Technische Universität Wien
Vienna University of Technology

DIPLOMARBEIT

Abnormal Event Detection By Using Data Mining And Machine Learning Methods

Modelling Normality and Anomalies

Ausgeführt am Institut für
Analysis & Scientific Computing
der
Technischen Universität Wien
unter der Anleitung von
Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Felix Breitenecker
durch

Matthias Wastian

Sonnenrain 11
A-9300 St. Veit/Glan

Wien, 25. November 2013

For my parents and Sarah, who always support me with their love

Zusammenfassung

In der vorliegenden Diplomarbeit werden unterschiedliche Ansätze diskutiert, um ungewöhnliche und außerordentliche Ereignisse, für die man eine separate und spezialisierte Betrachtung als wünschenswert erachtet, anhand regelmäßig mitprotokollierter Daten in dieser Zeitreihe möglichst frühzeitig zu detektieren und bei Möglichkeit sogar vorherzusagen. Dieser Problemstellung, die in einem einleitenden Kapitel zusammen mit unterschiedlichen Anwendungsbereichen, in denen ebendiese auftritt, etwas ausführlicher vorgestellt wird, wird mittels einiger Methoden aus dem Bereich des Data Mining, des Machine Learning und des Soft Computing hybrid begegnet.
Nach einer kurzen Grundlageneinführung zu Zeitreihen mitsamt den zugehörigen statistischen Vorhersagemodellen werden die erwähnten Begriffe näher abgesteckt, bevor der Fokus auf die einzelnen Teilmethoden gelegt wird. Auf eine Vorstellung von Werkzeugen zur Ausreißer- bzw. Novumserkennung folgt eine abschließende Diskussion der Simulationsergebnisse, die im Rahmen jenes Projekts erzielt wurden, in Zuge dessen diese Arbeit entstand. Der Text endet mit einem Ausblick auf mögliche Modellerweiterungen und zukünftige Arbeiten.

Abstract

This diploma thesis will discuss several approaches to detect unusual and extraordinary events, which we consider to be worth a separate and specialised further investigation, in a time series of frequently collected data as early as possible and - wherever applicable - to even predict them. We rise to this task, which will be introduced together with some different scopes of application in a more detailed way in the opening chapter, using various methods originating in the field of data mining, machine learning and soft computing in a hybrid manner.
Following a short and basic introduction to time series including several statistical prediction models, I will delimit and discuss these terms in general, before I will focus on the modular parts of the proposed methodology. After the presentation of some algorithms to detect outliers and novelties, the results of the simulation gained in the project this work was part of are put up for discussion. The text ends with some prospects of possible extensions and enhancements as well as future research work.

Contents

List of Figures

List of Tables

1 Problem Statements and Applications

Almost every system in the real world can be observed by measuring certain parameters at certain times, some of them being very important, others not so important and others maybe even completely irrelevant for the actual or future system behaviour. In general the parameters' importance for the system behaviour is a priori not known, it can vary over time and may also depend on the values of other observed or not observed parameters. The values taken by these parameters can be seen as the result of one or several events that took place inside or outside the system.

Definition 1 (Event)**.** An event is an occurrence happening at a determinable time and place and has a certain duration. It may be a part of a chain of occurrences as an effect of a preceding occurrence and as the cause of a succeeding occurrence. It is possible that more than one event occurs at the same time and/or place.

In many cases the observer of a system wants to detect severe changes of a system's behaviour as early as possible to have the opportunity to decide if any possible actions should be taken or not.

Example 1. The tectonic movements of the earth are observed constantly by GPS stations and seismic sensors especially in areas with a large risk of earthquakes. In case of a detected imminent earthquake, an evacuation of certain areas that are in danger of a possible tsunami could be ordered by the local governments. Such tsunami warning systems are established in Honolulu, Hawaii (Pacific Tsunami Warning Center PTWC), Malaysia, Taiwan and Indonesia (German Indonesian Tsunami Early Warning System GITEWS).

Example 2. In the local area network of a large company a huge amount of parameters like the number of logged in users, the number of mails sent, the free space left on a hard disc of a server or the percentage of the memory of a server that is used at a certain time can be measured. This thesis suggests a conglomerate of intelligent mathematical algorithms that analyses these parameters and detects problems in such a network that could cause server outages or a rapid decline in a server's performance. In case of a detected problem the network's administrator could carry out a software update, supply additional hardware or replace defect hardware as early as possible to guarantee the functionality of the server.

Before the term *abnormal event* is discussed in more detail, the terms *abnormal*, *outlier* and *novelty* are defined:

Definition 2 (Abnormal)**.** The Oxford English Dictionary defines abnormal as: deviating from the ordinary type, especially in a way that is undesirable or prejudicial; contrary to the normal rule or system; unusual, irregular, aberrant.

Definition 3 (Outlier)**.** An outlier is an observation that deviates so much from other observations of a set of data as to arouse suspicion that it was generated by a different mechanism. [20]

Definition 4 (Novelty)**.** Given a set of data not polluted by any outliers, new observations are made and checked whether they are regular *inliers*. A novelty is an anomaly in these new observations.

Definition 5 (Abnormal Event)**.** An abnormal event is an outlier in a chain of events, an event that deviates so much from the other events as to arouse suspicion that it was caused by something that does not follow the usual behaviour of the considered system and that it could change the entire system behaviour.

Applications of abnormal event detection can be found in a broad variety of areas, almost all of them following the idea to guarantee a certain level of safety of the system considered:

- natural catastrophes
    - earthquakes
    - floodings
    - hurricanes
- server outages
- stock market breakdowns
- network intrusion
- safeguards system for a series of nuclear fuel cycle facilities
- audio and video surveillance
    - security
    - video traffic analysis
    - crowd behaviour
    - ambient assisted living.

Due to the large variety of applications, various different approaches have been suggested for abnormal event detection. While this thesis is going to mainly focus on time series forecasting with artificial neural networks (section 3.4.5) and outlier detection with one-class support vector machines (section 4.4), some other possible methods to deal with abnormal event detection problems are listed below:

- sparse reconstruction cost [22]
- wavelet decomposition [23]
- statistical methods
  - explicit descriptors statistical model
  - bayes estimation
  - maximum likelihood
  - correlation analysis
  - principal component analysis (PCA).

2 Introduction to Time Series

Definition 6 (Time Series)**.** A sequence of vectors or scalars in successive order depending on time t is a so-called *time series.* The term univariate time series refers to such sequences of scalars, the term multivariate time series refers to sequences of vectors which all have the same dimension $m \geq 2$.

Remark. Although this does not necessarily have to be the case for time series, in this thesis only sequences spaced at (at least except for very small measurement errors) uniform time intervals are considered. The most recent time series element is going to be called x_n.

Remark. Multivariate time series are best understood as being a set of simultaneously built time series. The values of each series have not only an internal dependency within the series itself, but also an interdependency with the values of other component series.

Talking of a univariate time series (but analogue for multivariate time series), a time series usually contains observation values of a physical, financial or sociological variable made at equally spaced time intervals Δt, represented as a set of discrete real, rational, or (positive) integer numbers:

$$\begin{aligned}
\{x[t]\} &= \{x(0), x(\Delta t), \ldots, x\big((i-1)\Delta t\big), x(i\Delta t), x\big((i+1)\Delta t\big), \ldots\},\\
\{x[t]\} &= \{x(t_0), x(t_1), \ldots, x(t_{i-1}), x(t_i), x(t_{i+1}), \ldots\},\\
\{x[t]\} &= \{x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots\} = \{x_i : i \in \mathbb{N}\}.^1
\end{aligned}$$

Examples for various time series of interest are given below. Some of them origin from discrete phenomena, others from continuous ones:

- Dow Jones Industrial Average
- price for a barrel of Brent oil
- number of inhabitants of Austria
- real average retirement age or unemployment rate in a country
- temperature in a nuclear reactor
- electricity and water demand of a huge city
- tectonic movements in an area where earthquakes are likely to occur
- air pressure, temperature, other physical variables to forecast weather.

[1] In this thesis 0 is NOT an element of $\mathbb{N}$.

Whereas the Dow Jones Industrial Average closing values of each day are a classic discrete case, temperature is a typical example for a continuous signal.

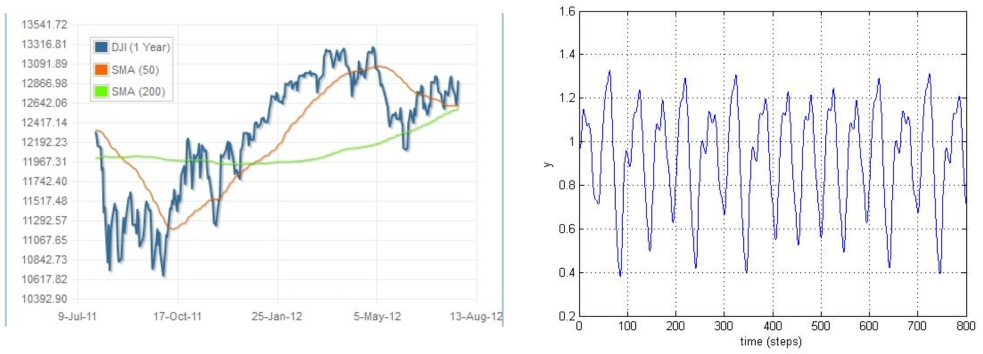


Figure 1: Graphs of two time series, on the left Dow Jones Industrial Average closing values and on the right a chaotic Mackey-Glass time series.

In engineering practice, the sequence of values $\{x[t]\}$ is obtained from sensors by uniformly sampling the related continuous signal $x(t)$ at discrete points $x_i, i \in \mathbb{N}$. The modeller wants to be able to interpolate $x(t)$ from $\{x[t]\}$ as true as possible, wherefore Δt must be chosen according to the Nyquist sampling theorem.

Definition 7 (Aliasing)**.** Aliasing refers to an effect that causes different signals to become indistinguishable (or aliases of one another) when sampled.

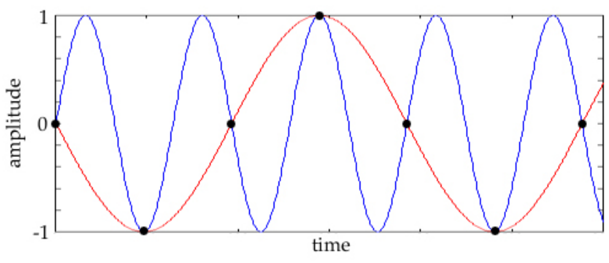


Figure 2: Aliasing. The blue graph is the original signal, the dots mark the sampling points. The red graph is an alias that is misleadingly recovered from the samples.

Theorem 1 (Nyquist-Shannon Sampling Theorem)**.** *If f_{max} is the highest frequency component of $x(t)$, the sampling rate must be larger than twice as high:*

$$\frac{1}{\Delta t} = f_{sampling} \; > \; 2f_{max} \qquad \Leftrightarrow \qquad \Delta t \; < \; \frac{1}{2f_{max}}. \tag{1}$$

If the inequality (1) *is obeyed, $x(t)$ is completely determined by giving its ordinates at a series of points spaced Δt seconds apart. If not, we would see aliasing of frequencies in the range $[f_{sampling}/2, f_{max}]$.*

Proof. Let $X(t)$ be the spectrum of $x(t)$ and W the bandwidth limit. Then

$$x(\omega) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{i\omega t}\mathrm{d}\omega = \frac{1}{2\pi}\int_{-2\pi W}^{2\pi W} X(\omega)e^{i\omega t}\mathrm{d}\omega$$

since $X(\omega)$ is assumed to be zero outside the band W. We choose $t = \frac{k}{2W}$ with $k \in \mathbb{Z}$ and obtain

$$x\Big(\frac{k}{2W}\Big) = \frac{1}{2\pi}\int_{-2\pi W}^{2\pi W} X(\omega)e^{i\omega\frac{k}{2W}}\mathrm{d}\omega.$$

On the left are values of $x(t)$ at the sampling points. The integral on the right will be recognized as the k^{th} coefficient in a Fourier-series expansion of the function $X(\omega)$, taking the interval $[-W, W]$ as the fundamental period. This means that the values of the samples $x\big(\frac{k}{2W}\big)$ determine the Fourier coefficients in the series expansion of $X(\omega)$. Thus they determine $X(\omega)$, since $X(\omega)$ is zero for frequencies greater than W, and for lower frequencies $X(\omega)$ is determined if its Fourier coefficients are determined. But $X(\omega)$ determines the original function $x(t)$ completely, since a function is determined if its spectrum is known. Therefore the original samples determine the function $x(t)$ completely. □

Basically, there are two approaches to time series analysis:

1. time domain approach, mainly based on the use of the covariance function of the time series, and

2. frequency domain approach, based on spectral density function analysis and Fourier analysis.

[1] states that both approaches are appropriate for application to a wide range of disciplines, but the time domain approach is mostly used in engineering practice. This is particularly due to the availability of the Box-Jenkins approach to time series analysis, which I will talk about later. As a matter of fact I only use the time domain approach in the simulation that is presented in section 5.

The broad term of time series analysis includes activities like

- definition, classification and description of time series
- model building using collected time series data (e.g. transforming one time series into another, for example oil prices into interest rates)
- forecasting or prediction of future values.

2.1 General Properties of Time Series

We distinguish between several major characteristic features of a time series $\{x[t]\}$, and although the time series can exhibit one or more of these properties, for the analysis and for a prediction each property is treated separately. Depending on the character of data that it carries, the time series could be

- univariate and multivariate
- stationary or non-stationary
- seasonal or non-seasonal
- linear or non-linear
- chaotic or random.

2.1.1 Deterministic and Stochastic Component

Most of the time series models look at the given time series values as the sum of true and predictable time series values and random disturbances which are expected to be Gaussian distributed.[2] Thus a time series, usually based on measured values that are corrupted by noise, can be considered as the composition of a deterministic and a stochastic signal. The deterministic component models the self-dependent aspects of the process, while the stochastic component simulates not only the usual noise interference, but also not known and not foreseeable aspects that cannot be modelled adequately. These not known aspects can either have a significant influence on the sequel of the process we look at (e.g. the sale of a huge amount of company shares by a single person or enterprise that makes other holders nervous, the announcement of the last years profit of a company that is very different to the expectations, a significant and maybe random change of a bank's or country's rating by

[2]This assumption has to be fulfilled for a variety of further approaches to time series analysis.

a rating agency, the outage of several memory components of a server) or they do not change the general behaviour of the process (e.g. the sale of many shares that does not have a large impact on the other holders, the outage of a memory component of a server which is compensated by other memory components). The goal of the proposed methods is to distinguish between such important abnormal events that influence the future quality of the process in a significant way and not so relevant events that do not.

Summing up and speaking from a modelling point of view that underlies this whole thesis, a time series $\{x[t]\}$ is the output of a process P we are interested in. In general, we do not have any additional knowledge about P and have to gain information from the output itself to detect important changes of P. Thus the process can be considered as a black box:

In this and the following chapters, various traditional approaches to time series classification, modelling and forecasting are introduced. They are needed for a better understanding of the modern approaches to time series analysis and forecasting using soft computing methods that will be presented especially in section 3.4.5.

2.1.2 Stationarity

For the following definition the time series is considered as a random process, i.e. a sequence of random variables, the elements of the time series.

Definition 8 (Stationarity)**.** A time series $\{x[t]\}$ is called stationary, if

1. $\mathbb{E}(x_t) = \mu < \infty \quad \forall t$, that is, the expectation of x_t is finite and does not change over time, and
2. $\mathbb{COV}(x_{t+\tau}, x_t) = \kappa_\tau < \infty \quad \forall \tau$, that is, for each τ the autocovariance of the random variables $x_{t+\tau}$ and x_t does not depend on time, but only on the distance between the two observations.

If we set $\tau = 0$, we see that this already includes the condition that the variance of x_t is also constant over time. If all x_t have the same finite variance, they are called *homoscedastic.*

Remark. The above given definition refers to the more exact statistical term (weak) stationarity. The difference to strict stationarity, which requests that all random vectors $(x_{t_1}, \ldots, x_{t_k})^T$ and $(x_{t_1+\tau}, \ldots, x_{t_k+\tau})^T$ have the same joint distribution for all sets of indices $\{t_1, \ldots, t_k\}$ and for all integers τ and $k > 0$, will not become important in this thesis.

The stationarity of a time series can be roughly checked by looking at the time series pattern. A flat-looking pattern with no seasonality or trend and with time-invariant variance indicates that this could be a stationary time series. Non-stationary time series can often be transformed into equivalent stationary time series by taking single or multiple differences between the successive data values along the time series pattern.

Theorem 2 (Wold-von Neumann Decomposition)**.** *Any stationary discrete time stochastic process can be decomposed into a pair of uncorrelated processes, one deterministic and the other being a moving average process (which will be introduced in section 2.2.2).*

Remark. The Wold-von Neumann decomposition originates from the field of operator theory and is a classification theorem for isometric linear operators on a given Hilbert space that states that any isometry is a direct sum of copies of the unilateral shift and a unitary operator.

2.1.3 Trend

Definition 9 (Trend)**.** The trend is the component of a time series that represents variations of low frequency in a time series, the high and medium frequency fluctuations having been filtered out.[3] Thus the trend determines the long-term behaviour of the time series that is manifested through the local or global increase or decrease of data values as a consequence of superposition of true time series values and a disturbance with upward or downward trend ([1]).

For identifying the trend present in an observed time series $\{x_1, \ldots, x_n\}$, one tries to fit the collected data by a certain relation as well as possible. Such relations (for a univariate time series) could be:

- linear: $\hat{x}_t = \lambda_1 t + \lambda_0 + \varepsilon_t$
- polynomial (degree $m \in \mathbb{N}$): $\hat{x}_t = \lambda_m t^m + \lambda_{m-1} t^{m-1} + \ldots + \lambda_0 + \varepsilon_t$
- exponential or log-linear: $\hat{x}_t = e^{\lambda_1 t + \lambda_0 + \varepsilon_t}$.

[3]Definition given by the OECD.

Example 3. Examples of linear models are the AR, MA, ARMA and ARIMA models (see section 2.2), based on autoregression and/or on a moving average technique.

Definition 10 (Regression)**.** Regression is the discovery of a predictive learning function - the regression function of the independent variables, which maps a data item to a real-value prediction variable, the dependent variable [3]. Regression analysis estimates the average value of the dependent variable when the independent variables are fixed. An autoregressive model tries to predict the output of a system based on the previous outputs, that means that the dependent and the independent variable become the same.

Assuming that the disturbances ε_t of the time series trend are normally distributed, the fitting is done by the *least-squares method.* λ contains the $m+1$ adjustable parameters, $f(x,\lambda)$ is the right hand side of the equation of the above relations without any disturbances (e.g. in the polynomial case $f(x,\lambda) = \lambda_m t^m + \lambda_{m-1} t^{m-1} + \ldots + \lambda_0$) and r is called *residual*, the difference between the actual value of the dependent variable and the value predicted by the model:

$$\min_{\lambda \in \mathbb{R}^{m+1}} \sum_{i=1}^{n} r_i^2 = \min_{\lambda \in \mathbb{R}^{m+1}} \sum_{i=1}^{n} (x_i - f(x_i, \lambda))^2. \tag{2}$$

Now the R^2-value can be calculated.

Definition 11 (Coefficient of Determination R^2)**.** Given n data points x_i with mean $\bar{x}$ and variance σ^2, the most general definition of R^2 is

$$R^2 = 1 - \frac{1}{n\sigma^2} \sum_{i=1}^{n} r_i^2 = 1 - \frac{\sum\limits_{i=1}^{n} (x_i - f_i)^2}{\sum\limits_{i=1}^{n} (x_i - \bar{x})^2}, \tag{3}$$

$$\text{with } \bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \text{ and } \sigma^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^2.$$

R^2 is the proportion of variance in a data set explained by the chosen statistical model and provides a measure of how good predictions by that model are likely to be.

Remark. In the case of linear least-squares regression, R^2 equals the square of the correlation coefficient between the observed and modelled data values. Sometimes R^2 is calculated as the square of the correlation coefficient, no matter whether a least-squares regression was done or not. If R^2 is calculated as stated in Definition 11, it is not the square of a real number R! If a very inadequate model is chosen, or if there are constraints that do not make sense for the given data, R^2 can also be negative.

Remark. R^2 does not relate to the statistical significance of the trend line, which is determined by its *t-statistic*. Very noisy series can have a low R^2-value, but a very significant trend. Nevertheless only trends of time series with R^2-values exceeding 0.65 are identified as meaningful.

2.1.4 Seasonality

Definition 12 (Seasonality)**.** Seasonality is defined by a periodically fluctuating pattern of the time series which leads to a repetitive and predictable movement around the trend line of the time series. The pattern may repeat itself hourly, daily, weekly, monthly, yearly etc., very often depending on the process the time series data is generated from.

Example 4. If a given time series represents the number of logged in users in the network of a company, it will exhibit a weekly seasonality. The number of logged in users on a Monday at 9 a.m. will strongly depend on the number of logged in users on the Monday the week before at 9 a.m., if none of these Mondays is a non-business day.

Example 5. Unemployment rates exhibit strong seasonal effects.

Removal of Seasonality. By removing the seasonal component, it is a lot easier to focus on other components like the trend. If the components of the decomposition of the time series - trend, seasonality, irregularity and cycle - act multiplicatively, the seasonal component is estimated and then the values are divided by it.
The *ratio-to-moving-average method* uses moving average windows for calculating the centered average time series values within the windows, whose width is exactly the length of the season. Each data value is expressed as percentage of the corresponding centered moving average value. These percentages are arranged according to weeks, months or quarters of given years, and the averages over all weeks, months or quarters are calculated and represent the seasonal index.

2.1.5 Chaos

Definition 13 (Chaotic Time Series)**.** A time series whose values are nonperiodic and highly sensitive to initial conditions, but result from a completely deterministic process (i.e., if you create two time series of this process with identical initial data and do not consider any noise, they are the same), is called chaotic. The deterministic nature of such processes does not necessarily make them predictable, especially in a long term.

Remark. One should notice the difference between the two terms *random* and *chaotic*, which are used more or less in the same way in daily life. Two successive realizations of a random process will give two different time series, even if the initial state is the same, i.e. a random process is non-deterministic.

Example 6. A so-called Mackey-Glass time series originates from the equation

$$x_{t+1} = \frac{a x_{t-d}}{b + x_{t-d}^h} - c x_t, \quad d, t \in \mathbb{N},\ a, b, c, h \in \mathbb{R} \backslash \{0\}. \tag{4}$$

and $d + 1$ given initial conditions. The right part of figure 1 on page 12 shows the graph of a Mackey-Glass time series with the parameters $a = 0.2, b = 1, c = -0.9, d = 17$ and $h = 10$ and the initial values listed in appendix B.1. Originally, Mackey and Glass presented equations of the form (4) to illustrate the appearance of complex dynamics in physiological control systems by bifurcations, which might arise from disease or environmental factors, such as drugs. A special Mackey-Glass equation is used in a model to describe the regeneration of white blood corpuscles.

For the following definition a dynamical system $x_{t+1} = f(x_t)$ is considered, where all x_t are vectors of $\mathbb{R}^m$.

Definition 14 (Lyapunov Exponent, Lyapunov Time)**.** Consider a univariate signal $\{x[t]\}$ of two possibly multivariate time series of the same process with two close points at step t, x_t and $x_t + \Delta x_t$. At the next time step they will have diverged, namely to x_{t+1} and $x_{t+1} + \Delta x_{t+1}$. It is this average rate of exponential divergence (or convergence) that the Lyapunov exponent captures. The Lyapunov exponents λ_i are given by

$$\lambda_i = \lim_{N \to \infty} \frac{1}{N} \sum_{t=1}^{N} \ln \left(\frac{\Delta x_{t+1}}{\Delta x_t} \right), \quad i = 1, \ldots, m \tag{5}$$

$\frac{1}{\lambda_i}$ is called Lyapunov time and gives an idea about the lapse of time which predictions are reliable for.

Definition 15 (Chaotic Time Series by Lyapunov Exponents)**.** A bounded time series $\{x[t]\}$ originating from the system $x_{t+1} = f(x_t)$ is called chaotic, if

1. $\{x[t]\}$ is not asymptotically periodic,

2. no Lyapunov exponent vanishes and
3. the largest Lyapunov exponent is strictly positive.

The following table lists the main properties of ordered, chaotic and random systems:

System	Order	Chaos	Randomness
Paradigm	Clock	Weather	Snow on TV Screen
Control	Easy	Tricky	Poor
Predictability	Very High	Short Term	None
Stability[4]	High	Very Low	No Wise Predictions
Dimension	Finite	Low	Infinite
Attractor[5]	Point, Cycle, Torus	Strange, Fractal	No

Table 1: A comparison of properties of ordered, chaotic and random systems.

2.2 Regression Models

The regressive models that are introduced in the following subsections are built using regression analysis, which is a collection of methods for the study of relationships between the variables and for estimation and prediction of values of one variable using the values of other variables incorporated in a joint time series.[6]

Because it is relevant to all the following models, the terms autocorrelation for time series and white noise are introduced before the presentation of the various regression models.

Definition 16 (Autocorrelation of Time Series)**.** Similar to the correlation coefficient r of two random variables x and y, a correlation coefficient between one time series $\{x[t]\}, t = 1, \ldots, n$ and the same series lagged by one

[4]A time series is called stable if small errors of the given data only lead to small errors of a prediction.

[5]An attractor is a set towards which a system variable evolves over time. That is, points close enough to the attractor remain close when disturbed.

[6]Definition by Draper and Smith. *Applied Regression Analysis*, 2nd edition, Wiley, New York, 1981.

or more time units can be calculated. If the lag is d time units, the autocorrelation coefficient r_d is the correlation coefficient of the observations $\{x[t]\}, t = 1, \ldots, n-d$ and the observations $\{x[t]\}, t = d+1, \ldots, n$. For n reasonably large, r_d can be approximated as follows:

$$r_d \approx \frac{\sum_{t=1}^{n-d} (x_t - \bar{x})(x_{t+d} - \bar{x})}{\sum_{t=1}^{n} (x_t - \bar{x})}. \tag{6}$$

The autocorrelation matrix $\mathbb{R}$ is built analogously to the covariance matrix and therefore

$$\mathbb{R} = \begin{pmatrix} r_0 & r_1^* & r_2^* & \cdots & r_{n-1}^* \\ r_1 & r_0 & r_1^* & \cdots & r_{n-2}^* \\ r_2 & r_1 & r_0 & \cdots & r_{n-3}^* \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{n-1} & r_{n-2} & r_{n-3} & \cdots & r_0 \end{pmatrix}. \tag{7}$$

Remark. $\mathbb{R}$ is positive semi-definite for stationary processes. For time series the equation $r_0 = 1$ holds.

Definition 17 (Autocorrelation Function, Partial Autocorrelation Function)**.** The sequence $\{r_i\}, i = 0, \ldots, n-1$, according to a time series $\{x[t]\}, t = 1, \ldots, n$, is called autocorrelation function (ACF). If first any linear dependence on the time series elements between those two elements that the autocorrelation is calculated for is removed, the sequence of autocorrelations is called partial autocorrelation function (PACF).

The ACF and the PACF are mathematical tools to find repeating patterns.

Definition 18 (White Noise Process)**.** A white noise process is a random process of a random variable x if and only if its mean vector and autocorrelation matrix $\mathbb{R}$ are the following:

1. $\mathbb{E}(x) = 0$,
2. $\mathbb{R}(x) = \mathbb{E}(xx^H) = \sigma^2 I$.

That means, a white noise process is a random process of random variables that are uncorrelated, have mean zero and a finite variance.

Example 7. The snow on the TV screen is visual white noise. Acoustic white noise sounds similar to static electricity discharge or a hiss. In a linguistic sense white noise is any random, collective occurrence of unrelated things.

2.2.1 AR Models

Algorithm 1 (AR(p) Model)**.** Autoregression models express the current value of a time series by a finite linear aggregate of previous values, by a white noise ε_t and sometimes a real constant c. Their validity assumes that the time series to be modelled is stationary, to assure stability the model parameters ϕ_i have to be within a certain range.
$p \in \mathbb{N}$ is the so-called *model order*, being the number of previous outputs that determine the actual output in the model:

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \ldots + \phi_p x_{t-p} + \varepsilon_t + c. \tag{8}$$

Remark. For an AR(p) model to be stationary, the roots of the polynomial $z^p - \sum_{i=1}^{p} \phi_i z^{p-i}$ must lie within the unit circle, i.e., each root z_i must satisfy $|z_i| < 1$.

Definition 19 (Delay Operator)**.** The delay operator B - also called backshift operator - is defined by

$$B(x_t) = x_{t-1}. \tag{9}$$

Definition 20 (Autoregressive Operator)**.** The autoregressive operator of order p is defined as follows:

$$\phi(B) = 1 - \sum_{i=1}^{p} \phi_i B^i. \tag{10}$$

Usually the time series is transformed to mean zero by $\tilde{x}_t = x_t - \mu$. Using the autoregressive operator, the AR(p)-model can be written in the compact form

$$\phi(B)\tilde{x}_t = \varepsilon_t. \tag{11}$$

Estimation of the parameters of the AR(p) model. All in all, an AR(p) model contains $p+1$ unknown parameters: the p internal parameters ϕ_i and σ_ε^2, the variance of the white noise. There are many ways to estimate the coefficients ϕ_i, among them the ordinary least-squares method, the Markov chain Monte Carlo method or the method of moments by using the *Yule-Walker equations*, which are created by multiplying (8) by an $x_i, i = t-1, \ldots, t-p-1$, taking the expectance, dividing by $n-1$ and using the evenness of the autocovariance, leading to

Algorithm 2 (Yule-Walker Equations).

$$\begin{pmatrix} 1 & r_1 & r_2 & \dots & r_{p-1} \\ r_1 & 1 & r_1 & \dots & r_{p-2} \\ r_2 & r_1 & 1 & \dots & r_{p-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r_{p-1} & r_{p-2} & r_{p-3} & \dots & 1 \end{pmatrix} \begin{pmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \vdots \\ \phi_p \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_p \end{pmatrix}, \tag{12a}$$

$$\text{or shorter: } \mathbb{R}\phi = r, \tag{12b}$$

$$\text{and } \sigma_\varepsilon^2 = 1 - \sum_{k=1}^{p} \phi_k r_k. \tag{13}$$

Selection of the order of the AR(p) model. The crucial point of an adequate selection of the model order p is done by an analysis of the autocorrelation function or in the case of lower order models sometimes the partial autocorrelation function which is computationally complicated. A further discussion on how to find a good AR model order will be presented in chapter 2.2.4 where various models are combined to one model.

2.2.2 MA Models

The basic assumption behind averaging and smoothing models like the MA models is that the time series is not globally stationary, but at least locally stationary with a slowly varying mean. Hence, a moving (i.e., local) average is taken to estimate the current value of the mean.

Remark. The moving average is often called a *smoothed* version of the original series, since short-term averaging has the effect of smoothing out the bumps in the original series.

Algorithm 3 (MA(q) Model). The MA(q) model is conceptually a linear regression of the current value of the series against previous (unobserved) white noise error terms ε_{t-i}. Again, the random shocks at each point are assumed to come from a normal distribution with mean zero and finite variance. The MA(q)-model suggests that these random shocks are propagated to future values of the time series.
$q \in \mathbb{N}$ is the model order, μ the mean of the series, θ_i are the model parameters.

$$x_t = \mu + \varepsilon_t - \theta_1 \varepsilon_{t-1} - \theta_2 \varepsilon_{t-2} - \ldots - \theta_q \varepsilon_{t-q} \tag{14}$$

Definition 21 (Moving-Average Operator)**.** The moving-average operator of order q is defined analogously to the autoregressive operator (10):

$$\theta(B) = 1 - \sum_{i=1}^{q} \theta_i B^i, \tag{15}$$

B being the delay operator (9).

Thus the MA(q) model can be written in the compact form

$$\tilde{x}_t = \theta(B)\varepsilon_t. \tag{16}$$

Remark. Estimating the MA parameters is more complicated than estimating the parameters ϕ_i in AR models, because the error terms are not observable. This means that iterative non-linear fitting procedures need to be used in place of linear least squares. MA models also have a less obvious interpretation.

Remark. An MA process is always stationary.

2.2.3 I Models

In this section the method of differencing time series is going to be introduced, which is the key point of *integrated* models.

A variety of time series exhibit non-stationary behaviour, but some of them become stationary after differencing the time series $\{x[t]\}$ with n elements d times in a non-seasonal way, which means building a new time series consisting of the $n - d$ successive differences of successive time series values. For example, a time series with a linear trend becomes stationary by differencing the time series once. The new time series does not exhibit any trend any more, i.e. the *detrending* of the time series was successful.

Definition 22 (Differencing Operator)**.** The differencing operator ∇ shall be introduced as

$$\nabla = 1 - B, \tag{17}$$

B again being the delay operator (9). Thus differencing a time series can be written as

$$\nabla x_t \equiv (1 - B)x_t = x_t - x_{t-1}. \tag{18}$$

Definition 23 (Generalised Autoregressive Operator)**.** For a fixed d the generalised autoregressive operator $\varphi(B)$ is

$$\varphi(B)x_t \equiv \phi(B)(1 - B)^d x_t, \tag{19}$$

$\phi(B)$ being the autoregressive operator (10).

Example 8 (Random Walk). The random walk model assumes that, from one period to the next, the original time series merely takes a random "step" away from its last recorded position, like an inebriated person who steps randomly to the left or right at the same time as he steps forward: the path he traces will be a random walk.
A typical random walk time series shows irregular growth. It is best to analyse the first difference of the series and not the series itself.

Selection of the order of differencing. In practice d is mostly 0 or 1, almost never greater than 2. The optimal order of differencing is often the order of differencing at which the standard deviation of the time series is lowest. If the series has a lot of positive autocorrelation coefficients r_i (from equation (6)) to a high number of lags, then the order of differencing is probably not high enough. Increasing standard deviation and $r_1 \leq -0.5$ are signs of *over-differencing.*

Seasonal differencing. The seasonal difference of a time series is the series of changes from one season to the next. For example, for monthly data, in which there are 12 periods in a season, the seasonal difference of a time series $\{x[t]\}$ at period t is $x_t - x_{t-12}$. Seasonal differencing therefore usually removes the seasonality from a time series, as well as most of the trend.

Definition 24 (Seasonal Differencing Operator)**.** s shall be the length of the season. Then the seasonal differencing operator ∇_s is defined as

$$\nabla_s = 1 - B^s \tag{20}$$

and

$$\nabla_s x_t \equiv (1 - B^s)x_t = x_t - x_{t-s}. \tag{21}$$

Remark. Seasonal differencing and differencing can both be necessary for the same time series to make it stationary. If the series has a strong and consistent seasonal pattern, first the order of seasonal differencing D should be chosen. Usually D is not greater than 1, the sum $d + D$ is not greater than 2 in general.

2.2.4 VSARIMA Models

All of the above mentioned models can be easily combined to a VSARIMA model (vector seasonal autoregressive integrated moving average model), the most general statistical time series model presented in this chapter.

Algorithm 4 (ARIMA(p,d,q) Model)**.** Using the notations from the equations (19) and (15), an ARIMA(p,d,q) model with d non-seasonal differences, p autoregressive and q moving average parameters is given by

$$\varphi(B)x_t = \theta(B)\varepsilon_t. \tag{22}$$

Remark. An ARIMA model with $d = 0$ is also called ARMA model.

Algorithm 5 (SARIMA$(p,d,q) \times (P,D,Q)_s$ Model)**.** If the time series is seasonal and D seasonal differences are added to the ARIMA(p,d,q) model, the model becomes a SARIMA$(p,d,q) \times (P,D,Q)_s$ model:

$$\Phi(B^s)\phi(B)\nabla_s^D\nabla^d x_t = \Theta(B^s)\theta(B)\varepsilon_t, \tag{23}$$

where $\Phi(z)$ and $\Theta(z)$ are polynomials of order P and Q.

Selection of the SARIMA model parameters. First of all the orders of differencing have to be identified according to section 2.2.3 to attain a stationary time series. By looking at the ACF and PACF plots - they are in fact bar charts - of the differenced series, the numbers of AR and/or MA terms that are needed can tentatively be identified.

- If the PACF bar chart of the differenced series displays a sharp cutoff while the ACF decays more slowly and/or r_1 is positive, then adding an AR term to the model should be considered. The lag at which the PACF cuts off is the indicated number of AR terms. An AR term can act like a *partial difference* in the forecasting equation, thus more AR terms are required if the series appears slightly *under-differenced.*

- If the ACF bar chart of the differenced series displays a sharp cutoff while the PACF decays more slowly and/or r_1 is negative, then adding an MA term to the model should be considered. The lag at which the ACF cuts off is the indicated number of MA terms. An MA term can *partially cancel* an order of differencing in the forecasting equation, thus more MA terms are required if the series appears slightly *over-differenced.*

- It is possible for an AR term and an MA term to cancel each other's effects, so if a mixed ARIMA model seems to fit the data, also a model with one fewer AR term and one fewer MA term could fit the data in a very adequate way, particularly if the parameter estimates in the original model require more than 10 iterations to converge.

- If there is a unit root in the AR part of the model, i.e., if $\sum_{i=1}^{p} \phi_i \approx 1$, the number of AR terms should be reduced by one, while the order of differencing should be increased by one.
- If there is a unit root in the MA part of the model, i.e., if $\sum_{i=1}^{q} \theta_i \approx 1$, both the number of MA terms and the order of differencing should be reduced by one.
- If the long-term forecasts appear erratic or unstable, there may be a unit root in the AR or MA coefficients.
- The signature of pure seasonal AR (SAR) or pure seasonal MA (SMA) behavior is very similar to the signature of pure AR or pure MA behaviour, except that the pattern appears across multiples of lag s in the ACF and PACF. For example, a pure SAR(1) process has spikes in the ACF at lags $s, 2s, 3s$ etc., while the PACF cuts off after lag s.
  Conversely, a pure SMA(1) process has spikes in the PACF at lags $s, 2s, 3s$ etc., while the ACF cuts off after lag s.
  If the autocorrelation r_s is positive, adding an SAR term to the model should be considered. If it is negative, adding an SMA term to the model should be considered.
  SAR and SMA terms should not be mixed in the same model, and using more than one of either kind should be avoided. [25]

Choosing the best of several adequate models. If several models seem to be adequate, the best one can be chosen according to so-called *information criteria*. Based on the concept of information entropy, they describe the trade-off between the accuracy of a model and its complexity, thus they provide a mean for model selection. The smaller the below defined AIC and BIC are, the better is the model.

Definition 25 (Akaike's Information Criterion)**.**

$$AIC = -2\ln(L) + 2k, \tag{24a}$$

L being the likelihood function, k the number of parameters of the model.

In the case of SARIMA models, AIC can be calculated as follows:

$$AIC = \ln\left(\frac{1}{n}\sum_{i=1}^{n} r_i^2\right) + 2k, \tag{24b}$$

n being the number of samples and r_i being the residuals from (2).

Definition 26 (Bayesian Information Criterion)**.**

$$BIC = -2\ln(L) + k\ln(n), \tag{25a}$$

In the case of SARIMA models, BIC can be calculated as follows:

$$BIC = \ln\left(\frac{1}{n}\sum_{i=1}^{n} r_i^2\right) + k\ln(n). \tag{25b}$$

The disadvantage of the AIC, which tends to prefer overfitting, is that the penalty term $2k$ is independent of the number of samples n. I.e., for large n the AIC tends to prefer models with many parameters. The BIC tends to prefer underfitting, worsening the score of models with more parameters starting from 8 samples ($\ln(8) > 2$). Apart from these two information criteria there exist some others, like the Hannan-Quinn criterion.

Example 9. The SARIMA$(0,1,1) \times (0,1,1)$ model is basically a seasonal random trend model. It is probably the most commonly used SARIMA model.

Example 10. A SARIMA model for a gas prices time series (which is provided by MATLAB) shall be established.

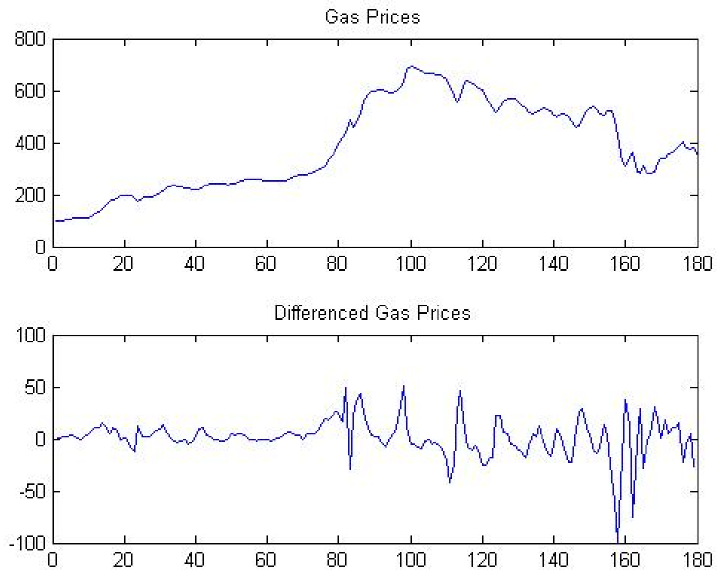


Figure 3: Gas prices time series and its differences.

The time series consists of 180 monthly observations of the gas and the oil prices, starting from July 1973. Only the gas prices shall now be of interest.

A graph of the time series as well as a graph of the differenced time series are shown in figure 10. The time series show an exponential trend in a little bit more than the first half of the series and an increasing variance - the gas time series is heteroscedastic. Both effects can be reduced by logging the time series, which dampens exponential growth patterns and reduces heteroscedasticity. The logged time series and its differences are shown in figure 30 in section B.3 in the appendix. The differenced logged series appears almost stationary.

When used in conjunction with differencing, logging converts absolute differences into relative differences. Thus, a differenced logged time series represents the percentage change from period to period. [25]

To establish a suitable SARIMA model, the ACF and the PACF need to be looked at. The correlation function of the differenced non-logged series are shown in B.3, those of the differenced logged series are shown in figure 10 below.

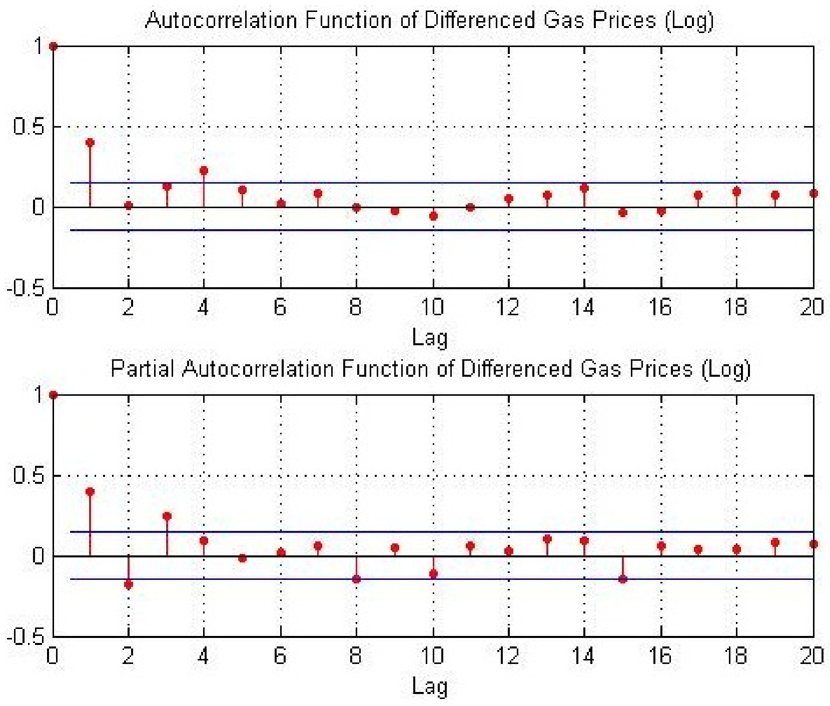


Figure 4: Autocorrelation and partial autocorrelation coefficients of the differenced and logged gas prices time series.

The ACF shows a sharp cutoff after lag 1, while the PACF decays more slowly. This indicates an MA(1)-model. The parameter θ_1 is -0.5273. The AIC of the model is -6.3594, which is very good.

Seasonality is only slightly notable for this series and is not considered in the final (0,1,1)-model.

Remark. Multivariate time series can be processed using multivariate analysis, which can be computationally very expensive for large vectors. The presented models have multidimensional analogons, in the most general case the VSARIMA model. For dimensionally reduced modelling of multivariable time series, the method of principal components analysis can be used.

Remark. Of course there exist further time series models not discussed in this thesis, for example generalised autoregressive conditionally heteroscedastic models (GARCH models).

2.3 Forecasting

Once an adequate time series model has been chosen, it can be used to forecast future values using an adequate forecasting method. The following two quotes should always be kept in mind by someone trying to predict[7] future values of a time series:

"I have seen the future and it is very much like the present, only longer."

Kehlog Albran, The Profit

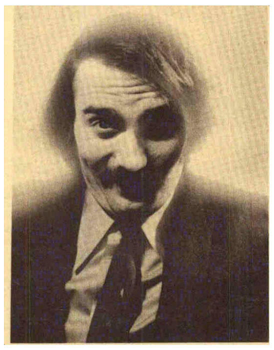


Figure 5: Kehlog Albran

This pseudo-philosophic quote is actually a concise description of statistical forecasting. A time series is described by statistical properties that are constant in trends, seasonal patterns, correlations and autocorrelations etc. It is assumed that those properties will describe the future as well as the present.

[7]Forecasting and *prediction* are two very similar terms, but forecasting is predominantly associated with time series analysis.

Figure 6: Niels Bohr in 1922,
the year he won the Nobel prize.

"Prediction is very difficult, especially if it's about the future."

Niels Bohr, Nobel laureate in Physics

This quote can be seen as a warning of the importance of validating a forecasting model out-of-sample. It is often easy to find a model that fits the past data very well - perhaps too well! -, but quite another matter to find a model that correctly identifies those patterns in the past data that will influence the time series values in the future. Only the future can really tell, if such an *overfitting* has been avoided or if maybe even new, previously unknown patterns arose. One way to minimise the risk of overfitting is cross-validation like it is introduced at the end of section 3.4.2.

2.3.1 Forecasting Methods

Before selecting an adequate forecasting method, it is essential to consider what forecasting accuracy is expected, what computational resources are available, how much data are available, how many items are to be forecast and how far ahead forecasts should be calculated. Apart from this, some forecasting methods simply produce point forecasts. In some cases it is more desirable to produce interval forecasts with an upper and lower limit like the Box-Jenkins method (named after the two statisticians George Box and Gwilym Jenkins) does.

Two of the easiest forecasting methods just use trend analysis or regression analysis. The Box-Jenkins method uses the ARIMA models described in section 2.2. For the calculation of the h-step-prediction, in general first the calculation of the 1- to h-1-step-prediction has to be done. Other advanced statistical forecasting methods include (multiple) exponential smoothing (for example by using the *Holt-Winter algorithm*) or adaptive smoothing - more details can be found in [1].

Interval Forecast. Assuming normally distributed errors and an unbiased forecast - i.e., $\mathbb{E}(e_n^2(h)) = \mathbb{VAR}(e_n(h))$ for the error $e_n(h)$ of the h-step-prediction at the time step n -, the general formula for an $100(1-\alpha)\%$ prediction interval is

$$\hat{x}_n(h) \pm z_{\alpha/2}\sqrt{\mathbb{VAR}(e_n(h))}, \tag{26}$$

$z_{\alpha/2}$ being the $\alpha/2$ percentage point of the standard normal distribution and $\hat{x}$ being the point forecast produced by the ARIMA model.

The key point is the evaluation of $\mathbb{VAR}(e_n(h))$. It may not always be easy for an ARIMA model, but it is possible, in contrary to GARCH or other more complicated statistical models. For example, for an AR(1) model $x_t = \phi_1 x_{t-1} + \varepsilon_t$ the variance of the error of the h-step-forecast is

$$\mathbb{VAR}_{AR(1)}(e_n(h)) = \sigma_\varepsilon^2 \frac{1-\phi_1^{2h}}{1-\phi_1^2}. \tag{27}$$

3 Machine Learning, Data Mining, Soft Computing

3.1 Definitions and Overview

Machine Learning is one part of modern Artificial Intelligence, which has its origins somewhere around 1943. Some exemplary areas of application are detecting spam mails (Naive Bayes, rule mining), giving automatically generated purchase recommendations on amazon.com (clustering), identifying faces on pictures uploaded on facebook (decision trees), identifying sound patterns with Shazam (feature extraction, support vector machines) and so on and so forth. Especially since the beginning of the new millennium a pretty strong overlap between machine learning and statistics can be observed.

Definition 27 (Machine Learning)**.** In 1959, nine years after the proposal of Alan Turing's test for Artificial Intelligence, Arthur Samuel defined machine learning (ML) as a "field of study that gives computers the ability to learn without being explicitly programmed".[8]

Tom Mitchell provided a widely quoted, more formal definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E".[9]

In contrary to mechanics, in many domains the underlying first principles are unknown, or the systems under study are too complex to be mathematically formalized. With the growing use of computers, it is not a great problem to record a huge amount of data being generated by such systems - in general not even knowing which part of the data is important and which part is not. In the absence of obvious glass box models, such data can be used to derive models by estimating useful relationships between some system's variables. If these relationships are interpretable, one might derive a new white or at least grey box model, if not, an input-output dependency might at least give an idea about the changes of the system.

The strong wish to understand large, complex, information-rich data sets is common to almost all fields of business, science and engineering. In the business world, corporate and customer data are becoming recognized as a strategic asset. The ability to extract useful knowledge hidden in these data

[8] `http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr.html`

[9] This definition can be found in his book *Machine Learning*, p. 2. McGraw Hill, 1997, ISBN 0070428077.

and to act on that knowledge is becoming increasingly important. The buzz phrase *personalised advertising* stands for itself. In engineering industry, the detection of system changes and their reasons is of great interest as well as production optimisation - a company that runs servers wants the recorded data to reveal when a hard disk is about to fail, a company that produces any machines wants to know how to reduce the heat loss in the factory depending on what is produced at a certain time etc.

The entire process of applying a computer-based methodology for discovering knowledge from data is called data mining.

Definition 28 (Data Mining)**.** Data mining (DM) is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.[10]

DM is an iterative process, the progress of which is defined by discovery through either automatic or manual methods. It is the cooperative search of humans and computers for new, valuable and non-trivial information in large amounts of data and is therefore most useful in an exploratory analysis scenario where it is not predetermined what might constitute an interesting outcome. Best results with DM are achieved by balancing the knowledge of human experts and the capabilities of computers.

Definition 29 (Knowledge Discovery in Databases)**.** Knowledge discovery in databases (KDD) is simply the extraction of (previously uncharted) knowledge out of the data. Data mining is the analysis step of knowledge discovery in databases.

In practice, the two primary goals of DM tend to be prediction and description, i.e. finding patterns describing the data that can be interpreted by human beings:

- Predictive data mining produces the model of the system described by the available data.
- Descriptive data mining produces new, non-trivial information based on the given data set.

The goals of prediction and description can be achieved by using data-mining techniques - partially already explained and partially yet to be explained in this thesis - for the following primary data-mining tasks:

[10]David Hand, Padhraic Smyth, Heikki Mannila. *Principles of data mining.* MIT Press, 2001.

- classification: discovery of a predictive learning function that classifies a data item into one of several predefined classes.
- regression: discovery of a predictive learning function, which maps a data item to a real-value prediction variable
- clustering: a common descriptive task in which one seeks to identify a finite set of categories or clusters to describe the data.
- summarisation: an additional descriptive task that involves methods for finding a compact description for a set (or subset) of data.
- dependency modelling: finding a local model that describes significant dependencies between variables or between the values of a feature in a data set or in a part of a data set.
- change and deviation detection: discovering the most significant changes in the data set.

In essence, data mining is like solving a puzzle. The individual pieces of the puzzle are not complex structures in and of themselves. Taken as a collective whole, however, they can constitute very elaborate systems. [3]

As the two terms machine learning and data mining are commonly confused, because they often employ similar methods and overlap significantly, the following can be seen as a rough distinction: Machine learning focuses on prediction, based on known properties learned from some training data, while data mining focuses on the discovery of (previously) unknown properties of the data. In ML, performance is usually evaluated with respect to the ability to reproduce known knowledge, while in KDD the key task is the discovery of previously unknown knowledge. Evaluated with respect to known knowledge, an uninformed (unsupervised) method will in general easily be outperformed by supervised methods, but in a typical KDD task, supervised methods cannot be used due to the simple fact that there is no training data available. An exact distinction between unsupervised and supervised methods as well as a presentation of other approaches to learning will be given in the following section 3.2.

Many of the data mining and machine learning methods are so-called soft computing methods. A definition of the term soft computing is given below:

Definition 30 (Soft Computing)**.** The aggregation of several intelligent computing techniques like artificial neural networks, fuzzy logic, probabilistic reasoning, chaos theory, genetic algorithms, swarm intelligence and parts of the learning theory, that can deal with the indeterminacy and imprecision of the real world, is called soft computing (SC). [4]

Figure 7: Lotfi Askar Zadeh (born February 4, 1921), proposer of fuzzy logic and creator of the term *Soft Computing.*

"As the complexity of a system increases, our ability to make precise and yet significant statements about its behaviour diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics."

Lotfi Zadeh, 1973

Soft computing methods deal with imprecision, uncertainty, partial truth and approximation to achieve practicability, robustness and low solution cost. Often resembling biological processes, these methods are intended to complement each other to achieve more efficient hybrid modelling.

"So far as laws of mathematics refer to reality, they are not certain, and so far they are certain they do not refer to reality."

Albert Einstein, 1921

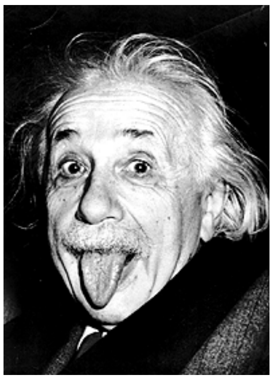


Figure 8: Albert Einstein, 14.03.1879 - 18.04.1955

3.2 Algorithm Types - Different Approaches to Learning

Learning and intelligence are two very important key words of DM, ML and SC. After a definition of the term *learning* several approaches to learning are presented. Which approach has to be chosen for a certain model heavily depends on the available data (e.g., if it is pre-labelled or not) and whether the nature of the main task of the model is descriptive (favours strict supervised learning) or predictive.

Definition 31 (Learning)**.** Learning is constructing or modifying representations of what is being experienced.[11]

Supervised Learning. Supervised learning generates a function that maps inputs to desired outputs, which can originate from measurements or experts. These desired outputs are also called *labels*, because they are often provided by human experts labelling the training examples, or *targets*, because the goal of this kind of learning is that the function maps the inputs as close as possible to these desired outputs. Learning in a supervised sense occurs by adapting the function according to the difference of the function outputs and the labels.

Of course supervised learning is only possible, if the target values are available! Unfortunately this is not always the case.

Unsupervised Learning. Unsupervised learning refers to the problem of finding hidden structures in unlabelled data. As there do not exist any labels, there is no possibility to calculate any error to evaluate a potential solution. There is no reward signal either. This clearly distinguishes unsupervised learning from supervised learning and reinforcement learning. The key point of unsupervised learning is very often an estimation of density.

Unsupervised learning algorithms include clustering (see section 3.5), feature extraction techniques for dimensionality reduction (PCA, singular value decomposition,. . .) and neural network models (self-organising map, adaptive resonance theory, see section 3.4.3).

Semi-supervised Learning. Semi-supervised learning combines both labelled and unlabelled examples to generate an appropriate function or classifier.

Deep Learning. The deep learning concept[12] is based on learning several levels of representations, corresponding to a hierarchy of features or factors, whose appropriate number of levels and whose structure is something that a deep learning algorithm is also expected to discover from examples. While

[11]Ryszard Michalski, Gheorghe Tecuci (editors). Volume IV of *Machine Learning: A Multi-Strategy Approach.* Morgan Kaufmann, San Francisco.

[12]A great introduction to this concept is [27].

the task is usually semi-supervised, the algorithms are often framed as unsupervised.

Reinforcement Learning. Inspired by behaviourist psychology, reinforcement learning is based on feedback in the form of rewards that guides the learning algorithm. This feedback can be as simple as one bit, telling the algorithm if the result was good or bad. In machine learning, the environment of reinforcement learning is typically formulated as a *Markov decision process.*

Although not an approach to learning itself, the term *linear separability* is introduced in the following definition because some learning algorithms need to be adapted to solve non-linearly separable problems - an attribute most real-life problems certainly have.

Definition 32 (Linearly Separable)**.** Two sets of points X_0 and X_1 in an m-dimensional space are linearly separable, if they can be separated by an $m-1$-dimensional hyperplane, i.e. if there exist $m+1$ real numbers $\omega_1, \ldots, \omega_m, k$, such that every point $x \in X_0$ satisfies $\sum_{i=1}^{m} \omega_i x_i \geq k$ and every point $x \in X_1$ satisfies $\sum_{i=1}^{m} w_i x_i < k$.

Usually the number of hyperplanes that separate the two linear separable classes is infinite, the best probably being the one whose orthogonal distance to the closest data point(s) is maximal. While an ANN generally only tries to find a more or less arbitrary separating hyperplane, a support vector machine intends to find the described best (see section 3.6 for further details).

Example 11. The logical operator AND is linearly separable.

Example 12. The logical operator XOR is the easiest example of non-linearly separable functions.

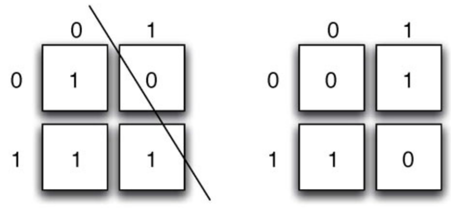


Figure 9: The linearly separable AND operator on the left and the non-linearly separable XOR operator on the right.

3.3 Introduction to Fuzzy Logic

Although the term *fuzzy logic* was introduced only in 1965 with the proposal of fuzzy set theory by Lotfi A. Zadeh, it has been studied since the 1920s as infinite-valued logics notably by Łukasiewicz and Tarski. Fuzzy logic has been applied to many fields, from control theory to artificial intelligence, experiencing its first boom in the 1980s in Japan, where the subway of Sendai could be run fully automatically due to fuzzy logic. About the same time lots of home appliances started to be operated using fuzzy rules. This nowadays makes fuzzy logic part of our every-day life.

Fuzzy logic is an extension of the classic binary logic, the first roots of which can be found in Aristotle's *Organon*, where his syllogism is introduced. More formally, the Boolean algebra was introduced by George Boole in his book *An Investigation of the Laws of Thought* in 1854. Boolean logic is a subarea of algebra in which the values of the variables are the truth values true and false, usually 1 and 0, while fuzzy logic allows the modeller to also handle the concept of partial truth. If linguistic variables (e.g. full) are used, these degrees of truth (e.g. rather full) may be dealt with specific functions.

Definition 33 (Fuzzy Logic)**.** Fuzzy Logic is a branch of logic designed to allow imprecisions in reasoning, knowledge and truth.

As fuzzy logic does not play an important role in this thesis, only some basic terms and concepts are introduced in the following subsections.

3.3.1 Fuzzy Sets and Membership Functions

A membership function of a fuzzy set is a generalization of the indicator function in classical sets. It represents the degree of truth.

Definition 34 (Membership Function)**.** A membership function for a fuzzy set A on the universe of discourse X is defined as $\mu_A : X \to [0,1]$, where each element of X is mapped to a value between 0 and 1, the so-called *membership value* or *degree of membership*, which quantifies the grade of membership of the element in X to the fuzzy set A.[13]

Membership functions allow to graphically represent a fuzzy set. While the x-axis represents the universe of discourse, the y-axis represents the degrees of membership in the unit interval.

Simple functions are used to build membership functions, which makes sense because of the nature of fuzziness itself:

[13]Definition from `http://www.dma.fi.upm.es/java/fuzzy/fuzzyinf/funpert_en.htm`.

- triangular functions,
- trapezoidal functions,
- R-/L-functions,
- Gaussian functions.

The membership functions of non-fuzzy sets - so-called *crisp* sets) - look like rectangles, as they are simply indicator functions.

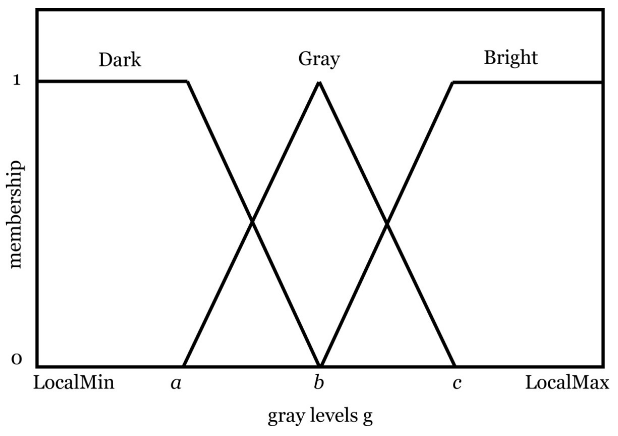


Figure 10: Three fuzzy membership functions of the linguistic variables *dark*, *gray* and *bright*. The membership function of *dark* is an R-function, the one of *bright* an L-function and the one of *gray* is in this case a triangular one.

Example 13. If a modeller wants to detect outliers, it is possible to assign each datum a value of *outlierness*, the potential to be an outlier, according to the values calculated using any method for outlier detection. If these values of outlierness are mapped into the unit interval, outlierness can be looked at as a fuzzy variable.

Fuzzy Logic and Probability Theory. Fuzzy logic and probability theory are mathematically similar, both use truth values ranging between 0 and 1. Conceptually they are distinct: Fuzzy logic corresponds to degrees of truth, while probability theory corresponds to terms like probability or likelihood. Concerning sets, the two concepts differ in the following way: Fuzzy set theory uses the concept of membership functions, the modeller wants to know how much a variable is in a set. On the contrary, a modeller that uses the probability theory asks how probable does he think that a variable is in a set. It has to be stated that both methods are subjective. Representatives of both theories have stated that the other theory is only a sub-concept of their own theory and therefore unnecessary.

One of many examples that at least shows some advantages of the fuzzy view to a modeller in my personal opinion, is the following: Let a person be 180cm tall. Then two concepts may be considered: The person is tall or the person is small. The meaning of each of them can be represented by a certain fuzzy set. One modeller might define the person as 0.5 tall and as 0.5 small, another designer as 0.8 tall and as 0.1 small. Of course this could also depend on the fact, if the person is male or female and if this fact is even known or not. The concept of size in this model is highly subjective and depends on the designer. The probabilistic view in such an example seems somehow strange: Should the person be considered as tall with a probability of 80%? This seems odd, because the size of the person can certainly be looked at as a constant.

It is essential to realize that fuzzy logic uses truth degrees as a mathematical model of the vagueness phenomenon.

3.3.2 Fuzzy Set Operators

Any kind of logic works on sets and uses operators like intersection or union of sets. For fuzzy sets the classic intersection and union operators need to be generalised, leading to fuzzy t-norms and fuzzy t-conorms.

Definition 35 (Fuzzy T-Norm)**.** A fuzzy t-norm is a function $\top : [0,1] \times [0,1] \to [0,1]$ which satisfies the following properties:

1. $\top(a,b) = \top(b,a)$ (Commutativity)
2. $\top(a,b) \leq \top(c,d)$, if $a \leq c$ and $b \leq d$ (Monotonicity)
3. $\top(a,\top(b,c)) = \top(\top(a,b),c)$ (Associativity)
4. $\top(a,1) = a$ (1 is an identity element)

Example 14. Examples for fuzzy t-norms fulfilling the above stated requirements are:

- $\top(a,b) = \min\{a,b\}$, the minimum or Gödel t-norm, is the pointwise largest t-norm.
- $\top(a,b) = ab$
- $\top(a,b) = \max\{0, a+b-1\}$ is called Łukasiewicz t-norm.

- the drastic t-norm is the pointwise smallest: $\top(a,b) = \begin{cases} b & \text{if } a = 1, \\ a & \text{if } b = 1, \\ 0 & \text{otherwise.} \end{cases}$

- the Hamacher product $\top(a,b) = \begin{cases} 0 & \text{if } a = b = 0, \\ \frac{ab}{a+b-ab} & \text{otherwise.} \end{cases}$

Definition 36 (Fuzzy T-Conorm)**.** T-conorms are dual to t-norms under the order-reversing operation which assigns $1-x$ to x on $[0,1]$. Thus a t-conorm $\perp : [0,1] \times [0,1] \to [0,1]$ is defined by:

1. $\perp(a,b) = \perp(b,a)$ (Commutativity)
2. $\perp(a,b) \leq \perp(c,d)$, if $a \leq c$ and $b \leq d$ (Monotonicity)
3. $\perp(a, \perp(b,c)) = \perp(\perp(a,b),c)$ (Associativity)
4. $\perp(a,0) = a$ (0 is identity element)

Example 15. Examples for fuzzy t-conorms fulfilling the above stated requirements are:

- $\perp(a,b) = \max\{a,b\}$, the maximum t-conorm, is the pointwise smallest t-conorm.
- $\perp(a,b) = a + b - ab$
- $\perp(a,b) = \min\{a+b, 1\}$ is called Łukasiewicz t-conorm.
- the drastic t-conorm is the pointwise largest: $\perp(a,b) = \begin{cases} b & \text{if } a = 0, \\ a & \text{if } b = 0, \\ 1 & \text{otherwise.} \end{cases}$
- the Einstein sum $\perp(a,b) = \frac{a+b}{1+ab}$.

As fuzzy t-norms and t-conorms are generalisations of intersection and union of sets, it suggests itself that also a generalisation of the De Morgan's laws can be proven:

Theorem 3 (Generalised De Morgan's Laws)**.**

$$\perp(a,b) = 1 - \top(1-a, 1-b). \tag{28}$$

3.3.3 Fuzzy Control System

For a fuzzy control system first of all the input variables need to be mapped to fuzzy variables by the membership functions. The conversion of crisp values to fuzzy values is called *fuzzification.* The micro-controller makes decisions according to a set of fuzzy rules applied to these variables defined on fuzzy sets.

Definition 37 (Fuzzy Rule)**.** A fuzzy rule is defined as a conditional statement of the form:

$$\begin{gathered}\text{IF } x \text{ is } A\\ \text{THEN } y \text{ is } B.\end{gathered}$$

x and y shall be linguistic variables, A and B linguistic values determined by fuzzy sets on the universe of discourse X and Y. The IF part is called *antecedent*, the THEN part *consequent.*

Example 16. If the fuzzy variable *height* increases towards *tall*, the fuzzy variable *weight* increases towards *heavy.*

If a rule consists of several antecedents, they can be combined using fuzzy operators as AND, OR or NOT (AND uses simply a t-norm, OR a t-conorm and NOT the complementary function.). The membership functions used in these fuzzy rules can also be modified by adjectives: *very* squares the membership function, *extremely* cubes it, *rather* takes the square root.

The result of the (usually dozens of) rules is determined using again the membership functions involved. Finally the fuzzy value of the output variable is transferred back to a crisp value - this is the so-called *defuzzification.* The crisp output value can now be used to adjust the system, for example the heater that shall be controlled.

There are several defuzzification methods. One of the most common techniques is the method of *center of gravity*, which simply calculates the centroid of the output membership function as the crisp value that is returned.

3.4 Artificial Neural Networks

3.4.1 Archetype Biology

There are a lot of problems that cannot be represented by a simple algorithm, but are solved by a human being who learned how to deal with similar kinds of problems. This exact ability to learn in a very adaptive way is something very important which computers and algorithms obviously have a lack of.

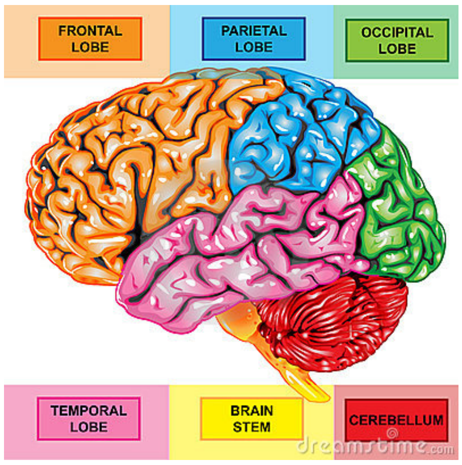


Figure 11: A human brain and its most important partitions.

In theory, a computer could outperform a human pertaining to efficiency in solving complex problems due to the short response time of its transistors (see table 2), but of course this is only true for complex numerical calculations. Some main advantages of the human brain compared to a computer are:

- the ability that many areas of the brain work in a parallel way at the same time, while most of the computer transistors only passively store data
- the ability to readjust its structure
- the ability to generalise
- the ability to tolerate errors.

The basic idea of an artificial neural network (ANN) is to harness these abilities for computers. [5]

Because of its above introduced modelling approach, ANNs are a typical soft computing method according to definition 30.

The basic element of a neural network is of course a neuron. All neurons are electrically excitable cells, maintaining voltage gradients across their membranes by means of metabolically driven and sometimes voltage-dependent ion pumps that move ions such as sodium, potassium, chloride and calcium from the inside to the outside of the cell or vice-versa.

	Brain	Computer
Number of Processing Units	$\approx 10^{11}$	$\approx 10^{9}$
Type of Processing Units	Neurons	Transistors
Form of Calculation	Massively Parallel	Generally Serial
Data Storage	Associative	Address-based
Response Time	$\approx 10^{-3}$s	$\approx 10^{-9}$s
Processing Speed	Very Variable	Fixed
Potential Processing Speed	$\approx 10^{13}$ FLOPS [14]	$\approx 10^{18}$ FLOPS
Real Processing Speed	$\approx 10^{12}$ FLOPS	$\approx 10^{10}$ FLOPS
Resilience	Very High	Almost None
Power Consumption per Day	20W	300W [15]

Table 2: A comparison of properties of a human brain and an average personal computer. [5]

If the voltage changes by a large enough amount, an all-or-none electrochemical impulse called an action potential is generated and propagated to other cells, where it again causes a voltage change. Thus a certain neuron works like a switch which is turned on if the sum of voltage changes arriving to this neuron is above some threshold.

Human beings possess many different kinds of neurons, all of them specialised in their own way. Basically, a human being has phasic and tonic receptors (i.e., special neurons), a distinction made due to the rate of adaptation. A phasic receptor adapts rapidly to a stimulus, fires its action potential and then stops at least for the absolute refractory period (approximately 2ms), which is followed by the relative refractory period (approximately 1.5ms), in which stimuli need to be stronger to produce an action potential. This kind of receptor does not provide information about the length of the stimulus. A tonic receptor adapts slowly to a stimulus, it is able to produce action potentials over the duration of a stimulus. Some tonic

[14] floating point operations per second

[15] A supercomputer needs up to 10 million watts a day!

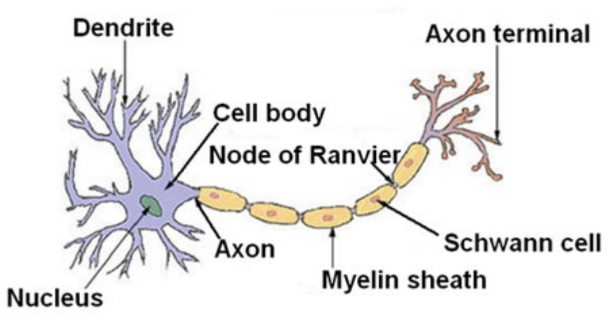


Figure 12: Structure of a typical human neuron.

receptors are even permanently active. Examples for tonic receptors are pain receptors and receptors for the perception of vibrations.

Also the propagation of an action potential between two neurons occurs in different specific ways: Human beings have some hard-coded electrical synapses (e.g. for the initiation of the flight reflex). Chemical synapses work with neurotransmitters like acetylcholine, dopamine or serotonin, which has - though slower and more complex - many advantages: the propagation of the electric impulse works one-way and there is a great adjustability due to different amounts of neurotransmitters with various restraining or stimulating effects. For example, the intensity of signals heard is propagated logarithmically by nerval stimuli. [5]

Keeping in mind that there are many different specialised neurons in a human being, a typical structure of a neuron is shown in figure 12. At the majority of synapses, signals are sent from the axon of one neuron to a dendrite of another, thus the dendritic tree of a neuron somehow collects the arriving stimuli. This is also the basic idea of the connection of artificial neurons.

3.4.2 Basic Considerations about Modelling Neural Networks

Without going into too much detail at this point, a definition of an artificial neural network by Teuvo Kohonen is given below:

Definition 38 (Artificial Neural Network)**.** Artificial neural networks are massively parallelly distributed networks consisting of simple, usually adaptive elements that are arranged and organised hierarchically. They are built on the analogy to the human information processing system and should interact with the world in the same way as biological neural networks do.

Of course it has to be stated that ANNs are a radical simplification of the highly specialised human neural network.

The term *time* in the context of ANNs usually refers to the number of iterations that were used in the implementation of the ANN. This has only practical reasons and has neither an analogon in real life nor anything to do with any time discretisation of the whole model.

The elements of an ANN are also called neurons - small processing units. The output of a neuron is a (usually non-linear) function of the (possibly weighted) sum of inputs minus some threshold θ. Thus the whole ANN is in general highly non-linear. While the input is usually mathematically spoken a vector, the output is a scalar. The neurons are arranged in layers and it has to be defined which neurons are connected with each other. The neurons, the connections between them and their weights fully define the ANN. Usually non-existing connections are simply defined by the weight 0, which makes it possible to assume that every ANN is fully connected. The weights symbolise the biological synapses, their entirety represents the known information of the network. Therefore it is necessary that the weights are variable.

The actual weight between the neuron N_i and the neuron N_j shall be ω_{ij}. All weights are stored in a quadratic matrix W - this is called the *Hinton representation*. o_i being the output of the Neuron N_i, the input net_j of a neuron N_j is usually calculated as the weighted sum

$$net_j = \sum_{i \in I} \omega_{ij} g_i(o_i), \tag{29}$$

the functions g_i often being the identity. The propagation of the data is modelled by the use of the activation function (also *transfer function*) in a neuron.

Definition 39 (Activation Function)**.** Let $o_k(t)$ be the activation[16] of the neuron N_k at iteration t, $net_k(t)$ the neuron input at t and θ_k the threshold of the activation function of N_k according to the following definition 40. The activation function f_{act} depends on the actual input, the last output as well as the threshold and returns the new output:

$$o_k(t) = f_{act}(net_k(t), o_k(t-1), \theta_k). \tag{30}$$

Definition 40 (Threshold θ_k)**.** Let N_k be a neuron. Its threshold θ_k is the value with the largest slope of the activation function of N_k.

Example 17. Widely used activation functions are the Heaviside function (not continuous), linear functions, the hyperbolic tangent (range $(-1, 1)$,

[16]In this thesis the terms activation and output shall be equivalent.

differentiable), the sigmoid function (see the following definition 41) and radial basis functions (see the following definition 42).

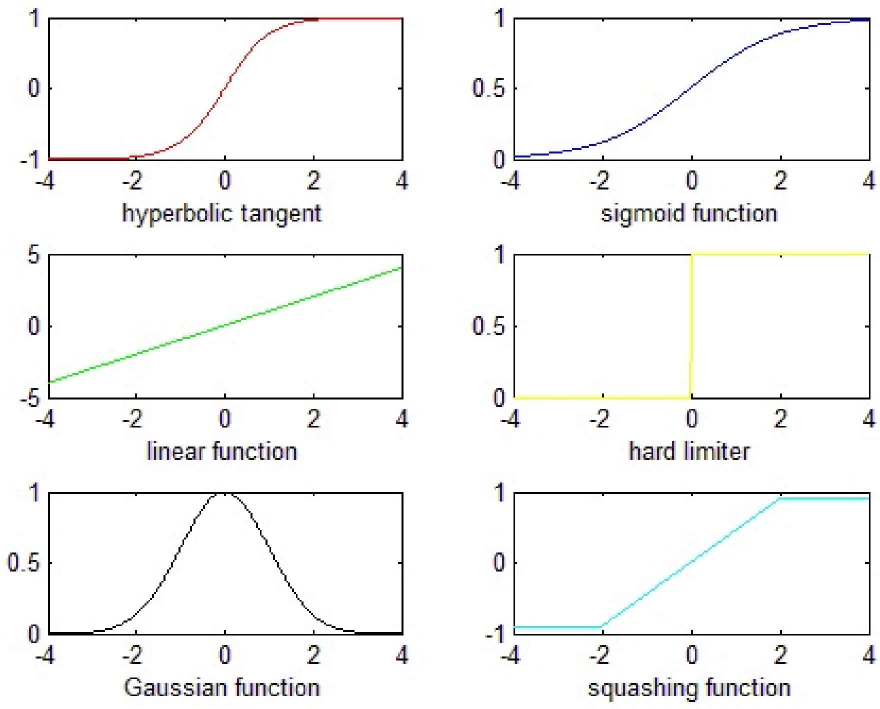


Figure 13: Graphs of the most commonly used activation functions.

Remark. Due to numerical reasons Davide Anguita et al.[17] suggested a fast approximation hyperbolic tangent, consisting of two parabolic arcs and two half-lines. Because evaluations of the exponential function are avoided, this function can improve the calculation time by the astonishing factor 200.

Definition 41 (Sigmoid Function)**.**

$$f(x) = \frac{1}{1+e^{-x}} \tag{31}$$

is called sigmoid or logistic function. Its range is the interval $(0, 1)$.

[17]D. Anguita, G. Parodi and R. Zunino. *Speed improvement of the back-propagation on current-generation workstations*, Volume 1 of the *World Congress on Neural Networks*, July 11-15, 1993, Oregon Convention Center, Portland, Oregon. Lawrence Erlbaum, 1993.

Definition 42 (Radial Basis Function)**.** A real-valued function ϕ is called radial basis function (RBF), if its values only depend on the distance from some center c, which is very often the origin. Thus the condition for being a RBF is

$$\phi(x, c) = \phi(\|x - c\|) \quad \forall x \text{ with a certain center } c \text{ or} \tag{32a}$$

$$\phi(x) = \phi(\|x\|) \quad \forall x. \tag{32b}$$

The threshold θ has been introduced as a parameter of the activation function f_{act}. During the training of the ANN (see section 3.4.4) it is complicated to access and change f_{act}, if θ should be changed. In this case it is easier to represent θ in a totally different way, indeed as the weight of an always firing *on-neuron.*

Definition 43 (Bias Neuron)**.** The output of a bias neuron or on-neuron is always 1. It is used to represent thresholds as weights, the weight of the connection between the bias neuron and a neuron N_j being the negative threshold $-\theta_j$.

The new threshold of N_j is 0, f_{act} only depends on the actual input and the last output. This makes the implementation a lot easier, although the schematic representation becomes more complicated.

The number of neuron layers of a typical ANN varies between 1 and 4. An ANN consists of an input and an output layer (which are the same in certain ANNs) and possibly some hidden layers between those two. The numbers of neurons per layer strongly depends on the complexity of the problem the ANN is applied to. Often not the neuron layers of an ANN, but the number of variable weight layers are counted. That means that an ANN with two neuron layers and one weight layer between them is often called a single-layer ANN.

Approximation Capacity of ANNs. There is a universal approximation theorem for ANNs that states that the standard multilayer feed-forward network with one hidden layer, which contains a finite number of hidden neurons, is a universal approximator among continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function. The following theorem was proved by George Cybenko in 1989 for a sigmoid activation function, and is therefore named after him.

Kurt Hornik from the Technical University Vienna showed in 1991 in [26] that it is not the specific choice of the activation function, but rather the multilayer feed-forward architecture itself which gives neural networks the potential of being universal approximators.

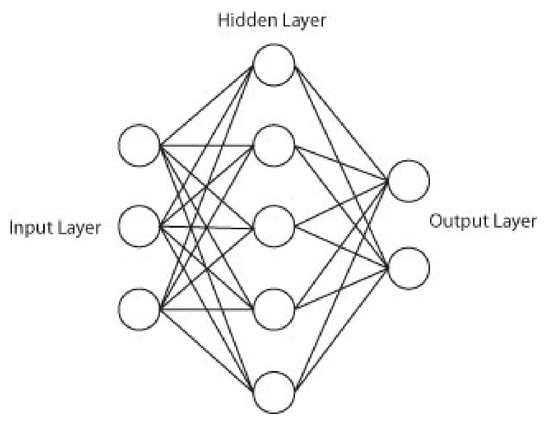


Figure 14: A possible structure of an ANN.

Theorem 4 (Cybenko)**.** *Let φ be a non-constant, bounded and monotonically-increasing continuous function. Let I_m denote the m-dimensional unit-hypercube $[0,1]^m$. $C(I_m)$ denotes the space of continuous functions on I_m. For any given function $f \in C(I_m)$ and $\varepsilon > 0$ there exist an integer N and sets of real constants $\alpha_i, \beta_i \in \mathbb{R}$ and vectors $\omega_i \in \mathbb{R}^n$, where $i = 1, \dots, N$, such that $F(x)$ can be defined as follows:*

$$F(x) = \sum_{i=1}^{N} \alpha_i \varphi \left(\omega_i^T x + \beta_i \right) \tag{33}$$

is an approximate realisation of the function f; that is in mathematical terms,

$$|F(x) - f(x)| < \varepsilon \qquad \forall x \in I_m. \tag{34}$$

Remark. The sums of the type of the right-hand side of (33) are dense in the space of continuous functions on the unit cube, if φ complies the above mentioned requirements. These sums generalise approximations by finite Fourier series. The proof uses Borel measures, the Hahn-Banach theorem, Riesz' representation theorem, the Lebesgue-bounded convergence theorem and that φ is discriminatory.[18]

Remark. The Cybenko theorem is formulated for only one (linear) output neuron simply due to notational convenience.

[18]A full proof can be found at:
`http://cs.haifa.ac.il/~hhazan01/AdvanceSeminaronNeuro-Computation/2010/nn1.pdf`

Applying the above formulated theorem 4 to an ANN with one hidden layer, N is the number of the neurons in the hidden layer, f the continuous function to be approximated (i.e. in the context of this thesis, representing the solution of the problem or the time series to be modelled), ω_i are the weights of the input layer, β_i the thresholds of the neurons in the hidden layer or the bias neuron inputs, α_i the weights in the hidden layer and φ is the activation function (The sigmoid function, the hyperbolic tangent, the fast approximation hyperbolic tangent and the squashing function are certainly non-constant, bounded and monotonically-increasing).

As a consequence of Cybenko's theorem, bad approximations of ANNs for continuous multivariate functions are caused by bad weights, a bad number of neurons, a bad learning rate or something similar. The fact that $f \in C(I_m)$ suggests a normalisation of the available data.

Network Topology. Until now, there does not exist a clearly formulated method to design a good topology of an ANN. It is thought that a huge performance improvement can be achieved by the ability to choose a good or maybe even optimal topology of an ANN. Different problems require different network topologies. The modelling approach makes it clear, how many input and output neurons are needed, but does not necessarily tell anything about the hidden layers or the connections between the neurons. Some rules of thumb for starting points for the number of hidden neurons n_h do exist: For example, for an ANN with an input layer with n_i neurons, one hidden and an output layer with n_o neurons, one could start with $n_h = \sqrt{n_i n_o}$ hidden neurons or something similar between n_i and n_o. A more general rule of thumb is the geometric pyramid rule:

$$n_h = \alpha\sqrt{n_i n_o}, \quad \alpha \in [0.5, 2]. \tag{35}$$

The use of too many hidden neurons not only increases the calculation time, but also represents the well-known problem of over-parameterised models with more degrees of freedom than observations in the training set [15]. It is recommendable to follow a bottom-up principle: First an ANN with few neurons is tested, and the number of neurons should be augmented as long as the performance of the network increases significantly. [5] The performance of an ANN can only be measured after an adequate training.

Initialisation. Before the training, the weights of an ANN need to be initialised. Usually this is done by *symmetry breaking*, by choosing random small initial weights, which are not too close to zero, i.e. from a set of the type $[-0.5, 0.5]\backslash(-\varepsilon, \varepsilon)$ for a small $\varepsilon > 0$. Symmetry breaking has the positive effect that the weighted input to a certain layer is probably close to 0, where the threshold of most activation functions is. This allows great learning impulses at the beginning of the training. [5]

Training. In this paragraph only a supervised learning approach is considered. Right after the initialisation, the ANN is not able to solve a specific problem. First it needs to be trained, wherefore a training set is presented numerous times to the ANN during the learning process. The training set does not only consist of observed input vectors, but also of the corresponding (except for possible measurement errors) correct output vectors, the *target vectors.* The ANN can be trained either in an *off-line* way (also called *batch*-training, i.e. the whole training set is presented to the net before changing the weights) or in an *on-line* way (the weights are adapted after every single input vector/target vector pair). During the learning process, the weights are changed in a way to minimise some norm of the error vectors, which are simply the differences between the target and the output vectors. The one-time presentation of the training set to the ANN and the related adaptation of the weights is called *training epoch.*

Training Set. The order of the training set is relevant to the performance of the ANN. A completely random presentation to the net does not guarantee an uniformly distributed learning success, presenting the training set always in the same order to the ANN constrains its ability to generalise, especially in the case of recurrent ANNs. *Shuffling* the training set (i.e., creating a random permutation of the training set) in every epoch avoids both problems, but by doing this the calculations for the training become more complex. The best way to achieve a good trade-off between high accuracy and a strong generalisation ability of the ANN is to use cross-validation by dividing the training set into three parts:

- a training set, which is really used to train the ANN (usually about 70% of the whole training data)
- a validation set, which is used to measure the learning progress (usually about 15% of the whole training set); furthermore the training and the validation set are used to decide when to stop the training
- a test set, which gives an idea about the quality of the ANN (usually about 15% of the whole training set).

Convergence. Nothing can be said in general about the convergence of an ANN. There might exist many local minima of the error function and the optimization method used might not converge, when starting far away from a local minimum. Apart from that also the topology of the ANN might be so bad that the found global minimum of the error function is not acceptable for practical reasons.

3.4.3 Different Types of Networks

A variety of different types of ANNs exists, which differ in their topology, in the existence of loops or in their modelling approaches. Some of the most important types are going to be presented in the following paragraphs.

Feed-Forward Neural Networks. In this most simple type of ANNs the information moves in only one direction, there are neither loops nor cycles. Examples of feed-forward networks are single-layer and multi-layer *perceptrons.* Applied to a classification problem, single-layer perceptrons can separate classes by hyperplanes, two-layer perceptrons by convex polygons and three-layer perceptrons can classify any given sets correctly. These networks are usually trained by the backpropagation algorithm.

Radial Basis Function Networks. RBF networks consist of exactly three neuron layers and have a feed-forward structure without bias neurons. The input layer is directly connected with the hidden layer whose activation function is an RBF function with centres c_i, only the connections between the hidden layer and the output layer are weighted (These linear weights influence the amplitude of the Gaussian bells). RBF networks are universal function approximators. The training procedure is different to the one of feed-forward networks, it is usually a combination of an algorithm based on a descending gradient and solving equation systems using the Moore-Penrose pseudo-inverse matrix. The most important issue of RBF networks is the selection of the centres c_i and the related spread parameters, which are either fixed (due to some available knowledge) or calculated by k-means clustering (see section 3.5), a self-organising map (see below in this section) or something else. RBF networks have advantages in comparison to multi-layer perceptrons, if the output dimension is high and if interpolation is more important than extrapolation. They have disadvantages, if the input dimension is high. Apart from that, RBF networks have the possibility to use the output 0 to create an answer to a classification problem that - transferred to words - is *"I don't know"*.

Replicator Neural Networks. A replicator neural network is a multilayer perceptron neural networks with three hidden layers. As the network tries to replicate the data (i.e., the input is the desired output), the number of output neurons equals the number of input neurons. Usually the first and third hidden layer use the hyperbolic tangent and the second hidden layer uses a staircase function as activation function. The replicator neural network forms an implicit, compressed model of the data during the training. A measure of outlierness of each datum is then developed as the reconstruction error of individual data points. The replicator neural network approach has its linear analogon in principal components analysis (PCA). [20]

Hopfield Networks. Hopfield networks are inspired by the behaviour of particles in a magnetic field. They consist of K fully connected neurons, which are not ordered in any layers. The activation function used is $-1+2H$, H being the Heaviside function. Thus the network status is a binary string $z \in \{-1, 1\}^{|K|}$. Nowadays they are not of huge interest anymore.

Recurrent Neural Networks. Contrary to the above described networks, recurrent neural networks (RNN) are models with bidirectional data flow. Two of the simplest models of this type are the *Elman* and the *Jordan network*, which both have three layers like the classic three-layer perceptron, but with the addition of a layer with *context neurons* in the input layer. A context neuron saves the previous output of the neuron it belongs to, thus the only incoming connection from a hidden neuron in the case of an Elman net or from an output neuron in the case of a Jordan net has the weight 1. Due to their internal memory RNNs deliver mentionable results in tasks like speech processing or handwriting recognition and are well applicable for dynamic systems. The training of these nets can be done by backpropagation through time: The network gets unfolded in time, every context neuron represents a neural network of the same architecture and the obtained huge network can be trained by the backpropagation algorithm. Other algorithms used to learn the RNN are real-time recurrent learning or evolutionary algorithms. A major problem with gradient descent methods for standard RNN architectures is that error gradients vanish exponentially with the size of the time lag, a problem that is overcome by long short-term memory networks.

Long Short-Term Memory Networks. Long short-term memory networks (LSTM networks) include special LSTM blocks apart from regular network units. Such a block basically consists of a normal input neuron and three gate control neurons, which control the *input gate*, the *forget gate* and the *output gate* respectively. Within the block the calculated value is fed back in without being altered by weights or an activation function and thus is remembered as long as the forget gate allows. Because of this property an LSTM network is well-suited to process time series when there are very long time lags of unknown size between important events. The architecture of these high-end ANNs was proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber.

Deep Networks. Following the deep learning idea (see section 3.2), these networks use a very high number of layers.

Neuro-Fuzzy Networks. As soft computing methods can be combined efficiently in a hybrid way, there are a lot of neuro-fuzzy systems which try to use the fact that ANNs are universal approximators and the fact that fuzzy systems are well interpretable, especially if the system includes fuzzy if-then-rules. The term neuro-fuzzy networks is a pretty general one and also involves deriving fuzzy rules or realising fuzzy membership functions from ANNs, fuzzy logic based tuning of ANN training parameters and ANNs whose inputs, outputs and/or weights are fuzzy values.

Unsupervised Neural Networks. The human brain does neither map input to output vectors nor has training examples with target vectors. Its reaction to an input is a change of its status, and this can be considered as its output. According to this unsupervised principle, Teuvo Kohonen proposed *self-organising maps* (SOM). The key question in this model is not what the neurons calculate, but which neurons are active. SOMs use a neighbourhood function to preserve the topological properties of the input space, which makes them useful to visualise two-dimensional views of high-dimensional data. The neurons of an SOM are arranged in a lattice (a square grid with the Von Neumann or Moore neighbourhood, a hexagonal grid), their weights initialisation can be done by sampling evenly from the subspace spanned by the two largest principal component eigenvectors. The training uses competitive learning, the best matching unit (i.e. the neuron whose weight is closest to the input) and its neighbours are adjusted to the input. Topologic functions used during the training are for example Gaussian bell functions, cone functions or the Mexican hat function. SOMs find similarities and can therefore be used to suggest similar songs, scientific papers or articles to a costumer.
The *adaptive resonance theory* (ART) model is an ANN that is not only used for binary classification but also should be able to encounter new classes. An

ART network consists of two layers, the input layer is fully connected with the recognition field (top-down) where the the-winner-takes-it-all-strategy is used. The activation of the recognition field is propagated back to the input layer, creating resonance (bottom-up). Learning an ART network includes either differential equations (slow learning) or algebraic equations (fast learning). ART networks are used for many pattern recognition tasks, such as automatic target recognition and seismic signal processing.

3.4.4 Learning Algorithms

Training a neural network model essentially means applying a learning algorithm that minimises the error function. There are numerous algorithms available, most of these algorithms employ some form of gradient descent. This is done by simply taking the derivative of the error function with respect to the network parameters and then changing those parameters in a gradient-related direction. Other methods that can be used for training an ANN are evolutionary methods like genetic algorithms, simulated annealing, expectation-maximisation or particle swarm optimization.

After feeding an ANN with some training data, the error function needs to be evaluated.

Definition 44 (Error Function)**.** In the case of on-line learning, the error function is usually

$$E_{SSE} = \frac{1}{2} \sum_{\mathit{\Omega} \in O} (t_{\mathit{\Omega}} - o_{\mathit{\Omega}})^2, \tag{36}$$

based on the summed squared error (SSE). $\mathit{\Omega}$ are output neurons, O is the set of all output neurons of the ANN. Other possibilities for error functions are based on the root mean square (RMS) or, especially in low-dimensional spaces, the euclidean distance. In the case of off-line learning the error function is the sum of all individual errors.

Plotting the error function of the training set and the test set against time/training epochs generates the two training curves. An ideal training curve of the training set behaves like e^{-t}, the training curve of the test set usually lies above the training curve of the training set and oscillates more wildly. If the training curve of the test set starts to ascend, this may be an indication that the ANN is starting to learn the training data by heart and is losing its ability to generalise. The strategy of *early stopping* quits the training, if the training curve of the test set rises over for example 6 consecutive epochs.

Definition 45 (Gradient Descent)**.** Given a differentiable m-dimensional function f and a starting point $p_0 \in \mathbb{R}^m$, a gradient descent method starts at $f(p_0)$ and moves into the direction of the negative gradient $\nabla f(p_0)$ to smaller values of f. This is repeated iteratively.

$$p_{i+1} = p_i - \eta \nabla f(p_i), \tag{37}$$

where η could be $|\nabla f(p_i)|$.

Remark. The gradient descent method is an optimisation method which is not faultless. In most cases these algorithms work, but it is not always possible to even see whether the method worked well or not.

Problems of Gradient Descent Methods. Gradient descent methods search for local minima. It is possible that the difference between a found local minimum and the global minimum is huge. If the function exhibits flat spots, many iterations are necessary to overcome this flat spot. It is possible that the method leaves a good minimum in future iterations or even oscillates in deep canyons without converging to a local minimum.

A basis for more complex learning algorithms is often the Hebbian learning rule:

Algorithm 6 (Hebbian Learning Rule)**.** Let neuron i be connected with neuron j, the weight of the connection shall be ω_{ij}. The basic idea of this algorithm is that the weight of the connection increases, if the activations of both neurons are high. The incrementation happens proportionally to the *learning rate* $\eta > 0$:

$$\Delta\omega_{ij} = \eta o_i o_j. \tag{38}$$

Remark. Hebb's rule (38) is in general unsupervised and unstable for any neuron model.

Algorithm 7 (Delta Rule)**.** For an output neuron Ω and its known teaching input t_Ω the weight update for an ANN with a linear activation function shall be calculated as follows:

$$\Delta\omega_{i\Omega} = \eta o_i (t_\Omega - o_\Omega) = \eta o_i \delta_\Omega. \tag{39}$$

The difference between the teaching input and the actual output of the neuron Ω is called δ_Ω. This is the reason why the algorithm is called delta or *Widrow-Hoff rule.*

The delta rule originates from the idea to see the error function as a function of the weights and to apply the gradient descent method:

$$\Delta W = -\eta \nabla E_{SSE}(W). \tag{40}$$

Using the chain rule

$$\frac{\partial E_{SSE}(W)}{\partial \omega_{i\Omega}} = \frac{\partial E_{SSE}(W)}{\partial o_\Omega} \frac{\partial o_\Omega}{\partial \omega_{i\Omega}}, \tag{41}$$

the first factor can be substituted by δ_Ω and the second is (due to the required linear activation function) o_i.

Remark. As the delta rule only provides an instrument to update the weights of output neurons, it is obviously suited for single-layer networks only. The delta rule belongs to supervised learning algorithms.

The expansion of the delta rule - including a more general delta - to a multi-layer network with more general activation functions is the back-propagation algorithm. Already published by Paul Werbos in 1974, it took the scientific society until 1986 and the publications of work by Rumelhart, McClelland and Hinton to realise the power of this algorithm. The backpropagation works only with monotonous and differentiable activation functions.

Algorithm 8 (Backpropagation)**.** A neuron h is considered. A neuron k is a predecessor of neuron h, a neuron l a successor of h. L shall be the set of all successors of neuron h.

$$\Delta\omega_{kh} = \eta o_k \delta_h \tag{42}$$

with

$$\delta_h = \begin{cases} f'_{act}(net_h)(t_h - o_h) & \text{if } h \text{ is an output neuron,} \\ f'_{act}(net_h)\sum_{l\in L}\delta_l\omega_{hl} & \text{if } h \text{ is an inner neuron.} \end{cases} \tag{43}$$

Choosing a good η. The smaller η is the higher is the chance to find a minimum of the error function, but the calculating time increases with smaller η. Experiences teach to choose the learning rate as follows:

$$0.01 \leq \eta \leq 0.9, \tag{44}$$

often starting with a large η and reducing it stepwise during the training. Learning rates closer to the input layer should be larger than learning rates closer to the output layer. These are two main issues of the backpropagation algorithm: That it is difficult to choose a good learning rate and that learning far away from the output layer is very slow. An extension to this algorithm that tries to deal with these issues is the so called *resilient backpropagation.*

Resilient Backpropagation. Resilient backpropagation (Rprop) does not use a global learning rate, every weight ω_{ij} has its own learning rate η_{ij} which is not chosen by the modeller, but by the Rprop algorithm itself. Furthermore Rprop updates the learning rates after every learning step. Thus it is better to talk about learning rates $\eta_{ij}(t)$. Another different approach of the Rprop is that the weight update is not proportional to the gradient of the error function anymore, as it is in the case of the backpropagation algorithm. This property of the backpropagation algorithm usually leads to jagged weight updates, if the error function is full of fissures. For the Rprop algorithm the following equation holds:

$$|\Delta\omega_{ij}(t)| = \eta_{ij}(t). \tag{45}$$

If the gradient $g = \dfrac{\partial E_{SSE}(W)}{\partial \omega_{ij}}$ is positive, $\Delta\omega_{ij}(t)$ is the negative learning rate $-\eta_{ij}(t)$, if the gradient is negative, then $\Delta\omega_{ij}(t) = \eta_{ij}(t)$, if the gradient is 0, the weight does not change. This leads to a smoother learning. The learning rates are updated according to the following formula:

$$\eta_{ij}(t) = \begin{cases} \eta^{\uparrow}\eta_{ij}(t-1) & \text{if } g(t-1)g(t) > 0, \\ \eta^{\downarrow}\eta_{ij}(t-1) & \text{if } g(t-1)g(t) < 0, \\ \eta_{ij}(t-1) & \text{if } g(t-1)g(t) = 0. \end{cases} \tag{46}$$

If the sign of the gradient has changed in the last two iterations, this means that there must be a local minimum between the two weights and the weight updates should be smaller to find it. The last update will not be done. If the sign of the gradient has not changed, it is possible to slightly increment the learning rate. Instead of choosing the learning rate manually for every iteration, for Rprop the modeller has to choose the two constants $\eta^{\uparrow}$ and $\eta^{\downarrow}$, the initial learning rates and an upper boundary η_{max} and a lower boundary η_{min} for the learning rates. Standard choices for these parameters are: $eta^{\uparrow} = 1.2, \eta^{\downarrow} = 0.5, \eta_{ij}(0) = 0.1, eta_{max} = 50$ and $\eta_{min} = 10^{-6}$.

Remark. Because the error function and its gradient need to be same for many iterations, Rprop is only applicable for off-line learning. As Rprop overcomes the problem that learning far away from the output layer is too slow, it is highly recommendable for deep networks.

As the error function E_{SSE} is a sum of squares of the residuals $r_{\Omega} = t_{\Omega} - y_{\Omega}$, it is also possible to use the *Gauss-Newton* algorithm instead of a gradient descent method to minimise it. w shall be the vector including all weights, r the vector of the residuals.

Algorithm 9 (Gauß-Newton).

$$w^{(s+1)} = w^{(s)} - \left(J_r^T J_r\right)^{-1} J_r^T r, \qquad \text{where } J_r = \frac{\partial r_i}{\partial w_j}(w^{(s)}). \tag{47}$$

Remark. $J_r^T J_r$ is used as an approximation of the Hessian matrix.

More commonly used than the Gauß-Newton algorithm is the *Levenberg-Marquardt* algorithm (LMA), a combination of a gradient descent method and Gauß-Newton.

Algorithm 10 (Levenberg-Marquardt). Basically, the main task of the Levenberg-Marquardt algorithm is to solve the following equation with respect to Δw:

$$(J^T J + \lambda I)\Delta w = J^T r \tag{48}$$

λ is the *damping factor*, which is adjusted at each iteration. If the reduction of r is rapid, a smaller value can be used, if the reduction is insufficient, the damping factor can be increased. $\lambda \to 0$ means that the LMA becomes the Gauß-Newton algorithm, $\lambda \to \infty$ signifies that the LMA becomes a gradient descent method. Usually the initial value of λ is 0.1.
If $J^T J$ is not singular, the equation system can be solved by using the LU decomposition, if it is singular, the system is solved by SVD decomposition.

Remark. The LMA converges locally quadratically. It is very sensitive to the initial network weights. Also, it does not consider outliers in the data. To avoid those situations a technique known as *regularisation* can be used. If a network training works with the LMA, the training is usually fast.

Further Algorithm Adaptations. There are lots of adaptations to the backpropagation and the other learning algorithms. Which one is the best for a certain application, depends on the problem that is modelled. Very often this is not known a priori and needs to be tested. Adaptations can include a momentum term, second order derivatives, flat spot elimination, pruning/optimal brain damage (elimination of neurons with weights close to 0) or the use of different error functions like the cross-entropy or a weight decay addend:

Definition 46 (Weight Decay Error Function).

$$E_{WD} = E_{SSE} + \beta \frac{1}{2} \sum_{\omega \in W} \omega^2 \tag{49}$$

β regulates the penalisation of large weights[19] and is usually chosen from the interval $[0.001; 0.02]$. W shall be the set of weights in this definition.

[19]Again, this is biologically inspired. Synaptic weights cannot become extremely large.

Dropout Technique. On each presentation of each training datum, each hidden unit is randomly omitted from the network with a probability of 0.5, so a hidden unit cannot rely on other hidden units being present. This mimics the standard technique of training several ANNs and averaging them, but it is faster. This dropout technique provides a more robust learning without overfitting.

3.4.5 Time Series Prediction with Neural Networks

Forecasting seasonal time series is still a difficult task for researchers. There are numerous models to analyse and forecast a time series (the classical statistical VSARIMA models have been presented elaborately in section 2), but unfortunately there is no single modelling approach that is best for all seasonal time series [14]. In fact there exist lots of different recommendations and conflicting results. One among many difficulties is to distinguish seasonal from non-seasonal fluctuations, an other is to choose a highly adequate model.

Using NNs, which are non-linear and data-driven by nature and are therefore theoretically very well suited to model seasonality interacting with other components, can at least transfer the burden of a difficult a priori model selection to the selection of NN parameters. Nevertheless, also in the case of using NNs results obtained by various authors differ widely in quality: Some suggest that NNs are better than other forecasting models, others contradict them. Some have seemed to obtain better results with seasonally adjusted data, others think that NNs are able to directly model seasonality in an implicit way, without any seasonal adjustments on the input data. Detailed research results are presented in [14], indicating that NNs may be inferior to the Box-Jenkins model, if they shall model seasonality directly for short time series, but even in this case some adaptations may lead to better performances than those achieved by linear time series models. Lapedes and Farber (1988) were the first to report that simple neural networks can outperform traditional methods by up to many orders of magnitude.

In 1991 Sharda, Patil and Tang identified a number of facts that determine which method is superior, by experiments:

- For time series with long memory, both approaches deliver similar results.
- For time series with short memory, NNs outperform the traditional Box-Jenkins approach in some experiments by more than 100%.

- For time series of various complexity, the optimally tuned neural network topologies are of higher efficiency than the corresponding traditional algorithms. [1]

When the modeller of a neuro-predictor is able to influence the data acquirement, two main issues should be considered: the sampling period (see theorem 1 for further details) and the number of data needed. Simon Haykin suggests to choose the number of training patterns based on

$$N \approx \frac{W}{\varepsilon}. \tag{50}$$

W shall be the number of weights used in the NN, ε shall be the error the training examples should be classified with and N shall be the number of patterns in the training set. [1]

When using NNs to forecast time series, data normalisation is a key issue. Various normalisation methods can be applied, logarithmic or exponential scaling can be used if problems with non-linearities are expected during the network training. Linear normalisations can be used to meet the requirements of the network input layer, as the input range must not be too wide. Significant patterns as seasonality and trends should be removed, if possible, to make the NN time series model easier. To be able to use the concept of cross-validation, appropriate training, test and validation data sets need to be chosen.

The tasks of structuring the data and choosing the number of input nodes of the NN predominantly depend on the number d of lagged values to be used for forecasting of the next value in the standard case of a one-step-ahead prediction. Additional dummy variables can also be used as input variables, but they have not proven to significantly improve the NN forecasting performance [14]. Thus the function to be modelled by the NN is of the type

$$x_{t+1} = f(x_t, x_{t-1}, \ldots, x_{t-d+1}). \tag{51}$$

The number of output neurons directly corresponds to the forecasting horizon, i.e. in the case of a one-step-ahead forecast there is only one output neuron. Usually only one hidden layer is used, the number of the neurons in the hidden layer can be chosen according to the geometric pyramid rule (35). Choosing the number of hidden neurons as well as the data normalisation involves trial-and-error experimentation.

As activation function in the hidden layer either the sigmoid function or the hyperbolic tangent shall be used, while for the output layer the linear activation function provides the best results. A non-linear activation function in the output layer is only needed, if the time series shows a significant trend even after the data preprocessing.

If there are several NN models that the modeller finally can choose from, he can apply an adapted version of the *AIC*:

Definition 47 (Akaike's Information Criterion for NNs)**.** With N being the number of training examples, n_o the number of output neurons, σ^2 the maximum likelihood estimate of the mean squared error for the training data and k the number of model parameters, the *AIC* from definition 25 can be rewritten as:

$$AIC = Nn_o \ln(\sigma^2) + 2k. \tag{52}$$

The model with the smallest *AIC* shall be preferred. The *network information criterion NIC* is a generalisation of the *AIC* and easier to apply. Further details can be checked in [1].

The network training occurs as described in section 3.4.4.

A hybrid combination of neural networks and traditional approaches has been found very promising, but excesses the range of this thesis.

3.5 Clustering

Definition 48 (Clustering)**.** Cluster analysis is a set of methodologies for automatic classification of samples into a number of groups using a measure of association, so that the samples in one group are similar and samples belonging to different groups are not similar. The input for a system of cluster analysis is a set of samples and a measure of similarity or dissimilarity between two samples. The output from a cluster analysis is a number of groups (the so-called *clusters*) that form a partition of the data set. [3]

Considering the already presented different learning approaches, a cluster analysis can be done either in an unsupervised or in a supervised way. It is a primary data mining task, and there are tons of different clustering algorithms, especially due to the fact that a cluster is defined rather imprecisely. The most common measures of similarity used in clustering algorithms include distance functions (some of them are presented in section 4.1) or statistical distributions.

One of the most simple clustering algorithms is the unsupervised *K-means algorithm*, that uses the Euclidean distance:

Algorithm 11 (K-Means Clustering - Lloyd's Algorithm)**.** The algorithm needs a fixed number K of clusters as well as n observations $\{x_1, \ldots, x_n\}$ as input. K means $m_1^{(1)}, \ldots, m_k^{(1)}$ need to be chosen initially[20]. The algorithm alternates between an assignment and an update step. First, each observation is assigned to the cluster C_i whose mean is closest to it:

$$C_i^{(t)} = \left\{ x_l : \|x_l - m_i^{(t)}\| \leq \|x_l - m_j^{(t)}\| \quad \forall\, 1 \leq j \leq k \right\} \tag{53}$$

After this assignment new means are calculated:

$$m_i^{(t+1)} = \frac{1}{|C_i^{(t)}|} \sum_{x_l \in C_i^{(t)}} x_l \tag{54}$$

The algorithm has converged when the assignments no longer change.

The idea that several examples which do not clearly belong to a single class could exist implies that a fuzzy approach should make sense for a lot of clustering tasks. In fuzzy clustering, also called soft clustering, an observation can belong to more than one class. The most simple fuzzy clustering algorithm is the *fuzzy c-means algorithm*:

Algorithm 12 (Fuzzy C-Means Clustering)**.** Given a chosen number C of clusters with initial centres $C_j^{(1)}, j = 1, \ldots, C$, n observations $\{x_1, \ldots, x_n\}$ and additionally a fuzzifier $m > 1$ and a threshold ε and/or a maximum number of iterations t_{max}, the algorithm returns the cluster centres $C_j, j = 1, \ldots, C$ and a partition matrix $W \in [0,1]^{n \times C}$, where each element w_{ij} tells the degree to which element x_i belongs to cluster C_j.

$$w_{ij}^{(t)} = \frac{1}{\sum\limits_{l=1}^{C} \left(\frac{d(x_i, C_j^{(t)})}{d(x_i, C_l^{(t)})} \right)^{\frac{2}{m-1}}}. \tag{55}$$

After calculating these degrees, the new means are the means of all points, weighted by their degree of belonging to the cluster:

$$C_j^{(t+1)} = \frac{\sum\limits_{i=1}^{n} x_i w_{ij}^{(t)}}{\sum\limits_{i=1}^{n} w_{ij}^{(t)}}. \tag{56}$$

The algorithm stops, if the maximum number of iterations is reached or the fuzzy coefficients do not change more than the given threshold ε:

$$\max_{i,j} |w_{ij}^{(t+1)} - w_{ij}^{(t)}| \leq \varepsilon \quad \vee \quad t \geq t_{max}. \tag{57}$$

[20]Superscript numbers in parenthesis indicate which iteration the algorithm is in.

Results depend on the choice of the distance function, the chosen number of clusters C and the fuzzifier m. The larger m, the fuzzier the clusters; $m \to 1$ leads to crisp clustering.

A complete cluster analysis usually also varies the number of classes to choose the most appropriate number of clusters. In general, this can be done in an agglomerative or a divisive way using *single, complete* or *average linkage* or *Ward's method*.

3.6 Support Vector Machines

Support vector machines (SVM) are (in general supervised) learning models used for classification and regression tasks. They were invented by Vladimir Vapnik. The SVM in its basic form aims directly for the decision boundary between the two classes the data can be separated into. Imagining two linearly separable two-dimensional data classes, many lines can separate the two classes. An SVM selects the line that maximises the *margin*, the space between the decision boundary and the closest points from each of the classes. These points closest to the decision boundary are called the *support vectors*. For m-dimensional data, the SVM tries to find the separating $m-1$-dimensional hyperplane with a maximised margin.

Given a set of labelled training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, depending on which side of the maximised gap they fall on. Any distance (or similar values) from the new example to the decision boundary can be seen as some degree of certainty the example belongs to the class the SVM model suggests. If these values are normalised, they can be looked at as fuzzy values.

Using training data $D = \big\{(x_i, y_i) | x_i \in \mathbb{R}^m, y_i \in \{-1, 1\}, i = 1, \ldots, n\big\}$, where y_i indicates which class x_i belongs to, the hyperplane can be written as the set of points x that satisfy

$$wx + b = 0 \tag{58}$$

for a normal vector w and a bias b ($-\frac{b}{\|w\|}$ determines the offset of the hyperplane from the origin along the normal vector w.). The region bounded by the hyperplanes

$$wx + b = +1 \tag{59a}$$
$$wx + b = -1 \tag{59b}$$

has no element of D in it and is exactly the margin. It can easily be calculated that the distance between those two hyperplanes is $\frac{2}{\|w\|}$, thus the SVM

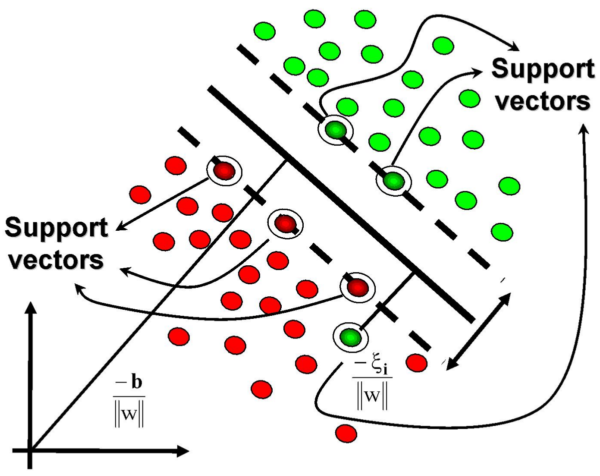


Figure 15: Illustration of the basic idea of an SVM.

algorithm tries to minimise $\|w\|$ to maximise the margin. Rewriting this as a quadratic optimisation problem - minimise $\frac{1}{2}\|w\|^2$ in (w, b) -, the margin is maximised subject to the following constraints, which assure that data points do not fall into the margin:

$$wx + b \geq +1 \quad \text{for } y_i = +1 \tag{60a}$$
$$wx + b \leq -1 \quad \text{for } y_i = -1 \tag{60b}$$

In addition to performing linear classification, SVMs can efficiently perform non-linear classification using two different approaches:

- If the two classes are almost linearly separable, the SVM allows a small number of observations to be on the wrong side of the decision boundary by adapting the equation (58) to equation (61), introducing the slack variables ξ_i and the budget B for observations on the wrong side of the decision boundary. The obtained margin is called *soft margin*.
- If the two classes are certainly not linearly separable, the so-called *kernel trick* can be used, an implicit mapping of the inputs x_i into high-dimensional feature spaces, where the mapped vectors are linearly separable. How this can be done is described in the following chapter 3.6.1.

$$wx + b \geq 1 - \xi_i \quad \text{for } y_i = 1 \tag{61a}$$
$$wx + b \leq -1 + \xi_i \quad \text{for } y_i = -1 \tag{61b}$$
$$\sum_{i=1}^{n} \xi_i \leq B \tag{61c}$$

With or without slack variables, the quadratic optimisation problem is solved by introducing *Lagrange multipliers.* As there are inequality constraints, the SVM algorithm also needs the *Karush-Kuhn-Tucker (KKT) conditions.*

Theorem 5 (Karush-Kuhn-Tucker Conditions)**.** *Considering a non-linear optimisation problem*

$$\min f(x) \quad s.t. \quad g_i(x) \leq 0, h_j(x) = 0; \quad i = 1, \ldots, k; j = 1, \ldots, l \tag{62}$$

and supposing that f, g_i and h_j are continuously differentiable in $x^\star$ which is a local minimum that satisfies some regularity conditions[21]*, there exist constant KKT multipliers μ_i and λ_j such that*

$$\nabla f(x^\star) + \sum_{i=1}^{k} \mu_i \nabla g_i(x^\star) + \sum_{j=1}^{l} \lambda_j \nabla h_j(x^\star) = 0, \tag{63a}$$
$$g_i(x^\star) \leq 0, h_j(x^\star) = 0 \quad \forall i, j, \tag{63b}$$
$$\mu_i \geq 0 \quad \forall i, \tag{63c}$$
$$\mu_i g_i(x^\star) = 0 \quad \forall i \tag{63d}$$

Remark. For $k = 0$ the KKT conditions turn into the Lagrange conditions.

Remark. The class of functions in which KKT conditions guarantee global optimality are *type 1 invex functions.* Subdifferential versions of the KKT conditions are also available.

With Lagrange multipliers α_i the previous constrained problem can be expressed as

$$\min_{w,b} \max_{\alpha_i \geq 0} \left\{ \frac{1}{2} \|w\|^2 - \sum_{i=1}^{n} \alpha_i \Big[y_i(wx_i + b) - 1 \Big] \right\} \tag{64}$$

[21]Some regularity conditions or constraints qualifications are needed, but they are not part of this thesis. For further details see `http://en.wikipedia.org/wiki/Karush-Kuhn-Tucker_conditions`.

The KKT condition (63a) implies that the solution can be expressed as a linear combination of the training vectors:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i. \tag{65}$$

All the points which can be separated by the margin do not matter since the corresponding α_i must be set to 0. The x_i that correspond to the non-zero α_i are exactly the support vectors, which lie on the margin. From this and averaging over all N support vectors

$$b = \frac{1}{N} \sum_{i=1}^{N} y_i - w x_i \tag{66}$$

can be derived. The problem can now be solved using some standard quadratic programming technique like interior point, active set, augmented Lagrangian, conjugate gradient, gradient projection or extensions of the simplex algorithm.

Writing the classification rule in its unconstrained *dual form* reveals that the maximum margin hyperplane and therefore the classification task is only a function of the support vectors. Without going into too much detail, the dual problem can be written as

$$\max_{\alpha_i} L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \tag{67a}$$

$$\text{s.t.} \quad \alpha_i \geq 0 \quad \text{and} \quad \sum_{i=1}^{n} \alpha_i y_i = 0; \quad i = 1, \ldots, n. \tag{67b}$$

K is any adequate kernel function.

3.6.1 Kernels

Definition 49 (Kernel)**.** A kernel K is a symmetric real function with arguments from $X \times X$ (X is the input space.), so that for all $x, y \in X$

$$K(x, y) = \langle \varphi(x), \varphi(y) \rangle, \tag{68}$$

where φ is a (non-linear) mapping from the input space X into the high-dimensional Hilbert space F (the feature space) provided with the inner product $\langle ., . \rangle$.

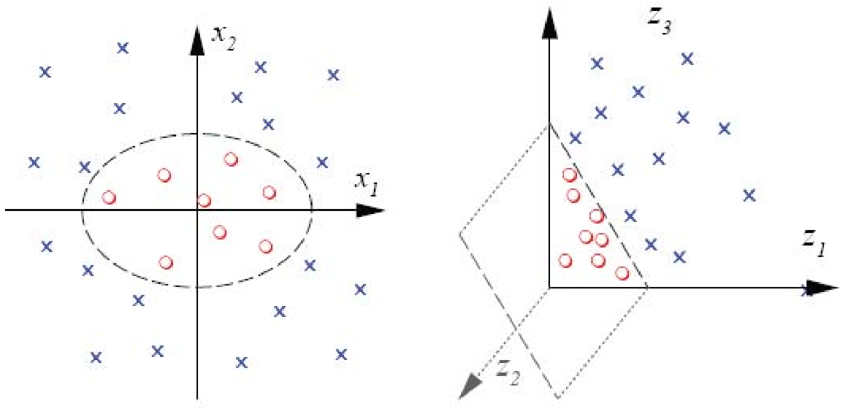


Figure 16: Illustration of the kernel trick by mapping the data into a high-dimensional feature space, where it is linearly separable.

In general it is not necessary to exactly know F, whose dimension is usually the number of pairs of data points, due to *Mercer's theorem.* Following this theorem, theoretically a kernel should be positive semi-definite. The kernel trick is to avoid the explicit mapping and to calculate the high-dimensional dot products within the original space by using the kernel function. The linear separation of the data occurs in the high-dimensional F. Manipulating points in the feature space has the effect of stretching or compressing areas of the original data space which may help to separate the data correctly.

It remains to say that with primitive data a good kernel is easy to find, for very complex problems finding a sensible kernel might be much harder. Some widely used kernels are presented below.

Algorithm 13 (Polynomial Kernel)**.**

$$K_p(x,y) = (x^T y)^p = \langle x, y\rangle^p \tag{69}$$

Algorithm 14 (Gaussian Kernel)**.**

$$K(x,y) = e^{\frac{-\|x-y\|^2}{2\sigma}}, \quad \sigma > 0 \tag{70}$$

Algorithm 15 (Sigmoid Kernel)**.**

$$K(x,y) = \tanh(\kappa\langle x, y\rangle + \vartheta), \tag{71}$$

where $\kappa > 0$ and $\vartheta < 0$.

Remark. The sigmoid kernel is an example for non-positive definite kernels that nevertheless have been used successfully.

Other kernels use B_n-splines of odd order or other RBFs than the Gaussian function.

4 Abnormal Event Detection

According to definition 5, an abnormal event is an outlier in a chain of events. There are three fundamental approaches to detect outliers [16]:

1. Model neither normality nor abnormality. Determine the outliers with no prior knowledge of the data. This is essentially a learning approach analogous to unsupervised clustering.

2. Model both normality and abnormality. This approach is analogous to supervised classification and requires pre-labelled data, tagged as normal or abnormal.

3. Model only normality, maybe tolerate abnormality in very few cases. Authors generally name this technique novelty detection or novelty recognition, especially if only normal data is given. It is analogous to a semi-supervised recognition or detection task. Only the normal class is taught but the algorithm learns to recognise abnormality. The approach needs pre-classified data but only learns data marked normal.

The curse of dimensionality is a main issue of anomaly detection, and the dimension of the data highly influences how good a chosen method can work for a certain data set. Another very important issue is data preprocessing.

Theorem 6 (Curse of Dimensionality)**.** *Considering a certain point p in an m-dimensional data set and any distance d, results of many studies state that the relative contrast of the distance d_{max} between the point farthest away from p and p and the distance d_{min} between the point nearest to p and p converges to 0 for increasing dimensionality:*

$$\lim_{m\to\infty} \frac{d_{max} - d_{min}}{d_{min}} \to 0. \tag{72}$$

4.1 Proximity-Based Anomaly Detection

Models based on spatial proximity are most commonly distance-based or density-based. The computational complexity is directly proportional to the dimension m of the data and n, the number of observations, which can become a major problem for high-dimensional data sets. Proximity-based techniques define a data point as an outlier, if its locality (or proximity) is sparsely populated. They usually do not make prior assumptions about the data distribution and can be used in the sense of the fundamental approaches 1 and 2 described at the beginning of this section 4.

4.1.1 Distance-Based Anomaly Detection

The distance-based approach takes into account the following basic ideas:

- A point is judged based on the distance(s) to its neighbours.
- Normal points have a dense neighbourhood.
- Outliers are far apart from their neighbours, i.e., they have a less dense neighbourhood.

Various distances can be used, some of the most common ones are presented below.

Definition 50 (Manhattan Distance)**.** Hermann Minkowski considered the so-called taxicab geometry in the 19th century. In taxicab geometry the distance between two points $x = (x_1, \dots, x_m)^T$ and $y = (y_1, \dots, y_m)^T$ is the sum of the absolute differences of their coordinates. This metric is called *Manhattan distance* and is induced by the ℓ_1-norm:

$$d_1(x, y) = \|x - y\|_1 = \sum_{i=1}^{m} |x_i - y_i|. \tag{73}$$

Definition 51 (Euclidean Distance)**.** The Euclidean distance is the 'ordinary' distance between two points $x = (x_1, \dots, x_m)^T$ and $y = (y_1, \dots, y_m)^T$ that one would measure with a ruler and that is given by the Pythagorean formula. This metric is induced by the ℓ_2-norm:

$$d_2(x, y) = \|x - y\|_2 = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}. \tag{74}$$

Definition 52 (Mahalanobis Distance)**.** The Mahalanobis distance, introduced by Prasanta Chandra Mahalanobis in 1936, is based on correlations. Let S be the covariance matrix:

$$d_M(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)}. \tag{75}$$

Remark. For $S = I$, I being the identity matrix, the Mahalanobis distance becomes the Euclidean distance (74). If S is a diagonal matrix, d_M is called *normalised Euclidean distance.*

The Mahalanobis distance is computationally expensive to calculate for large high dimensional data sets compared to the Euclidean distance as it requires a pass through the entire data set to identify the attribute correlations. Apart from that the Mahalanobis distance values become more similar with increasing degrees of freedom (curse of dimensionality). d_M is closely related to the leverage h:

$$d_M^2(x, \bar{x}) = (m-1)(h - \frac{1}{m}). \tag{76}$$

Definition 53 (Leverage)**.** The leverage is defined as the diagonal of the so-called *hat matrix* H of a regular square matrix X:

$$H = X(X^T X)^{-1} X^T. \tag{77}$$

Remark. High-leverage points are those that cause large changes in the parameter estimates when they are deleted and can therefore be seen as outliers with respect to the independent variables.

Concrete distance-based outlier detection algorithms are for example the DB(ε, π)-method and outlier scoring based on k nearest neighbour (kNN) distance d_k, the distance to the k^{th} nearest neighbour of a datum.

Algorithm 16 (DB(ε, π) Outlier Detection)**.** A radius ε and a percentage π shall be given beside the data set $\mathcal{D}$. A point p is considered an outlier if at most π percent of all other points have a distance to p less than ε. Thus the set of outliers O is defined as

$$O = \left\{ p \in \mathcal{D} : \frac{|\{q \in \mathcal{D} : d(p,q) < \varepsilon\}|}{|\mathcal{D}|} \leq \pi \right\} \tag{78}$$

If the data set is clustered, the kNN approach can be substituted by comparing new data only with the cluster means, which reduces the computational complexity a lot.

4.1.2 Density-Based Anomaly Detection

The general idea of this approach is to compare the density around a point with the density around its local neighbours. The relative density of a point compared to its neighbours can be computed as an outlier score. The density around a normal data object should be similar to the density around its neighbours, whereas the density around an outlier tends to be considerably different to the density around its neighbours.

Density-based methods overcome the problem of data sets with different densities, which usually cannot be solved by distance-based methods.

The local outlier factor (LOF) algorithm developed by Markus Breunig, Hans-Peter Kriegel et. al. does exactly that; it compares the density of a point with the density of its neighbours, a point with a significantly lower density is considered as an outlier. The LOF algorithm uses the kNN distance and the *reachability distance*, which is mathematically spoken not a distance, because it is not symmetric:

Definition 54 (Reachability Distance)**.**

$$d_{reach,k}(A,B) = \max\{d_k(B), d(A,B)\} \tag{79}$$

The use of $d_{reach,k}$ makes the local outlier factor algorithm more stable.

Definition 55 (Local Reachability Density)**.** $N_k(A)$ shall be the set of the k nearest neighbours of the point A.[22] Then the local reachability density $lrd_k(A)$ of the point A is defined as:

$$lrd_k(A) = \frac{|N_k(A)|}{\sum\limits_{B \in N_k(A)} d_{reach,k}(A,B)}. \tag{80}$$

Algorithm 17 (Local Outlier Factor)**.** Using the definition from above, the local outlier factor of a point A and for a chosen k is defined as:

$$LOF_k(A) = \frac{\sum\limits_{B \in N_k(A)} \frac{lrd_k(B)}{lrd_k(A)}}{|N_k(A)|}. \tag{81}$$

A value of approximately 1 indicates that the considered objects is comparable to its neighbours, a value significantly higher than 1 indicates an outlier.

If clusters of different densities are not clearly separated, the LOF algorithm will be confronted with problems. These can be overcome by the influenced outlierness (INFLO) algorithm, which was developed by Jin et al. in 2006. It takes into account a symmetric neighbourhood; the influence space of a point p includes not only the kNN, but also the reverse kNN (the set of objects whose kNN contain p). Another rather specified algorithm is the local outlier correlation integral algorithm.

4.1.3 Statistical Anomaly Detection

A certain kind of statistical distribution (e.g., Gaussian) must be given. All the parameters like mean and standard deviation of this distribution can be calculated under the assumption that all data points have been generated by

[22]In case of a tie this set can include more than k elements.

the chosen statistical distribution. The idea of statistical outlier detection is that outliers are those points that have the lowest probability to be generated by the distribution. Basically, it is up to the modeller to choose how sensitive the detection should be, but a rule of thumb is that those data points that deviate more than three times the standard deviation from the mean are the outliers.

Some of the main problems of statistical tests to detect outliers are:

- Robustness:
  - Mean and standard deviation are very sensitive to outliers.
  - d_M is used to detect outliers, although it is influenced by them.
- The distribution is usually fixed.

4.2 Angle-Based Outlier Detection

Angles are more stable than distances in high-dimensional spaces, which suggests the use of angles instead of distances for high-dimensional data. In fact, the situation is contrary for low-dimensional data. The angle-based outlier detection (ABOD) method alleviates the effects of the notorious curse of dimensionality compared to purely distance-based methods.

Following the idea of the algorithm developed by Kriegel et al. in [17] in 2008, a point is considered as outlier, if most other points are located in a similar direction, and a point is considered as an inlier, if many other points are located in varying directions. The broadness of the spectrum of the angles between a certain point p and all pairs of the other points is a score for the outlierness of p: The smaller the score, the greater is the point's outlierness. The angles in the so-called *angle-based outlier factor* are weighted by the squared inverse of the corresponding distances to avoid bigger problems with low-dimensional data sets.

Algorithm 18 (Angle-Based Outlier Detection)**.** A, B and C shall be any points in a data set $\mathcal{D}$. The angle-based outlier factor $ABOF$ is defined as:

$$\begin{aligned} ABOF(A) &= \underset{B,C\in\mathcal{D}}{\mathbb{VAR}} \left(\frac{\langle \vec{AB}, \vec{AC} \rangle}{\|\vec{AB}\|^2 \|\vec{AC}\|^2} \right) \qquad (82) \\ &= \sum_{B\in D} \sum_{C\in\mathcal{D}} \left(\frac{\langle \vec{AB}, \vec{AC} \rangle}{\|\vec{AB}\|^2 \|\vec{AC}\|^2} \right)^2 - \left(\sum_{B\in\mathcal{D}} \sum_{C\in\mathcal{D}} \frac{\langle \vec{AB}, \vec{AC} \rangle}{\|\vec{AB}\|^2 \|\vec{AC}\|^2} \right)^2 . \end{aligned}$$

The naive algorithm is $O(m^3 n)$, which is not very attractive, but most implementations do not use all pairs of other points and use a good approximation of the $ABOF$ instead of the exact value.

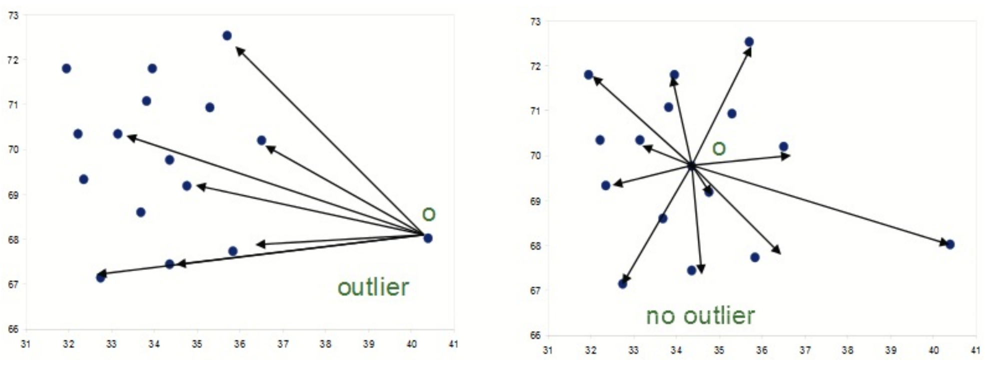


Figure 17: Illustration of the angle-based outlier detection method.

Algorithm 19 (Approximative Angle-Based Outlier Detection by Kriegel et al.)**.** One possibility to approximate the $ABOF$ is to calculate the variance of the angles only of the pairs of points which are kNN of A, since these are the ones with the largest weights in the formula (82). This FastABOD algorithm is $O(m^2 + k^2m)$. Again, $N_k(A)$ shall be the set of the k nearest neighbours of the point A.

$$\begin{aligned} ABOF_k(A) &= \underset{B,C\in N_k}{\mathbb{VAR}} \left(\frac{\langle \vec{AB}, \vec{AC} \rangle}{\|\vec{AB}\|^2 \|\vec{AC}\|^2} \right) \qquad (83) \\ &= \sum_{B\in N_k} \sum_{C\in N_k} \left(\frac{\langle \vec{AB}, \vec{AC} \rangle}{\|\vec{AB}\|^2 \|\vec{AC}\|^2} \right)^2 - \left(\sum_{B\in N_k} \sum_{C\in N_k} \frac{\langle \vec{AB}, \vec{AC} \rangle}{\|\vec{AB}\|^2 \|\vec{AC}\|^2} \right)^2 . \end{aligned}$$

Experiments show that the quality of the results of the FastABOD algorithm becomes worse for higher-dimensional data, but it works great on lower-dimensional big data sets. The lower bound FastABOD (LB-ABOD) algorithm, which is also presented in [17], works great for higher-dimensional data sets: Candidates for the top l outliers with respect to the lower bound are selected and afterwards this list is refined until none of the remaining candidates can have a lower $ABOF$ than the largest of the best already examined data objects.

Remark. For any triple in question, the three points A, B and C have to be mutually different. This is left out in the algorithm description above in favour of readability.

In [18], Ninh Pham and Rasmus Pagh introduce an approximative ABOD algorithm that is based on random projections and AMS sketches. This approximation algorithm runs in near-linear time in the size of the data set and is also well suited for parallel processing. To be exact, the algorithm is $O(n \log n(m + \log n))$. In their paper, the two Danes present a theoretical analysis of the quality of approximation to guarantee the reliability of their estimation algorithm as well as empirical experiments on synthetic and real world data sets to demonstrate the scalability, effectiveness and efficiency of their approach on detecting outliers in very large high-dimensional data sets.

As the angles in the classic ABOD algorithm are weighted in the calculation of the variance in (82), different weights can be used for similar algorithms, of course. One of the most promising approaches is to use time-dependent weights, if the available data is time-stamped.

Two other outlier detection algorithms for high-dimensional data that follow the idea of searching for outliers in subspaces are:

- grid-based subspace outlier detection
- subspace outlier degree (SOD).

Other approaches to detect outliers in high-dimensional data sets use feature reduction methods like PCA.

4.3 Neural Networks for Anomaly Detection

If normality and abnormality are modelled and therefore a supervised approach is used, a standard multi-layer-perceptron should be able to solve the classification task. As always, it is far more difficult if it is not possible to model abnormality and if an unsupervised or only semi-supervised model can be used. Neural networks that can be used to detect outliers in a not supervised way are:

- self-organising maps
- adaptive resonance theory
- replicator neural networks.

As described in section 3.4.3, SOMs are competitive, unsupervised ANNs, and the modeller is interested in what neuron is the most active for some input data. Each neuron in the grid has an associated weight vector that can be seen as analogon to the mean vector representing each cluster in a K means system. The updating processes of SOMs and K means systems differ, as SOMs also update the neighbouring neurons, which clusters the network into regions of similarity. When applied to the field of novelty recognition, Saunders and Gero as well as Vesanto defined a new observation as novelty, if the error between the best matching unit and the input vector (i.e. the new observation) is greater than a chosen threshold ε that directly controls the sensitivity of the anomaly detection. [16]

As ART models are designed to also encounter new classes, it is possible to use a ART network that is plastic while learning and stable while classifying. The network can return to plasticity to learn again. If a new input vector does not match an existing class well enough (This is controlled by a user-specified vigilance threshold ε.), the ART model creates a new class by adding a new node. Every time the network adds a new node, this indicates a change in the time series. [16]

The replicator neural network used for outlier detection aims to reproduce the input data pattern at the output layer after an internal data compression and extraction. During the training the mean square error - in this context also called *reconstruction error* - is minimised for all training examples, which makes common patterns more likely to be well reproduced by the trained replicator ANN [20]. The network uses three hidden layers, both the input and output layer have m neurons, the dimension of the data. It is fully connected, the activation functions of the two outer hidden layers are

$$f_{act,k}(net_j) = \tanh(a_k net_j) \qquad i \in \{2, 4\}, \tag{84}$$

where net_j is the weighted input sum of the neuron N_j, k indicates the number of the layer including the input layer and a_k is a tuning parameter which can be set to 1 initially. The key idea of the replicator ANN is the use of a staircase activation function with K steps in the middle hidden layer. This divides the continuously distributed data points into a number of discrete valued vectors. This data compression can be seen as similar to clustering. As always for this kind of tasks, a good choice of K is essential for a satisfactory outcome of the outlier detection. After the extraction of the data in the lowest hidden layer, the output layer can either use a linear or a sigmoid function as activation function. Usually the outer hidden layers consist of more neurons than the middle hidden layer with n_3 neurons, it is reasonable that they even use the same number of neurons. Checking the outputs of the middle hidden layer, the modeller automatically obtains K^{n_3} clusters.

The average reconstruction error over all m features is defined as outlier factor OF_i of the corresponding i^{th} observation.

$$OF_i = \frac{1}{m} \sum_{j=1}^{m} (x_{ij} - o_{ij})^2, \tag{85}$$

where x_{ij} is the value of the j^{th} feature of the i^{th} observation and o_{ij} is the output of the j^{th} output neuron, when the i^{th} observation is the input of the replicator ANN.

After calculating the outlier factor for the whole data set, the observations can be ranked according to a large OF, the observations at the top of this list are the main candidates for outliers, which can be investigated further by other methods, if necessary.

4.4 One-Class Support Vector Machines

4.4.1 Idea

In general, one-class support vector machines (OC-SVMs) are designed for the certain kind of a $(1 + x)$-*class learning task*. This is a model with an unknown number of classes, but the modeller is only interested in one specific class. Typical examples for these kinds of tasks are content-based image retrieval or document-retrieval in general. Making enquiries for this thesis on the internet can be seen as such a task: Papers which treat relevant topics are alike, they represent the class the modeller is interested in. These are the positive examples and it is easy to find some good representatives of this class. The negative examples are simply the rest of the web pages or papers, and they originate from an unknown number of different negative classes.

It is daunting and wrong to try to characterise the distribution of the negatives in such cases; they could belong to any negative class, and the modeller is not even interested which exact negative classes they might belong to. Each negative example is negative in its own way, but as the positive ones are alike, it is possible to model their distribution. According to this the OC-SVM is a typical example of a model of normality, matching the third approach described at the beginning of section 4.

The OC-SVM tries to fit a tight hypersphere W to include most, but not all positive examples. If it attempted to fit all positive examples, this would lead to overfitting. In fact, the OC-SVM searches for the maximal margin hyperplane which separates the training data from the origin in the best way. It may be interpreted as a regular two-class SVM, where almost all the training data lies in the first class and the origin is the only member of the second class.

If the one class the modeller is interested in is considered as the regular data, resulting from normality, the negative examples detected by the OC-SVM can be considered as outliers of a different nature resulting from anomaly. This makes the OC-SVM an effective outlier detection tool.

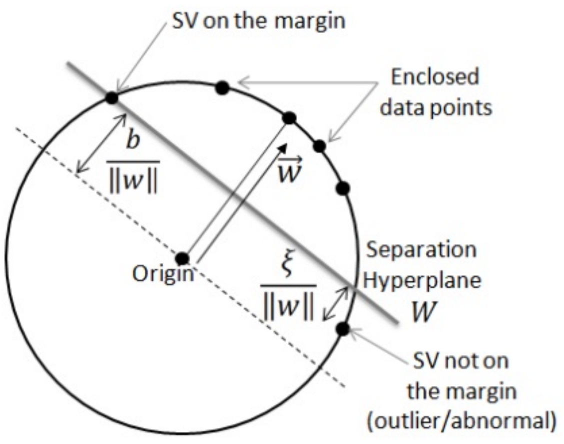


Figure 18: Illustration of the basic idea of an OC-SVM model.

4.4.2 Algorithm

Let $\{x_1, \ldots, x_n\}, x_i \in X \subseteq \mathbb{R}^m$ be the training set of $n \in \mathbb{N}$ observations that belong to a single class. The OC-SVM aims to define the minimum volume region enclosing $(1-\nu)n$ observations. The parameter $\nu \in [0,1]$ thus controls the fraction of observations that are allowed to be outliers. K shall be a kernel with a mapping function φ as defined in definition 49. Like for the general SVM ξ_i shall be the slack variables, non-zero slack variables correspond to the tolerated outliers.

Remark. It is essential not to use an OC-SVM model with ν extremely close to 0, because for a very small ν the OC-SVM is not able to find a good representation of the data distribution.

Algorithm 20 (One-Class SVM)**.**

$$\min_{w,\xi,b} \frac{1}{2}\|w\|^2 - b + \frac{1}{\nu n}\sum_{i=1}^{n} \xi_i \tag{86a}$$

$$\text{s.t. } w^T\varphi(x_i) - b \geq \xi_i \geq 0, \quad i = 1, \ldots, n \tag{86b}$$

Again, solving the OC-SVM optimisation problem is equivalent to a dual quadratic programming problem with Lagrangian multipliers α_i that can be solved with standard methods:

$$\max_{\alpha_i} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j K(x_i, x_j) \tag{87a}$$

$$\text{s.t.} \quad \sum_{i=1}^{n}\alpha_i = 1 \quad \text{and} \quad 0 \leq \alpha_i \leq \frac{1}{\nu n} \tag{87b}$$

Those patterns with corresponding $\alpha_i > 0$ are the support vectors. By using the KKT conditions from theorem 5, w and b can be obtained as:

$$w = \sum_{i=1}^{n}\alpha_i x_i \tag{88a}$$

$$b = \sum_{i=1}^{n}\alpha_i x_i^T x_j \tag{88b}$$

for any x_j with $0 < \alpha_j < \frac{1}{\nu n}$. For new data x that shall be checked by the OC-SVM the *decision function*

$$f(x) = \operatorname{sgn}(w^T\varphi(x) - b) \tag{89}$$

will be positive for positive examples and negative for outliers. Introducing D_0 as the decision and H_0 as the hypothesis that some observation x belongs to the normal class - the ambience - and D_1 as the decision and H_1 as the hypothesis that x is an abnormal event, the decision function can be formed to

$$\begin{cases} \text{if } f(x) \geq 0, & \text{then } D_0 \\ \text{if } f(x) < 0, & \text{then } D_1. \end{cases} \tag{90}$$

Adapting the Decision Function. It is easily possible to define a family of decision rules introducing a threshold $\gamma \in \mathbb{R}$:

$$\begin{cases} \text{if } f(x) \geq \gamma, & \text{then } D_0 \\ \text{if } f(x) < \gamma, & \text{then } D_1. \end{cases} \tag{91}$$

This formulation allows controlling the trade-off between the probability to miss outliers $P(D_0|H_1)$ and the probability to falsely declare an observation an outlier $P(D_1|H_0)$. A good threshold γ might be determined experimentally by operational requirements, which can vary a lot. Often missed outliers are not tolerable to the user of an abnormal event detection software, especially if the field of application is the security sector or something similar. Mathematically spoken γ controls a translation of the separating hypersphere W in the feature space F and therefore the resulting contour of the decision function in the representation space. In [19], Sébastien Lecomte et al. suggest to select an optimal value of ν that is only driven by the quality of the input signal, estimating the number of potential outliers, at the beginning and then to try to raise the performance according to operational requirements using a suitable threshold γ.

4.5 Using Temporal Integration to Detect Abnormal Events

All the above sections have left out the fact that in most cases an event is not determined by a single observation. Of course, an outlying observation or especially a significantly increased density of outliers can indicate an abnormal event. And in some cases simple rules for determining events are well known, e.g. if the temperature inside a computer or a boiler goes above a certain threshold, an alarm occurs and the computer/boiler is switched off. However, if the system is more complex or less deterministic, a more systematic approach is required for processing the raw sensor data to generate an event sequence.

How many observations belong to a certain event directly depends on the sampling rate $f_{sampling}$ and the duration M of the event, and therefore it is highly reasonable to include M as a parameter into an abnormal event detection model.

Most of the presented anomaly detection algorithm return an outlier score. It is easy to construct a decision function $f(x)$ from these outlier scores in a heuristic and application orientated way. Assuming M is the same and constant duration for all events, this decision function for single observations can be median-filtered to assign an outlier score to events during which M observations have been recorded:

Definition 56 (Median-Filtered Decision Function). $\mathcal{M}_M(x_k)$ shall be the operator returning the median value of the series $\{x_{k-M+1}, \ldots, x_k\}$, $f(x)$ shall be a decision function and γ the decision threshold as it is introduced in (91). Then a median-filtered decision function is defined as:

$$\begin{cases} \text{if } \mathcal{M}_M(f(x)) \geq \gamma, & \text{then } D_0 \\ \text{if } \mathcal{M}_M(f(x)) < \gamma, & \text{then } D_1. \end{cases} \tag{92}$$

Remark. The median-filtered decision function checks the data like a sliding window.

Remark. Of course other filtered decision functions can be designed, if other quantiles seem more reasonable in an abnormal event detection model.

This filter provides a good performance, if M is a constant and identified correctly. [19] If events are of different length in the field of application the model is used for, possible further adaptations of the model are presented below.

There are different ways to segment the raw data to an event sequence. In [19], Lecomte suggests to use a buffer. If it is full, the segmentation process starts. Based on a predefined similarity criterion, the closest successive pairs of segments are iteratively merged until only one segment remains, starting with single observations. This is a bottom-up hierarchical-clustering-based merging process. The resulting structure can be represented as a dendrogram. In this dendrogram it is possible to look for the best segmentation level. Lecomte et al. computed the intra-segment correlation coefficient and chose the segmentation level which provides correlation coefficients above a given threshold for all segments (e.g., 0.98). The segmentation at this level is kept except for the last segment that contains the first frames of the next buffer segment. After the segmentation process, the decision statistic is integrated over each homogeneous segment. The final decision function is defined as follows:

$$\begin{cases} \text{if } \langle f(x) \rangle_{S_j} \geq \gamma, & \text{then } D_0 \\ \text{if } \langle f(x) \rangle_{S_j} < \gamma, & \text{then } D_1, \end{cases} \tag{93}$$

where $\langle y \rangle$ is the operator returning the average value of the series $\{x_i\}$ $\forall i \in S_j$ and S_j is the set of frame indices belonging to segment j.

Algorithms to Detect Change Points. A more conventional approach to segment a time series to a sequence of events can be found in [28], which introduces some adaptations to the statistical *change point detection problem*: The problem investigated is identifying the time points at which the behaviour change of the system occurs. Two algorithms are presented by the authors Valery Guralnik and Jaideep Srivastava: The *batch algorithm*, which needs the entire data set to be available before the analysis, and the *incremental algorithm*.

The key idea of one iteration of Guralnik's and Srivastava's algorithm is to examine each segment and to decide whether it can be split into two significantly different segments or not. This decision is based on the maximum likelihood function L. The batch algorithm determines the index j that minimises $L(1, j) + L(j + 1, n)$, the sum of the maximum likelihood functions for the models for $\{x_1, \ldots, x_j\}$ and $\{x_{j+1}, \ldots, x_n\}$. Of course the range of j depends on p, the least points needed for model fitting in each segment. Such models could be VSARIMA models, for example. Thereafter, each of the two segments is analysed in the next iteration step and the best candidate change points c_1 and c_2 of each segment are located. Only the better of these two candidates is selected, yielding a division of the original sequence into three segments. Gaining one additional segment per iteration, the algorithm should stop when the likelihood criterion becomes stable or starts to increase. Formally, this stopping condition is

$$\frac{L_k - L_{k+1}}{L_k} < \varepsilon, \tag{94}$$

where ε is a user-defined stability threshold and L_k is the likelihood criterion value of the iteration k. When the stability threshold ε is set to 0, the algorithm stops exactly when the likelihood criterion starts to increase.

As mentioned before, the batch algorithm is only useful, if the data collection precedes the data analysis, which is not always the case: for example, when a system is controlled in an on-line way (Server outage prediction certainly requires an on-line system analysis.). For this kind of task Guralnik's and Srivastava's incremental algorithm can be used. Its key idea is that if the next observation collected by the sensor reflects a significant change of the system, then its likelihood criterion L_{change} of being a change point is going to be smaller than the likelihood criterion L_{no_change} that it is not. Small likelihood differences may be caused by the noise of the data, wherefore the criterion for the detection of a change point is

$$\frac{L_{no\_change} - L_{change}}{L_{no\_change}} < \varepsilon, \tag{95}$$

where ε is a user-defined likelihood increase threshold.

Assuming that the last change point was t_{k-1}, the algorithm starts with simply collecting enough data to build an adequate regression model. If the local regression model is available, candidate change points are those t_j that minimise the likelihood criterion

$$L_{min}(k,i) = \min_{k<j\leq i} L(k,j) + L(j+1,i). \tag{96}$$

If $L_{min}(k,i)$ is significantly smaller than $L(k,i)$, t_j is considered a changing point. Because computations can become extremely expensive, if no change point has been detected for quite a long time, it seems reasonable to use a sliding window technique instead of taking into account all the data including older observations to detect a new change point.

4.6 Event Classification

If an abnormal event has been detected by one of the methods presented, one could further investigate which kind of abnormal event the detected anomaly is. This classification of abnormal events requires at least a decent amount of abnormal events, so that the modeller is able to use one of the presented classification or clustering methods like an ANN, an SVM or some clustering algorithms. This makes the task of event classification almost impossible at the beginning of most projects, but of course it is very useful to store the data belonging to abnormal events so that these can be classified at a later date at which this data may be analysed and labelled by experts. If this is the case, the classification can be done in a supervised way.

If a detection method like a replicator ANN that already provides the modeller with a cluster topology is used, further investigations on these clusters seem useful for the event classification task.

5 Results of a Simulation to Detect/Predict Server Problems

5.1 Problem Statement

Panagenda develops software for IBM Lotus Notes and Domino. The company is located in Vienna and Heppenheim near Frankfurt am Main. Over 4 million licensed module users in over 70 countries use their software to reduce their operating costs. With clients ranging from 10 to 140000, many of them operate in highly secure environments such as financial institutions and need their servers to run in a consistent and secure way with as few outages as possible.

It is easy to gather tons of data of a server running, but it is hard to tell what the data reveals about the status of the server, whether an outage could happen within a short time or the server runs as the clients want it to run.

Panagenda intends to develop a software tool that detects possible server outages as early as possible in order to allow their clients a more comfortable operation of the server. Of course, it is of high interest not only to know when a certain problem is going to cause an outage, but also to know why this outage might happen. In the end, fewer server outages save everybody's money.

It has to be stated that it is almost impossible to precisely define the term *server outage*, wherefore a definition is not given in this thesis. Any limitation to the normal operation of a server is unwanted. Many times only a certain kind of tasks is delayed or cannot be executed at all. The severity of this limitation also depends on the fact whether users can carry out other tasks in the mean time. The only possibilities to give the modeller an idea about the severity of an outage are the total downtime minutes or the downtime minutes per user.

It has been proposed to combine several of the mathematical models described in the earlier sections to achieve the goal of server outage prediction. How these partial models and algorithms can be combined to a whole model is presented in the following sections. It has to be noted that Panagenda did not release its software up to the day that this thesis was published; but they had run several tests. Thus it has not been possible to fully evaluate the whole model in a daily-life-scenario up to now. It also seems logical that not every server behaves in the same way. These are the two main reasons why it is intended to make the whole model adjustable via a few tuning parameters that allow the users of the software also to control the sensitivity of outage alarms. The role of these model tuning parameters is discussed in section 5.5.

5.2 Software Environment and Data Generation

Though the expensiveness of most calculations needed for the simulation runs of the proposed models is an important factor, it was possible to use a standard personal computer for most calculations due to the fact that all the data was labelled with priorities and only the most influencing data was taken into account for the simulation runs (See section 5.3 for further information.).

The personal computer used was equipped with an AMD Athlon 64 X2 Dual Core Processor 4200+ with 2.21GHz, 3GB RAM and Microsoft Windows 7. Most of the calculations and most of the programming were done using MATLAB R2011b, but also the statistical software R (version 2.15.3) was used as well as ggobi (version 2.1.10) and Mondrian (version 1.2) as data analysis visualisation tools. Especially Mondrian is able to produce scatter plots, bar charts and histograms of huge data sets very quickly.

For the simulation runs done in MATLAB the following four additional toolboxes were used: the *Neural Network Toolbox* and the *System Identification Toolbox*, both developed by MathWorks, the *OpenTSTool* and the *SVM and Kernel Methods Toolbox* developed by S. Canu, Y. Grandvalet, V. Guigue and A. Rakotomamonjy from Rouen, France. The latter is fully written in MATLAB, even the QP solver. The last update of the *SVM and Kernel Methods Toolbox* used for the simulation runs for this thesis was on 20.02.2008. Due to the newer MATLAB version used some small adaptations in the code of the toolbox were necessary. The key features of the toolbox are:

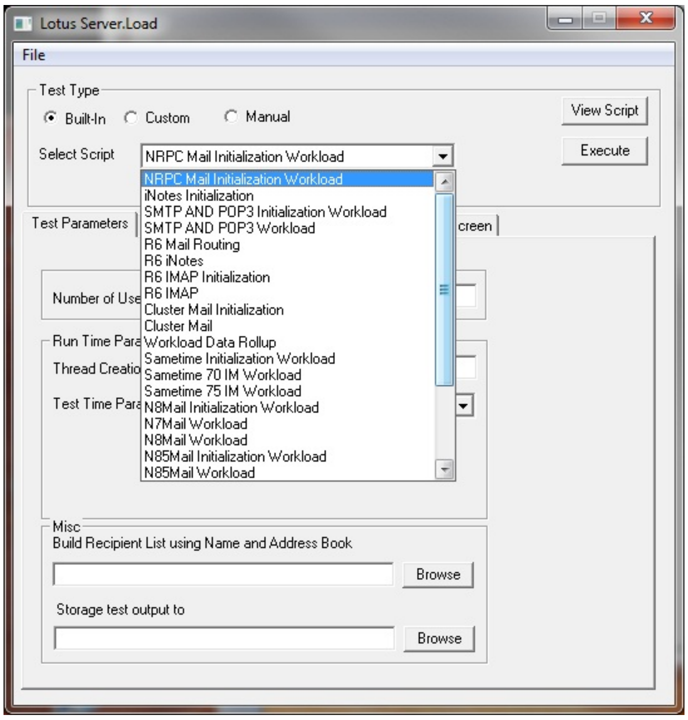
- SVM classification using linear and quadratic penalisation of misclassified examples (penalisation coefficients can be different for each example)
- SVM classification with the nearest point algorithm
- multiclass SVM
- large scale SVM classification/regression
- SVM epsilon and nu regression
- one-class SVM

- regularisation networks
- SVM bounds (span estimate, radius/margin)
- wavelet kernel
- SVM based feature selection
- kernel PCA
- kernel discriminant analysis
- SVM based feature selection
- SVM AUC optimization (ranking SVM, ROC SVM) and RankBoost
- kernel basis pursuit and least angle regression (LARS) algorithm
- wavelet kernel regression with backfitting
- interface with a version of libsvm.

The OpenTSTool, a toolbox especially designed for non-linear time-series analysis, is partly written in Matlab and partly written in C++, using as advantages of Matlab the reduced development time, the extensive collection of intrinsic mathematical functions, the excellent graphical capabilities as well as the high portability from Unix to other platforms and as advantages of C++ the better performance of computationally demanding algorithms. The C++ code is saved in .mex files. The version 1.2, developed by Christian Merkwirth, Ulrich Parlitz, Immo Wedekind, David Engster and Werner Lauterborn, was used.

All the algorithms needed for the outage detection model were tested within this mathematical software environment. A prototype of the outage detection software was written in Java; it also used several already existing Java toolboxes.

Besides historical data, Panagenda also used the additional software tool *IBM Lotus Domino Server.Load V8* to simulate several different servers operating in order to provide the dwh GmbH with the data needed for the simulation test runs (See figures 19 and 20.). Server.Load is a capacity-planning tool that is used to run tests, also called *scripts* and *workloads*, against a targeted IBM Lotus Domino server to measure its server capacity and response metrics. Each client running Server.Load generates a simulated user load of IBM Lotus Notes transactions against the server under test, which reports server statistics back to the client. There are several built-in scripts, but


Test Type: [✓ Built-In] Custom Manual

Select Script: NRPC Mail Initialization Workload

Figure 19: Screenshot of IBM Lotus Domino Server.Load - Script selection.

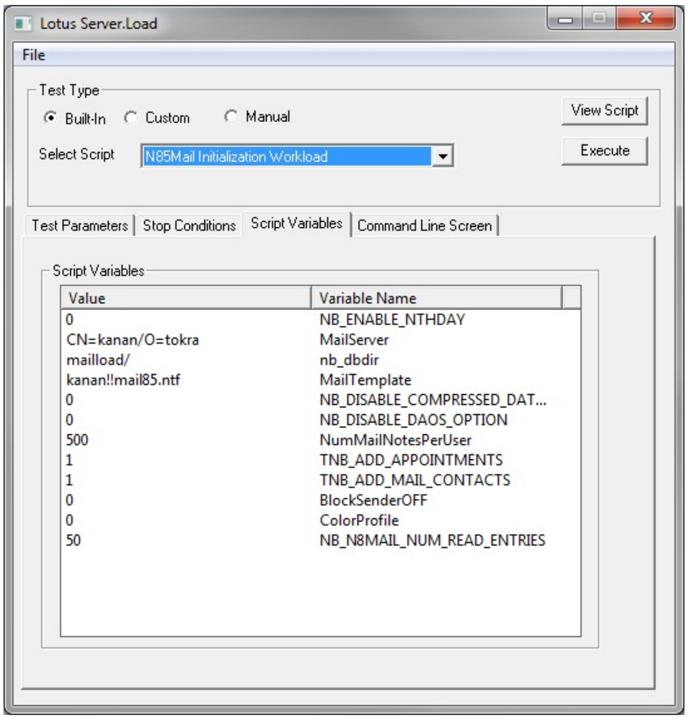
also custom scripts can be created. The users of Server.Load have real-time control of the test environment and variables.

The parameters were measured approximately every minute or approximately every 15 minutes. The two sampling rates were chosen to compare the respective results. Measuring the parameters every minute and especially analysing the data can certainly influence the performance of the server. Of course the goal is to lose as less computing capacity as possible due to the gathering and the analysis of the server data.

5.3 Feature Selection and Data Preprocessing

The central assumption when using a feature selection technique is that large data sets contain many redundant features that do not provide mode information than the previously selected ones or irrelevant features that do not provide any useful information in any context. Feature selection is supposed to

- improve the interpretability of the model



Test Type: [✓ Built-In] Custom Manual

Select Script: N85Mail Initialization Workload

Script Variables

Value	Variable Name
0	NB_ENABLE_NTHDAY
CN=kanan/O=tokra	MailServer
mailload/	nb_dbdir
kanan!!mail85.ntf	MailTemplate
0	NB_DISABLE_COMPRESSED_DAT...
0	NB_DISABLE_DAOS_OPTION
500	NumMailNotesPerUser
1	TNB_ADD_APPOINTMENTS
1	TNB_ADD_MAIL_CONTACTS
0	BlockSenderOFF
0	ColorProfile
50	NB_N8MAIL_NUM_READ_ENTRIES

Figure 20: Screenshot of IBM Lotus Domino Server.Load - Some script variables.

- shorten training times
- enhance the generalisation ability of the model by reducing overfitting.

Though a broad field of scientific research interest, for the simulation carried out within the scope of this thesis, automatic feature selection was not a main issue, mainly because of the fact that an exact evaluation of most feature selection methods is not possible at this early project stage. Feature selection was done in a manual way, categorising all the variables observed into four main classes of priorities. This classification in categories was done by experts from Panagenda who have maintained servers and observed their outages for a long time and therefore have the essential expert knowledge to do this categorisation. Variables labelled with priority 0 are thought to have the largest impact on server outages, variables labelled with priority 1 are thought to also play an important role for the functionality of the servers the data is recorded from. In contrary, the experts think that those variables they labelled with priority 3 do absolutely scarcely reflect changes in the behaviour of the servers. As the total of up to 1439 variables recorded for a single server is such a huge amount of data many algorithms cannot deal with within an acceptable running time, most of the sub-models were optimised for the data set $\mathcal{D}_1$ with priorities 0 and 1 or for the data set $\mathcal{D}_0$ with priority 0 variables only. Whenever possible, the results of the simulations with $\mathcal{D}_0$ and $\mathcal{D}_1$ were compared. Some variables cannot be observed for all servers (For example, if a server has only two hard disks, it is not possible to measure the percentage of the utilisable space of the third, forth or fifth disk.), thus $\mathcal{D}_0$ and $\mathcal{D}_1$ do not contain the same number of variables for all servers. To get an idea about the dimension of the data the simulations have to deal with, it can be stated that

$$|\mathcal{D}_0| \leq 14, \qquad |\mathcal{D}_1| \leq 73. \tag{97}$$

If a more highly sophisticated feature selection method is desired, the *Pearson correlation coefficient* and the *Spearman's rank correlation coefficient* could be used to cluster the variables into groups or a PCA could be done.

Besides the feature selection task the simulations to detect server outages have another sine qua non: Data preprocessing is indispensable, many algorithms used would be confronted with problems if all the variables were left in their generic ranges, which differ a lot. Some variables are measured in milliseconds or bytes and take on values in the region of 10^{11}, other variables take on values between 0 and 0.4 (e.g. *Platform.System.PctTotalPrivilegedCpuUtil*) and the variable *greenlight.time.s2c.difference* is always negative.

Accumulated Values. Some of the variables accumulate values and it seems more reasonable to use the differences of two consecutive observations, because the differences describe what happened within the system and not the status quo as the original variable did. Thus all accumulated variables were transformed to their differences:

$$x_t \mapsto z_t = \nabla x_t = (1 - B)x_t, \tag{98}$$

using the notations (9) and (17).

Example 18. In table 3 the 14 variables of $\mathcal{D}_0$ are listed together with their ranges for a certain server test run. The variables of $\mathcal{D}_1$ can be found in the appendix section B.2.

Variable	Range	
api_statistics_status	0	2051
Database.Database.BufferPool.PerCentReadsInBuffer	99.33	99.39
Database.DbCache.OvercrowdingRejections	0	0
Mail.Mailbox.AccessConflicts	0	4
Platform.LogicalDisk.1.PctUtil[23]	0	$1.8447 \cdot 10^{17}$
Platform.LogicalDisk.2.PctUtil	NaN	NaN
Platform.LogicalDisk.3.PctUtil	NaN	NaN
Platform.LogicalDisk.4.PctUtil	NaN	NaN
Platform.LogicalDisk.5.PctUtil	NaN	NaN
Platform.Memory.RAM.PctUtil	67	99
Platform.System.PctTotalPrivilegedCpuUtil	0.01	18.52
Platform.System.PctTotalUserCpuUtil	0.01	10.65
Server.AvailabilityIndex	0	100
Server.ConcurrentTasks	0	11

Table 3: Variables of $\mathcal{D}_0$ with their ranges.

The server this data was recorded from has only one disk and is probably not a mail server, because the total number of mailbox access conflicts does not exceed 4.

Example 19. *Mail.Mailbox.AccessConflicts* counts the total number of mailbox access conflicts since the start of the observation of the server. It is of greater interest to detect whether there have been a lot of access conflicts in the last few minutes (E.g., this could indicate an event like an intrusion attempt.) than to know the exact number of total access conflicts.

Example 20. *Server.AvailabilityIndex* is 0, if there is no obvious dysfunction of the server. It takes values like 2051, if a dysfunction has been detected. Thus *Server.AvailabilityIndex* cannot be seen as a continuous variable, it should be considered a categorical variable which is almost impossible to predict. That is why it has not been attempted to predict this variable. It has to be stated that this variable does not cover all dysfunctions that can cause an outage and sometimes indicates minor dysfunctions which do not influence the over-all status of the server at all. It seems reasonable to simply slightly increase the outlierness of an observation with *Server.AvailabilityIndex* not equalling 0, no matter what exact value the variable takes on.

[23]If all outliers obviously caused by a bad sensor or measurement are deleted, the upper boundary of the range is only 5.93. See example 21 for further details.

Outliers and NaNs. Another part of data preprocessing included scanning the data for outliers obviously caused by a bad sensor or measurement and for NaNs. In order to not confuse the outlier detection task to determine abnormal events with this outlier detection, an example is given:

Example 21. *Platform.LogicalDisk.1.AvgQueueLen* is a variable that measures the average queue length for tasks that should be executed on the first hard disk of a server. For a specific data set of a server the range of *Platform.LogicalDisk.1.AvgQueueLen* is

$$[0, 5.93] \cup \{2.6352 \cdot 10^{10}\} \cup \{3.0745 \cdot 10^{10}\} \cup \{1.8447 \cdot 10^{17}\} \tag{99}$$

It can be assumed that the queue is rather long, when the variable takes on one of these extremely large values, or that maybe the sensor does not work properly. But it is obvious that the exact value that is taken is of no interest. It has probably something to do with the computational accuracy and does not reflect the true queue length. Outliers such as these were treated like NaNs in all the further simulation runs or replaced by the preceding value.

Another question during the data preprocessing that had to be dealt with was the role of NaN values. If they originate from an abnormal event, they are of high interest. If they originate from a bad sensor, this raises the question whether any data obtained from this sensor can be trusted for further investigations. Apart from the difficulties most algorithms used are faced with when they have to process NaNs, it was decided that all variables which contained more than a certain percentage of NaNs should be deleted during the data preprocessing. This threshold was initially set to 60%. For the remaining NaNs it can be stated that an accumulation of NaNs for many variables at a certain time can certainly indicate an abnormal event. An analysis of when how many sensors return an NaN should therefore always be done besides all the other approaches used in these simulation runs.

Adapting the Variable Ranges. To gain appropriate variable ranges, several transformations were considered, but two transformations introduced below seemed to deliver the best results. The first transformation is called *Z-score* due to the fact that the normal distribution is also called Z-distribution:

Algorithm 21 (Z-Score)**.**

$$f_{zscore} : \mathbb{R} \to \mathbb{R},\ x \mapsto z = \frac{x - \mu}{\sigma} \tag{100}$$

The quantity z represents the distance between the raw score and the data's mean μ in units of the standard deviation σ. z is negative when the raw score

is below the mean, positive when above. As in most cases the true standard deviation is not known, it has to be estimated. Especially if there are no outliers in the given data, but as outliers are expected to occur in the future, the quality of this estimation could be rather bad.
Of course a Z-score can be defined without assumptions of normality.

Algorithm 22 (Minmax-Mapping)**.** Given a set $\{x_1, \ldots, x_n\}$, the following mapping transforms this set to a new set $\{y_1, \ldots, y_n\}$ in the interval $[y_{min}, y_{max}]$ in an affine way.

$$f_{minmax} : [x_{min}, x_{max}] \rightarrow [y_{min}, y_{max}], \\ x \mapsto y = \frac{(y_{max} - y_{min})(x - x_{min})}{x_{max} - x_{min}} + y_{min} \qquad (101)$$

Remark. The parameters y_{min} and y_{max} have to be chosen in a manner adequate for the further methods applied, for example -0.6 and 0.6 for the input vectors of an ANN.

Remark. Because f_{minmax} is an affine transformation the parameters of which are chosen in the preprocessing phase of a training and remain fix, new arriving and transformed data does not necessarily have to be an element of the interval $[y_{min}, y_{max}]$!

5.4 Applied Methods

Basically, the applied methods of the proposed server outage prediction model belong to one of the following two sub-tasks:

a) predicting the next observations

b) deciding whether the deviation of the prediction and the actual value indicates an abnormal event.

General Model Assumption. The predictors work very well, if the status of the server is good, i.e., if the system works in a normal way. At least at the beginning of outage-like problems these predictions, i.e., if the status of the system becomes abnormal, the predictions become notably worse and originate from a different distribution.

5.4.1 Predictor

For the above stated sub-task a) two different approaches were used:

a.1) a neuro-predictor

a.2) an ARIMA model.

Univariate/Multivariate Approach. Both approaches were tested in a univariate and a multivariate way. The multivariate way is for both approaches computationally very demanding and restricted the number of variables predicted on the standard PC used to marginally fewer than 10. Apart from this problem many of the variables are not correlated and do not have any causal dependencies, wherefore it is not reasonable to predict multivariate vectors of a large size. All further simulation runs were modelled in a univariate way, using a predictor of its own for every feature. Nevertheless, all the features were categorised into groups of possible causal relations, e.g. one group containing all the variables that have something to do with mails. If a multivariate approach is considered, it only seems auspicious to use one predictor per group and not an overall predictor. The groups are presented in table 4. Every feature can belong to more than one group.

Group	Number of Variables in $\mathcal{D}_0$	Number of Variables in $\mathcal{D}_1$
Agents	0	9
Cache	1	9
Cluster	0	5
CPU	2	3
Database	2	7
Disk	5	0
Full Text	0	3
http	0	2
Indexer	0	2
Mail	1	10
Network	0	4
RAM	1	2
Replicator	0	5
Resources	8	8
Server	3	9
Tasks	0	16
Transactions	0	2
Users	0	3

Table 4: Variable groups of the Panagenda data set.

While an ARIMA model needs a training data set to estimate its parameters before it can be used to predict future values (See section 2.2 for further details.), a neuro-predictor needs a classical neural network training to adjust the weights of the ANN before it can be used for the prediction task. Of course this will have to be done for every server after the server outage prediction software has been installed.

Example 22. Like in example 10, the gas prices time series is used for demonstration purposes in this example. This time the oil prices were considered as well; the prediction was done in a bivariate way. Figure 21 shows the untransformed time series and the predictions as well as the prediction errors below. Accumulation of larger prediction errors could indicate abnormal events. Further details to this example are discussed in example 25.

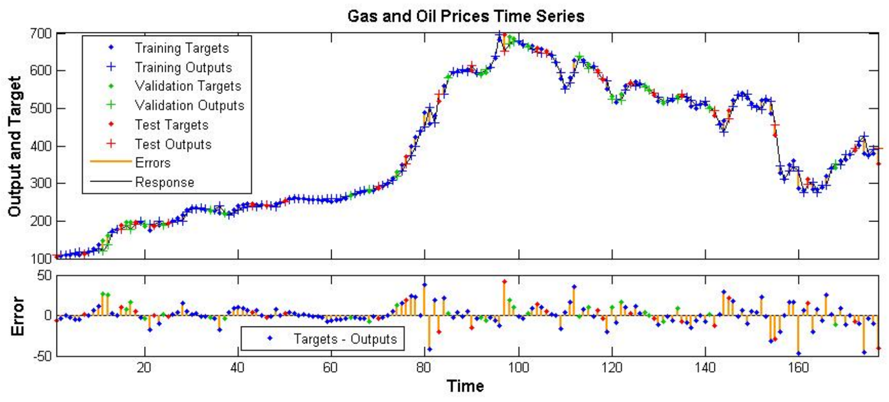
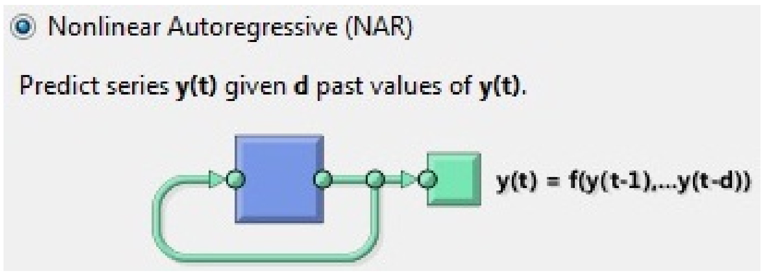


Figure 21: Gas prices time series and neuro-prediction errors using an ANN with 10 hidden neurons and a delay of 3 months.

Neuro-Predictor. The ANNs used as neuro-predictors require pre-labelled data. This is simply done by preprocessing the data into an adequate block structure. The block size depends directly on the tuning parameter *argBlockSize* that defines the number of rows of the block. The number of columns of the block is m, the number of those features that are taken into account for the simulation. Thus one column contains the *argBlockSize - 1* inputs for one of the m neuro-predictors as well as the latest observation, which is the target value for the neuro-predictors. As *argBlockSize* will be chosen rather low in general (probably below 10, but certainly below 100), it seems promising to adapt the blocks in the following way: The rows should not only contain the *argBlockSize - 1* former observation values in order to predict the actual one, but they should contain at least some observation values that were measured exactly a week before.[24] The sampling rate of some minutes is too high and the possible number of input neurons of the neuro-predictors must be too low to implicitly take into account a possible (and most likely) weekly seasonality. Thus the weekly seasonality can be explicitly modelled by preparing adequate blocks during the data preprocessing. *argBlockSize* could then become a vector, the sum of the entries minus 1 defining the number of input neurons and the entries defining how many observations of which period should be considered for the prediction.

Figure 22 shows how the neuro-predictor was modelled with the MATLAB Neural Network Toolbox. *d* in the figure equals *argBlockSize - 1*.


Nonlinear Autoregressive (NAR)

Predict series **y(t)** given **d** past values of **y(t)**.

Figure 22: Illustration of the non-linear autoregressive neuro-predictor model using the MATLAB Neural Network Toolbox.

Changing the tuning parameter *argBlockSize* requires a complete restart of the training and should therefore not be done unless it seems really necessary.

Before being able to begin the training, some further preprocessing steps need to be taken. Many algorithms used cannot deal with NaNs, so they have to be deleted within the data preprocessing. First of all, variables with more than *argPercentNanColumn* percent NaN entries are totally deleted. It is assumed that the sensor does not work properly and that the (few) observations cannot be trusted. Apart from that, if theses variables were not deleted, many predictions would be impossible to make due to the large amount of NaNs remaining in the data set. Although this sounds a little bit paradox, deleting variables can delimit the necessity of deleting too many data blocks.

After this column-wise scanning for NaNs, the rows are checked for NaNs. If an NaN is located, the whole blocks the NaN entry belongs to are deleted.

To improve the performance of the ANNs especially at the beginning of the training, algorithm 22 is used to map each variable to the range

[24]This follows the idea of example 4 in section 2.1.4.

$[-0.6, 0.6]$ in an affine way. Changing the weights (i.e., the learning process) occurs faster this way due to the ranges of the activation functions used. All the ANNs are designed in the same way: The input layer consists of *argBlockSize - 1* nodes, the output layer only has one neuron and the one hidden layer shall use $\lceil$*sqrt*(*argBlockSize - 1*)$\rceil$ neurons, where $\lceil\cdot\rceil$ is the ceiling function. All the nets were also tested with 10 hidden neurons, which worked as well as the choice of $\lceil$*sqrt*(*argBlockSize - 1*)$\rceil$ hidden neurons. The latter is the more generic choice and for most tuning parameters *argBlockSize* $\lceil$*sqrt*(*argBlockSize - 1*)$\rceil$ is lower than 10, which means less computing time. As shown in figure 23, the activation function between the input and the hidden layer is the hyperbolic tangent and the activation function between the hidden and the output layer is simply the linear activation function. The training algorithm used is the Levenberg-Marquardt algorithm 10 introduced in section 3.4.4.

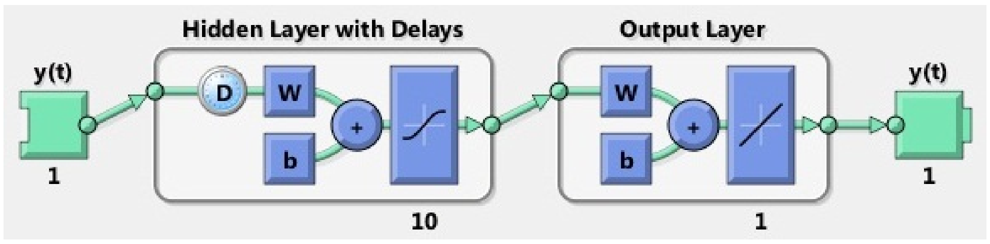


Figure 23: A detailed look at the artificial neural network used to predict the next observation using the MATLAB Neural Network Toolbox.

Within the final step of the data preprocessing, the set of data blocks is divided into a training data set (70%), a validation data set (15%) and a test data set (15%). Only the training data set influences the change of the ANN weights.

The training sets are presented to the ANNs in several epochs. The supervised learning stops as soon as one of the following three break conditions is met:

1. The number of training epochs exceeds the value of the tuning parameter *setAbortNumberOfEpoch*.
2. The number of back-to-back epochs, which the error function of the validation set increases in, exceeds the value of the tuning parameter *setAbortRisingErrors*.
3. The error value of the test data set falls below the minimal error value, which was set to 10^{-6}.

When all the trainings have terminated, the ANNs can be used as neuro-predictors. Further investigation should be done on the issue of detecting time points, when the ANNs should be retrained with up-to-date data.

Example 23. The following figure shows the time series of the feature *Platform.System.PctTotalPrivilegedCpuUtil* of a certain server test run and its predictions by a neuro-predictor with 10 hidden neurons. To get a better idea of the local quality of the predictions, also the differences between *Platform.System.PctTotalPrivilegedCpuUtil* and its predictions are shown in figure 25.

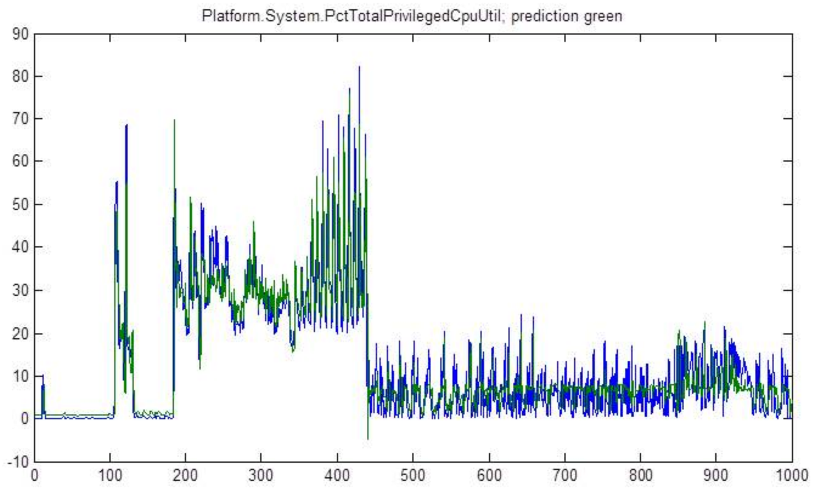


Figure 24: *Platform.System.PctTotalPrivilegedCpuUtil* and the predictions of *Platform.System.PctTotalPrivilegedCpuUtil* for a certain server test run.

ARIMA Predictor. Several simulation runs on Panagenda data sets did not indicate a notable difference between using an ARIMA model and the ANN. Thus predictions by ARIMA models look very much like figure 24. General considerations about which approach works better for which data are listed in section 3.4.5. It should be considered that for the time series of some features the neuro-predictor could be able to make more exact predictions, whereas for time series of other features ARIMA models might be the better choice due to the properties of the time series. Which model is able to predict more exactly for a certain feature can also be server-dependent. Nevertheless, simulation runs are easier if only one prediction method is chosen for all time series of all features.

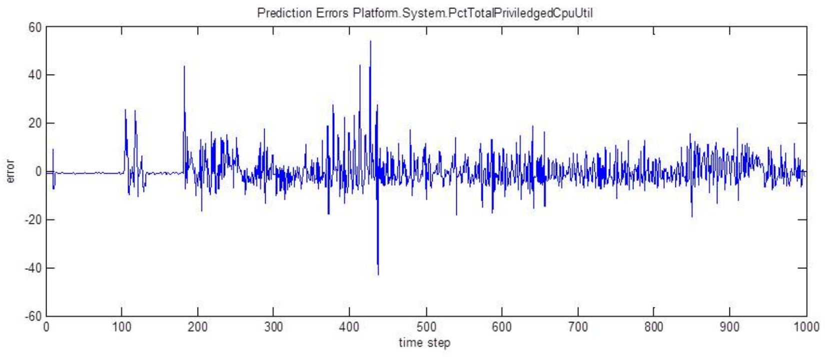


Figure 25: Differences between *Platform.System.PctTotalPrivilegedCpuUtil* and its predictions, which shall be analised by an anomaly detector.

All in all, the aim of both of the presented approaches is to predict the expected developing of the values as well as possible. These predictions are then compared with the next observations as soon as they are measured.

5.4.2 Anomaly Detector

Judging upon the differences between the predicted and the measured values, the linked anomaly detector has to decide if these differences underlie a small and tolerated prediction error or if these differences are the effect of an arising problem of the server that finally could cause an outage. While the prediction was done in univariate way, the anomaly detector has to deal with the vectors of the prediction differences of all variables, i.e., works in a multivariate way.

Example 24. As a univariate example, the following figure 26 shows a histogram of the differences between the predictions and the new observations of the variable *Platform.System.PctTotalPrivilegedCpuUtil*. This is more or less exactly the distribution that was expected for a single variable: a rather clear Gaussian bell and a few values rather far away from the centre, indicating large differences between the prediction and the new observation.

For the above stated sub-task b) the following three different approaches were used:

b.1) a threshold

b.2) ABOD

b.3) a OC-SVM.

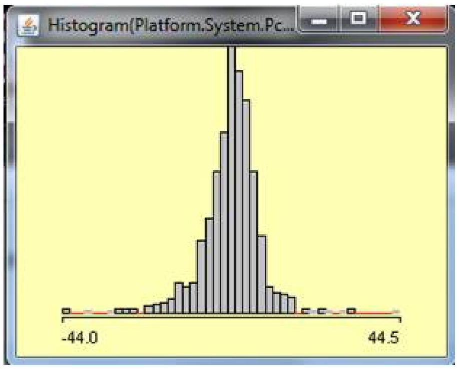


Figure 26: Prediction differences of the variable *Platform.System.PctTotalPrivilegedCpuUtil*, having used the neuro-predictor and Mondrian as visualisation tool. The prediction differences using the ARIMA predictor look extremely similar - see figure 27 below.

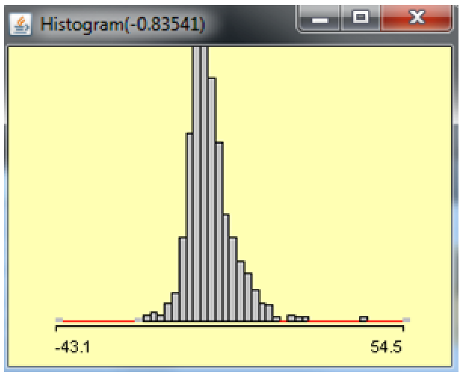


Figure 27: Prediction differences of the variable *Platform.System.PctTotalPrivilegedCpuUtil*, having used an ARIMA model and Mondrian as visualisation tool. Not only the distributions of the prediction differences look very much alike, the most extreme values also belong to the same or at least very close time steps.

All approaches are not able to deal with NaN values, wherefore they do not deliver any output if it was not possible to predict the values or if one of the actual observations is an NaN. For all three different approaches the sliding window technique can be applied to detect the accumulation of outliers and deflate single outliers.

Threshold. The most simple approach is the use of a threshold for any distance function applied on the actual feature values and its predictions; this threshold can be determined heuristically and adjusted as a tuning parameter.

Example 25. This example is a continuation of examples 22 and 10. A sliding window of length 6 was used for filtering the prediction error norm, following the idea of definition 56. The monthly time series starts in July 1973, the clear peak where the threshold is clearly passed for several months belongs therefore to the end of 1979 and the beginning of 1980, the period of the second oil crisis, when with the Iranian Revolution and the First Persian Gulf War two external events took place that had a massive influence upon the oil and gas prices and that were not predictable by any means within the rather simple prediction model.

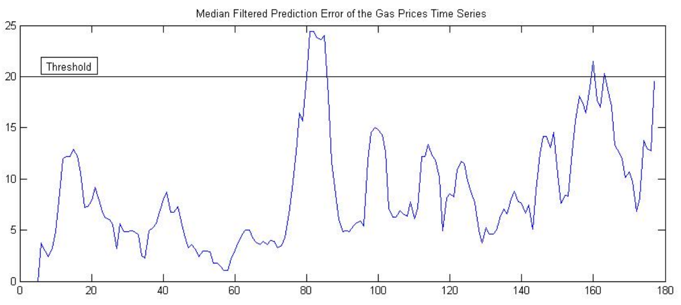


Figure 28: Median error of the neuro-predictions of the gas time series.

ABOD-Detector. Concerning the angle-based outlier detection, first the classic which was presented as algorithm 18 in section 4.2 was implemented. Via the tuning parameter *setWeightFunction*, also the adapted quicker algorithm 19 can be used. In the latter case every point A in space is compared only with those in N_k, those sets that include the k nearest neighbours of A; the neighbours of a point can be defined on grounds of spatial or spatiotemporal proximity - this has to be done by setting *addTimeWeight* adequately. The classic algorithm is rather slow, so if the amount of data which is the model input is huge, the adapted algorithm should be preferred.

For every point A of the multivariate time series the corresponding angle-based outlier factor $ABOF(A)$ is calculated. The lower this value is, the more likely it is that A belongs to an abnormal event. It can either be returned that the $ABOF$-value of the actual feature vector is among the lowest x values ever measured (x being the number the system administrator might be interested in) or the threshold value *setLimitPosition*. Feature vectors A with an $ABOF$ value below this threshold are detected as outliers. The degree of outlyingness can be defined as follows:

Definition 57 (Outlierness (ABOD))**.**

$$O_{ABOD}(x) = 1 - \frac{ABOF(A)}{setLimitPosition}. \tag{102}$$

Additionally, a sliding window technique with a sliding window of length *setWindowLength* can be applied.

Example 26. During several test runs, the anomaly detectors easily detected when the servers changed their status from idle to busy and vice versa. A graphical analysis of one of the test runs is shown in the following figure 29. The OC-SVM-detector worked as well as the ABOD-detector and therefore gave a very similar output on these simulation runs.

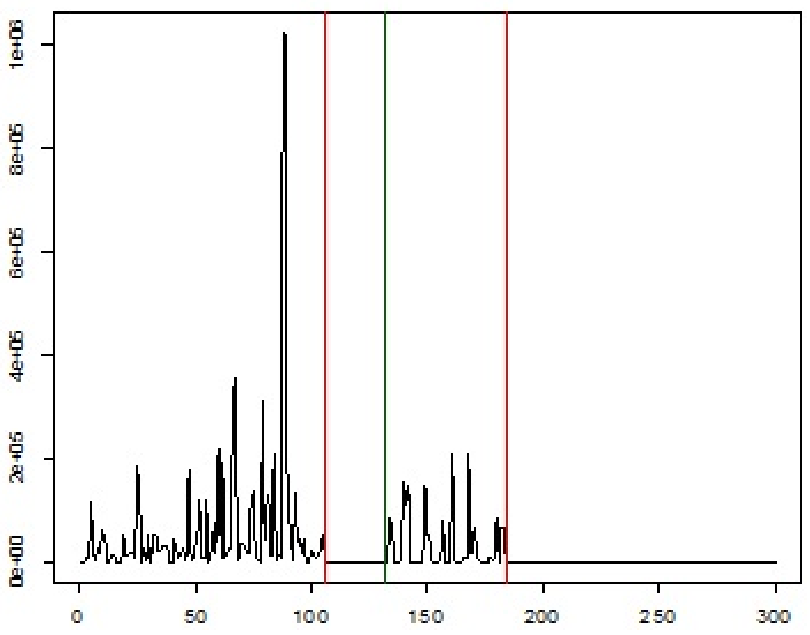


Figure 29: Angle-based outlier detector detecting the server change from idle to busy (green) and busy to idle (red), using one of the test data sets provided by Panagenda.

OC-SVM-Detector. As indicated in section 4.4, the one class support vector machine only models normal data, but also tolerates a few outliers. These outliers accord to a somehow restricted operation of a server. The input values of the OC-SVM are differences between the predictions and the actual values. Also the OC-SVM needs some training time to adjust the internal parameters. After this training the OC-SVM can be used to detect possible server outages.

First, the input data is mapped to the unit hypersphere of a feature space. This mapping is done by a Gaussian kernel (algorithm 14) with bandwidth *setGamma*. Then the OC-SVM calculates the hyperplane $f(x) = 0$ with the maximum distance from the origin that separates approximately *setNu* percent of the training data set - the inliers, which $f(x) > 0$ holds for - from approximately *100 - setNu* percent of the training data set. These are the outliers, which $f(x) < 0$ holds for. *setGamma* and *setNu* are two important tuning parameters. b being the bias of the hyperplane, now each prediction difference vector x or each sliding window (of the length *setWindowLength*) belonging to such an x can be assigned a value of outlierness $O_{OC-SVM}(x)$:

Definition 58 (Outlierness (OC-SVM))**.**

$$O_{OC-SVM}(x) = 100o(x) = 100\left|\frac{f(x)}{b}\right|, \quad \text{for } f(x) < 0. \tag{103}$$

Remark. The origin is the point with the maximum outlierness possible.

Remark. The range of $o(x)$ is $[0, 1]$, thus it can be interpreted as a fuzzy membership function. If it seems more adequate to the system administrator, $o(x)$ can be replaced by an $o^q(x)$ for $q \in \mathbb{R}$ that empirically fits the severeness of outages better than $o(x)$. This follows the idea of the fuzzy linguistic variables which are introduced at the end of section 3.3.

By evaluating the detector with many training data, a threshold can be also set to determine the severity of the anomalies. In [7], Zhang et al. also use the return value of the decision function to distinguish between minor and major outliers. This is very helpful for server administrators to help them prioritising their work.

After the training, the hyperplane remains fixed and the outlierness values of recently calculated prediction differences can be computed. Again, it is not a priori clear, when the hyperplane parameters need to be adjusted, i.e. when the OC-SVM needs to be retrained.

Following the idea presented in the paragraph *Adapting the Decision Function* in section 4.4, it may enhance the performance of the server outage detection model to separate the data set with a hyperplane $f(x) = setLambda$

and varying yet another tuning parameter *setLambda*. This basically establishes the possibility to control the trade-off between false and missed alarms.

5.5 List of the Most Important Tuning Parameters

Choosing a good parameter setting is of uttermost importance for the performance of the server outage detection model. Readapting the parameters might be necessary from time to time.

Parameter Name	Model Part	Type	Default Value
argData	ANN-Predictor	Array	None
argBlockSize	ANN-Predictor	Integer	None
argPercentNanColumn	ANN-Predictor	Integer	60
setAbortNumberOfEpoch	ANN-Predictor	Integer	1000
setAbortRisingErrors	ANN-Predictor	Integer	6
setWeightFunction	ABOD	Integer	0
setLimitPosition	ABOD	Double	2
addTimeWeight	ABOD	Array, Calender	None
setGamma	OC-SVM	Double	σ
setNu	OC-SVM	Double	0.99
setLambda	OC-SVM	Double	0
setWindowLength	OC-SVM, ABOD	Integer	0

Table 5: List of the tuning parameters for the server outage detection model.

5.5.1 ANN Tuning Parameters

Array *argData*: full, specifically sorted, but in other respects unedited data set of all features. The way *argData* is sorted is directly influenced by the seasonality modelled and also fully determines all seasonal dependencies within the model proposed. If the array is read column-wise, one gets the time series of each feature which are organised row-wise.

Integer *argBlockSize*: *argBlockSize* - 1 is the total number of relevant points in time that influence the prediction of the next time series element of every feature. *argBlockSize* determines the size of all blocks within the array *argData*, the last entry of a block being the respective target value for the predictor. Changing it makes a complete restart of all trainings necessary and is therefore only recommended, if the seasonal dependencies modelled by the initial choice of *argBlockSize* proved to be not sufficient for adequate predictions.

Integer *argPercentNanColumn*: If the ratio of NaN values per column exceeds the percentage *argPercentNanColumn*, the whole column is deleted from the array *argData*. Obviously this feature cannot be measured with adequate quality to precisely predict its future values. The default value of *argPercentNanColumn* is 60.

Integer *setAbortNumberOfEpoch*: This integer parameter represents the maximum number of training epochs for every ANN. In case it is set too small, the accuracy of the predictions might be not as sufficient as wanted; in case it is set too large, undesirable long computation times might occur. The default value of *setAbortNumberOfEpoch* is 1000.

Integer *setAbortRisingErrors*: This tuning parameter defines the maximum number of consecutive epochs, in which the error function of the validation set rises with respect to the prevenient epoch. Is *setAbortRisingErrors* set too small, the training might terminate too early, which can lead to bad predictions. If it is set too large, the generalisation ability of the ANN might not be guaranteed due to overfitting. The default value of *setAbortRisingErrors* is 6.

General Remarks: The two most performance affecting tuning parameters of the above described are *argBlockSize* and the way *argData* is sorted, because they establish the seasonality within the model. All the other of the above parameters are rather common default values for neuro-predictors, which do not necessarily have to be tuned in the first place, but for sure also leave some room for improvements.

If an ARIMA predictor is used, the first three tuning parameters mentioned above are used in the same way, before the ARIMA models are generated automatically for each feature.

5.5.2 ABOD Tuning Parameters

Integer *setWeightFunction*: *setWeightFunction* can be set to 0, which means that no additional weighting function is used for the ABOD algorithm (This is also the default setting.), 1, which means that a distance function is considered additionally, leading to a comparison with the spatially closest points, or 2, which means that a temporal distance function is considered additionally, i.e. a certain point is compared with the spatiotemporally closest points. Setting this tuning parameter to 0 corresponds to applying algorithm 18, while setting this tuning parameter to 1 or 2 corresponds to applying algorithm 19 for certain sets of k nearest neighbours N_k.

Double *setLimitPosition*: The tuning parameter *setLimitPosition* defines the threshold of $ABOF(A)$, which separates outliers from inliers. The lower $ABOF(A)$ is, the more likely it is that A is an outlier. Thus *setLimitPosition* can be seen as an adjustable screw of the outageness; varying this tuning parameter gives a system administrator the possibility to control the total number of alarms.

Array *addTimeWeight*: If *setWeightFunction* is set to 2, *addTimeWeight* defines the sets N_k which are used to calculate $ABOF(A)$.

5.5.3 OC-SVM Tuning Parameters

Double *setGamma*: *setGamma* is the bandwidth of the Gaussian kernel. The lower the bandwidth is, the larger the number of inliers; the higher the bandwidth is, the larger the number of outliers and their outageness. The default value of *setGamma* is the approximated standard deviation of the prediction errors.

Double *setNu*: *setNu* is the desired inlier ratio for the training of the OC-SVM. It is definitely the most important parameter of which the vernier adjustment during the operation is highly recommended. It can be seen as an adjustable screw of the outageness, and different system administrators will probably prefer a different outageness for different servers. The range of *setNu* is $[0, 1]$, but in general *setNu* will be very close to 1, e.g. 0.99.

Double *setLambda*: *setLambda* is used to separate the data set by dint of the OC-SVM with the translated hyperplane $f(x) = setLambda$ instead of $f(x) = 0$. This basically establishes the possibility to control the trade-off between false and missed alarms and therefore allows a system administrator to tune this parameter according to his preferences. Following the suggestions by Lecomte made in [19], first *setNu* should be tuned before tuning *setLambda*. If *setLambda* remains set to its default value 0, the hyperplane translation technique is not used.

Integer *setWindowLength*: *setWindowLength* defines the length of the sliding window that is used for the outlier detection model. A reasonable window length also depends on the sampling rate of the model. If *setWindowLength* remains set to its default value 0, the sliding window technique is not applied within the model.

6 Comparison of the Applied Methods, Résumé and Outlook

First of all, it has to be stated again that it is almost impossible to precisely define the term *server outage*, wherefore a definition is not given in this paper. Any limitation to the normal operation of a server is unwanted. Many times only a certain kind of tasks is delayed or cannot be executed at all. The severity of this limitation also depends on the fact whether users can carry out other tasks in the mean time. The only possibilities to give the modeler an idea about the severity of an outage are the total downtime minutes or downtime minutes per user. Thus the basic idea of this model is to be able to provide the administrator of a server with the detection/prediction of irregularities, of anomalies which differ from the usual server operation.

Huge amounts of data that describe the actual server status are recorded with a nearly constant sampling rate and constitute the model input that needs to be preprocessed adequately. Inter alia, a feature selection is done using a classification done by experts according to the features' importance.

Basically, the proposed server outage detection model consists of two parts: a prediction model and an outlier detection model. For the prediction model two different methods have been implemented: a statistical ARIMA model and a neuro-predictor that uses an artificial neural network. Both prediction models work in a univariate way and are supposed to produce very accurate predictions, if the server status is ok. Both methods use cross-validation and can return the coefficient of determination R^2 for a test data set that was not used for establishing the model. The R^2-values are very high for almost all features for both methods, not indicating a significant preference for one of the two methods. As listed in section 3.4.5, there are certain time series properties like short memory and various complexity that can favour the neuro-predictor according to the findings by Sharda, Patil and Tang mentioned in citets1. Although it would be possible to choose a different predictor for every feature, this is not recommended due to the fact that this makes the implementation more difficult; in the end, a server outage detection tool should work in an on-line way. On these grounds also the numbers of lagged time series elements that are relevant for the univariate prediction models for each server feature and the seasonality parameters of the feature time series were chosen globally, although the optimal parameters certainly vary for each feature. All in all, the prediction models with global parameters for all the predictors worked very well during a normal operation of servers and seem to be sufficient for an on-line server outage detection model.

For the connected outlier detection model also two differents methods have been implemented: a one-class support vector machine and angle-based outlier detection, both methods suiting the requirements of the dimension of the outlier detection model which is usually higher than 70 and can be as high as 1439, if all the data is available and considered.

Model Validation. During several test runs, the anomaly detectors easily detected when the servers changed their status from idle to busy and vice versa. They also detected abnormal events within the gas price time series which was used as a benchmark data set. For this time series, an abnormal event is for example the oil crisis of 1979, which was caused by the Islamic revolution in Iran and the first gulf war, i.e. by external events. Other benchmark data sets for abnormal event detection are mostly data belonging to images or videos and would therefore require a totally different preprocessing that suits the requirements of image processing. The validation of the proposed server outage detection model is very difficult due to its nature. In the end, models including aspects about the future can only be validated, if the future has already become the present. Historic data set are available, but they are not labelled with respect to outages. The sensitivity of the detectors must be adjusted while the programme and the servers are running, and this will only be possible, if Panagenda finally publishes its software tool and allows the developers of this model to access actual data as well as to get some feedback of the users of this server outage detection software tool in a real-life scenario. So far, the detectors worked well with the test data sets.

The idea seems promising that several outlier detection methods could be used in a parallel way and that the outlierness increases if both methods classify an observation or a sequence of observations as an outlier.

Of course, a server outage prediction software has a cold start: During the training some internal model parameters that are required to run the model need to be adjusted, before an expert can adjust several tuning parameters to control the alert sensitivity of the software. The most important tuning parameters are part of the anomaly detector, all the tuning parameters are listed and discussed in section 5.5. One could say that the server outage detection model needs to get to know the server that the outages shall be predicted of. As parts of the model are able to learn from the past, the software will highly improve its performance after several days. An important question that still remains unanswered is when the neuro-predictors should be retrained or when the ARIMA models should be updated. Certainly, if the way the server is used changes considerably, a re-start of the model is necessary.

In the future, an automatic classification of outages would be very useful, but requires outage data to learn from. This data should be labelled with the outage cause by experts and could build the basis for an intelligent *catalogue of outages*. This classification would allow a far more exact description of outage causes, but this still remains future work. A big advantage would probably be that such a catalogue should be easy to port to different server systems, which would also reduce the time needed for the cold start of the server outage detection software.

A Abbreviations and Mathematical Symbols

a_k	tuning parameter
A	fuzzy set; a point
ABOD	angle-based outlier detection
ABOF	angle-based outlier factor
$ABOF_k$	angle-based outlier factor belonging to the FastABOD algorithm
ACF	autocorrelation function
AIC	Akaike information criterion
AR	autoregressive
ARIMA	autoregressive integrated moving average
ARMA	autoregressive moving average
ART	adaptive resonance theory
AUC	area under the curve
b	bias of a support vector machine
B	delay operator; a point; budget
BIC	Bayesian information criterion
c	constant
C	a point
$\mathbb{COV}(x, y)$	covariance of two random variables x and y
d	delay; distance; number of differences
d_1	Manhattan distance
d_2	Euclidean distance
d_k	distance to the k^{th} nearest neighbour
d_M	Mahalanobis distance
d_{max}	distance to the point farthest away
d_{min}	distance to the closest point
$d_{reach,k}$	reachability distance
D	number of seasonal differences
$\mathcal{D}$	data set
$\mathcal{D}_0$	Panagenda data set including only variables with priority 0
$\mathcal{D}_1$	Panagenda data set including only variables with priority 0 and 1
D_0	decision that an observation is an inlier
D_1	decision that an observation is an outlier
DM	data mining
e	error
E	error
$\mathbb{E}(x)$	expectation of a random variable x
f	a function; frequency
f_{act}	activation function
f_{minmax}	minmax-mapping function

$f_{sampling}$	sampling rate
f_{zscore}	Z-score-mapping function
FastABOD	approximative angle-based outlier detection
g	a function; gradient
GARCH	generalised autoregressive conditionally heteroscedastic
h	leverage; h-step-prediction; a function
H	hat matrix
H_0	hypothesis that an observation is an inlier
H_1	hypothesis that an observation is an outlier
i	iteration indices
I	identity matrix
INFLO	influenced outlierness
j	iteration indices
k	iteration index; number of model parameters
K	kernel function
KDD	knowledge discovery in databases
KKT	Karush-Kuhn-Tucker
kNN	k nearest neighbours
L_{change}	maximum likelihood criterion for being a change point
L_{no_change}	maximum likelihood criterion for being no change point
L_k	maximum likelihood criterion for the iteration k
LARS	least angle regression algorithm
LB-ABOD	lower bound angle-based outlier detection
LMA	Levenberg-Marquardt algorithm
LOF	local outlier factor
lrd_k	local reaching density
m	dimension
M	duration of an event
MA	moving average
ML	machine learning
n	number of time series elements
n_i, n_h, n_o	number of input, hidden and output neurons
N	number of training examples; number of support vectors
N_j	the neuron j
N_k	set of the k nearest neighbours of a point
NaN	not a number
net_i	input of neuron N_i
NIC	network information criterion
NN	artificial neural network
o_j	output of neuron N_j
O	set of outliers

$O(x)$	outlierness of x
OC-SVM	one-class support vector machine
p	order of an AR model; a point
P	probability
PACF	partial autocorrelation function
PCA	principal component analysis
q	order of an MA model; a point
r	correlation coefficient; residual
$\mathbb{R}$	autocorrelation matrix
R^2	coefficient of determination, see definition 11
RBF	radial basis function
RL	reinforcement learning
RMS	root mean square
RNN	recurrent neural network
ROC	receiver operatoring characteristic
Rprop	resilient backpropagation
s	length of the season
S	covariance matrix
SARIMA	seasonal autoregressive integrated moving average
SC	soft computing
SOD	subspace outlier degree
SOM	self-organising map
SSE	summed squared error
s.t.	subject to
SVD	singular value decomposition
SVM	support vector machine
t	time, time index
$\top$	fuzzy t-norm
$\bot$	fuzzy t-conorm
$\mathbb{VAR}(x)$	variance of the random variable x
VSARIMA	vector seasonal autoregressive integrated moving average
w	weight vector of a neural network; normal vector of a hypersphere
W	weight matrix of a neural network; set or number of weights; hypersphere
WD	weight decay
x	time series element; input neuron
X	set of points; universe of discourse; a matrix
$\bar{x}$	mean value of observed data x_i
y	vector; output neuron
z	polynomial variable

α_i	real constant; Lagrange multiplier
β_i	real constant
γ	threshold of the decision function of a one-class support vector machine
Δ	differencing operator
ε	white noise, error; radius; threshold
η_{ij}	learning rate for the weight ω_{ij}
$\eta^{\uparrow}, \eta^{\downarrow}, \eta_{min}, \eta_{max}$	resilient backpropagation parameters
θ	the moving-average operator; threshold
θ_i	MA model parameter; threshold of neuron N_i
ϑ	kernel parameter
λ	Lyapunow exponent; model parameter; Levenberg-Marquardt algorithm damping factor
λ_i	Karush-Kuhn-Tucker multiplier
μ	expectation of the random variable that is talked about
μ_A	membership function of the fuzzy set A
μ_i	Karush-Kuhn-Tucker multiplier
ν	fraction of data that are allowed to be outliers
ξ_i	slack variable
π	percentage
σ	variance of the random variable that is talked about
τ	a not specified number of time steps into the past or future
ϕ	the autoregressive operator; a function; kernel mapping function
ϕ_i	AR model parameter
φ	generalized autoregressive operator; kernel mapping function
ω_{ij}	weight between input neuron N_i and output neuron N_j
∇	differencing operator
∇_s	seasonal differencing operator

B Auxiliary Calculations and Examples

B.1 Initial Values for Example 6

$$\begin{pmatrix} x_1 \\ \vdots \\ x_{18} \end{pmatrix} = \begin{pmatrix} 0.9697 \\ 0.9699 \\ 0.9794 \\ 1.0003 \\ 1.0319 \\ 1.0703 \\ 1.1076 \\ 1.1352 \\ 1.1485 \\ 1.1482 \\ 1.1383 \\ 1.1234 \\ 1.1072 \\ 1.0928 \\ 1.0820 \\ 1.0756 \\ 1.0739 \\ 1.0759 \end{pmatrix}$$

B.2 List of the Variables of $\mathcal{D}_1$

api_statistics_status
Database.Database.BufferPool.PerCentReadsInBuffer
Database.DbCache.OvercrowdingRejections
Mail.Mailbox.AccessConflicts
Platform.LogicalDisk.1.PctUtil
Platform.LogicalDisk.2.PctUtil
Platform.LogicalDisk.3.PctUtil
Platform.LogicalDisk.4.PctUtil
Platform.LogicalDisk.5.PctUtil
Platform.Memory.RAM.PctUtil
Platform.System.PctTotalPrivilegedCpuUtil
Platform.System.PctTotalUserCpuUtil
Server.AvailabilityIndex
Server.ConcurrentTasks
Agent.Hourly.AccessDenials
Agent.Hourly.ScheduledRuns

Agent.Hourly.UnsuccessfulRuns
Database.DbCache.CurrentEntries
Database.DbCache.Hits
Database.FreeHandleStack.MissRate
Database.NAMELookupCacheHits
Database.NAMELookupCacheMisses
Database.NAMELookupCacheNoHitHits
Database.NIFPool.Used
Database.NSF.Replicate.SCR.SrvTotalOps
Database.NSFPool.Used
Domino.Cache.Design.HitRate
Domino.Cache.User.Cache.HitRate
FT.Index.Total.Bytes
FT.Index.Total.TimeMS
FT.Search.Total.TimeMS
Http.Accept.Errors
Http.CurrentConnections
Mail.AverageDeliverTime
Mail.DBCacheHits
MAIL.Dead
MAIL.Hold
Mail.Queue.Dispatch.Waiting
Mail.Queue.Event.Waiting
Mail.Queue.Mailbox.Waiting
Mail.Queue.Sweep.Waiting
Mail.Queue.ThreadPool.Waiting
MAIL.Waiting
Mem.Free
NET.TCPIP.BytesReceived
NET.TCPIP.BytesSent
Platform.Memory.PageFaultsPerSec
Platform.Network.Total.NetworkBytesPerSec
Platform.Network.Total.PctUtilBandwidth
Platform.PagingFile.Total.PctUtil
Platform.Process.ActiveDomino.TotalCpuUtil
Platform.Process.amgr.1.PctCpuUtil
Platform.Process.amgr.1.PgFaultsPerSec
Platform.Process.amgr.2.PctCpuUtil
Platform.Process.amgr.2.PgFaultsPerSec
Platform.Process.amgr.3.PctCpuUtil
Platform.Process.amgr.3.PgFaultsPerSec

Platform.Process.server.1.PctCpuUtil
Platform.Process.server.1.PgFaultsPerSec
Platform.Process.update.1.PctCpuUtil
Replica.Cluster.Failed
Replica.Cluster.SecondsOnQueue
Replica.Cluster.WorkQueueDepth
Replica.Failed
Server.Cluster.OpenRequest.ClusterBusy
Server.Cluster.OpenRequest.LoadBalanced
Server.ExpansionFactor
Server.Sessions.Dropped
Server.Trans.PerMinute
Server.Trans.Total
Server.Users
Server.Users.Active
Update.PendingList

B.3 ARIMA Model for Gas Prices

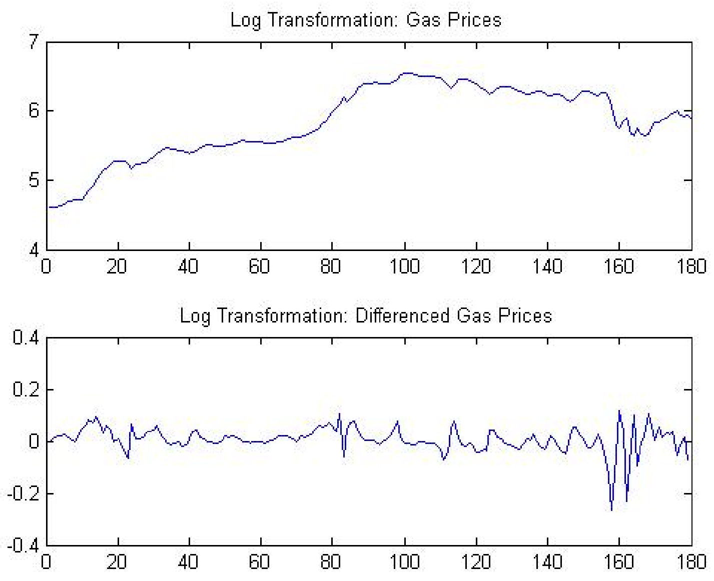


Figure 30: Gas prices time series and its differences after the logarithmic transformation.

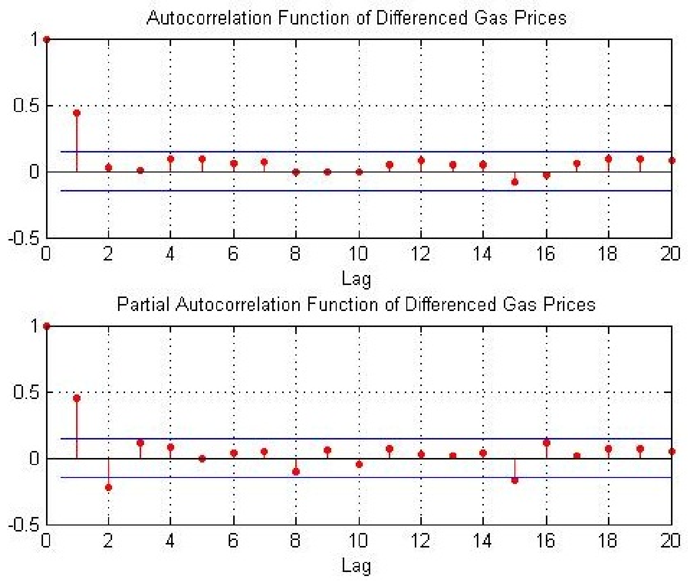


Figure 31: Autocorrelation and partial autocorrelation coefficients of the differenced gas prices time series.

C References

[1] Aloy Palit, Dobrivoje Popovic. *Computational Intelligence in Time Series Forecasting - Theory and Engineering Applications.* Springer Verlag, London, 2005, ISBN 1852339489.

[2] Mark Last, Abraham Kandel, Horst Bunke (editors). *Data Mining in Time Series Databases*, Volume 57 of *Series in Machine Perception & Artificial Intelligence.* World Scientific, Singapore, 2004, ISBN 9812382909.

[3] Mehmed Kantardzic. *Data Mining - Concepts, Models, Methods, and Algorithms.* John Wiley & Sons, Hoboken, New Jersey, 2003, ISBN 0471228524.

[4] Rafig Älijev, Karl Bonfig, Fuad Älijev. *Soft Computing - eine grundlegende Einführung.* Verl. Technik, Berlin, 2000, ISBN 3341012389.

[5] David Kriesel. *Ein kleiner Überblick über Neuronale Netze.* Published online, 2007, available at `http://www.dkriesel.com`, downloaded 21.02.2012.

[6] Stephan Dreiseitl, Melanie Osl, Christian Scheibböck, Michael Binder. *Outlier Detection with One-Class SVMs: An Application to Melanoma Prognosis*, Proceedings of the *AMIA Annual Symposium 2010*, pp. 172-176. Published online, 2010, available at `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3041295`, downloaded 17.07.2012.

[7] Riu Zhang, Shaoyan Zhang, Yang Lan, Jianmin Jiang. *Network Anomaly Detection Using One Class Support Vector Machine*, Volume 1 of the Proceedings of the *MultiConference of Engineers and Computer Scientists 2008.* IAENG, Hong Kong, 2008, ISBN 9789889867188.

[8] Xiaoming Wang, Shitong Wang. *On the Hyperplane of One-class Support Vector Machine*, Number 6 of Volume 8 of the *Journal of Computational Information Systems*, pp. 2301-2308. Binary Information Press, USA, 2012, ISSN 15539105.

[9] Katherine Heller, Krysta Svore, Angelos Keromytis, Salvatore Stolfo. *One Class Support Vector Machines for Detecting Anomalous Windows Registry Accesses*, Proceedings of the *Workshop on Data Mining for Computer Security* in conjunction with the *IEEE International Conference on Data Mining 2003*, pp. 2-9. Melbourne, Florida, 2003, available at `http://cs.fit.edu/~pkc/dmsec03`.

[10] Larry Manevitz, Malik Yousef. *One-Class SVMs for Document Classification*, Volume 2 of the *Journal of Machine Learning Research*, pp. 139-154. MIT Press, Cambridge, Massachusetts, 2001, ISSN 15324435, available at `http://jmlr.csail.mit.edu`.

[11] Régis Vert. *The Limit of One-Class SVM* University of Paris-Sud 11. Video published 25.02.2007, recorded in October 2005, available at `http://videolectures.net/mcslw04_vert_locs`.

[12] Bernhard Schölkopf, Alexander Smola. *Learning with Kernels*. MIT Press, Cambridge, Massachusetts, 2002, ISBN 0262194759.

[13] Paul Evangelista, Piero Bonnisone, Mark Embrechts, Boleslaw Szymanski. *Fuzzy ROC Curves for the 1 Class SVM: Application to Intrusion Detection*, Proceedings of the *13th European Symposium on Artificial Neural Networks 2005*, pp. 345-350. d-side, Bruges, Belgium, 2005, ISBN 2930307056, available at `http://www.dice.ucl.ac.be/esann/`.

[14] G. Peter Zhang, Douglas Kline. *Quarterly Time-Series Forecasting With Neural Networks*, Number 6 of Volume 8 of the *IEEE Transactions on Neural Networks*, pp. 1800-1814. IEEE Computational Intelligence Society, November 2007, ISSN 10459227.

[15] Sven Crone, Rohit Dhawan. *Forecasting Seasonal Time Series with Neural Networks: A Sensitivity Analysis of Architecture Parameters*, Proceedings of the *International Joint Conference on Neural Networks 2007*, pp. 2099-2104. IEEE, Orlando, Florida, 2007, ISSN 10987576.

[16] Victoria Hodge, Jim Austin. *A Survey of Outlier Detection Methodologies*, Issue 2 of Volume 22 of the *Artificial Intelligence Review*, pp. 85-126. Kluwer Academic Publishers, Netherlands, 2004, ISSN 02692821.

[17] Hans-Peter Kriegel, Matthias Schubert, Arthur Zimek. *Angle-Based Outlier Detection in High-Dimensional Data*. Proceedings of the *14th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining 2008*, pp. 444-452. Las Vegas, Nevada, 2008. ACM, New York, 2008, ISBN 9781605581934.

[18] Ninh Pham, Rasmus Pagh. *A Near-Linear Time Approximation Algorithm for Angle-Based Outlier Detection in High-Dimensional Data*. Proceedings of the *18th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining 2012*, pp. 877-885. ACM, New York, 2012, ISBN 9781450314626, available at `http://www.itu.dk/people/pagh/papers/outlier.pdf`.

[19] Sébastian Lecomte, Régis Lengellé, Cédric Richard, Francois Capman, Bertrand Ravera. *Abnormal Events Detection using Unsupervised One-Class SVM - Application to Audio Surveillance and Evaluation.* Proceedings of the *8th IEEE International Conference on Advanced Video and Signal-Based Surveillance 2011*, pp. 124-129. IEEE, Klagenfurt, Austria, 2011, ISBN 9781457708442.

[20] Simon Hawkins, Hongxing He, Graham Williams, Rohan Baxter. *Outlier Detection Using Replicator Neural Networks*, Proceedings of the *4th International Conference on Data Warehousing and Knowledge Discovery 2002.* Aix-en-Provence, France, 2002. Published in *Lecture Notes in Computer Science 2454*, Springer 2002, pp. 113-123, ISBN 3540441239.

[21] Fan Jiang, Ying Wu, Aggelos Katsaggelos. *Abnormal Event Detection Based on Trajectory Clustering by 2-Depth Greedy Search*, Proceedings of the *IEEE International Conference on Acoustics, Speech and Signal Processing 2008*, pp. 2129-2132. IEEE, Las Vegas, Nevada, 2008, ISBN 1424414849.

[22] Yang Cong, Junsong Yuan, Ji Liu. *Sparse Reconstruction Cost for Abnormal Event Detection*, Proceedings of the *24th IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3449-3456. IEEE, Colorado Springs, Colorado, 2011, available at `http://pages.cs.wisc.edu/~ji-liu/paper/Cong-Yuan-CVPR11.pdf`.

[23] Mitsutoshi Suzuki, Hitoshi Ihara. *Development of Safeguards System Simulator Composed of Multi-Functional Cores*, Number 2 of Volume 2 of the *Journal of Power and Energy Systems*, pp. 899-907. J-Stage, Japan, 2008, ISSN 18813062, available at `https://www.jstage.jst.go.jp/browse/jpes`.

[24] Alireza Ferdowsizadeh Naeeni. *Advanced Multi-Agent Fuzzy Reinforcement Learning*, Master Thesis Computer Engineering, E3098D, 2004, available at `http://www2.informatik.hu-berlin.de/~ferdowsi/`.

[25] Robert Nau. *Forecasting - Decision 411*, online course, 2005, available at `http://people.duke.edu/~rnau/Decision411CoursePage.htm`.

[26] Kurt Hornik. *Approximation Capabilities of Multilayer Feedforward Networks*, Number 2 of Volume 4 of *Neural Networks*, pp. 251-257. Elsevier, 1991, ISSN 08936080.

[27] Yoshua Bengio. *Learning Deep Architectures for AI*, Number 1 of Volume 2 of *Foundations and Trends in Machine Learning*, pp. 1-127.

2009, ISSN 19358237, available at `http://www.iro.umontreal.ca/~bengioy/papers/ftml_book.pdf`.

[28] Valery Guralnik, Jaideep Srivastava. *Event Detection from Time Series Data*, Proceedings of the *5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 33-42. ACM New York, USA, 1999, ISBN:1-58113-143-7, available at `dmr.cs.umn.edu/Papers/P1999_6.pdf`.

D Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources and resources and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Date, Signature

E Acknowledgements

I would like to express my gratitude to Michael Landsiedl and Christoph Urach of the dwh GmbH for the useful comments, remarks and engagement through the learning process of this master thesis. Furthermore I would like to thank Niki Popper and Martin Bruckner of the dwh GmbH as well as supervisor Dr. Felix Breitenecker for their support on the way. Also, I like to thank the partners from Panagenda, especially Franz Walder, who have willingly provided tons of data. A special thanks to my father Josef Wastian and my friend Christoph Topitschnig for their remarks on how to express myself in English as correctly as possible.

I would like to thank all my loved ones, especially my love Sarah Wagner, who have supported me throughout the entire process. I will be grateful forever for your love.

Matthias Wastian