}[X^2]\mathbb{E}[Y^2]} \tag{B.83}$$

it follows that

$$\mathrm{Cov}[X,Y]^2 \leq \mathrm{Var}[X]\,\mathrm{Var}[Y] \tag{B.84}$$

B.7.7. Union bound

For any set of events $\mathcal{A}_i$, the *Union bound*[12] states that

$$P\left(\bigcup_i \mathcal{A}_i\right) \leq \sum_i P(\mathcal{A}_i) \tag{B.85}$$

In plain English: The probability that at least one event in a set of events occurs is always less or equal than the sum of the probabilities of each individual event.

B.8. Law of large numbers

Given a set of i.i.d. random variables X_i with mean $\mathbb{E}[X_i]$, then the empirical mean is equal to the true mean

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} X_i = \mathbb{E}[X] \tag{B.86}$$

Sometimes the law of large numbers B.86 is stated as

$$\lim_{n\to\infty} P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mathbb{E}[X]\right| < \epsilon\right) = 1 \quad \forall\,\epsilon > 0 \tag{B.87}$$

or equivalent

$$\lim_{n\to\infty} P\left(\left|\frac{1}{n}\sum_{i=1}^{n} X_i - \mathbb{E}[X]\right| > \epsilon\right) = 0 \quad \forall\,\epsilon > 0 \tag{B.88}$$

The type of convergence in B.87 and B.88 is called *convergence in probability.*

[12] also called *Boole's inequality*

Eq.B.87 is called the *weak law of large numbers*. There exists a stronger version called the *strong law of large numbers* stated as

$$P\left(\lim_{n\to\infty}\frac{1}{n}\sum_i X_i = \mathbb{E}[X]\right) = 1 \tag{B.89}$$

where the convergence in B.89 is called *almost surely convergence*.

Note: The law of large number is the reason why in practice people often replace the expectation by the average (empirical mean).

B.9. Central limit theorem

Given a set of i.i.d. random variables X_i with mean $\mathbb{E}[X_i] = \mu$ and variance $\mathrm{Var}[X_i] = \sigma^2$, we define a new random variable Z_n as

$$\begin{aligned} Z_n &= \frac{S_n - n\mu}{\sqrt{n}\sigma} \\ &\text{where} \\ S_n &= \frac{1}{n}\sum_i X_i \end{aligned} \tag{B.90}$$

The *central limit theorem* states that in the limit ($n \to \infty$ - if we average infinitely many random variables) Z_n and S_n follow a normal distribution

$$\begin{aligned} \lim_{n\to\infty} Z_n &\sim \mathcal{N}(0,1) \\ \lim_{n\to\infty} S_n &\sim \mathcal{N}(\mu, \frac{\sigma^2}{n}) \end{aligned} \tag{B.91}$$

The type of convergence in B.91 is called *convergence in distribution*.
Some times B.91 is rewritten in terms of the cumulative distribution. Two random variables are equal (in the sense that they have the same probability distribution) if their cumulative distributions are equal

$$\lim_{n\to\infty} \widehat{F}_{Z_n}(x) = F_{N(0,1)}(x) \quad \forall\, x \tag{B.92}$$

where F_{Z_n} is the *empirical cumulative distribution* function of Z_n.
The empirical distribution function of a set of i.i.d. observations (random variables) X_i is defined as

$$\widehat{F}_n(x) = \frac{1}{n}\sum_i \mathbb{1}(X_i \le x) \quad \forall\, x \tag{B.93}$$

Note: The central limit theorem is the reason why we can use the Normal distribution to approximate the binomial distribution (if n is sufficiently large) and the Poisson distribution (if λ is sufficiently large).

B.10. Information theory

Information theory is about quantifying information in communication channels.
In the following subsections, we introduce important concepts from information theory which are often used/needed in machine learning.

B.10.1. Entropy

We define the *entropy* $\mathrm{H}(X)$ of a discrete random variables X as

$$\begin{aligned}\mathrm{H}(X) &= -\mathbb{E}[\log(P(X))] \\ &= -\sum_i P(x_i)\log(P(x_i)) \\ &= \sum_i P(x_i)\log\left(\frac{1}{P(x_i)}\right)\end{aligned} \tag{B.94}$$

where $P(X)$ denotes the probability mass function of X.
Similar to B.94, we define the entropy $\mathrm{H}(X)$ of a continuous random variable X as

$$\mathrm{H}(X) = -\mathbb{E}[\log(p_X(X))] \tag{B.95}$$

where p_X denotes the probability density function of X.

B.10.2. Kullback-Leibler divergence

The *Kullback-Leibler divergence* (short *KL-divergence*) $\mathrm{D}_{\mathrm{KL}}(p_X \,\|\, p_Y)$ of two continuous random variables X and Y is defined as

$$\mathrm{D}_{\mathrm{KL}}(p_X \,\|\, p_Y) = \int p_X(x)\log\left(\frac{p_X(x)}{p_Y(x)}\right)\mathrm{d}x \tag{B.96}$$

where p_X and p_Y denotes the corresponding probability density functions.
For two discrete random variables X and Y, the Kullback-Leibler divergence

is defined as

$$\begin{aligned} \mathrm{D_{KL}}(P_X \parallel P_Y) &= \sum_i P_X(x_i) \log\left(\frac{P_X(x_i)}{P_Y(x_i)}\right) \\ &= -\sum_i P_X(x_i) \log\left(\frac{P_Y(x_i)}{P_X(x_i)}\right) \\ &= -\sum_i P_X(x_i) \log\left(P_Y(x_i)\right) + \sum_i P_X(x_i) \log\left(P_X(x_i)\right) \end{aligned} \tag{B.97}$$

Note that we require equal images for both random variables.

The Kullback-Leibler divergence is sometimes used a measure for the similarity of two probability distributions. However, it is not a distance metric because it is not symmetric. We know that

$$\begin{aligned} &\mathrm{D_{KL}}(P_X \parallel P_Y) \geq 0 \quad \forall\, P_X, P_Y \\ &\mathrm{D_{KL}}(P_X \parallel P_Y) = 0 \quad \Leftrightarrow\ P_X = P_Y \\ &\mathrm{D_{KL}}(P_X \parallel P_Y) \neq \mathrm{D_{KL}}(P_Y \parallel P_X) \end{aligned} \tag{B.98}$$

B.10.3. Mutual information

The *mutual information* of two discrete[13] random variables X and Y is defined as

$$\mathrm{I}(X,Y) = \sum_{y_i} \sum_{x_i} P_{X,Y}(x_i, y_i) \log\left(\frac{P_{X,Y}(x_i, y_i)}{P_X(x_i) P_Y(y_i)}\right) \tag{B.99}$$

The mutual information of two random variables is always non-negative and measures the dependency/correlation between the two random variables. The higher the mutual information is, the more dependent we expect them to be. The mutual information of two independent random variables is zero - thus knowing something about one of the variables does not tell us anything about the other variable.

Note that B.99 is equal to the KL-divergence of the joint probability $P_{X,Y}$ and the product $P_X P_Y$ of the individual probabilities.

$$\mathrm{I}(X,Y) = \mathrm{D_{KL}}(P_{X,Y} \parallel P_X P_Y) \tag{B.100}$$

[13] the analogue for continuous random variables can be obtained by replacing the sums by integrals

B.10.4. Cross entropy

We define the *cross entropy* of two random variables X and Y as

$$\mathrm{H}(p_X, p_Y) = \mathrm{H}(p_X) + \mathrm{D}_{\mathrm{KL}}(p_X \parallel p_Y) \tag{B.101}$$

where p_X and p_Y are either the probability density functions or the probability mass function of X and Y - X and Y are either both continuous or both discrete.

If we assume that X and Y are discrete random variables, we can rewrite/simplify B.101 by making use of B.95 and B.97

$$\begin{aligned}
\mathrm{H}(P_X \parallel P_Y) &= \mathrm{H}(P_X) + \mathrm{D}_{\mathrm{KL}}(P_X \parallel P_Y) \\
&= -\sum_i P_X(x_i) \log\left(P_X(x_i)\right) - \sum_i P_X(x_i) \log\left(P_Y(x_i)\right) + \\
&\quad \sum_i P_X(x_i) \log\left(P_X(x_i)\right) \\
&= -\sum_i P_X(x_i) \log\left(P_Y(x_i)\right)
\end{aligned} \tag{B.102}$$

B.11. Inference

Statistical inference is all about using data to infer properties/parameters of a statistical model. We assume that the data have been generated by a statistical model (e.g. a probability distribution which is defined by a set of parameters) and we want to use the data to infer some details (e.g. shape or parameters) of the underlying model.

B.11.1. Estimator

A *point estimator* (also called *statistic*) $\hat{\theta}_n$ of a parameter θ is *a random variable* that depends/is computed on n random variables $X_1, ..., X_n$.

$$\begin{aligned}
&\hat{\theta}_n = e(X_1, ..., X_n) \\
&\text{where} \\
&e : \mathrm{Img}(X)^n \mapsto \mathbb{R}^d
\end{aligned} \tag{B.103}$$

In plain English: A point estimator estimates a single value based on a set of samples.

We call an estimator $\hat{\theta}_n$ *consistent* if and only if it converges in probability to the true parameter θ

$$\lim_{n\to\infty} P(|\hat{\theta}_n - \theta| < \epsilon) = 1 \quad \forall\, \epsilon > 0 \tag{B.104}$$

We call an estimator $\hat{\theta}_n$ *unbiased* if and only if the difference of the true parameter θ and the expectation of the estimator is zero

$$\mathbb{E}[\hat{\theta}_n] - \theta = 0 \tag{B.105}$$

The term $\mathbb{E}[\hat{\theta}_n] - \theta$ is also called *bias*.

The *mean squared error* (short *mse*) of an estimator $\hat{\theta}_n$ is defined as

$$\mathrm{MSE}(\hat{\theta}_n, \theta) = \mathbb{E}[(\hat{\theta}_n - \theta)^2] \tag{B.106}$$

The mean squared error of an estimator can be decomposed as

$$\begin{aligned} \mathbb{E}[(\hat{\theta}_n - \theta)^2] &= \Big(\mathbb{E}[\hat{\theta}_n] - \theta\Big)^2 + \mathbb{E}\left[\Big(\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n]\Big)^2\right] \\ &= \mathrm{bias}(\hat{\theta}_n, \theta)^2 + \mathrm{Var}[\hat{\theta}_n] \end{aligned} \tag{B.107}$$

Note: The decomposition B.107 is also called *bias-variance decomposition*.

The *sample mean* $\hat{\mu}_n$ B.108 is an unbiased and consistent estimator of the mean $\mathbb{E}[X] = \mu$.

$$\hat{\mu}_n = \frac{1}{n}\sum_i X_i \tag{B.108}$$

where $\{X_i\}$ is a set of i.i.d. random variables.

The *sample variance* $\widehat{\sigma^2}_n$ B.109 is an unbiased and consistent estimator of the variance $\mathrm{Var}[X] = \sigma^2$.

$$\widehat{\sigma^2}_n = \frac{1}{n-1}\sum_i (X_i - \hat{\mu}_n)^2 \tag{B.109}$$

where $\{X_i\}$ is a set of i.i.d. random variables and $\hat{\mu}_n$ denotes the sample mean.
Sometimes B.109 is stated as

$$\widehat{\sigma^2}_n = \frac{1}{n}\sum_i (X_i - \mu)^2 \tag{B.110}$$

where we assume that we know the true mean $\mu = \mathbb{E}[X]$.
Note that the besides sample mean vs. mean, the only difference between B.109 and B.110 is the $\frac{1}{n-1}$ vs. $\frac{1}{n}$. The $\frac{1}{n-1}$ in B.109 is necessary to be an unbiased estimator. B.110 do not need the -1 in the denominator because instead of using an estimate of the mean it uses the true mean. Both estimators are unbiased and consistent.

The *sample covariance* $\widehat{\mathrm{Cov}}_n$ B.111 of two random variables X and Y is an unbiased and consistent estimator of the covariance.

$$\widehat{\mathrm{Cov}}_n = \frac{1}{n}\sum_i (X_i - \mu_x)(Y_i - \mu_y) \tag{B.111}$$

where we assume that we know the true means $\mu_X = \mathbb{E}[X]$ and $\mu_Y = \mathbb{E}[Y]$.

The *empirical cumulative distribution function* B.112 of a random variable X is a consistent estimator of the cumulative distribution function.

$$\hat{F}_n(x) = \frac{1}{n}\sum_i \mathbb{1}(X_i \leq x) \tag{B.112}$$

where $\{X_i\}$ is a set of i.i.d. random variables.

B.12. Bootstrapping

Assume that we are given data set $\mathcal{D} = \{x_i\}$ with $|\mathcal{D}| = n$ and an estimator $\hat{\theta}_n$ that computes an estimate of some quantity based on n samples.
Bootstrapping is a method for estimating the sample distribution of an estimator, where *sample distribution* denotes the distribution of an estimator. We do so by sampling from the distribution of an estimator and using these samples to estimate the quantity of interest (e.g. the mean or variance of the distribution).

We can compute a sample from the distribution of an estimator by first computing a *bootstrapped dataset* from the original dataset.
A bootstrapped data set $\mathcal{D}_B$ of an original data set $\mathcal{D}$ is obtained by randomly sampling n points from $\mathcal{D}$ with replacement.
Next, we obtain a sample from the distribution of $\hat{\theta}_n$ by computing $\hat{\theta}_n(\mathcal{D}_B)$ - that is computing the estimator based on the bootstrapped data set.

B.13. Constructing estimators

There exist many different methods/strategies/approaches for constructing estimators.
In the next subsections we will discuss the method of moments B.13.1, maximum likelihood B.13.2 and maximimum a posteriori B.13.3. The latter two are very popular and frequently used in machine learning.

B.13.1. Method of moments

We are given n i.i.d. samples $\{x_1, ..., x_n\}$ of a random variable X whose probability density function is parameterized by $\vec{\theta} = (\theta_1, ..., \theta_k)^\top$. We want to estimate $\vec{\theta}$ based on the n samples.

We estimate the k-th moment of X as

$$\hat{m}_k = \frac{1}{n} \sum_i x_i^k \tag{B.113}$$

The *method of moments* constructs a system of k equations with k unknowns by setting each of the k moment estimates equal to its theoretical moment $m_k(\vec{\theta}) = \mathbb{E}[X^k]$[14].

$$\begin{aligned} \hat{m}_1 &= m(\vec{\theta})_1 \\ \vdots &= \vdots \\ \hat{m}_k &= m_k(\vec{\theta}) \end{aligned} \tag{B.114}$$

We can construct an estimate of $\vec{\theta}$ by solving B.114 for $\vec{\theta}$.

B.13.1.1 Example

If we have n i.i.d samples $\{x_1, ..., x_n\}$ of a normal distribution with mean μ and variance σ^2 (thus $\vec{\theta} = (\mu, \sigma^2)^\top$), we can construct estimators of μ and σ by using the method of moments.
First, we setup the system of equations.

$$\begin{aligned} \hat{m}_1 &= m(\vec{\theta})_1 = \mu \\ \hat{m}_2 &= m(\vec{\theta})_2 = \mu^2 + \sigma^2 \end{aligned} \tag{B.115}$$

[14] the probability density function of X depends on $\vec{\theta}$, so does the moment

Next, we solve B.115 for μ and σ to obtain the estimators.

$$\begin{aligned} \hat{\mu} &= \hat{m}_1 = \frac{1}{n}\sum_i x_i \\ \widehat{\sigma^2} &= \hat{m}_2 - \hat{\mu}^2 = \frac{1}{n}\sum_i (x_i - \hat{\mu})^2 \end{aligned} \tag{B.116}$$

B.13.2. Maximum likelihood

The concept of *maximum likelihood* is a general principle for estimating parameters of a statistical model and occurs many times in machine learning.

Assume that we have a data set $\mathcal{D} = \{x_i\}$ with $|\mathcal{D}| = n$ and $x_i \in \mathcal{X}$.
Next, we assume that we have a statistical model (e.g. a probability density function) parameterized by a set θ, describing" our data $\mathcal{D}$. "Describing" means that, given the parameter of the model, we can compute the *"plausibility" of our data under the model.*
In plain English: How likely is it to observe our data $\mathcal{D}$ given a model parameter θ?

We define the *likelihood* $\mathrm{L}_{\mathcal{D}}$[15] as

$$\mathrm{L}_{\mathcal{D}}(\theta) = p_\theta(x_1, ..., x_n) \tag{B.117}$$

where p_θ denotes the statistical model with parameter θ.
If the data points are i.i.d., we can rewrite B.117 as

$$\mathrm{L}_{\mathcal{D}}(\theta) = \prod_i p_\theta(x_i) \tag{B.118}$$

Furthermore, we define the *log-likelihood* $\mathrm{LL}_{\mathcal{D}}$ to be

$$\mathrm{LL}_{\mathcal{D}}(\theta) = \log\Big(\mathrm{L}_{\mathcal{D}}(\theta)\Big) \tag{B.119}$$

Finally, we define the *negative-log-likelihood* $\mathrm{NLL}_{\mathcal{D}}$*)* as

$$\mathrm{NLL}_{\mathcal{D}}(\theta) = -\log\Big(\mathrm{L}_{\mathcal{D}}\theta)\Big) \tag{B.120}$$

[15]Note: Sometimes people denote the likelihood as $P(\mathcal{D} \mid \theta)$. However, we think that this notation is confusing because it tempt people to interpret the likelihood as "the probability of the data given the model parameter", **which is a wrong interpretation/statement!** See remark at the end of this section.

Sometimes we have a set of *input-output pairs* $\mathcal{D} = \{(x_i, y_i)\}$ instead of a set of single values $\mathcal{D} = \{x_i\}$. In this case we might not be interested (or do not know) in the joint probability of X_i and Y_i but rather in the conditional probability $Y_i \mid X_i$ - we assume that x_i and y_i are realizations/observations of the corresponding random variables X_i and Y_i. For instance in predictive modeling, we often only model the conditional distribution $p(Y \mid X)$.
We define the *conditional likelihood* for a set of i.i.d. pairs of random variables as

$$\mathrm{L}_{\mathcal{D}}(\theta) = \prod_i p_\theta(y_i \mid x_i) \tag{B.121}$$

Like we did with the likelihood B.117, we can define the log-likelihood and negative-log-likelihood using the conditional likelihood.
For the rest of this section, we will use the likelihood B.117 only, but we could do exactly the same things with the conditional likelihood B.121.

Next, our goal is to find a θ which maximizes the likelihood B.117.
In plain English: Find the parameter θ which makes our data most probable.
To be more formal, our goal is to solve the following *optimization problem* - also known as *maximum likelihood estimate (MLE)*:

$$\theta^* = \arg\max_{\theta \in \Theta} \mathrm{L}_{\mathcal{D}}(\theta) \tag{B.122}$$

Which is equivalent to

$$\theta^* = \arg\max_{\theta \in \Theta} \mathrm{LL}_{\mathcal{D}}(\theta) \tag{B.123}$$

and

$$\theta^* = \arg\min_{\theta \in \Theta} \mathrm{NLL}_{\mathcal{D}}(\theta) \tag{B.124}$$

Note: B.122 and B.123 are equivalent optimization problems because the log is a monotonically increasing function. B.123 and B.124 are equivalent problems because maximizing a function is equivalent to minimizing the negative of the function - thus $\arg\max_x f(x) = \arg\min_x -f(x)$.

Attention: The likelihood B.117 is a function of θ - the data $\mathcal{D}$ is fixed. From a frequentists point of view, θ is *not* a random variable. Furthermore, the likelihood in general is **neither** a probability **nor** a probability distribution. People often call it the "probability of the data given the parameter" - **this is wrong!** However, we will encounter a related interpretation in the section on Bayesian inference B.13.3.

B.13.2.1 Example

Assume that we have a data set

$$\mathcal{D} = \{x_i\} = \{0, 0, 0, 0, 1, 1, 1, 1, 1, 1\} \tag{B.125}$$

where $x_i \in \mathcal{X} = \{0, 1\}$.
We assume that the data x_i in $\mathcal{D}$ comes from a stochastic model and are i.i.d.. Thus, we introduce a random variable $X : \mathcal{X} \mapsto \mathcal{X}$ with $X(x) = x$. We assume that this random variable follows a *Bernoulli distribution* $x_i \sim \text{Ber}(p)$ with *unknown parameter* p.

We want to know the value of p which explains the given data set $\mathcal{D}$ the best. Thus, which p makes the observation of $\mathcal{D}$ most likely? We estimate p by using the maximum likelihood approach from the previous section.

$$p = \underset{p \in [0,1]}{\arg\max}\ \text{L}_{\mathcal{D}}(p) \tag{B.126}$$

Note: This equation looks very similar to B.122. The only difference is, that we substituted the model parameter θ by p and maximize over all possible values of p which is the interval $[0, 1]$ - since p is a probability.
By using the i.i.d. assumption and the Bernoulli distribution we can rewrite $\text{L}_{\mathcal{D}}(p)$ in B.126 as

$$\begin{aligned} \text{L}_{\mathcal{D}}(p) &= \prod_i \text{Ber}(x_i \mid p) \\ &= \prod_i p^{x_i}(1-p)^{1-x_i} \end{aligned} \tag{B.127}$$

Next, we plug in the specific values from the data set B.125 and obtain

$$\begin{aligned} \text{L}_{\mathcal{D}}(p) &= \prod_i p^{x_i}(1-p)^{1-x_i} \\ &= p^0(1-p)^{1-0}p^0(1-p)^{1-0}p^0(1-p)^{1-0}p^0(1-p)^{1-0} \\ &\quad p^1(1-p)^{1-1}p^1(1-p)^{1-1}p^1(1-p)^{1-1}p^1(1-p)^{1-1} \\ &= p^6(1-p)^4 \end{aligned} \tag{B.128}$$

Finally, we obtain the final optimization problem for estimating p

$$p = \underset{p \in [0,1]}{\arg\max}\ p^6(1-p)^4 \tag{B.129}$$

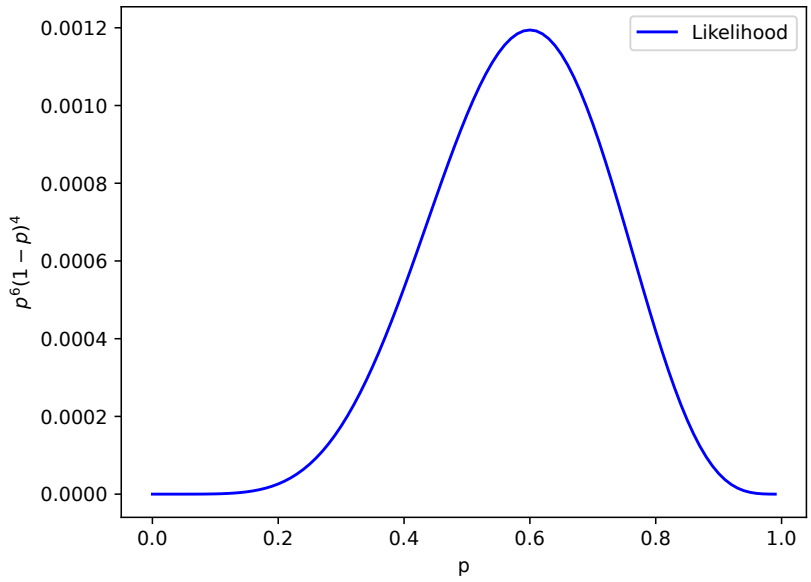


Figure B.8: Parameter vs. Likelihood

As we can see in Fig. B.8, the solution of B.129 is $p = 0.6$. Note that this is the example we worked on in the section on closed-form solutions (see section A.3.1.1).

In plain English: The maximum likelihood estimator of the unknown parameter p of the data set B.125 - under the modeling assumption - is $p = 0.6$.

B.13.3. Bayesian inference - Maximum a posteriori

So far, we always assumed that the true parameter θ is unknown but fixed. In *Bayesian inference* we assume that the parameter θ itself is *random*. That is, θ is a random variable. We denote the probability distribution of θ as $p(\theta)$. The choice of prior $p(\theta)$ reflects our believes/assumptions on θ - the prior expresses which values of θ we think/want are more likely than others. Next, we assume that the data set $\mathcal{D} = \{X_1, ..., X_n\}$ with $|\mathcal{D}| = n$ is a set of i.i.d. random variables $X_i \sim p(x \mid \theta)$ - note that the parameter θ of the probability distribution of X_i is a random variable.
We write the conditional distribution of the parameter θ given the data $\mathcal{D}$ as

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{\int p(\mathcal{D} \mid \theta)p(\theta)\mathrm{d}\theta} \tag{B.130}$$

If we fix the data set - we observed an outcome x_i of each random variable X_i, we can rewrite $p(\mathcal{D} \mid \theta)$ as

$$\begin{aligned} p(\mathcal{D} \mid \theta) &= p(x_1, ..., x_n \mid \theta) \\ &= \prod_i p(x_i \mid \theta) \\ &= \mathrm{L}_{\mathcal{D}}(\theta) \end{aligned} \tag{B.131}$$

Note that $p(\mathcal{D} \mid \theta)$ in B.131 is formally equivalent to the likelihood B.118. However, from a conceptual point of view, the likelihood in B.131 is **not** the same as in B.118. In the frequentist approach B.118 we assumed that θ is fixed, whether in the Bayesian approach B.131 we assume that θ is random. This is the subtle but crucial difference - do not mix Bayesian and frequentist views[16]!
Because θ itself is random, we can construct a probability distribution over it. This probability distribution is called *posterior* and is defined in B.130. By making use of B.131, we can rewrite this expression as

$$p(\theta \mid \mathcal{D}) = \frac{\mathrm{L}_{\mathcal{D}}(\theta)p(\theta)}{\int \mathrm{L}_{\mathcal{D}}(\theta)p(\theta)\mathrm{d}\theta} \tag{B.132}$$

The *normalizing constant* $\int \mathrm{L}_{\mathcal{D}}(\theta)p(\theta)\mathrm{d}\theta$ in the denominator of B.132 is necessary to turn the product of likelihood and prior into a valid probability distribution. If we ignore the normalizing constant in the denominator, we find that

$$p(\theta \mid \mathcal{D}) \propto \mathrm{L}_{\mathcal{D}}(\theta)p(\theta) \tag{B.133}$$

Now, we are ready for constructing estimators of θ using Bayesian inference.

The *maximum a posteriori estimator* (short *MAPE*) is defined as

$$\arg\max_{\theta \in \Theta} p(\theta \mid \mathcal{D}) \tag{B.134}$$

Because the normalizing constant is a constant, we can rewrite B.134 as

$$\arg\max_{\theta \in \Theta} \mathrm{L}_{\mathcal{D}}(\theta)p(\theta) \tag{B.135}$$

[16]If people call/interpret the likelihood "the probability of the data given the parameter", they 1. forgot the normalizing constant, 2. forgot the difference between density and probability, 3. might mixed up frequentist and Bayesian interpretations and/or they did not understand the difference between the two interpretations.
If you are ever debating with such people, feel free to refer them to these notes.

Because the normalizing constant is not needed and because it is often difficult or even impossible to compute, people often mean B.135 when talking about maximum a posteriori.

Another common estimator of θ is the *posterior mean estimator* which is defined as the expected value of the conditional distribution.

$$\theta = \mathbb{E}_{p(\theta|\mathcal{D})}[\theta] = \int \theta p(\theta \mid \mathcal{D}) \mathrm{d}\theta \tag{B.136}$$

The posterior mean estimator B.136 is optimal under the MSE risk B.106.

B.14. Outlook

In this chapter we took a look at the central concepts of basic probability theory and statistical inference, whereas we omitted many/all proofs and derivations.
We started with events and probabilities and went on to random variables. We introduced the most common probability distributions for random variables like the normal distribution. We talked about expectations, moments and the most important upper bounds like Jensen's and Hoeffding's inequality. Next, we introduced essential concepts (like entropy and Kullback-Leibler divergence) from information theory that are relevant in machine learning. Finally, we turned to statistical inference and discussed basic properties of estimators as well as some common methods for constructing estimators - in particular maximum likelihood and Bayesian inference.

If you want to learn more on probability theory and statistical inference, a good starting point is the excellent textbook "All of Statistics: A Concise Course in Statistical Inference (Springer Texts in Statistics)" [18] by Larry Wasserman (Professor at CMU).

B.15. Exercises

Bibliography

[1] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007, pages 1027–1035, 2007.

[2] Christopher M. Bishop. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] Stephen (Stanford University California) Boyd and Lieven (University of California Los Angeles) Vandenberghe. Convex Optimization. Camebridge University Press, 2004.

[4] Brian Caffo. Regression Models for Data Science in R. Leanpub, 2015.

[5] Brian Caffo. Advanced Linear Models for Data Science. Leanpub, 2017.

[6] P. Compeau and P. Pevzner. Bioinformatics Algorithms: An Active Learning Approach. Number Bd. 2 in Bioinformatics Algorithms: An Active Learning Approach. Active Learning Publishers, 2015.

[7] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. Statistical Learning with Sparsity: The Lasso and Generalizations. Chapman & Hall/CRC, 2015.

[8] Kevin P. Murphy. Machine Learning A Probabilistic Perspective. MIT Press, 2012.

[9] Jorge Nocedal and Stephen J. Wright. Numerical Optimization. Springer, New York, NY, USA, second edition, 2006.

[10] Shai Shalev-Shwartz (Hebrew University of Jerusalem) and Ontario) Shai Ben-David (University of Waterloo. Understanding Machine Learning From Theory to Algorithms. Camebridge University Press, 2014.

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[12] `https://ocw.mit.edu`. License: Creative Commons BY-NC-SA. Philippe Rigollet. 18.657 Mathematics of Machine Learning. Fall 2015. Massachusetts Institute of Technology: MIT OpenCourseWare.

[13] Bernhard Schölkopf, Alexander J. Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation, 10(5):1299–1319, 1998.

[14] John Shawe-Taylor and Nello Cristianini. Kernel Methods for Pattern Analysis. Cambridge University Press, New York, NY, USA, 2004.

[15] Robert Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B, 58:267–288, 1994.

[16] Jerome Friedman Trevor Hastie, Robert Tibshirani. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics, 2009.

[17] Ulrike von Luxburg. A tutorial on spectral clustering. Statistics and Computing, 17(4):395–416, 2007.

[18] Larry Wasserman. All of statistics : a concise course in statistical inference. Springer, New York, 2010.

[19] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society, Series B, 67:301–320, 2005.