Fachhochschule Südwestfalen
University of Applied Sciences
South Westphalia University of Applied Sciences
Campus Hagen, Germany
Department of Technical Economics (TBW)

Machine Learning

Master courses Informatics and Business & Wirtschaftsingenieurwesen

Lecture Notes

Andreas de Vries

Version: May 27, 2025

Contents

Preface

These lecture notes serve as the basis for the *Machine Learning* course of the Master's degree programs in Informatics and Business and *Wirtschaftsingenieurwesen* (Business Administration and Engineering) at University of Applied Sciences Südwestfalen, Campus Hagen. Machine learning as a term is hype. There is a veritable abundance of good literature, endlessly many textbooks — so why these lecture notes?

Well, exactly because there is so much good literature and sources. One of the main problems in designing this course was to collect the material in such a way that it could be covered in one semester. Another problem was to bring the many approaches from quite different schools and disciplines into a unified formalism. The latter sounds rather simple — but it is not.

A one-semester course on machine learning can only provide a first insight into the wide field. Nevertheless, it was the goal from the beginning to adequately convey this topic to the students, since it is currently essential in business and science and will become presumably increasingly more important in the future. First of all, to achieve this goal there is some theory to be taught to a sufficient degree. Long and tedious literature researches are not effective for an introduction; the theory should be presented as precisely and concisely as possible, but as much as necessary. In order to present the material, emphasis was also placed on a uniform formalism. The devil is in the details here, after all, the variable x may have a completely different meaning in the formalism commonly used in data analysis than it does in time series analysis.[1]

Therefore these lecture notes! As readers, judge at the end whether it has succeeded.

The contents. The course is basically divided into the following parts, with the first parts providing the methodological and theoretical tools that are to be applied in the last parts:

1. Introduction to Python
2. Statistical models
3. Data analysis (especially regression, principal component analysis, naive bayes classification)
4. Time series analysis

From my point of view, the role of statistical models cannot be overestimated, because they form the logical and formal brackets for all machine learning approaches, even those not mentioned here.

The form of teaching. The theory is quite extensive and is sketched in some lectures. The main part of the course, however, is strongly project-based teaching, an approach that has proven very successful for programming-related subjects in general.

[1] If, nevertheless, contradictions are observed in the formalism of this lecture notess, please let me know!

References. The literature supplementing this lecture notes includes introductions to the underlying mathematics and to the programming of the mathematical procedures in Python. We recommend for instance:

- For the mathematics part: Backhaus et al. (2016), Backhaus et al. (2015), Cowpertwait and Metcalfe (2009), Downey (2011), Handl and Kuhlenkasper (2017), Hastie et al. (2009), James et al. (2013), K. P. Murphy (2012), Palma (2016), Sen and Srivastava (1990), and Tabachnick and Fidell (2018);
- for the programming part VanderPlas (2018), as well as the API documentations of the Python libraries Scikit-Learn (`https://scikit-learn.org/`) and Statsmodels (`https://www.statsmodels.org/`);
- and for both parts Denis (2021) and Subasi (2020).

Hagen,
May 2025

Andreas de Vries

Part I

Foundations

1
Probability theory

Overview

This chapter devotes some time to designations and basic notions of probability theory which will be used in the sequel. In the best case it might be a simple refreshment of your knowledge about stochastics, but at least it clarifies the notations.

Historically, probability theory was initiated in 1654 by the French Salon theorist Antoine Gombaud, who called himself Chevalier de Méré, posing some questions about gambling which were then tackled by the two mathematicians Pierre Fermat and Blaise Pascal in a series of letters. Based on their results, Christiaan Huygens published the first systematic treatise on probability theory in 1657. About half a century later, Jakob Bernoulli continued Huygens's work and published a treatise mainly on combinatorics in 1713. Shortly after the French Revolution, in 1795, Pierre-Simon Laplace released the seminal textbook *Théorie analytique des probabilités*. The strict mathematical foundation of probability theory was challenged as the sixth out of the famous 23 problems by David Hilbert in 1900, and Kolmogorov developed the final axiomatic approach in his article "Grundbegriffe der Wahrscheinlichkeitsrechnung" in the journal *Ergebnisse der Mathematik* **2**, Heft 3 (1933) which in fact established probability theory as a special area of measure theory.[1]

1.1 Random variables and probabilities

The notion of *random variable* is central in probability theory. Sloppily said, a random variable X is a function assigning randomly an outcome from given a sample space to a real number x.[2] In other words, the values of a random variable X depend on outcomes ω

random variable X

[1] Bandelow (1989):§1.

[2] Bauer (1991):p. 14.

of a random phenomenon of a sample space Ω:

$$X : \Omega \to \mathbb{R}, \qquad \omega \ \mapsto \ X(\omega) = x \tag{1.1}$$

probability $P(X = x)$

Now, the probabilities of the outcomes in the sample space are inherited to the random variable, and we designate $P(X = x)$ for the probability that the random variable X takes on the value x. Therefore we have especially $0 \leqq P(X = x) \leqq 1$. In machine learning, the sample space usually is a finite set, and so is the image of the random variable. The values X can thus be represented by $\mathscr{X} = \{x_1, \ldots, x_N\}$, and the respective probabilities $p_1, \ldots, p_N$ are given by

$$p_i = P(X = x_i) \qquad \text{and satisfy} \qquad p_i \geqq 0, \qquad \sum_{i=1}^{N} p_i = 1. \tag{1.2}$$

brief notation $P(x)$, $P(A)$

We often will use briefer notations and write $P(x)$ or $P(A)$ instead of $P(X = x)$, where A means the statement "$X = x$", or equivalently A = "X takes on the value x". Sometimes we abbreviate even complex statements by a single capital letter such as A, B, ... Note that a statement can attain one of the values true or false.

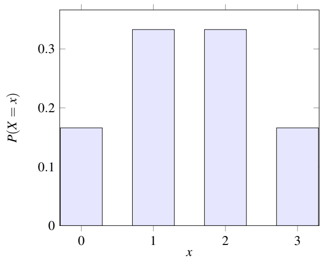
Example 1.1. *(Die roll)* A typical example of a random variable is the result of rolling a die: Here the sample space consists of 6 possible outcomes. If for instance X is the random variable assigning the result of a die roll ω modulo 4, we obtain the following table of values:

Number of pips ω	1	2	3	4	5	6
$X(\omega)$	1	2	3	0	1	2

(1.3)

Thus the probability distribution reads

x	0	1	2	3
$P(X = x)$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$



(How can we get these values? Cf. exercise 1.1.) □

probability of a subset

If $\mathscr{V}$ is a subset of the sample space, then

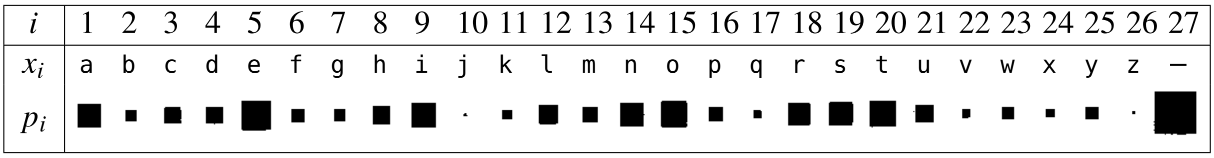
$$P(\mathscr{V}) = P(x \in \mathscr{V}) = \sum_{x \in \mathscr{V}} P(X = x). \tag{1.4}$$

If $\mathscr{V}$ is the total sample space, we have $P(\mathscr{V}) = 1$.

Example 1.2. *(Letters in a document)* Another example is a letter that is randomly selected from an English document. There are 27 letters: `a`–`z`, and a space character '–'. Graphically these probabilities are shown in Figure 1.1. Here the probabilities $p_i = P(x_i)$ are depicted by squares the size of which corresponds to the value of p_i. If we define $\mathscr{V}$ to be vowels from Figure 1.1, i.e., $\mathscr{V} = \{\texttt{a}, \texttt{e}, \texttt{i}, \texttt{o}, \texttt{u}\}$, then

$$\begin{aligned} P(\mathscr{V}) &= P(\texttt{a}) + P(\texttt{e}) + P(\texttt{i}) + P(\texttt{i}) + P(\texttt{o}) + P(\texttt{u}) \\ &= 0.058 + 0.091 + 0.060 + 0.069 + 0.033 = 0.311. \end{aligned} \tag{1.5}$$

(The probability values are taken from MacKay (2003:p. 22)) □



i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
x_i	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	–
p_i	■	■	■	■	■	■	■	■	■	·	■	■	■	■	■	■	■	■	■	■	■	■	■	■	■	·	■

Figure 1.1. Probability distribution over the letters in an English document. The size of a square refers to the probability $p_i = P(x_i)$. Modified from MacKay (2003:p. 22)

joint probability $P(x, y)$

Given *two* random variables X and Y, each outcome is an ordered pair (x, y) with $x \in \mathcal{X} = \{x_1, \dots, x_N\}$ and $y \in \mathcal{Y} = \{y_1, \dots, y_M\}$. We call $P(x, y)$ the *joint probability* of the outcomes of X and Y. The comma and the brackets here are optional when writing such ordered pairs, that is $xy \Leftrightarrow (x, y)$. Note that the two random variables X and Y need not be independent.

Example 1.3. An example of a joint probability is the ordered pair xy consisting of two successive letters ("bigrams") in an English document.[3] The possible outcomes are ordered pairs of letters such as `aa`, `ab`, `ac`, ..., `zz`. Of these, we might expect `ab` and `ac` to be more probably than `aa` and `zz`. An estimate of two neighboring letters is shown graphically in Figure 1.2. □

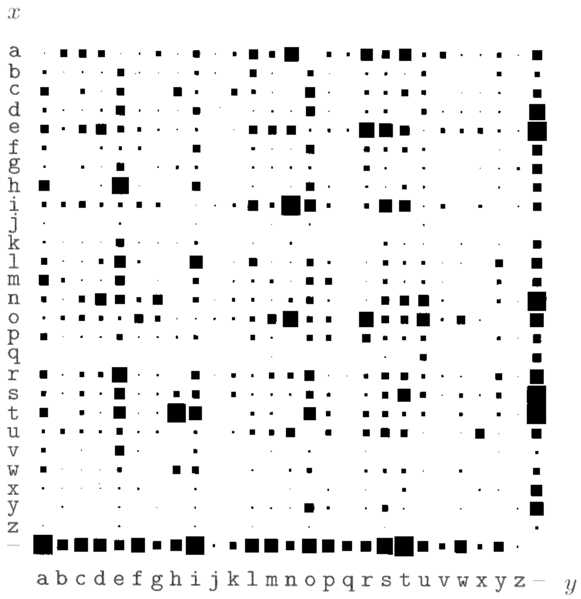


Figure 1.2. The joint probability distribution over the 27 × 27 possible bigrams xy in an English document. The size of the square (i, j) refers to the probability $p_{ij} = P(x_i, y_j)$. Modified from MacKay (2003:p. 23)

Given a joint probability $P(x, y)$ for sample spaces with finitely many values, we can obtain the *marginal probability* $P(x)$ from it by summation over all values of y:

marginal probability

$$P(x) = \sum_{y \in \mathcal{Y}} P(x, y) \tag{1.6}$$

Accordingly, the marginal probability $P(y)$ is achieved by summation over all values of x:

$$P(y) = \sum_{x \in \mathcal{X}} P(x, y) \tag{1.7}$$

[3] MacKay (2003):p. 23.

conditional probability $P(x \mid y)$

Given a joint probability $P(x, y)$, the *conditional probability* $P(x \mid y)$ is defined by the quotient of the joint probability $P(x, y)$ and the marginal probability $P(y)$,

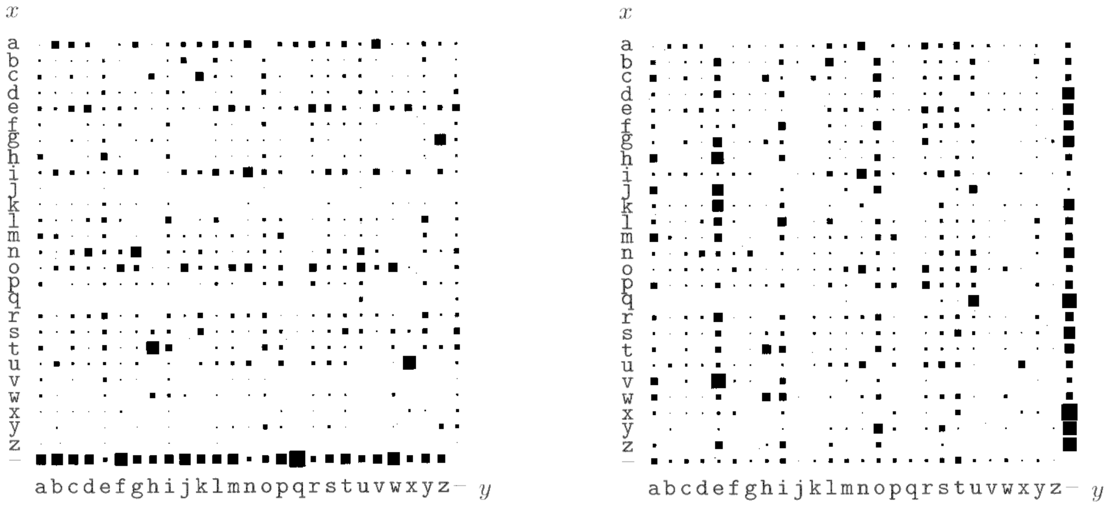
$$P(x \mid y) = \frac{P(x, y)}{P(y)} \tag{1.8}$$

We pronounce $P(x \mid y)$ "the probability that X equals x, given Y equals y", or even shorter "the probability of x, given y".

prior/posterior probability

The marginal probability $P(x)$ is often also called the *prior probability* of x, and the conditional probability $P(x \mid y)$ the *posterior probability* of x, given the data y.[4]

Example 1.4. *(Example 1.3 revisited)* From the joint probability distribution $P(x, y)$ in Example 1.3 and the probability distribution $P(x)$ in Example 1.2, the conditional probability distribution $P(x \mid y)$ can be determined. It is depicted in Figure 1.3 (a). For instance, $P(x \mid y = \mathsf{u})$ is the probability of the first letter x, given that the second letter is a u. As we can see in Figure 1.3 (a), the most probable values for x given $y = \mathsf{u}$ are o and n.



(a) $P(x \mid y)$ (b) $P(y \mid x)$

Figure 1.3. The conditional probability distributions (a) $P(x \mid y)$ and (b) $P(y \mid x)$ over the 27×27 possible bigrams xy in an English document. The size of the square (i, j) refers to the probability (a) $p_{ij} = P(x_i \mid y_j)$ or (b) $p_{ij} = P(y_i \mid x_j)$, rsp. Modified from MacKay (2003:p. 24)

Analogously, the probability $P(y \mid x = \mathsf{q})$ is the probability of the second letter y, given that the first letter is a q. As we can see in Figure 1.3 (b), the most probable values for y given $x = \mathsf{q}$ are u and $-$. □

1.2 Bayes' Theorem

The Bayes theorem is a direct mathematical consequence of the definition of the conditional probability.

Satz 1.5 (Bayes 1763). *Let be $P(y \mid \boldsymbol{x})$ be the conditional probability to observe the value y, having observed the values $\boldsymbol{x} = (x_1, \ldots, x_n)$, $P(\boldsymbol{x} \mid y)$ the conditional probability to*

[4] MacKay (2003):p. 6; Wermuth and Streit (2007):p. 137.

observe $\boldsymbol{x}$, given the value y, as well as $P(\boldsymbol{x})$ and $P(y)$ the prior probabilities to observe the values $\boldsymbol{x}$ and y, respectively. Then

$$P(y \,|\, \boldsymbol{x}) = \frac{P(y)\, P(\boldsymbol{x} \,|\, y)}{P(\boldsymbol{x})} \tag{1.9}$$

Proof. By (1.8) we have

$$P(\boldsymbol{x}, y) = P(y) P(\boldsymbol{x} \,|\, y) \qquad \text{and} \qquad P(\boldsymbol{x}, y) = P(\boldsymbol{x}) P(y \,|\, \boldsymbol{x}), \tag{1.10}$$

i.e., $P(y)P(\boldsymbol{x} \,|\, y) = P(\boldsymbol{x})P(y \,|\, \boldsymbol{x})$. □

Figure 1.4 depicts the situation and illustrates the proof above: For instance, the leftmost branch in the first probability tree must equal the leftmost branch in the second one.

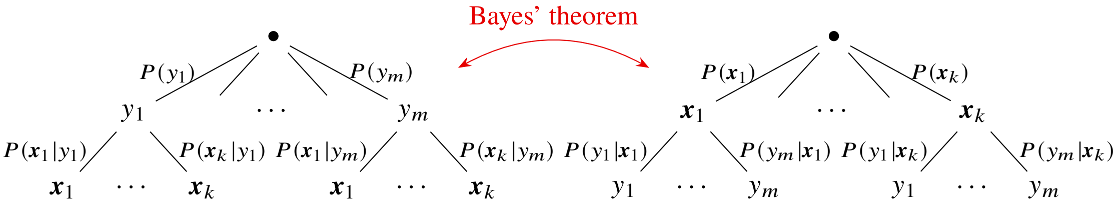


Figure 1.4. Effect of Bayes' theorem: Change of the information situation $P(y \,|\, \boldsymbol{x}) \leftrightarrow P(\boldsymbol{x} \,|\, y)$ for m possible outcomes of y and k possible outcomes of $\boldsymbol{x}$.

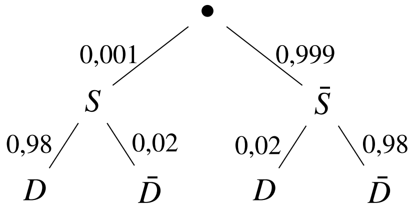
On the surface, Bayes' theorem does not seem very useful. It allows us to compute the single probability $P(y \,|\, x)$ in terms of three other probabilities $P(x \,|\, y)$, $P(x)$, and $P(y)$. So what? Even worse, this seems like computing two steps backwards. But Bayes' theorem is useful in practice because there are many cases where we have good estimates for the three probabilities and need to compute the fourth one. Often, e.g., we observe as evidence the *effect* of some unknown *cause*, and we would like to determine that cause in turn. In this case, Bayes' theorem becomes

cause and effect

$$P(\text{cause} \,|\, \text{effect}) = \frac{P(\text{effect} \,|\, \text{cause}) P(\text{cause})}{P(\text{effect})}. \tag{1.11}$$

The conditional probability $P(\text{effect} \,|\, \text{cause})$ quantifies the relationship in the *causal* direction, whereas $P(\text{cause} \,|\, \text{effect})$ describes the *diagnostic* direction, i.e., the direction of *observation* or *measurement*. In a task such as medical diagnosis, we usually have conditional probabilities in causal relationships. The doctor knows $P(\text{symptoms} \,|\, \text{disease})$, but both doctor and patient want a diagnosis $P(\text{disease} \,|\, \text{symptoms})$.[5]

Example 1.6 (Patient's view of a diagnosis)**.** Let $P(S)$ be the prior probability to have a given desease; $P(D \,|\, S)$ the conditional probability for a positive diagnosis, given the patient has the desease; and $P(S \,|\, D)$ the conditional probability to have the desease, given a positive diagnosis.



$$\begin{aligned} P(\bar{S} \,|\, D) &= \frac{P(\bar{S})\, P(D \,|\, \bar{S})}{P(D)} \\ &= \frac{0{,}999 \cdot 0{,}02}{0{,}999 \cdot 0{,}02 + \mathbf{0{,}001} \cdot 0{,}98} \\ &= \frac{0{,}01998}{0{,}02096} = 0{,}95324 \end{aligned} \tag{1.12}$$

[5] Russell and Norvig (2022):p. 417.

Analogously Bayes' theorem implies

Description	**Designation**	**Probability**
healthy despite a positive diagnosis	$P(\bar{S} \mid D)$	0,95324
sick, given a positive diagnosis	$P(S \mid D)$	0,04676
sick despite a negative diagnosis	$P(S \mid \bar{D})$	0,00002
healthy, given a negative diagnosis	$P(\bar{S} \mid \bar{D})$	0,99998

(1.13)

A positive diagnosis is *wrong* with 95,3 %! (However, a negative diagnosis is correct at 99,998 % ...) The cause of this unexpected result is the low incidence of disease ("prevalence") **1 ‰**. □

Definition 1.7 (Likelihood function). Bayes theorem is often interpreted as a statement about how observed data $\boldsymbol{X}$ affect the probability of a statistical model specified by parameters $\boldsymbol{\theta}$:

$$P(\boldsymbol{\theta} \mid \boldsymbol{X}) = P(\boldsymbol{\theta})\, \frac{P(\boldsymbol{X} \mid \boldsymbol{\theta})}{P(\boldsymbol{X})} \tag{1.14}$$

likelihood of parameters

Here $P(\boldsymbol{\theta})$ is the prior probability that the model parameters are true, and $P(\boldsymbol{\theta} \mid \boldsymbol{X})$ its posterior probability given the observations $\boldsymbol{X}$. However, $P(\boldsymbol{X} \mid \boldsymbol{\theta})$ can also be viewed as a function of both the data $\boldsymbol{X}$ and the parameters $\boldsymbol{\theta}$. For fixed parameters $\boldsymbol{\theta}$ it is a probability over $\boldsymbol{X}$, but for fixed data $\boldsymbol{X}$ it defines the *likelihood* of the parameters $\boldsymbol{\theta}$ given the oberved data $\boldsymbol{X}$. The likelihood is denoted by L,[6]

$$L(\boldsymbol{\theta} \mid \boldsymbol{X}) = P(\boldsymbol{\theta} \mid \boldsymbol{X}). \tag{1.15}$$

In the sequel we will usually be interested in the maximum likelihood L_* for a fixed statistical model and given data $\boldsymbol{X}$, with $L_* = L(\boldsymbol{\theta}_* \mid \boldsymbol{X})$.

In principle, Bayes's theorem describes how the probability of a hypothesis gets updated over time, observing new data. In practice, however, the marginal probability $P(\boldsymbol{X})$, expressing the probability to observe $\boldsymbol{X}$ under all circumstances is hard to estimate. But we will see below (Section 2.4) that it is not necessary to know if we want to *compare* the posterior probabilities for different model hypotheses, given the observations $\boldsymbol{X}$.[7]

1.3 * Conditional independence

Two random variables are called *(unconditionally) independent* if

$$P(x, y) = P(x)\ P(y). \tag{1.16}$$

In real-world contexts, observed quantities are very unlikely to be independent. In fact, if it happens, the initial definition of the object is not relevant and it makes more sense to construct two separate models. A phenomenon more common in reality is the conditional independence.[8]

Definition 1.8. *(Conditional independence)* Let X, Y, Z be random variables with outcomes x, y, z. Then X and Y are called *conditionally independent* given Z if and only if $P(z) > 0$ and

$$P(x \mid y, z) = P(x \mid z). \tag{1.17}$$

Sometimes this property is written $(X \perp Y \mid Z)$

[6] cf. Hastie et al. (2009):p. 265; for a short description see also James et al. (2013):p. 133.

[7] Downey (2011):p. 75.

[8] Pourret et al. (2008):p. 9; Wermuth and Streit (2007):p. 152.

Lemma 1.9. *Let X, Y, Z be random variables with outcomes x, y, z. Then X and Y are conditionally independent given Z if and only if*

$$P(x, y \mid z) = P(x \mid z)\, P(y \mid z). \tag{1.18}$$

Here $P(x, y \mid z)$ is the joint probability of X and Y given Z. This alternate formulation states that X and Y are independent random variable, given Z.

Proof. Starting with equation (1.18) we have the following equivalency conversions

$$\begin{aligned} P(x, y \mid z) = P(x \mid z)\, P(y \mid z) &\overset{(1.8)}{\iff} \frac{P(x, y, z)}{P(z)} = \frac{P(x, z)}{P(z)} \frac{P(y, z)}{P(z)} \\ &\iff P(x, y, z) = \frac{P(x, z) P(y, z)}{P(z)} \\ &\iff \frac{P(x, y, z)}{P(y, z)} = \frac{P(x, z)}{P(z)} \\ &\overset{(1.8)}{\iff} P(x \mid y, z) = P(x \mid z), \end{aligned}$$

applying the definition (1.8) twice. The last equation exactly is the definition (1.17) and thus it is equivalent to equation (1.18). □

Let us now depict the conditional dependence between two random variables by an arrow which expresses the influence or causal relation. If two random variables are conditionally independent, they are not connected by an arrow. This way there are only two possibilities to express a conditional independence of two random variables X and Y given Z, as is depicted in Figure 1.5. Conditional independence therefore is induced either by intermediate random variables ("interrupting" a direct relation), or by a random variable influencing several other random variables. For multiple random variables this procedure forms a directed graph ("digraph") called *Bayesian network.*

Bayesian network

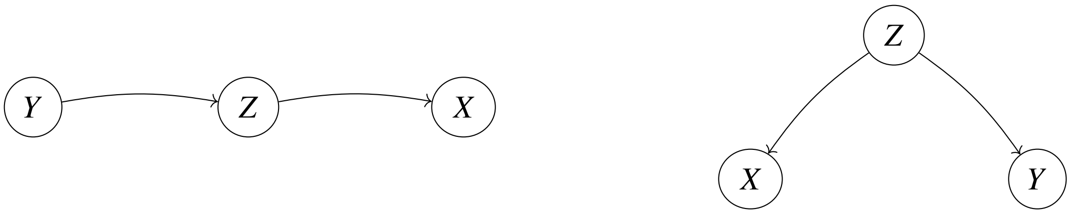


Figure 1.5. The two basic relations expressing the conditional independence of two random variables X and Y given Z.

Example 1.10. *(The lorry driver)*[9] A lorry driver is due to make a 600 mile triTo analyze the risk that X: "he will fall asleep while driving", let us consider whether Y: "he slept well in the night before" (i.e., more than seven hours), and Z: "he feels tired at the beginning of the trip". Obviously, there are causal relationships between the driver's sleep, his perceived fatigue, and the risk of falling asleeTherefore, the three (binary) random variables cannot be independent. Now let us suppose that we know that the lorry driver feels tired at the beginning of the triThen knowing whether this is because of a bad sleep the night before, or to any other reason, does not matter to evaluate the risk. Similarly, if the lorry driver does *not* feel tired at tge beginning of the trip, the quality of his sleep may be considered to have

[9] Pourret et al. (2008):p. 9.

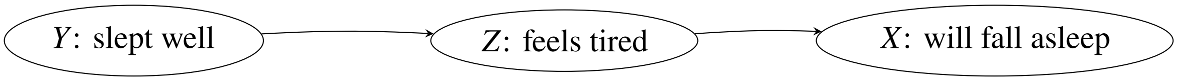


Figure 1.6. The influences of the three random variables in Example 1.10. Variable X is conditionally independent of Y given Z.

no influence on the risk. Thus the risk of falling asleep (X) is conditionally independent of the quality of sleep (Y), given the lorry driver's fatigue (Z). In terms of probabilities of the respective outcomes x, y, z we therefore have

$$P(x \,|\, y, z) = P(x \,|\, z). \tag{1.19}$$

So knowing the values of y and z is not better than knowing the only the value of z. It is useless to describe the behavior of X, Y, Z by a function of three variables, instead we may deduce from (1.19) that

$$P(x, y, z) = P(y)\; P(z \,|\, y)\; P(x \,|\, z) \tag{1.20}$$

Thus the risk model can be constructed by successively studying the quality of sleep, then its influence on the state of tiredness, and then the influence of the tiredness on the risk of falling asleep (Figure 1.6). □

Example 1.11. *(The doped athlete)*[10] During a sports competition, each athlete undergoes two doping tests, aimed to detect if she or he has taken a given prohibited substance: a blood test and a urine test. Both tests are carried out in two different laboratories, without any form of consultation.

Obviously, the two tests are not independent. If the blood test is positive, then the athlete is likely to be doped and thus the urine test will probably be positive, too. But in case the athlete is doped, both tests can be viewed as independent since they use different methods. The same holds true if the athlete is *not* doped: Therefore the two tests are conditionally independent, given the status of the athlete. Formally, if z is the outcome of the binary random variable Z whether the athlete is doped or not, x is the outcome of the blood test, and y the result of the urine test, we can write

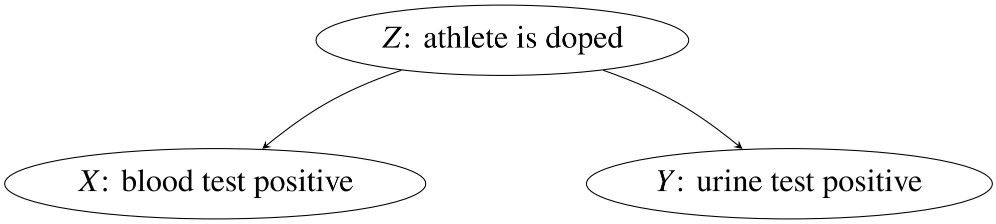


Figure 1.7. The influences of the three random variables in Example 1.11. Variable X is conditionally independent of Y given Z.

$$P(x \,|\, y, z) = P(x \,|\, z), \qquad \text{and symmetrically} \qquad P(y \,|\, x, z) = P(y \,|\, z) \tag{1.21}$$

Both equations tell us that knowing whether the athlete has taken the substance is enough information to the chances of either test being positive, cf. Figure 1.7. It is useless to

[10] Pourret et al. (2008):p. 10.

describe the behavior of X, Y, Z by a function of three variables. Instead, the equations (1.21) yield:

$$P(x, y, z) = P(z)\, P(x \mid z)\, P(y \mid z). \tag{1.22}$$

This means that considerations on the proportion of doped athletes $P(z)$ and on the reliabilities $P(x \mid z)$, $P(y \mid z)$ of each test are sufficient to construct a model. □

1.4 Problems

Problem 1.1 (Random variables). (a) We introduced the notion of a random variable somewhat sloppily on page 7 above. What is the correct formal definition of a random variable X if we designate the sample space by Ω?

□ $X : \Omega \to \Omega$

□ $X : \Omega \to \mathbb{R}$

□ $X : \mathbb{R} \to \Omega$

□ $\mathbb{R} : X \to \mathbb{R}$

(b) Let $\Omega = \{1, 2, 3, 4, 5, 6\}$ be the sample space of a die roll. What is the table of values of the random variable X, yielding the value 1 if an even number of pips is rolled, and 0 otherwise?

Number of pips ω	1	2	3	4	5	6
$X(\omega)$						

What is the probability $P(X = 1)$ that the random variable takes on the value 1? What is the implicit assumption made to compute this probability?

2
Models and theories

Overview

> Erst die Theorie entscheidet, was beobachtet werden kann. *(It is the theory which decides what can be observed.)*
>
> Einstein to Heisenberg (Source: von Weizsäcker, 1985:p. 331)

A basic concept in machine learning and data analysis is the notion of the model. The choice of a model decides how the observed data can be analysed and interpreted. The term "model" in the senses we employ it today only emerged in the 16th or 17th century[1] and was used in science primarily since the formation of quantum theory in the first decades of the 20th century. The severe difficulties to understand the quantum phenomena changed the view on what until then had been designated as a "theory", by relativizing the categorical claim of a theory *to represent the truth.* Especially, various atomic models formed over time, each one restricted to a specific scope, i.e., to a confined section of the reality which could be explained by it. This approach was so successful that even today we use Bohr's model to understand the spectral lines of the hydrogen atom, and the orbital model to explain more complex atoms and molecules.

The aim of this chapter is to clarify the interrelations between the notions of theory and model from a modern point of view, and to correlate it to observations of phenomena of the real world and to the making of hypotheses. All these notions and their interrelations base upon the methods of logical inference. We will start with a short overview of these methods, proceed then with the definitions of theories and models. The chapter ends introducing Occam's razor, a principle to select an optimal model from a collection of several models for a given problem.

[1] from Latin *modulus* – "measure, manner", `https://en.wiktionary.org/wiki/model`, `https://de.wiktionary.org/wiki/Modell`

2.1 Methods of logical inference

There are different forms of logical inference. In ancient Greece, Aristotle distinguished two forms, deduction and induction. However, since the works of the logician and philosopher Charles Sanders Peirce in the beginning of the 20th century, induction is confined from a third form, abduction.[2] Nowadays, however, the term is used in philosophy of science slightly different than Peirce originally defined it.

Deduction is the reasoning from one or more statements, the *premises*, to reach a logical conclusion. Thus deductive reasoning links premises with conclusions: If all premises are true, the terms are clear, and the rules of deductive logic are followed, then the conclusion reached is necessarily true.

deduction

Deduction contrasts with induction. *Inductive reasoning* is a method of reasoning in which the premises are viewed as supplying some evidence of the truth of the conclusion, but not full assurance. It is also described as a method where experiences and observations, including what is learned from others, are synthesized to come up with a general rule. In deductive reasoning, a conclusion is reached by the reduction of general rules, narrowing the range under consideration until only the conclusion remains. In deductive reasoning there is no uncertainty. In inductive reasoning, the conclusion is reached by generalizing or extrapolating from specific cases to general rules resulting in a conclusion that has epistemic uncertainty.

induction

Deduction	**Induction**	**Abduction**
Rule: $A \Rightarrow B$	*Premise:* A	*Rule:* $A \Rightarrow B$
Premise: A	*Conclusion:* B	*Conclusion:* B
Conclusion: B	*Rule:* $A \Rightarrow B$	*Hypothesis* H_1*:* A_1
		$\vdots$ $\vdots$
		Hypothesis H_n*:* A_n
		Best Premise H_**:* A

Table 2.1. Principles of deduction, induction, and abduction.

There is a third form of logical inference, the "abduction." *Abduction* is a method of reasoning to find hypotheses from observations and select the simplest and most likely one of them. A *hypothesis* is an unproved but plausible assumption or a proposed explanation for a phenomenon. It often serves as a *working hypothesis* that is provisionally accepted as a basis for further research. A scientific hypothesis especially requires to be testable by observations or experiments, i.e., verifiable or falsifiable. Abductive reasoning can thus be understood as the inference to the best explanation of an observed phenomenon. Unlike deductive reasoning, abductive conclusions have a remnant of uncertainty or doubt. During the 20th century abduction was widely ignored, but with growing computing power it was rediscovered in the fields of law, computer science, and artificial intelligence research since the 1990s.[3]

hypothesis

abduction

The relationships between deduction and abduction as methods of inference are as follows. Both start from the rule, but suppose either the result or the premise. Deduction

[2] Strictly speaking, the idea of abduction goes back to Aristotle, too, who mentions it by the term *apagoge* (Priori Analytics, II. 25, 69a, `https://logicalstudy.ihcs.ac.ir/article_5171.html`). It also already contrasts with induction. The translation of the term *apagoge* with *abduction* was first done in 1597 by the Italian legal scholar Julius Pacius. Peirce thus only revisited the term, but defined it more precisely and thus made it useable as a scientific method for the philosophy of science and for computer science.

[3] Flach and Kakas (2000).

concludes the result, whereas abduction selects the best hypothesis as premise. Moreover, induction and abuction coincide supposing the conclusion, but alternate the roles of rules and premise as the result of inference.

Example 2.1. The following example is modified from one that Peirce published in 1902.[4] Here the notions of Table 2.1 are assigned by the following statements:

Rule $A \Rightarrow B$:	"All the beans from this bag are white."
Premise A:	"These beans are from this bag."
Conclusion B:	"These beans are white."

For the abduction we formally have the two hypotheses H_1: A and M_2: $\neg A$, from which we choose the most probable one. Note that both the inductive and the abductive inferences may be wrong. Only the deduction is always logically correct. □

Deduction	**Induction**	**Abduction**
All the beans from this bag are white	These beans are from this bag	All the beans from this bag are white
These beans are from this bag	These beans are white	These beans are white
These beans are white.	All the beans from this bag are white	H_1: The beans are from this bag
		M_2: The beans are not from this bag
		H_1: The beans are from this bag

Table 2.2. Examples of deduction, induction, and abduction.

2.2 Theories

> Wie ist Theorie möglich? Sie folgt niemals mit logischer Notwendigkeit aus der Erfahrung. Aus Gesetzen, die sich in der Vergangenheit bewährt haben, folgt nicht mit logischer Notwendigkeit, was in Zukunft geschehen wird.
>
> *(How is theory possible? It never follows with logical necessity from experience. From laws that have proved themselves in the past, it does not follow with logical necessity what will happen in the future. [Translated by the author])*
>
> Carl Friedrich von Weizsäcker (1985:p. 24)

A theory is a finding about a phenomenon of reality, obtained by contemplative and rational thinking.[5] Here *reality* means a consistent world outside of and independent from the observer.[6] In modern science, a *theory* is an explanation of nature, of aspects of the natural world, or of real world phenomena; it can be verified in accordance with the scientific method, using commonly accepted protocols of observation, measurement, and evaluation of results.[7] A theory can produce particular models for real world phenomena.

Example 2.2. *(Quantum Theory)* One of the most spectacular examples of a theory with controversial relationship to reality is quantum theory. Its object is to explain the dynamics of elementary particles and atoms, i.e., the microscopic world. It can be

[4] `https://books.google.com/books?id=E7fnCAAAQBAJ&pg=PA13&redir_esc=y#v=onepage&q&f=false`

[5] In ancient Greek, *θεωρία* (theoria – "looking at", "viewing", "beholding") meant the contemplation of truth through pure thought or speculative understandings of natural things, independent of its realization.

[6] cf. Zeh (2012):p. 50.

[7] For more details cf. `https://plato.stanford.edu/archives/sum2021/entries/scientific-method/`

formulated by Dirac's fundamental superposition principle in an abstract Hilbert space, where the Schrödinger equation describes the dynamics of the Hilbert space vectors representing quantum states.[8] The application of the theory *to local particles*, however, leads to fundamental paradoxes, the most famous of which is the *Einstein-Podolsky-Rosen (EPR) paradox*.[9] It can only be resolved by abandoning at least one the three concepts locality, reality, or free will:[10]

- *No locality:* Locality is the physical principle that interactions cannot propagate faster than light. According to general relativity, locality is necessary for causality, implying that a cause always lies in the past of its effect. However, one resolution of the EPR-paradox is to assume that particle interactions are propagated instantaneously, i.e., faster than light.

- *No reality: Reality* means a consistent world outside of and independent from the observer. One resolution of the EPR-paradox, however, is to assume that microscopic particles do not have properties which can be measured independently from the observer. This is the viewpoint of the model called "Copenhagen interpretation" initiated by Bohr and Heisenberg in the 1920's.

- *No free will:* An observer cannot configure a measurement apparatus independently from the objects to be observed. Any experiment is correlated to an unknown cause in the past or by some mysterious "conspiracy" to the quantum states of the objects.

Each of these options have severe impact on the fundamental validity of physical theories in general. What is wrong? A way out of this dilemma is to question the underlying concept of *particles*: There is no paradox at all, if the superposition principle and the non-local Schrödinger wave function are accepted as real, instead of particles.[11] By this model, or "interpretation", particles are realizations through measurements of the non-local quantum field ψ, similarly to the (widely accepted) model that electrons are realizations of the electromagnetic field.[12] Born was the first to interpret the complex-valued wave function ψ as a "probability wave" such that the probability $P(x, t)$ to measure a quantum particle at point x at time t is given by its modulus square:[13]

$$P(x, t) = |\psi(x, t)|^2. \tag{2.1}$$

In this model, the momentum p is given by the term $i\hbar \frac{\partial}{\partial x} \psi(x, t)$, such that Heisenberg's uncertainty relation $\Delta x \Delta p \geqq \hbar$ is a purely mathematical consequence of the Fourier theorem. These ideas especially go back to de Broglie, Schrödinger, Born, Bohm, Everett,

[8] Zeh (2012):p. 52.

[9] Zeh (2012):p. 15.

[10] Zeh (2012):p. 16.

[11] "I think that it has to be the wavefunction [...] that describes quantum reality." (Penrose, 2004:p. 508). "We have to think of the entire wave as describing (or 'being') just a single particle. [...] We must think of a wavefunction as one entire thing. If it causes a spot at one place [by a measurement], then it has done its job." (Penrose, 2004:p. 512). "The reality of new physical concepts was often initially disputed. Galileo was accused because he considered the Copernican system as real and not only as a calculation method. [...] In the nineteenth century, the electrical field was also initially regarded as a purely formal auxiliary construct for calculating forces on charges. Since we cannot 'directly behold' the field, the question is what we mean by its reality. In the justification of a real electrical field, the consistent 'cogitability' of small sample charges plays an essential role with the help of which it *could* be operationally proved everywhere, without disturbing it noticeably." (Translated from Zeh, 2012:pp. 48–49)

[12] Zeh (2012):p. 52.

[13] Goswami (1997):p. 13; Scheck (2013):p. 39.

and Zeh.[14] However clear and consistent quantum theory might be, until to date there is no generally accepted ontology for it. “It is a common view among many of today’s physicists that quantum mechanics provides us with *no* picture of ‘reality’ at all!”[15] The case remains unsolved. □

> I do not believe that we have yet found the true ‘road to reality’, despite the extraordinary progress that has been made over two and one half millenia.
>
> Roger Penrose in *The Road to Reality* Penrose (2004:p. 1027)

Paradigms and revolutions

According to Heisenberg (1948) a theory is called *closed* if it cannot be improved by minor modifications. An new theory improving a closed theory radically differs from it in certain fundamental notions, while preserving the scope of its successful parts. Thomas Kuhn[16] sees the evolution of such theories in science governed by changing paradigms. Such changes of paradigms he calls “revolutions.”[17]

Example 2.3. *(Copernican Revolution)* A famous example of a revolution in scientific thought is the Copernican Revolution. In Ptolemy’s school of thought, cycles and epicycles (with some additional concepts) were used for modeling the movements of the planets in a cosmos that had a stationary Earth at its center. As accuracy of celestial observations increased, complexity of the Ptolemaic cyclical and epicyclical mechanisms had to increase also to maintain the calculated planetary positions close enough to the observed positions. Copernicus instead proposed a cosmology in which the Sun was at the center and the Earth was one of the planets revolving around it. Both theories in fact are mathematically equivalent. But whereas the Ptolemaic model requires two radii per planet (the radius of the deferent and the radius of the epicycle), the Copernican theory requires only one radius per planet (the radius of its orbit). Hence the Copernican theory is more elegant.

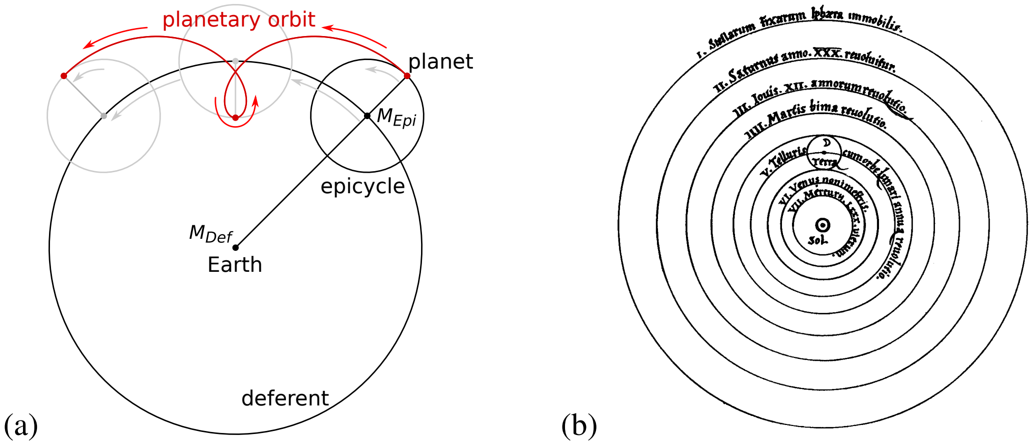

planetary orbit
planet
M_{Epi}
epicycle
M_{Def}
Earth
deferent
(a)
I. Stellarum fixarum ſphæra immobilis.
II. Saturnus anno. XXX. reuoluitur.
III. Iouis. XII. annorum reuolutio.
IIII. Martis bima reuolutio.
V. Telluris cum orbe lunari annua reuolutio.
Terra
VI. Venus nonimestris.
VII. Mercury. LXXX. dierum.
Sol.
(b)

Figure 2.1. The epicycle theory and Copernicus cosmology. Modified from: Koyré (1992:p. 53)

Nonetheless, Copernicus’s theory was rejected at first. One severe objection was that a motion of the Earth should be observable with respect to the “celestial sphere” the stars

[14] von Weizsäcker (1985):pp. 492–499; Zeh (2012):p. 60; cf. also Penrose (2004):p. 507.

[15] Penrose (2004):p. 782.

[16] Kuhn (1962).

[17] Cf. also von Weizsäcker (1985):p. 219.

were assumed to be part of. But since such relative motions of the stars were not observed, it was clear that they must be negligibly small, i.e., light years away – a distance considered impossible at the time. Moreover, the Copernican circular orbits did not fit to the more and more accurate observational data of the motions of the planets, an error which Kepler could resolve first one and a half centuries later by assuming *elliptical* planetary orbits. It was Newton at last who finalized the revolution with his theory of gravitation, unifying the Copernican-Keplerian heliocentric cosmology with Galileo's conjecture about the inertia, breaking with the Aristotelian paradigm that masses always tend to come to rest.[18] □

2.3 What exactly is a model?

The world around us is complex. To understand and handle it — in spite of our limitations and biases — we use representations of reality, called models. In general, a *model* is an informative representation of a system, or an object.[19] Models can be broadly divided into concrete models (e.g. prototypes in engineering, scale models such as architectural models or model railways) and abstract models (e.g. conceptual models, often in mathematical or in diagrammatic forms). Conceptual models are central to philosophy of science, as almost every scientific theory nowadays effectively embeds some kind of model of the physical or human sphere.

models in general

Example 2.4. *(Atomic models)* In the history of the natural sciences there have been emerged various models about the structure of matter. In ancient Greece the philosophers Leucippus and Democritus theorized that the natural world consists of atoms – i.e., indivisible bodies – and the void.[20] But whereas they considered their model to be a theory *that is real*, the atomic models developed in the 20th century have a well-defined scope to represent only limited aspects of reality. For instance the *Bohr model*, assuming that electrons revolve on discrete stable orbits with angular momenta $n\hbar$, $n = 1, 2, \ldots$, around the central nucleus, can well explain the spectral lines of the hydrogen atom, but it fails for more complex atoms. To explain the locations of several electrons around the nucleus, the *orbital model* basing on the Schrödinger wave function may be used, but it cannot be applied to compute molecular orbits. This in turn is the scope of the *molecular orbital model* where electrons are not assigned to individual chemical bonds between atoms, but are treated as moving under the influence of the atomic nuclei in the entire molecule. □

Remark 2.5. Very important models in machine learning are statistical models. They are special forms of mathematical models which consist of functional equations and distinguish variables from model parameters. We will study them below in section 3.3. □

Remark 2.6. Although there are obvious differences in the meanings of a model and a hypothesis, we will use both notions interchangeably in the sequel. The main reasons are the following. Firstly, scientific hypotheses usually have the form of a mathematical model.

[18] My former colleague at Hagen University, Dieter Bangert, stated in a talk: "Aristotle said what daily experience teaches us and gave a plausible explanation. Example 'rolling ball': *A moving body has the tendency to take up the state of rest. The movement therefore comes to a halt by itself; if it is to be maintained, the constant action of a force is required.* Example 'falling body': *Heavy bodies fall faster than light ones.* Aristotle describes what he observes. He gives a plausible, but false, explanation." (http://haegar.fh-swf.de/spielwiese/ART/2_Bangert_Modelle-und-Wirklichkeit.pdf)

[19] The term originally denoted the plans of a building in late 16th-century English, and derived via French and Italian ultimately from Latin *modulus*, a measure.

[20] https://plato.stanford.edu/entries/atomism-ancient/#LeucDemo

Secondly, in machine learning the process of model selection consists of comparing several statistical models, each being viewed as a different hypothesis explaining a given set of data. Moreover, a hypothesis usually is made by abduction from observations, and so are models in model selections. Therefore, a model always is a hypothesis, going to be tested by observations. A hypothesis, in turn, is usually called a model only if it states a more or less complex mathematical relationship of observable data. In contrast, simple statements such as "This drug is safe for humans" are usually not called "models". □

The complex relationships between reality, theory, models, and observations are depicted in Figure 2.2. We see that they mainly base on the methods of logical inference, namely deduction, induction and abduction. Especially, a *deductive model* is a logical structure based on a theory, and an *inductive model* arises from empirical findings and generalization from them.

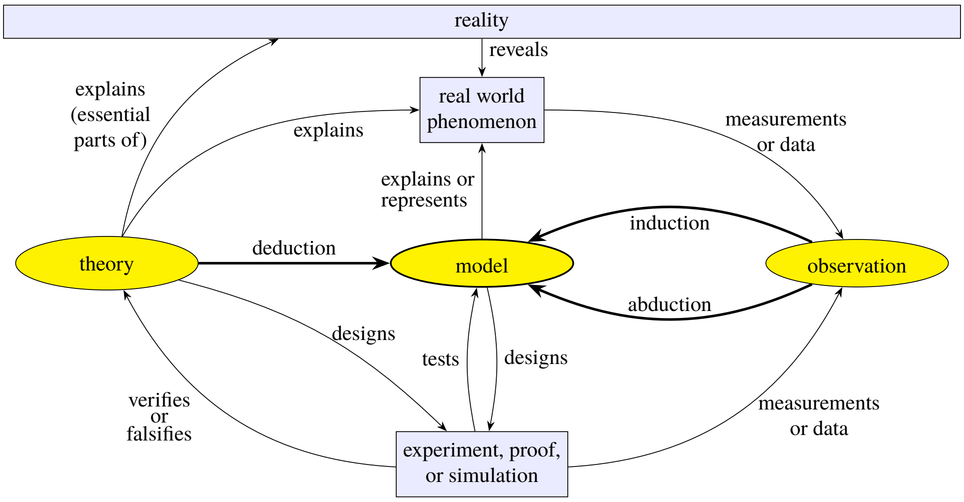


Figure 2.2. A model as a representation of reality and its relationships to theory, models, and observation.

deductive and inductive models

Mathematical models

Very important models in machine learning are statistical models. They are special forms of mathematical models which consist of functional equations and distinguish variables from model parameters.

mathematical models

Definition 2.7. A *mathematical model* is a representation of a system, i.e., a set of related object of the real world, which is expressed by a functional equation

$$y = f(x, \theta) \tag{2.2}$$

of some model variables y and x, some model parameters θ and a function f. Usually, $y \in \mathbb{R}^l$, $x \in \mathbb{R}^n$, $\theta \in \mathbb{R}^k$, and $f : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}$, for some given numbers $n, k, l \in \mathbb{N}$.

Example 2.8. *(Atomic models revisited)* The atomic models in Example 2.4 all are mathematical models, each having specific parameters and variables. For instance, the parameter of the Bohr model is the Planck constant $\hbar$, and the variables are the principal quantum number n and the gain ΔE of energy from orbit $n+1$ to orbit n, determining as outcomes the angular momentum $L = n\hbar$ of the electron on the n-th shell and the frequency $\omega = \Delta E/\hbar$ of the emitted photon. Formally, (2.2) then reads

$$\begin{pmatrix} L \\ \omega \end{pmatrix} = f(n, \Delta E; \hbar) = \begin{pmatrix} n\hbar \\ \Delta E/\hbar \end{pmatrix}, \tag{2.3}$$

i.e., $x, y \in \mathbb{R}^2$, $\theta \in \mathbb{R}$ and $f : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^2$, with $x = \binom{n}{\Delta E}$, $y = \binom{L}{\omega}$, and $\theta = \hbar$. □

Example 2.9. *(Climate models)* Among the most complex mathematical models with considerable social and economical impact are the *climate models* of the IPCC, an intergovernmental body of the United Nations mandated to provide scientific information to understand the human influence on the climate change on Earth.[21] The climate change is documented well and is caused mainly by steadily increasing emission of greenhouse gases,[22] see also Figure 7.1 on page 92. The economic losses due to the climate change have increased during the last 50 years (Fig. 2.3). The climate models that the IPCC reports use aim to represent the essential influences on the climate.[23] The sixth assessment report of the IPCC applies CMIP6, a system which couples various climate models, so-called general circulation models (GCM) modeling the circulations of the oceans and the atmosphere and basing on a system of physical equations.[24] CMIP6 serves to compute five possible socio-economic developments as *scenarios*. They are called *shared socio-economic pathways*, numbered from SSP1 to SSP5 and extend the former scenarios RPCs *(representative concentration pathways)* that describe a set of alternative trajectories for the atmospheric concentrations of key greenhouse gases.[25] Each of these five SSP scenarios represents an individual narrative. They are described as follows.

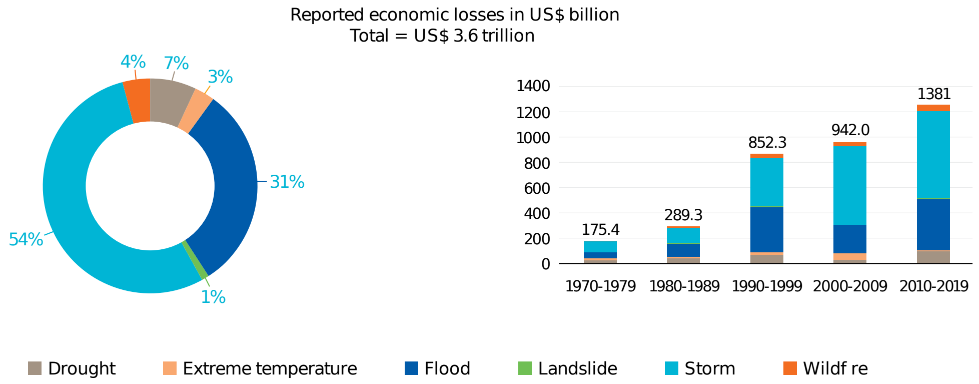


Figure 2.3. Global economic losses due to disasters by decade. Source: WMO (2021:p. 19)

- *SSP1: Sustainability.* The world is looking less at economic growth and more at global well-being and resource-efficient consumption. Investments in education and health promote the demographic transition to a global population size that will soon decline. Inequalities within as well as among nations are declining.

[21] The United Nations formally endorsed the creation of the IPCC (Intergovernmental Panel on Climate Change) in 1988. It does not do own original research, but produces comprehensive assessment reports, each of which builds on previous reports and highlights the current scientific knowledge. The working group 1, especially, published its contribution to the sixth assessment on the physical science basis in 2021 (IPCC, 2021).

[22] e.g. Feldman et al. (2015); IPCC (2021):§1.2, §1.4.1.

[23] IPCC (2021):§1.5.3.

[24] cf. Schönwiese (2008):p. 245.

[25] O'Neill et al. (2017); Tollefson (2020); IPCC (2021):TS.1.3.1, §1.6.1.1; for the methodology, see e.g. Rotmans et al. (2000).

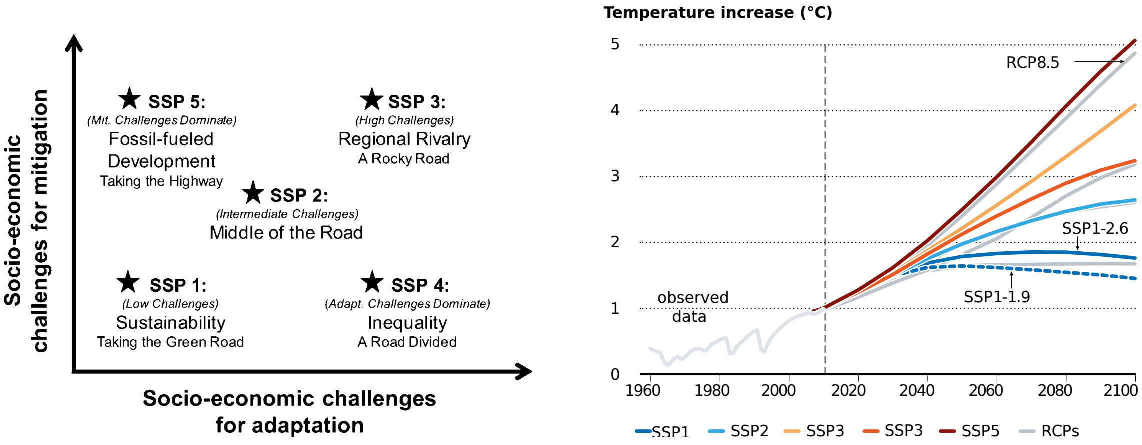


Figure 2.4. The five SSPs considered by IPCC (2021) and their respective predictions of temperature increases until 2100. Sources: O'Neill et al. (2017) and Tollefson (2020)

- *SSP2: Middle of the Road.* Previous social, economic, and technological patterns continue similarly into the future. Nations make unevenly distributed progress. Continued environmental change occurs despite gradually less intensive use of raw materials. Social and environmental vulnerabilities persist longer and are harder to eliminate.

- *SSP3: Regional Rivalry.* Emerging nationalist tendencies make regional security issues seem more important than global problems. The desire for local self-sufficiency in energy and food inhibits interstate cooperation, developments are slower and more costly, and protectionist measures impede trade and oppose common goals. In less prosperous parts of the world, there is strong population growth and environmental degradation.

- *SSP4: Inequality.* Economic and political power is increasingly distributed differently, both within and between states. A divide is emerging: on the one hand, a well-connected, internationalized social class driving an economic and scientific boom; on the other hand, globally fragmented milieus with poor education and limited financial opportunities that hardly benefit from it. Social cohesion is dwindling, and unrest is frequent. Environmental protection measures focus on the elite environment.

- *SSP5: Fossil-fueled Development.* The world relies on innovation, shared economies, and social participation to produce rapid advances that in turn enable sustainable development. However, the processes are driven by intensive use of fossil energy sources as well as energy-intensive lifestyles. The global economy is growing rapidly. Local environmental problems such as air pollution can be managed, and risks from severe climate change are also to be managed technologically.

These scenarios were developed in the year 2014, and since then scenario SSP3 seems to be a storyline that is very close to the path that the major powers of the world are taking. It is defined by a resurgence of nationalism. It sees concerns about economic competitiveness and security lead to trade wars. As the decades progress, national efforts to lock down energy and food supplies short-circuit global development. Investments in education and technology decline. Curbing greenhouse gases would be difficult in such a world, and adapting to climate change wouldn't be any easier. Under this scenario, the average global temperature is projected to soar to more than 4 °C above pre-industrial levels. The scenario seemed rather unlikely when it was developed. But unfortunately it is not. □

Mathematical models can be divided into deterministic models and statistical models. In a *deterministic model*, every variable states is uniquely determined by the parameters in the model and possibly by previous states of these variables. Conversely, in a *statistical model* randomness is present, i.e., variables are not described exactly and uniquely by parameter values, but rather by some probability distributions.

deterministic vs. statistical models

2.4 Occam's razor and model selection

A frequent problem in machine learning and in statistics is to select the best model from several model candidates, explaining some observed data. But what is a "best" model? Amazingly, an old philosophical principle gives us the guideline to formulate a mathematically precise way of quantifying the goodness of a model. This principle we will consider now. Let us start with a simple "model problem" from every-day life. Here we generally do not speak of a "model," but of a "hypothesis." Therefore we will use the terms interchangeably in this section. How many boxes are behind the pillar in Figure 2.5? The usual spontaneous answer is: One! Is this guess rational? Or is it completely arbitrary? MacKay (2003:Chapter 28) argues that this guess is grounded on the research principle called *Occam's razor*. It is the principle that states a preference for simple models and hypotheses: "Accept the simplest explanation that fits to the observational data." Thus according to Occam's razor, we *should* deduce that there is only one box behind the pillar.

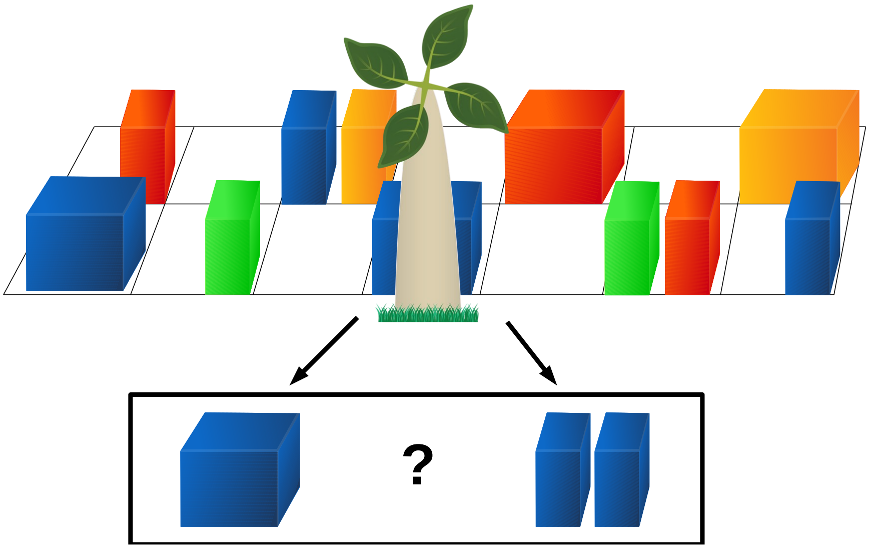


Figure 2.5. How many boxes are behind the pillar? One or two? Or even more? Source: MacKay (2003:p. 343)

Is there a convincing reason for believing that there is most likely one single box? Perhaps your intuition used the argument "well, it would be a remarkable *coincidence* for two boxes to be just the same height and color as each other". To make machine learning and artificial intelligence algorithms interpret that data correctly, we must translate this intuitive feeling into a concrete mathematical formalism.[26]

[26] MacKay (2003):p. 343.

Besides the past empirical success of Occam's razor – William of Occam lived about 700 years ago! – there are two reasons for its justification. The first one is aesthetic: "A theory with mathematical beauty is more likely to be correct than an ugly one that fits some experimental data," as the great physicist Paul Dirac put it.[27] The second reason is probability: If we can estimate or even compute the probability of various possible models to explain a phenomenon, then the model with the highest probability is to be accepted. Indeed it is more probable that there is one box behind the tree than that there are two ones. Let us examine this more closely in the following example.

Example 2.10. How many boxes are behind the pillar in Figure 2.5?[28] Let us assume that each box is either a thin cubiod or a cube standing on equipartitioned squares arranged by a 7 × 2 grid. Two cuboids may share a single square. We assume moreover that there are 4 different colors of boxes, and that all properties of the boxes – form, position and color – have uniform probabilities. Then the model M_1 saying that there is only one box behind the pillar has two free parameters, its position and its color. The position can attain one of $7 \cdot 2 = 14$ squares, and the color is one of four. The model M_2 saying that there are two boxes behind the pillar accordingly has four free parameters, for each box its square and its color. What is the evidence of each model? Let us first state that the prior probabilities $P(H_i)$ of the hypotheses the are equal,

$$P(M_1) = P(M_2) = \frac{1}{2}, \tag{2.4}$$

since by assumption the a randomly chosen box is a cube with probability $\frac{1}{2}$. Then the probability of observing the data X that a green cube is behind the pillar given model M_1 is given by

$$P(X \mid M_1) = \frac{1}{14} \cdot \frac{1}{4} = \frac{1}{56}, \tag{2.5}$$

since it is placed on the square behind the pillar with probability $\frac{1}{14}$, and it is green with probability $\frac{1}{4}$. Analogously, for model M_2 conjecturing that there are two cuboids behind the pillar, the parameters of each single cuboid have the same probabilities as the cube for model M_1, but there are two different combinations of the two colors, i.e.,

$$P(X \mid M_2) = \left(\frac{1}{14} \cdot \frac{1}{4}\right)^2 = \frac{1}{3136}, \tag{2.6}$$

i.e., $P(X \mid M_2) = P^2(X \mid M_1)$. Then the probability $P(M_i \mid X)$ of the validness of model i is given by Bayes theorem, and thus the likelihood ratio of both models reads

$$\frac{P(M_1 \mid X)}{P(M_2 \mid X)} = \frac{P(X \mid M_1)\, P(M_2)}{P(X \mid M_2)\, P(M_1)} = \frac{P(X \mid M_1)}{P(X \mid M_2)} = \frac{1}{P(X \mid M_1)} = 56. \tag{2.7}$$

Therefore the probability of model M_1 is much greater than that of model M_2, given the observational data X. □

probability ratio

Definition 2.11. The ratio in Equation (2.7) we generally call

$$\boxed{\frac{P(M_1 \mid X)}{P(M_2 \mid X)} = \frac{P(X \mid M_1)}{P(X \mid M_2)} \cdot \frac{P(M_2)}{P(M_1)},} \tag{2.8}$$

[27] `https://www.jstor.org/stable/24936146`

[28] Modified from MacKay (2003):§28.2.

the *probability ratio*,[29] given two models, or hypotheses, M_1 and M_2, and observational data X. If the probability ratio is greater than 1, model M_1 is to be preferred, if it is smaller than 1, model M_2. The first ratio of the right-hand side of Equation (2.8),

$$\mathrm{BF}\,(X) = \frac{P(X \mid M_1)}{P(X \mid M_2)} \tag{2.9}$$

is called the *Bayes factor*.[30]

Remark 2.12. In model selection we will usually assume that the prior probabilities $P(M_1)$ and $P(M_2)$ are unknown, but equal. Then the probability ratio (2.8) yields

$$\frac{P(M_1 \mid X)}{P(M_2 \mid X)} = \frac{P(X \mid M_1)}{P(X \mid M_2)}, \tag{2.10}$$

i.e., the ratio of the probabilities of the models equals the ratio of their likelihoods, given the data X. □

Example 2.13. *(The suspicion)* One person left traces of his or her own blood at the scene of a crime. A suspect, Oliver, is tested and found to have type 0 blood. Type 0 is a common type in the local population, with frequency 60%. What is the evidence that Oliver was present at the crime?

Solution: We have two hypotheses, M_1: Oliver was present at the scene, M_2: Oliver was not present. The data are simply X: type 0 is found. Then we see immediately the probabilities

$$P(X \mid M_1) = 1, \qquad P(X \mid M_2) = 0.6, \tag{2.11}$$

since given the hypothesis M_1 that Oliver was present, type 0 is found with certainty, but given hypothesis M_2 that he was not, the probability is just as a random selection from the population. Then Bayes' theorem 1.5 implies

$$P(M_1 \mid X) = \frac{P(X \mid M_1)\, P(M_1)}{P(X)} = \frac{P(M_1)}{P(X)}, \qquad P(M_2 \mid X) = \frac{P(X \mid M_2)\, P(M_2)}{P(X)}. \tag{2.12}$$

Although we do not know the probability $P(X)$ to observe type 0 at the scene of the crime, we can deduce the likelihood ratio (2.8) with $P(X \mid M_1) = P(X \mid M_2) = \frac{1}{2}$ as

$$\frac{P(M_1 \mid X)}{P(M_2 \mid X)} = \frac{P(X \mid M_1)}{P(X \mid M_2)} = \frac{1}{0,6} = 1.667 > 1. \tag{2.13}$$

Therefore hypothesis M_1 is to be slightly preferred. Thus the data provide a weak evidence that Oliver was present at the scene of the crime. □

Example 2.14. *(The crime with 2 suspects)*[31] Two persons left traces of their own blood at the scene of a crime. The blood groups of the two traces are O and AB. A supect, Oliver, is tested and found to have type O blood. Type O is a common type in the local population, with frequency $p_O = 60\%$, type AB is a rare type, with frequency $p_{AB} = 1\%$. What is the evidence that Oliver was present at the crime?

Solution: We have two hypotheses, M_1: Oliver and one unknown person were present at the scene, M_2: two unknown people were present. The data are simply X: one type O and one type AB were found. Then we see immediately the probabilities

$$P(X \mid M_1) = p_{AB} = 0.01, \qquad P(X \mid M_2) = 2\, p_O p_{AB} = 0.012, \tag{2.14}$$

[29] Brandt (1999):p. 186; MacKay (2003):pp. 28, 344; Hastie et al. (2009):p. 234.

[30] Hastie et al. (2009):p. 234.

[31] MacKay (2003):pp. 55–56.

since given the hypothesis M_1 that Oliver was present, type O is found with certainty and type AB by p_{AB}, but given hypothesis M_2 that he was not, the probability is just as a random selection from the population for the first person to have type O and the second one of type AB, as well as that for the first person to have type AB and the second one type O. Then the likelihood ratio (2.8), with $P(X \mid M_1) = P(X \mid M_2) = \frac{1}{2}$, is given by

$$\frac{P(M_1 \,|\, X)}{P(M_2 \,|\, X)} = \frac{P(X \,|\, M_1)}{P(X \,|\, M_2)} = \frac{p_{AB}}{2\, p_O p_{AB}} = \frac{1}{2\, p_O} = 0.83 < 1. \tag{2.15}$$

Therefore hypothesis M_2 is to be slightly preferred. Thus the data in fact provide a weak evidence *against* the suspicion that Oliver was present. □

Remark 2.15. *(Relationships to statistic hypothesis testing)* The procedure to apply Occam's razor as described in this section is part of the so-called "Bayesian inference." It provides an alternative to the standard statistical hypothesis testing, or "null hypothesis testing". Here a so-called "null hypothesis" H_0 is tested against an alternative hypothesis H_1. The alternative hypothesis is the research hypothesis that is to be tested.[32] It is thus accepted when the null hypothesis is rejected. Thus, from the point of view of the research hypothesis, there are two possible errors:

null hypothesis

	H_0 is true	H_1 is true
H_0 is accepted	*specifity* $(1-\alpha)$ "true negative"	type 2 error (β) "false negative"
H_0 is rejected	type 1 error (α) "false positive"	*sensitivity* $(1-\beta)$ "true positive"

(2.16)

The goal of hypothesis testing is to calculate the probability to obtain a type 1 error, i.e., that H_0 is rejected although it is true:

$$p = P(H_0 \text{ rejected} \,|\, H_0 \text{ valid}) < \alpha. \tag{2.17}$$

This probability is the so-called "p-value" and should be smaller than a given "significance level" α to reject the null hypothesis and to accept the alternative one. It should be very small, usually $p < 0.05$. Null hypothesis testing therefore is a *reductio ad absurdum* argument, since the research hypothesis is assumed valid if its counterclaim is highly implausible. The great disadvantage as compared to Occam's razor is that we have no idea how *much* better or worse the null hypothesis is. Moreover, Occam's razor is much more flexible to compare several optional alternatives and to select the most plausible. It even yields a degree of belief given by the likelihood ratio.[33] □

The research hypothesis is accepted only if the null hypothesis is rejected

2.5 Problems

Problem 2.1 (**Logical reasoning**). Consider the three statements "It rains", "The street is wet", "When it rains, the street is wet". Formalize these statements by the variables A, B and $A \Rightarrow B$, and apply them to form a deduction, an induction, and an abduction according to Table 2.1, as is done in Example 2.1.

[32] cf. Wermuth and Streit (2007):p. 192.

[33] for a critical comparison see, e.g., MacKay (2003):p. 458.

Problem 2.2 (Model selection by probability ratio). There two types of bent coins, A and B. A type A coin shows head with probability $p_A = \frac{1}{3}$, whereas a type B coin shows head with probability $p_B = \frac{2}{3}$.

	A	B
head	$1/3$	$2/3$
tail	$2/3$	$1/3$

(2.18)

Suppose we choose a coin at random and toss four tails and one head. Is it more likely to be of type A or B? Or in other words, does model A or model B fit better to the data X?

3
Theoretical foundations of machine learning

Overview

This chapter briefly explains the main terms that will be used in the sequel. Although they mainly describe facts that might be familiar from basic courses of statistics or of machine learning, they are also included here to introduce the notations used in these lecture notes.

3.1 What is machine learning?

The concept of machine learning provides a different perspective from which to view and classify statistical analysis techniques. *Machine learning* is the technique of programming computers so that they can learn from data without having explicitly implemented the rules or patterns to be recognized.[1] Machine learning is a subfield of computer science and is closely related to the concepts of artificial intelligence, data science, and data mining, see table 3.1. *Data Science* is an important subfield of Artificial Intelligence and generally deals with the extraction of knowledge from data. Its main approaches are machine learning and data mining. While data mining is mainly aimed at providing tools to analyse data sets, machine learning focuses on the automated generation of knowledge, i.e. the mechanization of learning: The recognition of rules or patterns should not be

[1] Géron (2017):§1.

Artificial Intelligence (AI)			
Data Science		**Intelligence research**	···
Machine Learning	**Data Mining**	···	···
learn automatically from data	detect patterns and regularities in data	···	···
neural nets, regression, cluster analysis, ...		···	···

Table 3.1. Classification of fields of Artificial Intelligence

done by explicit programming instructions, but by learning processes based on training or observation data. However, the line between machine learning and data mining is blurred. Many of the methods of machine learning and data mining are the same, especially much of the statistical techniques used. The two terms emerged almost simultaneously in 1990s.[2]

Example 3.1 (*Spam filter*)**.** An instance of machine learning software is the spam filter of your e-mail program. It permanently learns from training data you mark as "spam" or "ham," but it is not actively updatet. □

In the field of machine learning, there are three main types of learning, supervised learning, unsupervised learning, and reinforcement learning.

Supervised learning**:** A prepared set of training data with the respective known solutions (*labels*) is created, similar to the sample solutions of mock exams. The system uses this to set the parameters of its (explicit or implicit) model. After this learning phase, the system then labels a new data point *(instance)*, whose solution it does not know, based on the set parameters. That is, the "real" exam is written. An example of this is a spam filter that labels each mail as "spam" or "ham", as already explained in Example 3.1.

Unsupervised learning**:** The system tries to learn from training data without instruction. This is similar to the situation of learning with mock exams without a sample solution. After the learning phase, any new data point can be labeled, i.e., the "real" exam can be written.

Reinforcement learning This is an iterative type of learning. The system *("agent")* observes its environment, selects *"actions"* from a *"policy"* and executes them. Each action is evaluated with a given *reward function*, i.e., it is rewarded or punished. The action is considered or discarded according to the policy before the environment is observed again in the next iteration step, and the cycle starts again with the modified policy. In exam learning, this would correspond to the procedure of taking a "real" exam, adjusting the learned topics ("policy") depending on the exam result, and taking the next exam again.

3.2 Statistical variables

Statistical analyses are based on observed or measured data. For information processing, they are represented as numbers. In principle, the property under consideration determines how well its expression can be measured, i.e. how well it can be expressed in numbers. For example, a person's height can be expressed very easily by numbers, while his intelligence, motivation or state of health are very difficult to measure.[3]

[2] Krahl et al. (1998); Mitchell (1997).

[3] Backhaus et al. (2016):p. 10.

The expressions of a measured trait are plotted on a specific scale. Depending on how the property of an object can be expressed, different levels of measurement are distinguished in statistics. There are two main categories *categorical* and the *metric* scale of measures. Essential for the categorical scale is that their values need not be numbers or, if they are numbers, then only as order numbers, not as computational quantities.

Scale of measure		**Feature**	**Measurable properties**	**Example**	**Calculable quantity**
categorical	**nominal**	qualitative	frequency	blood type, gender	mode
	ordinal	sortable	ranking	educational attainment, tournament ranking	median, quantile
metric	**interval**	subtractable	distance	date, temperature in °C	arithmetic mean, standard deviation
	ratio	dividable	natural zero point	age, price, percent, temperature in Kelvin	geometric mean, variational coefficient

Table 3.2. Scales of statistical variables. Each scale contains the characteristics, measurable properties, and computable quantities of all scales above it.

In contrast, the values of metric scales can be added and subtracted, and statistical quantities such as mean or standard deviation can be calculated. An overview of the different scales of measures is given in table 3.2. There they are arranged such that each scale contains the characteristics, measurable properties, and computable statistical quantities of all preceding scales.

nominal scale

The lowest scale of measure is the nominal scale, whose values are purely qualitative and can only be tested for equality or inequality ($x = y$, $x \neq y$). Examples are scales for blood groups (0, A+, A-, . . .) or for sex (m, w, d). The only statistically measurable property that can be determined for nominal scaled data is the frequency of the individual characteristic values, from which the mode index can be calculated as a statistical measure, i.e. the most frequently occurring value of a sample. However, the values of a nominal scale cannot be sorted meaningfully by size.

However, this is possible with the next higher scale of measure, the ordinal scale, mode index, whose values express a ranking order. While they can be compared in terms of magnitude ($x < y$), the distances between the rank values or the ordinal categories say nothing about the distances between the characteristics of the objects on which they are based. An example is the rank in the Premier Ligue – or Bundesliga, if you like – which expresses the performance of one team compared to the others, but the first-place team is hardly not better than the second-place team by the same amount as the second-place team is better than the third-place team.

interval scale

The next higher scale of measure is the interval scale. It has equal scale segments and thus belongs to the metric scale of measures. A typical example is the Celsius scale for temperature measurement, where the distance between the freezing point and boiling point of water is divided into a hundred equal sections. With interval-scaled data, even the differences between individual measured values have informational content – e.g. large or small temperature difference –. This is exactly not the case with nominal or ordinal data. However, the zero point (freezing point, "Christ's birth") is basically arbitrary in an interval scale, since it depends on the unit used (Celsius versus Fahrenheit, Gregorian versus Jewish or Islamic calendar).

The highest scale of measure is represented by the ratio scale. It differs from the interval scale by the fact that additionally a natural zero point exists. It can be interpreted for the characteristic concerned mostly in the sense of "not present". This is e.g. the case with the Kelvin scale or the big bang as beginning of the time, not however with the Celsius scale or the Gregorian calendar. This is also true for most physical quantities (length, weight, velocity) or most economic characteristics (income, cost, price). In the case of ratio-scaled data, not only the difference possess, but, due to the fixation of the zero point, also the quotient or the ratio of the measured values have information content. Ratio scaled data allow the application of all arithmetic operations as well as the application of all statistical measures mentioned above. Additionally, e.g. the geometric mean or the coefficient of variation can be calculated in a meaningful way. ratio scale

In summary, the higher the scale of measure, the greater the information content. the greater the information content of the data in question and the more computational more arithmetic operations and statistical measures can be applied to the data. It is generally possible to transform data from a higher scale of measure to a lower scale of measure, but not vice versa. This can be useful to increase the clarity of the data or to simplify their analysis. For example, income classes or price classes are often created that transform originally ratio-scaled data down to an interval, ordinal, or nominal scale. Of course, a transformation to a lower scale of measure is always associated with a loss of information.

Remark 3.2. Though the classification of scale of measures looks plausible, there are borderline cases that cannot be assigned at first glance, at least not so easily. For example, school grades ("very good", . . . , "insufficient") are nominally scaled, i.e., strictly speaking, it does not make sense to compute an average grade; however, in most cases, the idea implicit in the evaluation is that each grade covers an equal difference in performance and that school grades are therefore interval-scaled. In this teaching brief, we will be dealing with procedures that expect an entire network to be the independent variable. Since this can be encoded as a matrix and matrices are not sortable, such procedures are only nominal scaled. □

3.3 Statistical models

A *statistical model* assumes *a priori* a factually founded causal relation between the variables. Thus, it presupposes a notion of which variables (“cause”) influence which other variables (“effect”). Those variables that influence the others are called “independent”, the influenced variables are called “dependent”.

Definition 3.3. If the data values of $n + 1$ observable features $x_1, \ldots, x_n, y \in \mathbb{R}$, can be measured with uncertainties or errors, then a *statistical model* with the k parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_k)$ is a functional relationship

$$\boxed{y = f(\boldsymbol{x}, \boldsymbol{\theta}) + \varepsilon} \tag{3.1}$$

with a function $f : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}$ and an error term ε, such that the values of feature y obey a conditional probability $P\big(y \mid f(\boldsymbol{x}, \boldsymbol{\theta})\big)$. The features $x_1, \ldots, x_n$ are called *independent variables* of the model, the feature y is called *dependent variable*, or *target*. Often, $\boldsymbol{x}$ are also called *exogenuous variables* or *predictors*, and y *endogenuous variables* or *response*.

Remark 3.4. In machine learning the function f usually is presupposed, e.g., by a theory or a hypothesis, and the parameters $\boldsymbol{\theta}$ are derived according to m observed data of the features

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \qquad \text{and} \qquad \boldsymbol{X} = \begin{pmatrix} \boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_m \end{pmatrix}. \tag{3.2}$$

Here each data set $\boldsymbol{X}_i$ denotes a data vector of the n features $x_1, \ldots, x_n$, such that $\boldsymbol{X}$ in fact is a matrix, i.e.,

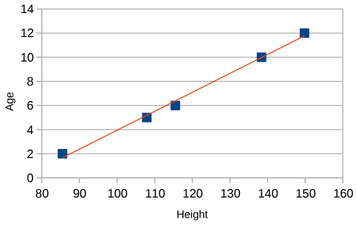
$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_m \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}. \tag{3.3}$$

The methods to determine the parameter values $\boldsymbol{\theta}$, given the data $\boldsymbol{y}$ and $\boldsymbol{X}$, vary depending on the problem. Many methods involve the minimization of residues, i.e., of appropriate distances of the observed data from the model values. Sometimes, Bayesian methods are used to choose the parameters with the highest likelihood.[4] □

Definition 3.5. A tuple $(y_i, \boldsymbol{X}_i)$, with $i = 1, \ldots, m$, of related data values for the features y and $\boldsymbol{x}$ as given by Equation (3.2) is called a *data point* or an *observation*.

Example 3.6 (*Linear regression: Height and age of children*)**.** Let be given the following small sample for age and height of $m = 5$ children.

Height x [cm]	Age y [yrs]
107.9	5
149.8	12
115.5	6
85,5	2
138.4	10



Height x and age y seem to be correlated. A simple model could be:

$$y = \theta_0 + \theta_1 x + \varepsilon \tag{3.4}$$

with an error term $\varepsilon \sim \text{WN}(0, \sigma^2)$ and the parameters $\theta = (\theta_0, \theta_1) \in \mathbb{R}^2$ (Here: $\theta_0 = -11.75, \theta_1 = 0.157)$[5]. □

Example 3.7 (*Logistic regression*)**.** The following influences of learning times on passing the exam have been observed:

Hours (x)	Passed (y)
0.50	0
0.75	0
1.00	0
1.25	0
1.50	0
1.75	0
1.75	1

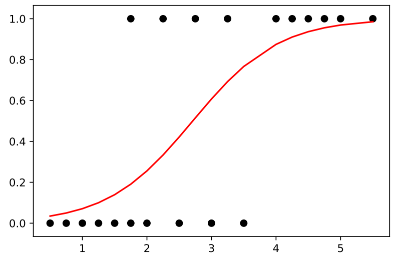
Hours (x)	Passed (y)
2.00	0
2.25	1
2.50	0
2.75	1
3.00	0
3.25	1
3.50	0

Hours (x)	Passed (y)
4.00	1
4.25	1
4.50	1
4.75	1
5.00	1
5.50	1

From this the following statistical model for y as the probability to pass the exam having learned x hours can be applied:

[4] MacKay (2003):pp. 28, 347, 535; Hastie et al. (2009):pp. 30, 34, 233.

[5] K. P. Murphy (2012):p. 19f.



Model:
$y = \mathrm{sigm}\,(\theta_0 + \theta_1 x + \varepsilon)$

(Here: $\theta_0 = -4.0777$, $\theta_1 = 1.5046$)

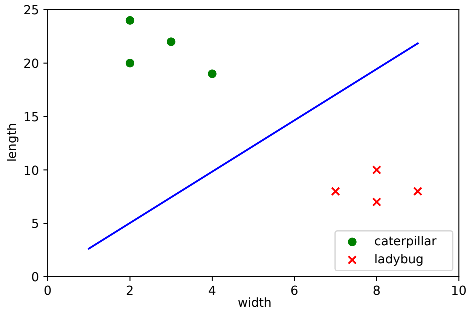
Here $\mathrm{sigm}\,(x) = \frac{1}{1+\mathrm{e}^{-x}}$ is the *sigmoid* or *logistic function*. □

Example 3.8 (*Classification*)**.** The following data of insects have been observed:

Width x_1	Length x_2	Label
9 mm	8 mm	ladybug x
3 mm	22 mm	caterpillar ●
8 mm	10 mm	ladybug x
2 mm	20 mm	caterpillar ●

Width x_1	Length x_2	Label
7 mm	8 mm	ladybug x
2 mm	24 mm	caterpillar ●
4 mm	19 mm	caterpillar ●
8 mm	7 mm	ladybug x

Depicted in a diagram, the data can be classified into two categories:



Model:
$y = \mathrm{sgn}\,(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \varepsilon)$

(Here: $\theta_0 = -0.025$, $\theta_1 = -0.259$,
$\theta_2 = 0.108$)

Here $y = +1$ represents the class “caterpillar” and $y = -1$ the class “ladybug.” □

3.4 Methods of machine learning

Methods of data analysis and machine learning are subject to permanent new development. In addition, the application of these methods is strongly increasing due to the availability of efficient and user-friendly software. Considering this background, we try to present here a selection of multivariate analysis methods that are of particular importance both in university education and in practical applications.

Table 3.3 gives an overview of common methods of machine learning and statistical analysis, most of which we will deal with in these lecture notes. For a more comprehensive survey see, for instance, Backhaus et al. (2016:p. 24), de Vries (2020:§1.2), and Hastie et al. (2009).

Each of these methods bases on a specific statistical model. For instance, linear regression applies a model with a linear function f, such as in Equation (3.4). So in fact the choice of the appropriate method to study given statistical data can be reduced to the selection of “the” appropriate model. Model selection is a complex issue and is discussed in detail in Hastie et al. (2009:§7). Eespecially for linear regression model selection, see James et al. (2013:[§6).

To offer an orientation in the “jungle” of models, in the following sections we classify the various statistical methods according to different criteria, namely according to the scales of their variables, to the type of learning, and to the level of structure recognition.

Method	Typical Example of Use
Regression	
regression analysis	influence of price, advertising expenses, and income on the sales volume
times series analysis	analysis und forecast of the temporal evolution of the sales volume
Classification	
analysis of variance	effect of alternative packaging design on the sales volume of a product
discriminant analysis	credit rating to grant a credit w.r.t. sociodemografic features (age, income, etc.)
logistic regression	determining the risk of heart attack according to a patient's age and cholesterol level
support vector machine	medical diagnosis "sick" or "healthy" on the basis of several symptoms, e.g., maximum pulse and age for a diagnosis "cardiac".
contingency table	relationship between smoking and lung disease
neural network	possible influence factors and forecasts of stock prices
A/B testing	determine which of two email texts is more effective to encourage customers to make a purchase
Clustering	
cluster analysis	formation of personality types based on psychographic characteristics of individuals
Louvain method	detecting clusters in networks
Dimensionality reduction	
(explorative) factor analysis	condensation of a large number of property assessments to underlying evaluation dimensions
conjoint analysis	deriving the utility contributions of alternative materials, shapes, and colors of products to total preference
confirmatoric factor analysis	verifying given indicator variables to measure hypothetical constructs like attitude, purchase intention, loyalty, trust, or reputation
multidimensional scaling	positioning of competing product brands in the perceptual space of consumers
correspondence analysis	representation of product brands and product features in a common space

Table 3.3. Common methods of data analysis

3.4.1 Classification by scales of variables

According to our discussion above statistical variables can be distinguished by their scales of measure, as summarized in Table 3.2. The main categories are categorical versus metric variables. Moreover we can distinguish independent variables from dependent ones. Therefore, each method of machine learning may be specified by the scale of the independent variables and the scale of the dependent variable.[6] So there are four logical categories which the methods can be sorted in, as is shown in Table 3.4. This table can be used to decide from the pure structure of the data to be analysed which method can be applied.

		Dependent variable	
		metric	**categorical**
Independent variable	**metric**	regression analysis principal component analysis factor analysis	discriminant analysis logistic regression
	categorical	analysis of variance social network analysis	correspondence analysis log-linear analysis contingency table choice-based conjoint analysis

Table 3.4. Classification by scales of variables

3.4.2 Classification by type of learning

As we have seen in Section 3.1 there are three types of machine learning: supervised, unsupervised, and reinforcement learning. In this respect, machine learning methods can be classified by their type of learning, as is shown in Table 3.5 for some of them. Regression

Type of Learning	**Method**
supervised learning	regression analysis time series analysis logistic regression discriminant analysis support vector machine neural network
unsupervised learning	principal component analysis cluster analysis factor analysis social network analysis Louvain method
reinforcement learning	deep neural network A/B-Test

Table 3.5. Classification of methods of data analysis by type of learning

[6] cf. Backhaus et al. (2015); Backhaus et al. (2016).

analysis, for instance, can be regarded as a method of supervised learning, since a set of training data with values of both the independent and the dependent variables is required. On the other hand, methods like principal component analysis or cluster analysis are methods of unsupervised learning since the training data do not contain the result explicitly, but it is only determined by the method itself. Finally, methods of reinforcement learning are characterized by the fact that they do not need any training data at all, but learn by doing. *The* prototypical example of this type of learning is deep neural networks like AlphaZero.[7]

3.4.3 Classification by level of structure recognition

In addition to the classification schemes mentioned above, machine learning methods can also be subdivided according to the way they deal with the structures of the respective application problem. In this regard a classification into structure-discovering methods and structure-testing methods often is proposed.[8] These two criteria are understood as follows:

Structure-testing methods are statistical methods to mainly test the causal dependency of a dependent variable on one or several independent variables (factors of influence). In contrast, *structure-detecting methods* are statistical methods to discover connections between variables or objects. A classification of methods of data anlysis according to their respective ability to recognize structures is listed in Table 3.6.

Structure-testing	**Structure-detecting**
regression analysis	principal component analysis
times series analysis	cluster analysis
discriminant analysis	factor analysis
logistic regression	neural networks
analysis of variance	social network analysis

Table 3.6. Classification of methods of data analysis by their levels of structure recognition

Structural testing methods are mainly used to perform analyses of causality. Examples include analyses of whether and to what extent the weather, soil conditions and different fertilizers and quantities affect crop yields, or how strongly the demand for a product depends on its quality, price, advertising and consumer income. These methods are based on a statistical model which assumes a priori a factually based causal relationship between variables, i.e., a notion of which variables (“cause”) influence which other variables ("effect"). Those variables that influence the others are then the independent variables according to definition 3.3, the influenced variables the dependent ones.

The structure-discovering procedures, too, are based on a statistical models whose parameters are set using training data. For the principal component analysis, for example, this is the covariance matrix of the observed data, whose eigenvalues are to be determined as parameters; in cluster analysis, it is the distance or similarity measure of the data objects, which is used to minimize the cluster assignments; and in a neural network, it is its weighted graph, whose edge weights are to be optimized as parameters.

[7] Silver et al. (2017).

[8] Backhaus et al. (2016):pp. 15; Backhaus et al. (2015); Ng and Soo (2018).

3.5 Challenges of machine learning

Machine learning does not necessarily run like clockwork returning perfect results. Instead, it has its pitfalls and cave-ats, but also some serious problems. Two principal problems of machine learning concern the applied model, the other the quality of the training data.

3.5.1 Problems of model choice: underfitting and overfitting

The basic problems of a wrong model choice are underfitting and overfitting. To clarify these notions we first recall the basic aim of machine learning: *Machine learning is a process to fit a model to some given observed or otherwise input data.* The model which is to be fitted depends on the method that is applied, cf. Table 3.3. Two typical classes of methods are regressions and classifications. Let us consider underfit and overfit for these two cases.

Remark 3.9. Roughly said, a regression fits the parameters $\boldsymbol{\theta}$ of a given continuous function $f(\boldsymbol{x}, \boldsymbol{\theta})$, given the data $\boldsymbol{x}$ and y, cf. Example 3.6. On the other hand, a classification fits a model with a discrete (i.e., not continuous) function $f(\boldsymbol{x}, \boldsymbol{\theta})$ which separates the data points by a quality y, cf. Example 3.8. Let us assume that we have the two data sets as depicted in Figure 3.1.

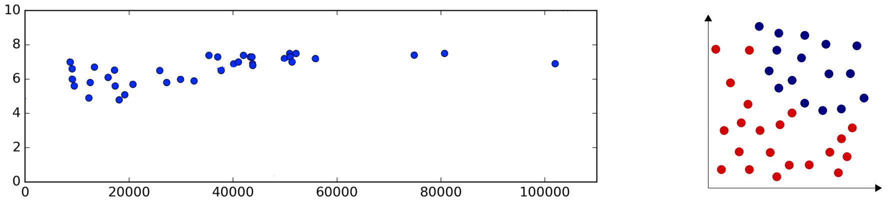


Figure 3.1. Given training data for a regression (left) and a classification (right).

The first crucial step is to guess a model which is suitable to the data. Should we use a linear model? Figure 3.2 depicts this approach in case of a regression and a classification.

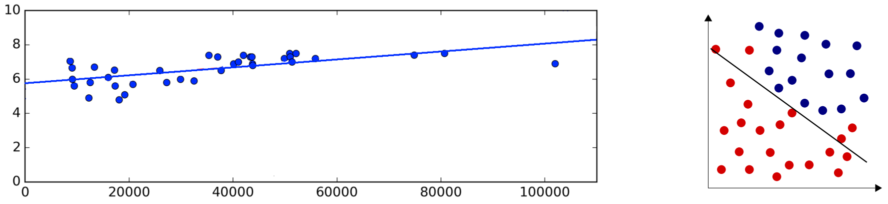


Figure 3.2. Linear models for regression (left) and classification (right).

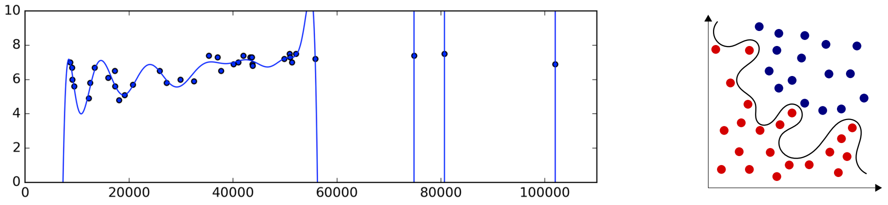


Figure 3.3. Nonlinear models for regression (left) and classification (right).

Or should we better use a nonlinear model as in Figure 3.3? There is no clear and unique criterion to decide which model should be used to express a functional relationship between the data $\boldsymbol{x}$ and y. Moreover, it is not clear how many parameters $\boldsymbol{\theta}$ should be provided. The real problem is that we usually *do not know* the model to be applied. □

We sum up the problems of underfitting and overfitting by Table 3.7. The reason for both of these phenomena is the same, namely an unappropriate model. Underfitting is caused by a model that is too simple to represent the true functional relationship, whereas overfitting is caused by a model that is to complex, i.e., dajusted too tightly to the training data, such that it cannot generalize to new data. Therefore, in both cases the statistical or the causal relations are not understood well enough.

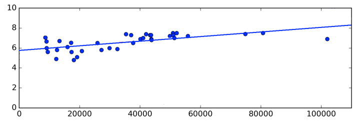
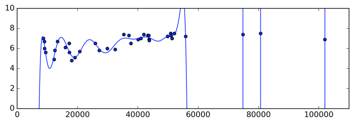
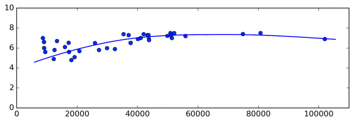
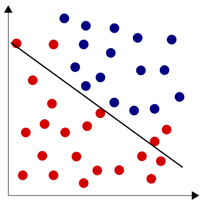
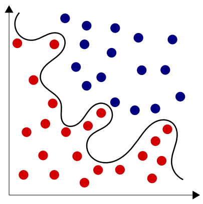
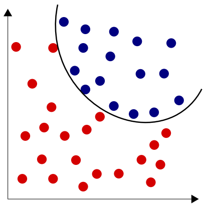
Underfitting	**Overfitting**	**True relation**
		
		

Table 3.7. Underfitting and overfitting of regressions and classifications.

In practice, the choice of a model is a trade-off between model complexity versus concordance with the training data, but we usually do not know the tipping point. That is, we usually do not know: When exactly is the optimum of generalization reached for a simplest possible model?

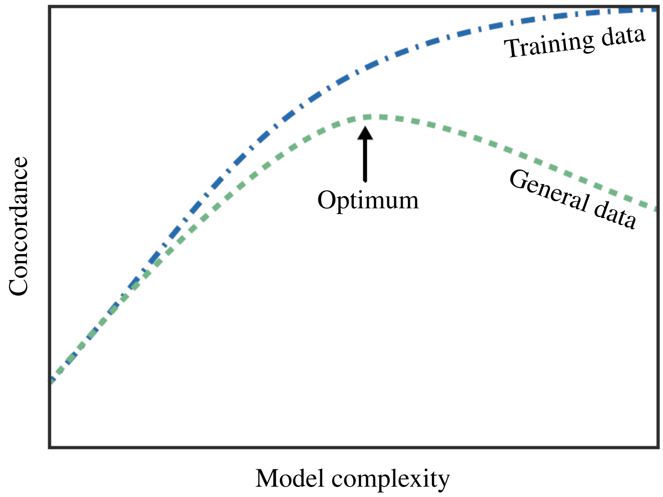


Figure 3.4. The choice of the best model usually is a trade-off between model complexity and concordance with the training data.

In fact the choice should be guided by *Occam's Razor*, cf. Section 2.4 on p. 25 above: From various models of a phenomenon the simplest one is to be preferred. In other words:

> *A "good" model is as simple as possible and as complex as necessary.*

For regressions we will consider some measures to compare different models quantitatively. A common measure is the BIC, basing on the Bayesian version of Occam's razor (2.8). It is defined on page 58 below.

3.5.2 Data quality

A further problem of machine learning concerns the quality of the training data. We usually do not know how "good" or "bad" the training data are that we apply. Bad data may be causd by the following issues.

- The data are imprecise; this may be caused by inaccurate measurements, misunderstandable questionaires, etc ...
- The data are erroneous or wrong
- The data are non-representative, "untypical".

Even applying an ideal model, with wrong data the machine learns the wrong!

4
Introduction to Python

Overview

In these lecture notes we will work with programs written in Python. We therefore will give here a short introduction to this language.

Python is a dynamically typed and object-oriented scripting language with support for functional programming. It was developed in 1991 by Guido van Rossum in Amsterdam and named after the British comedians Monty Python which were popular in the 1970s. The API documentation is online at

`https://docs.python.org/`

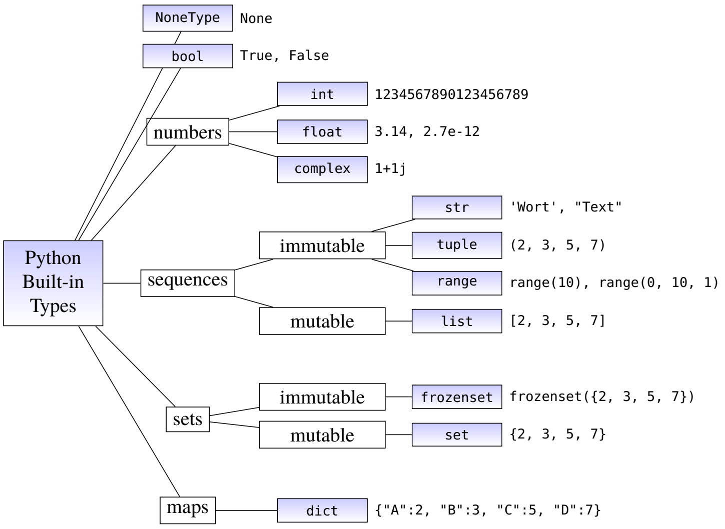
A quick guide to install Python and the packages essential for this teaching brief on various platforms can be found on my web page

`https://math-it.org/AI/installations.html`

There the steps are given to install the *Jupyter Notebook*, which is commonly used for programming with Python and which allows interactive in the browser input of source code and its execution.

Reservierte Wörter				
`and`	`as`	`assert`	`break`	`class`
`continue`	`def`	`del`	`elif`	`else`
`except`	`finally`	`for`	`from`	`global`
`if`	`import`	`in`	`is`	`lambda`
`nonlocal`	`not`	`or`	`pass`	`raise`
`return`	`try`	`while`	`with`	`yield`
Literale				
`False`	`None`	`True`	`12.3`	`'abc'`

(Bildquelle: python.org)



Documentation: `https://docs.python.org/3/library/stdtypes.html`

Figure 4.1. Python's reserved words and built-in datatypes

4.1 Basic language elements

Python's reserved words and built-in datatypes are shown in shown in Figure 4.1. So Python has several basic data types like strings (`"..."` or `'...'`), Boolean values (`True`, `False`), integers of any size, floating point numbers and complex numbers, lists `[..., ...]`. Statements in Python are terminated by a linebreak. Blocks of statements are formed by indentation. Line-by-line comments are marked by `#`. The assignment operator `=` determines both value and type of a variable:

```
a = 1                 # assigning an integer
a = 1.2               # assigning a floating-point number
a = 1 + 5j            # assigning a complex number
a = 'a string'        # assigning a string
a = "also " + a       # assigning a concatenated string
a = [2, 3, 5, 7]      # assigning a list
```

4.1.1 Input and output

The simplest ways for inputs and outputs in Python are the functions `input` and `print`. A string or the value of a variable `a` is output with the with the command

```
print(a)
```

multiple values to be printed can be separated with commas:

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

Numbers can be output as f-strings, or *formatted string literals*, formatted with the prefix `f` or `F`:

```
print(f'{314.1592:.2f}')  # => 314.16
print(F'{31415.92:,.0f}') # => 31,416 with digit group separator
print(f'{0.314159:.2%}')  # => 31.42%
```

Immediately after the formatting prefix there must be a string with a curly bracket, the value to be formatted, a colon, the format specifier, and a closed curly bracket: `f"{`*wert*`:`*spec*`}"`.

With the specifier ".2f" the value is represented as a decimal number rounded to two decimal places. With ",.0f" (i.e., with comma before the dot!) it will be rounded accordingly to 0 decimal places and provided additionally with a comma as digit group separator for numbers greater than or equal to 1000.

Input from the keyboard can be read in with input:

```
input = input("Enter text: ")
```

A text box then appears where text can be entered and stored in the variable input by pressing the return key.

4.1.2 Elementary operations

As in most programming languages, Python enables the basic arithmetic operations by simple operators. Accordingly, string concatenation and common arithmetic operations in Python are listed in Table 4.1. Special features of Python are the two ways to divide two numbers, namely the always real-valued division with / according to IEEE 754), and the integer division with //.

Operator	Meaning	Example	Result
+	addition, sign	2 + 4, +1	6, 1
+	concatenation (strings)	"ab"+"c", '12'+'3'	'abc', '123'
-	subtraction, sign	2 - 4, -1	-2, -1
*	multiplication	2 * 4	8
/	division	7 / 2	3.5
//	integer division	7 // 2	3
%	modulo (remainder of division)	7 % 2	1
**	power	7 ** 2	49

Table 4.1. Concatenation und arithmetic operations in Python

Logical (or Boolean) operators for NOT, AND as well as OR also exist in Python, they are listed in table 4.2. Here the negation receives only one Boolean value True or False and always results in the other value of it. The logical operators and and or, on the other hand, always combine two Boolean values, where and is True only if both values are True, and vice versa or only if both values are False.

Operator	Meaning	Example	Result
not	Verneinung	not True, not False	True, False
and	logisches UND	True and False	False
or	logisches ODER	True or False	True

Table 4.2. Logical operations in Python

Important standard functions in Python are listed in table 4.3. They are used throughout in this lecture notes, especially len() and range(). The function len expects a sequence or a set and returns its length as a value. The function range(x_0, $x_{\max}$, Δ) creates a sequence of numbers (actually an "iterator")

$$x_0 \to x_0 + \Delta \to x_0 + 2\Delta \to \ldots \to x_0 + n\Delta,$$

so that the last sequence member is real smaller $x_{\max}$. All parameters must be integers. If the third parameter is omitted, the step size is 1. If the second one is also omitted, the sequence goes from 0 to $x_0 - 1$. If you want to have the sequence as a list, you can do this by the statement

Function	Meaning	Example	Result
`int()`	conversion to an integer	`int(1.2)`, `int("123")`	`1`, `123`
`float()`	conversion to a floating point number	`float(12)`, `int("1E-12")`	`12.0`, `1e-12`
`str()`	conversion to a string	`str(1.2)`, `str("123")`	`"1.2"`, `"123"`
`len()`	length of a sequence or set	`len([2, 3, 5])`	`3`
`sum()`	sum of a sequence or set	`sum([2, 3, 5])`	`10`
`list()`	sequence or set as a list	`list({2, 3, 5})`	`[2,3,5]`
`range()`	creation of a range	`range(3,18,5)`	`3⟶8⟶13`
`round()`	rounding of floating point numbers	`round(3.1415,2)`	`3.14`

Table 4.3. Important standard functions in Python

```
list(range(4)) # gives [0,1,2,3]
```

Similarly, with

```
sum(range(1,9,2)) # => 1 + 3 + 5 + 7 = 16
```

we obtain the sum of all numbers in the range.

4.2 Control structures

Python has the control structures common in an imperative programming language, controlling the flow of the individual statements. In detail, these are conditional statements, loops, and subroutines, programmable in Python as functions.

4.2.1 Conditional statements with `if`

The conditional execution of a block of statements is programmed in Python by `if`, an (optional) alternative by `else`:

```
x = 6
if x < 0:
    print('negative')
else:
    print('non-negative')
```

Te expression following `if` is a condirion and can therefore be either `True` or `False`. In case that the if-branch contains only a single statement it can be written directly behind the colon. Several statements, however, must be indented to form a block. The following comparison operators for numbers are possible:

equal to:	`a == b`	less than:	`a < b`	greater than:	`a > b`
not equal to:	`a != b`	less or equal:	`a <= b`	greater or equal:	`a >= b`

A multiple case discrimination, i.e., a selection from multiple options, can be programmed by an “else-if ladder” with `elif` (“else if”):

```
month = 4
if month in (1, 3, 5, 7, 8, 10, 12):
    days = 31
elif month in (4, 6, 9, 11):
    days = 30
else:  # only remaining:  month == 2
    days = 28
print("Month ", month, " has ", days, " days")
```

If a single value is to be determined by a simple case distinction, Python still allows the possibility of a single conditional expression with trailing if – else:

```
x if condition else y
```

Depending on the condition, this expression yields either the value `x` or the value `y`. It can be stored in a variable or printed directly with `print`. For example, with the statements

```
from time import localtime
print("Good " + ("morning" if localtime().tm_hour < 12 else "day"))
```

will print a greeting that matches the time of day (... at least between 2am and 6pm).

4.2.2 Loop structures

In Python there are two types of repetition structure by which blocks of statements can be executed several times:

list-controlled loop with `for`

- The list-controlled loop *(for loop)*:

  ```
  for i in range(3):  # [start = 0,] end = 3-1 = 2
     print(i)         # -> 0, 1, 2
  ```

  Property: The number of interations is certain at the start of the loop.

while loop

- The while loop:

  ```
  not_guessed_yet = True   # Boolean variable
  while not_guessed_yet:   # repeat while True ...
      tip = input("Guess the number: ")
      not_guessed_yet = (tip != "7")
  ```

  Property: The number of iterations may be undetermined at the start of the while loop.

Every repetition structure in Python can be interrupted by the expression **`break`**.

```
n = int(input("Input an integer > 2 ein: "))
for i in range(2,n):
    if n % i == 0:  break   # i divides n:  interrupt the loop!
if i == n - 1: print (n, " is prime")
else: print (n, " is not prime: ", n, "/", i, "=", n//i)
```

In Python, if you want to iterate through a given list `[...]` and create a new one from its elements, you can use the *list comprehension*

```
new_list = [expression for entry in list]
```

The entries of the new list are then computed in `expression`. Example:

```
new_list = [i**2 for i in [2,3,5,7,11]] # => [4,9,25,49,121]
```

You can also include a filter condition with an `if` expression before the outer closing parenthesis:

```
new_list = [x for x in range(30) if x % 7 == 0]  # => [7,14,21,28]
```

4.2.3 Deep loops

Loops can be cascaded, for instance to run through two-dimensional data structures such as matrices:

```
for x in range(4):
    for y in range(5):
        print((x,y), end=" ")  # blank instead of linebreak
    print()                    # print linebreak
```

yields the output:

```
(0, 0) (0, 1) (0, 2) (0, 3) (0, 4)
(1, 0) (1, 1) (1, 2) (1, 3) (1, 4)
(2, 0) (2, 1) (2, 2) (2, 3) (2, 4)
(3, 0) (3, 1) (3, 2) (3, 3) (3, 4)
```

With a deep loop we can also print the truth table of logical operations of several variables:

```
print("a b c\t (a or b) and c")
for a in range(2):
    for b in range(2):
        for c in range(2):
            # 3 Boolean variables and a logical expression:
            print(a, b, c, "\t   ", int( (a or b) and c ))
```

This yields the truth table of the logical expression $(a \vee b) \wedge c$. (Try it!):

```
a b c    (a or b) and c
0 0 0       0
0 0 1       0
0 1 0       0
0 1 1       1
1 0 0       0
1 0 1       1
1 1 0       0
1 1 1       1
```

4.2.4 Functions

Functions are defined in Python with the reserved word `def` and a colon, and are called with their names and inserted values. For example, we define the function $f(x, y) = x^2 + y^2$ by:

```
def f(x,y):
    return x**2 + y**2
f(3,4)  # -> 25
```

The variables of a function are called *parameters*, the values used in the call *arguments*. Parameters may be given default values which will be used if they are not given a value when the function is called:

```
def f(x=3,y=4):
   return x**2 + y**2

f(), f(9, 12), f(y=0)   # -> (25, 225, 9)
```

starred expression * unpacks a list of parameters

A special feature of Python: a tuple, invoked as the last parameter of a function as a *starred expression*, can be unpacked with the splat operator :

```
def f(x, y):
    return (x**2 + y**2)**0.5

lst = (3, 4)
f(*lst) # -> 5
```

This way the list `(3,4)` is split into the two separate parameters 3 and 4. However, the expression works exclusively in function calls.

In Python, a function can also be defined within a function, thus being returned as a variable:

```
def f(x):
    def g(y): return x**y
    return g
f(2)(3) # => 8
```

More briefly, we can define the inner function as a *lambda expression*, that is, an anonymous function:

```
def f(x):
    return lambda y : x**y
f(2)(3)    # => 8
```

4.3 Libraries and modules

module

library, package

In addition to the built-in language elements of Python's standard library there are numerous modules for special purposes that can be combined into libraries, or packages, that provide special functions or objects.[1] They can be imported anywhere in a program.

The Python libraries and modules used in this lecture notes are shortly introduced in this section. Hints for complete installation we refer to `https://math-it.org/AI/installations.html`.

NumPy

matplotlib

In the following exemplary program, two very commonly used libraries are imported, NumPy (pronounced "NUM-py") for numerical calculations and arrays (which do not exist in Python as a standard data type, cf. Fig. 4.1), as well as matplotlib for visualizations, especially for the output of function plots. Mostly, however, not the entire library Matplotlib is used, but only the module pyplot:

```
import numpy as np              # module for numerical computations
import matplotlib.pyplot as plt # modul for funkcion plots

x = np.arange(0, 6*np.pi, 0.1)  # array [0, 0.1, ..., 6*pi]
plt.plot(x, np.sin(x))          # function plot (x, sin x)
plt.show()                      # close and show all plt actions so far
```

arrays as argument of a function

The program first creates an array `x` from 0 to 6π (exclusive) in 0,1 steps with the function `np.arange`; `np.arange` is thus the "array" variant of the Python standard function **`range`**, which can only take integer values. In the next statement, the values of the array `x` are plotted against the array `np.sin(x)` of their sine values. This is a very remarkable feature of many NumPy functions: If a single number is passed to it, it returns a number, too;

[1]. `https://docs.python.org/3/library/`

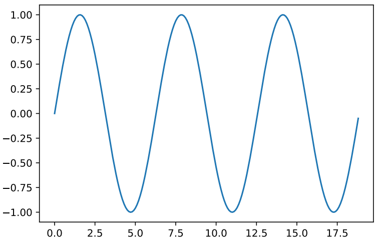


Figure 4.2. The function graph of the sine function with the module `matplotlib`

but if an array is passed to it, the function calculates the function values of the individual array values and returns them in an array of the same size as a result! The standard variant of `plt.plot` represents the two passed arrays as a two-dimensional function graph whose points are connected by a line. With `plt.show()` all previous plt actions (here only one) are completed, summarized, and shown. In this program, this yields the function graph of the sine function shown in Figure 4.2. In the source code you can see that you can give the module an alias name with the reserved word **`as`** that can be used instead of the module name.

`plt.plot`
`plt.show`
alias with **`as`**

In the following example program a variant of the **`import`** statement g imports only a single module or library elements with **`from`** which can be used:

`from`

```
%matplotlib inline
from numpy import pi, linspace, cos
import matplotlib.pyplot as plt

x = linspace(0, 6*pi, 100) # array [0, 0.1, ..., 6*pi]
plt.plot(x, cos(x))        # function plot (x, cos x)
plt.savefig("cos.png")     # saves the graphics as png
plt.show()                 # closes an shows all plt actions
```

However, having imported elements from a library, only these elements are available from it. The example program shows the function graph of the cosine function from figure 4.3. But there are also other variants in this program as compared to the previous one. E.g., instead of `np.arange` here the very similar function `np.linspace` is used. It expects start and end value of the array and the number of entries as parameters. Thus for `np.linspace` the number of entries is in the foreground, whereas for `np.arange` it is the step size between the entries. In the second to last line of the source code, the statement `plt.savefig("cos.png")` saves the graphic to a file `cos.png` in the root directory of the Jupyter notebook. Matplotlib automatically recognizes which format to use by the extension, here it is PNG, but possible are also PDF, SVG, TIFF or JPG.

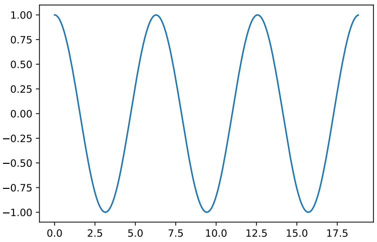


Figure 4.3. The function graph of the cosine function

`np.linspace`
`plt.savefig`

Note: Working with the Jupyter notebook, the statement `%matplotlib inline` in the first line of a Python program effects that the plots are displayed on the first run of the program.

Remark 4.1. There are some good books about Python. One that I always use as a reference for the basic language elements is Weigend (2019). However, since Python has had a very active community of developers for new libraries and modules from the beginning, especially due to the open architecture of the language, books on these points are often quickly out of date. In my opinion, the best way to stay up to date on Python & Co are internet sources, and here especially the *Scipy Lecture Notes* [SciPy], the link to which is listed at the end of these lecture notess. They contain a good introduction to NumPy [NumPy] and to Matplotlib [MPT]. □

4.3.1 Pandas

Pandas has established itself as a kind of base module for machine learning in Python, `https://pandas.pydata.org/`. It provides powerful and flexible methods for imports, transformations, and simple analyses of external data. For our purposes, particularly important are the methods described in `https://pandas.pydata.org/docs/user_guide/` for numerous data formats, especially text-based CSV files (with extension `.csv`, sometimes also `.xls`), or binary file formats like Excel, OpenDocument Spreadsheets, or SPSS.

DataFrames and Series. The two central data structures in Pandas are the *DataFrame* and the *Series*. A DataFrame is a two-dimensional data structure, i.e., a table in which the data are organized as columns (`column`) and each row is identified by an index.

Index	**column name 1**	**column name 2**	$\cdots$
i_0	x_{01}	x_{02}	$\cdots$
i_1	x_{11}	x_{12}	$\cdots$
$\vdots$	$\vdots$	$\vdots$	

Each column has a name with which it can be addressed as in a list with square brackets:

```
df['column name']
```

The index values can be natural numbers, strings or times (Datetime objects). Accordingly, a single value is obtained by addressing its index value with a second square bracket:

```
df['column name']['index value']
```

The name of the row number i is obtained with `df.index[i]`, the names of column number j with `df.columns[j]`.

A *Series* is a one-dimensional structure roughly equivalent to a DataFrame with only one column. In particular, it also has an index. We will use Series in part III to store time series.

To get values of a DataFrame `df` as a numeric array, we need to pass the desired column (DataFrame or Series) to the Numpy object `n_c`:

```
import numpy as np
X = np.c_[df['column name']]
```

The `c_` object in Numpy expects a list of entries and generates a numeric column matrix from it (hence `c_` like "column"). Note that `c_` expects an *angular* bracket immediately, i.e., it does not expect a round bracket like a function. Often we will need the column values of a Pandas object not as a numeric column matrix `array[[...]]`, but as a one-dimensional array. For this, the Numpy function `ravel` is useful, which generates a one-dimensional array from the entries of a matrix, but also of a higher level tensor. With

```
X = np.array([[1, 2, 3], [4, 5, 6]])
x = np.ravel(X)
```

for instance, a matrix X and an Array x is generated:

```
 X = [[1 2 3]
      [4 5 6]]             x = [1 2 3 4 5 6]
```

Example 4.2. In machine learning and data analysis, we will work almost exclusively with external data. The following typical case study will show the first steps for this. Given is a file `children.csv` with the content

```
Größe;Alter
107.9;5
149.8;12
85.5;2
115.5;6
138.4;10
```

the columns of which are separated by a semicolon. To import them into the directory `datasets`, the following statements

```
import pandas as pd
df = pd.read_csv("./datasets/kinder.csv", sep=";")
```

are sufficient It should be noted that the folder here is located in the root directory of the Jupyter notebook. Then the whole content is saved as DataFrame stored in the variable `df`. With `print(df)` you can control its content:

```
   Height   Age
0  107.9      5
1  149.8     12
2   85.5      2
3  115.5      6
4  138.4     10
```

The parameter `sep` is optional, the default value is `','` for the comma as column separator. The optional parameter `encoding` for text encoding is often important as well, here the default value is `'UTF-8'`. For Windows files it is often `'ISO-8859-1'`. □

Pandas DataFrames and Series also provide convenient functions for plotting their data. The optional parameter `kind` can be used to specify the type of plot, its default value is `'line'`. For example, the statement

```
df.plot(kind='bar')
```

yields the function graph in Figure 4.4. In the standard variant of the function for DataFrames, a legend with the column names is displayed. If you do not want them to be displayed, pass `legend=None` as a parameter. Other possible display types are listed under `https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html#other-plots`.

4.3.2 Scikit-Learn

Scikit-Learn is a library for Python for machine learning. and is imported as `sklearn`. The library provides very many models for data analysis as classes (also called "estimators").

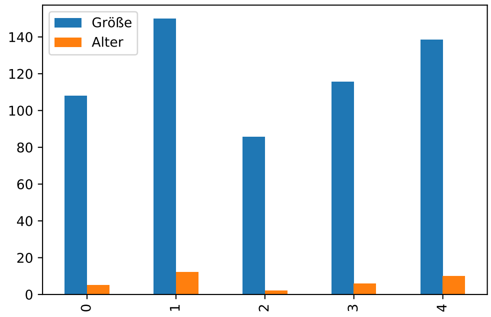


Figure 4.4. Graph of the instruction `df.plot(kind="bar")`

Most of them provide the same functions and syntax for specifying and adjusting model parameters, so for the most part they are uniformly and largely interchangeably applicable. Important classes of scikit-learn are listed in table 6.1. The models range from regression to classification to clustering, but there are also classes for dimension reduction and functions for model selection and data preprocessing. Scikit-Learn is extensively documented under `http://scikit-learn.org`.

4.3.3 Statsmodels

The Python module `statsmodels` enables statistical and econometric analyses. For time series analysis it is better suited than scikit-learn. The API documentation can be found at: `http://www.statsmodels.org/`. For time series analysis the library `statsmodel.tsa.statespace` for non-periodic and periodic processes is provided, in particular the SARIMAX model. We will deal with statsmodels in depth in part III of this lecture notes in connection with the topic of time series analysis. Further information is available at `https://www.statsmodels.org/stable/statespace.html` can be found.

4.4 Problems

Problem 4.1 (Literals in Python). Which of the following strings are valid Python literals?

Literal	**Valid**	**Result / Explanation**
`1.000e-0.2`		
`2e+1j`		
`0x567`		
`00x567`		
`0o567`		
`0o568`		
`'Größe'`		
`''Größe''`		
`"Größe"`		
`b'Größe'`		
`"Komm 'rein!"`		
`00023e001`		
`(1; 2; 3)`		

First, fill in the table according to your thoughts before you test your guesses with Python. The last column should contain the result of the string if it is a valid literal and always a short explanation.

Problem 4.2 (Data types in Python). Generally, in a computer program concrete or mental objects of reality must be represented by suitable data types of a programming language. In the following table, specify a suitable data type in Python for each object of reality.

Object of Reality	Data Type	Exemplary Literal
radius of atoms	`float`	`3.2e-13`
name of flower		
name of participants of a race		
score of a soccer match (e.g., 3:1)		
name, prename and age of a person		
name, prename and age of a participant of a race		
table, in which the chemical element symbols are stored with their English and German names (e.g., H → hydrogen, Wasserstoff)		

Problem 4.3 (Words of an alphabet). Write a Python program that prints all two-letter words of a given alphabet on the screen. The alphabet here should be given as a string, e.g., `alphabet = '01'` with two symbols ("letters") or `alphabet = 'abc'` with three symbols.

Problem 4.4 (Brute force algorithm of DNA). The DNA, carrier of the genetic information of living organisms and some viruses, is a nucleic acid and contains the bases adenine (A), guanine (G), cytosine (C) and thymine (T). It is a giant molecule in the form of a double helix and consists essentially of a sequence of the four base pairs AT, TA, GC and CG.

(a) How many combinations are theoretically possible to form a sequence from four of these base pairs? (An example of such a combination is AT AT GC TA.)

(b) Write a Python program that outputs all possible combinations of four base pairs of DNA.

Problem 4.5 (Multiplication trainer for the small multiplication tables). Program a multiplication trainer for the small multiplication table using Python. The user is to be presented with a total of five arithmetic problems in succession, following the pattern `9*8 =` □ which is to be answered by entering the result. If the entered result is wrong, there is an appropriate feedback and a new answer to this task is expected. The next arithmetic task is not presented until the correct result has been entered beforehand. After five correct entries, the user is told how many seconds it took in total.

notes: In Python, the system time (in seconds since 1/1/1970 0:00 AM) is obtained with the `time()` function of the `time` module. A random number is generated with the `randint` function from the `numpy.random` module. For the syntax please consult the API.

Problem 4.6 (Error bars with NumPy). (a) The function `randn` of the module `numpy.random` seem to have something to do with standard normally distributed random numbers $\varepsilon \sim N(0, 1)$. But what exactly do `randn(3)` and `randn(2,3)` output?

(b) What instruction do you need to program to use `randn` to generate normally distributed random numbers $\varepsilon \sim N(\mu, \sigma)$ with a mean μ and standard distribution σ?

(c) The cosmic background radiation is an electromagnetic radiation flowing through the whole universe in the microwave range, with a frequency around $f = 160$ GHz or a wavelength around $\lambda = c/f = 1{,}7$ mm. It can only be explained as an "afterglow" of the photon flash after the Big Bang, which in the meantime has very much weakened and shifted into the long-wave radio range due to the expansion of the universe. In the early 1990s, the COBE satellite in Earth orbit measured the intensity of the background radiation. The measurement results confirmed the predicted Planck radiation distribution to within 0,01% accuracy for the temperature value $T = 2{,}728$ K, cf. 4.5 figure at left. This distribution function is given by

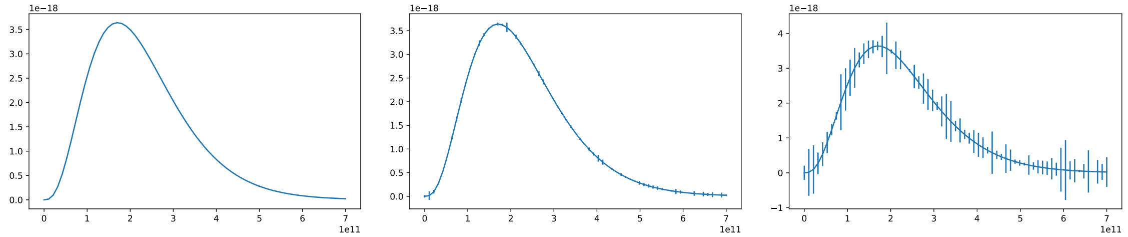


Figure 4.5. The spectrum of the cosmic microwave background for $T = 2{,}728$ K with errorbars of width $\sigma = 0{,}1\%$ of $y_{\max}$, $\sigma = 1\%$ of $y_{\max}$ and $\sigma = 10\%$ of $y_{\max}$, where $y_{\max}$ is the maximum value of the radiant energy density.

$$B(f,T) = \frac{2hf^3}{c^2\left(\exp\left(\frac{hf}{kT}\right) - 1\right)} \tag{4.1}$$

with the constants c for the speed of light, h for Planck's quantum of action, and k for the Boltzmann constant[2]. To illustrate how amazingly small this inaccuracy is, create a Python program in which the Planck distribution is implemented as a Python function $B(f,T)$ and, for $T = 2{,}728$ and a range of values for f from 0 to $7 \cdot 10^{11}$ with 67 data points three error bar plots for an $N(0,\sigma)$-normally distributed error vector ε with 66 measured values each for the error widths $\sigma = 0{,}1\%$ of $y_{\max}$, $\sigma = 1\%$ of $y_{\max}$ and $\sigma = 10\%$ of $y_{\max}$ to the maximum value $y_{\max}$ of radiance as shown in Figure 4.5.

Hints: Use the physical constants from the module `scipy.constants` and the matplotlib function `errorbar`.

[2] Unsöld and Baschek (1999):pp. 110, 493.

Part II

Data analysis

5
Regression

Overview

In Definition 3.3 above we introduced the notion of a statistical model $f : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}$,

$$y = f(\boldsymbol{x}, \boldsymbol{\theta}) + \varepsilon.$$

Here the function f describes a relationship between observed data $\boldsymbol{x} \in \mathbb{R}^n$ and $y \in \mathbb{R}$, depending on some parameters $\boldsymbol{\theta} \in \mathbb{R}^k$. The error term $\varepsilon \in \mathbb{R}$ is a random variable and not directly observable in the data. The observed data can be represented by vectors and matrices,

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \qquad \boldsymbol{X} = \begin{pmatrix} \boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_m \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & & \vdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}, \tag{5.1}$$

cf. Remark 3.4. In other words, the function f represents the model and the parameters $\boldsymbol{\theta}$ its "degrees of freedom". The problem of machine learning, and statistical data analysis, then is to derive the values of the parameters $\boldsymbol{\theta}_*$ from some observed data $\boldsymbol{X}$ and $\boldsymbol{y}$. Especially in machine learning, this derivation process is called "fitting the model."

If now all observational data, i.e, both the independent variables $\boldsymbol{x}$ and the dependent variable y, are metric quantities, the model is called a *regression model*, and the process of deriving the parameters from the observations is called *regression analysis*. The most common form of regression is the linear regression. But before we deal with it, we first consider the question of how to score the goodness of a fitted regression model.

5.1 Residuals and scoring of fitted regression models

If a regression model $f(\boldsymbol{x}, \boldsymbol{\theta})$ has been fitted, the parameters $\boldsymbol{\theta}$ have been determined to be $\boldsymbol{\theta}_*$ such that the *residuals*

$$e_i = y_i - f(\boldsymbol{X}_i, \boldsymbol{\theta}_*) \tag{5.2}$$

are minimized, given m data points $\boldsymbol{y}$ and $\boldsymbol{X}$ by

$$\boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \quad \text{and} \quad \boldsymbol{X} = \begin{pmatrix} \boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_m \end{pmatrix}, \tag{5.3}$$

as has been shown in Remark 3.4. Therefore, with m data points we have m residuals, as is depicted in Figure 5.1. But what does it mean to "minimize" the residuals?

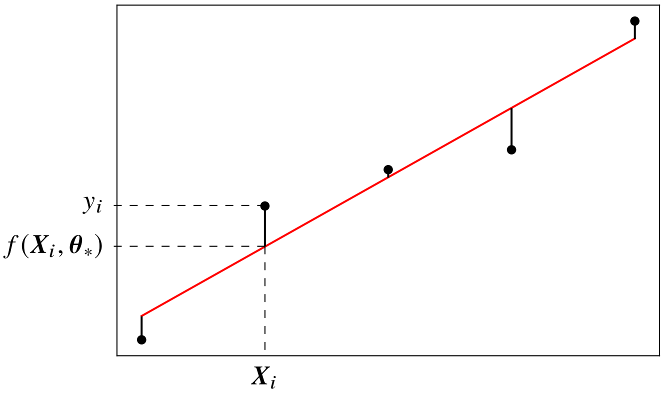


Figure 5.1. The residuals $e_i = y_i - f(\boldsymbol{X}_i, \boldsymbol{\theta}_*)$ of a fitted regression model $f(\boldsymbol{x}, \boldsymbol{\theta}_*)$.

The calculation of the minimum residuals depends on the underlying notion of *distance*. Usually the method of least squares is used which takes the sum of squares of the residuals. Historically, it was introduced independently and nearly simultaneously by Gauss and Legendre. However, different notions of distance are possible and may be more appropriate to the given application. Some of them we will mention below in section 6.1.1.

The least square method relies on the *residual sum of squares* RSS. This is the deviation of the measured value y_i from the corresponding model value $f(\boldsymbol{X}_i, \boldsymbol{\theta}_*)$:

$$\text{RSS} = \sum_{i=1}^{m} \big(y_i - f(\boldsymbol{X}_i, \boldsymbol{\theta}_*)\big)^2. \tag{5.4}$$

It is the RSS which is minimized to determine the coefficients $\boldsymbol{\theta}$, if the least square methods is applied. Vice versa, the sum of squared residues can therefore be used for a fitted model to determine the *coefficient of determination* R^2 (pronounced "R squared"):

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}} \quad \text{with} \quad \text{TSS} = \sum_{i=1}^{m} (y_i - \bar{y})^2. \tag{5.5}$$

R^2 measures the goodness of a regression model

Here TSS represents the total sum of squares of the observed target values. The coefficient of determination thus indicates the proportion of the scatter of the data that the assumed model can explain. The closer it is to 1, the better it is. The coefficient of determination is thus a measure of the goodness of fit of the model and can be used for comparison of other linear models, for instance with different numbers of coefficients.

Remark 5.1. In Python, a convenient way to compute the goodness of a regression model fit is to use the `r2_score` from the package `sklearn.metrics`. As input it requires the true target values and the values predicted by the fitted model:

```
from sklearn.metrics import r2_score
R2 = r2_score(y, y_pred)
```

□

Definition 5.2. A regression model $f(\boldsymbol{x}, \boldsymbol{\theta})$ being linear in its parameters $\boldsymbol{\theta}$, i.e., satisfying

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = g(\boldsymbol{x}) \cdot \boldsymbol{\theta} \tag{5.6}$$

for some function $g : \mathbb{R}^n \to \mathbb{R}^k$, is called *parameter-linear model.*[1]

Remark 5.3 (Caveat to use R^2 for nonlinear regression models). Although R^2 is broadly used to quantify the goodness of a model fit, it is not in general the appropriate measure for a nonlinear regression model. R^2 works quite well for parameter-linear regression models and is a valid measure of the goodness of fit, but this is not the case for nonlinear models in general.[2] The reason is that the total sum-of-squares (TSS) is not necessarily equal to the regression sum-of-squares (RegSS) plus the residual sum-of-squares (RSS), as is the case in linear regression. Here the regression sum of squares is given by

$$\text{RegSS} = \sum_{i=1}^{m} \left(f(\boldsymbol{X}_i, \boldsymbol{\theta}_*) - \overline{f(\boldsymbol{X}_i, \boldsymbol{\theta}_*)} \right)^2, \tag{5.7}$$

and represents the sum of squares of the differences of the model predictions, given the observed $\boldsymbol{x}$ values, and their overall mean.[3] The point now is that for a generalized model f satisfying (5.6) we have $\sum y_i = \sum_i f(\boldsymbol{X}_i, \boldsymbol{\theta}_*)$, i.e., the mean of the observed target values equals the mean of the predicted values.[4] Thus, for these models the equality

$$\text{RegSS} = \sum_{i=1}^{m} \left(f(\boldsymbol{X}_i, \boldsymbol{\theta}_*) - \bar{y} \right)^2, \qquad \text{(for parameter-linear models)} \tag{5.8}$$

holds, and therefore

$$\text{TSS} = \text{RegSS} + \text{RSS} \qquad \text{(for parameter-linear models).} \tag{5.9}$$

In particular, this implies that $R^2 \geqq 0$. However, this inequality might be wrong for nonlinear models. Moreover, the comparison of nonlinear models R^2 does not always yield a clear criterion to select the best one given some observed data. One of the problems is that R^2, as well as RSS, of models with a different number of parameters cannot be compared to each other.[5] □

Definition 5.4. Given a statistical model M with k parameters $\boldsymbol{\theta}$ and a data sample $(\boldsymbol{X}, \boldsymbol{y})$ of size m, the *Bayesian information criterion* BIC is defined by[6]

$$\text{BIC} = k \ln m - 2 \ln L_* \tag{5.10}$$

[1] Hastie et al. (2009):§5.1.

[2] Spiess and Neumeyer (2010).

[3] Spiess and Neumeyer (2010):Additional File 1, esp. Remark 5.

[4] For a proof, cf. Sen and Srivastava (1990):Theorem 2.1, with X replaced by $g(\boldsymbol{x})$, and Corollary 2.2.

[5] cf. James et al. (2013):p. 210.

[6] cf. Durbin and Koopman (2012):p. 188; Hastie et al. (2009):p. 233; James et al. (2013):p. 212; Pourret et al. (2008):p. 62; Shumway and Stoffer (2017):p. 50.

where $L_* = P(\boldsymbol{\theta}_* \,|\, (\boldsymbol{X}, \boldsymbol{y}))$ is the maximum likelihood of the model, cf. (1.15) on p. 12, i.e., $\boldsymbol{\theta}_*$ are the parameter values that maximize the likelihood function. The BIC can be derived[7] from the probability ratio in Definition 2.11, p. 26. How can we interprete the BIC? Given a specified data sample, the lower the value of BIC of a model, the better the model.

Remark 5.5. Especially for least square regression models with identically normally distributed noise variance σ_ε ("Gaussian models") we have $-2 \ln L_* = \ln \sigma_\varepsilon^2 = \ln(\mathrm{RSS}/m)$.[8] Up to a constant depending only on the sample size m, this yields[9]

$$\mathrm{BIC} = k \ln m + m \ln(\mathrm{RSS}/m). \tag{5.11}$$

On some web resources this is given as a definition of the BIC, but it is not mentioned that it is valid only for Gaussian models.[10] □

Remark 5.6. In Python, the easiest way to calculate the BIC of a Gaussian model, i.e., where the residuals e are normally distributed around zero, is to define a function `bic` which expects as input the residuals e and the number of parameters of the model, k, and returns the BIC of the model:

```python
import numpy as np

def bic(e, k):
    return len(e)*np.log(np.sum(e**2)/len(e)) + k*np.log(len(e))
```

cf. also `https://pypi.org/project/RegscorePy/`. □

5.2 Linear regression in one dimension

Example 5.7. As a first example, let us consider a linear regression on the data of the file `children.csv`, whose column separator is a semicolon. To do this, we first import it as Panda's dataframe `df` in line 6, and then filter the columns `"size"` and `"age"` as column arrays in lines 8 to 11, using the statements from section 4.3.1:

```python
 1 %matplotlib inline
 2 import matplotlib.pyplot as plt
 3 import numpy as np
 4 import pandas as pd
 5
 6 df = pd.read_csv("./datasets/children.csv") # loads file into a Pandas DataFrame
 7
 8 features = ["Height"]               # list of independent variable(s)
 9 target = "Age"                      # dependent variable
10 X = np.c_[df[features]]             # extract feature values as columns of a matrix
11 y = np.c_[df[target]]               # extract target values
12
13 from sklearn.linear_model import LinearRegression
14 model = LinearRegression()          # generate linear regression as model
15 model.fit(X, y)                     # adjust the model parameters
16 print("Parameter: θ_0 =", model.intercept_, ", θ_1 =", model.coef_)
```

[7] Hastie et al. (2009):p. 234.

[8] Hastie et al. (2009):p. 233; Shumway and Stoffer (2017):p. 49.

[9] Shumway and Stoffer (2017):p. 50.

[10] e.g., `https://pypi.org/project/RegscorePy/`, `https://stackoverflow.com/questions/60823638`

```
17
18 y_pred = model.predict(X)           # model predictions
19
20 plt.scatter(X, y                )   # scatter diagram of training data
21 plt.plot(X, y_pred, color="red")    # regression line of predictions
22 plt.tight_layout()                  # adjust paddings
23 plt.savefig("children-lin-reg.png") # save plot as PNG
24 plt.show()
```

In line 13 we import the linear regression model from Scikit-Learn. (Of course, we could just as easily have done this at the top of the program, but at this position the dependency appears more clearly). In line 14, we create the model before calculating the relevant model parameters with the statement `model.fit(X,y)` in line 15. The two parameters of the linear regression are the intercept as the first parameter θ_0 and the slope θ_1 as the second one, cf. example 3.6. Now to calculate the regression line there are different possibilities, here we use in line 18 the model prediction with the "test matrix" `X`, being equal to the original training data. We store the result as "prediction" in `y_pred`. The output of the program is the plot in Figure 5.2 and is caused by the instructions from line 20 to the end.

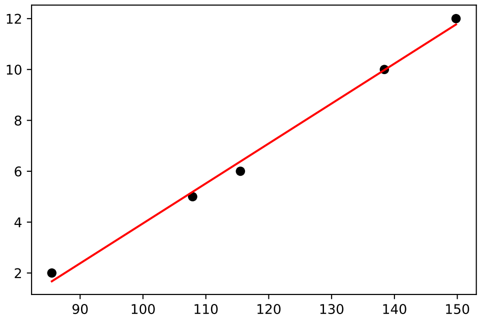


Figure 5.2. Output of the program for linear regression

How good is the model? To evaluate the goodness-of-fit, we apply the R^2 score and the BIC as considered above:

```
# Model evaluation:
from sklearn.metrics import r2_score
R2 = r2_score(y, y_pred)
print("R² =", f"{R2:.3%}")

def bic(e, k):
    return len(e)*np.log(np.sum(e**2)/len(e)) + k*np.log(len(e))
BIC = bic(y - y_pred, 2)
print("BIC =", f"{BIC:.3f}")
```

For our small data set, we obtain

$$R^2 = 99.463\%, \qquad \text{BIC} = 0.542.$$

Thus R^2 is close to 1 and seems to be quite good, as we can confirm intuitively from the plot. Note that the BIC is a measure for model comparison on some given data set, so its value alone does not have an immediate meaning. □

5.3 Multiple linear regression

So far we have considered regression problems with a single independent variable x, i.e., regression models $y = f(x, \theta)$ with a model function $f : \mathbb{R} \times \mathbb{R}^k \to \mathbb{R}$. Regressions with several features $\boldsymbol{x} = (x_1, \ldots, x_n)$, with $n \geqq 2$, are called *multidimensional*. In principle, multidimensional regression models can be linear or nonlinear. However, keep in mind the problems of non-linear models we mentioned at the beginning of section 5.4, which do not become easier in th multidimensional case. A possible way to do multiple linear regression in Python is by using `PolynomialFeautures` of `scikit.learn`, as described in section 5.4.1.

For multidimensional regressions the framework as given in Definition 3.3 and Remark 3.4 is still valid. Furthermore, all computations to fit the model formally remain the same, the only difference being that the algebraic operands of addition and multiplication no longer are pure numbers x and y but vectors and matrices. Consequently, the Python library `scikit-learn` still is applicable for the multidimensional case. (However, the curve fitting with `scipy.optimize` does not work for the multidimensional case.) Let us inspect this in more detail by the following two-dimensional example.

Example 5.8. [11] The SAT *(Scholastic Assessment Test)* is a standardized test widely used for college admissions in the United States. It has two main sections, namely *Evidence-Based Reading and Writing* (EBRW, normally known as the "English" portion of the test) and the Math section. The following table shows the SAT scores of nine students and their grade point average (GPA, maximum is 4):

SAT-Math	79	71	75	74	70	67	73	79	76
SAT-English	74	76	66	76	76	66	71	71	57
GPA	3.95	3.84	3.68	3.59	3.57	3.49	3.47	3.40	3.08

How do the SAT scores influence the GPA? This question can be answered by a two-dimensional linear regression, i.e., with the model

$$y = f(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \tag{5.12}$$

and the data

$$\boldsymbol{y} = \begin{pmatrix} 3.95 \\ 3.84 \\ 3.68 \\ 3.59 \\ 3.57 \\ 3.49 \\ 3.47 \\ 3.40 \\ 3.08 \end{pmatrix}, \qquad \boldsymbol{X} = \begin{pmatrix} 79 & 74 \\ 71 & 76 \\ 75 & 66 \\ 74 & 76 \\ 70 & 76 \\ 67 & 66 \\ 73 & 71 \\ 79 & 71 \\ 76 & 57 \end{pmatrix} \tag{5.13}$$

($m = 9$) Linear regression now solves the equation $\boldsymbol{y} = f(\boldsymbol{X}, \boldsymbol{\theta}) + \boldsymbol{\varepsilon}$ with respect to the three parameters $\boldsymbol{\theta} = (\theta_0, \theta_1, \theta_2)^T$ minimizing the residual squares. This equation is explicity given by the system of $m = 9$ equations

$$\begin{aligned} 3.95 &= \theta_0 + 79 \cdot \theta_1 + 74 \cdot \theta_2 + \varepsilon_1 \\ 3.84 &= \theta_0 + 71 \cdot \theta_1 + 76 \cdot \theta_2 + \varepsilon_2 \\ &\ldots \\ 3.08 &= \theta_0 + 76 \cdot \theta_1 + 57 \cdot \theta_2 + \varepsilon_9 \end{aligned}$$

[11] Sen and Srivastava (1990):pp. 29, 34.

We can implement the regression by the following Python program:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

def bic(e, k):
    return len(e)*np.log(np.sum(e**2)/len(e)) + k*np.log(len(e))

df = pd.read_csv("./datasets/GPA-vs-SAT-scores.csv", sep="\t") # loads file

features = ["SAT-Math","SAT-English"] # list of independent variables
target   = "GPA"                      # dependent variable
X = np.c_[df[features]]               # extract feature values as a matrix
y = np.c_[df[target]]                 # extraxt target values

model = LinearRegression()            # generate linear regression as model
model.fit(X, y)                       # adjust the model parameters
print("Parameter: $\theta_0$ =",model.intercept_,", $\theta_1$ =",model.coef_[0,0],", $\theta_2$ =",model.coef_[0,1])

y_pred = model.predict(X)             # model predictions

# Model evaluation:
R2 = r2_score(y, y_pred)
print("R$^2$ =", f"{R2:.3f}")
print("BIC =", f"{bic(y - y_pred, len(features)+1):.3f}")
```

This gives the solution $\theta_0 = 1.22$, $\theta_1 = 0.0044$, $\theta_2 = 0.029$, i.e.,

$$\text{GPA} = 1.22 + 0.0044 \cdot \text{SAT-Math} + 0.029 \cdot \text{SAT-English}. \tag{5.14}$$

The model evaluations give

$$R^2 = 0.516, \qquad \text{BIC} = -25.686. \tag{5.15}$$

Equation (5.14) seems to indicate that SAT mathematics scores have very little effect on GPA. But we have to keep in mind that the effect of a variable depends on the other variables in the model. With x_1 = SAT-Math alone we get

$$\text{GPA} = 3.5538 + 0.0001 \cdot \text{SAT-Math}$$

and model scores of

$$R^2_{\text{Math}} = 4 \cdot 10^{-6}, \qquad \text{BIC} = -21.356. \tag{5.16}$$

Thus SAT mathematics scores alone are even worse predictors of GPA, at least for the nine students of this sample. With x_2 = SAT-English alone we get

$$\text{GPA} = 1.566 + 0.02840 \cdot \text{SAT-English}$$

and model scores of

$$R^2_{\text{Engl}} = 0.511, \qquad \text{BIC} = -27.794. \tag{5.17}$$

Comparing the three models with respect to their scores in Equations (5.15), (5.16), and (5.17), we obtain a contradictory picture. According to the ergibt sich ein widersprüchliches Bild. Gemäß der coefficients of determination, R^2, the two-dimensional model is better than the two one-dimensional ones. The BIC, however, yields a better ranking for the one-dimensional models. □

Example 5.9. To visualize the regression, we first have to understand that we have to plot a three-dimensional scatter plot, since together with the target values we have three variables x_1, x_2, y. A 3D scatter plot can be implemented in Python with the module `mpl_toolkits.mplot3d`. Given the observational data `X` and `y` from the code in Example 5.8 above, we obtain have the 3D scatter plot as follows:

```
%matplotlib notebook
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d

ax = plt.axes(projection="3d")

ax.scatter(X[:,0], X[:,1], y.ravel(), color="black") # scatter plot of training data
ax.set_yticks(range(55,76,5))                        # sets y-ticks
ax.set_xlabel("SAT-Math"); ax.set_ylabel("SAT-English"); ax.set_zlabel(target)
ax.view_init(azim=-130, elev=24)                     # camera position
plt.show()
```

It is depicted in Figure 5.3 a). For the two-dimensional case, we evidently do not have a regression *line*, but a regression *plane* instead. Given the observational data `X` and `y`, a possible way to plot this plane is by a Numpy meshgrid which is spanned by the minimum and maximum values of the two features x_1 = `X[:,0]` and x_2 = `X[:,1]` and serves as test data predicting the target values by the model. The plane is then plotted by the function `plot_trisurf`:

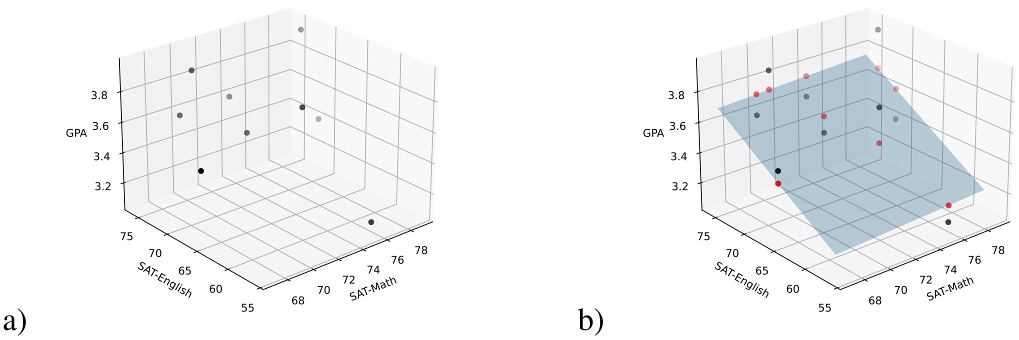


Figure 5.3. a) Scatter plot of the SAT score data in Example 5.8. b) Scatter plot with the regression plane. The red dots are the model values lying in the plane, the residuals are given by the distances to the black-dotted data values with corresponding SAT values.

```
%matplotlib notebook
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d

# Regression plane with min/max of training data:
mg = np.meshgrid(np.r_[X[:,0].min(), X[:,0].max()], np.r_[X[:,1].min(), X[:,1].max()])
zz = model.predict(np.c_[mg[0].ravel(),mg[1].ravel()]).ravel() # predict the training data

# -- 3D scatter plot:
ax = plt.axes(projection="3d")

ax.scatter(X[:,0], X[:,1], y.ravel(), color="black") # scatter plot of training data
ax.scatter(X[:,0], X[:,1], y_pred, color="red")      # scatter plot of predictions
ax.plot_trisurf(mg[0].ravel(), mg[1].ravel(), zz, linewidth=0, alpha=0.3) # regression plane
ax.set_yticks(range(55,76,5))                        # set y-ticks
ax.set_xlabel("SAT-Math"); ax.set_ylabel("SAT-English"); ax.set_zlabel(target)
ax.view_init(azim=-130, elev=24)                  # camera position
plt.show()
```

The result of this program is shown in Figure 5.3 b). Here the red dots are the model values lying in the plane, the residuals are given by the distances of them to their corresponding black-dotted data values. Compare this plot with the plots of the regression lines for the two one-dimensional models in Figure5.4. Note that the SAT-Math regression line has a very small slope $\theta_1 = 10^{-4}$ such that the intercept $\theta_0 = 3.544$ is very close to the mean of the GPA, $\bar{y} = 3.56\bar{3}$. As we saw in Example 5.8 above, the coefficients of determination ranks the models according to their goodness of fit in favor to the two-dimensional model. By the left plot in Figure 5.4 we see another argument in favor of the two-dimensional model: Several SAT-English scores occur more than once, i.e., $x_2 = 76$ occurs three times, and $x_2 = 66$ as well as $x_2 = 71$ each double. However, they all have different GPA values. From the explanatory point of view of the SAT-English score model this phenomenon remains completely obscure. Only in the two-dimensionsal model the influence of the SAT-Math scores become apparent. □

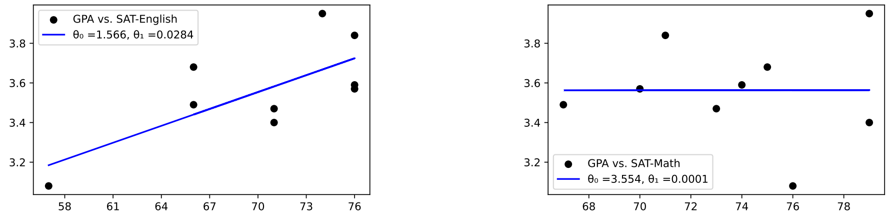


Figure 5.4. Scatter plots GPA versus individual SAT scores and the corresponding regression lines.

5.4 Nonlinear regression

Occam's razor

Often there are problems for which more complex models have to be applied than the ones we have got to know so far. This is the case, for example, when observed data values show a causal relationship, but this relationship is not linear. Actually, this is even the rule, since reality is mostly nonlinear and can at best be approximated by a linear model. In science the accuracy of a model must always be weighed against its ability to explain the essential influences of the phenomena under consideration, cf. Occam's razor cf. Section 2.4.

However, if a nonlinear functional relationship between observed data is sought, we should apply nonlinear regression.[12] Here the fitted model function $f : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}$,

$$y = f(\boldsymbol{x}, \boldsymbol{\theta}) \tag{5.18}$$

Nonlinear problems do not always have a unique solution, sometimes none at all, sometimes several ones

is nonlinear with respect to the independent variables $\boldsymbol{x}$. Unfortunately, we are confronted with a whole series of problems in a nonlinear regression analysis. It is often difficult just to identify the appropriate type of function. Here, insights from the factual logical context of the data should be incorporated, and a certain amount of experience and intuition is also helpful. For example, nonlinear regression models are often based on entire scientific theories, but sometimes a quick glance at the plot of the data is enough.[13] Second, nonlinear regressions are often very intensive computationally. In unfavorable

[12] Backhaus et al. (2015):§1.

[13] Sen and Srivastava (1990):p. 298.

cases, the computations then do not even lead to the best solution, or perhaps even to no solution at all – either because the computational iterations do not converge, or because a solution does not exist at all! Abstractly, nonlinear approaches, in contrast to linear ones, bear the risk of having multiple local optima, of not finding the global optimum, or of not having an optimum at all.

Often it is decisive when fitting nonlinear regression models is often the specification of suitable initial values for the approximation of the parameters, so that the computational methods provide a solution. However, this problem depends individually on the model function f, i.e., there is no general working procedure. There often only "fiddling" helps, i.e. *trial-and-error*, usually supported by function plots.

initial values critical!

In short, the whole thing is so complicated, that a nonlinear regression is possible only for limited cases. Essentially, there are two viable approaches: (1) resolving the nonlinearity by projecting it into higher dimensions, or (2) balancing. Scikit-Learn can be used to implement the first approach. For the second approach, although no option is provided in Scikit-Learn, it is provided in the Python library `scipy`. We will look at both approaches in the following sections.

2 approaches to nonlinear regression in Python

5.4.1 Resolving the nonlinearity with basis functions

In Scikit-Learn, we can implement nonlinear regression if the function f in (5.18) is a linear combination of finitely many basis functions $g_1, \ldots, g_k : \mathbb{R}^n \to \mathbb{R}$:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) \;=\; \theta_1 g_1(\boldsymbol{x}) + \ldots + \theta_k g_k(\boldsymbol{x}) \tag{5.19}$$

We can then project the nonlinearity of f into a higher dimensional *linear* model — at least with respect to the parameters $\boldsymbol{\theta}$:[14]

$$\begin{tikzcd} \text{Nonlinear in} \\ n \text{ dimensions} \arrow[d] & \text{Linear in} \\ & n \times k \text{ dimensions} \\ f(\boldsymbol{x}, \boldsymbol{\theta}) \arrow[r] & \theta_1 g_1(\boldsymbol{x}) + \ldots + \theta_k g_k(\boldsymbol{x}) \arrow[u] \end{tikzcd}$$

Such models are called *parameter-linear*, cf. Equation (5.6) above. How can such a linearization look like? An example is the polynomial regression, i.e., a nonlinear regression with the model function

$$f(x, \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_k x^k, \tag{5.20}$$

a polynomial of degree k. Since this model is already linear with respect to the $(k+1)$ model parameters θ_i (i.e. they are never multiplied or divided by each other), we can express the basis functions

$$g_0(x) = 1, \quad g_1(x) = x, \quad g_2(x) = x^2, \quad \ldots, \quad g_k(x) = x^k \tag{5.21}$$

by the function f as in (5.19) – although supplemented by the constant term $\theta_0 g_0(x)$. In Scikit-Learn this projection is implementable with the transformation `PolynomialFeatures` of the package `sklearn.preprocessing`. It maps each value x of a column vector into a row with its individual powers x^0, x^1, ..., x^k, i.e.,

projection with `PolynomialFeatures`

$$\begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \longrightarrow \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^k \\ \vdots & & & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^k \end{pmatrix} \tag{5.22}$$

To do this, let us consider the instructions:

[14] Hastie et al. (2009):§5.1; VanderPlas (2018):§5.6.2.

```
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
X = np.c_[[2,3]]
poly = PolynomialFeatures(3)
X_trans = poly.fit_transform(X)
print(X_trans)
```

They result in the output:

```
[[ 1.  2.  4.  8.]
 [ 1.  3.  9. 27.]]
```

With the option `include_bias=False` one can exclude the constant terms $x^0 = 1$:

```
X = np.c_[[2,3,4]]
poly = PolynomialFeatures(3, include_bias=False)
X_trans = poly.fit_transform(X)
print(X_trans)
```

Thus we obtain:

```
[[ 2.   4.   8.]
 [ 3.   9.  27.]
 [ 4.  16.  64.]]
```

The success of such an approach depends crucially on the appropriate choice of the basis functions. For possibilities and limitations, consider the following example of a nonlinear polynomial of degree 7 as a model function, which is analyzed as a linear 7-dimensional regression model.

```
%matplotlib inline
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model  import LinearRegression
import matplotlib.pyplot as plt

# -- 0.  Simulate observational data:  --
num = 50  # number of observations
X = np.c_[np.linspace(0, 10, num)]
y = np.sin(X.ravel()) + 0.1*np.random.RandomState(1).randn(num)

#plt.scatter(X,y)

# -- 1.  Nonlinear model with base polynomial of degree 7:  --
poly = PolynomialFeatures(7)
X_trans = poly.fit_transform(X)
model = LinearRegression()
model = model.fit(X_trans, y)

y_pred = model.predict(X_trans)
print("score=",model.score(X_trans, y))

#-- 2.  Plot data points and regression curve:  --
```

```
plt.scatter(X, y)
plt.plot(X, y_pred)  # predictions in traing data
plt.show()
```

Comparison of the simulated training data plotted in a cloud of points with the modeled regression curve in Figure 5.5 shows that a 7th degree polynomial models the data quite well.

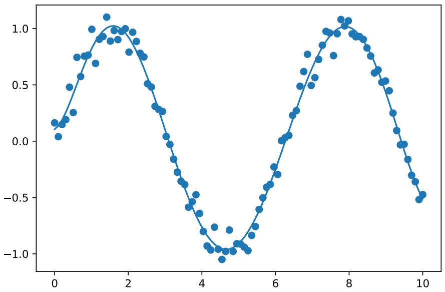


Figure 5.5. parameter-linear model for nonlinear training data

But isn't there something wrong? The data for y are simulated as a sinusoid with random disturbance values. Also on the graph we intuitively expect the data to continue periodically. A polynomial, however, can never be periodic! For example, if we were to use this model to predict data whose x values do not lie in the training interval $[0, 10]$, we would experience a great disappointment:

problem case: unsuitable basis functions

```
X_curve = np.c_[np.linspace(0, 11.5, 1000)]
X_trans = poly.fit_transform(X_curve)
y_curve = model.predict(X_trans)
plt.scatter(X, y)
plt.plot(X_curve, y_curve)
```

gives the plot in Figure 5.6. Thus, as good as the regression model is on the training interval, so useless it is outside of it. We therefore have not found out the "true" character of the functional relation $y = f(x)$, but trained an overfit!

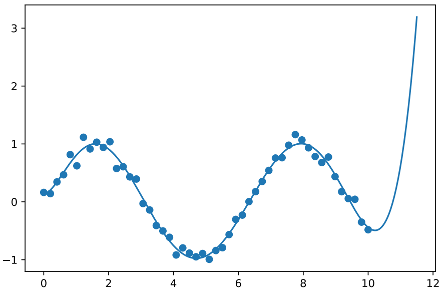


Figure 5.6. parameter-linear model for nonlinear training data

Choosing appropriate basis functions therefore requires that we "understand" the data. For example, if we want to model periodic processes, we also need periodic basis functions. Scikit-Learn does not provide for such functions. However, VanderPlas (2018) gives a nice example in Section 5.6.2 that can serve as a model for creating your own classes that can be used to integrate arbitrary basis functions into a Scikit-Learn model.

In Scikit-Learn, custom basis functions can be implemented

By the way, for polynomial regression, the class `SVR` of the module sklearn.svm also provides possible solutions.

5.4.2 Curve fitting

For the one-dimensional case there exists a simpler process is *curve fitting* for an arbitrary function

$$f(x, \theta_1, \theta_2, \ldots) \tag{5.23}$$

based on observed data x and y. In Python it is provided by the method `curve_fit(f,x,y)` of the module `scipy.optimize`. Here, the function f must always contain the independent variable x as the first parameter, just as in (5.23), and can be supplemented by the model parameters $\theta_1, \theta_2, \ldots$ to be fitted as further parameters. In addition, the method `curve_fit` expects the observed data of the independent variable x and the dependent variable y as Numpy arrays (here scipy is unfortunately not as flexible as other modules like Scikit-Learn or Statsmodels). A usual linear regression, for example, can programmed as:

`curve_fit` expects a certain structure of the function f and the data x, y

```
import numpy as np
from scipy.optimize import curve_fit
def f(x,a,b): return a + b*x  % the regression model
x = np.array([x_1, ... x_m])
y = np.array([y_1, ... y_m])
coefs, res = curve_fit(f, x, y)
```

Here the model parameters are denoted by a and b and represent the intercept and the slope of the regression line. The method `curve_fit` is now called with the function as the first input parameter and the observed data x and y. Its result is a pair of arrays, the first containing the values of the model parameters and the second their covariance matrix. The calculation of these values is done using the method of least squares. It minimizes the sum of squares of the residuals SSR as has been defined in Equation (5.4) above, and is used to compute R^2 in Equation (5.5).

Example 5.10. *(Polynomial of degree 3)* As a first test case, consider the data simulated by the following statements for x and y:

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

x = np.array([-2,-1,0,1,2,3,4,5,6])
y = x**3 - 6 * x**2 + 3*x + 20
```

Thus, by construction, the values of y here are functionally related to the values of x, viz.,

$$y = x^3 - 6x^2 + 3x + 20, \tag{5.24}$$

i.e., a polynial of degree three. But let us pretend we don't know that and just see the data points as a scatter plot as in Figure 5.7. There the function is not directly recognizable. If we now want to apply to these data the model

$$f(x) = a + bx + cx^2 + dx^3 \tag{5.25}$$

of a polynomial of degree three with the four model parameters a, b, c and d, then we can achieve this by the following instructions:

```
from scipy.optimize import curve_fit
def f(x,a,b,c,d): return a + b*x + c * x**2 + d * x**3
coefs, cov = curve_fit(f, x, y)
y_pred = f(x, *coefs)  # model predictions
```

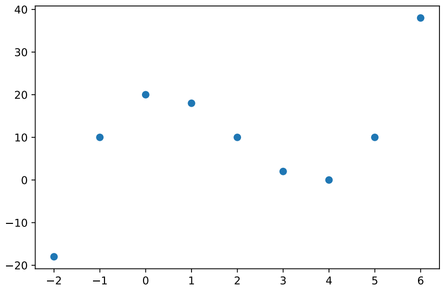


Figure 5.7. Data of a cubic polynomial

The values $y_{\text{pred}} = f(X, \boldsymbol{\theta}_*)$ predicted by the fitted model can be printed by the instruction,

```
print("Fitted coefficients:", [round(t,3) for t in coefs])
```

where we obtain

```
Fitted coefficients: [20.0, 3.0, -6.0, 1.0]
```

i.e., $a = 20$, $b = 3$, $c = -6$ und $d = 1$. Goal! The fact that this estimated model perfectly represents the data is on the one hand visually plausibilized by plotting the regression curve (Fig. 5.8). Evaluating the model with the coefficient of determination R^2 and the BIC, we let run the following instructions:

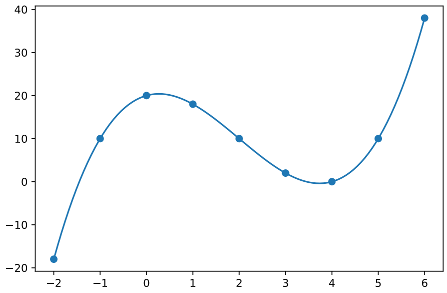


Figure 5.8. The data and the estimated regression curve

```
y_pred = f(x, *coefs)
from sklearn.metrics import r2_score
R2 = r2_score(y, y_pred)

def bic(e, k):
    return len(e)*np.log(np.sum(e**2)/len(e)) + k*np.log(len(e))

print("R² =", f"{R2:.3%},", "BIC =", f"{bic(y - y_pred, len(coefs)):.0f}")
```

The output shows that the coefficient of determination $R^2 = 100\%$, and BIC $= -56$. □

5.4.3 Initial values of the fitting process

Calculating the model parameters for a nonlinear regression can lead to the serious problems, which are unknown from the linear regression. For instance, the fitted values can produce a nonsensical result or no solution is found at all and the calculation is stopped. (`curve_fit` stops its calculation after 800 iterations). Finding a nonsensical solution is perhaps even worse than when the procedure stops, because often there is no warning message and one easily does not notice the error. A nonsensical solution can often be recognized visually by plotting the observed values and the estimated model in a diagram, or by the fact that the coefficient of determination is greater than 1 or less than 0.

The reason for these effects is that the iterative estimation method depends very sensitively on the initial values of the model parameters. The method `curve_fit` starts with the value 1 for all model parameters by default. In some cases, this causes the estimation procedure not to converge. The optional parameter `p0` can be used to pass a list of initial values to `curve_fit` when it is called, where they must appear in the same order and number as in the regression function, e.g.,

```
def f(x,a,b,c): return a*x + b/x + c
curve_fit(f, x, y, p0=(0.5,2,100))
```

for the initial values $a = 0.5$, $b = 2$ und $c = 100$.

Finding suitable initial values can be a difficult task and cannot be automated. *"Unfortunately, there are no good rules for starting values."*[15] Often you have to approach it with trial and error. And sometimes even that, as well as a lot of patience does not help: Maybe the regression model is unsuitable or even wrong!?

5.4.4 Case study: Kepler's model of planetary orbits

As an example, we consider a problem from astronomy that Kepler solved on May 15, 1608, and published a year later in his work *Harmonice Mundi*. As we would put it today,

Planet	Mercury	Venus	Earth	Mars	Jupiter	Saturn	Uranus	Neptune
Period [a]	0.2408	0.6152	1	1.8810	11.8626	29.4475	84.0168	164.7913
Semi-axis [AU]	0.3871	0.7233	1	1.5237	5.2033	9.5371	19.1912	30.0690

Table 5.1. Orbital data of the planets of the solar system: the major semimajor axis in astronomical units (AU) and the orbital periods in years (a).
(Source: `https://en.wikipedia.org/wiki/List_of_gravitationally_rounded_objects_of_the_Solar_System#Major_planets`)

he succeeded in establishing a functional relationship between the orbital period T and the major semi-axis r of the planets of the solar system. From the point of view of data analysis, his problem can be conceived as the development of a nonlinear regression model from given observational data. The data are listed in table 5.1 They are depicted as the scatter plot in Figure 5.9 by the following instructions:

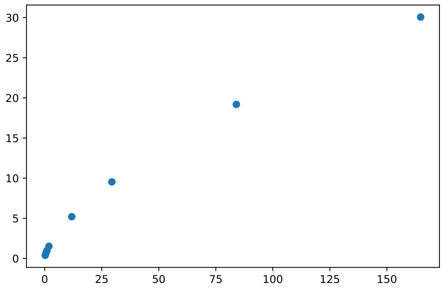


Figure 5.9. The semi-axes of the planets plotted against their orbital periods

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

[15] Greene 2003, quoted from Backhaus et al. (2015:p. 31)

```
T = np.array([0.2408, 0.6152, 1, 1.8808, 11.8626, 29.4475, 84.0168, 164.7913])
r = np.array([0.3871, 0.7233, 1, 1.5237, 5.2034, 9.5371, 19.1913, 30.0690])
plt.scatter(T, r)
plt.show()
```

This visualization may suggest a connection through a continuous function. What function might that be? Let us apply three different regression models to explore, a linear model f_1, a quadratic model f_2, and a power model f_{pow}:

$$f_1(t,a,b) = a + bt, \quad f_2(t,a,b,c) = a + bt + ct^2, \quad f_{\text{pow}}(t,a) = t^a. \tag{5.26}$$

In Python, we can then estimate the model parameters a, b, ... using the orbital data of the planets as follows:

```
from scipy.optimize import curve_fit
def f1(t,a,b)  : return a + b*t
def f2(t,a,b,c): return a + b*t + c*t**2
def f3(t,a) : return t**a
coefs1, cov1       = curve_fit(f1, T, r)
coefs2, cov2       = curve_fit(f2, T, r)
coefs3, cov3 = curve_fit(f3, T, r)
```

The computed parameter values then are printed by the following instructions:

```
print("Coefficients of the linear model", coefs1)
print("Coefficients of the square model", coefs2)
print("Coefficients of the power model", coefs3)
```

They read

$$\begin{aligned} \text{linear model:}\quad & a = 1{,}79, \quad b = 0{,}18 \\ \text{square model:}\quad & a = 0{,}97, \quad b = 0{,}28 \quad c = -6{,}26 \cdot 10^{-4} \\ \text{power model:}\quad & a = 0{,}6667 \approx \tfrac{2}{3}. \end{aligned}$$

The instructions

```
plt.scatter(T, r)
plt.plot(T, f1(T, *coefs1), label="linear model")
plt.plot(T, f2(T, *coefs2), label="square model")
plt.plot(T, f3(T, *coefs3), label="power model")
plt.legend()
plt.show()
```

yield the diagram in Figure 5.10. You can see at a glance that the two nonlinear models reproduce the concave curve much better than the linear one. Which one is better, however, is not immediately obvious. Only a comparison of the corresponding coefficients of determination and the BICs provides clarity:

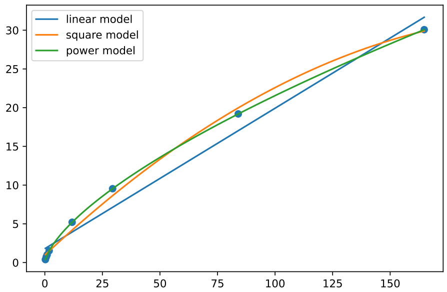


Figure 5.10. Orbital data of the planets and the three regression models

```
# 3.  Model evaluation
# 3.1 Coefficients of determination:
from sklearn.metrics import r2_score
R2_1 = r2_score(y, f1(X, *coefs1))
R2_2 = r2_score(y, f2(X, *coefs2))
R2_3 = r2_score(y, f3(X, *coefs3))

print("Coefficients of determination")
print(" linear:", f"{R2_1:.3%},", "\tsquare:", f"{R2_2:.3%},", "\tpower: ", f"{R2_3:.3%}")

# 3.2 BICs:
def bic(e, k):
    return len(e)*np.log(np.sum(e**2)/len(e)) + k*np.log(len(e))

print(
    " BIC₁ =", f"{bic(y - f1(X, *coefs1), len(coefs1)):.1f},",
    "\t\tBIC₂ =", f"{bic(y - f2(X, *coefs2), len(coefs2)):.1f},",
    "\t\tBIC₃ =", f"{bic(y - f3(X, *coefs3), len(coefs3)):.1f}"
)
```

They have the following values, in percent and rounded to three decimal places:

$$R_1^2 = 97{,}636\%, \qquad R_2^2 = 99{,}618\%, \qquad R_3^2 = 100\%. \tag{5.27}$$

as well as

$$\text{BIC}_1 = 11, \qquad \text{BIC}_2 = -1, \qquad \text{BIC}_3 = -109, \tag{5.28}$$

respectively. By both scorings, especially by the BIC, the power model therefore is by far the most suitable! We can also confirm this visually by plotting the three models in a log-log scale:

```
plt.scatter(T, r)
plt.plot(T, f1(T, *coefs1), label="lineares Modell")
plt.plot(T, f2(T, *coefs2), label="quadratisch Modell")
plt.plot(T, f3(T, *coefs3), label="Potenzmodell")
plt.legend()
plt.xscale("log")
plt.yscale("log")
plt.show()
```

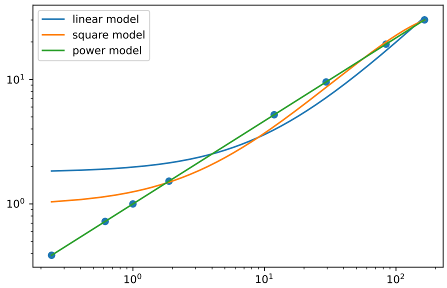


Figure 5.11. The three regression models in log-log scale

In this scale it can be clearly seen that the power model has a linear curve progression and especially the semi-axis values of the three inner planets Mercury, Venus and Earth are modeled completely wrong by the other two models. The power model is thus very accurately actuated on the basis of the observational data and delivers the relations

$$r = T^{2/3} \qquad \Longleftrightarrow \qquad r^3 = T^2 \qquad \Longleftrightarrow \qquad \boxed{\frac{r^3}{T^2} = 1.} \tag{5.29}$$

The last equation is Kepler's third law in the units of the Earth's orbit ($r_{♁} = T_{♁} = 1$). Kepler achieved the derivation of his law incomparably in a much more difficult way than we can do it today: He needed a whole decade for this derivation. At that time, however, the concept of function did not exist at all, and Gauss invented his method of least squares almost 200 years later. Well, and he didn't have Python either. So what seems to be a short example of a regression model based on observationaldata in fact was a scientific masterpiece and made history. About 80 years later Newton designed his groundbreaking theory of gravitation on the basis of Kepler's model, and Einstein explained a tiny error of the model for the orbit of the inner planet Mercury, the "perihelion motion", with his general theory of relativity more than 300 years later. By the way, in Kepler's lifetime the two outer planets Uranus and Neptune were not yet known.

Is the model found overfitted? Having derived a suitable model, a serious question remains open: Is the model overfitted? Is it too much adopted to the training data at hand and does not generalize, as in Figure 5.6 above? Occam's razor gives a clear answer: No! The power model predicts the values almost perfectly, but it only requires a single parameter: It seems as simple as possible, but as complex as necessary. This is clearly reflected in the low BIC value.

5.4.5 Confidence intervals of fitted models

Fitting the parameters θ_i of a regression model can also be viewed as estimating for the unknown "true" values of the parameters. This is because the result provides the best estimate for the unknown parameters using the observed data. In statistics, this is referred to as parameter estimation using a sample.

confidence interval and confidence level

A *confidence interval* indicates the range in which, with a certain probability, the "true" value of a random value (more precisely, a random variable) lies around a sample value. This probability is called *confidence level*[16] and is often denoted as $1 - \alpha$, where α is the probability of error. Confidence intervals are thus another way to measure the goodness of fit of a regression for given data points. We can visualize them well by plotting them as an error band around a regression curve, as shown in Figure 5.12 for different confidence levels.

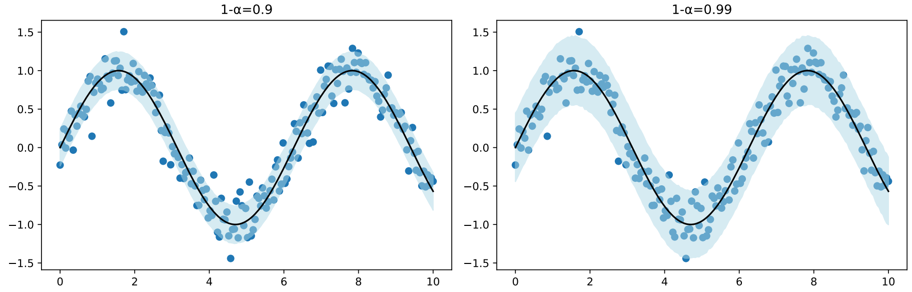


Figure 5.12. Confidence intervals around a regression curve to the confidence levels $1 - \alpha = 0.9$ and 0.99. The larger the confidence level $1 - \alpha$, the wider the confidence interval.

In Python, suitable functions for this purpose are the Numpy function `quantile` and the plot function `fill_between`. Passing to `quantile` a matrix `sims` and a list `[alpha,`

[16] Backhaus et al. (2016):§1.2.4.2.

1-alpha], it returns the two quantiles $q = \alpha$ and $q = 1 - \alpha$. With the optional parameter axis=0, each row of the matrix preds is considered as a single record:

```
x = np.linspace(0,10,100)
y_pred = np.sin(x)                  # supposed regression curve
y = np.random.normal(y_pred,0.2) # simulated data
sims = np.array([np.random.normal(y_pred,0.2) for j in range(1000)])

alpha = 0.01                        # probability of error
u, l = np.quantile(sims, [alpha, 1 - alpha], axis=0)

plt.scatter(x,y)
plt.plot(x,y_pred, color="black")
plt.fill_between(x, l, u, color="lightblue", alpha=0.5)
```

Here y_pred contains the values of the assumed or determined regression curve and y simulated "observed" data scattered around the y values with a standard deviation $\sigma = 0.2$. The matrix sims contains 100 such random simulations. With np.quantile the quantiles $q = \alpha$ and $q = 1 - \alpha$ are stored in the variables u and l and plotted in the plot function plt.fill_between as upper and lower bounds. (Note that the parameter alpha in plot functions represents the opacity of the object to be plotted!) In a regression analysis with curve_fit, the fitted model function must of course be used for the values of y_pred instead of the function np.sin defined statically here, and for sigma the determined standard deviation of the data series y and y_pred, instead of the constant 0.2:

```
y_pred = f(x,*params)
sigma = np.std(y - y_pred)
```

5.5 Problems

Problem 5.1. Under https://www.itl.nist.gov/div898/strd/nls/nls_main.shtml, the NIST provides some challenge datasets for testing robustness and reliability of statistical software with varying levels of difficulty. The "Thurber problem" is classified to be of higher level of difficulty. The data involve semiconductor electron mobility, where the independent variable x is the natural logarithm of the density and the target variable y a measure of electron mobility. The data are available under https://www.itl.nist.gov/div898/strd/nls/data/thurber.shtml, or in Moodle. Test whether Python's curve_fit

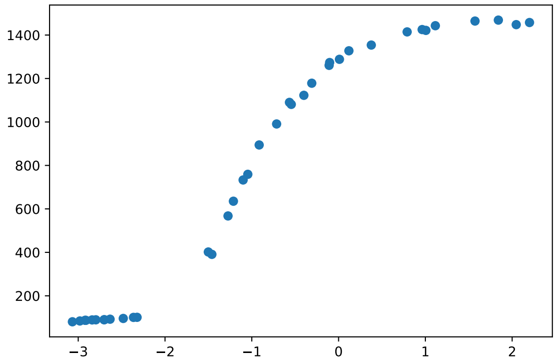


Figure 5.13. Semiconductor electron mobility y versus the log electron density x.

function fits the model

$$f(x, \boldsymbol{\theta}) = \frac{\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3}{1 + \theta_4 x + \theta_5 x^2 + \theta_6 x^3}$$

correctly, with initial values $(1000, 1000, 400, 40, 1, 0.5, 0.05)$. Plot the regression curve along with a scatter plot of the data. Evaluate goodness of fit of the model.

Problem 5.2. [17] The marketing management of a company wants to find out what the relationship is between advertising expenditure and the sales volumes of a particular product. To do this, it randomly allocates different advertising budgets to 15 different sales areas and records the respective sales volumes at the end of the quarter, see table 5.2. Estimate the parameters of the following four regression models for these data. Here, advertising spend should represent the independent variable. Also, use the initial values of the parameters where indicated.

Sales area	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Advertizing expenses	22	8	14	26	12	2	10	24	6	30	16	20	18	4	28
Sales volume	264.9	176.1	222.7	269.3	194.5	101.7	187.9	275.3	148.4	308.3	241.0	265.9	243.4	129.3	288.6

Table 5.2. Data for studying the effect of advertising. Advertising expenditures are given in 1000 €, sales volumes in 1000 kg.

Model	**Function**	**Initial values**
linear model:	$f(x) = a + bx$	–
square root model:	$f(x) = a + b\sqrt{x}$	–
power model:	$f(x) = a + bt^c$	$a = 1, b = 1, c = 0.5$
exponential model:	$f(x) = a + b\,\mathrm{e}^{cx}$	$a = 1, b = -1, c = 0.05$

Evaluate the four estimated models based on their goodness of fit. So which model explains best the relationship between advertising spend and sales success?

[17] Backhaus et al. (2015):S. 27ff.

6
Data analysis with Python

Overview

6.1 Parametric statistical models in Python

An overview of common models and associated Python classes in Python is given in Table 6.1.

Model	**Class (“Estimator”)**	**Module**
linear regression	`LinearRegression()`	`sklearn.linear_model`
nonlinear regression	`SVR(kernel="rbf")`, `curve_fit`	`sklearn.svm`, `scipy.optimize`
regression of time series	`SARIMAX(`$(p,d,q)\times(P,D,Q)_s$`)`	`statsmodels.tsa.statespace.sarimax`
Classification		
logistic regression	`LogisticRegression()`	`sklearn.linear_model`
support vector machines	`LinearSVC()`, `SVC()`	`sklearn.svm`
naive Bayes Classifier	`GaussianNB()`	`sklearn.naive_bayes`
Clustering	`MeanShift()`	`sklearn.cluster`
Louvain methode	`best_partition()`	`community`
neuronal networks	`Sequential()`, `Model()`	`keras.layers`, `keras.models`
dimension reduction		
principal component analysis	`PCA()`	`sklearn.decomposition`
factor analysis	`FactorAnalysis()`	`sklearn.decomposition`
lineare discriminant analysis	`LinearDiscriminantAnalysis()`	`sklearn.discriminant_analysis`
quadratic discriminant analysis	`QuadraticDiscriminantAnalysis()`	`sklearn.discriminant_analysis`

Table 6.1. Common models for data analysis in Python.

6.1.1 Generalized linear models

A *generalized linear model (GLM)*[1] is a model where the target variable y is a linear combination of the features $\boldsymbol{x} = (x_1, \ldots, x_n)$, transformed by an invertible function $h : \mathbb{R} \to \mathbb{R}$:

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = h\,(\theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n). \tag{6.1}$$

The function h is the *response function* of the model, its inverse h^{-1} is called *link function*. Both functions provide the relationship between the linear predictor $f(\boldsymbol{x}, \boldsymbol{\theta})$ and the mean μ of the distribution function which underlies the independant variables. Generalized linear models allow response variables that have arbitrary distributions, rather than simply normal distributions. GLM's especially include the distributions listed in Table 6.2. [2] Regression models such as linear regression, but also classifications such as logistic regression or so-called support vector machines (SVM) are contained in this class. To score a fitted model, a measure called *deviance* is applied. It is defined for generalized linear models by the following equation:[3]

Distribution	Target Domain	Response Function	Unit Deviance $d(\boldsymbol{y}, \hat{\boldsymbol{y}})$
Normal	$y \in (-\infty, \infty)$	$h(x) = x$	$(y - \hat{y})^2$
Bernoulli	$y \in \{0, 1\}$	$h(x) = \dfrac{1}{1 + \mathrm{e}^{-x}}$	$2(y \ln \frac{y}{\hat{y}} + (1 - y) \ln \frac{1-y}{1-\hat{y}})$
Poisson	$y \in \{0, 1, 2, \ldots\}$	$h(x) = \mathrm{e}^x$	$2\,(y \ln \frac{y}{\hat{y}} - y + \hat{y})$

Sources: https://en.wikipedia.org/wiki/Generalized_linear_model, https://bookdown.org/egarpor/PM-UC3M/glm-model.html, https://scikit-learn.org/stable/modules/linear_model.html#generalized-linear-models, https://scikit-learn.org/stable/modules/model_evaluation.html#log-loss

Table 6.2. Some generalized linear models (6.1). Here $\hat{y} = f(\boldsymbol{x}, \boldsymbol{\theta})$.

$$D(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{i=1}^{m} d(y_i, \hat{y}_i) \tag{6.2}$$

where $d(y, \hat{y})$ denotes the *unit deviance*. Some unit deviances are given in Table 6.2. In the Python module `scikit-learn.linear_models` the vector $(\theta_1, \ldots, \theta_n)$ is denoted by `coef_`, the intercept θ_0 by `intercept_`. The models in `scikit-learn.linear_models` differ in the distance function

$$\operatorname{dist}\big(\boldsymbol{y}, f(\boldsymbol{X}, \boldsymbol{\theta})\big) = D\,\big(\boldsymbol{y}, f(\boldsymbol{X}, \boldsymbol{\theta})\big) + r(\boldsymbol{\theta})$$

where $r : \mathbb{R}^k \to [0, \infty)$ is a "penalty function" specified for each model, also called *regularization function*. From the geometrical point of view, the distance function measures the geometric distance between the m model values $f(\boldsymbol{X}, \boldsymbol{\theta})$ of the scatter points and the m target values $\boldsymbol{y}$ of the sample. By a model fit, this function is minimized. The most common models are listed in Table 6.2. They are implemented in Scikit-learn as follows:[4]

- `LinearRegression`: Assuming a normal distribution for y, we obtain from the first row in Table 6.2 the unit deviance and thus the deviance D by Equation (6.2). With a penalty function $r(\boldsymbol{\theta}) = 0$ we then have the distance function

$$\operatorname{dist}\big(\boldsymbol{y}, f(\boldsymbol{X}, \boldsymbol{\theta})\big) = |\boldsymbol{y} - f(\boldsymbol{X}, \boldsymbol{\theta})|^2, \tag{6.3}$$

[1] https://scikit-learn.org/stable/modules/linear_model.html#generalized-linear-models

[2] More distributions are given in https://en.wikipedia.org/wiki/Generalized_linear_model#Link_function

[3] https://bookdown.org/egarpor/PM-UC3M/glm-deviance.html

[4] cf. https://scikit-learn.org/stable/modules/linear_model.html

i.e., simply by mean squares. This is the RSS from usual linear regression, cf. (5.4).

- `Ridge(alpha=`α`)`, often called “L2”: The distance is given by

$$\operatorname{dist}\big(\boldsymbol{y}, f(\boldsymbol{X},\boldsymbol{\theta})\big) = |\boldsymbol{y} - f(\boldsymbol{X},\boldsymbol{\theta})|^2 + \alpha\,|\boldsymbol{\theta}|^2 \tag{6.4}$$

  by mean squares with regularization factor α on the magnitude of the parameter $\boldsymbol{\theta}$. The regularization factor prevents a pathological build-up of the parameter values.

- `Lasso(alpha=`α`)`, often called “L1”. Here the distance function is defined by

$$\operatorname{dist}\big(\boldsymbol{y}, f(\boldsymbol{X},\boldsymbol{\theta})\big) = \frac{1}{2m}|\boldsymbol{y} - f(\boldsymbol{X},\boldsymbol{\theta})|^2 + \alpha\,|\boldsymbol{\theta}|, \tag{6.5}$$

  similar to Ridge. However, it is more suitable for features with many zero values.[5]

- `LogisticRegression`: Although a linear classification model, logistic regression is a generalized linear model with Bernoulli distribution (binary values for y). For the default parameter value `penalty='l2'` with $r(\boldsymbol{\theta}) = \frac{1}{2}|\boldsymbol{\theta}|^2$ the distance is

$$\operatorname{dist}\big(\boldsymbol{y}, f(\boldsymbol{X},\boldsymbol{\theta})\big) \;=\; -y\ln h(\boldsymbol{X\theta}) - (1-y)\ln h(\boldsymbol{X\theta}) \;+\; \tfrac{1}{2}|\boldsymbol{\theta}|^2$$

  between the data values and the model values is minimized.[6] The deviance is derived from the second row of Table 6.2. (Here the binary character of y is exploited since for $y = 0$ both summands in the unit deviance vanish.)

In Scikit-learn there are also implemented several SVM’s, the most common are the following:[7]

- `LinearSVC`: linear support vector classification.

- `SVC`: nonlinear support vector classification.

Moreover, the following naive Bayes classifiers are implemented:[8]

- `GaussianNB`: Gaussian distribution (continuous target values)

- `BernoulliNB`: multivariate Bernoulli distribution (only binary target values)

- `MultinomialNB`: multinomial models (several discrete target values)

- `ComplementNB`: Complement naive Bayes classifier (similar to `MultinomialNB`, appropriate for very ununiformly distributed data)

[5] cf. `https://scikit-learn.org/stable/auto_examples/applications/plot_tomography_l1_reconstruction.html`

[6] `https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression`

[7] cf. `https://scikit-learn.org/stable/modules/svm.html`

[8] Cf. `https://scikit-learn.org/stable/modules/classes.html#module-sklearn.naive_bayes`

6.1.2 Choice of the appropriate model class

Often the most difficult step in solving a problem in machine learning is to find the appropriate model. This is not only due to the large selection from the "zoo" of models in Scikit-Learn, but rather to identify the exact problem behind it. Is it a classification problem, or rather a clustering problem? Is it possible to apply dimensionality reduction?

In the Scikit-Learn tutorial, the decision tree shown in Figure 6.1 is given as a rough guide to action. For us it is not relevant to the last detail, because we can only give an introduction here in this lecture notes and cannot go in depth that far. Nevertheless, it can serve us well as a first orientation framework – and as a reference for further specializations. Above all, it is a good guidance to find the actual problem class to work on: Is it a regression, a classification, a clustering, or a dimension reduction?

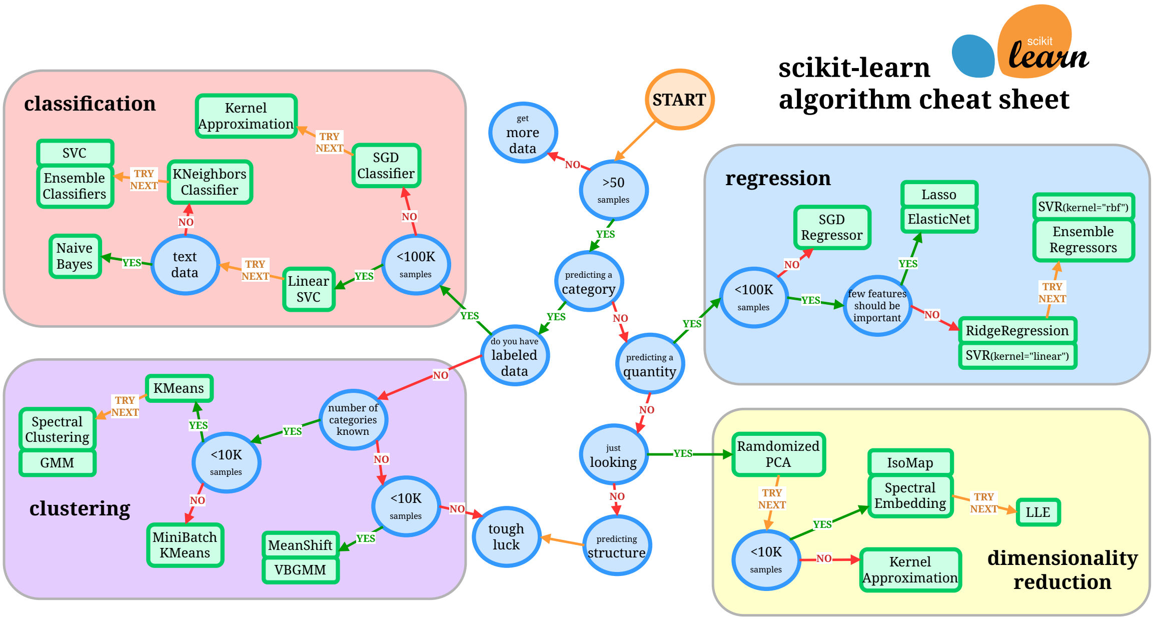


Figure 6.1. Decision tree to choose an appropriate model. Quelle: `https://scikit-learn.org/stable/tutorial/machine_learning_map/`

6.2 Principal component analysis

The idea of principal component analysis (PCA) is to summarise the information of a dataset into its principal components. These components are linear combinations of the variables in their most "interesting" direction. Here "interesting" means the direction of most variance. This is similar to a linear regression but instead of projecting the results onto a line that uses x to capture as much information as possible about y, we are using both variables trying to capture as much information as possible in the x *and* the y direction that has the most variance. This difference between the variable loadings of a principal component line and the variable weights of a regression line, due to the different notions of distance, is sketched in Figure 6.2.

directions of most variance

Now, whereas the differences between the x-values and the regression line, the residuals, contribute to the mean error, the differences between the x-values and the principal component line determines the remaining values that contributes to the *second* principal component. The then remaining value contribute to the *third* one, and so on. In principle,

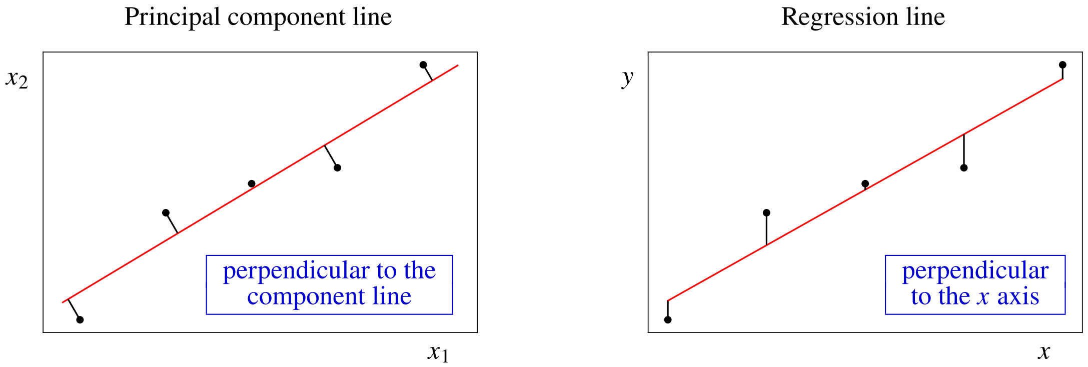


Figure 6.2. The subtle difference between the notions of distance for PCA and regression.

there are as many principal components as features (as long as we have more observations than features, i.e., $m > n$). With the notations of Equations (3.2) and (5.2) each single observation of the n features is given by a row vector $\boldsymbol{X}_i = (x_{i1}, \ldots, x_{in})$, i.e., the j-th principal component is given by[9]

$$z_{ij} = \phi_{1j}\, x_{i1} + \phi_{2j}\, x_{i2} + \ldots + \phi_{nj}\, x_{in} \qquad (i = 1, \ldots, m,\ j = 1, \ldots, n) \tag{6.6}$$

where the numbers $\phi_{ij} \in \mathbb{R}$, $i = 1, \ldots, n$, are called the *loadings* of the n features with respect to the principal component and solve the optimization

$$\max_{\phi_{1k},\ldots,\phi_{nk}} \left\{ \sum_{i=1}^{m} \left(\sum_{j=1}^{n} \phi_{jk}\, x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^{n} \phi_{jk}^2 = 1 \tag{6.7}$$

for each observation $i = 1, \ldots, m$. The values z_{ik} in Equation (6.6) are called *scores* of the m observations, and the vectors $\boldsymbol{Z}_j = (z_{1j}, \ldots, z_{mj})^{\mathrm{T}}$ are called *score vectors*. The loading vector $\boldsymbol{\phi}_j = (\phi_{1j}, \ldots, \phi_{nj})$ defines the direction of the j-th principal in feature space along which the data vary the most, up to the principal components smaller than j, all direction being orthogonal to each other.[10] Once we successively have computed the principal components, starting from $k = 1$, then going to $k = 2$, and so on, until we reach $k = K$ for some $K \leqq n$, we can rewrite the loading vectors as a matrix and obtain the following relation to the score vectors $\boldsymbol{Z}_k$:

$$\begin{pmatrix} | & & | \\ \boldsymbol{Z}_1 & \cdots & \boldsymbol{Z}_K \\ | & & | \end{pmatrix} = \begin{pmatrix} — & \boldsymbol{X}_1 & — \\ & \vdots & \\ — & \boldsymbol{X}_m & — \end{pmatrix} \cdot \Phi^{\mathrm{T}} \qquad \text{with} \qquad \Phi^{\mathrm{T}} = \begin{pmatrix} | & & | \\ \boldsymbol{\phi}_1 & \cdots & \boldsymbol{\phi}_K \\ | & & | \end{pmatrix} \tag{6.8}$$

where $\boldsymbol{X}$ is the matrix in Equation (3.2). In Scikit-learn the matrix Φ is called “components”,[11] so we will refer to it as the component matrix, or load matrix. We can plot the principal components pairwise against each other to view the data with respect to them. Such a projection of the data on a pair of principal components, together with the positions of the feature vectors X_i, is called a *biplot*.[12] Besides the observations themselves, the feature variables are drawn as arrows from the origin. The coordinates of each feature

biplot

[9] James et al. (2013):p. 376.

[10] From a mathematical point of view, the n loading vectors $\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_n$ are the ordered sequence of eigenvectors of the $n \times n$ matrix $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$, given the notation of (3.2), and the variances of the components are its eigenvalues. (Footnote on James et al. (2013):p. 377)

[11] Géron (2017):pp. 213–214.

[12] James et al. (2013):p. 377.

with respect to the principal components are given by the respective row of the load matrix Φ, or in Scikit-learn by one of the columns of the component matrix Φ. A biplot is given below in the right-hand panel of Figure 6.3.

The property of the first principal component being the line in k-dimensional feature space that is *closest* to the m observations, as indicated in Figure 6.2, can be generalized. For instance, the first two components of a data set span the plane that is closest to the m observations. Moreover, the first three principal components of a data set span the three-dimensional space (as a hyperplane of a higher-dimensional space for $k > 3$) closest to the m observations, and so forth.[13] Therefore, the main goal of PCA usually is not to map the n feature axes to n principal component axes, but to find a *lower dimensional* representations of the observations that explain a good fraction of the variance. Therefore, only the first few principal components should be regarded. The crucial question then is: How much of the information in a given data set gets lost by projecting the observations in the first few principal components?

scree plot

There is no definite method or criterion to determine, how many principal components are necessary such that "enough" information is projected onto them. However, a common step to decide this question is to visualize the proportions of explained variances by a *scree plot*.[14]

For a nice and humourous introduction to PCA see Harriet Mason's post `https://numbat.space/posts/pca/`.

scaling data

Before PCA can be performed, the feature variables should be centered to have mean zero. Moreover, since the variables usually are individually scaled – one feature may be in units of currency, another one may be in tons or kg – they should be scaled to standard deviation.[15] In Scikit-learn, scaling the data is conveniently done with the Transformer `StandardScaler` of the module `sklearn.preprocessing`.

6.2.1 A case study for principal component analysis

The World Happiness Report is published annually by the United Nations. For 2021, e.g., it is available at `https://worldhappiness.report/ed/2021/`, Appendices & Data, Data for Figure 2.1. In the report, the happiness-index is calculated for almost all nations of the world as a key figure from various economic and social characteristic data such as social benefits, life expectancy, degree of freedom or corruption. With the following sample program we want to investigate the question of which of these characteristics have the most significant influence on the satisfaction of a population based on the data.

First visualizations of the principal components

As a first step in data analysis it is common to visualize the underlying data. For this purpose we first import the data as usual, before we perform the principal component analysis with Scikit-learn. According to the discussion above, this will yield us the calculation of the principal components as a matrix Φ, showing how the features contribute to the first few principal components.

```
%matplotlib notebook
import matplotlib.pyplot as plt, numpy as np, pandas as pd, mpl_toolkits.mplot3d
from sklearn.preprocessing   import StandardScaler   # scales data to standard deviation
from sklearn.decomposition   import PCA              # principal component analysis
```

[13] James et al. (2013):pp. 380–381.

[14] James et al. (2013):pp. 382–383.

[15] Géron (2017):p. 213; James et al. (2013):pp. 381–382.

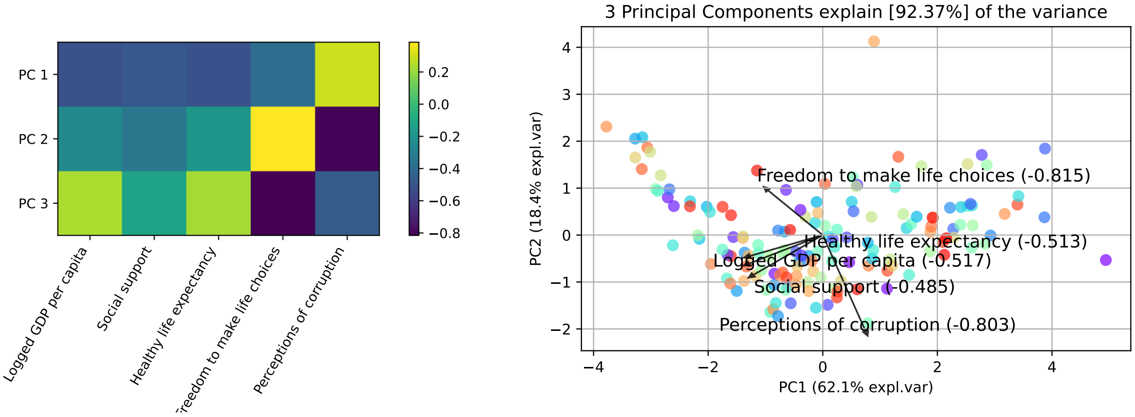


Figure 6.3. Left: Heatmap of the loads of the first three principal components with respect to the features. Right: Biplot of the five features projected into the plane spanned by the first two principal components.

```
from sklearn.linear_model    import LinearRegression # linear regression

# 1.  Import data as a Pandas DataFrame and preprocess them for scikit-learn:
df = pd.read_csv("./datasets/happiness-report-2021.csv", sep='\t')  # loads CSV to Pandas
features = ["Logged GDP per capita", "Social support", "Healthy life expectancy",
  "Freedom to make life choices", "Perceptions of corruption"]
target = "Happiness"                # dependent variable
X = np.c_[df[features]]             # extracts feature values as a matrix
y = np.c_[df[target]]               # extracts target values as a one-column matrix

# 2.1 Scaling the data
model1 = StandardScaler()
model1 = model1.fit(X)
X_scaled = model1.transform(X) # compute and store the scaled data

# 2.2 Principal component analysis of the feature data
model2 = PCA(3) # for example, three principal components ...
model2 = model2.fit(X_scaled, y)
X_trans = model2.transform(X_scaled) # compute and store the projected data

# 2.3 Print and plot the principal components as a heat map
print(model2.components_)
plt.figure(figsize=(7,4))
plt.imshow(model2.components_)
plt.colorbar()
plt.xticks(range(len(features)), features, rotation=60, ha="right")
yticks = [""] * len(model2.components_[:,0])
for i in range(len(yticks)):
    yticks[i] = "PC " + str(i+1)
plt.yticks(range(len(yticks)), yticks)
plt.show()

# --------------------------------------
# 3 Biplot:
from pca import pca

model = pca(n_components=3, verbose=0)
results = model.fit_transform(X_scaled, col_labels=features, row_labels=df['Country name'],
    verbose=0)
```

```
fig, ax = model.biplot(
    figsize=(10,5), PC=[0,1], alpha_transparency=0.7, SPE=True,
    cmap="rainbow", color_arrow='black', verbose=0, label=None,
)
fig.show()
```

The variances – or in other words the correlations[16] – of the individual characteristics on the principal components are given by the matrix <

$$\Phi = \begin{pmatrix} -0.517 & -0.485 & -0.513 & -0.386 & 0.293 \\ -0.244 & -0.340 & -0.180 & 0.385 & -0.803 \\ 0.238 & -0.123 & 0.225 & -0.815 & -0.462 \end{pmatrix} \tag{6.9}$$

Each column yields the loading vector of the corresponding feature. The loadings can be represented by a heat map as shown in Figure 6.3 on the left. The larger the absolute amount of variance, the greater is the influence of the particular characteristic. Negative amounts are negatively correlated with the principal component, but this has no deeper meaning in principle since only the direction of the principal component is relevant. Figure 6.3 on the right depicts the happiness index of each country plotted against the first two principal components. The biplot is generated by the library pca. Here the coordinates of the five feature vectors with respect to the three principal components are given by the columns of the component matrix Φ.

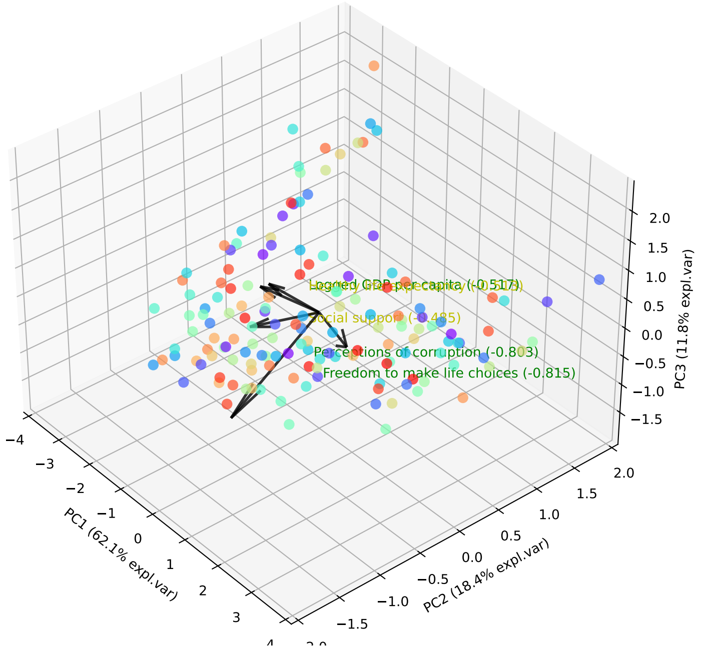


Figure 6.4. Biplot with the first three principal components.

To get an intuition of the relationship of the principal components to the data and to the feature vectors, as well as and to understand the underlying geometry, we could plot a three-dimensional biplot (strictly speaking, a "triplot" instead) with the following code snippet.

```
# 3.2 Three-dimensional biplot
```

[16] Backhaus et al. (2016):p. 394.

```
fig, ax = model.biplot(
    figsize=(14,7), PC=[0,1,2], d3=True, alpha_transparency=0.7,
    cmap="rainbow", color_arrow='black', verbose=0, label=None, legend=False
)
ax.set_xlim(-4, 4)
ax.set_ylim(-2, 2)
ax.view_init(azim=-40, elev=36) # position of camera
fig.show()
```

This gives the plot in Figure 6.4.

Determining the number of components

Now we can tackle problem to determine the number of principal components to characterize the data good enough. A first and not unusual step is to use some hints by regarding the scree plot. For this purpose the following code snippet can be implemented:

```
model = PCA()  # enable all possible principal components ...
model = model.fit(X_scaled,y)
print("Explained variance ratio:", model.explained_variance_ratio_)
plt.figure(figsize=(6,4))
pc_values = np.arange(model.n_components_) + 1
plt.plot(pc_values, model.explained_variance_ratio_, 'o-')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Explained Variance')
plt.xticks(pc_values)
plt.show()
```

This gives the plot in Figure 6.5. Since there are five features (and mopre than 140 data points), the maximum number of principal components is five. The scree plot simply

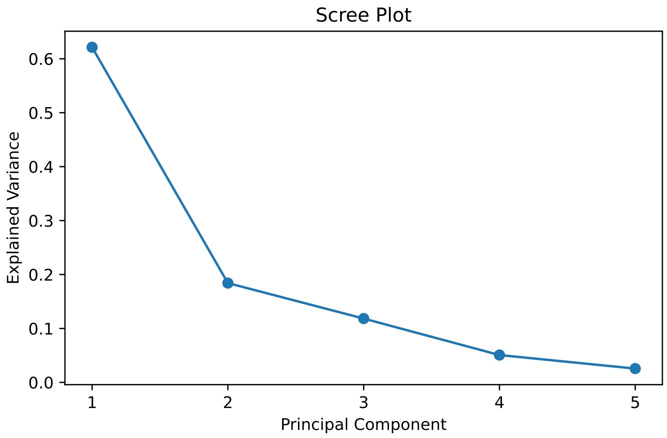


Figure 6.5. Scree plot of the possible five principal components. Its “elbow” may be located at the second component.

plots the explained variance of each principal component, given by the model attribute `explained_variance_ratio`. The scree plot does not show a definite “elbow”, but we may located it at the second component. Therefore, two principal components may be considered to be “enough” to explain the relevant feature combinations. Considering the output of the program

```
Explained variance ratio: [0.6212864  0.18409387 0.11832235 0.0507004  0.02559698]
```

we conclude that the first two components explain 62.13+18.41 = 80.54 % of the variances.

General overview with scatterplot matrices

A convenient way to give a visualized overview over multidimensional data is to use a *scatterplot matrix*. Here every numerical variable is plotted against every other numerical variable, arranged in a tabular form.[17] In Python, scatterplot matrices can be generated from a Pandas DataFrame by the Pandas function `scatter_matrix` in the module `pandas-plotting`. Another possibility is the module `plotly.express`, which is more convenient for Numpy arrays as in Scikit-learn. If `X_trans` represents the observations projected onto the space of the first three principal components, as done in step 2.2 above by `model2`, we could depict the $3 \cdot (3-1) = 6$ plots by the following snippet:

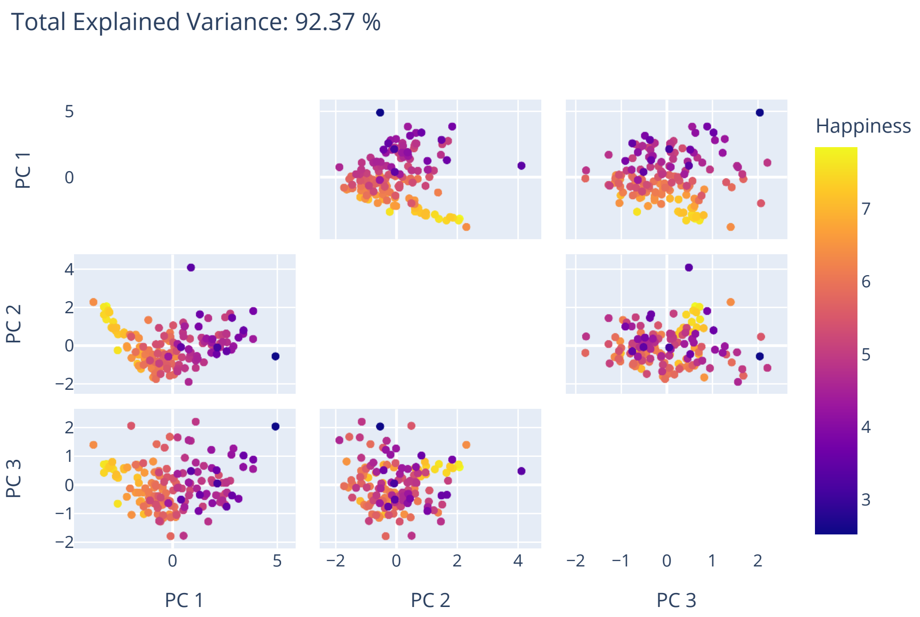


Figure 6.6. Scatterplot matrix of the happiness indices of each country, plotted against each pair of the first three principal components PC1, PC2, and PC3.

```
import plotly.express as px

n_components = model2.n_components
total_var = model2.explained_variance_ratio_.sum() * 100

labels = {str(i): f"PC {i+1}" for i in range(n_components)}
labels['color'] = target

fig = px.scatter_matrix(
    X_trans,
    color=y.ravel(),
    dimensions=range(n_components),
    labels=labels,
    title=f'Total Explained Variance: {total_var:.2f} %',
```

[17] cf. Géron (2017):pp. 57–58; James et al. (2013):p. 50.

```
)
fig.update_traces(diagonal_visible=False)
fig.show()
```

Since the diagonal would give two-dimensional plots showing all observations on a straight line, we suppress them with the option `diagonal_visible=False`. The result of this program gives the scatterplot matrix in Figure 6.6.

6.2.2 Evaluating of predictions

Predictive models can predict target values. In Scikit-Learn each predictive model has a method `predict`, which calculates the target values corresponding to the input feature values `X_test` according to the model:

```
y_pred = model.predict(X_test)
```

To evaluate the predictions the method `score` can be applied, which yields a relevant key figure measuring how well values calculated from the data `X_test` match the actual values `y_test`.

```
print( model.score(X_test, y_test) )
```

Thus, one should not use all available data as training data, but use a part of it as test data. The method `train_test_split` is suitable for this purpose:

```
from sklearn.model_selection import train_test_split
# Select randomly 30% of the data as test data (i.e., 70% training data):
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Other ways to evaluate forecasts are listed at `https://scikit-learn.org/stable/modules/model_evaluation.html`.

6.3 The pipeline: automating data analysis

If we want to execute several models and data transformations one after the other, one can use a so-called pipeline. A *pipeline* is a sequence of models with the methods `fit` and `transform`, the last model of which requires only the `fit` method. It is used to automate elaborate modeling processes and thus to run through entire modeling scenarios. For example, a pipeline may include a scaler, a principal component analysis, and a linear regression model can be connected in series:

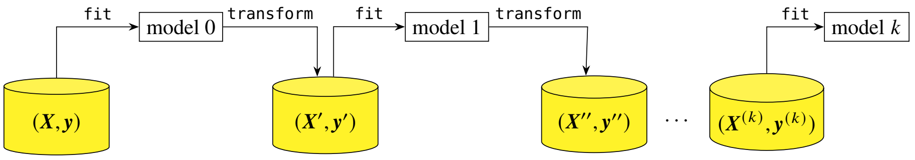


Figure 6.7. Principle of a pipeline in Scikit-learn

```
pipe = make_pipeline(StandardScaler(), PCA(2), GaussianNB())
```

In the `steps` attribute of the pipeline the models are listed, namely as 2-tuple `('name', model)`. E.g. you get the standard scaler and the principal component model with

```
pca = pipe.steps[0][1]
```

(i.e., start with index 0 and end with index 1). Since the last model of the pipeline contains a method `predict` a prediction of the target values can be for further test data can be created.

```
y_pred = pipe.predict(X_test)
```

Example 6.1. *(Principal component analysis and linear regression with pipeline)* If we want to perform a linear regression instead of a Gaussian naïve Bayesian classification as in example 6.2, a simple replacement of the model in the pipeline is sufficient. However, since a linear regression expects real values of the target variables, categorization of the data as in step 3 in Example 6.2 should be omitted.

```
%matplotlib notebook
import matplotlib.pyplot as plt, numpy as np, pandas as pd, mpl_toolkits.mplot3d
from sklearn.model_selection import train_test_split # splits data into training and test data
from sklearn.pipeline        import make_pipeline    # makes a pipeline
from sklearn.preprocessing   import StandardScaler   # scales data to standard normal distribution
from sklearn.decomposition   import PCA              # principal component analysis
from sklearn.linear_model    import LinearRegression # linear regression

# 1.  Import data as a Pandas DataFrame and preprocess them for scikit-learn:
df = pd.read_csv("./datasets/happiness-report-2021.csv", sep='\t')  # loads CSV file as a Pandas dataframe
features = ["Logged GDP per capita", "Social support", "Healthy life expectancy", "Freedom to make life choices",
            "Perceptions of corruption"]
target = "Happiness"                # dependent variable
X = np.c_[df[features]]             # extracts feature values as a matrix
y = np.c_[df[target]]               # extracts target values as a one-column matrix

# 2.  Choose by random 30 % of data as test data, i.e., 70 % as training data:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)

# 3.  Fit and predict with a pipeline of scaling, PCA, and linear regression:
pipe = make_pipeline(StandardScaler(), PCA(2), LinearRegression())
pipe.fit(X_train, y_train)

# 4.  Print model score:
print("score (train values): ", f"{pipe.score(X_train, y_train):.2%}")
print("score (test values):",f"{pipe.score(X_test, y_test):.2%}")

# 5.  Plot 3D scatter plot:
from mpl_toolkits.mplot3d import Axes3D

# 5.1.  Choose PCA model from pipeline and project data onto the principal components:
X_scaled = pipe.steps[0][1].fit_transform(X) # scaled data
X_trans  = pipe.steps[1][1].fit_transform(X_scaled) # Dimensionsreduzierung auf die Hauptkomponenten
y_pred  = pipe.predict(X)               # Vorhersagewerte der Pipeline ...

# 5.2.  Plot 3D scatter diagram:
fig = plt.figure(figsize=(7,5))
ax = fig.add_subplot(111, projection="3d")
ax.scatter(X_trans[:,0], X_trans[:,1], y, marker="o", c='red', label='actual values')
ax.set_xlabel("PC 1"), ax.set_ylabel("PC 2"), ax.set_zlabel(target)
ax.view_init(azim=-60, elev=20) # position of camera
plt.tight_layout()
plt.show()

# 5.3.  Plot regression plane witht min/max of the transformed data:
x0 = np.linspace(X_trans[:,0].min(), X_trans[:,0].max(), num=2)
x1 = np.linspace(X_trans[:,1].min(), X_trans[:,1].max(), num=2)
xx0, xx1 = np.meshgrid(x0,x1) # 2x2 - Gitter
X0, X1 = xx0.ravel(), xx1.ravel()
yy = pipe.steps[2][1].predict(np.c_[X0, X1]).ravel()  # Prediction values in the regression plane
ax.plot_trisurf(X0, X1, yy, linewidth=0, alpha=0.3)
plt.tight_layout()
plt.show()
```

The resulting scatter plot is shown in Figure 6.8. On the right you can see the scatter plot with the regression plane on which the values predicted with the linear regression (here marked as x) lie. □

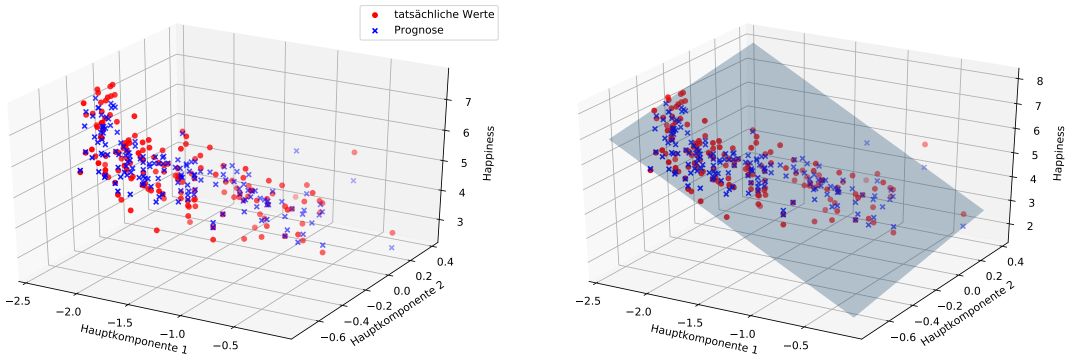


Figure 6.8. Scatter plot of the projection on the two principal components of the actual data and the values predicted after linear regression (example 6.1). The predicted values are in the regression plane (right).

As discussed at the beginning, the pipeline mechanism can easily be used to change concatenations of models. The following example demonstrates this possibility by replacing the linear regression with a Gaussian naive Bayes classifier. Here this is still done manually, but of course we can easily imagine programs that mutually exchange this mechanism for a whole list of models in an automated way, and thus in a sense allow testing of whole chains of models.

Example 6.2. *(Principal Component Analysis and Classification with Pipeline)* In the following program, a principal component analysis is performed followed by a pipelined classification.

```
%matplotlib notebook
import matplotlib.pyplot as plt, numpy as np, pandas as pd, mpl_toolkits.mplot3d
from sklearn.model_selection import train_test_split # teilt Daten in Training und Test
from sklearn.preprocessing   import KBinsDiscretizer # teilt stetige Werte in Kategorien
from sklearn.pipeline        import make_pipeline    # erstellt eine Pipeline
from sklearn.preprocessing   import StandardScaler   # normalverteilte Skala für alle Daten
from sklearn.decomposition   import PCA              # Hauptkomponentenanalyse
from sklearn.naive_bayes     import GaussianNB       # Gauss'scher naiver Bayes-Klassifikator

df = pd.read_csv("./datasets/happiness-report-2021.csv", sep='\t')  # lädt CSV-Datei in Pandas
merkmale = ["Logged GDP per capita", "Social support", "Healthy life expectancy", "Freedom to make life choices", "
     Perceptions of corruption"]
ziel = "Happiness"                  # abhängige Variable
X = np.c_[df[merkmale]]             # extrahiert die Merkmalswerte als Matrix
y = np.c_[df[ziel]]                 # extrahiert die Zielwerte

# Wähle per Zufall 30 % der Daten als Testdaten aus, d.h.  70 % als Trainingsdaten:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)

# Teile die Trainings- und Testdaten der abhängigen Variablen in 3 Kategorien („Bins“):
y_train = KBinsDiscretizer(n_bins=3, encode='ordinal').fit_transform(y_train).ravel()
y_test  = KBinsDiscretizer(n_bins=3, encode='ordinal').fit_transform(y_test).ravel()

# Fitting und Prognose mit Pipeline aus Skalierung, PCA und Gauss Naive-Bayes-Klassifizierer:
pipe = make_pipeline(StandardScaler(), PCA(2), GaussianNB())
pipe.fit(X_train, y_train)

# Vorhersagegenauigkeit angeben:
print("score (train values): ", f"{pipe.score(X_train, y_train):.2%}")
print("score (test values):",f"{pipe.score(X_test, y_test):.2%}")

# -- 3D-Streudiagramm:
X_trans = pipe.steps[0][1].transform(X) # Dimensionsreduzierung auf die Hauptkomponenten!
y_pred  = pipe.predict(X)               # Vorhersagewerte der Pipeline

plt.figure(figsize=(8,5)); ax = plt.axes(projection="3d")
ax.scatter(X_trans[:,0], X_trans[:,1], y, c=y_pred)
ax.set_xlabel("Hauptkomponente 1"); ax.set_ylabel("Hauptkomponente 2"); ax.set_zlabel(ziel)
plt.show()
```

In detail, the following steps are performed in the program:

1. The CSV file is imported as a Pandas DataFrame and the data is stored in Numpy arrays `X` and `y`.
2. The data `X` and `y` are separated into training and test data.
3. Nominal target data `y_train` and `y_test` for training and testing are ordinally divided into 3 categories.
4. A pipeline `pipe` is formed from a standardizer, principal component analysis, and a Gaussian naive Bayes classifier, and fitted with the training data.
5. A vector `y_pred` with the predicted values of the test data is determined from `X_test`.
6. The prediction accuracies of the naive Bayes classifier are output.

The program thus outputs the following results:

```
score (train values):  76.95%
score (test values):  78.59%
```

Instructions for plotting a scatter plot with the data projected onto the principal components are then performed:

7. The original feature data `X` with its 5 dimensions are projected geometrically onto the two principal components and accordingly the data for the entire feature expressions are predicted using the pipeline and stored in the array variable `y_pred`.
8. The three-dimensional scatter plot is plotted. Here, the color of a data point is determined by the value of the category determined by the Gaussian NB classifier using the parameter `c` in `ax.scatter`.

The resutling scatter plot is shown in Figure 6.9. In the scatter plot, the individual countries are represented as data points that differ in color based on their respective class determined by the NB classifier. In this way, one can see the distribution of the three categories in step 3 in relation to the two principal components. □

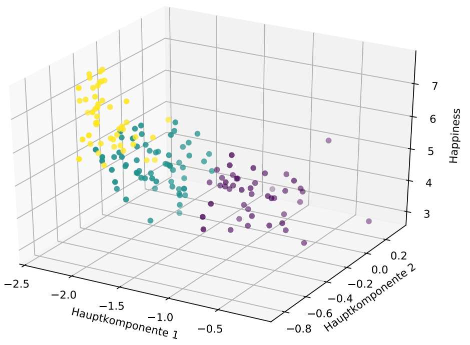


Figure 6.9. Scatter plot of the projection on the two principal components according to a Naive Bayes classification (example 6.2). The classes are shown in different colors.

Part III

Time series

7
Time series analysis

Overview

7.1 introduction

A time series is a time-dependent finite sequence of real-valued data points $y_0, y_1, y_2, \dots, y_t \in \mathbb{R}$, which is given an index reflecting the passage of time. An individual data point is an observed or measured value of a certain process. Typical examples of such processes are the development of Stock market prices, election intention polls, weather observations or temperature values. A time series is often plotted against time by connecting the data points are connected to a curve (Figure 7.1).[1] The points in time to which data points are assigned can be arranged equidistantly, i.e., at constant intervals (for example every 5 seconds), in other regularity (for example every working day), or irregular. Frequently equidistant time series are considered, and the reciprocal value of the duration between two measured values is in this case called *sampling rate*.

sampling rate

A time series can take a single numerical value at a time, it is then called univariate or scalar. If, on the other hand, it assumes in each case a tuple of several numerical values it is called multivariate. Typical time series arise from the interaction of regular and random causes. The regular causes may vary periodically or seasonally and may contain long-term trends. Random influences are referred to as noise.

univariate, multivariate

noise

Time series are often analyzed to predict their future development. Time series analysis is a special form of regression analysis and tries to identify and model patterns or regularities in the development of time series by means of statistical methods. Such

time series analysis

[1] cf. also: Burke et al. (2018); Kalvelage (2018).

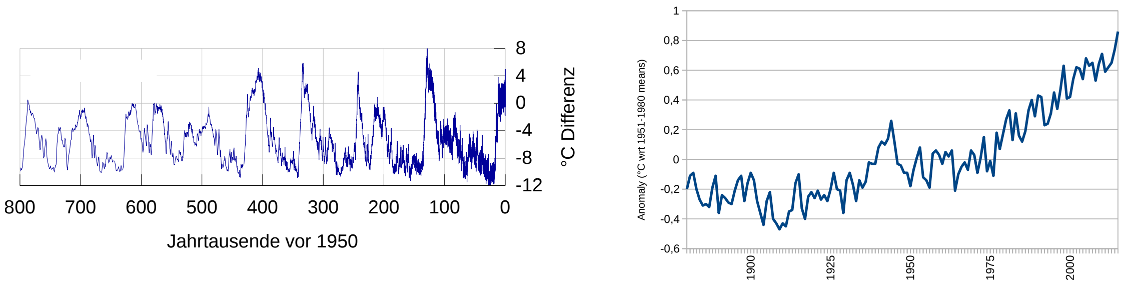


Figure 7.1. Left: Temperature changes in Antarctica over the last 800 000 years compared to the average of the last 1000 years, carried out by EPICA (Jouzel et al., 2007). Right: Global temperature anomalies 1880 – 2015 (to the 1951–1980 mean), Data source: `http://cdiac.ornl.gov/ftp/trends/temp/hansen/gl_land_ocean.txt`; see also `https://xkcd.com/1732/`

regularities are of quite principal interest, since given scientific theories often let expect certain regularities.

The following textbooks on time series are recommended for further information and are specifically referenced later at appropriate places in this lecture notes: Deistler and Scherrer (2018), Kreiß and Neuhaus (2006), Neusser (2011), Palma (2016), and Vogel (2015).

7.1.1 Time series and machine learning

Time series play a rather special role in machine learning. In many problems, the dependent variable y, i.e., what we want to predict, depends on very clear inputs, such as pixels of an image, words in a sentence, the characteristics of a person's buying behavior, etc. In time series, these independent variables are often unknown. For example, in stock markets, we have no unique independent variables to which we can apply a model. Do stock markets depend on the characteristics of a company or the characteristics of a country? Or on the mood of the news? Certainly we can try to find a relationship between these independent variables and stock market outcomes, and perhaps we can find some good models that capture these relationships. The point is that these relationships are not very clear, nor are the independent data readily available.

In the classical analysis of time series, one takes the radically different approach of not looking for or assuming independent variables, but inferring their properties from the past performance of the time series. I.e., the only independent variable of a time series is time.

7.1.2 Model forms of time series

Usually time series analysis is considered in combination with forecasting. However, strictly speaking forecasting and the analysis of time series are two distinct activities. A forecast is a view in an uncertain future. Time series is a description of the past. The main precursors to the forecasting activity are the construction of a suitable model based upon analysis of the historical development of the series and utilisation of information relevant to the series' likely future development.

That is not to say that the modelling and analysis process is concluded once a forecast has been produced. Eventually the quantity being forecast will become known, this value providing new information to incorporate into the analysis. Thereafter, in a typical live situation, a forecast for a subsequent time will be produced and the forecast-observation cycle repeated.

Three basic model forms encompass the great majority of time series and forecasting situations. They are:

- trend models,
- seasonal models, i.e., models for systematic cyclical variation, and
- regressions, i.e., models with influential or causal variables.

Trend models are the simplest component models. They represent a system with a straightforward linear progression: growing, decreasing, or staying roughly constant. Seasonal models provide the mechanism to model systematic cyclical variation. This kind of variation is often present in commercial series, and typically related to the passage of the seasons as the earth orbits the sun during the course of a year. Regressions on explanatory variables are potentially the most valuable models because they may incorporate much external information.[2]

A forecast is a statement about an uncertain future. It is a belief, not a statement in fact. Representing uncertainty in scientific analysis is the province of probability, its practical application the domain of statistics. Whenever we make a forecast we actually make a statement of probability, or more generally, state a probability distribution that quantifies the nature of our uncertainty. Any and every forecast is predicted upon a fount of knowledge; forecasts are therefore conditional probability statements, the conditioning being in the existing state of the knowledge. Knowledge is available in several forms, a useful classification being into historical information and professional wisdom or expertise.[3]

7.2 Stochastic processes as the basis of time series

In stochastics, the data points $y_0, y_1, \ldots, y_t$ of a time series are regarded as a sample or concrete realization of a stochastic process, i.e., as a sequence of random variables $(Y_t)_{t \in \mathbb{T}}$ with an infinite index set $\mathbb{T}$. We have already learned about the notion of random variable in section 1.1 on page 7.

For more details on random variables in the context of time series, see e.g. Vogel (2015:pp. 19–21).

Definition 7.1. Let $\mathbb{T}$ be an infinite subset of the set of real numbers, e.g. $\mathbb{N}_0$, $\mathbb{Z}$ or $\mathbb{R}_0^+ = [0, \infty)$. Then a *stochastic process* is a family

stochastic process

$$(Y_t)_{t \in \mathbb{T}}$$

of infinitely many random variables on a common probability space.

We can think of a stochastic process as a random choice of an event ω from the given sample space Ω, and our observations are a time window $\mathbb{T}$ from the sequence of realizations (y_t) with $y_t = Y_t(\omega)$. The sequence of numbers $(y_t)_{\mathbb{T}}$ is called a *realization* or a *trajectory* of the stochastic process.

trajectory

Definition 7.2. A stochastic process $(Y_t)_{t \in \mathbb{Z}}$ is called *stationary*, if the following statistical functions are all independent of time t:

stationary process

[2] Pole et al. (1994):pp. 4.

[3] Pole et al. (1994):p. 9.

$$\boxed{\begin{array}{rlr} \text{mean} & \mu(t) = \mathbb{E}(Y_t) & (7.1) \\ \text{variance} & \sigma^2(t) = \operatorname{Var}(Y_t) = \mathbb{E}\big((Y_t - \mu(t))^2\big) & (7.2) \\ \text{autocovariance} & \underline{\gamma(t,s) = \operatorname{Cov}(Y_t, Y_s) = \mathbb{E}\big((Y_t - \mu(t)) \cdot (Y_s - \mu(s))\big)} \quad \text{for } s \in \mathbb{R} & (7.3) \end{array}}$$

For a stationary process especially $\mu(t) = \mu$ and $\sigma(t) = \sigma$ are thus constant.

Example 7.3 (*White noise*)**.** A *white noise* is defined as a process

$$Y_t = \varepsilon_t \tag{7.4}$$

white noise

with a sequence (ε_t) of uncorrelated and identically distributed random variables whose variance exists. Specifically, if the variance σ_ε^2 is the same for all ε_t we write $\varepsilon_t \sim \mathrm{WN}(0, \sigma_\varepsilon^2)$. Moreover, if all random variables are normally distributed, we write $\varepsilon_t \sim \mathrm{N}(0, \sigma_\varepsilon^2)$. The term "white noise" comes from the engineering sciences and describes interfering signals which do not contain useful information because of their constant spectral density, like white light. White noise is the simplest stochastic process. It plays a prominent role in time series analysis. A white noise with constant mean and constant variance, i.e., $\varepsilon_t \sim \mathrm{WN}(\mu_\varepsilon, \sigma_\varepsilon^2)$, is a stationary process. □

Example 7.4 (*Random Walk*)**.** A *random walk* is defined as a process

$$Y_t = Y_{t-1} + \varepsilon_t \tag{7.5}$$

random walk

with a random variable $\varepsilon_t \sim \mathrm{WN}(\mu_\varepsilon, \sigma_\varepsilon^2)$. This is a process with independent increments. If the expected value μ_ε of the increments is different from zero, the process has a drift. However, it is not so easy to tell from the time series plot whether the random walk has a drift. The variance of each random walk is $\operatorname{Var}(Y_t) = t\sigma_\varepsilon^2$ and is therefore linear with time[4]. Also a random walk without drift can look in sections like this as if it drifts away. But above all a random walk is *not* stationary. □

7.3 Causal linear processes

linear process

Definition 7.5. A stochastic process $(Y_t)_{t\in\mathbb{Z}}$ is called *linear process*, if for all $t \in \mathbb{Z}$ it can be represented by

$$Y_t = \sum_{j=-\infty}^{\infty} \psi_j \, \varepsilon_{t-j}. \tag{7.6}$$

Where $\varepsilon_t \sim \mathrm{WN}(0, \sigma_\varepsilon^2)$ is a centered white noise and (ψ_j) is a sequence of of real numbers with $\sum\limits_{j=-\infty}^{\infty} |\psi_j| < \infty$ is. Equation (7.6) is also called *Wold expansion.*[5].

The latter condition guarantees that the series in (7.6) converges with probability 1 and in the quadratic mean.[6]. A linear process is always stationary and has the following values for expected value, variance, and covariance function, respectively:

$$\mu = 0, \qquad \sigma^2(t) = \sigma_\varepsilon^2 \sum_{j=-\infty}^{\infty} \psi_j^2, \qquad \gamma(t) = \sigma_\varepsilon^2 \sum_{j=-\infty}^{\infty} \psi_j \psi_{j+t}. \tag{7.7}$$

(Of course, $\sigma^2(t) = \gamma(0)$.) The white noise perturbation terms $(\varepsilon_t) \sim \mathrm{WN}(0, \sigma_\varepsilon^2)$ are also called *shocks* or *innovations*, because they give the impulses driving the linear process.

shocks, innovations

In signal theory, the sequence (ψ_j) is called a "linear filter" (actually: the "impulse response" of a "linear filter"). A linear process thus corresponds to the convolution of a linear filter (ψ_j) with a white noise (ε_j).[7].

Remark 7.6. If we interpret the time t in (7.6) as the present, then the state Y_t of a linear process at the present time obviously also depends on $\varepsilon_{t+1}, \varepsilon_{t+2}, \ldots$ and thus depends on the future. If we take the process to be a real time series, this property is useless, of course. For deterministic processes which – as in classical mechanics – are determined by exact laws of motion and which we can measure exactly, this is true to some extent: For such processes, however, a Laplacian demon only needs to know *a single* state at any given time exactly, and it then knows *all* states of the process!

depends on the *future*?

But as soon as we are dealing with random processes or with measurement inaccuracies, we can know the states of the time series only in the past and the present, states of the future we can forecast, at most. Therefore, we restrict ourselves to processes independent of the future, which are given when the coefficients ψ_j vanish for all negative i. □

Definition 7.7. A stochastic process $(Y_t)_{t\in\mathbb{Z}}$ is called a *causal linear process*, if for all $t \in \mathbb{Z}$ it can be represented as

causal linear process

$$Y_t = \sum_{j=0}^{\infty} \psi_j \, \varepsilon_{t-j}, \tag{7.8}$$

where $\varepsilon_t \sim \mathrm{WN}(0, \sigma^2)$ is a centered white noise and (ψ_j) is a sequence of of real numbers with $\sum\limits_{j=0}^{\infty} |\psi_j| < \infty$.

The term "causal" ensures that the past and present, but not the future, are causal for the further course of the process. Causal linear processes have a special meaning for the analysis and prognosis of stationary processes. This is based on the decomposition theorem of Wold stating that almost every stationary process can be formed by a causal linear process and a deterministic process.[8] An important class of linear processes are formed by the ARMA (p, q)-models with parameters p, $q \in \mathbb{N}_0$. They are given by the equation

ARMA(p, q)

$$Y_t = \varphi_1 Y_{t-1} + \ldots + \varphi_p Y_{t-p} + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \ldots - \theta_q \varepsilon_{t-q}, \tag{7.9}$$

where the coefficients $\varphi_1, \ldots, \varphi_p$ and $\theta_1, \ldots, \theta_q$ are real numbers and (ε_t) is a centered white noise with variance σ^2 is. For such a process to be causal, the coefficients satisfy certain conditions. Let us first consider the autoregressive processes AR(p).

7.4 The random walk hypothesis in economics

In the economic sciences for certain economic processes the random walk hypothesis according to which the further development of a process depends solely on its current state (Y_{t-1}), but not on further historical data or time-lagged variables. For example, according

[4] Vogel (2015):p. 27.

[5] Palma (2016):p. 93.

[6] Vogel (2015):p. 77.

[7] cf. Palma (2016):p. 122.

[8] Palma (2016):§2.5.1; Vogel (2015):§5.3.

to this hypothesis, the growth rate of real private consumption is determined neither by past growth rates nor by past disposable income; stock prices cannot be predicted from the past either.[9] Specifically, this assumption means that the trajectories of the time series in question represents a random walk, possibly with a nonvanishing mean μ. As an example, consider the real price trajectory of the Dow Jones Index in 2019 compared to a random walk simulated with Python, see Fig. 7.2. Here, the index values divided by the starting

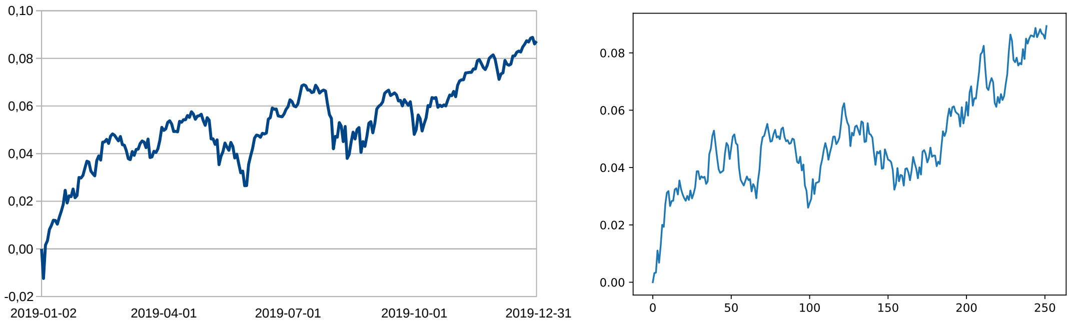


Figure 7.2. Time series of the Dow Jones index throughout 2019 (left hand side, logarithm of the quotients of the index values by the starting value). And a simulated random walk (right hand side). Data source: finance.yahoo.com

value on January 2, 2019 are plotted logarithmically against time. This compares to a simulated random walk with vanishing mean and innovation terms $\varepsilon_t \sim \text{WN}(0, \sigma_\varepsilon^2)$ with mean $\mu_\varepsilon = 0$ and standard deviation $\sigma_\varepsilon = 0{,}0035$ as shown in equation (7.5).

The random walk hypothesis is a widely accepted basic assumption in economics. In the limit for infinitesimally small time steps, the random walk for normally distributed innovation terms is a one-dimensional Wiener process, which in turn underlies the theory of financial derivatives such as options, futures, and swaps.[10] The random walk hypothesis is consistent with the hypothesis of market efficiency of financial markets, according to which current prices always include all available information. Consequently, with the assumption of this hypothesis, no market participant is in a position to achieve permanently above-average profits on financial markets by technical analysis, fundamental analysis, insider trading or otherwise[11]. In[12], three variants of the random walk hypothesis are discussed that differ in the distribution of innovation terms ε_t.[13]

market efficiency

hypothesis disputed

Is the random walk hypothesis realistic? Some economists doubt it. In particular, proponents of *technical analysis* do not see random terms with identical distributions as determining the price performance of securities, but rather certain geometric patterns of the curves.[14].

7.5 Problems

Problem 7.1 (**Stationary processes**). (a) Let be given the two stochastic processes

$$Y_1(t) = \sin t + \varepsilon_t, \qquad Y_2(t) = \sqrt{|t|} + \varepsilon_t. \tag{7.10}$$

[9] Neusser (2011):p. 3.

[10] Hull (2000):§10.

[11] Hull (2000):§10.

[12] Campbell et al. (1997):§2.1.

[13] `https://books.google.de/books?id=7Gkri6HWWkgC&pg=PA28`

[14] Campbell et al. (1997):§2.3.2; J. J. Murphy (2016).

with innovation terms $\varepsilon_t \sim \mathrm{WN}(0,1)$. Program these two functions in Python and plot the function graphs for 100 time points $t = 1, 2, 3, \ldots, 100$.

(b) Assume that the mean and the standard deviation of Y_1 are exactly $\mu_1(t) = 0$ and $\sigma_1 = 1/\sqrt{2}$, rsp., and correspondingly $\mu_2(t) = \frac{2}{3}\sqrt{t}$ and $\sigma_2(t) = \mu_2(t)/\sqrt{8}$. Are Y_1 and Y_2, taken as stochastic processes, stationary in each case? Can you recognize it from the graphs of the functions?

(c*) Derive a general formula that gives the mean $\mu(t)$ of the stochastic process $Y(t) = f(t) + \varepsilon_t$ where $f : \mathbb{R} \to \mathbb{R}$ and $\varepsilon_t is \sim \mathrm{WN}(0, \sigma_\varepsilon^2)$. What are then specifically the formulas for $\mu_1(t)$ and $\mu_2(t)$ from (b)? What follows approximately for the limiting case $t \longrightarrow \infty$ of large time periods?

Hints: Geometrically, the mean of a function f in an interval $[0, t]$ can be thought of as the quotient $\mu = A/t$, where A denotes the area of the function graph between 0 and t with the t axis. Alternatively, one can use the definition $\mu = \frac{1}{t} \sum_{\tau=0}^{t} f(\tau)\Delta\tau$ for the limit $\Delta\tau \longrightarrow 0$.

8
Autoregressive models

Overview

8.1 Stochastic processes in economics

Multiplier approach of Keynes

austerity

What do causal linear processes have to do with reality? The following example is a classic from economics, which on closer inspection describes the influence of investments on an economy as exactly such a stochastic process. It is the multiplier accelerator model developed by Paul Samualson and published shortly after the seminal work of John Maynard Keynes in the mid-1930s. Keynes proved with his multiplier approach that the government, by increasing its spendings, increases the national income far more than it actually spends. This was in sharp contradiction to the usual austerity policy of the states at that time, according to which the state should strictly avoid debt. According to many economists and historians this restrictive austerity policy exacerbated the Great Depression of 1929 and led in particular to the end of the Weimar Republic.[1]

Samuelson's multiplier accelerator model

Example 8.1 (*Samuelson's multiplier accelerator model)***.** Paul Samuelson's 1939 economic multiplier accelerator model[2] assumes a closed economy in which the demand for goods can always be satisfied. According to this model, aggregate demand Y_t at time t is composed additively of firms' demand I_t for capital goods and the demand C_t of households for consumption goods,[3]

$$Y_t = I_t + C_t. \tag{8.1}$$

[1] Büttner (2008):p. 423ff.

[2] Samuelson (1939).

[3] Krugman and Wells (2006):p. 283; Samuelson and Nordhaus (1995):p. 450; Bofinger (2007):p. 352; Felderer and Homburg (1989):p. 112.

In addition, the individual types of demand should be connected via the relationships

$$C_t = \alpha Y_{t-1} \tag{8.2}$$

$$I_t = \beta\,(C_t - C_{t-1}) + I_{\text{Unplanned}}, \tag{8.3}$$

$0 < \alpha,\ \beta < 1.$[4]. The reasoning behind this is that, on the one hand, general demand Y_t is time-delayed via the average propensity to consume α has a proportional effect on consumption C_t,[5] and, on the other hand, the necessary investments are associated with the *accelerator* β and the unplanned investment demand $I_{\text{Unplanned}}$ depend linearly on the change in consumption. Incidentally, the term multiplier here refers to the limit of the series $1 + \alpha + \alpha^2 + \ldots \to (1 - \alpha)^{-1}$. Equations (8.2) and (8.3) inserted into (8.1) give the linear difference equation of 2nd order

$$Y_t = \beta\,(\alpha Y_{t-1} - \alpha Y_{t-2}) + I_{\text{Unplanned}} + \alpha Y_{t-1} = \varphi_1 Y_{t-1} + \varphi_2 Y_{t-2} + I_{\text{Unplanned}}$$

with the coeffitients $\varphi_1 = \alpha(1 + \beta)$ und $\varphi_2 = -\alpha\beta$. Depending on the parameters α and β, the solution of the difference equation can take manifold forms,[6]. The model becomes stochastic if, for each time t, investment $I_{\text{Unplanned}}$ is replaced by innovations $\varepsilon_t \sim WN(0, V_t)$, which form a centered white noise. The demand for goods, conceived as a stochastic process (Y_t), then satisfies the model equation

$$Y_t = \varphi_1 Y_{t-1} + \varphi_2 Y_{t-2} + \varepsilon_t. \tag{8.4}$$

See also section A.2 in the appendix.[7] Blanchard (1981) determined the values $\varphi_1 = 1.34$, $\varphi_2 = -0.42$ for the gross domestic product (actually the GNP) of the USA minus a linear trend in the period March 1947 to April 1978. □

8.2 Definition and properties of autoregressive processes

A model like Samuelson's one in example 8.1 is called "autoregressive" because, except for random innovation, the process value at time t is a linear combination of values of the same process at previous points in time. Formally:

Definition 8.2. For $p \in \mathbb{N}_0$ is called a stochastic process (Y_t) *autoregressive of order p*, or AR(p) process for short, if it satisfies the model equation AR(p)

$$Y_t = \varphi_1 Y_{t-1} + \ldots + \varphi_p Y_{t-p} + \varepsilon_t \tag{8.5}$$

with real constants $\varphi_1, \ldots, \varphi_p$ and a centered white noise $\varepsilon_t \sim \text{WN}(0, \sigma_\varepsilon^2)$. The coefficients φ_i may vanish, but it must hold $\varphi_p \neq 0$.

More generally, we also can describe an autoregressive process of order p with non-vanishing expected value by

$$Y_t = \varphi_0 + \varphi_1 Y_{t-1} + \ldots + \varphi_p Y_{t-p} + \varepsilon_t \tag{8.6}$$

with another real coefficient φ_0. By moving to the process $Y_t' = Y_t - \mu$ we are immediately led back to the model equation (8.5). For reasons discussed in the theory of difference equations. the expected value for (8.6) however, is not φ_0 but[8]

$$\mu = \mathbb{E}(Y_t) = \frac{\varphi_0}{1 - \varphi_1 - \ldots - \varphi_p}. \tag{8.7}$$

[4] Krugman and Wells (2006):pp. 272, 278.

[5] Felderer and Homburg (1989):p. 106.

[6] cf. Rinne and Specht (2002):pp. 152–154.

[7] for further informations cf. Samuelson and Nordhaus (1995):pp. 448, 557; Vogel (2015):p. 79.

[8] cf. Goldberg (1958):p. 170; Vogel (2015):p. 80.

8.3 Causality of autoregressive processes

At first glance, however, it is not necessarily obvious that (8.5) is a linear process at all. Strictly speaking, this is not always the case, but most autoregressive processes are linear, after all. The following theorem clarifies this statement.

Satz 8.3. *An* AR(p) *process (8.5) is a causal linear process if and only if the coefficients* φ_i *are so that the zeros of the* characteristic polynomial

$$\Phi_p(z) := 1 - \varphi_1 z - \varphi_2 z^2 - \ldots - \varphi_p z^p \tag{8.8}$$

all lie outside of the unit circle in the complex plane $\mathbb{C}$*, i.e., have an absolute value* > 1 *haben. The expected value of the processes is then is zero.*

This proof is not relevant for the exam, but it explains the origin of Φ_p.

Proof. The role of the characteristic polynomial results from the Theory of linear difference equations with constant coeffitients[9] if we rewrite (8.5) with the help of the substitution $y_t = Y_{t-p}$, i.e., $Y_t = y_{t+p}$, to the form

$$y_{t+p} - \varphi_1 y_{t+p-1} - \ldots - \varphi_p y_t = \varepsilon_t. \tag{8.9}$$

Its solution behavior depends on the roots of the characteristic polynomial

$$f_p(x) = x^p - \varphi_1 x^{p-1} - \varphi_2 x^{p-2} - \ldots - \varphi_p \tag{8.10}$$

In particular, every solution y_t converges for $\varepsilon_t = 0$ (of the "homogeneous difference equation") converges stably to 0, if the possibly complex roots x lie within the unit circle.[10] Since $\Phi_p(1/x) = f_p(x)$, the roots of $\Phi_p(z)$ with $z = 1/x$ then lie exactly outside the unit circle. In this case there exists a stable constant solution for the difference equation (8.9),[11] i.e., the process (Y_t) will always reach a stable equilibrium, namely its vanishing expectation value, cf. (8.7).[12]. But is (8.5) a linear process at all? To see this, we need to rewrite (8.5) as an infinite series (7.6). However, to avoid tedious and confusing "index battles", we introduce the backshift operator B, which simply resets a stochastic process at time t by one time unit:

backshift operator against index battles

$$BY_t = Y_{t-1} \tag{8.11}$$

If it is applied j times, it causes a time shift back i time units, $B^j Y_t = BB\ldots BY_t = Y_{t-j}$. This way we can simplify (8.5), after rearranging it to ε_t, by the operator equation

$$\Phi_p(B)\, Y_t = \varepsilon_t \tag{8.12}$$

that is, completely without indices. Defining now the polynomial

$$\Psi(z) = \sum_{j=0}^{\infty} \psi_j z^j, \tag{8.13}$$

we can also rewrite (7.6) as an index-free operator equation

$$Y_t = \Psi(B)\, \varepsilon_t. \tag{8.14}$$

[9] Goldberg (1958):pp. 163–164.

[10] Goldberg (1958):p. 152.

[11] Goldberg (1958):pp. 164, 171.

[12] cf.also Goldberg (1958):p. 170; Vogel (2015):p. 80.

Substituting the term ε_t by the term on the left hand side of Equation (8.12), division by Y_t yields the equation

$$1 \;=\; \Psi(B)\,\Phi_p(B) \;=\; (\psi_0 + \psi_1 B + \psi_2 B^2 + \ldots)\,(1 - \varphi_1 B - \ldots - \varphi_p B^p). \qquad (8.15)$$

Since now on the left side there is only a constant, a comparison of the coefficients after multiplication shows that $\psi_0 = 1$ and the coefficients of all higher powers of B must vanish:

$$\begin{array}{ll}
\text{coefficient of } B^0: & \psi_0 = 1 \\
\text{coefficient of } B\ : & \psi_1 - \varphi_1 = 0 \\
\text{coefficient of } B^2: & \psi_2 - \psi_1\varphi_1 - \varphi_2 = 0 \\
\text{coefficient of } B^3: & \psi_3 - \psi_2\varphi_1 - \psi_1\varphi_2 - \varphi_3 = 0 \\
 & \vdots
\end{array}$$

This gives the recursive definitions of the coefficients

$$\psi_0 = 1,\;\; \psi_1 = \varphi_1,\;\; \psi_2 = \varphi_1\psi_1 + \varphi_2\psi_0,\;\ldots,\;\; \psi_j = \sum_{k=1}^{j} \varphi_k \psi_{j-k},\;\ldots \qquad (8.16)$$

with $\varphi_k = 0$ for $k > p$. Therefore we can rewrite the autoregressive process as

$$Y_t = \sum_{j=0}^{\infty} \psi_j\, \varepsilon_{t-j}. \qquad (8.17)$$

Since we already know from the premises of the theorem that the process (Y_t) reaches a stable equilibrium, the condition $\sum_j^\infty |\psi_j| < \infty$ is necessarily satisfied. □

If one of the unit roots of the characteristic polynomial (8.8) has absolute value 1, then the AR(p) process (8.5) is non-stationary.[13] In this case, there exist constant or periodic solutions[14] for the associated difference equation (8.9), i.e., the process (Y_t) will not reach a stable equilibrium.[15]

Corollary 8.4. *For $p \leqq 2$ an* AR(p) *process (8.5) is a causal linear process if and only if we have:*

$$\begin{array}{lll}
|\varphi_1| < 1 & \textit{for } p = 1, & (8.18) \\
|\varphi_1| < 1 - \varphi_2 \;\; \textit{and} \;\; |\varphi_2| < 1 & \textit{for } p = 2. & (8.19)
\end{array}$$

Proof. For $p = 1$: The characteristic polynomial (8.8) is $\Phi_1(z) = 1 - \varphi_1 z$ and thus has only a single zero, $z = 1/\varphi_1$. Since it only lies outside the unit circle for the case $|\varphi_1| < 1$, according to theorem 8.3 the assertion holds.

This proof, too, is not relevant for the exam, but it gets by with purely elementary mathematical transformations

For $p = 2$: The characteristic polynomial (8.8) is $\Phi_2(z) = 1 - \varphi_1 z - \varphi_2 z^2$ and has the two (possibly equal) roots

$$a_\pm = -\frac{\varphi_1}{2\varphi_2} \pm \sqrt{\frac{\varphi_1^2}{4\varphi_2^2} + \frac{1}{\varphi_2}} = \frac{-\varphi_1 \pm \sqrt{\varphi_1^2 + 4\varphi_2}}{2\varphi_2} \;\in\; \mathbb{C}. \qquad (8.20)$$

[13] see Vogel (2015):p. 83.

[14] Goldberg (1958):p. 164.

[15] Goldberg (1958):p. 171.

In order to determine their position with respect to the unit circle, we must calculate their absolute values $|a_\pm|$ and determine when $|a_\pm| > 1$, but equivalently we can also calculate in case $|1/a_\pm| < 1$. The reciprocals are given by

$$\frac{1}{a_\pm} = \frac{-\varphi_1 \pm \sqrt{\varphi_1^2 + 4\varphi_2}}{2}, \tag{8.21}$$

as can be checked verified with the help of the tird binomial formula:

$$a_\pm \cdot \frac{1}{a_\pm} = \frac{-\varphi_1 \pm \sqrt{\varphi_1^2 + 4\varphi_2}}{2\varphi_2} \cdot \frac{\varphi_1 \mp \sqrt{\varphi_1^2 + 4\varphi_2}}{2} = \frac{\varphi_1^2 + 4\varphi_2 - \varphi_1^2}{4\varphi_2} = 1.$$

Now we have to distinguish two cases, depending on whether the expression under the root sign is positive or negative.

The case $\varphi_1^2 + 4\varphi_2 \geqq 0$: With (8.21) then $|1/a_\pm| < 1$ is equivalent to

$$-1 < \frac{1}{a_-} = \frac{-\varphi_1 - \sqrt{\varphi_1^2 + 4\varphi_2}}{2} \leqq \frac{1}{a_+} = \frac{-\varphi_1 + \sqrt{\varphi_1^2 + 4\varphi_2}}{2} < 1$$

i.e.,

$$\varphi_1 - 2 < -\sqrt{\varphi_1^2 + 4\varphi_2} \leqq 0 \leqq \sqrt{\varphi_1^2 + 4\varphi_2} < \varphi_1 + 2.$$

Squaring the first and the last inequality we obtain

$$\varphi_1^2 - 4\varphi_1 + 4 > \varphi_1^2 + 4\varphi_2 \qquad \text{und} \qquad \varphi_1^2 + 4\varphi_2 < \varphi_1^2 + 4\varphi_1 + 4$$

thus $1 - \varphi_2 > \varphi_1$ und $-\varphi_1 < 1 - \varphi_2$, i.e., $|\varphi_1| < 1 - \varphi_2$.

The case $\varphi_1^2 + 4\varphi_2 < 0$: In this case the roots are non-real and pairwise complex-conjugate, i.e., $a_\pm = -\left(\varphi_1 \pm \mathrm{i}\sqrt{|\varphi_1^2 + 4\varphi_2|}\right)$. Especially we have

$$\left|\frac{1}{a_+}\right|^2 = \left|\frac{1}{a_-}\right|^2 = \frac{\varphi_1^2 - (\varphi_1^2 + 4\varphi_2)}{4} = -\varphi_2,$$

i.e., $0 \leqq |1/a_\pm| < 1 \Leftrightarrow 0 \geqq \varphi_2 > -1$. All in all, the roots $a_\pm$ are outside of the unit circle, if the inequalities (8.19) hold.[16] □

In the sequel we will consider some examples to illustrate the meaning of the corollary.

AR(1) **Example 8.5** (AR(1) processes). [17] With Equation (8.5) an AR(1) process is given by

$$Y_t = \varphi_1 Y_{t-1} + \varepsilon_t. \tag{8.22}$$

According to corollary 8.4 it represents a causal linear process in the case $|\varphi_1| < 1$. In fact we van directly recalculate,

$$\begin{aligned} Y_t &= \varphi_1 Y_{t-1} + \varepsilon_t \\ &= \varphi_1^2 Y_{t-2} + \varphi_1 Y_{t-1} + \varepsilon_t \\ &= \varphi_1^3 Y_{t-3} + \varphi_1^2 Y_{t-2} + \varphi_1 Y_{t-1} + \varepsilon_t = \dots \\ &= \sum_{j=0}^{\infty} \varphi_1^j \varepsilon_{t-j}. \end{aligned}$$

This series converges exactly for $|\varphi_1| < 1$. For $|\varphi_1| = 1$, (Y_t) does not represent a stationary process. As a special case, the random walk (7.5) with $\varphi_1 = 1$ is *no* causal process. Finally, for $|\varphi_1| > 1$ the process is stationary and linear, but not causal.[18]. □

random walk ... again!

Example 8.6 (AR(2) processes)**.** With Equation (8.5) an AR(2) process is given by

AR(2)

$$Y_t = \varphi_1 Y_{t-1} + \varphi_2 Y_{t-2} + \varepsilon_t. \tag{8.23}$$

Then it is a causal linear process if and only if condition (8.19) is satisfied. □

8.4 Problems

Problem 8.1 (Simulated AR(1)-processes). (a) Program in Python a function `AR_1(phi)` that implements an AR(1) process with the parameter φ and a random variable $\varepsilon_t \sim \mathrm{WN}(0,1)$ as a time series with 1000 data values. Display with it four function plots for $\varphi = -1$, $\varphi = 0.5$, $\varphi = 1$, and $\varphi = 2$.

(b) Given such AR(1) processes, we have for $\varphi \leqq 1$:

$$\mu = 0, \qquad \sigma^2 = \begin{cases} 1 & \text{if } |\varphi| < 1, \\ t, & \text{if } \varphi = 1. \end{cases} \tag{8.24}$$

Using this information, describe the graphs of the functions.

[16] cf. also Goldberg (1958):p. 171; Vogel (2015):p. 85.

[17] Vogel (2015):p. 81.

[18] Vogel (2015):p. 81.

9 Autoregressive models with moving average

Overview

causal ⇒ stationary

Autoregressive models seem to describe real time series well. After all, classics of economic theory such as Samuelson's multiplier accelerator model can be represented as such processes. However, autoregressive models also have drawbacks. One of them is that if they are causal, they are automatically stationary. This sounds unspectacular at first, but in reality it is a highly uninteresting property. For example, consider the simulations

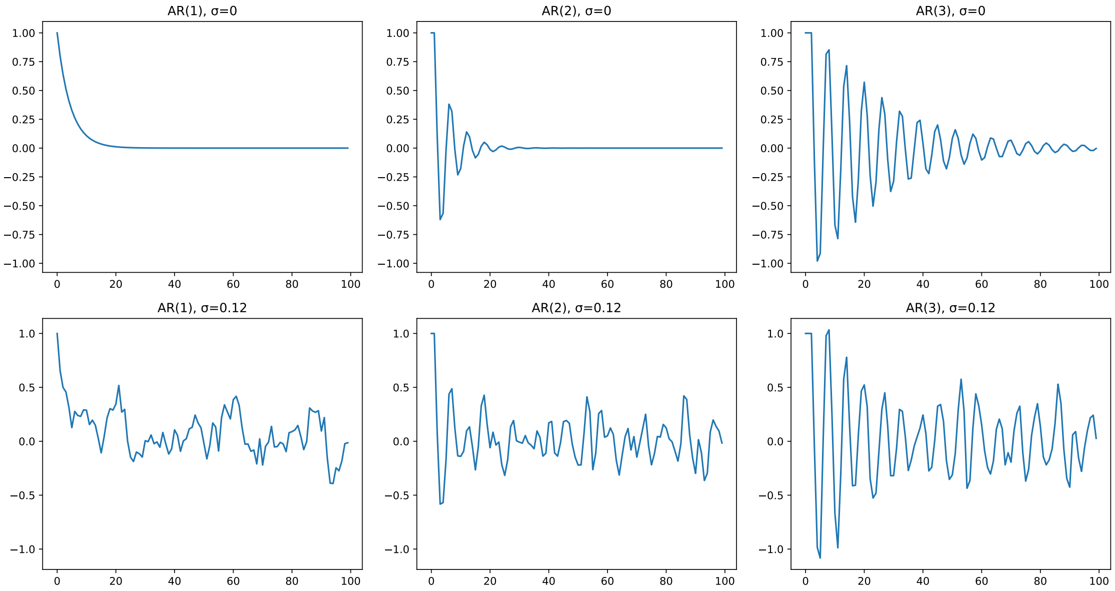


Figure 9.1. Simulations of different AR processes with the coefficients $\varphi_1 = 0.8$ for AR(1), $\varphi_1 = 0.8$, $\varphi_2 = -0.7$ for AR(2), and $\varphi_1 = 0.8$, $\varphi_2 = -0.7$, $\varphi_3 = -0.2$ for AR(3), and line by line each with the values $\sigma_\epsilon = 0$ und 0.12.

of autoregressive processes in Figure 9.1. In the first line of the figure, the trajectories of the three simulated processes AR(1), AR(2), and AR(3) are sketched without shocks (i.e., $\sigma_\varepsilon = 0$). They converge more or less briskly toward the stationary value zero. In contrast, if the same AR processes are now given a non-vanishing value (here $\sigma_\varepsilon = 0{,}5$), we see that it is only the shocks that prevent a causal autoregressive process from remaining in its stationary equilibrium.

stationarity is mostly boring

What does this tell us? Self-reference ("autoregression"!) without randomness only leads us to zero. Pretty boring, actually. But with shocks of sufficiently high standard deviation the deterministic part of the time series decreases over the long run. So why not go the other way and use models that do not rely on the previous values Y_{t-j} of the time series, but exclusively on the shocks ε_{t-j}? This is exactly what happens in the MA model, i.e., moving average models. We will discuss them in the next section.

Self-reference without randomness → 0

MA = Moving Average

9.1 MA models

The basic idea of MA models is that the innovations ε_{t-j}, $j = 1, 2, \ldots$, of the present and the past determine the current value Y_t of the time series, as a moving average depending on weighting factors θ_j. Thus, quite similar to the autoregressive models, where the past observed values Y_{t-j} determine the current value as a weighted sum. Formally, then, determining the parameters of an MA model using a given time series is a linear regression. However, since the values of the innovations are not observable, it is not a regression in the usual sense. Let us define MA processes more precisely for this purpose.

MA and regression

Definition 9.1. For $q \in \mathbb{N}_0$ is called. a stochastic linear process (Y_t) *process of moving averages of order* q, or in short MA(q) process, if it satisfies the model equation

MA(q)

$$Y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q} \tag{9.1}$$

with real constants $\theta_1, \ldots, \theta_q$ and a white noise $\varepsilon_t \sim \mathrm{WN}(0, \sigma_\varepsilon^2)$.[1] The coefficients θ_i may be zero for $i < q$, only the highest coefficient θ_q must not vanish, $\theta_q \neq 0$.

Remark 9.2. A causal linear process (7.8) is thus an MA(∞) process. In other words, an MA process is by construction always causal and thus stationary. On the other hand, this makes every autoregressive causal process an MA(∞) process. □

MA(∞)

Thus defined, a general MA process always has an expected value of zero. An MA process with an expected value $\mu = \mathbb{E}(Y_t) \neq 0$ we could describe by adding in (9.1) the constant μ:

$$Y_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q} \tag{9.2}$$

However, we can transform this model equation by considering the process $Y_t' = Y_t - \mu$ to (9.1). Because of the uncorrelated nature of the variables ε_t of the white noise, the autocorrelation function $\rho(k)$ of a MA(q) process satisfies

$$\rho(k) = \begin{cases} \dfrac{\theta_1 + \theta_1\theta_{k+1} + \theta_2\theta_{k+2} + \ldots + \theta_{q-k}\theta_q}{1 + \theta_1^2 + \theta_2^2 + \ldots + \theta_q^2} & \text{für } 1 \leqq k \leqq q, \\ 0 & \text{für } k > q. \end{cases} \tag{9.3}$$

[1] In the literature, the coefficients θ_j are often defined with the opposite sign, e.g., Vogel (2015). We stick here to the convention of Brockwell and Davis (2016), Deistler and Scherrer (2018), Kreiß and Neuhaus (2006), Neusser (2011), Palma (2016), and Shumway and Stoffer (2017), which is also used in the Python module `statsmodels`.

Although any MA process is causal and stationary by construction, it is common to impose conditions on the parameters θ_i. A key reason is that without such constraints we cannot unambiguously infer the coefficients from the autocorrelation function. For example an MA(1) process with the parameter $1/\theta_1$ has the same ACF and thus the same PACF as the MA(1) process with parameter θ_1, as shown by the following transformation of (9.3):

$$\rho(1) = \frac{\frac{1}{\theta_1}}{1 + \frac{1}{\theta_1^2}} = \frac{\frac{1}{\theta_1}}{\frac{\theta_1^2+1}{\theta_1^2}} = \frac{\theta_1}{1 + \theta_1^2}$$

A remedy for this is the criterion of invertibility of a MA process, which requires that it can be described as an AR(∞) process.

invertible

Definition 9.3. An MA(q) process $(Y_t)_{t\in\mathbb{Z}}$ is called *invertible* with respect to (ε_t), if it can be described as an AR(∞) process, i.e., in th form

$$Y_t = \sum_{i=1}^{\infty} \varphi_i Y_{t-i} + \varepsilon_t \tag{9.4}$$

such that (φ_i) is a sequence of real numbers satisfying $\sum\limits_{i=1}^{\infty} |\varphi_i| < \infty$.

What is the benefit of invertibility of an MA process? Since an invertible MA process with (9.4) satisfies the equation

$$\varepsilon_t = Y_t - \sum_{1}^{\infty} \varphi_i Y_{t-i} \tag{9.5}$$

invertible: AR process reconstructable from MA

its unobservable innovations ε_t can be reconstructed from the observable process values Y_t of the past an the present. Thus, for an invertible process the equivalent AR process can also be reconstructed. For non-invertible MA processes such a reconstruction is not possible.

Moreover, the restriction to invertible MA-processes is practically no restriction, because for an MA(q) process with a characteristic polynomial having no root on the unit circle, we can always find an MA(q) representation with a characteristic polynomial whose roots are neither on nor inside the unit circle, as we will see later with Theorem 9.8.

Satz 9.4. *An* MA(q) *process (9.1) is invertible with respect to* (ε_t)*, if and only if all roots of the characteristic polynomial*

$$\Theta_q(z) = 1 + \theta_1 z + \theta_2 z^2 + \ldots + \theta_q z^q \tag{9.6}$$

lie outside the unit circle.

Proof. [2]. □

MA(0) = white noise

Example 9.5. Since a MA(0) process of the model equation is $Y_t = \varepsilon_t$ it is a white noise (Example 7.3). Since the characteristic polynomial $\Theta_0(z) = 1$ has no zeros at all, in particular it has no zeros for $|z| \leqq 1$. Which AR process corresponds to it then? Of course the AR(0) process. □

MA(1)

Example 9.6. An MA(1) process is given by the equation $Y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1}$. To find the coefficients φ_i in its series (9.4), we set $\varepsilon_t = Y_t - \theta_1 \varepsilon_{t-1}$ successively with decreasing t recursively into itself and thus obtain

$$\varepsilon_t = Y_t - \theta_1 \varepsilon_{t-1} = Y_t - \theta_1 \varepsilon_{t-1} - \theta_1^2 \varepsilon_{t-2} = \ldots = Y_t - \sum_{i=1}^{\infty} \theta_1^i Y_{t-i}$$

Thus, the process is invertible exactly when $|\theta_1| < 1$. With the characteristic polynomial $\Theta_1(z) = 1 - \theta_1 z$ this is quite consistent with Theorem 9.4.[3]. □

9.2 ARMA

The starting point of this chapter was to find models for time series that are more expressive than autoregressive models. But unfortunately we did not succeed so far, despite the new approach with moving averages: By remark 9.2, every autoregressive causal process is indeed a special MA(∞) process, i.e., MA processes are a true extension of autoregressive processes. However, with definition 9.3 an invertible MA process is immediately an AR(∞) process. Invertible MA processes are of particular interest, however, because only for them the unobservable innovations ε_{t-j} can be uniquely reconstructed from the process variables Y_t of the present and the past. A next step to increase the expressive power of the models is to use the ARMA models which combine both models. A comparison of AR(2) processes with ARMA(2,2) processes of corresponding AR coefficients for different innovation terms σ is depicted in Figure 9.2. It shows that the basic structure of the autoregression is smoothed by the normally distributed innovation terms of the MA coefficients, or even dominated by them at high values for θ_j or σ_ε.

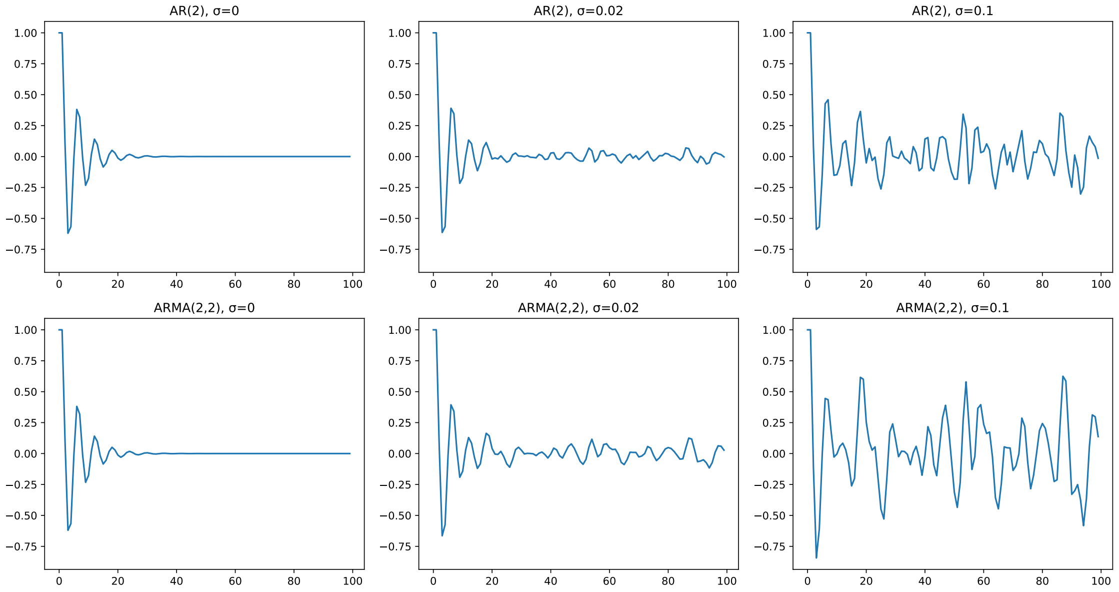


Figure 9.2. Comparison of simulated AR and ARMA processes with the coefficients $\varphi_1 = 0.8$, $\varphi_2 = -0.7$ for AR(2), and $\varphi_1 = 0.8$, $\varphi_2 = -0.7$, $\theta_1 = 0.8$ for ARMA(2,2), depicted line by line with the standard deviations $\sigma_\varepsilon = 0, 0.02$ und 0.1.

ARMA = AR + MA

[2] Brockwell and Davis (2016):§3.1.

[3] cf. also Shumway and Stoffer (2017):p. 83.

Overall, ARMA processes do not qualitatively increase the expressive power of AR processes, but they do expand the possibilities to model the influence of past innovations in a differentiated way.

Definition 9.7. A stochastic process $(Y_t)_{t\in\mathbb{Z}}$ is called ARMA(p,q)-Prozess, if it is stationary and satisfies the equation

$$Y_t - \varphi_1 Y_{t-1} - \ldots - \varphi_p Y_{t-p} = \varepsilon_t + \theta_1\varepsilon_{t-1} + \ldots + \theta_q \varepsilon_{t-q} \tag{9.7}$$

where we have $\varphi_p \neq 0$ and $\theta_q \neq 0$, and where the characteristic polynomials $\Phi_p(z)$ and $\Theta_q(z)$ defined according to (8.8) and (9.6), respectively, have no common zeros.

ARMA$(p,0)$ = AR(p)

ARMA$(0,q)$ = MA(q)

The model equation (9.7) is of course equivalent to equation (7.9), which we have learned in the Introduction. A ARMA$(p,0)$ process is a pure AR(p)-process, while a ARMA$(0,q)$ process is a pure MA(q) process. The conditions for stationarity (definition 7.2), causality (definition 8.2) and invertibility (definition 9.3) apply unchanged also to the ARMA model. Thus, in particular, we have:

- A stationary solution of Equation (9.7) exists exactly when the AR polynomial Φ_p has no zero on the unit circle; the stationary solution is unique.
- An ARMA process is causal with respect to ε_t exactly when all zeros of the AR polynomial Φ_p lie outside the unit circle.
- An ARMA process is invertible with respect to ε_t if and only if all zeros of the MA polynomial Θ_q lie outside the unit circle.

If the characteristic polynomials Φ_p and Θ_q had common zeros a_i, then there could be several stationary solutions of the equation (9.7), i.e., one solution of the equation would not be unique. However, the uniqueness can be easily established by dividing both polynomials by the common factors $(z - a_i)$. Thereby the orders p and q reduce by one for each linear factor.[4] The following remarkable theorem goes far beyond this property, for it states that not only can ARMA processes be suitably *transformed* into causal and invertible processes, but that in almost all cases they even *are* so.

Satz 9.8. *Let (Y_t) be an ARMA(p,q) process given by equation (9.7) whose characteristic polynomials have no zeros on the unit circle, i.e. that $\Phi_p(z) \neq 0$ and $\Theta_q(z) \neq 0$ hold for all $|z| = 1$. Then there are always two polynomials $\tilde{\Phi}_p(z)$ and $\tilde{\Theta}_q(z)$ with degree p and q, respectively, and a white noise $\tilde{\varepsilon}_t$, such that*

$$Y_t - \tilde{\varphi}_1 Y_{t-1} - \ldots - \tilde{\varphi}_p Y_{t-p} = \tilde{\varepsilon}_t + \theta_1\tilde{\varepsilon}_{t-1} + \ldots + \theta_q \tilde{\varepsilon}_{t-q} \tag{9.8}$$

and such that (Y_t) is causal and invertible ist.

Proof. [5] Let $a_r, \ldots, a_p$ and $b_s, \ldots, b_q$be the roots of Φ_p and Θ_q, respectively, with $|z| < 1$ inside the unit circle. The zeros thus obtained by replacing them by their reciprocals, i.e., by

$$\tilde{\Phi}_p(z) = \Phi_p(z) \cdot \prod_{j=r}^{p} \frac{(1 - a_j z)}{(1 - a_j^{-1} z)}, \quad \tilde{\Theta}_q(z) = \Theta_q(z) \cdot \prod_{j=s}^{q} \frac{(1 - b_j z)}{(1 - b_j^{-1} z)} \tag{9.9}$$

[4] Vogel (2015):p. 102.

[5] Brockwell and Davis (1991):Proposition 3.5.1.

defined polynomials have no zeros $|z| \leqq 1$. Using the backshift operator, already defined in the proof of Theorem 8.3 in equation (8.11), we can then calculate the innovation terms

$$\tilde{\varepsilon}_t = \frac{\tilde{\Phi}_p(B)}{\tilde{\Theta}_q(B)}\,\varepsilon_t \tag{9.10}$$

With Brockwell and Davis (1991:p. 105) they are white noise again. Therefore byEquation (9.8) the process (Y_t) is causal and invertible. □

Remark 9.9. Theorem 9.8 is amazing. It goes back to work done by U.S. statisticians Brockwell and Davis in the 1980s and states nothing less than that *all* ARMA processes are causal and linear as long as their characteristic polynomials do not have unit roots. What may have looked more like a purely technical reformulatio for the statement on invertibility – suitably transforming random terms ε_t to $\tilde{\varepsilon}_t$ does not change anything about causal relations – means an almost dramatic consequence for the philosophy of time at first sight: A non-causal process (with characteristic polynomials without unit roots), which by Remark 7.6 depends on random terms of the future, is by theorem 9.8 seamlessly representable as a process which exactly does not have this dependence into the innovations of the future. At second glance, however, the meaning for causality then does reduce to a more technical aspect, because non-causal processes do depend on future *random events* ε_t, but not on future *observations* Y_t. Strictly speaking, therefore, the term "causal" in definition 8.2 is not appropriate, because a non-causal process may well be causal in the usual literal sense of "causality"! The technical terms are what they are, though. However, since they are very common in the technical literature on the one hand and very useful on the other hand, since causal and invertible processes are mathematically easier to treat with, they are also used in this lecture notes. □

philosophy of time?

misunderstanding of "causal"?

9.3 Estimation of the order of ARMA models

So far we have worked out the theory of ARMA processes, but have not yet dealt with the practical problem of classifying a real observed time series as an ARMA process. The first task must therefore be to find suitable methods for this purpose. In practice, the autocorrelation function ACF and the partial autocorrelation function PACF have proven useful for this purpose, and we discuss them in this section.

estimating an ARMA model for a given time series

Definition 9.10. The *estimated autocorrelation function* (ACF) $\hat{\rho} : \mathbb{N}_0 \to \mathbb{R}$, for a time series (y_t), also called *empirical autocorrelation function*, is defined by

ACF = estimated autocorrelation function

$$\hat{\rho}(k) \;=\; \frac{\displaystyle\sum_{t=1}^{n-k} (y_t - \bar{y})\,(y_{t+k} - \bar{y})}{\displaystyle\sum_{t=1}^{n} (y_t - \bar{y})^2}, \tag{9.11}$$

with the arithmetic mean $\bar{y}$ of the measured values $\{y_t\}$.[6].

The autocorrelation function $\hat{\rho}(k)$ describes the correlation between an observation y_t and all other observation values $y_{t+1}, \ldots, y_{t+k}$ thereafter,

idea of ACF

[6] cf. Vogel (2015):p. 32.

$$\dots, y_{t-1}, y_t, y_{t+1}, \dots, y_{t+k-1}, y_{t+k}, y_{t+k+1}, \dots$$

With the help of the autocorrelation function we can now define the partial autocorrelation function as follows.

PACF = estimated partial autocorrelation function

Definition 9.11. The *estimated partial autocorrelation function* (PACF) $\hat{\pi} : \mathbb{N}_0 \to \mathbb{R}$, often called *empirical partial autocorrelation function*, is defined by

$$\hat{\pi}(k) = C_{k,k}$$

Help, a complicated recursive definition!? ... but easy to program!

with the coefficients $C_{k,t}$, which are given by the *Durbin-Levinson recursion*[7]

$$C_{k,k} = \frac{\hat{\rho}(k) - \sum_{t=1}^{k-1} C_{k-1,t}\, \hat{\rho}(k-t)}{1 - \sum_{t=1}^{k-1} C_{k-1,t}\, \hat{\rho}(t)}, \qquad C_{k,i} = C_{k-1,i} - C_{k,k}\, C_{k-1,k-i} \qquad (0 < i < k)$$

The initial values $C_{0,0} = 1$ and $C_{1,1} = \hat{\rho}(1)$ result from the first of the two equations.

idea of PACF

The PACF $\hat{\pi}(k)$ yields a measure of the correlation of y_t and y_{t+k}, which subtracts the influences of the values in between,

$$\dots, y_{t-1}, y_t, y_{t+1}, \dots, y_{t+k-1}, y_{t+k}, y_{t+k+1}, \dots$$

For a given time series, we will generally not know the orders p and q. The autocorrelation functions ACF and PACF are important tools to determine the nature of the underlying process. They can be used to estimate the orders p and q for a given time series. The estimates are based on the theoretical properties of ACF and PACF summarized in table 9.1. In a AR(p) process, the the PACF for lags $k > p$ strives exponentially towards zero, for a MA(q) process it is the ACF for lags $k > q$.[8]. For a ARMA(p, q) process with p, $q > 0$ finally, neither of the autocorrelation functions breaks down, but both decrease exponentially towards zero, possibly oscillatory[9]. If (Y_t) is a causal and invertible

ACF and PACF help to determine ARMA(p, q)

Table 9.1. Theoretical properties of the autocorrelation functions[a]

Prozess	ACF	PACF
AR(p)	decreases exponentially to zero for $k > p$ (monotone or oscillating)	$\pi(k) = 0$ für $k > p$
MA(q)	$\rho(k) = 0$ für $k > q$	decreases exponentially to zero for $k > q$ (monotone or oscillating)
ARMA(p, q)	decreases exponentially to zero for growing $k > \max\{p, q+1\}$ (monotone or oscillating)	decreases exponentially to zero for great $k > \max\{p, q+1\}$ (monotone or oscillating)

[a] Neusser (2011):pp. 64–65; Vogel (2015):p. 110.

[7] cf. Vogel (2015):p. 35; Kreiß and Neuhaus (2006):pp. 40–41, 69–71; Neusser (2011):p. 63.

[8] cf. Vogel (2015):pp. 88–89; Kreiß and Neuhaus (2006):p. 143; Neusser (2011):64f.

[9] Vogel (2015):p. 110.

ARMA(p, q) process. the following holds. The ACF satisfies for $k > \max\{p, q + 1\}$ the difference equation

$$\rho(k) = \varphi_1 \rho(k-1) + \ldots + \varphi_p \rho(k-p).$$

The zeros of the characteristic equation lie within the unit circle (cf. the discussion following equation because of the causality (cf. the discussion follwing Equation (8.10)). That is, the autocorrelation function $\rho(k)$ decreases exponentially toward zero when k tends to infinity. Whether $\rho(k)$ decreases monotonically or oscillatingly towards zero depends on the zeros of the characteristic equation. The PACF $\pi(k)$ starts to decrease toward zero at $k > p$.

To get a guess about the magnitude of p and q, a visual evaluation with the help of the correlograms of the empirical ACF and PACF should be done, as shown in in Figure 9.3. A correlogram is the functional graph of the autocorrelation functions $\hat{\rho}(k)$ and $\hat{\pi}(k)$, respectively, often plotted as a bar plot against lag k. Frequently, a confidence interval for

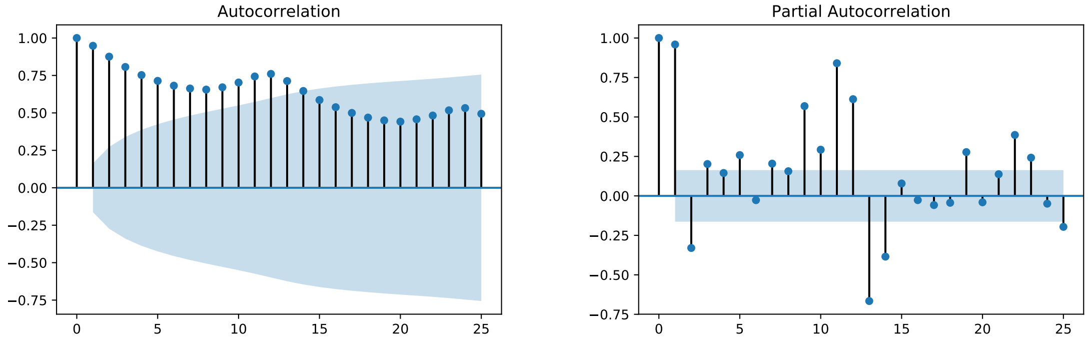


Figure 9.3. Correlograms of the estimated autocorrelation functions. ACF and the PACF for a real time series. The seasonal period 12 can be seen well here

the confidence level 95% is plotted. For the autocorrelation function $\hat{\rho}(k)$ this confidence level denotes the probability with which the respective hypothesis is to be rejected for a MA(k) process with normally distributed noise $\varepsilon_t \sim N(0, \sigma_\varepsilon^2)$. The interval limits are given by the Bartlett formula[10]

confidence interval for filtering the MA and AR coefficients

$$\rho_\pm(k) = \pm 1{,}96 \cdot \sqrt{\frac{1 + 2\left[\rho(1) + \rho(2) + \ldots + \rho(k)\right]}{n}}. \tag{9.12}$$

In the correlogram for the partial autocorrelation function, on the other hand, the interval boundaries are drawn accordingly[11]

$$\pm \frac{1{,}96}{\sqrt{n}}. \tag{9.13}$$

With correlograms we can thus at least identify pure AR and MA processes, namely when the ACF or PACF suddenly decays after a few lags k. For ARMA(p, q) models with $p \cdot q > 0$, however, the orders on the correlograms are not so easy to identify, because both ACF and PACF here decay exponentially. A listing of various simulated general ARMA processes is shown in Figure 9.4. Here the tendency for increased autocorrelations if the order of the autoregression is increasing is well seen, starting with the completely

[10] Vogel (2015):pp. 96–98.

[11] Neusser (2011):p. 66.

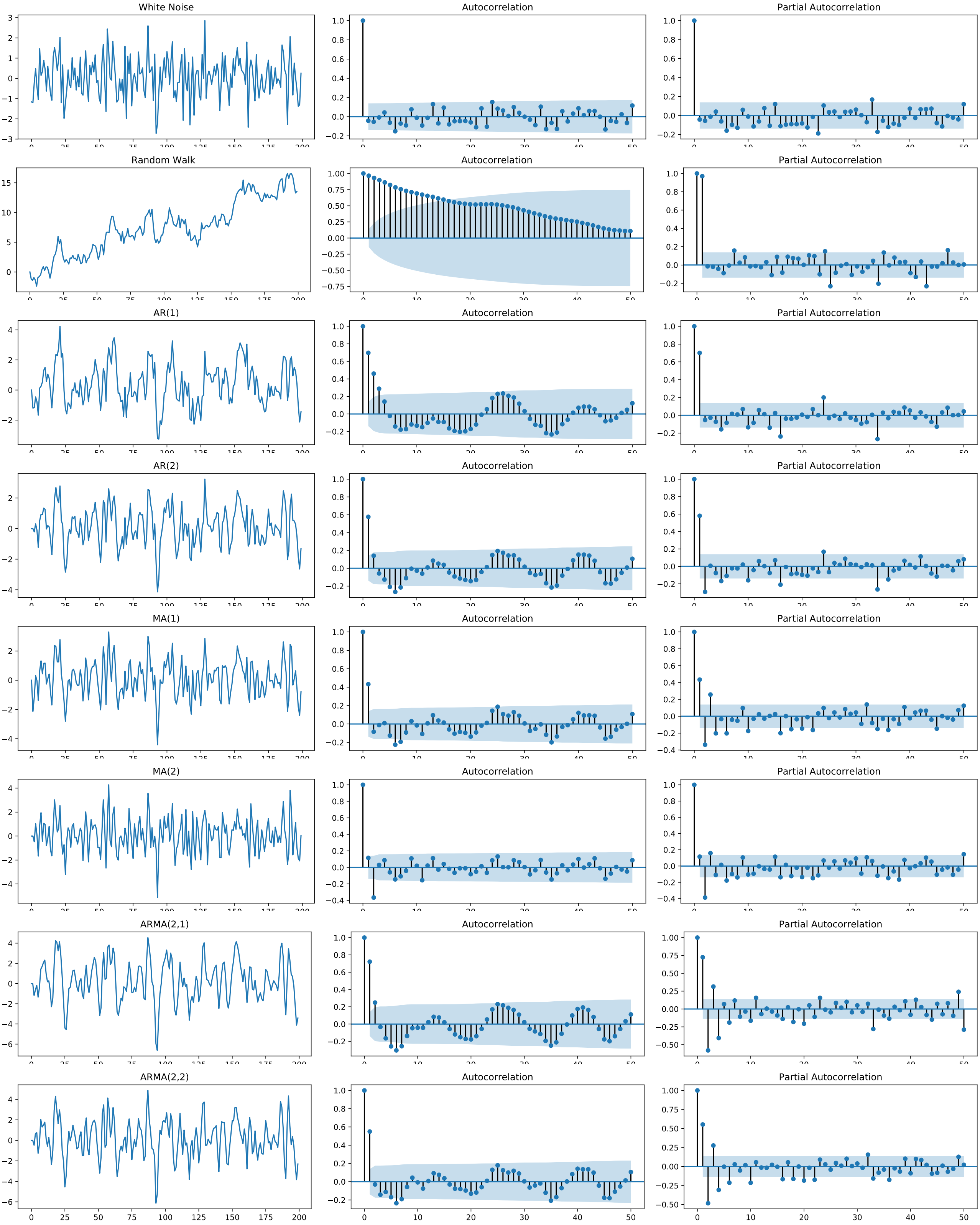


Figure 9.4. Simulation of different ARMA processes and their respective autocorrelation functions.

uncorrelated white noise and moving through the random walk to the higher order ARMA processes.

10 Trends and periods: SARIMA models

In practice, especially in economics, time series are not stationary. Therefore the ARMA models considered so far are basically not suitable for their analysis, unfortunately. Nevertheless, they are not completely unusable. For firstly, they and their properties are quite well understood, in particular their order can be well estimated by the methods described in section 9.3. Secondly, they can be understood as elementary stochastic processes of more general time series, which we can analyze with this understanding. A class of non-stationary processes suitable for this purpose are the "integrated" processes that we will discuss in this chapter.

non-stationary models

10.1 Time series with trends: Integrated processes

An ARMA process can be Differentiate to an ARIMA-process can be generalized. Thus, processes with a trend can now also be represented, i.e. non-stationary time series. In order to illustrate the idea behind it, we define first the differentiation indexdifferentiation of a process of the process (Y_t), actually the difference formation, by

$$\Delta Y_t := Y_t - Y_{t-1}. \tag{10.1}$$

Conversely, with $\Delta Y_t + Y_{t-1} = Y_t$ a differentiated process can be "integrated" back. As an example, consider the process $Y_t = a + bt + \varepsilon_t$ with a linear trend $b \neq 0$. Then we have

$$\Delta Y_t = Y_t - Y_{t-1} = a + bt + \varepsilon_t - (a + b(t-1) + \varepsilon_{t-1}) = b + \varepsilon_t - \varepsilon_{t-1}.$$

Then the differentiated process (ΔY_t) in general is not centered, but the trend has disappeared. If one differentiates a process (Y_t) twice,

$$\begin{aligned}\Delta^2 Y_t &= \Delta(\Delta Y_t) = \Delta(Y_t - Y_{t-1}) = \Delta Y_t - \Delta Y_{t-1} = Y_t - Y_{t-1} - (Y_{t-1} - Y_{t-2}) \\ &= Y_t - 2Y_{t-1} + Y_{t-2}, \end{aligned} \tag{10.2}$$

then a quadratic trend is filtered out of the time series. Accordingly, by further differentiation one can remove higher nonlinear trends from the process.

ARIMA(p,d,q) **Definition 10.1.** Let $d \in \mathbb{N}_0$ be a non-negative integer. A stochastic process (Y_t) is called an ARIMA(p,d,q) process if $Y'_t = (\Delta^d Y_t)$ is a causal ARMA(p,q) process. In other words, an ARIMA(p,d,q) process satisfies the model equation

$$Y'_t - \sum_{k=1}^{p} \varphi_k Y'_{t-k} = \varepsilon_t + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} \qquad \text{mit} \qquad Y'_t = \Delta^d Y_t, \tag{10.3}$$

where we have $\varepsilon_t \sim \mathrm{WN}(0, \sigma_\varepsilon^2)$, as well as φ_p and $\theta_q \neq 0$, and where the charcteristic polynomials Φ_p und Θ_q of the AR and MA part, defined respectively according to (8.8) and (9.6), donot share common zeros.

The "I" in the artificial word ARIMA appears because a d-fold integrated ARMA(p,q) process is exactly an ARIMA(p,d,q) process. An ARIMA proprocess (Y_t) also satisfies the model equation (9.7) of an ARMA process, with $\Phi_p(z)(1-z)^d$ as its characteristic polynomial and the corresponding modified coefficients φ_k.

Example 10.2. The simplest example of an ARIMA process not being an ARMA process is the random walk: It is created by differentiating once from the ARMA(0,0) process

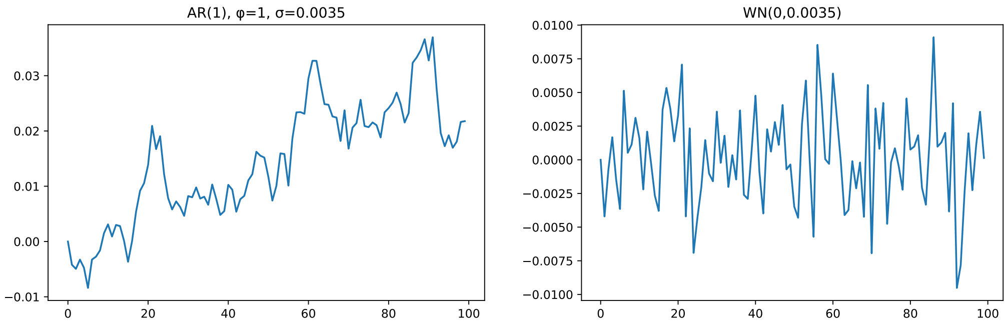


Figure 10.1. A random walk and its differentiated process

$Y_t = \varepsilon_t$, thus according to example 7.3 from the white noise, for

$$\Delta Y_t = Y_t - Y_{t-1} = \varepsilon_t \qquad \Longleftrightarrow \qquad Y_t = Y_{t-1} + \varepsilon_t. \tag{10.4}$$

cf.Figure 10.1. The random walk is therefore an ARIMA(0,1,0) process, its integrated process is the white noise. □

10.2 Approach to trends

To a given time series, which may not be considered stationary because of an obviously contained trend, a ARIMA(p,d,q) model is to be fitted. Before we now have to worry about the orders p and q, we first have to solve the problem of choosing the integration order d. For this purpose, it is advisable to differentiate the time series as long until no trend can be recognized. If after a single difference formation a stationary process already is obtained — which is often the case with economic time series — the original process is called *integrated*, or also *differential stationary*.

Caution is advised, however, when differentiating because both too large and too small d makes further analysis with an ARMA model impossible. If one chooses the order d too small, the d-fold differentiation leads to a process which is is not (yet) stationary. An non-stationary stochastic process, which in fact satisfies the ARMA model equation (9.7), but whose AR polynomial has has a unit root, is sometimes difficult or even impossible to recognize in the time series plot as being non-stationary. To objectify the choice of d, so-called unit root tests have been developed, with which one can investigate the question whether the AR polynomial Φ_p has unit roots. Such tests are used, e.g., in computer programs for the automatic determination of the order d in the search for a suitable ARIMA model. The two most known representatives are the ADF test and the KPSS-Test.

tests for unit roots in AR processes

If one differentiates an ARMA process that is stationary, then again a stationary process is obtained. However, this "overdifferentiated" process then has a unit root in the MA polynomial and is therefore not invertible. Testing for root 1 in the characteristic polynomial of an MA model is much more difficult than for the AR model. However, for the simple MA(1) model

overdifferentiation

tests for unit roots in MA processes

$$Y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} \qquad \text{mit} \qquad \varepsilon_t \sim \text{IID}(0, \sigma_\varepsilon^2)$$

there exists a test of Davis, Chen und Dunsmuir: Under the null hypothesis $H_0 : \theta_1 = 1$ the random variables $n(\theta_1 - 1)$ of the maximum likelihood estimator θ_1 for θ_1 converge, and for the marginal distribution α-quantiles are given. The null hypothesis that there is a unit root is rejected in favor of the alternative $H_1 : \hat{\theta}_1 < 1$ to the significance level α if $\hat{\theta}_1 < 1 - c_\alpha/n$ holds. The most common quantiles are $c_{0,01} = 11{,}93$, $c_{0,05} = 6{,}80$ und $c_{0,1} = 4{,}90$.[1]

10.3 SARIMA

Besides trends, there is another important deterministic part in real processes, namely periodic recurring effects. Such effects are also called seasonal in the economic field. Sales figures, for instance, are shaped by regular events, such as seasons or certain holidays like Christmas or Easter. To account for seasonal effects in time series analysis, an ARIMA(p, d, q) model is augmented by the four parameters $(P, D, Q)_s$, which respectively account for seasonal AR effects for period s of a season with P, seasonal MA effects with Q, and seasonal integration effects with D. In the literature, the entire model is denoted SARIMA (for "seasonal ARIMA model") with seven parameters $(p, d, q) \times (P, D, Q)_s$.

seasonal effects

$(p, d, q) \times (P, D, Q)_s$

Definition 10.3. A stochastic process (Y_t) is called SARIMA$(p, d, q) \times (P, D, Q)_s$ with the parameters p, d, q, P, D, Q, $s \in \mathbb{N}_0$, if it satisfies the following model equation:

$$y'_t = \underbrace{\sum_{i=1}^{p} \varphi_i y'_{t-i}}_{\text{AR}} + \varepsilon_t + \underbrace{\sum_{i=1}^{q} \theta_i \varepsilon_{t-i}}_{\text{MA}} + \underbrace{\sum_{i=1}^{P} \tilde{\varphi}_i y'_{t-s-i}}_{\text{seasonal AR}} + \underbrace{\sum_{i=1}^{Q} \tilde{\theta}_i \varepsilon_{t-s-i}}_{\text{seasonal MA}} \tag{10.5}$$

with the differentiated process

$$\underbrace{y'_t = \Delta^d \Delta_s^D y_t,}_{\text{(seasonal) I}} \tag{10.6}$$

and the relations

[1] cf. Vogel (2015):pp. 124–125.

y_t – observational data at time t,
s – period of a season (e.g., 12 for monthly data, 4 for quaterly data)
p, P – order of the autoregression = *lag*
= number of the autoregressive terms,
d, D – order of the differnce formation Δ^d ($\Delta y_t = y_t - y_{t-1}$, $\Delta^2 y_t = \Delta\Delta y_t$, ...)
= number of differences to acieve stationarity
q, Q – order of the moving average of the noise terms

(Capital letters here denote the seasonal portion in each case). Here the characteristic polynomials

$$\Phi_p(z) = 1 - \sum_{i=1}^{p} \varphi_i z^i, \quad \Theta_p(z) = 1 + \sum_{i=1}^{q} \theta_i z^i, \quad \tilde{\Phi}_p(z) = 1 - \sum_{i=1}^{P} \tilde{\varphi}_i z^{i+s}, \quad \tilde{\Theta}_p(z) = 1 + \sum_{i=1}^{Q} \tilde{\theta}_i z^{i+s} \tag{10.7}$$

do not share common zeros, respectively.

In practice, the polynomial degrees p, q, P,and Q will not be greater than 2. For the prediction of economic time series, most commonly SARIMA$(p, d, q) \times (0, 1, 1)_s$ models with $p + d + q \leqq 4$ are used. The SARIMA$(0, 1, s + 1) \times (0, 1, 0)_s$-model is equivalent to the additive Holt-Winters method of seasonal exponential smoothing, which is used, for example, in logistics for inventory optimization[2].

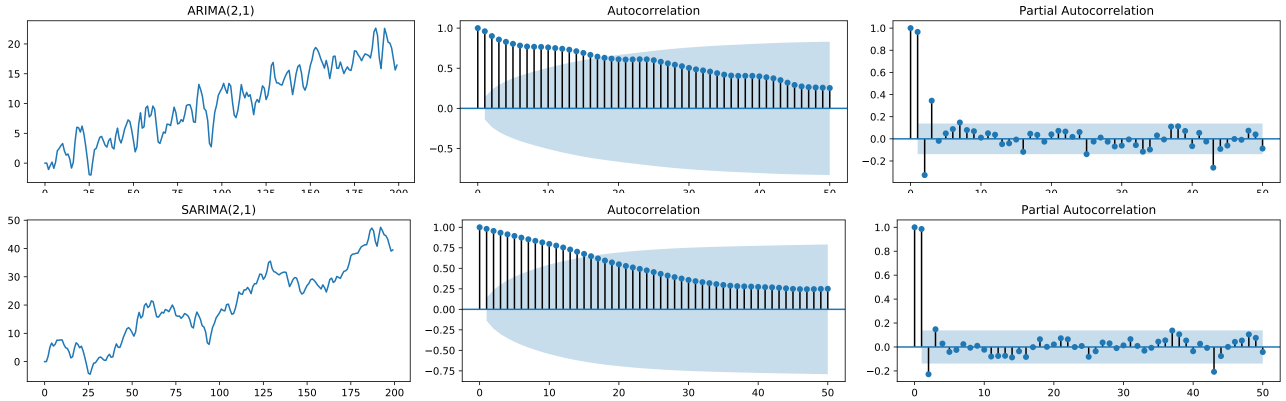


Figure 10.2. Simulation of different SARIMA processes and their autocorrelation functions. Both processes are non-stationary.

10.4 SARIMA models in `statsmodels`

The central class for general SARIMA-Modelle in the library `statsmodels` is `SARIMAX`

`https://www.statsmodels.org/stable/generated/statsmodels.tsa.statespace.sarimax.SARIMAX.html`

in the module `sarimax`. It can be imported by the instruction

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

Pandas `Series`

Necessary parameter for the creation of a SARIMAX instance is `endog` for the time series to be modeled. The time series can be a numeric array, but a Pandas Series, too, is possible and usually recommended. The advantage of a Pandas Series is that the points in time of the time series values can also be stored in date or time formats and the type of measurement period (`freq`) can be specified. For example, the statements

```
y = pd.Series([1000, 2500, 500, 3000])
y.index = pd.DatetimeIndex(
    ["2020-01-01", "2020-04-01", "2020-07-01", "2020-10-01"],
    freq='QS-JAN')
```

generate the time series

```
2020-01-01    1000
2020-04-01    2500
2020-07-01    500
2020-10-01    3000
Freq: QS-JAN, dtype: int64
```

the measurement dates of which occur quarterly from January 1 on (`"QS-JAN"`). The possible strings for the date offset are listed under the URL

`https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#dateoffset-objects`

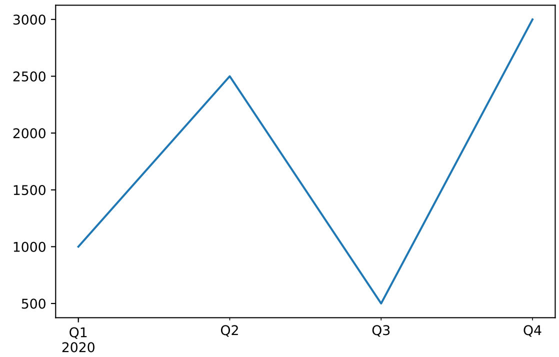


Figure 10.3. Plot of a trajectory of a time series stored as a Pandas Series

Common values are `"D"` for daily time data, `"M"` and `"MS"` for monthly data at the first or last of the month, respectively, `"Q"` and `"QS"` for quarterly data at the beginning or end of the month, and `"AS"` and `"A"` for annual data at the beginning or end of the year, respectively. With the plot command `y.plot()`, this will plot the time series trajectory in Figure 10.3.

10.5 Parameters to generate a SARIMAX model

`order`

`seasonal_order`

The order of the model is determined by the parameter `order`, expecting a tuple (p, d, q) of three numbers. Furthermore, the seasonal component $(P, D, Q)_s$ is determined by the parameter `seasonal_order` with a tuple (P, D, Q, s) of four numbers. The given default values of the two parameters are `order = (1, 0, 0)` and `seasonal_order = (0,0,0,0)`, i.e., the model represents an AR(1) process.

trend

Another optional parameter that is very useful in practice is `trend`, which determines the deterministic trend polynomial $A(t)$ on which the time series is based. It can either be one of the four strings `'n'`, `'c'`, `'t'`, `'ct'` or a list $[e_0, e_1, \dots, e_k]$ of zeros and ones, $e_i \in \{0, 1\}$. The strings cause no trend at all, a constant trend, a trend linear in t, or a linear and constant trend to be applied. A list $[e_0, e_1, \dots, e_k]$ of zeros and ones determines the non-vanishing terms of the applied polynomial, i.e. $A(t) = \sum_0^k e_i a_i t^i$. For example, the list $[1, 1, 0, 1]$ leads to the applied trend polynomial $A(t) = a_0 + a_1 t + a_3 t^3$.

10.6 Fitting a SARIMAX model

fit()

The most important method of SARIMAX is `fit()`. As usual in machine learning, it fits the parameters of the model to the observed data . In contrast to the principle implemented in Scikit-Learn that a new model is created from a model after fitting, which can be used to modify the dataset and thus fit another model, the `fit()` method in `statsmodels` creates as a result an instance of a new class, the class `MLEResults`[3]. In the documentation of `statsmodel`[4] it is recommended accordingly to assign the result to an own variable, as for instance

```
1 from statsmodels.tsa.statespace.sarimax import SARIMAX
2 model = SARIMAX(y, order=(2,0,0), trend='ct')
3 result = model.fit()
```

However, in order to take up and continue one of the basic ideas of machine learning, in these lecture notes the result of fitting the model parameters is stored in the variable of the model itself, like in the example in line 3 the variable `model`. Of course, this slight abuse of notation is justified only in the author's subjective trade-off against the fact that the unfitted model normally has no further use. If you rate this consideration differently, you can — and should — of course follow the recommendation of the documentation of `statsmodels` for your programs.

10.7 Predictions of a SARIMAX model

in-sample prediction

out-of-sample forecasting

In `statsmodels` a basic distinction is made between the two terms *predict* and *forecast*. While *predict* mainly denotes the performance of a so-called *in-sample prediction*, i.e., a reproduction of predictive values of the fitted model provided in a temporal window from the sample area, *forecasting* is understood as the true prediction of the model for a temporal range from the future of the sample, called *out-of-sample forecasting*. Accordingly, there are two methods for a fitted time series model, `predict` and `forecast`.[3]

predict()

However, the more flexible method of a fitted SARIMAX model for calculating a forecast here is `predict`, because it actually allows for both *in-sample prediction* and *out-of-sample forecasting*. The two important parameters are of `predict` are `start` and `end`, which determine the indexes of start and end time of the prediction related to the sample y, so for n_0 and n_k for example.

`pred = model.predict(start=`n_0`, end=`n_k`)`

[2] Katzenberger (2013).

[3] https://www.statsmodels.org/dev/generated/statsmodels.tsa.statespace.mlemodel.MLEResults.html

[4] https://www.statsmodels.org/stable/examples/notebooks/generated/statespace_sarimax_stata.html

Both parameters are optional, their default values are `start=0` and `end=`m, where m denotes the last index of the sample y. In other words, in this case *in-sample prediction* is performed over the entire sample.

`forecast()`

To compute a pure *out-of-sample forecasting*, the method `forecast` can also be called, which requires as parameter `steps` only the number of forecast values into the future after the last value of the sample. The parameter is optional, but default value is 0, i.e., no prediction is made at all. In practice, one will use this method only if an *out-of-sample* prediction is wanted, for most cases `predict` will be applicable.

The values for the parameters `start`, `end` and `step` do not have to be integer values, but can also be of type `datetime`.

10.8 Choosing a SARIMA model

Given a time series, its analysis is generally a complex procedure, as in machine learning in general, and is individually dependent on the data at hand. Here we will not clearly define a proceeding, but rather a collection of heuristic rules of thumb. Nevertheless, the following steps are generally accepted as a rough guide to action of a time series analysis[5].

1. *Data import:* Read the data as a Pandas DataFrame, analyze its structure, and preprocess the data if necessary.

2. *First Visualization:* Plot the trajectory of the time series.

3. *Rough data analysis:* Preliminarily categorize the time series, possibly transform the data.

4. *Determination of the hyperparameters:* Estimate the order parameters $(p, d, q, (P, D, Q)_s)$.

5. *Diagnosis:* Evaluate the model with statistical key figures, go back to step 3 or 4, if necessary.

6. *Determination of the model:* Select the model that is evaluated best.

The first step is to import the data into a program. Indeed, `Python Pandas` is aware of the mostly available data formats, e.g., text-based CSV files (extension `.csv` or often also `xls`), SPSS (`.sav`). Often, however, the data are in a structure that must first be modified as part of preprocessing to yield a processable time series, or processable time series, respectively.

As with almost any data analysis, we should first create the trajectory of the time series as a function graph to get a first impression. Are the values increasing or periodic? Is there a trend? Are the fluctuations around the trend always about the same? Especially to stabilize the variances, a *Box-Cox-Transformation* with the parameter $\alpha \geqq 0$ can be used:

$$y_t \quad \mapsto \quad y_t^{(\alpha)} = \begin{cases} \log y_t & \text{für } \alpha = 0, \\ \frac{1}{\alpha}\,(y_t^\alpha - 1) & \text{für } \alpha \neq 0. \end{cases} \tag{10.8}$$

The most common one is the logarithmic transformation ($\alpha = 0$), where usually the base of the logarithm does not matter for the analysis. It is often applied to growth processes.

[5] Shumway and Stoffer (2017):p. 135ff.

10.9 Problems

Problem 10.1. (a) Under the link `https://fred.stlouisfed.org/series/GDPC1` one can obtain the historical U.S. real gross domestic product data from March 1947 to April 1978 mentioned in Example 8.1. The data are available as quarterly data as of April 1, July 1, October 1, and January 1, respectively. Write a Python program in which the data are read and stored as Pandas Series and plot the time series.

(b) Apply an AR(2) model using SARIMAX with linear trend to the time series as training data for the first 70 % of the time series values.

(c) Compute the forecast for the last 30 % of the time series values as test data and plot it using a panda Data frame together with the trajectory of the entire time series (i.e., training and test data).

A Appendix

A.1 Solutions to selected problems

Problem 1.1 (a) A random variable X on a sample space Ω is a mapping $X : \Omega \to \mathbb{R}$. (Actually the topic of this part of the problem is less a question about random variables than about the mathematical notation of a mapping between two sets!)

(b) The table of values of the random variable X reads:

dots ω	1	2	3	4	5	6
$X(\omega)$	0	1	0	1	0	1

Since the probability of each outcome equals $\frac{1}{6}$, we have $P(X = 1) = P(2) + P(4) + P(6) = \frac{1}{6} + \frac{1}{6} + \frac{1}{6} = \frac{1}{2}$. The implicit assumption made by this deduction is that the six outcomes are uniformly probable, i.e., that each occurs with the probability $\frac{1}{6}$. (This part of the problem thus more deals with the formal derivation than with the – intuitively "somehow logical" – result: Every random variable assigns a real number to each outcome $\omega \in \Omega$; but we can determine the probability distribution only if we know the probabilities of the outcomes.)

Problem 2.1 We first have to associate the three statements to their respective roles of inference, i.e., rule, premise, and conclusion:

Rule $A \Rightarrow B$:	"When it rains, the street is wet"
Premise A:	"It rains"
Conclusion B:	"The street is wet"

Therefore,

Deduction	**Induction**	**Abduction**
When it rains, the street is wet	It rains	When it rains, the street is wet
It rains	The street is wet	The street is wet
The street is wet	When it rains, the street is wet	H_1: It rains
		H_2: It does not rain.
		H_3: The street is sprayed.
		H_1: It rains

(Here the third hypothesis H_3 of course is only exemplary.)

Problem 2.2 We have $P(X \mid A) = \frac{1}{3} \cdot (\frac{2}{3})^4$ and $P(X \mid B) = \frac{2}{3} \cdot (\frac{1}{3})^4$. Let formally $M_1 = A$ denote the model that the coin A is chosen, and $M_2 = B$ that coin B is chosen, respectively. Assuming that the prior probabilities of each model is equal,

$$P(A) = P(B) = \frac{1}{2}$$

we obtain by Equation (2.8) the probability ratio

$$\frac{P(A \mid X)}{P(B \mid X)} = \frac{P(X \mid A)}{P(X \mid B)} \cdot \frac{P(B)}{P(A)} = \frac{P(X \mid A)}{P(X \mid B)} = \frac{\frac{1}{3} \cdot (\frac{2}{3})^4}{\frac{2}{3} \cdot (\frac{1}{3})^4} = \frac{1 \cdot 2^4}{2 \cdot 1^4} = 2^3 = 8, \qquad (A.1)$$

in favor of type A.

Problem 4.1

Literal	Valid	Result / Explanation
`1.000e-0.2`	no	exponent after `e` must be an integer
`2e+1j`	yes	`20j`, a purely imaginary number
`0x567`	yes	`1383`, hexadecimal representation
`00x567`	no	Literals for hexadecimal numbers must have precisely one leading zero
`0o567`	yes	`375`, octal representation
`0o568`	no	8 is not an octal symbol
`'Größe'`	yes	`'Größe'`, string with Unicode symbols
`''Größe''`	no	Two apostrophs `''` define an empty string
`"Größe"`	yes	`'Größe'`, string with Unicode symbols
`b'Größe'`	no	Bytes can only contain ASCII symbols
`"Komm 'rein!"`	yes	A string in quotation marks may contain apostrophs – and vice versa
`00023e001`	yes	`230.0`, floating-point number; mantissa and exponent must be integers, but may have leading zeros and are interpreted as decimal numbers
`(1; 2; 3)`	no	Round brackets define tuples, the entries of which must be separated by commas

Problem 4.2

Object of Reality	Data Type	Exemplary Expression
radius of atoms	`float`	`3.2e-13`
name of flower	string (`str`)	`'Tulpe'`
name of participants of a race	`list` or `set` of Strings	`['Leonie', 'Stephanie', 'Alina']` or `{'Leonie', 'Stephanie', 'Alina'}`
score of a soccer match (e.g., 3:1)	tuple of two integers	`(3,1)`
name, prename and age of a person	tuple with three entries	`('Schmitz', 'Otto', 24)`

name, prename and age of a participant of a race	a list of tuples	`[('Meier', 'Leonie', 23),`
		`('Müller', 'Stephanie', 21),`
		`('Schmitz', 'Alina', 24)]`

table, in which the chemical element symbols are stored with their English and German names (e.g., H → hydrogen, Wasserstoff)	dictionary with symbols as keys and a list of English and German notions as values	`{'H': ['hydrogen','Wasserstoff'],`
		`'O': ['oxygen','Sauerstoff'],`
		`'C': ['carbon', 'Kohlenstoff']}`

Problem 4.3 Assuming the alphabet `'abc'` we obtain the output of all words with two letters as follows:

```
alphabet = 'abc'
for a in alphabet:
    for b in alphabet:
        print(a+b, end=' ')
```

With the parameter `end` the `print` method does not terminate with a line break (`'\n'`) but with a blank space (`' '`).

Problem 4.4 (a) For n elements the total number of k-digit combinations is n^k. Since here the number of base pairs is $n = 4$ and the number of digits is $k = 4$, the total number of combinations is $n^k = 4^4 = 256$.

(b) To print all 256 four-digit combinations of the base pairs AT, TA, GC and CG, it is appropriate in Python to firts define a list of four pairs as strings and then to run through a nested loop of depth four:

```
# Print out all DNA sequences with the four base pairs
base_pairs = ['AT', 'TA', 'GC', 'CG']
for a in base_pairs:
    for b in base_pairs:
        for c in base_pairs:
            for d in base_pairs:
                print(a, b, c, d)
```

Problem 4.5 The descreption of the random function `randint`, as given under

`https://numpy.org/devdocs/reference/random/generated/numpy.random.randint.html`,

reads `numpy.random.randint(low, high=None, size=None, dtype=int):` it expects the value `low` mandatorily; if `high` is not set, a random number is returned from the intervall `[0, low)`, otherwise from the intervall `[low, high)`. Then a solution may look like:

```
from numpy.random import randint
import time
print('Multiplication trainer')
print('----------------------')
start = time.time()
for i in range(5):
    m = randint(1,10)
    n = randint(1,10)
    result = 0    # can never be a product of positive numbers
    while result != m*n:
        result = int(input(str(m) + '*' + str(n) + '='))
        if result == m*n:
            print('Correct!')
        else:
            print('Unfortunately wrong! Try again ...')
end = int(time.time() - start)
print('For the tasks you needed', end, 'seconds.')
```

Problem 4.6 (a) The function `randn`$(d_1, d_2, \ldots, d_n)$ of the module `numpy.random`, according to the API `https://numpy.org/devdocs/reference/random/generated/numpy.random.randn.html`, generates one or several standard normally distributes samples as an array of dimension n (or expressed geometrically ausgedrückt: a "tensor of order n"). If no parameter is inputted, a random number $z \in \mathbb{R}$ (an array of dimension 0) is returned, for an input of one parameter an array ("tensor of order 1") of length d_1, for two parameters a matrix as an array consisting of d_1 Arrays of length d_2 (i.e., $d_1 \times d_2$), etc. For instance,

```
np.random.randn(3)
```

generates an array with three random numbers, say

```
[0.94759771,0.1613764, -0.25537882]
```

whereas

```
np.random.randn(2,3)
```

generates an array of random arrays with 3 entries, say:

```
[[ 0.24416202,  0.67809254, -0.15421969],
 [-1.19499275,  0.15050603, -1.84171316]]
```

(b) If we want to achieve a random sequence the mebers of which are distributed accordinhg to a normal distribution $N(\mu, \sigma)$ with mean μ and standard deviation σ, we apply:

```
sigma * np.random.randn(...) + mu
```

(c) With (a) and (b) a program to generate an error bar diagram with a $N(0, \sigma)$-normally distributed random array with standard deviation $\sigma = 0{,}001 \cdot \max\limits_{f\in[0,\infty)} \{B(f,\ 2{,}75)\}$ reads like:

```
 1 import numpy as np
 2 import matplotlib.pyplot as plt
 3 from numpy import exp
 4 from scipy.constants import pi, h, c, k
 5 
 6 def planck(f,T):
 7     return 2 * h * f**3 / (c**2 * exp(h*f / (k*T)) - 1)
 8 
 9 n = 67
10 x = np.linspace(0, 7e11, n)
11 y = planck(x, 2.75)
12 sigma = 0.001 * max(y)
13 e = sigma*np.random.randn(n)
14 
15 plt.errorbar(x, y, yerr=e)
16 plt.show()
```

For an error bar diagramm with $\sigma = 0{,}01 \cdot \max\limits_{f\in[0,\infty)} \{B(f,\ 2{,}75)\}$ line 12 must be modified:

```
sigma = 0.01 * max(y)
```

(or `1e-2` instead of `0.01`), and for $\sigma = 0{,}1 \cdot \max\limits_{f\in[0,\infty)} \{B(f,\ 2{,}75)\}$:

```
sigma = 0.1 * max(y)
```

Problem 5.1 Following the example of the case study from section 5.4.4 the program could look like as follows.

```python
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

#=====================================================================
daten = pd.read_csv("datasets/semiconductor-electron-mobility.csv", sep="\t")
#=====================================================================

x = np.c_[daten["x"]].ravel() # extracts feature values as a column vector
y = np.c_[daten["y"]].ravel() # extracts feature values as a column vector

plt.scatter(x,y)
plt.show()

def f(x,t0,t1,t2,t3,t4,t5,t6):
    return (t0 + t1 * x + t2 * x**2 + t3 * x**3) / (1 + t4 * x + t5 * x**2 + t6 * x**3)

coefs, cov = curve_fit(f, x, y, p0=(1000,1000,400,40,1,0.5,0.05))
y_pred = f(x, *coefs)   # model predictions
print("Fitted coefficients:", [round(t,3) for t in coefs])

xs = np.linspace(min(x), max(x), 100)  # take 100 points for a smooth regression line ...
plt.scatter(x, y)
plt.plot(xs, f(xs, *coefs), 'red')
plt.show()

# Model evaluation:
from sklearn.metrics import r2_score
R2 = r2_score(y, y_pred)
print("R² =", f"{R2:.3%}")

def bic(e, k):
    return len(e)*np.log(np.sum(e**2)/len(e)) + k*np.log(len(e))
print("BIC =", f"{bic(y - y_pred, len(coefs)):.0f}")
```

Results: The Python `curve_fit` solves this problem classified as difficult. Especially, we obtain $R^2 = 99.951\%$, BIC = 211. That is, the coefficient of determination is rather well, the value of the BIC is meaningful only in comparison to another model.

Problem 5.2 Following the example of the case study from section 5.4.4 the program could look like as follows.

```python
%matplotlib inline
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt

# 1.  Import data as a Pandas DataFrame, preprocess them for scipy curve_fit, and plot them:
df = pd.read_csv("./datasets/advertizing-and-sales.csv", sep='\t')
features = ["Advertizing expenses"]
target = "Sales volume"             # dependent variable
X = np.c_[df[features]].ravel()     # extracts feature values as a column vector
y = np.c_[df[target]].ravel()       # extracts target values as a column vector

plt.scatter(X, y)
plt.show()

# 2.  Curve fit:
def f1(x,a,b)  : return a + b*x
def f2(x,a,b)  : return a + b*np.sqrt(x)
def f3(x,a,b,c): return a + b*x**c
def f4(x,a,b,c): return a + b*np.exp(c*x)
coefs1, cov1 = curve_fit(f1, X, y)
coefs2, cov2 = curve_fit(f2, X, y)
coefs3, cov3 = curve_fit(f3, X, y, p0=(1,1,0.1))
coefs4, cov4 = curve_fit(f4, X, y, p0=(1,-1,0.05))

# 3.  Output model parameters:
```

```
27 print("Coefficients of models:")
28 print(" linear:",      [round(t,2) for t in coefs1])
29 print(" square root:", [round(t,2) for t in coefs2])
30 print(" power:",       [round(t,2) for t in coefs3])
31 print(" exponential:", [round(t,2) for t in coefs4])
32
33 # 4.  Plot data and regression curves:
34 xp = np.linspace(min(X), max(X), 100)   # take 100 points for smooth regression lines ...
35 plt.scatter(X, y)
36 plt.plot(xp, f1(xp, *coefs1), label="linear model")
37 plt.plot(xp, f2(xp, *coefs2), label="square root model")
38 plt.plot(xp, f3(xp, *coefs3), label="power model")
39 plt.plot(xp, f4(xp, *coefs4), label="exponential model")
40 plt.legend()
41 plt.show()
42
43 # 4 Evaluate models
44 # 4.1 Cofficients of determination:
45 from sklearn.metrics import r2_score
46 R2_1 = r2_score(y, f1(X, *coefs1))
47 R2_2 = r2_score(y, f2(X, *coefs2))
48 R2_3 = r2_score(y, f3(X, *coefs3))
49 R2_4 = r2_score(y, f4(X, *coefs4))
50
51 print("Coefficients of determination")
52 print(
53     " linear:", f"{R2_1:.2%},", "\tsquare root:", f"{R2_2:.2%},",
54     "\tpower:", f"{R2_3:.2%},", "\texponential:", f"{R2_4:.2%}"
55 )
56
57 #4.2 BICs:
58 def bic(e, k):
59     return len(e)*np.log(np.sum(e**2)/len(e)) + k*np.log(len(e))
60
61 print(
62     " BIC_1 =",    f"{bic(y - f1(X, *coefs1), len(coefs1)):.1f},",
63     "\t\tBIC_2 =", f"{bic(y - f2(X, *coefs2), len(coefs2)):.1f},",
64     "\t\tBIC_3 =", f"{bic(y - f3(X, *coefs3), len(coefs3)):.1f},",
65     "\t\tBIC_4 =", f"{bic(y - f4(X, *coefs4), len(coefs4)):.1f}"
66 )
```

Note the input of the initial values `p0` for the model parameters in lines 22 and 23. The estimated model parameters and the corresponding coefficients of determination are listed in the following table.

Model	**Parameter**	R^2	**BIC**
linear:	$f(x) = 112.64 + 6,78\,x$	${R_1}^2 = 95.31\%$	82.4
square root:	$f(x) = 31.71 + 49,65\,\sqrt{x}$	${R_2}^2 = 98.67\%$	63.4
power:	$f(x) = 9.55 + 65.79\,x^{0.44}$	${R_3}^2 = 98.72\%$	65.5
exponential:	$f(x) = 359,22 - 283,84\,\mathrm{e}^{-0,05}$	${R_4}^2 = 98,82\%$	64.3

(A.2)

The regression curves of the models are shown in Figure A.1. Results: Thus, the exponential model has the best goodness of fit with a coefficient of determination of 98.82

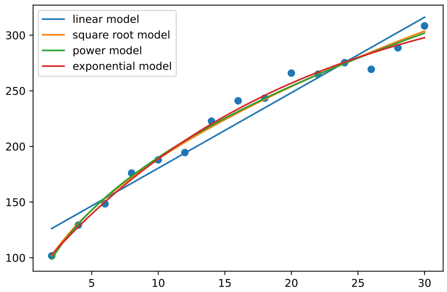


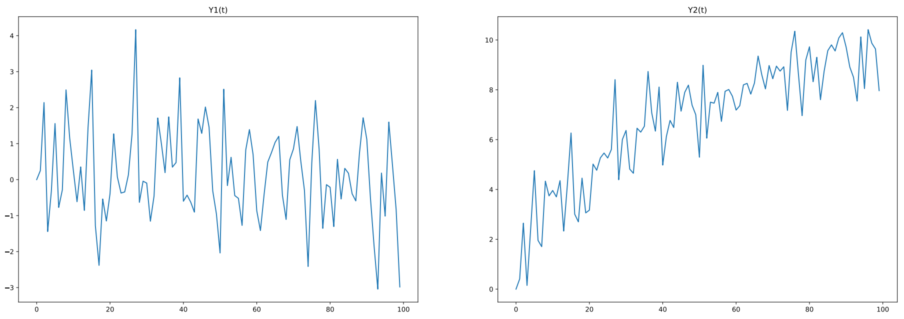
Figure A.1. Regression curves showing the effect of advertising

%. However, if one weighs the complexity of the models against R^2, one can argue with the BIC – and thus with Occam's razor – and conclude that, due to the lower number of parameters, the square root model with the lowest BIC of 63.4 can be considered as the "best" one.

Problem 7.1 (a)

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

def Y1(t, e):
    return np.sin(t) + e

def Y2(t, e):
    return np.sqrt(t) + e

size = 100
e = np.random.randn(size)
y1 = np.zeros(size)
y2 = np.zeros(size)

for t in range(1,size):
    y1[t] = Y1(t,e[t])

for t in range(1,size):
    y2[t] = Y2(t,e[t])

fig = plt.figure(figsize=(24,8))
ax1 = fig.add_subplot(121)
ax1.set_title("Y1(t)")
ax1.plot(y1)
ax2 = fig.add_subplot(122)
ax2.set_title("Y2(t)")
ax2.plot(y2)
plt.show()
```

The function plots are then given as follows:



(b) For Y_1, the mean and the standard deviation are constant in each case, i.e., Y_1 is stationary. However, since the mean and the standard deviation of Y_2 depend on time, Y_2 cannot be stationary. We can see these two findings in the function graphs: Y_1 moves in a corridor between -4 and 4 around its mean, while Y_2 shows growth.

(c*) The mean of f on the intervall $[0, t]$ is given approximately for great t by

$$\mu(t) = \frac{1}{t}\int_0^t f(\tau)\ \mathrm{d}\tau. \tag{A.3}$$

The innovation terms ε_t play no role here since they have the mean value zero. For μ_1 and μ_2 from (b), the following results are obtained

$$\mu_1(t) = \frac{1}{t}\int_0^t \sin\tau\ \mathrm{d}\tau = -\left.\frac{\cos\tau}{\tau}\right|_0^t = \frac{1-\cos t}{t} \quad\to\quad 0 \qquad \text{as } t \to \infty$$

and

$$\mu_2(t) = \frac{1}{t}\int_0^t \sqrt{\tau}\ \mathrm{d}\tau = \left.\frac{2}{3}\sqrt{\tau}\right|_0^t = \frac{2}{3}\sqrt{t} \quad\to\quad \infty \qquad \text{as } t \to \infty.$$

Thus for large values of t the mean of the first time series converges to 0, while that of the second diverges with $O(\sqrt{t})$.

Problem 8.1 (a) An implementation according to the task may look like as follows:

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

def AR1(phi1):
    size = 1000
    e = np.random.randn(size)
    y = np.zeros(size)
    for t in range(1,size):
        y[t] = phi1 * y[t-1] + e[t]
    return y

fig = plt.figure(figsize=(24,8))
ax1 = fig.add_subplot(411)
ax1.set_title("AR1(-1)")
ax1.plot(AR1(-1))
ax2 = fig.add_subplot(412)
ax2.set_title("AR1(0.5)")
ax2.plot(AR1(0.5))
ax3 = fig.add_subplot(413)
ax3.set_title("AR1(1)")
ax3.plot(AR1(1.0))
ax4 = fig.add_subplot(414)
ax4.set_title("AR1(1.01)")
ax4.plot(AR1(1.01))
plt.savefig('AR1-processes.pdf')
plt.show()
```

The otput is depicted in Figure A.2.

(b) For the first three graphs, the mean μ of the respective time series is zero, the variance of the second time series is $\sigma^2 = 1$, and of the third (a random walk) $\sigma^2 = t$.

(c) An AR(1) process with $|\varphi| < 1$ is stationary; with $\varphi = 1$ it is a random walk (example 7.4) and therefore non-stationary with example 8.5; with $\varphi > 1$ it is hyperexponentially growing or shrinking.

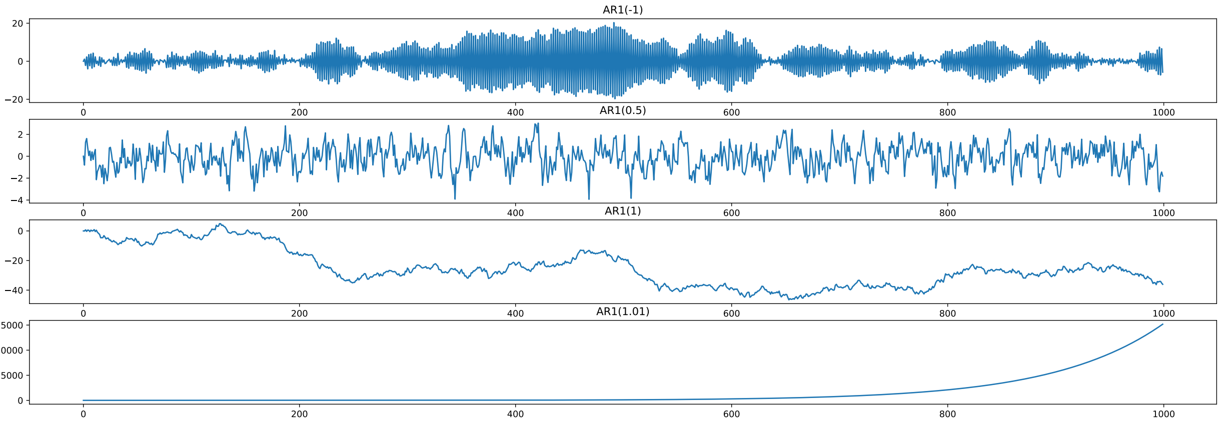


Figure A.2. AR(1)-Prozesse mit den Parametern $\varphi = 0.5$, $\varphi = 1$, $\varphi = 1.01$.

Problem 10.1 (a) = steps 1 and 2, (b) = step 3, (c) = steps 4 and 5:

```
import os, pandas as pd, matplotlib.pyplot as plt
from statsmodels.tsa.statespace.sarimax import SARIMAX

# 1.  Einlesen der CSV-Datei in Pandas:
verzeichnis  = "datasets"
datei  = "GDPC1.csv" # https://fred.stlouisfed.org/series/GDPC1

df = pd.read_csv(os.path.join(verzeichnis,  datei), sep=",")
df.index = pd.DatetimeIndex(df['DATE'], freq='QS-Apr')
del df['DATE']
df.rename(columns = {'GDPC1':'Real GDP'}, inplace=True)

# 2.  Filtern der gewünschten Daten und als Pandas Series speichern:
y = pd.Series(df['1947-04-01':'1978-04-01'][df.columns[0]])

# 3.  Konfigurieren der Trainings und Testdaten:
train_size = int(len(y) * 0.7)
test_size  = len(y) - train_size
y_train = y[:y.index[train_size]] # hier als Series
y_test  = y[y.index[train_size]:] # hier als Series

# 4.  Modell SARIMA((2,0,0) × (0,0,0)_0) anwenden:
model = SARIMAX(y_train, order=(2,0,0), trend='ct')
model = model.fit()
print(model.summary())

# 5.  Out-of-sample Prognose:
pred = model.predict(start=train_size, end=train_size+test_size-1)
pd.DataFrame({'Real':y,'Prediction':pred}).plot()
plt.show()
```

Instead of the `predict` method in step 5, we could have used the `forecast` method here:

```
# 5.  Out-of-sample Prognose:
forecast = model.forecast(steps=test_size - 1)
```

```
pd.DataFrame({'Real':y,'Forecast':forecast}).plot()
plt.show()
```

The respective function plots are depicted in Figure A.3. Note that in step 1 the measurement period could have been set as "QS-Jan", or also "QS-Jul" oder "QS-Oct".

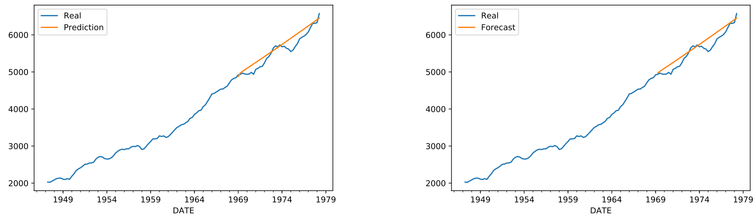


Figure A.3. Trajectory of the time series and prediction of the test data with `predict` (left) or with `forecast` (right)

A.2 Heuser about the Samuelson multiplyer

The mathematician Harro Heuser (1927–2011) describes in Chapter 7 "Recursive definitions and inductive proofs. Combinatorics" of his textbook on calculus[1] *the idea of Samuelson's accelerator as an application of the geometric series in economics. Here the entire section is reproduced in its original wording, in a translation of one of the authors (de Vries). The term "mark" here denotes the legal currency valid in Germany from June 21, 1948 to December 31, 1998.*

Impact of investment on national income

Suppose that the (producing and consuming) members of an economy consistently spend a fraction q $(0 < q < 1)$ of their income on consumption goods (q is the *limit propensity to consume*). Now let an entrepreneur make an investment of K marks (building a factory, purchasing machinery, etc.). According to our assumption, the *recipients* of this amount (bricklayers, plumbers, machine builders, ...) spend qK marks, the recipients of this amount (secondary recipients) consume q^2K marks, etc. After the n-th recipients have spent q^nK marks, total expenditures in the amount of

$$K \sum_{i=0}^{n} q^i = K \, \frac{1-q^{n+1}}{1-q} = K \, \frac{1}{1-q} - K \frac{q^{n+1}}{1-q} \text{ Mark} \tag{A.4}$$

have been made, and by this amount the national income has increased (Samuelson und Nordhaus (cf. 1995:§24.B)). [...] Thus, because of (A.4), $K/(1-q)$ [for large n] will sufficiently well indicate the increase in national income brought about by the initial investment of K marks.

[1] Heuser (1980):S. 67.

Bibliography

Backhaus, K., B. Erichson, and R. Weiber (2015). *Fortgeschrittene Multivariate Analysemethoden*. 3rd ed. Springer Gabler: Berlin Heidelberg. DOI: 10.1007/978-3-662-46087-0.

Backhaus, K. et al. (2016). *Multivariate Analysemethoden*. 14th ed. Springer Gabler: Berlin Heidelberg. DOI: 10.1007/978-3-662-56655-8.

Bandelow, C. (1989). *Einführung in die Wahrscheinlichkeitstheorie*. 2nd ed. BI Wissenschaftsverlag: Mannheim Wien Zürich.

Bauer, H. (1991). *Wahrscheinlichkeitstheorie*. 4th ed. Walter de Gruyter: Berlin New York.

Blanchard, O. J. (May 1981). "What is Left of the Multiplier Accelerator?" In: *The American Economic Review* 71(2). http://www.jstor.org/stable/1815709, pp. 150–154.

Bofinger, P. (2007). *Grundzüge der Volkswirtschaftslehre. Eine Einführung in die Wissenschaft von Märkten*. 2nd ed. Pearson Studium: München.

Brandt, S. (1999). *Datenanalyse*. 4th ed. Spektrum Akademischer Verlag: Heidelberg Berlin.

Brockwell, P. J. and R. A. Davis (1991). *Time Series: Theory and Methods*. 2nd ed. Springer. DOI: 10.1007/978-1-4419-0320-4.

— (2016). *Introduction to Time Series and Forecasting*. 3rd ed. Springer. DOI: 10.1007/978-3-319-29854-2.

Burke, K. D. et al. (2018). "Pliocene and Eocene provide best analogs for near-future climates". In: *Proceedings of the National Academy of Sciences* 115(52), pp. 13288–13293. ISSN: 0027-8424. DOI: 10.1073/pnas.1809600115.

Büttner, U. (2008). *Weimar. Die überforderte Republik 1918–1933*. Klett-Cotta: Stuttgart.

Campbell, J. Y., A. W. Lo, and A. C. MacKinlay (1997). *The Econometrics of Financial Markets*. Princeton University Press: Princeton.

Cowpertwait, P. S. P. and A. N. Metcalfe (2009). *Introductory Time Series with R*. Springer: Dordrecht Heidelberg London New York. DOI: 10.1007/978-0-387-88698-5.

de Vries, A. (2020). *Netzökonomie Lerneinheit 2. Einführung in das maschinelle Lernen mit Python*. Vorlesungsskript. Hagen. URL: https://fh-swf.sciebo.de/s/y80tdcBeby5mxow.

Deistler, M. and W. Scherrer (2018). *Modelle der Zeitreihenanalyse*. Birkhäuser: Cham. DOI: 10.1007/978-3-319-68664-6.

Denis, D. J. (2021). *Applied Univariate, Bivariate, and Multivariate Statistics Using Python: A Beginner's Guide to Advanced Data Analysis*. 2nd ed. Wiley: Hoboken. ISBN: 9781119578147. DOI: 10.1002/9781119583004.

Downey, A. B. (2011). *Think Stats. Probability and Statistics for Programmers*. 1st ed. Green Tea Press: Needham, Massachusetts. URL: https://greenteapress.com/thinkstats/.

Durbin, J. and S. J. Koopman (2012). *Time Series Analysis by State Space Methods*. 2nd ed. Oxford University Press: Oxford.

Felderer, B. and S. Homburg (1989). *Makroökonomik und neue Makroökonomik*. 4th ed. Springer-Verlag: Berlin etc.

Feldman, D. R. et al. (2015). "Observational determination of surface radiative forcing by CO_2 from 2000 to 2010". In: *Nature* 519(7543), pp. 339–343. DOI: 10.1038/nature14240.

Flach, P. A. and A. C. Kakas, eds. (2000). *Abduction and Induction: Essays on their Relation and Integration*. Applied Logic Series. Springer Science+Business Media: Dordrecht. DOI: 10.1007/978-94-017-0606-3.

Géron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly: Sebastopol.

Goldberg, S. (1958). *Introduction to Difference Equations*. John Wiley & Sons: New York.

Goswami, A. (1997). *Quantum Mechanics*. 2nd ed. Wm. C. Brown publishers: Dubuque, IA.

Handl, A. and T. Kuhlenkasper (2017). *Multivariate Analysemethoden: Theorie und Praxis mit R*. 3rd ed. Statistik und ihre Anwendungen. Springer: Berlin, Heidelberg. DOI: 10.1007/978-3-662-54754-0.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning*. 2nd ed. Springer: New York. DOI: 10.1007/b94608.

Heisenberg, W. (1948). "Der Begriff »abgeschlossene Theorie« in der modernen Naturwissenschaft". In: *Dialectica* 2, pp. 331–336.

Heuser, H. (1980). *Lehrbuch der Analysis. Teil 1*. B.G. Teubner: Stuttgart.

Hull, J. C. (2000). *Options, Futures & Other Derivatives*. 4th ed. Prentice-Hall International: Upper Saddle River, NJ.

IPCC (Aug. 2021). *AR 6 Climate Change 2021. The Physical Science Basis*. Cambridge University Press: Cambridge New York. URL: https://www.ipcc.ch/report/ar6/wg1/.

James, G. et al. (2013). *An Introduction to Statistical Learning*. Springer: New York Heidelberg Dordrecht London. DOI: 10.1007/978-1-4614-7138-7.

Jouzel, J. et al. (Aug. 2007). "Orbital and Millennial Antarctic Climate Variability over the Past 800,000 Years". In: *Science* 317(5839), pp. 793–797. DOI: 10.1126/science.1141038.

Kalvelage, T. (2018). "Chronisten der Erdgeschichte". In: *Spektrum der Wissenschaft* 10, pp. 50–55. URL: https://spektrum.de/artikel/1757380.

Katzenberger, M. (2013). *Algorithmen zur Losgrößenoptimierung*. Vol. 3. Hagener Berichte der Wirtschaftsinformatik. Books on Demand: Norderstedt.

Koyré, A. (1992). *The Astronomical Revolution. Copernicus – Kepler – Borelli*. Dover Publications: New York.

Krahl, D., U. Windheuser, and F.-K. Zick (1998). *Data Mining: Einsatz in der Praxis*. Addison-Wesley-Longman: Bonn. ISBN: 9783827313492.

Kreiß, J.-P. and G. Neuhaus (2006). *Einführung in die Zeitreihenanalyse*. Springer: Berlin Heidelberg. DOI: 10.1007/3-540-33571-4.

Krugman, P. R. and R. E. Wells (2006). *Macroeconomics*. Worth Publishers: New York.

Kuhn, T. S. (1962). *The Structure of Scientific Revolutions*. University of Chicago Press: Chicago.

MacKay, D. (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press: Cambridge.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill. ISBN: 9780071154673.

Murphy, J. J. (2016). *Technische Analyse der Finanzmärkte*. FinanzBuch Verlag: München.

Murphy, K. P. (2012). *Machine Learning. A Probabilistic Perspective*. MIT Press: Cambridge London.

Neusser, K. (2011). *Zeitreihenanalyse in den Wirtschaftswissenschaften*. 3rd ed. Vieweg + Teubner: Wiesbaden. DOI: 10.1007/978-3-8348-8653-8.

Ng, A. and K. Soo (2018). *Data Science – was ist das eigentlich?! Algorithmen des maschinellen Lernens verständlich erklärt*. Springer: Berlin. DOI: 10.1007/978-3-662-56776-0.

O'Neill, B. C. et al. (2017). "The roads ahead: Narratives for shared socioeconomic pathways describing world futures in the 21st century". In: *Global Environmental Change* 42, pp. 169–180. ISSN: 0959-3780. DOI: 10.1016/j.gloenvcha.2015.01.004.

Palma, W. (2016). *Time Series Analysis*. Wiley: Hoboken.

Penrose, R. (2004). *The Road to Reality*. Vintage Books: New York.

Pole, A., M. West, and J. Harrison (1994). *Applied Bayesian Forecasting and Time Series Analysis*. Chapman & Hall: New York.

Pourret, O., P. Naim, and B. Marcot, eds. (2008). *Bayesian Networks. A Practical Guide to Applications*. John Wiley & Sons: Chichester.

Rinne, H. and K. Specht (2002). *Zeitreihen. Statistische Modellierung, Schätzung und Prognose*. Vahlen: München.

Rotmans, J. et al. (2000). "Visions for a sustainable Europe". In: *Futures* 32(9–10), pp. 809–831. URL: https://www.pik-potsdam.de/ateam/avec/peyresq2003/internal/articles_pdf/europe_scenarios.pdf.

Russell, S. J. and P. Norvig (2022). *Artificial Intelligence. A Modern Approach*. Pearson: Harlow.

Samuelson, P. A. (1939). "Interactions between the Multiplier Analysis and the Principle of Acceleration". In: *The Review of Economics and Statistics* 21(2), pp. 75–78. DOI: 10.2307/1927758.

Samuelson, P. A. and W. D. Nordhaus (1995). *Economics*. 15th ed. McGraw-Hill: New York etc.

Scheck, F. (2013). *Theoretische Physik 2. Nichtrelativistische Quantentheorie*. 3rd ed. Springer Spektrum: Berlin Heidelberg.

Schönwiese, C.-D. (2008). *Klimatologie*. 3rd ed. Eugen Ulmer: Stuttgart.

Sen, A. K. and M. S. Srivastava (1990). *Regression Analysis: Theory, Methods and Applications*. Springer Texts in Statistics. Springer: Berlin Heidelberg. DOI: 10.1007/978-3-662-25092-1.

Shumway, R. H. and D. S. Stoffer (2017). *Time Series Analysis and Its Applications*. 4th ed. Springer: Cham. DOI: 10.1007/978-3-319-52452-8.

Silver, D. et al. (2017). *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. http://arxiv.org/abs/1712.01815.

Spiess, A.-N. and N. Neumeyer (2010). "An evaluation of R^2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a Monte Carlo approach". In: *BMC Pharmacology* 10(6). ISSN: 1471-2210. DOI: 10.1186/1471-2210-10-6.

Subasi, A. (2020). *Practical Machine Learning for Data Analysis Using Python*. Academic Press Elsevier: London. ISBN: 9780128213803.

Tabachnick, B. G. and L. S. Fidell (2018). *Using Multivariate Statistics*. 7th ed. Pearson Education: Harlow. ISBN: 9780134790541.

Tollefson, J. (2020). "How hot will Earth get by 2100?" In: *Nature* 580, pp. 443–445. DOI: https://doi.org/10.1038/d41586-020-01125-x.

Unsöld, A. and B. Baschek (1999). *Der neue Kosmos. Einführung in die Astronomie und Astrophysik*. 6th ed. Springer Verlag: Berlin Heidelberg New York.

VanderPlas, J. (2018). *Data Science mit Python*. 1st ed. mitp: Frechen.

Vogel, J. (2015). *Prognose von Zeitreihen. Eine Einführung für Wirtschaftswissenschaftler*. Springer Gabler: Wiesbaden. DOI: `10.1007/978-3-658-06837-0`.

von Weizsäcker, C. F. (1985). *Aufbau der Physik*. Carl Hanser Verlag: München Wien.

Weigend, M. (2019). *Python 3. Das umfassende Praxisbuch*. 8th ed. mitp: Frechen.

Wermuth, N. and R. Streit (2007). *Einführung in statistische Analysen. Fragen beantworten mit Hilfe von Daten*. Springer-Verlag: Berlin Heidelberg. DOI: `10.1007/978-3-540-33931-1`.

WMO, ed. (2021). *WMO Atlas of Mortality and Economic Losses from Weather, Climate and Water Extremes (1970–2019)*. WMO-No. 1267. World Meteorological Organization: Genève. ISBN: 978-92-63-11267-5. URL: `https://library.wmo.int/doc_num.php?explnum_id=10769`.

Zeh, H. D. (2012). *Physik ohne Realität: Tiefsinn oder Wahnsinn?* Springer: Berlin Heidelberg. DOI: `10.1007/978-3-642-21890-3_5`.

Internet References

[MP] `https://matplotlib.org/stable/tutorials/` – Matplotlib tutorial

[NumPy] `https://realpython.com/numpy-tutorial/` NumPy tutorial

[Py] `https://docs.python.org` – Python documentation

[PyK] `https://www.python-kurs.eu/` – Python tutorial (German)

[PyW] `https://www.w3schools.com/python/` – Python tutorial on w3schools

[SciPy] `https://scipy-lectures.org/` – SciPy lecture notes

[SKL] `https://scikit-learn.org` – free software machine learning library for Python

Index