Fabian Peter Pribahsnik, BSc.

Let the data tell us their story
Machine learning with insurance policies

Master's Thesis

to achieve the university degree of

Master of Science

Financial and Actuarial Mathematics

submitted to

Vienna University of Technology

Institute of Statistics and Mathematical Methods in Economics

Supervisor

Univ.Prof. Dipl.-Math. Dr.rer.nat. Thorsten Rheinländer

Date	Supervisor	Author

Affidavit

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

Date	Fabian Pribahsnik

Abstract

Insurance companies are facing a huge amount of regulations, including various guidelines addressing forecast scenario calculations for the policies in the portfolio. Taking the hundreds of thousands policies into account an average insurance company has in its portfolio one can easily see that these scenario calculations are very time consuming. Due to the rising number of policies and the very tight time schedule introduced with Solvency II insurance companies are looking for ways to reduce the computational time significantly. In the past years different approaches were developed and already used for grouping similar policies together and therefore reducing the computation time. The currently used algorithms are ranging from just grouping policies with exactly the same attributes together to basic cluster algorithms like k-means. This work highlights potential problems with the algorithms currently used and shows the implementation of some machine learning techniques which can be used to replicate cash flows of insurance policies.

Kurzfassung

Versicherungsunternehmen sehen sich mit einer immer größer werdenden Anzahl von Vorschriften konfrontiert. Ein Teil dieser Richtlinien beschäftigt sich dabei mit der Fragestellung wie Versicherungsportfolios in die Zukunft projiziert werden sollen. Bedenkt man dabei, dass ein durchschnittliches Versicherungsunternehmen hunderttausende von Polizzen im Bestand hat, kann man den Zeitaufwand erahnen, der für solche Prognoserechnungen aufzuwenden ist. Steigende Bestände von Versicherungspolizzen in Kombination mit engen aufsichtsrechtlich gesetzten Fristen führen dazu, dass Versicherungsunternehmen nach Möglichkeiten suchen, die Rechenzeit für Projektionen zu verkürzen. In den letzten Jahren wurden bereits verschiedene Ansätze entwickelt und zum Einsatz gebracht, um ähnliche Polizzen zusammenzufassen und damit die Rechenzeit zu verkürzen. Dabei reichen die derzeit verwendeten Methoden von einfachen Zusammenfassungen von Polizzen mit exakt gleichen Attributen bis hin zu Clusteralgorithmen wie beispielswiese k-means. Diese Arbeit zeigt mögliche Probleme mit den derzeit verwendeten Algorithmen auf und präsentiert neue Implementierungsansätze aus dem Bereich des maschinellen Lernens, um eine Gruppierung des Versicherungsbestandes noch effektiver umsetzen zu können.

Acknowledgement

Many people accompanied me on my way to the graduation of this study, whose acquaintance I enjoyed very much. I would also like to thank my supervisor Prof. Thorsten Rheinländer very much for his patience which he has given me in the course of the longer supervision time and for his outstanding support with each of my questions. My colleagues at work also deserve a big thank you for always having an open ear for all my questions. I would especially like to thank my boss Dietmar Hareter for his continuous expertise and mental support, which has certainly been an essential factor in helping me to complete all my open tasks. I particularly want to thank my fellow students Birgit Bauer and Markus Kakac, who have made a significant and essential contribution to the fact that I always look back on my studies with pleasure. An important constant in my life that supports me in all my decisions and is always there for me is my beloved girlfriend Claudia Hilsberg who helped me never lose sight of the goal. All that I have achieved so far would not have been possible without the great support of my parents. I look back with great joy on the talks we had together about all those things that we could do in the future, but unfortunately it was not meant to be that we could celebrate my graduation together. You will always be an important part of my life.

Contents

List of Figures

1. Introduction

A fundamental problem faced by many insurance companies is the projection of the insurance portfolio into the future. In a projection, the future cash flows for each individual policy must be calculated on the basis of actuarial principles and then saved for further analysis. Depending on the purpose of the forecast, these cash flows have to be provided either on a monthly or annually basis. Together with the contractually agreed benefits to the policyholder, a wide variety of other parameters must also be taken into account and modelled in such projections. These include, for example, all contract changes that may occur during the duration of the contract, such as a premium pause, a surrender or the occurrence of a claim. While the calculation of a small set of policies over a relatively short time horizon does not pose a challenge in terms of computation time, this fact changes greatly when projecting entire portfolios. This problem is particularly severe for life insurance portfolios. These contracts often have durations of several decades and the portfolios tend to grow over time as fewer policyholders leave than new ones are added. Even with the optimistic assumption that the complete projection of a single contract only takes a hundredth of a second, the computing time adds up to several hours for a portfolio with several million policies. As the liabilities are projected over a period of several decades, it is also essential to simulate the development of the assets, underlying the liabilities, over this period. Even under the assumption that the assets can be represented by a small number of different types of investments, their projection additionally increases the run time of the projection. The simultaneous simulation of the asset and liability portfolios results in further factors that have to be taken into account with respect to the behaviour of policyholders during the contract term. If one assumes that the policyholder behaviour during the projection period also depends on external parameters such as the current interest rates on savings, one also has to include that effect into the simulation. This finally results in a dynamic interaction of all components which can be summarized in a simplified pseudo code given in algorithm 1:

Note that during an iteration (algorithm 1, 2a - 2d) all results must be kept in

Algorithm 1 Simplified dynamic interaction scheme of portfolio projection

1. Initialisation of assets and liabilities at time $t = 0$.
2. Iteration till the end of the projection horizon is reached:
   a) Calculation of one time step for the liabilities. This means that all policies are rolled forward to the next year ($t \mapsto t + 1$).
   b) Calculation of one time step for the assets. This means that the returns of all investments are simulated for the next year.
   c) Simulation of the dynamic policyholder behaviour based on the current economic parameters from the previous step.
   d) Application of pre-defined management decisions based on current developments of assets and liabilities. This means that in this step it is determined how high the dividend per share, for example, will be for this year.

memory until every decision for that loop is made and the next iteration starts. Assuming that there is a portfolio of several million policies and for each policy at least a few dozen variables are relevant for the decision making process, a memory requirement of at least hundreds of million individual values emerges. Also the simulations of the assets and the subsequent management decisions have a considerable memory requirement. All this leads to the fact that the projection models currently used in the insurance industry are extremely demanding in terms of their memory requirements and execution time. There are at least two different ways to address this problem:

- Outsourcing of calculations to an external high performance infrastructure.
- Data compression in the sense that similar policies are grouped together and only the grouped portfolio is calculated.

With the increasing availability of cloud services, it is already an option these days to outsource complex computing operations to specialized providers. It is possible to rent different types of computing capacities for a certain period of time without much effort and run the projections there. The big advantage of this option is that there is no need to set up, maintain and configure an infrastructure. Since personal data such as age or gender are always used in the projection of insurance contracts, potential problems could arise with respect to the General Data Protection Regulation [43]. Many companies have therefore decided to compress the insurance portfolio and therefore reducing

the memory requirements using various techniques. It is then feasible to calculate the projections with infrastructure that is owned by the company. This approach has the big advantage that it avoids possible problems with data protection issues and also reduces the dependency on external services. On the other hand, an additional effort has to be made to compress the portfolio in a proper way. Of course, it is of utmost importance that the compressed portfolio has the same characteristics and produces the same cash flows as the non-compressed one. This is exactly where this thesis comes in by presenting possible ways and methods of compressing a portfolio of insurance contracts or simulating their cash flows.

1.1. Problem formulation

For the sake of simplicity all values calculated by the projection tool are referred to as cash flows regardless of whether they are non-cumulative values such as the reserve or actual cash flows such as the premium. The task is to find a portfolio with a reduced number of policies, the so called grouped portfolio, so that the projected cash flows match those from the reference portfolio as accurately as possible. In order to be able to define what accurately means in terms of cash flow deviations, some basic concepts must be defined first .

Definition 1.1. *Let $\mathcal{V} = \{V,\ V \text{ is a valid insurance contract}\}$ be the set of all valid insurance contracts. A finite set of elements $P \subset \mathcal{V}$ is called a portfolio and the number of policies within that portfolio is determined by its cardinality.*

Definition 1.2. *Let $P \subset \mathcal{V}$ be a portfolio of n policies (i.e. $|P| = n$) and m the number of projected cash flows by the simulation s. Then the individual cash flows corresponding to P are encoded in $A \in \mathbb{R}^{m \times n}$ and the summed cash flows are encoded in $b \in \mathbb{R}^m$.*

$$\begin{aligned} s : \mathcal{V} &\to \mathbb{R}^{m \times n} & \qquad s : \mathcal{V} &\to \mathbb{R}^m \\ P &\mapsto s(P) = A & \qquad P &\mapsto A \cdot \mathbb{1}_{n \times 1} = b \end{aligned}$$

Remark 1.1. *It should be noted that the definition of portfolio is ambiguous. It can include anything from a single policy to all policies held by an insurance company. The usual segmentation into portfolios is often based on the following characteristics:*

- *All contracts of a singe tariff are put into one portfolio.*

- *All contracts of a tariff group (endowment, annuity, ...) are put into one portfolio.*
- *All contracts sold in the same country are put into one portfolio.*

Example 1.1. *Let $P \subset \mathcal{V}$ be a portfolio with $|P| = 1000$. The number of cash flows is given by $m = 180$. Then the cash flows generated by the projection software can be described as:*

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,1000} \\ a_{2,1} & \ddots & & a_{2,1000} \\ \vdots & & \ddots & \vdots \\ a_{180,1} & a_{180,2} & \cdots & a_{180,1000} \end{pmatrix} \qquad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{180} \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{1000} a_{1,i} \\ \sum_{i=1}^{1000} a_{2,i} \\ \vdots \\ \sum_{i=1}^{1000} a_{180,i} \end{pmatrix}$$

A column in matrix A describes all cash flows generated by this policy. One row of matrix A describes the same cash flow generated by the different policies. By calculating line totals, the corresponding cash flow is obtained at portfolio level. The individual policy by policy cash flows are therefore given as matrix A, and the portfolio cash flows as vector b.

Definition 1.3. *Let $P \subset \mathcal{V}$ be a portfolio with $|P| = n$, $\tilde{P} \subset \mathcal{V}$ the grouped portfolio with $|\tilde{P}| = \tilde{n}$, $\tilde{n} < n$, $w \in \mathbb{R}^m_{\geq 0} = \{x \in \mathbb{R}^m, x \geq 0\}$ a vector of weights and $A \circ B$ the Hadamard product of two matrices A and B. Then the weighted sum of squares of the cash flows over the entire projection horizon between the grouped portfolio $\tilde{P}$ and the ungrouped one P is given by:*

$$WSS_{total} = \|w \circ (b - \tilde{b})\|_2^2 \tag{1.1}$$

The weighting parameter w makes it possible to make deviations from certain cash flows more or less important by weighting them differently. This can be useful, for example, to weight deviations at the end of the projection period less heavily compared to deviations at the beginning of the projection horizon. Particularly with projection horizons of 30 years and more, it may make sense to weight deviations in cash flows less strongly at a later point in time.

Remark 1.2. *In a situation where every deviation of cash flows between the grouped and the ungrouped portfolio is equally important, the elements of w are all one. In this situation equation (1.1) simplifies to:*

$$WSS_{total} = \|b - \tilde{b}\|_2^2 \tag{1.2}$$

The aim is to develop a method which is capable of finding a suitable grouped portfolio for each portfolio at hand so that (1.1) is sufficiently small. If there are several methods that come to similar results, then of course the one that needs the smallest grouped portfolio (i.e. smallest $\tilde{n}$) to generate the desired cash flows is of interest.

This thesis reviews the currently used technique for grouping policies together and introduces furthermore some new approaches on how life insurance policies can be grouped together. Therefore drawbacks and advantages with a special emphasis on the regulatory requirements of every approach discussed will be highlighted. Theoretical considerations as well as practical implementations and tests with real world data will provide some information on which method an insurance company should work with in order to obtain the best grouping results.

This thesis is structured as follows: First, an overview is given of the legal framework that defines the minimum requirements for grouping approaches in insurance companies. The next chapter is dedicated to the question on how sensitive main characteristics of a policy are with respect to the interest rate, the age or the duration. This analysis, carried out on a real insurance portfolio with a widely used projection tool, already gives a first impression which parameters are important for grouping purposes. In the next chapter one of the most popular unsupervised learning algorithms called k-means is presented. Besides the algorithm itself, further theoretical aspects regarding the optimal choice of clusters are given. Then the next chapter is dedicated to an optimization algorithm called non-negative least squares. For this algorithm, which is often used in practice, the focus is also on the possible numerical instabilities that can occur during execution. This is followed by a chapter which shows that machine learning methods can also be used to simulate cash flows of forecast models. A special focus therefore is put on neural networks and their application to real data. The final chapter then provides a brief summary of the results obtained and offers an outlook on possible further fields of research.

1.2. Legal Framework

Solvency II - entered into force on 1. January 2016 - is the European framework for a common insurance supervision. It is intended to achieve a harmonization of the European insurance sector and was implemented in accordance with

the Lamfalussy architecture which works on a 4 level basis [8]. The most significant elements and aims of the new regulation framework can be studied on the homepage of the financial market authority (FMA) [18] and on the homepage of the European Insurance and Occupational Pensions Authority (EIOPA) [16]. This work is intended not to cover all aspects and aims of the new Solvency II regulation framework but focuses on the topic of data quality regarding to the actuarial function. In order to meet all the requirements imposed by Solvency II, insurance companies need to process large amounts of data within a short period. One critical aspect of these calculations is the projection horizon which however should cover the full lifetime of all obligations as stated in [6]:

3.83.

> The projection horizon used in the calculation of best estimate should cover the full lifetime of all obligations related to existing insurance and reinsurance contracts on the date of the valuation.

3.84.

> The determination of the lifetime of insurance and reinsurance obligations shall be based on up-to-date and credible information and realistic assumptions about when the existing insurance and reinsurance obligations will be discharged or cancelled or expired.

Another aspect needed to be considered is the fact that cash flow calculations need to be done for a variety of different economic scenarios which yields to an enormous computational effort. Due to the tight time schedule, insurance companies are looking for new possibilities to speed up these time consuming calculations. One approach is not to make all these calculations on a per policy level, but on a grouped level where similar policies are grouped together and represented by only a few policies. This approach raises the question of how to maintain data quality as mentioned in the level 1 directive [40] while reducing the number of policies.

Article 82

Data quality and application of approximations, including case-by-case approaches, for technical provisions

> Member States shall ensure that insurance and reinsurance undertakings have internal processes and procedures in place to ensure

> the appropriateness, completeness and accuracy of the data used in the calculation of their technical provisions...

By publishing the level 2 regulations, supplementing the level 1 directive [40] the European Commission is getting more specific on data quality (Article 19 in [7]) and also formulates concrete requirements for grouped policies [7].

> *Article 35*
>
> **Homogeneous risk groups of life insurance obligations**
>
> The cash flow projections used in the calculation of best estimates for life insurance obligations shall be made separately for each policy. Where the separate calculation for each policy would be an undue burden on the insurance or reinsurance undertaking, it may carry out the projection by grouping policies, provided that the grouping complies with all of the following requirements:
>
> *a*) there are no significant differences in the nature and complexity of the risks underlying the policies that belong to the same group;
>
> *b*) the grouping of policies does not misrepresent the risk underlying the policies and does not misstate their expenses;
>
> *c*) the grouping of policies is likely to give approximately the same results for the best estimate calculation as a calculation on a per policy basis, in particular in relation to financial guarantees and contractual options included in the policies.

These level 2 regulations are a reference point on what to consider when grouping policies together and they are even further specified in the level 3 guidelines issued by EIOPA[15]. Further details on the level 3 guidelines including feedback statements to the consultation paper (EIOPACP-14/036) and the guidelines can be obtained from [14].

1.3. Statistical Learning

Statistical learning refers to a set of methods which deals with predicting outcomes based on input variables or finding patterns in data sets. In order to accomplish the task of grouping together similar policies, different approaches from statistical learning can be applied. All these methods can be classified

either as supervised or unsupervised. Within the framework of supervised methods, statistical models try to predict output variables y_i based on some input variables x_i where the relation $y = f(x)$ is unknown. It is therefore indispensable to have input as well as output data to parameterize such a model in order to find a prediction $\hat{f}$ of f. Unsupervised methods, in contrast, are used when inputs x_i but no corresponding outputs y_i are available. These methods then try to find some hidden patterns within to data. The task of grouping insurance policies involves many different aspects. On the one hand we have all data needed to apply supervised methods, but on the other hand we are only interested in the patterns that can be revealed by an unsupervised method. The input variables are given by the characteristics of each policy and the corresponding output variables are determined by the projection tool. Our primary goal is not to get a good $\hat{f}$ because the projection tool, which stands for f, is already known. We are more interested in hidden patterns that can be used for grouping purposes. In a first step we will apply unsupervised methods to the data and try to group the policies based on their characteristics and their cash flows. In a further step we will try to use the additional information of f to improve the grouping results if possible.

2. Sensitivities

Grouping together single policies and representing them by just a few representative ones always comes along with a loss of information. A natural question which arises when grouping insurance contracts together is how to determine the main characteristics of the new representative policy. Some characteristics should for technical reasons be defined as the sum of the individual ones, like the sum insured, the premium or the accumulated reserve. This is needed to guarantee the equality between the ungrouped and the grouped portfolio in terms of these characteristics at the beginning of the projection horizon. For other characteristics like the age, the duration or the gender it is not intuitively clear how they should be defined for a representative policy. Possible solutions which can be implemented easily range from taking the weighted average over the value with the highest relative frequency or to just taking the median of the grouped policies. Another difficulty which arises when grouping together policies from different product generations is, how the technical interest rate of the representative contract should be defined. Even if the policies are identical in terms of age, sex, sum insured, duration, costs,... and just differ on their issue date the huge possible differences with respect to the technical interest rate as shown in table 2.1 can have enormous impacts on the projected cash flows. Already a relative small difference in the technical interest of only one percent causes double digit differences in the guaranteed capital after 1 decade.

31.12. 1994	30.06. 2000	31.12. 2003	31.12. 2005	31.03. 2011	20.12. 2012	31.12. 2014	31.12. 2015	31.12. 2016
4%	3.25%	2.75%	2.25%	2%	1.75%	1.5%	1%	0.5%

Table 2.1.: Maximum technical interest rates for life insurance contracts issued after the given dates. (see [17])

It is therefore important to know how sensitive the different output variables of interest which are calculated by the projection tool react if various input parameters are changed slightly. The most basic task is to determine whether the correlation between the input and output variables is positive or negative.

2. Sensitivities

There are many output variables which are important for determining if a grouping process has been successful in terms of accuracy or not, but in the subsequent we will focus only on a few of them, namely the premium, the present value of future profits at time 0, the reserve and the yearly claims. Due to the big variety of different insurance products we will just give some general guidelines based on the most important input variables. Most of the life insurance contracts can be built up by the following elementary insurance types and some additional factors for different types of costs. [1]:

$$A_x = \sum_{k=0}^{\infty} v^{k+1}{}_k p_x q_{x+k} \qquad \text{(Whole life insurance)} \tag{2.1}$$

$$A^1_{x:\overline{n}|} = \sum_{k=0}^{n-1} v^{k+1}{}_k p_x q_{x+k} \qquad \text{(Term insurance)} \tag{2.2}$$

$$A_{x:\overline{n}|} = \sum_{k=0}^{n-1} v^{k+1}{}_k p_x q_{x+k} + v^n{}_n p_x \qquad \text{(Endowment)} \tag{2.3}$$

$$\ddot{a}_x = \sum_{k=0}^{\infty} v^k{}_k p_x \qquad \text{(Whole life annuity)} \tag{2.4}$$

$$\ddot{a}_{x:\overline{n}|} = \sum_{k=0}^{n-1} \ddot{a}_{\overline{k+1}|k} p_x q_{x+k} + \ddot{a}_{\overline{n}|}\,{}_n p_x \qquad \text{(Temporary life annuity)} \tag{2.5}$$

These basic types already show that the main characteristics which should be taken care of, when it comes to a grouping process are the age x, the duration n and the technical interest rate which is implicitly given by v. The following graphs and analyses are based on an endowment policy with a duration of 25 years, a sum insured of 10.000 € and an investment return of 3% p.a. over the whole projection horizon. The duration of 25 years is in all subsequent considerations equal to the duration of the premium payments which are made on a yearly basis. All other parameters especially the lapse, paid-up and surrender rates as well as first and second order assumptions can't be given here in detail.

[1] For detailed definitions and explanations see [19].

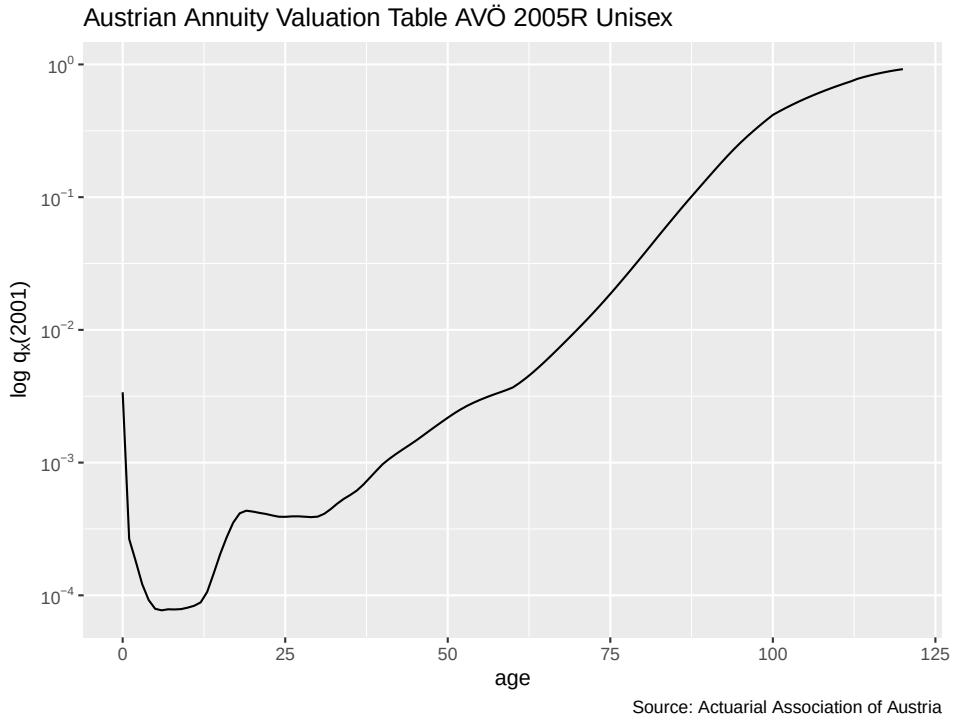


Figure 2.1.: Logarithm of the yearly mortality defined by the Austrian Annuity Valuation Table AVÖ 2005R Unisex.

2.1. Age

The age of an insured person is one of the main factors which drives the projected outcome because it directly influences the probability of death and survival as shown in (2.1) - (2.5). When the age is changed from x to $x+1$ the survival- and death-probabilities ${}_kp_x$ and ${}_kq_x$ change as well. It is impossible to predict in general whether the probabilities will rise or fall. Figure (2.1) shows, for example, the graph of logarithmic mortality rates based on the values of the unisex mortality table from the Actuarial Association of Austria [24]. A high level of non-linearity can be observed, which naturally leads to greater challenges in the grouping process.

As known from life tables it is a bit more likely to die just after birth than a bit afterwards and the same is true for people aged around 20. The exact ages where the probability of survival increases and the probability of death decreases when a person gets a year older depends heavily on the life table and the sex of the insured person. Whether the values for (2.1) - (2.5) will rise or

fall when the age x is increased by 1 year will therefore depend on n, x and the sex of the insured person. To get a better insight into the portfolio, simulation runs for various parameters should be done. In figure (2.2) the development of the yearly total claims is plotted against the duration of 25 years. The claims are the amount of money that must be paid to the policyholder at the time of an insured event multiplied by the probability that such an event will occur. Such events can be of all kinds, but the most common are, for example, the death of the insured person or a surrender of the contract.

For a clearer chart the last cash flow which is the sum insured and therefore substantially larger than the yearly claims is omitted. We see that for younger people (green and red lines) the claims are almost identical in the first years and only deviate slightly at the end of the duration due to minor differences in the probabilities of death. For an insured person aged 60 we observe over the entire projection horizon considerably higher claims compared to younger policyholders. This gap between young and old policyholders which is mostly driven by mortality effects even increases with time. An insured person aged 60 which gets one year older faces in absolute values an higher increase in the mortality rate compared to an insured person aged 40 and so the claims will be higher in absolute values for the older person. This effect is partially compensated by lower surrender claims due to the higher mortality rates. If one compares the development of the claims also with respect to different interest rates, one can see in figure (2.2) that there are hardly any differences between the values of 2%, 3% and 4% shown.

In figure (2.3) the development of the booked premium at time 0 is plotted against the age. For policyholders aged between 15 and 30 the premium stays almost constant and then starts to increase exponentially. The increase of the premium is of exponential order due to the fact that the mortality rate is also increasing exponentially. Another fact that is not surprising is that the premium is the lower the higher the technical interest rate is, because the technical interest rate is used as a discount factor in (2.1) - (2.5). In figure (2.4) the present value of future profits (PVFP) at time 0 is plotted against the age. The PVFP is the higher the higher the entry age of the policyholder is, which is a counter intuitive relation at first sight. An analysis of the yearly cash flows for two different policyholders aged 15 and 70 reveals that this phenomenon is based on two different aspects as given in detail in table A.1.

1. When the guaranteed interest rate is roughly equal to the investment return or even higher then it is more advantageous for the insurance company when the policyholder is older and therefore dies earlier because

the difference between the guaranteed interest rate and the investment return need not to be financed over a long period. This leads to the observed fact that the PVFP is the lower the higher the technical interest rate is.

2. The differences of the first and second order assumptions of the mortality rates are in absolute values the bigger the higher the age is, because in almost all cases the second order assumptions of of the mortality rates are just a fixed fraction of the first order assumptions. This directly leads to a higher risk margin and therefore to a higher premium for elder persons in absolute values as shown in column *prem_diff* in table A.1. The higher premium overcompensates the higher death claims and therefore increases the surplus in absolute values and leads to a higher present value of future profits.

In figure (2.5) the value of the reserve is plotted against the time. In all three cases the reserve is zero at the beginning and then starts to increase. This is due to the fact, that the valuation date of the projection is Q1 but the begin month of the policy is later. We see that for the ages of 15 and 30 the difference in the reserve is negligible for all different values of the technical interest rate. The reserve for a 45 year old person is almost identical to the one of younger policyholders at the first 10 years of the endowment but then increases at a slightly slower rate. For a person aged 60 we get a different picture because the reserve is not always monotonically increasing as it is true for the younger policyholders. The reserve increases after approximately 10 years at a much slower rate compared to the other policyholders and even starts to decrease after roughly 20 years. This effect can again be explained by the much higher morality rates in absolute values for older policyholders as time increases. Due to that fact the reserve is at the end of the projection horizon when the maturity claims are paid out for an older policyholder approximately half of the size as for a younger policyholder.

2.2. Technical interest rate

The technical interest rate is one of the key assumptions in the life insurance business. It determines the factor by which the reserve and the savings premium increases during the contract period. After the contract has been concluded the technical interest rate is fixed and can't be changed by the insurance company. The maximum technical interest rate has been reduced

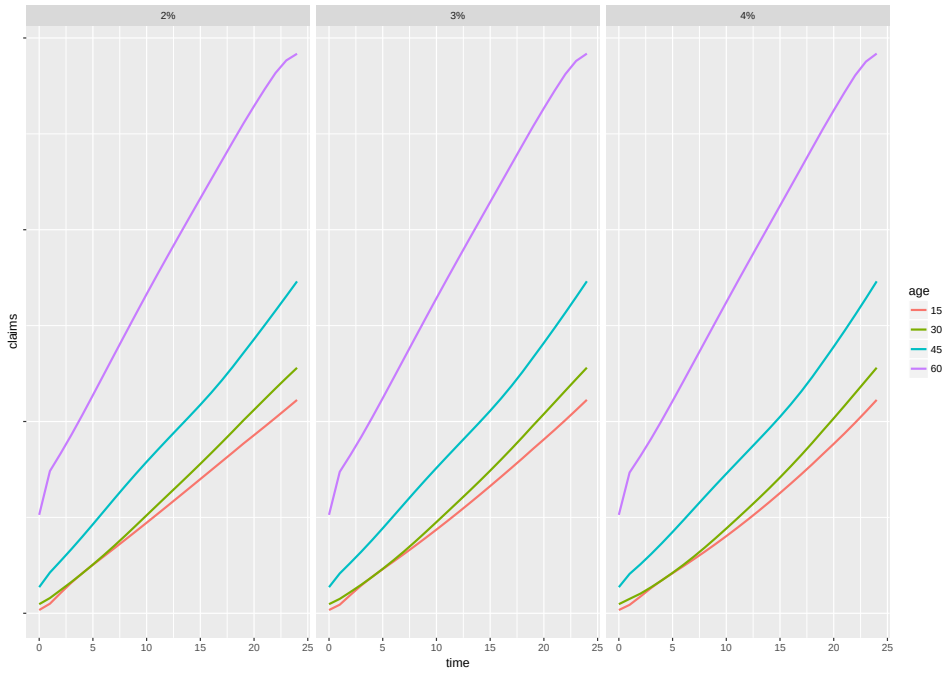


Figure 2.2.: Yearly cash flow for claims depending on the age and the technical interest rate.

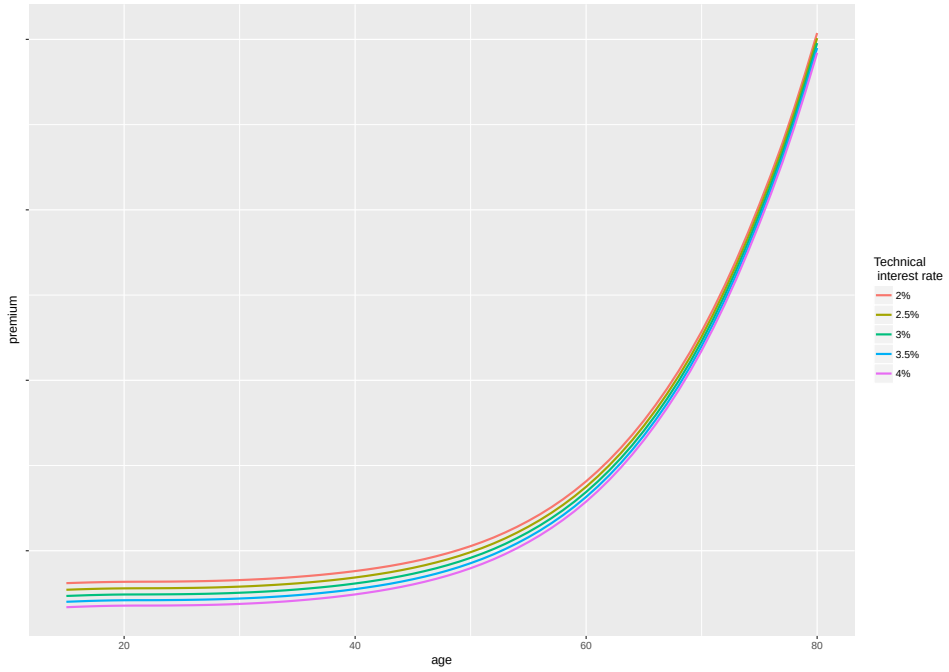


Figure 2.3.: Premiums depending on the age and the technical interest rate.

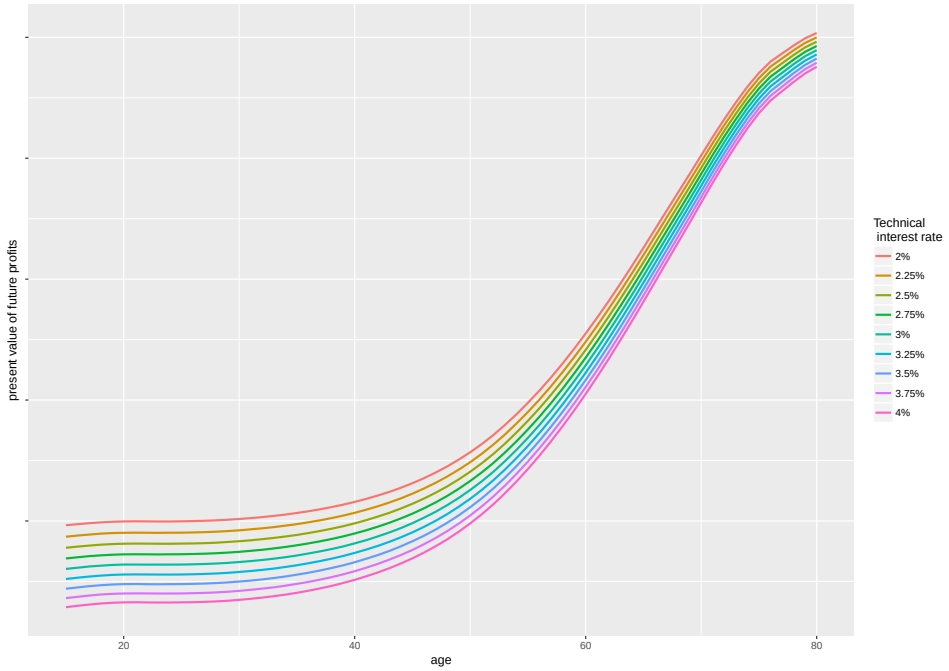


Figure 2.4.: Present value of future profits at time 0 depending on the age and the technical interest rate.

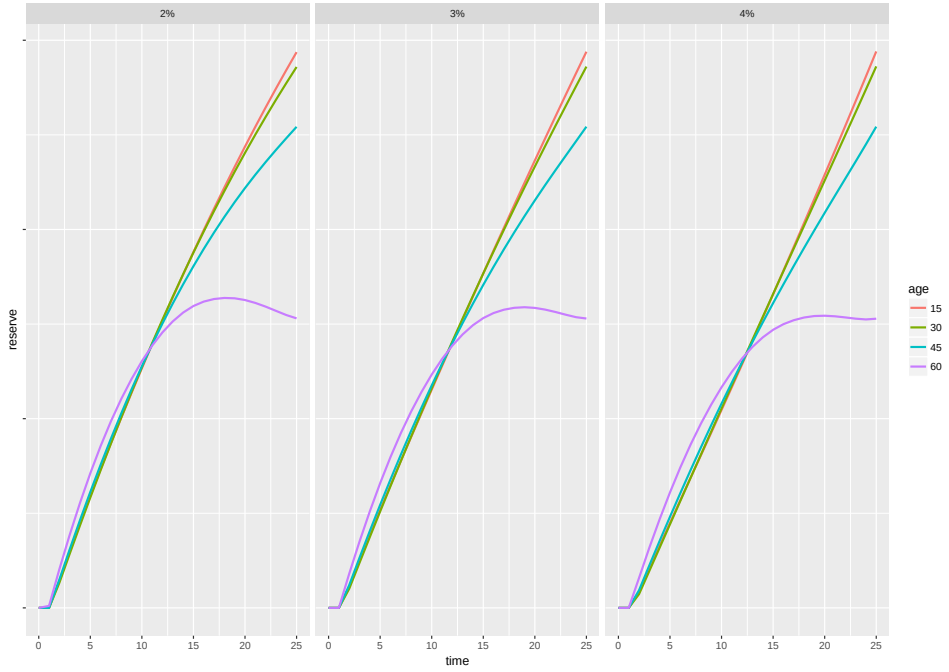


Figure 2.5.: Yearly reserve depending on the age and the technical interest rate.

dramatically by the Financial Market Authority in recent years from 4% to 0.5% as shown in table 2.1. The regulation of this upper bound is also relevant as it is often related to the minimum interest rate guaranteed to the policy holder. It is obvious that the higher the technical interest rate, the higher the guaranteed benefit or the lower the premium will be. When the grouping algorithm forces policies from different product generations with same payout characteristics but different technical interest rates to be grouped together one important thing to be aware of are sensitivities. Another important aspect which need to be taken care of when policies with different technical interest rates are grouped together is the one of consistent management rules. Take for example policy 1 with a technical interest rate of 2% and a bonus rate of 2% and policy 2 with a technical interest rate of 4% and a bonus rate of 0%. Lets assume that the two policies are similar and the grouped policy has a technical interest rate of 3% and a bonus rate of 1%. The total interest rate then is the same for the grouped and the ungrouped policies. Assume that the bonus rate is reduced by 1% caused by a management decision. Then policy 1 has only a bonus rate of 1% and policy 2 doesn't change at all, but the grouped policy has now a bonus rate of 0%. The total interest rate is not the same for the grouped and the ungrouped policies which can potentially have major impacts on the projected cash flows. When the technical interest rate increases, the present value of future profits as well as the premium will decrease as shown in figure (2.4) and (2.3) respectively. This behaviour is not surprising at all, because as the technical interest rate rises the insurance company guarantees a higher benefit and this yields ceteris paribus to a lower PVFP. The reserve and the claims are not that much affected by an increase of the technical interest rate as shown in figure (2.2) and (2.5) respectively.

2.3. Duration

The duration n of an insurance contract is next to the age and the technical interest rate another main characteristic which need to be taken care of when a grouping process is carried out. In figure (2.6) the yearly claims except the last claim, which is the maturity claim, are shown for different ages. One obvious conclusion that can be derived is that the sum of all sorts of claims except the maturity claim is getting the higher the higher the age is. Another observation that can be made is that for any given time t the claims are the higher the shorter the duration gets when we keep the age fixed. This is not surprising at all, because a shorter duration goes along with a higher premium (see figure

(2.7)) and a higher reserve (see figure (2.9)) which yields to higher claims for every fixed t. In figure (2.7) we see that the premium is decreasing with an exponential order when the duration is increased. The difference between the premiums for policyholders with different ages is indistinguishable small for short term contracts and is getting bigger as duration increases. In figure (2.8) the present value of future profits at time 0 is plotted against the duration of the contract. We see the same effect as in figure (2.4) where contracts with higher ages lead to a higher PVFP. The effect of the absolute difference between the first and second order mortality assumptions is getting the bigger the longer the duration is and therefore the PVFP is getting the higher the longer the duration is. In the last sensitivity chart (2.9) the reserve is plotted against the time for different values of x and n. We see that for every time t the reserve is the higher the shorter the duration is, because a shorter duration leads, ceteris paribus, to a higher premium which then results in a higher reserve. For short term contracts up to 10 years the reserve is approximately the same across different ages and is then getting the lower the higher the age and the longer the duration is. After the sensitivity analysis has already provided the first important insights from the data, the next chapter discusses the first potential algorithm that can be used for the grouping of policies.

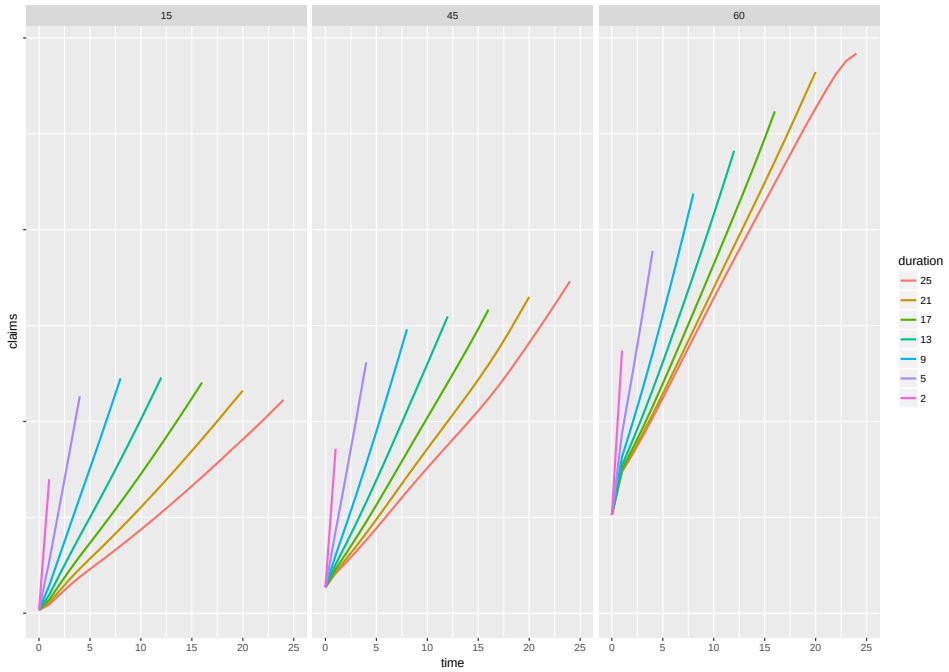


Figure 2.6.: Yearly cash flow for claims depending on the duration and the age.

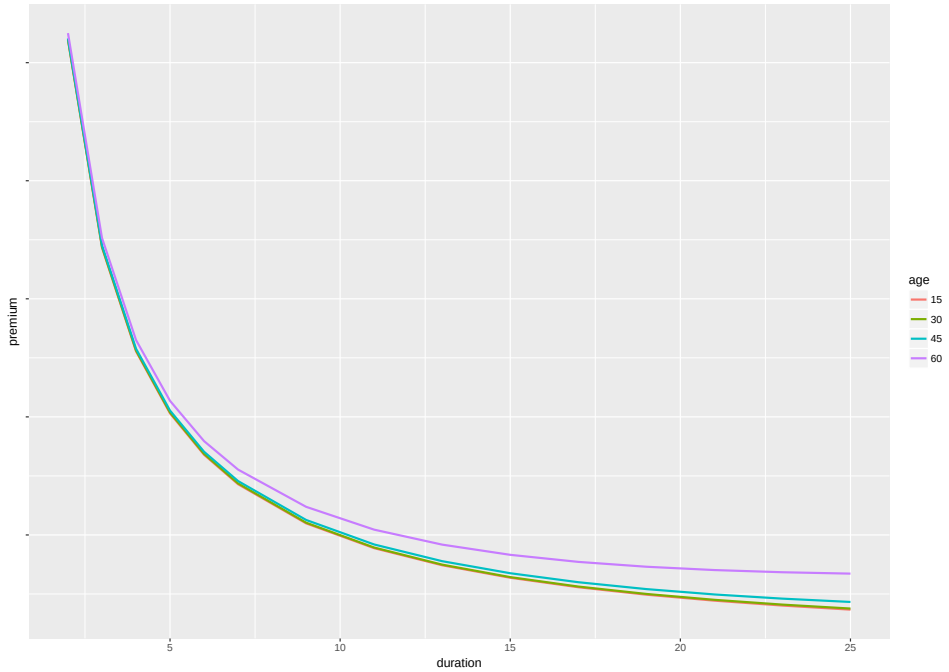


Figure 2.7.: Premiums depending on the duration and the age

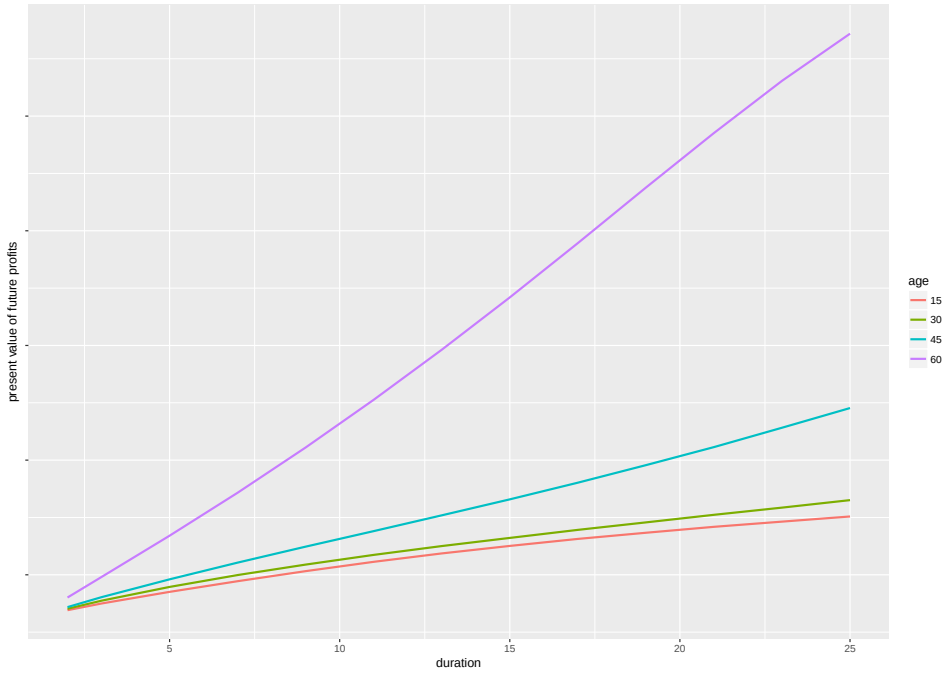


Figure 2.8.: Present value of future profits at time 0 depending on the duration and the age.

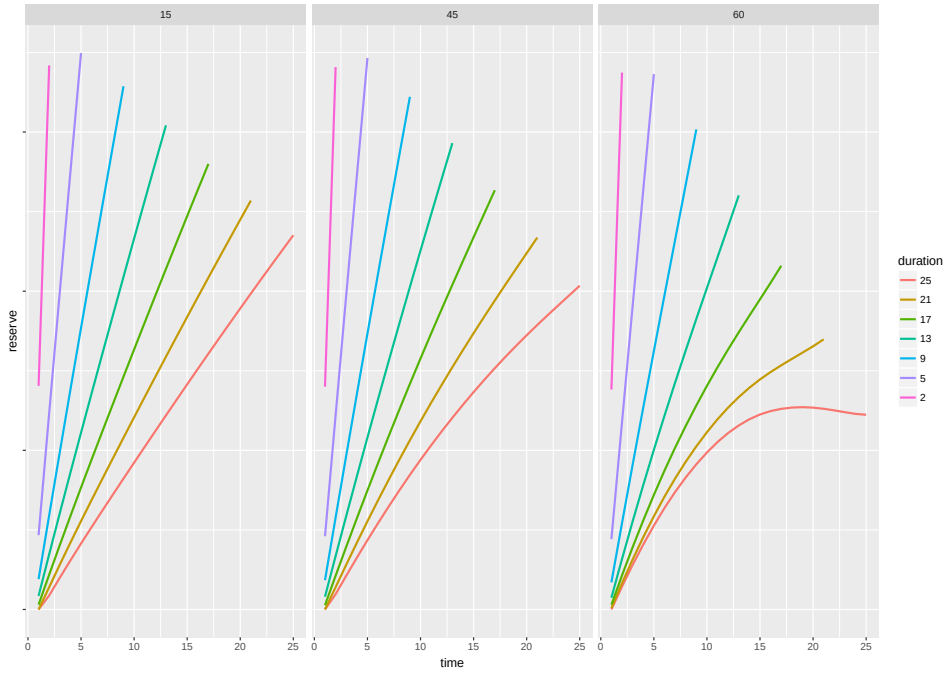


Figure 2.9.: Yearly reserve depending on the duration and the age.

3. k-means

To be able to process, summarize and understand huge amounts of data better, one is interested in methods that are able to find patterns in the data. The characteristics of these patterns then can be represented by just a few representative data points which behave like the whole data set. Given a data set the challenge is, based on a measure of similarity, to find groups of observations which are quite similar within each group but quite different to all the other groups. If this task has to be done with unlabelled data it is referred to as unsupervised clustering (c.f.[22]). One of the most widely used unsupervised clustering approaches is the k-means clustering. The k-means method is a simple approach in cluster analysis which splits a data set of n p-dimensional observations into k distinct clusters. Each observation belongs uniquely to exactly one of the k clusters, where k is a predefined number of clusters. Let $C = \{C_1, C_2 \ldots, C_k\}$ denote the sets containing the indices of the observations related to the clusters, then we get:

Definition 3.1. *Let $X = \{x_1, ..., x_n\}$ be a data set. X is said to be partitioned into k different clusters $C_1, C_2 \ldots, C_k$ if*

$$\textit{(i)}\ C = C_1 \cup C_2 \cdots \cup C_k = \{1, \ldots, n\}$$

$$\textit{(ii)}\ C_i \cap C_j = \emptyset \quad \forall i \neq j$$

The basic idea of the k-means clustering is to minimize the variation within the clusters. For this purpose some distance measure is needed in order to be able to define variation within clusters.

Definition 3.2. *Let X be a set, d: $X \times X \to \mathbb{R}$ a function. Then d is called a metric (or distance) on X if for all x, y, $z \in X$ the following conditions are fulfilled:*

(D_1) $d(x, y) = 0 \Leftrightarrow x = y$ *(Identity of indiscernibles)*

(D_2) $d(x, y) = d(y, x)$ *(symmetry)*

(D_3) $d(x, z) \leq d(x, y) + d(y, z)$ *(triangle inequality)*

Remark 3.1. *Given the axioms from definition 3.2 it can be shown that a non-negativity property can be deducted e.g.* $d(x,y) \geq 0 \ \forall x, y \in X$.

$$\begin{aligned} d(x,y) + d(y,x) &\geq d(x,x) \\ d(x,y) + d(x,y) &\geq d(x,x) \\ 2d(x,y) &\geq 0 \\ d(x,y) &\geq 0 \end{aligned}$$

Example 3.1. *(c.f. [25]) The most frequently used distance functions for two points* $x = (x_1, ..., x_p)$ *and* $y = (y_1, ..., y_p)$ *in* $\mathbb{R}^p$ *are:*

- ⋆ *Euclidean distance:*

$$d_2(x,y) := \sqrt{\sum_{j=1}^{p} (x_j - y_j)^2}$$

- ⋆ *Manhattan distance:*

$$d_1(x,y) := \sum_{j=1}^{p} |x_j - y_j|$$

- ⋆ *Chebyshev (maximum) distance:*

$$d_\infty(x,y) := \max_j |x_j - y_j|$$

- ⋆ *Minkowski distance (*L^q *distance) with* $q \geq 1$*:*

$$d_q(x,y) := \left(\sum_{j=1}^{p} |x_j - y_j|^q \right)^{\frac{1}{q}}$$

Typically the Euclidean distance is used as a measure of similarity to compute the distance between the different points. By using the Euclidean metric as a measure of similarity one assumes that the clusters are spherical.

Definition 3.3. *Let the Euclidean metric be the measure of similarity for the data points in the data set* $X = \{x_1, ..., x_n\}$ *and* p *the dimension of the data. Then the variation within one cluster* $C_l,\ l = 1, ..., k$ *is defined as:*

$$D(C_l) := \frac{1}{|C_l|} \sum_{j=1}^{p} \sum_{i,i' \in C_l} (x_{ij} - x_{i'j})^2$$

Remark 3.2. *For the average over one dimension in one cluster we use the short notation* $\bar{x}_{lj} = \frac{1}{|C_l|} \sum_{i \in C_l} x_{ij}$.

Remark 3.3. *The identity* $\sum_{i \in C_l} (x_{ij} - \bar{x}_{lj})^2 = \left(\sum_{i \in C_l} x_{ij}^2 \right) - |C_l| \bar{x}_{lj}^2$ *can be verified by simple calculus.*

Corollary 3.1. *The variation within one cluster,* $D(C_l)$ *can be written as:*

$$D(C_l) = 2 \sum_{i \in C_l} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{lj})^2 \tag{3.1}$$

Proof.

$$\begin{aligned}
D(C_l) &= \frac{1}{|C_l|} \sum_{j=1}^{p} \sum_{i,i' \in C_l} (x_{ij} - x_{i'j})^2 \\
&= \frac{1}{|C_l|} \sum_{j=1}^{p} \sum_{i,i' \in C_l} x_{ij}^2 - 2 x_{ij} x_{i'j} + x_{i'j}^2 \\
&= \sum_{j=1}^{p} \left(\frac{1}{|C_l|} \sum_{i,i' \in C_l} x_{ij}^2 - 2 \frac{1}{|C_l|} \sum_{i,i' \in C_l} x_{ij} x_{i'j} + \frac{1}{|C_l|} \sum_{i,i' \in C_l} x_{i'j}^2 \right) \\
&\overset{(Remark\ 3.2)}{=} \sum_{j=1}^{p} \left(\sum_{i \in C_l} x_{ij}^2 - 2 \bar{x}_{lj} \sum_{i \in C_l} x_{ij} + \sum_{i' \in C_l} x_{i'j}^2 \right) \\
&\overset{(Remark\ 3.2)}{=} 2 \sum_{j=1}^{p} \left(\sum_{i \in C_l} x_{ij}^2 - |C_l| \bar{x}_{lj}^2 \right) \\
&\overset{(Remark\ 3.3)}{=} 2 \sum_{i \in C_l} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{lj})^2
\end{aligned}$$

□

The approach of k-means is to find a partitioning of the data set in such a way that the sum of the variations is minimized.

Definition 3.4 (k-means)**.** *Let* $X = \{x_1, ..., x_n\}$ *be a data set and* $\{C_1, ..., C_k\}$ *a partition. Then k-means tries to find an optimal partition* $C^* = \{C_1^*, ..., C_k^*\}$ *such that:*

$$\sum_{l=1}^{k} D(C_l^*) = \min_{C_1, ..., C_k} \sum_{l=1}^{k} D(C_l) = \min_{C_1, ..., C_k} 2 \sum_{l=1}^{k} \sum_{i \in C_l} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{lj})^2 \tag{3.2}$$

When it comes to solving the optimization problem defined by formula (3.2), the computational complexity of the algorithm to be used is of high interest. Therefore computational complexity theory categorizes problems into different classes that have some defining properties, one of which is called NP-hard. The definition of these classes would go beyond the scope of this work and therefore only a reference to the literature is given here e.g. [39]. In very simplified terms one can say that there are currently no efficient algorithms for this type of problem.

Corollary 3.2. *Solving the k-means problem defined in (3.2) is NP-hard.*

Proof. c.f. [32] □

Even though the problem is NP-hard it is still possible to provide algorithms which converge to a local optimum. The algorithms used for finding a local minimum work on an iterative basis and involve just a few different steps. The k-means algorithms either start with an initial assignment of all observations to k different clusters or with k distinctly selected cluster centres. The next step is to find a new cluster centre such that the variation $D(C_l)$ is minimized within each cluster. Then all data points $X = \{x_1, ..., x_n\}$ are reassigned to the cluster which is nearest and the minimization procedure is repeated until convergence.

Remark 3.4.

(i) For $k = n$, (3.2) is zero because each data point represents a cluster.

One possible formulation for an algorithm that converges to a local optimum is the one from Lloyd [30] which is given below as pseudo code. Note that for algorithm 2 we have a fixed number of clusters k as well as a finite set of possible partitions k^n. We can therefore show that the stated algorithm converges to a local minimum by minimizing (3.1).

Remark 3.5.

(i) For any set of observations S it holds that:

$$\bar{x}_S = \underset{x}{argmin} \sum_{i \in S} (x_i - x)^2$$

Hence, step 2a) minimizes the sum of squared deviations and therefore $D(C_l)$.

*(ii) Step 2*b*) which reassigns the observations to the new nearest centroid can only reduce the objective function.*

Algorithm 2 k-means clustering [23] - Lloyd's algorithm

1. Choose k initial centroids randomly.
2. Iterate till the cluster assignments stop changing:
   a) For each of the k clusters, compute the cluster centroid. The i-th cluster centroid is the vector of the p parameter means for the observations in the ith cluster.
   b) Assign each observation to the cluster whose centroid is closest in the sense of Euclidean distance.

Remark 3.6.

(i) The k-means algorithm always converges and finds a local optimum which need not to be the global one.

(ii) Different clustering results can be obtained when different initial cluster assignments in step 1 of algorithm 2 are chosen.

Remark 3.7 (Running Time)**.**

(i) Algorithm 2 has a running time of $O(nkpi)$, with n being the number of data points, k the number of cluster, p the number of dimensions and i the number of iterations needed to converge. The only unknown variable is the number of iterations.

(ii) The trivial upper bound for the number of iterations needed is given by $O(k^n)$, because the algorithm visits every partition of points only once.

(iii) It can be shown ([3]) that in the worst case scenario the runtime of the algorithm is superpolynomial with a lower bound for the number of iterations of $O(2^{\Omega(\sqrt{n})})$. This means that the runtime is not bounded from above by any polynomial function.

(iv) In practice, the number of iterations required is often small, which makes the algorithm appear to be linearly complex.

It is advisable to run the algorithm several times with different initial centroid assignments. This increases the likelihood of finding a partition that is close to the optimum and thus provides a low value for the objective function

(3.2). However, the biggest challenge when using k-means is the estimation of the optimal number of clusters k. Figure (3.1) shows an example with three clusters that illustrates the different clustering results when the number of k increases. Panel (a) shows the raw data with three clusters ($k_{true} = 3$), each generated by an uniform distribution. Panels (b), (c) and (d) show the cluster results with $k = 2, 3$ and 7, respectively. If the number of clusters k is smaller than the actual number of clusters in the data, then k-means merges clusters, while for k greater than k_{true}, k-means divides well separated clusters. Panel (b) shows a merge of the clusters at the top right, whereas panel (d) shows an artificial split of two natural clusters for $k = 5$. For the grouping of similar data points and their representation by a cluster centre, an underestimation of the actual number of clusters is more critical than an overestimation. If k underestimates the true value of clusters ($k < k_{true}$), then it's not possible to capture the cluster specific characteristics for the merged clusters because they are represented by only one cluster centre. However, an overestimation of k_{true} is not so critical because some natural clusters will be represented by two cluster centres which has a negative impact on the compression ratio but not the clustering quality. For the most crucial step of k-means, a comprehensive collection of methods for estimating the correct number of clusters can be found in [34]. In the following, the 'elbow' and silhouette method, two widely used graphical methods for estimating k_{true}, are presented and applied to the sample data set.

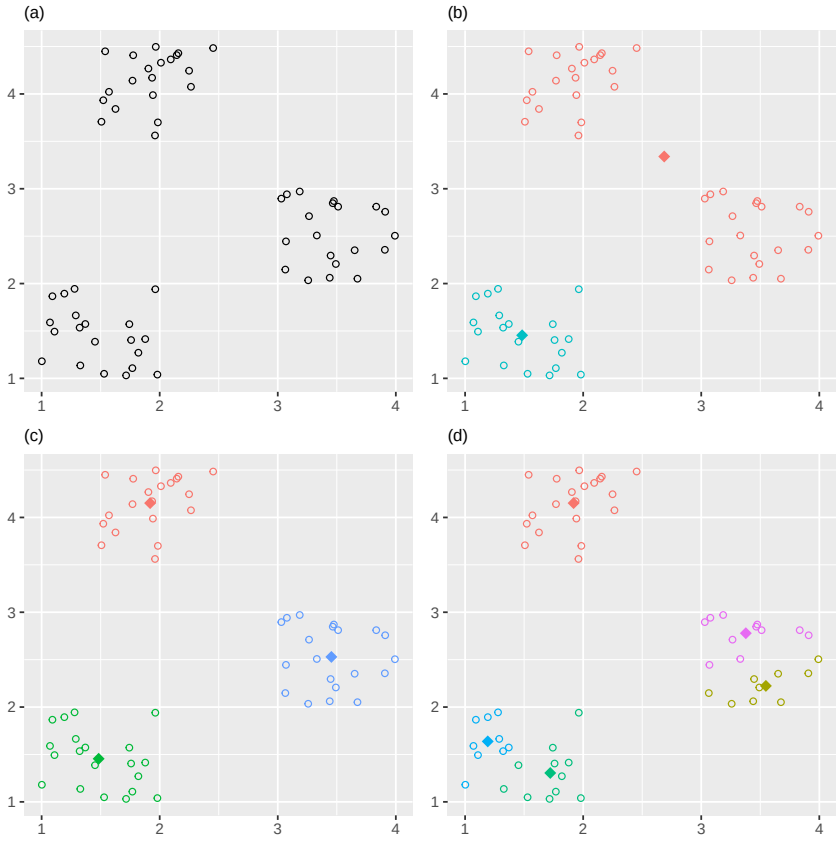


Figure 3.1.: Results for a three-cluster example: (a) raw data; (b) $k = 2$; (c) $k = 3$; (d) $k = 5$

3.1. Elbow method - gap statistic

Estimating k, the parameter that defines the number of clusters, is one of the most difficult tasks when using k-means. While it is still possible to graphically determine the number of clusters by plotting the data in 2 or 3 dimensional spaces, other methods must be used in higher dimensions. The 'elbow' method is one of the most commonly used methods in this context.

Definition 3.5. *Let $X = \{x_1, ..., x_n\}$ be a data set and k the number of clusters. Then $C = \{C_1, ..., C_k\}$ is the corresponding partition with the sum of variation within all clusters defined as:*

$$V_k := \sum_{r=1}^{k} \frac{D(C_r)}{2}$$

Plotting V_k, the sum of variation within all clusters as a measure of total error versus the number of clusters used gives a good indication for the true value of k. Figure 3.2(a) shows that the error measure V_k decreases monotonically as the number of clusters increases but there seems to exist a k from which the decline is clearly flattened. Such an 'elbow' indicates that any additional cluster reduces the total variation V_k only slightly and an appropriate number of clusters can be derived from the location of the 'elbow'. In accordance with $k_{true} = 3$, the 'elbow' plotted in 3.2(a) indicates that the true number of clusters is three, because there the curve starts to flatten dramatically. Even if the sample data set consists of very well separated clusters, the example indicates that the method could also be suitable in general use cases. For a small number of tasks the graphical determination of the "elbow" is practicable, but an automated method is needed with an increasing number of clustering operations. A statistical method that formalizes this procedure of finding the 'elbow' is described in [48]. The basic idea is to make the sum of variation V_k comparable to a reference. For this purpose, the sum of deviations is calculated for each k and then compared with the expected sum of the deviations derived from a reference data set with no obvious clustering. The reference data set is generated by sampling uniformly over the range of the observed values for every feature from the original data set. This means it is sampled uniformly from the smallest p-dimensional cube that contains all data points $\{x_1, ..., x_n\}$ of the original data set.

Definition 3.6. *Let k be the number of clusters and $\mathbb{E}_n$ the expectation under a sample of size n from the reference distribution. Then the gap-statistic is*

defined as:

$$Gap_n(k) := \mathbb{E}_n\big[log(V_k)\big] - log(V_k) \tag{3.3}$$

To determine the expected value $\mathbb{E}_n\big[log(V_k)\big]$ we draw B different samples $\{x_1^*, ..., x_n^*\}$ from the p-dimensional cube and average over the B values of $log(V_k)$. Accounting for the simulation error introduced by using the B Monte Carlo samples the standard deviation is given by:

$$s_k = \sqrt{1 + \frac{1}{B}} sd(k)$$

The optimal cluster size k is then determined by the following rule which identifies the 'elbow':

Definition 3.7. *Let $Gap(k)$ be the statistic defined above and s_k the standard error. Then the rule for choosing k is given by: (c.f. [48])*

$$\hat{k} := \textit{smallest } k \textit{ such that } Gap(k) \geq Gap(k+1) - s_{k+1} \tag{3.4}$$

In order to find the optimal number of clusters in a computational way, the following steps described in algorithm 3 are necessary:

Algorithm 3 Elbow method [48]

1. Define the maximal number of clusters k_{max}
2. Iterate over $k = 1, ..., k_{max}$:
   a) Apply the k-means algorithm to the data set $\{x_1, ..., x_n\}$ and calculate $log(V_k)$.
   b) Generate B reference data sets, each of them having n samples $\{x_1^*, ..., x_n^*\}$.
   c) Cluster each of those B data sets and calculate $\mathbb{E}_n\big[log(V_k)\big] = \frac{1}{B}\sum_{b=1}^{B} log(V_{kb})$
   d) Compute the standard deviation:

$$s_k(k) = \sqrt{1 + \frac{1}{B}}\left(\frac{1}{B}\sum_{b=1}^{B}(log(V_{kb}) - \mathbb{E}_n\big[log(V_k)\big])^2\right)^{\frac{1}{2}}$$

   e) End the loop if: $Gap(k) \geq Gap(k+1) - s_{k+1}$

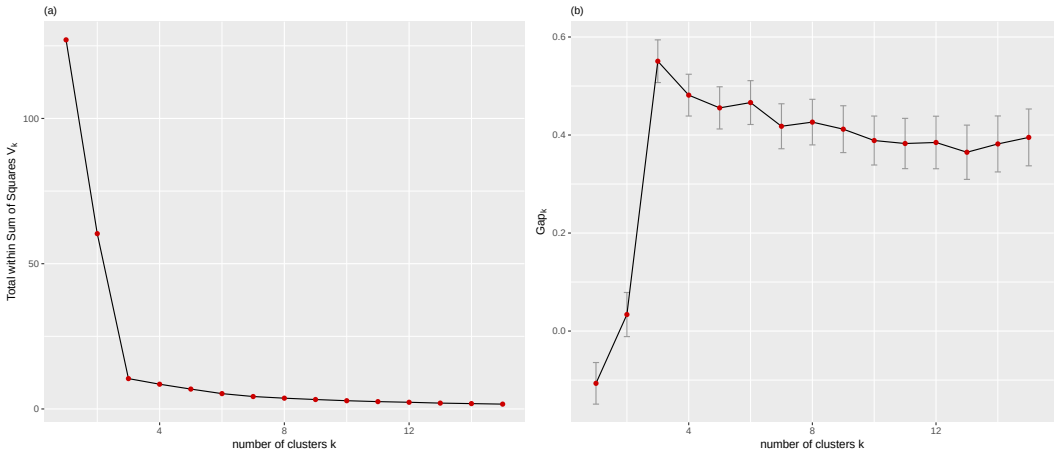


Figure 3.2.: Three-cluster example: (a) Total within Sum of Squares; (b) Gap-Statistic

Figure (3.2)(b) shows the gap-statistic for different values of k with the corresponding standard errors as a vertical bar. The application of the proposed rule (3.4) for selecting the number of clusters results in $\hat{k} = 3$ which corresponds exactly to the actual number of clusters in the data set.

3.2. Silhouette method

Although the "elbow method" is well suited in many cases to determine the number of clusters, the resulting partitioning is not visually displayable if the dimension is larger than three. A visually appealing graphical display called silhouette plot introduced in [44] tries to overcome this shortcoming in order to be able to interpret cluster results more properly. The plot shows whether a specific partitioning result reflects a cluster structure actually present in the data set or not, by comparing the within dissimilarity with the between dissimilarity for every data point. It can thus be determined how similar a data point is to its own cluster compared to the other clusters. The measure of similarity can be calculated with any distance metric appropriate to the specific problem and will be called $dist(x_i, x_j)$ for two data points x_i and x_j.

Definition 3.8 (Within dissimilarity)**.** *Let $X = \{x_1, ..., x_n\}$ be a data set and $C = \{C_1, ..., C_k\}$ the corresponding partitioning with k cluster. Assume that the data point x_i is assigned to cluster C_i with $C_i \in C$ and $1 \leq i \leq n$. Then the*

average dissimilarity of x_i to all other objects assigned to cluster C_i is defined by:

$$WD(C_i, i) := \frac{1}{|C_i|} \sum_{x_l \in C_i} dist(x_i, x_l)$$

One can think of $WD(C_i, i)$ as a measure of how well the data point x_i is embedded in its cluster C_i. The smaller the value, the closer the data points of the cluster are to each other on average.

Definition 3.9 (Between dissimilarity)**.** *Let $X = \{x_1, ..., x_n\}$ be a data set and $C = \{C_1, ..., C_k\}$ the corresponding partitioning with k cluster. For any cluster C_j different from C_i (i.e. $i \neq j$) the average dissimilarity of $x_i \in C_i$ to the cluster C_j is given by:*

$$BD(C_j, i) := \frac{1}{|C_j|} \sum_{x_l \in C_j} dist(x_i, x_l)$$

One can see that $BD(C_j, i)$ is the average distance from data point x_i in cluster C_i to all data points x_j in cluster C_j.

Definition 3.10. *Let $X = \{x_1, ..., x_n\}$ be a data set and $C = \{C_1, ..., C_k\}$ the corresponding partitioning with k cluster. For any data point x_i assigned to cluster C_i (e.i. $x_i \in C_i$) the distance to the closest neighbor cluster is given by:*

$$dist(i) := \min_{C_i \neq C_j} BD(C_j, i)$$

Cluster C_b with $1 \leq b \leq k$, which shares the smallest average dissimilarity with point $x_i \in C_i$ is called the neighbor cluster of x_i. This neighbor cluster is the first choice if cluster C_i is removed from our analysis and we have to reassign x_i to a new cluster. After computing $WD(C_i, i)$ and $dist(i)$ for every data point $x_i,\ i = 1, ..., n$ one can define the silhouette statistic $s(i)$ as follows.

Definition 3.11. *Let $X = \{x_1, ..., x_n\}$ be a data set and $C = \{C_1, ..., C_k\}$ the corresponding partitioning with k cluster. For every $x_i \in X$ the silhouette $s(i)$ is defined as:*

$$s(i) = \begin{cases} 1 - \frac{WD(C_i,i)}{dist(i)} & \text{if } WD(C_i, i) < dist(i) \\ 0 & \text{if } WD(C_i, i) = dist(i) \\ \frac{dist(i)}{WD(C_i,i)} - 1 & \text{if } WD(C_i, i) > dist(i) \end{cases}$$

Remark 3.8. *We can write the silhouette $s(i)$ in a more compact form:*

$$s(i) = \frac{dist(i) - WD(C_i, i)}{max\{WD(C_i, i), dist(i)\}}, \qquad \text{if } |C_i| > 1$$
$$s(i) = 0 \qquad \text{if } |C_i| = 1$$

Remark 3.9. *For every x_i it is true that $-1 \leq s(i) \leq 1$.*

Remark 3.10. *In order to understand which values of $s(i)$ correspond to which clustering results, it makes sense to look at values at the boundaries:*

- $s(i) \in (1-\epsilon, 1]$*: For $s(i)$ close to 1 the within dissimilarity of x_i is much smaller than the minimum between dissimilarity. This shows on the one hand that the data point x_i is very well embedded in its cluster and has a small distance to the other data points of its cluster (i.e. $WD(C_i, i)$ small). On the other hand a relatively large value of $dist(i)$ shows that the minimum distance from x_i to the nearest cluster is very large. Thus one can speak of an adequate clustering result.*
- $s(i) \in (-\epsilon, +\epsilon)$*: The within dissimilarity of x_i is approximately the same as the minimum between dissimilarity which indicates that x_i lies between two cluster. A small change of the data point x_i could cause it to be assigned to another cluster, suggesting unstable results.*
- $s(i) \in [-1, -1+\epsilon)$ *The minimum between dissimilarity of x_i is much smaller than the within dissimilarity which indicates that x_i may not be correctly clustered. In this case x_i is on average much closer to the neighbor cluster than to the actual cluster which rises doubts if the cluster assignment is correct.*

Remark 3.11. *Let x_i be a data point which is assigned to cluster C_i and cluster C_j the neighbor cluster of x_i. If x_i is reassigned from cluster C_i to cluster C_j then $s(i)$ becomes $-s(i)$.*

In order to get a visually appealing overview if a clustering result is good or not all silhouettes are plotted on top of each other. Silhouettes which belong to the same cluster are plotted together and ranked in decreasing order.

Figure (3.3) shows the silhouette plot for the sample data set with $k = 3$ and 5, respectively. Each silhouette plot shows on the right side the cluster name, the number of data points assigned to the cluster and the average silhouette width. In the right panel of figure (3.3) cluster 1 consists of 11 data points

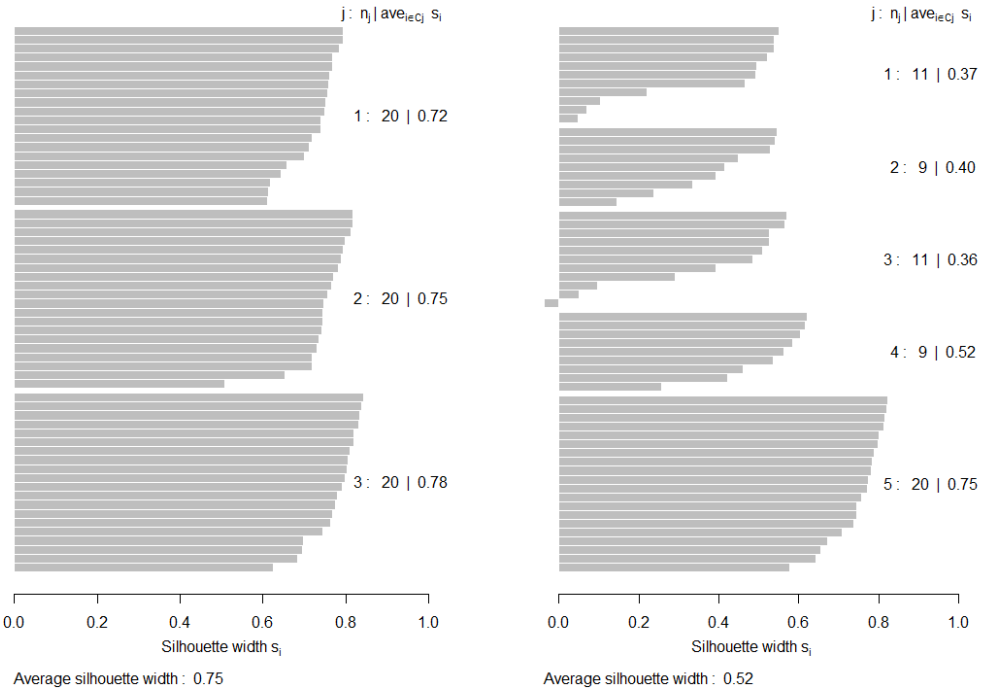


Figure 3.3.: Left: silhouette plot for $k = 3$; Right: silhouette plot for $k = 5$

and has a average silhouette width of 0.37. In cluster 3, for example, two data points have a silhouette width $s(i)$ close to zero and one data point has a negative value for $s(i)$. Except for cluster 5, all other clusters include data points which have a small or even negative value for $s(i)$. At the bottom of each silhouette plot the average silhouette width for all data points is given. The silhouette plot with $k = 3$ has much wider silhouettes compared to the plot with $k = 5$ which is an indicator that only 3 clusters are present in the data. Similar plots for $k = 2$ and $k = 4$ also lead to the conclusion that the data consists of 3 natural clusters. If there are too many ($k > k_{true}$) or too few clusters ($k < k_{true}$) some of the cluster silhouettes will be much narrower compared to the others.

Remark 3.12. *A high average cluster silhouette width indicates that the cluster is well separated from other clusters and is not split up artificially.*

Remark 3.13. *It can be seen that artificial splits of clusters gets quite heavily penalizes by the silhouette coefficient. The average silhouette width drops from 0.75 to 0.52 if the number of cluster is increased from $k = 3$ to $k = 5$ as seen in figure (3.3).*

3.3. Curse of dimensionality

After presenting two methods namely the elbow method and the silhouette method which help identifying the correct number of clusters one should also be aware of a phenomena that arise when analyzing data in high dimensional spaces. The 'curse of dimensionality' is a term introduced by Richard Bellman [4] to describe the rapid increase in volume and therefore the intractability of algorithms, when adding more dimensions of data to a mathematical space. Nowadays there are many different phenomenons referred to when talking about the curse of dimensionality but in the subsequent the focus is on distance functions as a measure of similarity. So far, all the methods presented are based extensively on the underlying distance functions used. To get a better understanding on the issue a simple example is given that helps to illustrate the problem.

Example 3.2.

(i) Imagine a line segment of length 1, and 10 data points which should represent the line. To capture the whole line segment one would distribute the points uniformly across the line. Therefore the line would be divided

into 10 segments with length $\frac{1}{10}$ and the points are centered within these segments.

(ii) By adding one dimension the line segment becomes a square segment with edge length 1. In order to represent the 'same' space with a data point as in the one-dimensional case, the square would have to be divided into 100 smaller squares with an edge length of $\frac{1}{10}$ each. In the centre of each of those square segments one data point is needed as a representative. A total of 100 data points are required to represent the square segment as exactly as the line segment.

(iii) By adding another dimension the square segment becomes a cube and 1000 points are needed.

Example 3.2 illustrates that as the number of dimensions increases the number of data points rises exponentially in order to represent the whole space properly. Thinking of insurance data, one can have data points with various numbers of dimensions ranging from just a few to several hundreds dimensions. Considering the fact that the cash flow projections of grouped policies should coincide over a 60 year horizon, it is easy to see that only 5 of these cash flow characteristics (e.g. premium, costs, ...) result in data points with a dimension of 300. Of course, it is not advisable to include, over a period of 60 years, all cash flow variables in the grouping process, but it is much more difficult to find only those variables that are relevant for a major part of the results than including everything. For this reason, in practice more variables are often used than would actually be necessary. Another unintuitive fact one needs to be aware of is that the volume of a hypersphere inscribed into a hypercube is getting relatively smaller as the dimension increases.

Theorem 3.1. *Let a hypersphere with radius r and dimension d be inscribed into a hypercube with edges of length $2r$. Then we get for the proportion of the volumes:*

$$\frac{V_{Sphere}}{V_{Cube}} = \frac{r^d \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)}}{(2r)^d} = \frac{\pi^{\frac{d}{2}}}{2^d \Gamma(\frac{d}{2}+1)} \to 0 \text{ as } d \to \infty$$

Remark 3.14. *Corollary 3.1 says that as dimension d increases, more and more volume of the hypercube is outside the hypersphere. This means that under a uniform distribution most of the data points are located far from the centre and thus close to an edge in a certain sense.*

Remark 3.15. *Some other examples why intuition fails in high dimensions are given in paragraph 6 of [12].*

Not only do the data points move closer to the edge as the dimension d of the data space increases, but the distance between the individual data points is becoming more and more similar. It can be shown ([5]) that under a broad set of conditions the distance to the nearest and to the farthest data point converges as dimensionality d increases. Experimental results in [5] show that this effect can occur even for relatively low dimensional data with only 10 to 15 dimensions. The fact that the distance to the farthest data point can get similar to the distance of the nearest data point makes clustering a hard job. The basic concept of k-means is to find, in the case of the Euclidean distance measure, spherically shaped clusters that have different characteristics. Therefore, data that is almost identical should not be clustered with a simple k-means algorithm without further analysis. Due to the fact that the k-means algorithm always returns a result even though there are obviously no clusters in the data because all data points are somehow similar, it is very difficult to determine whether k-means is a suitable tool for clustering or not. Situations in which all data points are similar and no cluster structure is present in the data are, as shown in the previous section, indicated by a low silhouette coefficient. When using the methods described in section 3.2, a silhouette plot with low silhouette coefficients for the clusters indicates that the clusters found by the algorithm are not well separated. Single clustering attempts can easily be verified by an visual inspection of plots described above. With an automated clustering approach, which is necessary for large insurance portfolios, visual control of the individual silhouette plots is not possible in most cases. In such cases, only a validation of the silhouette coefficient is feasible, but this leads ultimately to a situation where a clustering result is validated by a single value. It is therefore advisable to use low-dimensional policy data sets or conduct a thorough analysis of the data to avoid situations where problems referred to as 'curse of dimensionality' occur.

4. Non-negative least squares (NNLS)

Another way to find a grouped portfolio that minimizes the deviation defined in formula (1.1) is to solve the problem using mathematical optimization methods. The goal is to find a subset of the portfolio and scale it in a way so that the square deviation becomes minimal. Of course, this approach must ensure that scaling is only possible in the positive direction. This means that it makes no sense to have a negative policy in the grouped portfolio, because it cannot be defined, not to mention explained logically.

Definition 4.1 (Non-negative least squares)**.** *Let $P \subset \mathcal{V}$ be a portfolio, $A \in \mathbb{R}^{m \times n}$ the matrix with the corresponding cash flows and $b \in \mathbb{R}^m$ the vector with the summed cash flows. Then $x \in \mathbb{R}^n$, the vector of scaling, should be optimized such that:*

$$\begin{aligned} &\arg\min_x \|Ax - b\|_2^2 \\ &\textit{subject to } x \geq 0 \end{aligned} \tag{4.1}$$

Remark 4.1.

- *The entries of the vector x are the so-called scaling values for the cash flows in matrix A. Each entry $x_i, i \in \{1, ..., n\}$ that is greater than zero scales to the i-th column of matrix A where column i represents the cash flows of the i-th policy of the portfolio.*
- *It should be noted that this approach optimizes cash flows, not policies. Since there is a one-to-one relationship between cash flows and policies, the scaling factors cannot only be used to scale the cash flows but also to scale the policies in order to create a grouped portfolio.*
- *Scaling policies can also involve risks, as a policy that is scaled by a factor of 2 will not necessarily produce cash flows that are also increased by a factor of 2. This can be attributed to the fact that, for example, non-linearities occur due to discount effects for higher premiums in the tariff.*

Remark 4.2. *The number of policies in the grouped portfolio corresponds exactly to the number of entries greater than zero in the vector x.*

One of the well known algorithms for solving the non-negative least squares problem is that of Lawson and Hanson, which uses an active set method. The steps necessary for solving that problem are given in [29]. Additionally to those parameters defined in definition 4.1 one also needs a real value variable ϵ as a stopping criterion.

Algorithm 4 Non-negative least squares [29]

1. Set $P = \emptyset$, $R = \{1, ..., n\}$, $x = 0_{n\times 1}$
2. Compute $w = A^\top(b - Ax)$.
3. While $R \neq \emptyset$ and $max(w) > \epsilon$
   - *a)* Find index $j \in R$ such that $w_j = max\{w_t, t \in R\}$.
   - *b)* Move the index j from R to P.
   - *c)* Let A^P be A restricted to the variables included in P.
   - *d)* Let s be a vector of same length as x. Let s^P denote the sub-vector with indexes from P, and let s^R denote the sub-vector with indexes from R.
   - *e)* Compute $s^P = ((A^P)^\top A^P)^{-1}(A^P)^\top b$
   - *f)* Set $s^R = 0$.
   - *g)* While $min(s^P) \leq 0$
     - i. Set $\alpha_k = min\frac{x_i}{x_i - s_i}$ for i in P where $s_i \leq 0$
     - ii. Set $x = x + \alpha_k(s - x)$
     - iii. Move from P to R all indices $k \in P$ for which $x_k = 0$.
     - iv. Compute $s^P = ((A^P)^\top A^P)^{-1}(A^P)^\top b$
     - v. Set $s^R = 0$
   - *h)* Set x to s
   - *i)* Compute $w = A^\top(b - Ax)$.

Algorithm 4 consists, apart from the initialization, of a main loop and an inner loop. The loops are highlighted by indentations and start at step 3 and step g) respectively. If for a variable reference is made to those indices of the variable which are contained in the set R, then these entries are 0. All indexes that exist in the set P, by contrast, have non-zero values. If such a variable has a negative value, the algorithm either moves it to the positive value range or

sets it to zero. By setting a variable to zero, the index is also shifted from the set P to the set R. This ensures that the following condition is met at the end of the algorithm.

$$\begin{aligned} x_j &> 0, \quad j \in P \\ x_j &= 0, \quad j \in R \end{aligned} \tag{4.2}$$

Remark 4.3. *Let $f(x) = \|Ax - b\|_2^2$, then the gradient of $f(x)$ is given by:*

$$\nabla f(x) = \nabla \|Ax - b\|_2^2 = A^\top (Ax - b)$$

Proof.

$$\begin{aligned} \nabla \|Ax - b\|_2^2 &= \nabla (Ax - b)^\top (Ax - b) \\ &= \nabla (x^\top A^\top - b^\top)(Ax - b) \\ &= \nabla (x^\top A^\top Ax - x^\top A^\top b - b^\top Ax + b^\top b) \\ &= \nabla (x^\top A^\top Ax - 2x^\top A^\top b + b^\top b) \\ &= 2(A^\top Ax - A^\top b) \\ &= 2A^\top (Ax - b) \end{aligned}$$

□

As shown in remark 4.3, step 2 of algorithm 4 calculates the negative gradient of the ordinary least squares problem. In the next step it is checked whether the inner loop still has to be executed or not. If the index set R correspond to the empty set then all indexes are already in P which means that all entries of x are positive (see formula (4.2)). This case is not desirable, as no compression can be achieved. If $max(w) \leq \epsilon$ is satisfied, the gradient has no entry large enough so that an substantial improvement can be achieved and the algorithm has reached the optimum. If one of the two conditions in step 3) is fulfilled, the main loop is executed.

The main loop starts with searching for the index of the gradient that is not yet present in the set P and has the highest value. After this index has been moved from the set R to the set P, the solution of a restricted least squares problem is calculated in step e). This least squares problem is limited to the columns of the matrix A whose indices occur in the set P. The result is then a vector of dimension P which is indicated by the notation s^P. To get a solution vector of the dimension n, the $n - |P| = |R|$ entries s^R of the vector s are filled with zeros - see step f).

Example 4.1. *Based on algorithm 4, be* $n = 15$, $P = \{2, 3, 4, 5, 7, 9\}$ *and* $R = P^c = \{1, 6, 8, 10, 11, 12, 13, 14, 15\}$ *then*

$$s = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_{15} \end{pmatrix} \qquad s^P = \begin{pmatrix} s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_7 \\ s_9 \end{pmatrix} \qquad s^R = \begin{pmatrix} s_1 \\ s_6 \\ s_8 \\ s_{10} \\ \vdots \\ s_{15} \end{pmatrix}$$

If all components of this least squares solution (i.e. s^P) are positive a new solution has been found and the solution vector x is overwritten with s as seen in step h). In step i) a recalculation of the gradient with the solution x, adapted in the previous step, is carried out and the main loop is restarted by checking the conditions in step 3.

If there are non positive entries in the solution of the restricted least squares problem computed in step e), the inner loop is executed. Basically, negative entries in the result vector s^P would lead to a new solution vector x which would also have negative entries without further adjustments, resulting in an undesired solution since x must be non-negative (i.e $x \geq 0$). Therefore, based on the current solution vector x which has only positive entries, a shift is carried out by using s^P. In the first step the index $k \in P$ is determined which leads to the smallest scaling factor α_k. The current solution vector x is then shifted by using this factor which causes the entry x_k to be zero after the shift. The index k can thus be moved from set P back to set R because the corresponding entry x_k is zero after the shift. In the unlikely event that several entries of the new solution vector x were changed to zero as a result of the shift, all these indexes must of course be moved from set P to set R (see iii.). Based on the new set P, a new limited least squares problem will be solved and the inner loop will be redone if necessary.

Remark 4.4. *Since* $x_i \geq 0$ *at all times and* $s_j \leq 0$ *within the inner loop, the scaling factor is bounded by* $0 \leq \alpha_k \leq 1$.

Remark 4.5. *Within the inner loop, at least one index* k *per iteration is transferred from the set* P *to the set* R. *As a result, with each pass of the inner loop, the number of entries in the solution vector* x *that are not equal to zero is reduced by at least one. Since the cardinality of the set* P *is finite, it is also determined how often the inner loop is run through at the most, namely* $|P| - 1$ *times.*

As stated in [29, p. 163], for many examples the steps of the outer loop are simply repeated by adding another positive coefficient to the solution vector x until one of the termination criteria in step 3 are fulfilled. This observation may be correct for many different application areas, but must be verified with respect to the policy data being optimized. For this purpose, a standard portfolio of policies is used in which the development of the vector x is analysed. The test portfolio consists of 20.000 randomly selected policies using 351 cash flows for each policy.

Figure (4.1) shows how the number of non-negative entries of the solution vector x evolves. After the algorithm is started with an initialization, the solution vector x only has entries that are zero and the next step is to execute the main loop for the first time. So the graph starts in (0,0). After passing through the main loop for the first time, the first entry of x became a positive value and the graph displays this as the point (1,1). Also the next 9 passes of the main loop run exactly as just described so that after 10 iterations 10 entries of x are not equal to zero and the graph is at (10,10). In the next pass of the main loop, the solution s^P, see step e), of the restricted least squares problem becomes for the first time not positive for at least one entry and the inner loop is executed. When entering the inner loop, s^P has the dimension 11. Within the inner loop, as explained above, a shift is performed and the index k is shifted from the set P to the set R. Due to this shift of the index and the subsequent recalculation of the now dimensionally reduced least squares problem (see iv.), s^P now has only dimension 10. Since none of these 10 entries is not positive, the inner loop is now left. The solution vector still has only 10 positive entries after the 11-th iteration and therefore the point (11,10) results. The just described phenomenon that the inner loop is left again after one iteration, since all negative entries have been removed, can be observed in figure (4.1) more often. This is exactly the case when a horizontal line is present. In the above example, it is also easy to see that this phenomenon of horizontal lines occurs more frequently after about 50 iterations. This can be explained by the fact that at the beginning of an optimization the limited least squares problem which need to be solved has a small dimension and therefore it is less likely to get negative solution values. In the progress of the optimization, the achievable improvements of the objective function become smaller and smaller and also the cardinality of the set P increases. The last feature in figure (4.1) that can be observed is that of a descending line. This case occurs for the first time at the transition from iteration 28 to iteration 29. In iteration 29 the solution vector of the limited least squares problem s^P has entries that are not positive such as in the case described previously. The

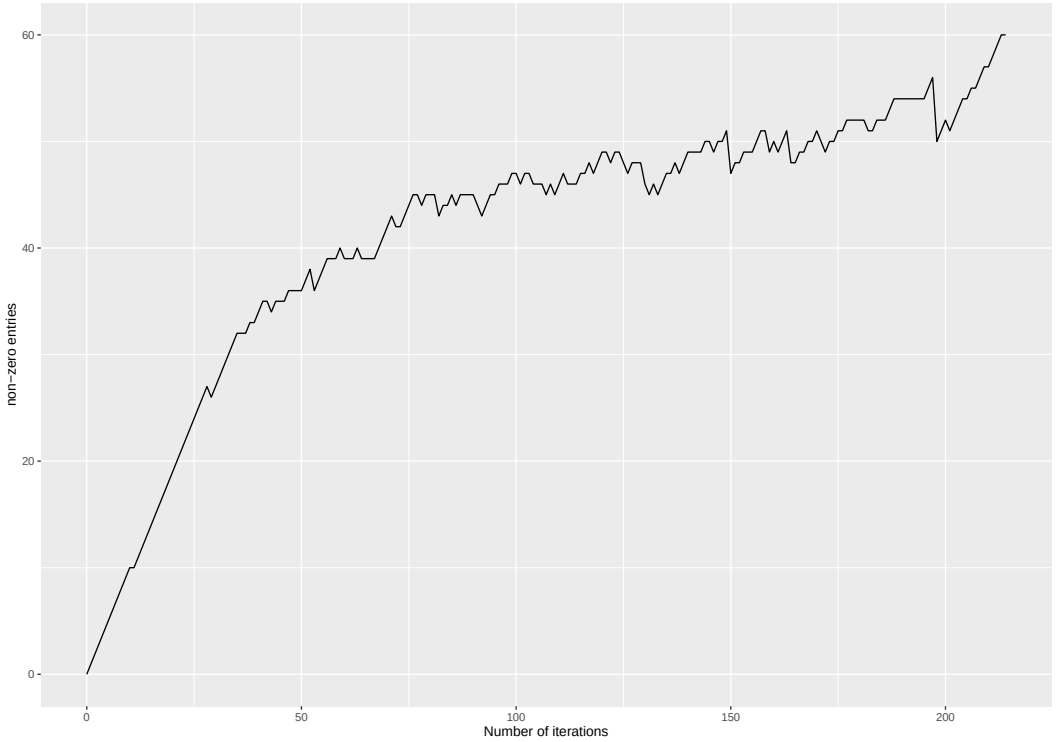


Figure 4.1.: Development of the non-negative entries of the solution vector x based on the number of iterations.

difference to before is that the inner loop is not left after one iteration. So there are several loop cycles required to adapt all negative entries by shifting. In the particular case of iteration 29, the inner loop is executed exactly twice until all entries of s^P are positive. Since with each pass of the inner loop the number of positive entries of the solution vector x is reduced by one, a descending line results after 2 passes. It is therefore clear that the steepness of the descent is a measure of how often the inner loop has been executed before all entries were positive.

If not the number of non-negative entries is considered but the development of formula 4.1 for each iteration, a similar picture emerges. Figure (4.2) shows the logarithmic sum of the squared deviations between the fitted cash flows Ax and the reference cash flows b. The graph is monotonously falling and can be compared in its characteristics with the previous figure (4.1). Starting with the initialization, the solution vector x has only zero entries and therefore $\|Ax - b\|_2^2$ reduces to $\|b\|_2^2$ which gives the first data point of the graph at (0,

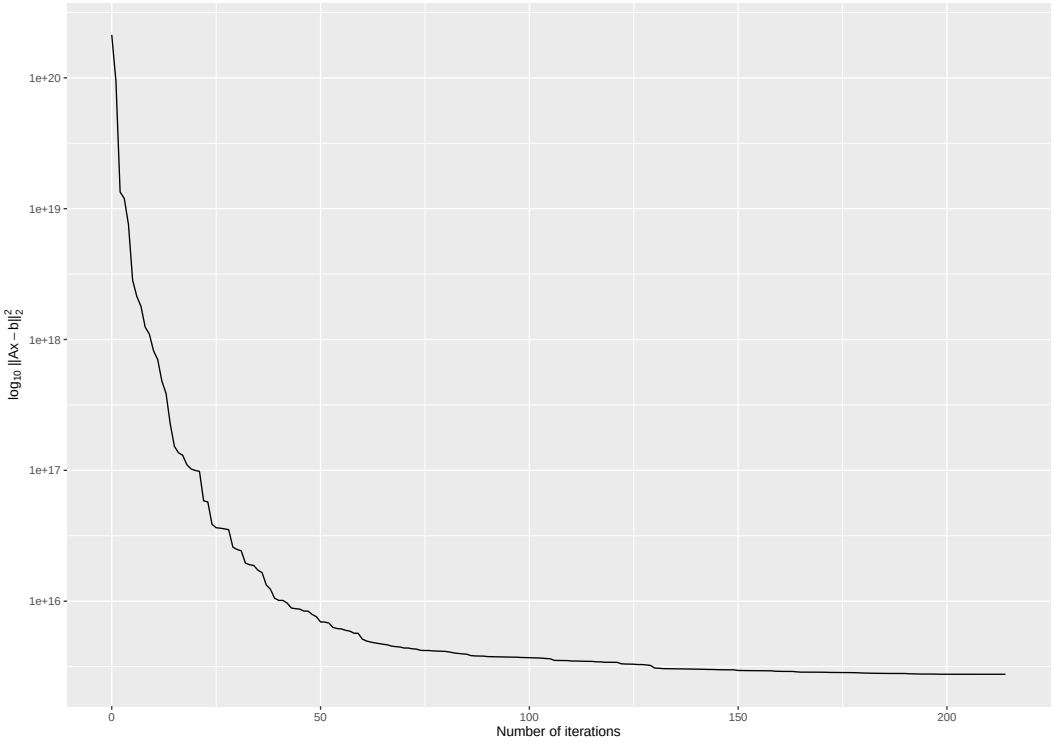


Figure 4.2.: Development of the deviance based on the number of iterations.

2.13e+20). Especially at the beginning of the optimization, a new non-negative entry in x is added with each iteration, as seen in figure (4.1). This is also reflected in the fact that in figure (4.2) the deviation falls significantly for the first iteration. From iteration 50 on there is a clear flattening of the curve which indicates that from then on the improvements of the target function are no longer possible to the same extent as at the beginning of the optimization. This is also confirmed by the fact that after 25% of the iterations 99.9% of the reduction of the target function has already taken place. The remaining 0.1% improvement to the final optimized value of formula (4.1) then takes place in the last 75% of the iterations. Since, as described in more detail in the next section, the inversion of matrices is one of the most time-consuming steps in terms of execution time, it can be useful to cancel an optimization prematurely. A large part of the computing time could be saved and at the same time only a fraction of the optimization quality would be lost.

4.1. Numerical Aspects

As evident in figure (4.1) as well as figure (4.2), the optimization is stopped after 214 iterations. The reason for this is neither that a sufficiently large gradient is no longer available nor that the set R is empty. So all conditions of the main loop stated in step 3 of algorithm 4 are still fulfilled. Rather, a stable state has occurred which does not allow any further improvement of the results. Reaching a stable state during an optimization is one of several reasons for an optimization to stop.

Facing a stable state involves a very special case of index shifts between the sets R and P, which must be handled separately by the algorithm. Starting with the calculation of the gradient in step i), the new index j in R is searched which has the highest gradient. This new index is then moved from the set R to the set P and the constrained least squares problem is calculated as specified in step e). Since the vector s^P now has exactly one negative entry, the inner loop must be entered in the next step. It turns out that exactly that index j of s is negative which was added by the shift from j to P in the last main loop. Since the corresponding entry x_j is zero and s has only one negative entry, the scaling factor α also has a value of zero. This leads to the fact that the shift specified in step ii. has no effect and the solution vector x remains the same. In step iii. the previously determined index j will then be shifted back from the set P to the set R. In the next iteration of the main loop, the index with the largest gradient is found again as the previously moved index j. This will lead to what has just been described, resulting in an infinite loop. A case such as this therefore only occurs if the following two conditions occur simultaneously:

1. The solution vector s^P of the constrained least squares problem has one negative entry.
2. This negative entry is in vector s exactly at the index that was transferred from set R to set P in the current loop pass in step b).

Remark 4.6. *If the stopping reason of the optimization algorithm is the reaching of a stable status, the number of non-negative entries of the solution vector x must be equal for the last two iterations. This means that in figure (4.1) the graph has to end with a horizontal line.*

Another reason that can lead to an unplanned termination of the optimization algorithm is related to the solving process of the restricted least squares

problem in step e) and iv). In these steps the solution of the constrained least squares problem is calculated which requires the use of an inverse matrix. If the inverse of $((A^P)^\top A^P)^{-1}$ does not exist, an alternative solution has to be found. The most practicable approach is to test whether an inverse exists using the determinant of $(A^P)^\top A^P$. If this is not the case, remove the just added index j from the set P and replace it with a j' which has the second largest gradient. Not only the existence of the inverse matrix is important but also its numerical stability. Therefore it should be ensured that with a small change of the matrix A, the inverse matrix A^{-1} do not change significantly in order to get stable results. To determine whether the inversion of a matrix A is numerically stable, a new concept must be introduced which characterizes the numerical stability.

Definition 4.2. *Let V and W be normed vector spaces and $f : V \to W$ a linear operator. Then the operator norm is given by:*

$$\|f\| := \sup_{x \in V \setminus \{0\}} \frac{\|f(x)\|_W}{\|x\|_V} = \sup_{\|x\|_V = 1} \|f(x)\|_W \tag{4.3}$$

Remark 4.7. *Since every real valued matrix $A \in \mathbb{R}^{m \times n}$ corresponds to a linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$, each pair of norms induces an operator norm. In the special case of choosing the Euclidean norm for both vector spaces, formula (4.3) simplifies to:*

$$\|A\|_2 := \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \max_{\|x\|_2 = 1} \|Ax\|_2 \tag{4.4}$$

which is a naturally induced matrix norm called spectral norm.

The natural matrix norm thus vividly corresponds to the greatest possible stretching factor, which results from the application of the linear mapping (i.e. matrix) to a unit vector.

Remark 4.8. *The naturally induced matrix norm satisfies the three norm axioms:*

- $\|A\| = 0$ *iff* $A = 0$ *(being definite)*
- $\|\alpha A\| = |\alpha| \|A\|$ *(being absolutely homogeneous)*
- $\|A + B\| \leq \|A\| + \|B\|$ *(being sub-additive)*

Remark 4.9. *The naturally induced matrix norm is also sub-multiplicative, which means that*

$$\|AB\| \leq \|A\|\|B\|$$

Proof.

$$\begin{aligned}
\|AB\| &= \max_{x \neq 0} \frac{\|ABx\|}{\|x\|} \\
&= \max_{Bx \neq 0} \frac{\|ABx\|}{\|x\|} \\
&= \max_{Bx \neq 0} \frac{\|ABx\|}{\|Bx\|} \frac{\|Bx\|}{\|x\|} \\
&\leq \max_{y \neq 0} \frac{\|Ay\|}{\|y\|} \max_{x \neq 0} \frac{\|Bx\|}{\|x\|} \\
&= \|A\|\|B\|
\end{aligned}$$

□

Remark 4.10 ([46])**.** *The 2-norm has the following properties:*

1. $\|A\|_2 = \sigma_{max}(A)$ *largest singular value of* A
2. $\|A\|_2^2 = \lambda_{max}(A^T A)$ *largest eigenvalue of* $A^T A$
3. $\|A\|_2 = \|A^T\|_2$

Definition 4.3. *Let $A \in \mathbb{R}^{m \times n}$ be a matrix, $\|.\|$ a matrix norm and A^+ the generalized inverse of matrix A. Then the condition number of A is defined as:*

$$\kappa(A) := \|A^+\|\|A\| \tag{4.5}$$

Remark 4.11. *For the special case that matrix A is regular, the pseudo inverse can be replaced with the inverse matrix and the condition number becomes:*

$$\kappa(A) = \|A^{-1}\|\|A\| \tag{4.6}$$

Remark 4.12. *Let A be a non-singular matrix then a lower bound for the condition number is given by:*

$$1 \leq \|I\| = \|AA^{-1}\| \leq \|A\|\|A^{-1}\| = \kappa(A)$$

Remark 4.13. *Let $A \in \mathbb{R}^{m\times n}$ be a matrix and $\|.\|_2$ the induced matrix norm. Then (4.5) simplifies to:*

$$\kappa_2(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}$$

As already mentioned above, it is necessary to estimate to what extent changes in the input variables have an effect on the output variables. In the case of the calculation of an inverse matrix the following approach can be followed. Starting from a matrix A which should be inverted another matrix E is defined. This matrix E represents small changes of matrix A and should therefore be seen as perturbation of matrix A. In order to determine how numerically stable the inversion of a matrix A is, the following term is considered.

$$\|A^{-1} - (A+E)^{-1}\| \tag{4.7}$$

The task now is to find a constant c, so that for all sufficiently small matrices E it applies that:

$$\|A^{-1} - (A+E)^{-1}\| \leq c\|E\| \tag{4.8}$$

Remark 4.14. *It applies:*

$$(A+E)^{-1} = (I + A^{-1}E)^{-1}A^{-1}$$

Corollary 4.1 ([26])**.** *Suppose that B is a bounded linear operator on a Banach space X with $\|B\| < 1$. Then*

$$S = \sum_{k=0}^{\infty} B^k = (I-B)^{-1} \tag{4.9}$$

It therefore follows by applying remark 4.14 and corollary 4.1 that:

$$\begin{aligned}(A+E)^{-1} &= (I + A^{-1}E)^{-1}A^{-1} \\ &= (I - (A^{-1}E) + (A^{-1}E)^2 - (A^{-1}E)^3 + h.o.t.)A^{-1} \\ &= A^{-1} - A^{-1}EA^{-1} + h.o.t.\end{aligned}$$

For the estimation of the stability the following inequality can be obtained by using the previous results as well as formula (4.7):

$$\begin{aligned}\|A^{-1} - (A+E)^{-1}\| &= \|A^{-1} - (A^{-1} - A^{-1}EA^{-1} + h.o.t.)\| \\ &= \|A^{-1}EA^{-1} - h.o.t.\| \\ &\leq \|A^{-1}\|^2\|E\| + h.o.t.\end{aligned}$$

The relative error caused by applying a perturbation to matrix A has therefore an upper bound which is given by:

$$\frac{\|A^{-1} - (A+E)^{-1}\|}{\|A^{-1}\|} \leq \|A\|\|A^{-1}\|\frac{\|E\|}{\|A\|} + h.o.t. \tag{4.10}$$

Remark 4.15. *If the naturally induced spectral norm is used as the matrix norm, formula (4.10) simplifies to:*

$$\begin{aligned}\frac{\|A^{-1} - (A+E)^{-1}\|}{\|A^{-1}\|} &\leq \kappa_2(A)\frac{\|E\|}{\|A\|} + h.o.t. \\ &= \frac{\sigma_{max}(A)}{\sigma_{min}(A)}\frac{\|E\|}{\|A\|} + h.o.t.\end{aligned}$$

To reduce the influence of numerical instabilities, a linear least squares problem is generally solved not by inverting the matrix of the normal equations like in step e) and iv) in algorithm (4) but by other numerical methods. A very frequently used method which avoids forming A^TA and inverting it is the QR decomposition.

4.2. QR - decomposition

A QR decomposition describes the decomposition of a matrix into a product of two matrices with special properties. This decomposition exists for every matrix and can be calculated with different algorithms where the best known are:

- Gram–Schmidt
- Householder transformation
- Givens rotations

Definition 4.4. *Let $A \in \mathbb{R}^{m\times n}$ with $m \geq n$ be a matrix. The decomposition of A into a product $A = QR$ with an orthogonal matrix $Q \in \mathbb{R}^{m\times m}$ and an upper triangular matrix $R \in \mathbb{R}^{m\times n}$ is called a QR-decomposition of A.*

Considering the fact that $m \geq n$ and the matrix R is always quadratic, it often makes sense to partition both R and Q in a way such that the special structure of these matrices can be used advantageously. Since R is an upper triangular

matrix, the last $m-n$ rows of matrix R consist only of zeros. Therefore it makes sense to split the matrix Q into two parts Q_1 and Q_2 which have n and $m-n$ columns respectively. The QR decomposition is reduced by the use of the special characteristics described above to:

$$\underbrace{A}_{m\times n} = \underbrace{Q}_{m\times m}\underbrace{\begin{bmatrix} R_1 \\ 0 \end{bmatrix}}_{m\times n} = [\underbrace{Q_1}_{m\times n}\ \underbrace{Q_2}_{m\times(m-n)}]\underbrace{\begin{bmatrix} R_1 \\ 0 \end{bmatrix}}_{m\times n} = Q_1R_1 \tag{4.11}$$

This notation is often referred to as reduced QR decomposition of A ([50]).

Remark 4.16. *The QR - decomposition is unique if $rank(A) = n$ and the diagonal elements of R_1 are required to be positive.*

Remark 4.17. *Let $Q \in R^{n\times n}$ be an orthogonal matrix and $x \in \mathbb{R}^n$ a vector, then the following properties can be derived:*

- $Q^\top Q = I$ *(Q orthogonal)*
- $\|Qx\|_2^2 = (Qx)^T(Qx) = x^TQ^TQx = x^TIx = \|x\|_2^2$ *(length-invariant)*

The QR-decomposition of matrix A can now be used to reduce the numerical instabilities that can occur in algorithm 4.

Remark 4.18. *Let A^P be the restricted matrix from algorithm 4 and QR the corresponding decomposition such that $A^P = QR$. Then steps e) and iv. from algorithm 4 can be written as:*

$$\begin{aligned} s^P &= ((A^P)^\top A^P)^{-1}(A^P)^\top b \\ &= ((QR)^\top QR)^{-1}(QR)^\top b \\ &= (R^\top Q^\top QR)^{-1}R^\top Q^\top b \\ &= R^{-1}(R^\top)^{-1}R^\top Q^\top b \\ &= R^{-1}Q^\top b \end{aligned} \tag{4.12}$$

A multiplication of formula (4.12) from the left with R results in a formula which is particularly easy for the calculation of the coefficients s_i^P.

$$Rs^P = Q^\top b \tag{4.13}$$

Since R is an upper triangular matrix, this system of equations can be solved very easily. Under the assumption that s^P has dimension $n \times 1$ and $Q^\top b$

is denoted as $\tilde{b}$, the parameters can be calculated by backward substitution following the rule:

$$
\begin{aligned}
s_n^P &= \frac{\tilde{b}}{r_{nn}} \\
s_i^P &= \frac{1}{r_{ii}} \left(\tilde{b}_i - \sum_{j=i+1}^{n} r_{ij} s_j^P \right) \qquad i = n-1, ..., 1
\end{aligned}
$$

After showing how algorithm 4 benefits by using the QR decomposition, a way how such a decomposition can be calculated will be presented. In the following, the Householder-transformation, one of the most widespread methods, is derived.

4.2.1. Householder transformation

The aim of the Householder transformation is to transform matrix A into an upper triangular matrix R by iterative multiplications of so-called Householder matrices H_i. The procedure is schematically shown in the following example:

Example 4.2. *Let $A \in \mathbb{R}^{m \times n}$, $m \geq n$ be the matrix for which a QR decomposition is to be performed. Let H_1 and H_2 be special Householder matrices. Then the QR decomposition is methodically calculated according to the follow-*

ing pattern.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ a_{3,1} & a_{3,2} & \cdots & a_{3,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

$$H_1 A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n} \\ 0 & a_{3,2} & \cdots & a_{3,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ 0 & & & \\ 0 & & & \\ & & A_2 & \\ 0 & & & \\ 0 & & & \end{pmatrix}$$

$$H_2 H_1 A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n} \\ 0 & 0 & \cdots & a_{3,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{m,n} \end{pmatrix} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ 0 & a_{2,2} & \cdots & a_{2,n} \\ 0 & 0 & & \\ & \vdots & A_3 & \\ 0 & 0 & & \end{pmatrix}$$

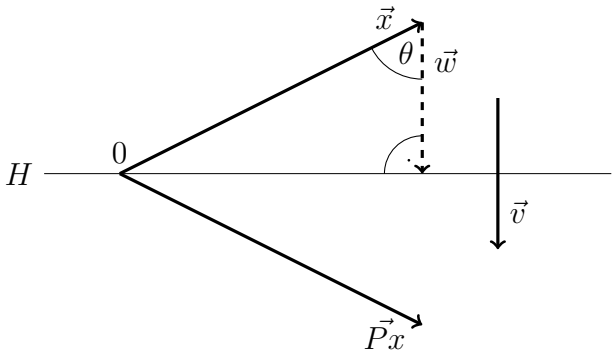
As shown in example (4.2), by applying the Householder matrix H_1 to matrix A, the first column of Matrix A is transformed to a multiple of the first unit vector. This transformation is implemented by a mirroring which is derived in the following section. After the transformation of the first column only the submatrix A_2 of the matrix $H_1 A$ is considered. This matrix consists of one row and one column less, but has a decisive advantage. Considering A_2 on its own, the first column can be mirrored to a multiple of the first unit vector as before and the same logic can be used iteratively. Altogether this means that the use of Householder matrices H_i iteratively generates an upper triangular matrix.

Remark 4.19. *Since the sub matrices (i.e. A_2, A_3, ...) that are to be transformed become in each step one row and one column smaller, this is also the case for the Householder matrices $\tilde{H}_i$. In order to preserve the transformations already carried out in the previous steps, the matrices $\tilde{H}_i$ are therefore enlarged in a way such that:*

$$H_i := \begin{bmatrix} I & 0 \\ 0 & \tilde{H}_i \end{bmatrix}$$

The task now is to find a matrix P that represents the desired reflection. To achieve this, a step-by-step approach is chosen. In a first step the reflection of

Figure 4.3.: Mirroring of $\vec{x}$ to $\vec{Px}$ through hyperplane H.



a vector at a hyperplane through the origin in Euclidean space is constructed. Once this general case has been derived, it can be used to construct a reflection such that the the first column of matrix A is transformed to a multiple of the first unit vector.

For the construction of the mirroring matrix P the case shown in figure (4.3) is considered. Let $\vec{x}$ be a vector in an Euclidean space and $\vec{Px}$ the vector into which $\vec{x}$ is to be transferred by a mirroring. Furthermore, H is the hyperplane at which the reflection should take place. H, that mirror-hyperplane which runs through the origin is defined by the normal vector $\vec{v}$, thus a vector which is orthogonal to the hyperplane. The difference between the vector $\vec{x}$ and the hyperplane H is called $\vec{w}$. The angle enclosed by the vectors $-\vec{x}$ and $\vec{w}$ is called θ. The goal is to identify a relation between $\vec{Px}$ and $\vec{x}$. In the sense of better readability and since there can be no misunderstandings in the following, the vector arrows are omitted from now on. The length of $\vec{w}$ is then given by:

$$\|w\| = \|x\| \cos(\theta) = \|x\| \frac{\langle -x, v \rangle}{\|-x\|\|v\|} = \frac{-x^\top v}{\|v\|}$$

Where in the second step the definition of the dot product was used. The vector w is then characterized by the length and the direction which leads to:

$$w = \frac{-x^\top v}{\|v\|} \frac{v}{\|v\|} = -v \frac{x^\top v}{v^\top v}$$

By referencing to figure (4.3), the mirrored vector Px can now be defined as:

$$Px = x + 2w = x - 2v \frac{x^\top v}{v^\top v} = x - 2 \frac{v v^\top x}{v^\top v} = \left(I - 2 \frac{v v^\top}{v^\top v} \right) x$$

The matrix constructed, representing the linear mapping described above is called Householder matrix. Householder matrices are defined by a normal vector v, i.e. a vector that is orthogonal to the mirror hyperplane and are typically denoted by H.

$$H = I - 2\frac{vv^\top}{v^\top v} \tag{4.15}$$

Where I in equation (4.15) is the identity matrix. In case that v is normalized to length one (4.15) simplifies to:

$$H = I - 2vv^\top \tag{4.16}$$

The concept of Householder matrices can now be used to formalize the process described in example 4.2. Let x be a vector which is to be mirrored to a multiple of the first unit vector e_1. This means that a vector v is required, so that with the corresponding Householder-Matrix H_v the following linear transformation can be achieved.

$$H_v x = ce_1$$

The required reflection vector v now results from normalizing the difference vector and is given by:

$$v = \frac{x - ce_1}{\|x - ce_1\|}$$

A basic property that is important for the construction of a QR decomposition is the fact that Householder matrices are orthogonal.

Remark 4.20. *Let H be a Householder matrix. Then H is symmetric and orthogonal.*

$$H^\top = (I - 2\frac{vv^\top}{v^\top v})^\top = I^\top - \left(2\frac{vv^\top}{v^\top v}\right)^\top = I - 2\frac{\left(vv^\top\right)^\top}{\left(v^\top v\right)^\top} = I - 2\frac{vv^\top}{v^\top v} = H$$

$$\begin{aligned} H^\top H = HH &= (I - 2\frac{vv^\top}{v^\top v})(I - 2\frac{vv^\top}{v^\top v}) \\ &= I^2 - 2I\frac{vv^\top}{v^\top v} - 2I\frac{vv^\top}{v^\top v} + 4\frac{vv^\top}{v^\top v}\frac{vv^\top}{v^\top v} \\ &= I - 4\frac{vv^\top}{v^\top v} + 4\frac{vv^\top vv^\top}{v^\top vv^\top v} \\ &= I - 4\frac{vv^\top}{v^\top v} + 4\frac{(v^\top v)vv^\top}{(v^\top v)^2} \\ &= I \end{aligned}$$

Remark 4.21. *Let $H_1, H_2, ..., H_n$ be Householder matrices as stated in example 4.2. Then by using the properties proofed in the previous remark the QR decomposition is given by:*

$$\begin{aligned} & H_n H_{n-1} \cdot \ldots \cdot H_1 A = R \\ \Leftrightarrow \quad & Q^\top A = R \\ \Leftrightarrow \quad & Q Q^\top A = QR \\ \Leftrightarrow \quad & A = QR \end{aligned}$$

After demonstrating how a QR decomposition can be performed using Householder transformations, the next section is dedicated to the question of performance.

4.2.2. Performance

Even if runtime analyses of algorithm 4 have shown that a large part of the computing time had to be dedicated to the calculation of the gradients, the importance of finding the solution for the least squares problems must not be underestimated and will therefore be analysed as well. Since not only computing time but also the numerical stability of the grouping process is of great interest in practice, the runtime and accuracy of different approaches for solving the least squares problem are compared in this section. Obviously, an implementation of a grouping algorithm which is numerically stable on the one hand and fast on the other hand with regard to the computing time is the preferred solution. Therefore seven different implementations will be compared using only functions that are available in R by default. Since these are standard functions in the area of linear algebra, it can be assumed that the functions used are already highly optimized with respect to performance. In particular, some functions access implementations of the software packages LINPACK and LAPACK directly and therefore act only as wrappers. Examples of such R-functions are `solve` and `qr` whose default methods are interfaces to the LAPACK routines `DGESV` and `ZGESV` as well as `DQRDC` from the LINPACK package.

- `DGESV` computes the solution to a real system of linear equations $A * X = B$, where A is an N-by-N matrix and X and B are N-by-NRHS matrices [2].

- `DQRDC` uses householder transformations to compute the qr factorization of an n by p matrix x. column pivoting based on the 2-norms of the reduced columns may be performed at the users option [13].

The two software libraries LINPACK and LAPACK are written in Fortran and enjoy great popularity in many areas of application due to their efficient implementations with respect to memory usage and computational speed. Of course, there is also a big number of packages available on CRAN (Comprehensive R Archive Network) that offer different implementations for solving least squares problems. Since a comprehensive analysis of these implementations is not possible due to the daily growing number of packages available, the focus is on standard functions available in R. The basic R-functions used to perform the comparison are therefore `solve`, `backsolve`, `qr.solve`, `qr` and `t`. The notation used in algorithm 4 steps e) and iv. is simplified for the following comparison to the degree that the restriction to set P is not explicitly specified, i.e. $A = A^P$. The first three approaches pursue a solution without the calculation of a QR decomposition whereas the last four approaches are based on a QR decomposition.

Method 1: This method uses the calculation procedure for solving a least squares problem given in algorithm 4. By using the function `solve` with only one matrix as parameter the inverse of that matrix will be returned, i.e. `solve(A)` $\widehat{=} Ax = I \Leftrightarrow x = A^{-1}$.

$$s = (A^\top A)^{-1} A^\top b$$

```
solve((t(A) %*% A)) %*% t(A) %*% b
```

Method 2: For this method, the initial approach from the previous method is transformed in such a way that no inverse has to be calculated any more. The least squares problem is then given by:

$$(A^\top A)s = A^\top b$$

```
solve(t(A) %*% A, t(A) %*% b)
```

Method 3: The last approach using no QR decomposition solves the least squares problem using the `crossprod` function. According to

the documentation `crossprod` should be slightly faster than a direct calculation via the transposed matrix. The approach is therefore identical to the one from method 2 except that a different implementation is used:

$$(A^\top A)s = A^\top b$$

```
solve(crossprod(A), crossprod(A,b))
```

Method 4: This approach uses a QR decomposition and directly implements formula (4.12) derived in remark 4.18 for the solution of the least squares problem:

$$s = R^{-1}Q^\top b.$$

```
deco <- qr(A, LAPACK = FALSE)
solve(qr.R(deco)) %*% t(qr.Q(deco)) %*% b
```

Method 5: This approach again uses a QR decomposition and applies formula (4.13), which is a transformation of formula (4.12). Therefore no inverse has to be calculated to solve the least squares problem:

$$Rs = Q^\top b.$$

```
deco <- qr(A, LAPACK = FALSE)
solve(qr.R(deco), t(qr.Q(deco)) %*% b)
```

Method 6: This method again uses a QR decomposition, but also makes use of the special form of the decomposition where R is an upper triangular matrix. The function `backsolve` solves a system of linear equations where the coefficient matrix is upper triangular [41]:

$$Rs = Q^\top b$$

```
deco <- qr(A, LAPACK = FALSE)
backsolve(qr.R(deco), t(qr.Q(deco)) %*% b)
```

Method 7: The last method uses a QR decomposition as well. The result of the QR decomposition is not saved in a temporary variable as before, but is passed directly to the `qr.solve` method.

$$QRs = b$$

```
qr.solve(qr(A, LAPACK = FALSE), b)
```

In order to determine how the number of columns of matrix A affects the run time, the following setup was chosen. Matrix A was defined as a square matrix with $n = 5000$, where the entries of A are uniformly distributed between 0 and 100000, i.e. $A \in \mathbb{R}^{n \times n}$. The values of vector b are uniformly distributed between 0 and 1000. By defining matrix A and vector b, the result vector s is determined as well. For each of the seven methods, the number of columns used from matrix A was successively increased and then the least squares problem was solved. The results obtained in the test runs were produced with the following configuration:

- Processor: Intel® Core™ i7-6700
- RAM: 64GB
- R-version: 3.5.0

In order to obtain reliable results, all calculations were repeated ten times and the individual results averaged. Figure (4.4) shows how the time needed to solve the least squares problem increases with the number of columns used. It is important to note that due to illustration purposes the methods are presented separately based on their underlying approach, but both panels use the same scaling. The left part of the graph shows the results of methods one to three, i.e. all those methods that do not use a QR decomposition to solve the least squares problem. Even if it is difficult to recognize in the left panel of figure (4.4) due to the scaling, the data shows that method one generally has a significantly longer runtime than method two and three even for small values of n. Starting from a column number of about 1000, it is clearly visible that the runtime for method one increases significantly faster than for the other two methods. This can be attributed to the fact that method one requires the explicit calculation of an inverse matrix which is not advisable. When an inverse

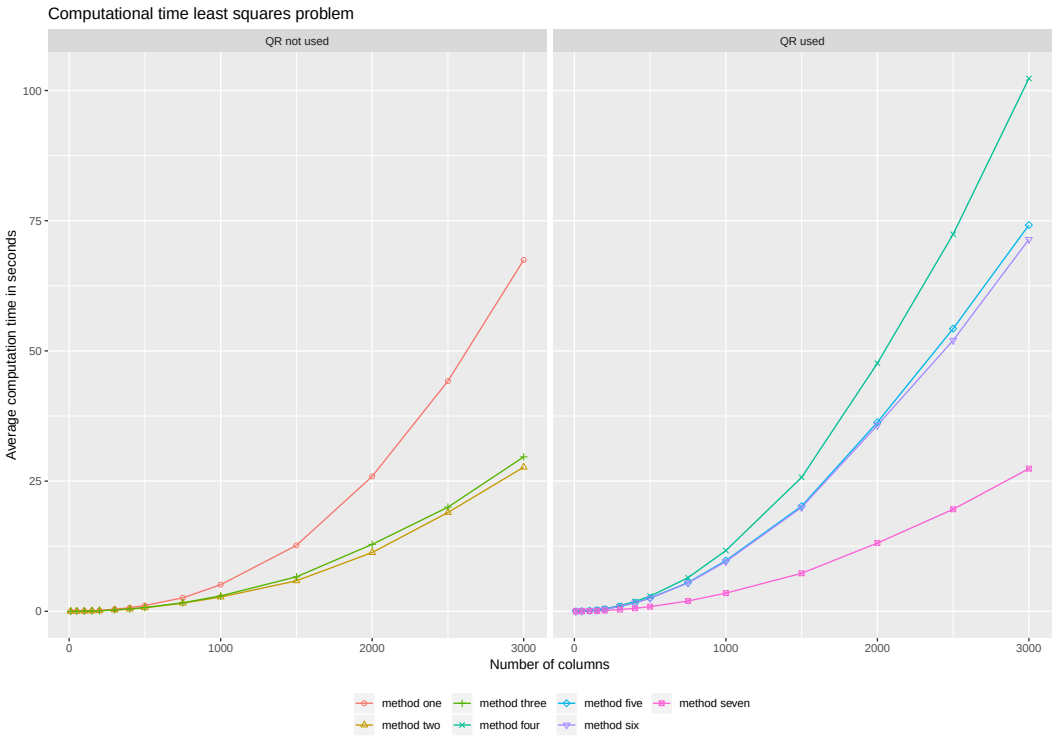


Figure 4.4.: Computation time for solving the least squares problem $As = b$ in seconds.

is needed rather an LU composition should be performed [31]. Moreover, there are no significant differences between method two and method three, although method two generally has a slightly shorter runtime. The usually existing speed advantage given in the help of the `crossprod` function could not be verified. In the right part of figure (4.4), the results of methods four to seven are shown, i.e. all those methods that use a QR decomposition. In general, it can be seen that, with the exception of method seven, all methods are considerably slower than those without QR decomposition. Method four shows significantly longer running times than all other methods, which is due to the fact that both, a QR decomposition and a matrix inversion must be performed. Methods five and six behave very similarly for the most part, with a small runtime advantage for method six being observed as the number of columns increases. Since the two methods are basically identical and method six uses only the special structure of the upper triangular matrix, it can be assumed that this advantage is only effective for larger systems of equations. The method that distinguishes significantly from the others in terms of runtime is method seven. Over the entire range, its computational time is significantly shorter than that of the other methods. Since method seven is the only method that does not temporarily save the result of the QR decomposition but directly processes it, it seems reasonable to conclude that this already results in a significant performance advantage. Considering the results regardless of whether a QR decomposition was used or not, the following summary results:

- Methods two, three and seven are to be classified as equivalent in terms of runtime and are also the fastest altogether.
- Methods one, five and six have a similar runtime, with method one always performing best.
- The slowest method by far is method four.

After analysing how the runtime of the different approaches behave in relation to the number of columns, there is still a need to analyse their accuracy. As mentioned above, methods two, three and seven scale best with an increasing number of columns of matrix A. Since method seven uses a QR decomposition, while methods two and three do not use it, an important aspect is to compare their accuracy. Figure (4.5) shows how much the calculated solutions s_{LS} deviate on average in absolute values from the actual values s_{true}, i.e. $err_{avg} = \sum_{i=1}^{n} \frac{1}{n} |s_{LS}^i - s_{true}^i|$. Each box represents the deviations of all seven methods for a fixed number of columns. For example, the box in the upper right corner of figure (4.5) shows the results for a matrix A with 300

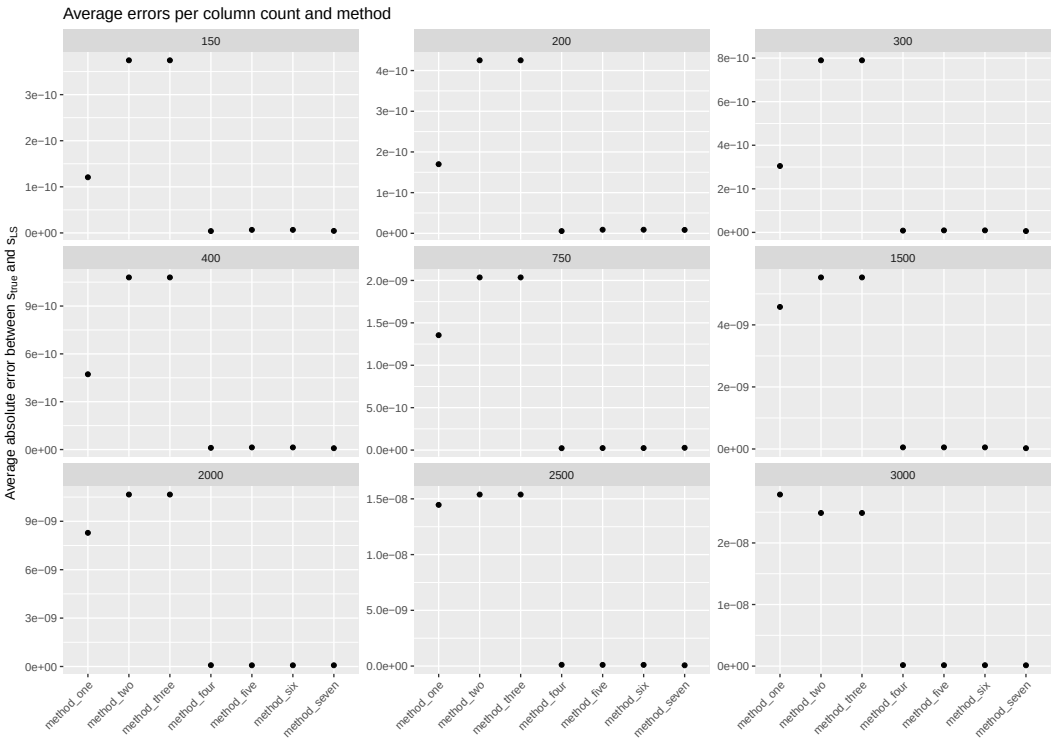


Figure 4.5.: Comparison of average absolute errors based on method and number of columns used from matrix A

columns. In contrast to figure (4.4), the data points for the column numbers 10, 50, 100, 500 and 1000 were not shown. This is because by omitting the plots a better readability and arrangement was possible without suffering a loss of information. It should therefore be noted once again that all missing boxes basically provide the same results as those shown here and therefore do not provide any information gain. Considering the nine panels, a clear pattern can be identified. In each individual case methods one to three show on average higher deviations than methods four to seven. Taking the definitions of the methods into account it can be revealed that methods four to seven are based on a QR decomposition. Within those four methods no significant difference can be seen with respect to the deviations. The results of the simulation therefore suggest that all methods based on a QR decomposition are equivalent in terms of accuracy. Methods one to three, which don't use a QR decomposition, show a differentiated picture. Methods two and three show similar deviations, which is not surprising because of their definition, as they differ only in the functions `t` and `crossprod`. For method one, there are strong indications that the average deviation increases with the number of columns. In the case that matrix A has 150 columns, the average differences between s_{LS} and s_{true} are significantly smaller for method one than for method two or method three. If the number of columns is increased step by step, this difference becomes smaller and smaller. Looking at the case where matrix A has 3000 columns, it is evident that method one has the highest deviation of all tested methods. Since method one is the only method which uses the calculation of an inverse, the results leads to the conclusion that calculating an inverse becomes more unstable as the number of columns increases. This shows, as already mentioned in previous sections, that an algorithm for finding the solution of the least squares problem should be implemented, which doesn't rely on inverse matrix calculations. After both the execution time and the accuracy of the seven approaches have been examined, the following picture emerges:

- In all test cases considered, it could be proven that those methods that use a QR decomposition show the highest accuracy.

- Among those methods that use a QR decomposition, the fastest is the one that does not temporarily store the QR results but directly processes them.

- Looking at the fastest methods of each group (QR decomposition used or not) no significant difference can be found.

- If an inverse matrix is calculated, the more columns the matrix has, the worse the overall accuracy gets for the test sample.
- The calculation time required to solve the least squares problem scales exponentially with the number of columns.

Taking all aspects into account, method seven is the preferred method in terms of both runtime and accuracy.

5. Neural Networks (NN)

In the last chapter of this thesis, a mathematical concept is discussed that has attracted much attention in the recent past, namely neural networks. The area of artificial intelligence (AI) in which neural networks are embedded has been the subject of an intense media hype in recent years. In 2016, for example, a computer program developed by the British company Google DeepMind succeeded for the first time in defeating Lee Sedol of South Korea, considered to be the strongest Go player in the world [51]. This victory of a machine against a human being is considered to be a milestone in the field of artificial intelligence [45]. Further successes were also achieved in the area of real-time games at the beginning of 2019. For the first time DeepMind's program called AlphaStar was able to defeat the world's best players in StarCraft which is considered to be one of the most challenging real-time strategy games [47]. Also Gartner, a global research and advisory firm, which publishes the well-known but definitely criticizable hype cycle representations considers AI as one of the most important technologies of recent years. In the hype cycle for emerging technologies from 2017 there are 4 out of the 32 listed technologies that can be attributed to the field of AI, such as Deep Learning or Machine Learning [36]. Also in the following years 2018 and 2019 technologies that clearly belong to the AI sector were mentioned in the hype cycle 5 and 6 times respectively [37], [38]. This trend towards new methods based on neural networks can also be seen in other places, such as Kaggle. With more than one million registered users, Kaggle is one of the world's largest platforms for data science competitions and attracts teams from all over the world with prize money in the millions [20]. It can be observed that, besides gradient boosting, one of the most important concepts with which to win Kaggle competitions is the concept of neural networks in various forms.

It is therefore clear to see that on the one hand the technology behind AI has enormous potential to solve problems that have so far been assumed to be solved only by humans. On the other hand, it can be assumed that this trend is not just a short-lived phenomenon, but a continuous process leading to business solutions which are based on AI-systems. Therefore, it is even

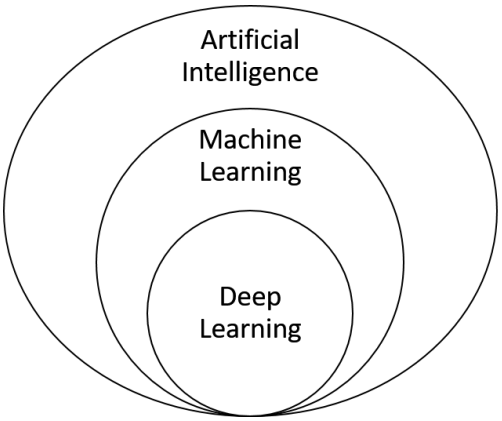


Figure 5.1.: Relation of Artificial Intelligence, Machine Learning, and Deep Learning.

more important to understand the underlying concepts of these technologies and to discuss their applicability in the insurance industry. Whether in the end a completely automated grouping algorithm based on neural networks is possible at all or can be implemented with the available resources remains open. However, the aim of this section is to provide an overview of the basic concepts and to present case studies with insurance cash flows. Since various terms and buzzwords related to artificial intelligence are often used differently in media reports, it is useful to classify some of the most often used terms in order to show their relationship and provide a general framework which is based on [1].

- Artificial Intelligence: Artificial Intelligence is a branch of computer science that deals with the programming of intelligent computer systems. Due to a missing definition of the term intelligence the question which types of computer programs are included is not clearly definable. In addition to machine learning and deep learning, there are many other approaches that do not include any kind of learning but are also part of AI.
- Machine Learning: Machine learning arises from this question if it is possible to design a program in such a way that it can learn how to perform a specific task automatically. Given a set of input data and the corresponding results the task is to derive rules. These rules are therefore the output of the machine learning algorithm and can then be used to derive the results for new input data. What all these methods have in common is the fact that they actually try to identify statistical patterns in the

data. Thus the aim is to find a meaningful representation of the given data by projections, translations, rotations, nonlinear transformations or any other method and to apply it then to new input data.

- Deep Learning: Deep learning, as a sub field of machine learning, focuses on the progressive learning of several levels of abstraction which are increasingly meaningful representations of the data. One can think of the method as a multistage information distillation process where in every single process step a more meaningful representation of the data is obtained. The term deep in deep learning is a reference to the multiple layers which store the abstract representations of the input data in such an algorithm. The number of layers that contribute to a meaningful representation of the data is called depth. Although there is a considerable amount of learning involved, models with several hundred layers are quite common, depending on the type of problem at hand. The concept of the neural network is a reference to the field of neurobiology and the neurons that are connected in different ways in the human brain. Although neuronal networks are not models of the human brain, it is surprising what amazing results can be achieved with such a simple idea and a sufficiently large amount of data.

After a brief classification of the terms, the next section describes the basic structure and functionality of neural networks.

5.1. Fundamentals of Neural Networks

In the previous section the functionality of neuronal networks was already described in a rather abstract way as a multi-stage information distillation process. After this high level explanation, the focus is now on pointing out which individual components make up a simple neural network and how they interact. Figure (5.2) shows a simplified representation of the individual components required to build one of the most basic neural networks possible. Starting point for explaining the learning process of a neural network is a data set consisting of both the input and the associated output data. If the cash flows of insurance policies are to be simulated, the individual policy parameters can be used as input data and the associated cash flows as output data. These two exogenous quantities are then processed in several steps.

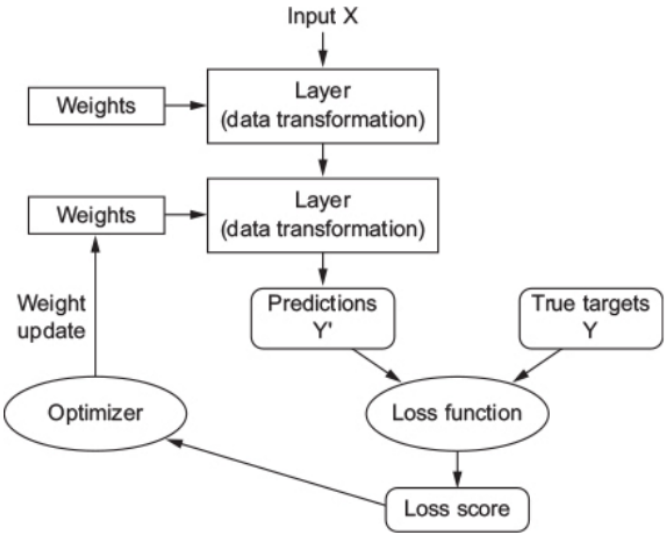


Figure 5.2.: Simplified representation of a neural network from [1]

Step 1: The process is initiated by providing the input data `Input X` in a suitably format to the first layer.

Step 2: This first layer then performs a data transformation based on some weights associated with that specific layer. In the first cycle those weights are randomly initialized.

Step 3: The transformed output from the first layer is then passed to the next layer and serves as its input. This layer also performs a data transformation based on weights which are again initialized randomly for the first cycle.

Step 4: After the data has passed through the last data transformation layer of the model the transformed output values are used as `Predictions Y'`.

Step 5: A predefined function, called the `Loss function`, measures the quality of the network's output by comparing the `Predictions Y'` and the `True targets Y`. The result is the `loss score`, which is used as a feedback signal to adjust the `weights` associated with the layers.

Step 6: The task of the `Optimizer` is to take the `loss score` as an input and update the `weights` in such a way that the `loss score` is lowered.

Remark 5.1. *The choice of the correct loss function is of great importance. If the loss function does not correctly reflect an improvement of the model, the neural net will inevitably drift in the wrong direction.*

Remark 5.2. *As already explained in the previous chapters, in the specific case of cash flow matching the sum of the squared residuals would be suitable as a loss score.*

The above steps describe a single learning cycle for a neural network. Of course, passing the input data through the net once is not sufficient to find combinations of weights that would lead to good prediction results at all. So what learning means, is to find a set of values for all weights such that the input is mapped correctly to the output for as many different input combinations as possible. This learning is achieved by grouping the input data into so-called batches and repeatedly passing those batches through the network. Associated with a batch is the batch size which defines the number of training samples sent through the neural network before the weights are adjusted by the optimizer.

Remark 5.3. *Three different batch size approaches can be used to learn the weights in a network.*

- *Batch Gradient Descent: Batch size is equal to the size of the training samples.*
- *Stochastic Gradient Descent: Batch size is equal to one.*
- *Mini-Batch Gradient Descent: Batch size is bigger than one but less than the size of the training samples.*

Depending on the design of the network there can be millions of weights that need to be learned. Adjusting those millions of weights based on the feedback signal is by far the most computing intensive part of training a network. Due to the fact that the learning process is based on a large number of similar matrix operations it is possible to use highly parallelized algorithms. The availability of relatively affordable hardware that can efficiently handle such parallelizable tasks is besides the existence of large amount of data another major reason why neural networks took off in recent years.

Remark 5.4. *Compared to a pure CPU system, graphics cards with their massively parallel architecture are predestined for handling such training tasks and can therefore massively reduce the time needed to train a network.*

After providing an brief overview of all components used in a neural network and presenting the idea of how a network learns, the next section focuses on the internals of a single layer.

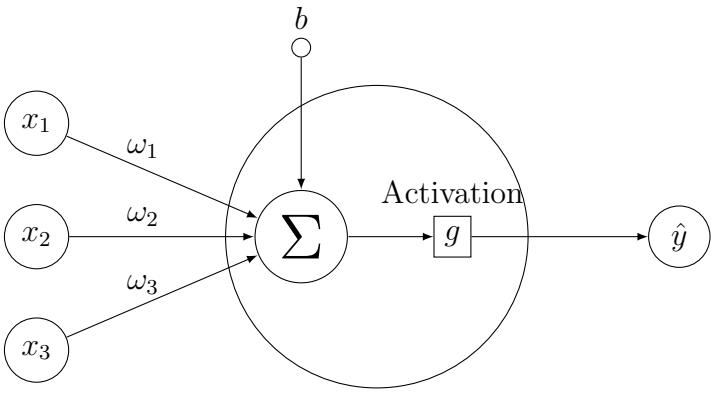


Figure 5.3.: Components of a single neuron.

5.1.1. Single neuron

The so called neurons form the basis of every neural network and are the elements which carry out the data transformations. Each layer of a neural network consists of one or more neurons which are connected differently depending on the structure of the underlying network. The task of a neuron is to take a predefined number of input values and map them to an one dimensional output. Figure (5.3) shows a single neuron with all its associated components needed to perform the data transformation given by:

$$\begin{aligned} \hat{y} &= g\left(\sum_{i=1}^{3} \omega_i x_i + b\right) \\ &= g(w^\top x + b) \\ &= g(z) \end{aligned} \tag{5.1}$$

In the example shown in figure (5.3) the neuron takes the three input values x_1, x_2 and x_3 and transforms them in several steps into the output value $\hat{y}$. In the first step, a weighted sum is formed from the input values and the corresponding weights w_1, w_2 and w_3. A bias b is then added to this weighted sum and the intermediate result is called z. This intermediate result is then the input for the activation function g. Applying the activation function to z gives the final output value of the neuron. The purpose of the activation function can be seen . Since both, the values of the inputs and the values of the weights are unrestricted, the value of z can be in the interval (-∞, +∞). A natural way of determining whether the next neuron should be activated or

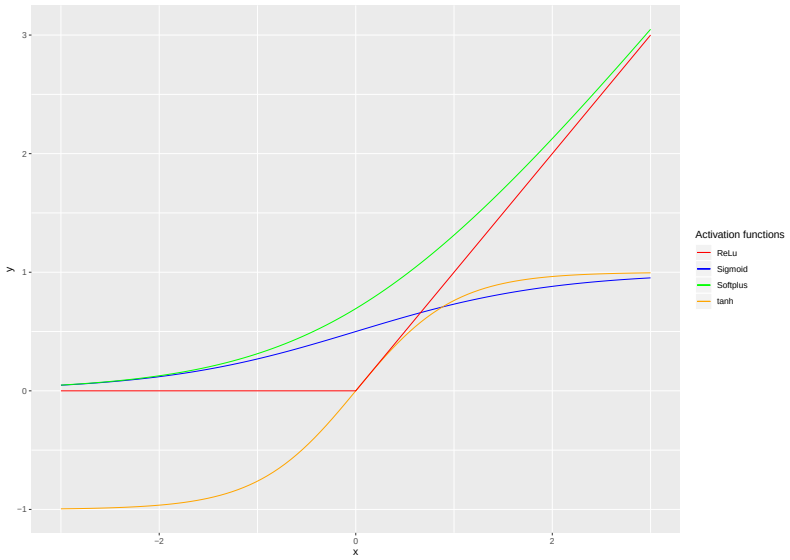


Figure 5.4.: Commonly used activation functions for neural networks.

not is to apply a transformation to z, which is done by an activation function. For this reason neural networks use non-linear activation functions. Those activation functions can limit the value range and also enable the network to learn complex data structures. Some of the most commonly used activation functions in neural networks are shown in figure (5.4):

- Sigmoid function:

$$\begin{aligned} g &: \mathbb{R} \to (0,1) \\ g(x) &= \frac{1}{1+e^{-x}} \end{aligned} \tag{5.2}$$

  The sigmoid function has an order of continuity of C^∞ and is convex for all values less than 0, and it is concave for all values greater than 0. Since the output is always in the range from 0 to 1, one of the use cases of the sigmoid function is to model probabilities. Applying the sigmoid function to strongly negative values results in values close to zero. This means that the next neuron is only activated if the linear transformation has given a value z that is not too negative.

- Hyperbolic tangent function:

$$\begin{aligned} g &: \mathbb{R} \to (-1,1) \\ g(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \end{aligned} \tag{5.3}$$

The hyperbolic tangent function is very similar to the sigmoid function and, just like the sigmoid function, also has an order of continuity of C^∞. Since the range of output values lies between -1 and 1, the possibilities of using this activation function as a gate are rather limited compared to the sigmoid function. Possible application scenarios can be found in the area of classification problems where all outputs below zero belong to class A and all outputs greater than zero belong to class B.

- ReLu function:

$$\begin{aligned} g &: \mathbb{R} \to [0, +\infty) \\ g(x) &= max(0, x) \end{aligned} \tag{5.4}$$

  One of the most successful and widely-used activation functions is the Rectified Linear Unit (ReLU). All values greater than zero activate the next neuron and all negative input values result in the next neuron not being activated by setting the output to zero. Because of the maximization the ReLu-function has only an order of continuity of C^0. Although it is non-differentiable, this function is popular because of it's simplicity and reliability as a gate function (see [42], [35]).

- Softplus function:

$$\begin{aligned} g &: \mathbb{R} \to (0, \infty) \\ g(x) &= ln(1 + e^x) \end{aligned} \tag{5.5}$$

  Since ReLu is non-differentiable at zero a smooth version of it, which also has an order of continuity of C^∞, is given by the softplus function. It can be used as a replacement for the ReLu function. As various analyses have shown the performance between the softplus function and the ReLu function is negligible in most cases (see [42]).

In the example given in figure (5.3), the output value of the neuron is already the prediction $\hat{y}$, but in almost every case the output of one neuron serves as an input for another neuron. The next section is therefore dedicated to the interaction of multiple neurons in multiple layers and also introduces a general notation based on [33], that allows to describe the mathematical concept of a neural network including an optimization algorithm.

5.1.2. Multiple neurons

Having described the basic function of a single neuron in the previous section, the focus is now on how multiple neurons interact in order to form a neural

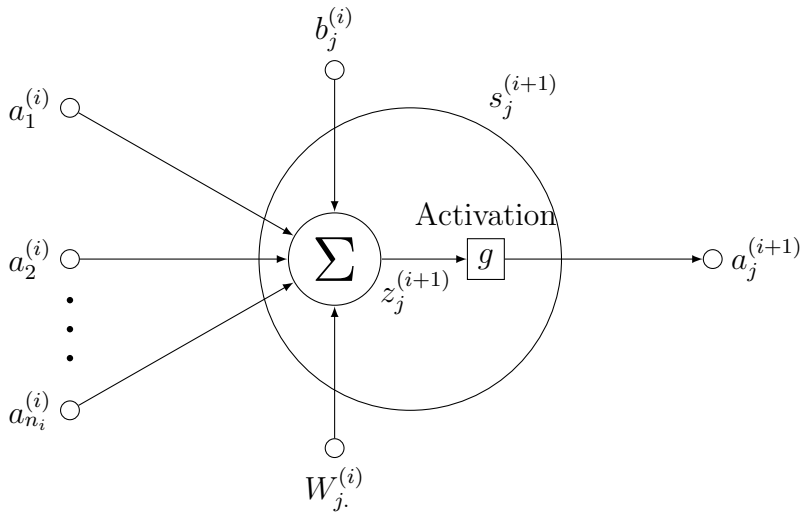


Figure 5.5.: Components of a single neuron.

network. Figure (5.5) shows the same single neuron as figure (5.3) but with a slightly adapted and extended notation. While in figure (5.3) the weights ω_i are directly assigned to the individual inputs x_i, in figure (5.5) these weights are represented by a weight vector $W_{j.}^{(i)}$. To be able to refer to a single neuron in a neural network with a large number of neurons, a single neuron is referred to in the form of $s_j^{(i)}$. Where s stands for single neuron, i for the layer in which the neuron is used and j for a consecutive number over all neurons in this layer i. The notation of the inputs and the output of a neuron have also changed, so that the notation $a_j^{(i)}$ is now used in both cases. The a refers to activation and the two indices i and j refer to the layer and respectively the neuron that generated the activation value. The value $a_1^{(i)}$ from figure (5.5) therefore identifies both, the output of the first neuron in the i-th layer and the input for the j-th neuron in the i+1-th layer. Depending on which and how many layers, and how the individual neurons in the individual layers or between the individual layers are connected to each other, different types of neural networks result. So-called feedforward networks, for example, are characterized by the fact that neurons from one layer are only connected to neurons from the next higher layer. The information flow therefore takes place only in one direction, namely from the input side to the output side. On the other hand there are so-called recurrent nets where feedback between different layers is also allowed. There are three different categories of recurrent neural networks which can be distinguished depending on how they use their feedback signals [10]:

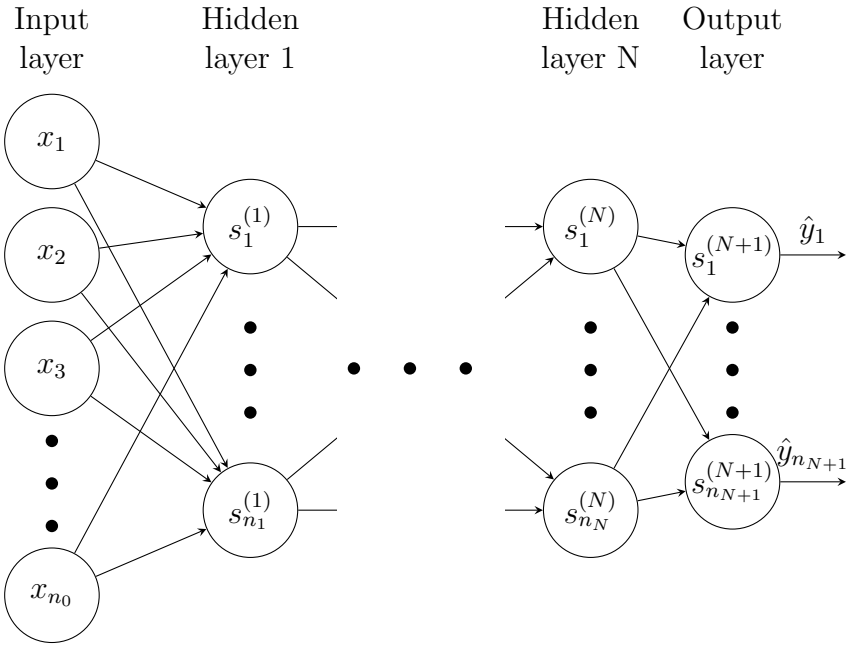


Figure 5.6.: Illustration of fully connected feedforward neural network with N hidden layers.

- Direct feedback: The own output of a neuron is used as an additional input for the neuron.
- Indirect feedback: The output of a neuron serves as an input to a neuron of a previous layer.
- Lateral feedback: The output of a neuron servers as an input to a neuron from the same layer.

Since the possibilities of how the different neurons are connected in a neural network are almost unlimited, the following section describes one of the most common forms [28]. It is a fully connected feedforward network as shown in figure (5.6). The most important properties of such a network are:

- Fully connected: All outputs from the neurons of layer i serve as inputs for all neurons in layer $i + 1$.
- Feedforward: The information flows only from one layer to the next higher layer.

For the calculation and analysis of such a neural network the following notation, which is taken from [33], is used in accordance with figure (5.5) and figure (5.6). Let

- $N \in \mathbb{N}$ be the number of hidden layers in the neural network.
- $N+2$ be the total number of layers in the network since the hidden layers are wrapped between an input and an output layer.
- $n_i \in \mathbb{N}, i \in \{0, 1, ..., N+1\}$, be the number of neurons in the i-th layer.
- $s_j^{(i)}$ denote the j-th neuron of the i-th layer.
- $a^{(i)} \in \mathbb{R}^{n_i}$, $a^{(i)} = (a_1^{(i)}, a_2^{(i)}, ..., a_{n_i}^{(i)})^\top$ be the vector of activation values produced by the neurons of the i-th layer.
- $b^{(i)} \in \mathbb{R}^{n_{i+1}}$, $b^{(i)} = (b_1^{(i)}, b_2^{(i)}, ..., b_{n_{i+1}}^{(i)})^\top$ be the bias vector for the linear transformation performed in all neurons of layer $i+1$.
- $W_{j.}^{(i)} \in \mathbb{R}^{1 \times n_i}$, $W_{j.}^{(i)} = (W_{j1}^{(i)}, W_{j2}^{(i)}, ..., W_{jn_i}^{(i)})$ be the weight vector for the linear transformation performed in the j-th neuron of the $i+1$-th layer. Combining all weights used in the $i+1$-th layer into a matrix $W^{(i)} \in \mathbb{R}^{n_{i+1} \times n_i}$ results in:

$$W^{(i)} = \begin{pmatrix} W_{11}^{(i)} & W_{12}^{(i)} & \cdots & W_{1n_i}^{(i)} \\ W_{21}^{(i)} & W_{22}^{(i)} & \cdots & W_{2n_i}^{(i)} \\ \vdots & \vdots & & \vdots \\ W_{n_{i+1}1}^{(i)} & W_{n_{i+1}2}^{(i)} & \cdots & W_{n_{i+1}n_i}^{(i)} \end{pmatrix} = \begin{pmatrix} W_{1.}^{(i)} \\ W_{2.}^{(i)} \\ \vdots \\ W_{n_{i+1}.}^{(i)} \end{pmatrix},$$

  with $W_{j.}^{(i)}$ being the j-th row of $W^{(i)}$.
- $z^{(i)} \in \mathbb{R}^{n_i}$, $z^{(i)} = (z_1^{(i)}, z_2^{(i)}, ..., z_{n_i}^{(i)})^\top$ be the result vector after the linear transformation performed in all neurons of layer i.
- $g : \mathbb{R} \mapsto \mathbb{R}$ be any activation function which is applied to the result of the linear transformation.
- $x \in \mathbb{R}^{n_0}$, $x = (x_1, x_2, ..., x_{n_0})^\top$ be a vector of input data to train the model.
- $y \in \mathbb{R}^{n_{N+1}}$, $y = (y_1, y_2, ..., y_{n_{N+1}})^\top$ be a vector of output data to train the model.

- $\hat{y} \in \mathbb{R}^{n_{N+1}}$, $\hat{y} = (\hat{y}_1, \hat{y}_2, ..., \hat{y}_{n_{N+1}})^\top$ be a vector of estimated output values obtained from the neural network by sending an input vector x through the network.

Remark 5.5. *Since the input layer has n_0 neurons, the input vector x must be an element of $\mathbb{R}^{n_0}$. Similarly, the estimated output $\hat{y}$ of the neural network is a vector which is an element of $\mathbb{R}^{n_{N+1}}$.*

Remark 5.6. *Since $a^{(i)}$ denotes the values that are transferred between neurons, it is both the output values of the neurons from layer i and the input values of the neurons from layer $i+1$.*

Remark 5.7. *The input values x for the neuronal network correspond to the first activation values $a^{(0)}$, e.g. $x = a^{(0)}$. Similarly, the estimated output $\hat{y}$ of the neural network corresponds to the last activation values $a^{(N+1)}$, e.g. $\hat{y} = a^{(N+1)}$.*

Remark 5.8. *For the training of a neural network many different sets of input and associated output vectors are needed. If a specific set of input or output data is to be referenced, this is done by adding a superscript, e.g. $x^{(k)}$ and $y^{(k)}$. The corresponding estimated values are also referred to as $\hat{y}^{(k)}$.*

Remark 5.9. *The weight defined as $W^{(i)}_{jk}$ connects the activation value from the k-th neuron in layer i with the j-th neuron from layer $i+1$.*

After a notation for the analysis of the neural network has been defined, the next section is devoted to the question of how data can flow through a network and how the network can learn.

5.2. Train a model

Figure (5.3) together with formula (5.1) already illustrated in the previous section how a single neuron is constructed and how it processes the inputs. Since the notation has been generalized in the previous section also formula (5.1) could be generalized.

Remark 5.10. *Let $a^{(i)} \in \mathbb{R}^{n_i}$ be the activation values from layer i, $b^{(i)}_j \in \mathbb{R}$ the bias term for the j-th neuron in layer $i+1$, $W^{(i)}_{j.}$ the weights and $g : \mathbb{R} \to \mathbb{R}$*

any activation function. Then the linear transformation performed by the j-th neuron of layer $i+1$ called $z_j^{(i+1)}$ is given by:

$$z_j^{(i+1)} = \sum_{k=1}^{n_i} W_{jk}^{(i)} a_k^{(i)} + b_j^{(i)} \tag{5.6}$$

Using vector notation gives:

$$z_j^{(i+1)} = W_{j.}^{(i)} a^{(i)} + b_j^{(i)} \tag{5.7}$$

Remark 5.11. *Based on the previous remark, the activation value $a_j^{(i+1)}$ generated by neuron $s_j^{(i+1)}$ is given by:*

$$a_j^{(i+1)} = g(z_j^{(i+1)}) = g\left(\sum_{k=1}^{n_i} W_{jk}^{(i)} a_k^{(i)} + b_j^{(i)} \right) \tag{5.8}$$

Using vector notation gives:

$$a_j^{(i+1)} = g(z_j^{(i+1)}) = g(W_{j.}^{(i)} a^{(i)} + b_j^{(i)}) \tag{5.9}$$

Performing these transformations for all neurons from layer $i+1$ gives a system of n_{i+1} equations:

$$\begin{aligned} a_1^{(i+1)} &= g(W_{1.}^{(i)} a^{(i)} + b_1^{(i)}) \\ a_2^{(i+1)} &= g(W_{2.}^{(i)} a^{(i)} + b_2^{(i)}) \\ &\vdots \\ a_{n_{i+1}}^{(i+1)} &= g(W_{n_{i+1}.}^{(i)} a^{(i)} + b_{n_{i+1}}^{(i)}) \end{aligned}$$

Since the activation function is only defined for scalar values, a small extension must be made.

Remark 5.12. *Let $g : \mathbb{R} \to \mathbb{R}$ be an activation function which operates on scalars only. In case that a vector is supplied the scalar activation function g is applied element by element.*

With the help of the previous remark it is now possible to represent the flow of information through the network in vector notation.

$$\begin{aligned} a^{(0)} &= x \\ z^{(i)} &= W^{(i-1)} a^{(i-1)} + b^{(i-1)} \\ a^{(i)} &= g(z^{(i)}) \\ \hat{y} &= a^{(N+1)} \end{aligned} \tag{5.10}$$

This step, where the neural network takes an input vector and transforms it to a prediction is the first step to train the network and is called forward propagation [49]. The process shown in figure (5.2) illustrates that after the generation of the predictions $\hat{y}$ the deviation between those and the real outputs y must be measured. A way of measuring the deviation between actual and estimated values is the L^2 norm.

Definition 5.1. *Let $x^{(i)}$ and $y^{(i)}$ be the vector of the input and the corresponding output values, respectively. Let W be the matrices of weights and b the vectors of biases used in the entire network for a single forward propagation. If $\hat{y}^{(i)}_{W,b,x^{(i)}}$ denotes the estimated outputs from the network then the loss score is given by:*

$$L(b, W, x^{(i)}, y^{(i)}) := \frac{1}{2} \left\| \hat{y}^{(i)}_{W,b,x^{(i)}} - y^{(i)} \right\|_2^2 \tag{5.11}$$

By using the notation given in definition (5.11) it is explicitly stated, that the loss of a single forward pass depends on the four parameters $b, W, x^{(i)}$ and $y^{(i)}$. In order to increase the readability in the following paragraphs from now on it is not longer explicitly stated that the estimated output depends on the three parameters W, b and $x^{(i)}$, e.g. $\hat{y}^{(i)}_{W,b,x^{(i)}} = \hat{y}^{(i)}$. Since in most cases not only a single pair of input and output data is used for the calculation of the loss function but several, the loss must also be defined for a sample of multiple input and output vectors. In the field of machine learning such a sample of multiple input and output vectors is called a batch. The batch size is one of many so-called hyperparameters of a neural network and specifies how many samples must be sent through the network before the model parameters are updated.

Definition 5.2. *Let $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$ be a sample of m input and the corresponding output vectors, e.g. the batch size is m. Then the total loss is given by:*

$$\begin{aligned} L(b, W) &= \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left\| \hat{y}^{(i)} - y^{(i)} \right\|_2^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} L(b, W, x^{(i)}, y^{(i)}) \end{aligned} \tag{5.12}$$

Remark 5.13. *In order to reduce the complexity of the model by using smaller weights and to reduce overfitting to some extend, a regularization term can be*

used in equation (5.12).

$$\begin{aligned} L(b, W) &= \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left\| \hat{y}^{(i)} - y^{(i)} \right\|_2^2 + \frac{\lambda}{2} \sum_{i=0}^{N} \left\| W^{(i)} \right\|_F^2 \\ &= \frac{1}{m} \sum_{i=1}^{m} L(b, W, x^{(i)}, y^{(i)}) + \lambda R(W) \end{aligned} \tag{5.13}$$

where $\left\| W^{(i)} \right\|_F^2$ *is defined as:* $\sqrt{\sum_{j=1}^{n_{i+1}} \sum_{k=1}^{n_i} W_{jk}^{(i)}}$

If a regularization term is used in a neural network, the choice of the regularization parameter λ plays an important role. When the regularization term λ is too high the weight matrices W are close to zero and this can lead to simple networks and underfitting. However, if the parameter is set too low and therefore higher weights of the matrices are not penalized enough, regularization has no effect at all.

Once a loss function is defined, the information of the loss score can be used to take the next step as shown in figure (5.2). The job of the optimizer is to evaluate how the weights of W and b have to be adjusted so that the loss score becomes smaller in the next forward pass. A reduction of the loss score indicates that the deviation between predicted output values and actual outputs is reduced. This iterative improvement of the loss score based on the insights given by the optimizer is the so called learning procedure of a neural network. The task of minimizing $L(b, W)$ with respect to the weights b and W can be solved by using the gradient descent method which is an first-order iterative optimization algorithm for finding the local minimum of a function [9]. This method calculates, using the gradients, for each weight of the network what effect a change would have on the loss score. The algorithm used to calculate the gradients is called backpropagation. Once all gradients have been calculated after a forward pass, one way to update the weights of the neural network is to adjusted them by a constant learning rate of α.

Remark 5.14. *Let* $W^{(i)}$ *and* $b^{(i)}$, $i = 0, 1, ..., N$ *be the weights used in the* $i+1$*-th layer and* $\alpha \in \mathbb{R}_+$ *the user defined learning rate. Then the weights are updated in every optimization step such that:*

$$b_j^{(i)} = b_j^{(i)} - \alpha \frac{\partial}{\partial b_j^{(i)}} L(b, W) \tag{5.14}$$

$$W_{jk}^{(i)} = W_{jk}^{(i)} - \alpha \frac{\partial}{\partial W_{jk}^{(i)}} L(b, W) \tag{5.15}$$

Since the learning rate α is a hyperparameter that can be chosen freely, the only parts that need to be analyzed are the partial derivatives. By using definition 5.2 those can be written as:

$$\frac{\partial}{\partial b_j^{(i)}} L(b, W) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_j^{(i)}} L(b, W, x^{(i)}, y^{(i)}) \tag{5.16}$$

$$\frac{\partial}{\partial W_{jk}^{(i)}} L(b, W) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{jk}^{(i)}} L(b, W, x^{(i)}, y^{(i)}) \tag{5.17}$$

This shows that the partial derivative of the loss function based on a batch of size m is given by the mean value of the partial derivatives of all single loss functions based on the individual input and output vectors used in this batch. If the regularization term introduced in remark 5.13 is used for the weight matrices W, this does not change anything for the partial derivatives with respect to the bias terms b, but the partial derivatives of the weight matrices given in formula (5.17) changes to:

$$\frac{\partial}{\partial W_{jk}^{(i)}} L(b, W) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{jk}^{(i)}} L(b, W, x^{(i)}, y^{(i)}) + \lambda W_{jk}^{(i)} \tag{5.18}$$

If the regularization term is used, just an additive expression of the form $\lambda W_{jk}^{(i)}$ is added. Since the construction of the partial derivatives with respect to $W_{jk}^{(i)}$ and $b_j^{(i)}$ are very similar, only the former will be explained step by step in the following. Before the partial derivative can be analysed in more detail a distinction must be made between two different cases.

Case 1: The first case deals with the special situation in which the weights $W^{(N)}$ and $b^{(N)}$ are directly related to the output. These weights are used in the transformation that results in the estimated output values $\hat{y}$ and thus directly influences the loss function $L(b, W, x^{(N)}, y^{(N)})$. Because of their direct influence on the loss function, the calculation of their gradients is easier and can be used as a template for the other case.

Remark 5.15. *Let $W^{(N)}$ be the matrix of weights used in the last layer of the neural network. Then the gradient is given by:*

$$\frac{\partial L(b, W)}{\partial W_{jk}^{(N)}} = \delta_j^{(N+1)} a_k^{(N)}$$

Proof.

$$\begin{aligned}
\frac{\partial L(b,W)}{\partial W_{jk}^{(N)}} &= \frac{\partial}{\partial W_{jk}^{(N)}} \frac{1}{2} \sum_{l=1}^{n_{N+1}} (\hat{y}_l - y_l)^2 \\
&= \frac{\partial}{\partial W_{jk}^{(N)}} \frac{1}{2} \sum_{l=1}^{n_{N+1}} (a_l^{(N+1)} - y_l)^2 \\
&= (a_j^{(N+1)} - y_j) \frac{\partial}{\partial W_{jk}^{(N)}} (a_j^{(N+1)} - y_j) \\
&= (a_j^{(N+1)} - y_j) \frac{\partial}{\partial W_{jk}^{(N)}} g\big(z_j^{(N+1)}\big) \\
&= (a_j^{(N+1)} - y_j) g'\big(z_j^{(N+1)}\big) \frac{\partial}{\partial W_{jk}^{(N)}} \left[\sum_{s=1}^{n_N} W_{js}^{(N)} a_s^{(N)} + b_j^{(N)} \right] \\
&= (a_j^{(N+1)} - y_j) g'\big(z_j^{(N+1)}\big) a_k^{(N)} \\
&= \delta_j^{(N+1)} a_k^{(N)}
\end{aligned}$$

Whereby in the last step all terms with index j were summarized in δ_j^{N+1}. □

Remark 5.16. *Let $b^{(N)}$ be the vector of biases used in the last layer of the neural network. Then the gradient is given by:*

$$\frac{\partial L(b,W)}{\partial b_j^{(N)}} = \delta_j^{(N+1)}$$

Case 2: The second case deals with those situations where the weights W and b are not directly but indirectly associated with the loss function. These are all weights that are used in the hidden layers 1 to N. The calculation of those gradients relies on the first case and requires several steps.

Remark 5.17. *Let $z_l^{(N+1)}$ be the result of the linear transformation of neuron l in the $N+1$-th layer and $W_{jk}^{(N-1)}$ the weight for the j-th neuron in the N-th layer. Then the partial derivative of $z_l^{(N+1)}$ with respect to $W_{jk}^{(N-1)}$ is given by:*

$$\frac{\partial z_l^{(N+1)}}{\partial W_{jk}^{(N-1)}} = \frac{\partial z_l^{(N+1)}}{\partial a_k^{(N)}} \frac{\partial a_k^{(N)}}{\partial W_{jk}^{(N-1)}} = W_{lk}^{(N)} g'\big(z_k^{(N)}\big) a_k^{(N-1)} \tag{5.19}$$

Proof.

$$\frac{\partial z_l^{(N+1)}}{\partial a_k^{(N)}} = \frac{\partial}{\partial a_k^{(N)}}\left[\sum_{s=1}^{n_N} W_{ls}^{(N)} a_s^{(N)} + b_l^{(N)}\right] = W_{lk}^{(N)}$$
$$\frac{\partial a_k^{(N)}}{\partial W_{jk}^{(N-1)}} = g'\big(z_k^{(N)}\big)\frac{\partial}{\partial W_{jk}^{(N-1)}}\left[\sum_{s=1}^{n_{N-1}} W_{ks}^{(N-1)} a_s^{(N-1)} + b_k^{(N-1)}\right] = g'\big(z_k^{(N)}\big)a_k^{(N-1)}$$

□

Remark 5.18. *Let $W^{(N-1)}$ be the matrix of weights used in the last hidden layer of the neural network. Then the gradient is given by:*

$$\frac{\partial L(b, W^{(N)})}{\partial W_{jk}^{(N-1)}} = g'\big(z_k^{(N)}\big)a_k^{(N-1)} \sum_{l=1}^{n_{N+1}} \delta_l^{(N+1)} W_{lk}^{(N)}$$

Proof.

$$\begin{aligned}
\frac{\partial L(b, W^{(N)})}{\partial W_{jk}^{(N-1)}} &= \frac{\partial}{\partial W_{jk}^{(N-1)}}\frac{1}{2}\sum_{l=1}^{n_{N+1}} (a_l^{(N+1)} - y_l)^2 \\
&= \sum_{l=1}^{n_{N+1}} (a_l^{(N+1)} - y_l)\frac{\partial}{\partial W_{jk}^{(N-1)}}(a_l^{(N+1)} - y_l) \\
&= \sum_{l=1}^{n_{N+1}} (a_l^{(N+1)} - y_l)\frac{\partial}{\partial W_{jk}^{(N-1)}} g\big(z_l^{(N+1)}\big) \\
&= \sum_{l=1}^{n_{N+1}} (a_l^{(N+1)} - y_l) g'\big(z_l^{(N+1)}\big)\frac{\partial}{\partial W_{jk}^{(N-1)}} z_l^{(N+1)} \\
&= \sum_{l=1}^{n_{N+1}} \delta_l^{(N+1)}\frac{\partial}{\partial W_{jk}^{(N-1)}} z_l^{(N+1)} \\
&= g'\big(z_k^{(N)}\big)a_k^{(N-1)} \sum_{l=1}^{n_{N+1}} \delta_l^{(N+1)} W_{lk}^{(N)}
\end{aligned}$$

Where in the last step the result from remark 5.17 was used. □

Remark 5.19. *Let $b^{(N-1)}$ be the vector of biases used in the last hidden layer of the neural network. Then the gradient is given by:*

$$\frac{\partial L(b, W)}{\partial b_j^{(N-1)}} = g'\big(z_j^{(N)}\big) \sum_{l=1}^{n_{N+1}} \delta_l^{(N+1)} W_{lk}^{(N)}$$

Those results derived in the last section can be used for any number of hidden layers so that the update process described in remark 5.14 can be slightly adopted and summarized in algorithm 5.

Algorithm 5 Stochastic gradient descent (SGD) with mini batches [21]

1. Define learning rates $\alpha_1, \alpha_2, ...$
2. Initialize weight parameters W and b and reference them by θ.
3. Set a counter: $k = 1$
4. While stopping criterion is not met
   a) Sample a minibatch of m examples from the training set with corresponding output values $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$
   b) Compute gradient estimate: $g = \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
   c) Apply update: $\theta = \theta - \alpha_k g$.
   d) Increase counter: $k = k + 1$

Remark 5.20. *The notation for the computed gradient* $\frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$ *in algorithm 5 is just a short form of formulas (5.16) and (5.17).*

Remark 5.21. *By setting all learning rates equal (e.g.* $\alpha = \alpha_1 = \alpha_2 = ... = \alpha_k$*), exactly the update process described in remark 5.14 is obtained.*

Remark 5.22. *In practice, the learning rates are often chosen in such a way that a linear decrease takes place up to a certain number of iterations and then the learning rates are kept constant.*

$$\alpha_k = \begin{cases} (1 - \frac{k}{\tau})\alpha_0 + \frac{k}{\tau}\alpha_\tau & k < \tau \\ \alpha_\tau & k \geq \tau \end{cases}$$

With α_0, α_τ *and* τ *set such that:*

- α_τ *is roughly 1 percent of the initial learning rate* α_0*.*
- τ *is set to the number of iterations required to pass the whole training set a few hundred times through the neural network.*
- α_0 *does not make the learning process of the model oscillate too much by setting it too high. Gentle oscillations are okay, but setting* α_0 *properly is part of a hyperparameter tuning.*

Remark 5.23 ([21])**.** *A sufficient condition to guarantee that the stochastic gradient descent algorithm converges is that:*

$$\sum_{k=1}^{\infty} \alpha_k = \infty \qquad \textit{and} \qquad \sum_{k=1}^{\infty} \alpha_k^2 < \infty$$

Based on SGD, the algorithm can be optimized in terms of time consumption. By adding a momentum, the learning process can be accelerated and the undesired case that the algorithm runs into a local minimum is partly counteracted.

Algorithm 6 Stochastic gradient descent (SGD) with momentum [21]

1. Initialize learning rate α and momentum parameter γ
2. Initialize weight parameter θ and velocity v.
3. Set a counter: $k = 1$
4. While stopping criterion is not met
   a) Sample a minibatch of m examples from the training set with corresponding output values $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$
   b) Compute gradient estimate: $\hat{g} = \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
   c) Compute velocity update: $v = \gamma v - \alpha \hat{g}$
   d) Apply update: $\theta = \theta + v$.
   e) Increase counter: $k = k + 1$

Remark 5.24. *The more consecutive gradients point in the same direction, the greater the updates of the weights become.*

Remark 5.25. *Usually the momentum parameter γ is initialized with values like 0.5, 0.9 or 0.99. Adopting γ over time is possible but typically less important than the linear decay of α shown previously.*

In the two algorithms 5 and algorithm 6 presented so far, the learning rate α is specified globally. As mentioned in remark 5.22 a sequence of learning rates α_i can be defined so that the learning rate is decreasing over time and is then kept constant after a certain number of iterations. If the idea of globally falling learning rates α_i is taken a step further, a group of algorithms can be presented which individually adapts the learning rates of every single model parameter. This then leads to a rapid drop in the individual learning rate for parameters

Algorithm 7 Adaptive Gradients (AdaGrad) [21]

1. Initialize learning rate α.
2. Initialize Small constant δ used for numerical stabilization. (Suggested default: 10^{-7})
3. Initialize weight parameter θ.
4. Initialize gradient accumulation variable: $r = 0$.
5. While stopping criterion is not met
   a) Sample a minibatch of m examples from the training set with corresponding output values $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$
   b) Compute gradient estimate: $\hat{g} = \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
   c) Accumulate squared gradient: $r = r + \hat{g} \odot \hat{g}$
   d) Compute update: $\Delta\theta = -\frac{\alpha}{\sqrt{r}+\delta} \odot \hat{g}$ (Division and square root applied element-wise)
   e) Apply update: $\theta = \theta + \Delta\theta$.

Algorithm 8 Root mean square propagation (RMSProp) [21]

1. Initialize learning rate α and decay parameter ρ.
2. Initialize Small constant δ used for numerical stabilization. (Suggested default: 10^{-6})
3. Initialize weight parameter θ.
4. Initialize gradient accumulation variable: $r = 0$.
5. While stopping criterion is not met
   a) Sample a minibatch of m examples from the training set with corresponding output values $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$
   b) Compute gradient estimate: $\hat{g} = \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
   c) Accumulate squared gradient: $r = \rho r + (1 - \rho)\hat{g} \odot \hat{g}$
   d) Compute update: $\Delta\theta = -\frac{\alpha}{\sqrt{r+\delta}} \odot \hat{g}$ ($\frac{1}{\sqrt{r+\delta}}$ applied element-wise)
   e) Apply update: $\theta = \theta + \Delta\theta$.

with large partial derivatives. In contrast, the learning rates of parameters with comparatively small partial derivatives are changed only slightly. This approach therefore allows an individual adjustment of the learning rate for each individual parameter depending on how the corresponding gradient behaves. Algorithms 7 and 8 show possible implementations of such adaptive learning approaches.

Remark 5.26. *Algorithm 7 implements an adoptive learning rate for each individual model parameter by summing all previous squared gradients. As the sum increases with each iteration, this means that the learning rate decreases monotonically.*

The strength of algorithm 7 lies in the fast convergence for convex functions. Since the entire history of the gradients is always used for the adjustment of the learning rates through summation, there may be cases in which the algorithm is trapped in convex sinks. This may be due to the fact that when such a sink occurs, the individual learning rate has already been reduced so much that this local minimum cannot be left. Algorithm 8 addresses this problem by making the adoptive learning rate more flexible.

Remark 5.27. *Algorithm 8 uses an exponentially weighted moving average to adjust the individual learning rates. The decay of the moving average is then controlled by a new hyperparameter called ρ, with small values increasing the magnitude of decay.*

By using the exponentially weighted moving average, previous gradients are considered less for the adaptive learning rate the further back in time they appeared. This means that, in contrast to the AdaGrad method, the individual learning rate does not have to fall monotonously but can develop more flexible.

The last algorithm presented here can be seen as a combination of RMSProp (algorithm 8) and SGD with momentum (algorithm 6). This algorithm, called Adam, shown in algorithm 9 is fairly new in the field of neural networks and uses momentum and adaptive learning rates to converge faster and more robust [27].

Remark 5.28. *Since the update process of the biased first moment estimate in step d) of algorithm 9 can be expressed as $s_t = (1 - \rho_1) \sum_{i=1}^{t} \rho_1^{t-i} g_i$ the correction term can be derived by:*

Algorithm 9 Adaptive Moments (Adam) [21]

1. Initialize learning rate α. (Suggested default: 0.001)
2. Initialize exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
3. Initialize Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
4. Initialize weight parameter θ.
5. Initialize 1st and 2nd moment variables $s = 0$, $r = 0$.
6. Initialize time step $t = 0$.
7. While stopping criterion is not met
   a) Sample a minibatch of m examples from the training set with corresponding output values $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\}$
   b) Compute gradient estimate: $\hat{g} = \frac{1}{m}\nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
   c) Increase time step: $t = t + 1$
   d) Update biased first moment estimate: $s = \rho_1 s + (1 - \rho_1)\hat{g}$
   e) Update biased second moment estimate: $r = \rho_2 r + (1 - \rho_2)\hat{g} \odot \hat{g}$
   f) Correct bias in first moment: $\hat{s} = \frac{s}{1-\rho_1^t}$
   g) Correct bias in second moment: $\hat{r} = \frac{r}{1-\rho_2^t}$
   h) Compute update: $\Delta\theta = -\alpha\frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$
   i) Apply update: $\theta = \theta + \Delta\theta$. (operations applied element-wise)
   j) Increase counter: $k = k + 1$

$$\begin{aligned}\mathbb{E}[s_t] &= \mathbb{E}\left[(1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i}g_i\right] \\ &= \mathbb{E}[g_t](1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i} + \zeta \\ &= \mathbb{E}[g_t](1-\rho_1^t) + \zeta\end{aligned}$$

Where $\zeta = 0$ if the true first moment is stationary and otherwise ζ can be kept small because of the exponential decay [27].

Remark 5.29. *Similarly to remark 5.28 the correction term for the biased second moment can be derived.*

After the basic concepts have been explained, the last section of this chapter is dedicated to the question how well cash flows can be replicated with neural networks using different neural networks.

5.3. Cash flow replication

For the analysis of the effectiveness of neural networks in replicating the cash flows of an insurance portfolio, a subset of an actual insurance portfolio was used. The portfolio used consists of risk insurance policies, whereby only policies of the most often sold tariff were included for the evaluations. This ensures that, if a neural network is successfully trained, a large part of the risk insurance portfolio can already be covered. In principle, all risk tariffs of the portfolio can be described with 37 different characteristics. These features include biometric data such as the sex or age of the insured person as well as actuarial parameters such as a discount label. The approximately 200,000 policies of the tariff analysed here, can be characterised by the following 17 attributes:

- Policy ID
- Age
- Premium
- Reserve
- Months since policy start
- Surpluses

- Policy status
- Policy duration
- Premium frequency
- Premium payment period
- Discount indicator
- Sex
- Main contract flag
- Sum assured
- Sales channel
- Type of sale
- Rider premium

Since most of the properties are self-explanatory, only those are briefly outlined here where there could be some ambiguity. The sales channel is a categorical variable and indicates how the policy was sold, either through exclusive distribution or through a broker. The type of sale is also a categorical variable and indicates how the contract was concluded. That is, whether it is a new business, a renewal or another type of business transaction. These 17 characteristics were then used to forecast the future portfolio development with the help of a proprietary projection software called Prophet. With a future projection period of 60 years and about 60 reporting variables, this results in hundreds of millions of data points. Since this amount of data could not be handled with the available resources, especially since the learning processes of a neural networks would become too costly, there were two simplifications:

- The number of policies was reduced to approximately 25,000.
- The reporting variables were reduced to the mathematical reserve.

The policies were chosen in such a way that they form a group that is as homogeneous as possible. These are policies that:

- Are in the premium payment period with monthly payments.
- Have no discounts.
- Were sold via a broker.
- Are no renewals.

It can therefore be seen that there have been some limitations in the selection of policies in terms of discrete features. This fact must be taken into account when evaluating the results, but should not interfere with the further procedure.

The projection software was then used to forecast the reserve trajectories for the next 60 years for those 25,000 model points. To be able to check the

Table 5.1.: Average relative error between true and predicted values over a projection period of 60 years.

Configuration		Average Error Year 1 - 25	Average Error Year 1 - 60
Nodes	Network Type		
8	I	1.971%	1.995%
8	II	2.213%	2.467%
20	I	0.963%	1.600%
20	II	3.001%	3.111%
37	I	0.653%	1.341%
37	II	1.714%	2.289%
50	I	0.673%	1.466%
50	II	1.181%	2.151%

quality of the tested neural networks, the data was split into a training and a test set. For this purpose, 80% of the data points were randomly selected and assigned to the training set and the remaining 20% to the test set. The test set was never used to train the neural nets, but was only used once after training has been finished to check the predictive power. To smooth possible volatility caused by the random initialization of the weight parameters, each test configuration was tested 40 times with randomly initialized parameters and the results were averaged. The two network structures analysed are:

I) 2 hidden layers.

II) 3 hidden layers.

In both configurations, the number of neurons in the hidden layers was the same for all hidden layers in the network and was increased in steps from 8 to 50. The individual layers used relu (see equation 5.4) as activation function and initialized the weights uniformly. As batch size 32 was chosen and the number of epochs was set to 50. As loss function the mean square error was chosen and as optimization routine the Adam algorithm described in algorithm 9. In addition, a dropout rate of 10% for each layer has been implemented to prevent the model from overfitting. To ensure the reproducibility of the results, the random generator was set before each test run. Table 5.1 shows the averaged relative errors of the eight configurations tested. It is evident that the relative deviations in the first 25 years are on average lower than over the entire projection horizon of 60 years. This is due to the fact that the longer a projection goes on, the more policies have already expired. This means that

towards the end of the projection there are not so many data points from which the neural network could learn and therefore the predictive power is lower. Since over the projection period the number of policies in the portfolio is decreasing, this means that the absolute values for the cash flows are also decreasing. A constant deviation in cash flows in absolute terms therefore has a much greater relative impact at the end of the projection than at the beginning. However, this phenomenon can be reduced by creating imaginary policies for the training portfolio that have a particularly long duration. This ensures that the neural network has enough data points even at the end of the projection period to learn from and can therefore make better predictions. Of course, this approach involves additional effort, since policies not in the portfolio must first be created and then projected into the future. Another finding of table 5.1 is that those models with network type I achieve better results than those with type II in all cases. This shows that those nets that have only 2 hidden layers give better results than those that consist of 3 hidden layers. The number of neurons within each layer also has an influence on the predictive power of a neural network. Nets with 8 or 20 neurons per layer perform worse than those with 37 or 50 neurons per layer as shown in table 5.1. Overall, the best performance is shown by the configuration with two hidden layers of 37 neurons each. In this neural network the number of neurons per layer is the average between the number of input neurons and output neurons. The predicted values for this configuration compared to the real values are shown in figure (5.7). A clear deviation in year 10 is striking which can be explained by an outlier. After the learning process, one of the 40 neural networks was not able to predict a proper reserve for year 10, so a value of 0 was predicted. If the hardware is designed for the training of many neural networks, such outliers can be compensated simply by increasing the number of samples from 40 to 100, for example. Apart from this outlier, the results up to year 25 are remarkably good and even after that, the deviations are within reasonable limits. It could thus be shown that surprisingly good forecasts of future reserve development are already possible with simple neural networks. Due to the limited technical possibilities, even for such simple configurations of neural networks as listed in table 5.1, no extensive tuning of the hyperparameters could be performed. Based on the configuration with the smallest deviations, an improvement of the predicted results can be achieved by adjusting the hyperparameters. Some possible adjustments would be:

- Adjusting the drop out rate for each layer.
- Adjusting the activation function for each layer.

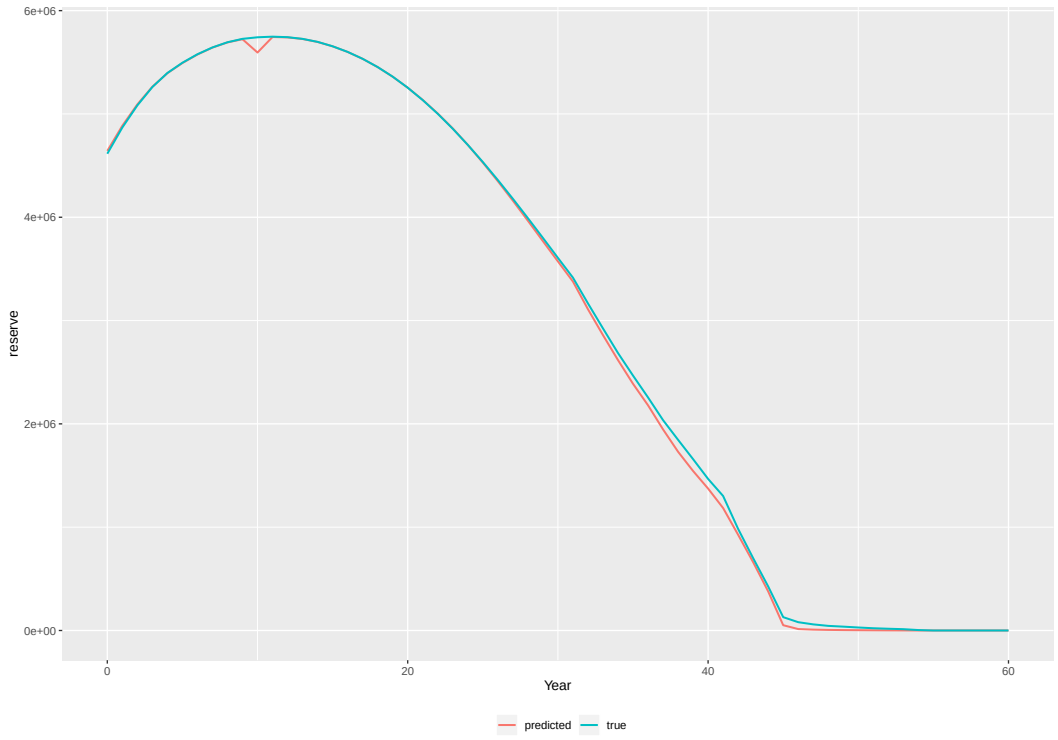


Figure 5.7.: Reserve predicted by neural network with 2 hidden layer with 37 nodes each.

6. Conclusion

The aim of this work was to analyse procedures that allow for a compression of an insurance portfolio. A representative subset of policies should be selected in such a way that projected cash flows can be replicated as accurately as possible. A special focus was placed on the high dimensionality of the data and the associated challenges. The k-means algorithm presented in chapter 3 was particularly convincing due to its simplicity of implementation, but also had weaknesses. Especially in the case of high-dimensional property spaces, the curse of dimensionality and the associated difficulty in determining the optimal number of cluster centres must be emphasized. The non-negative least squares algorithm presented in Chapter 4 is also well established in practice and is therefore used in insurance companies. However, numerical instabilities must be pointed out when using this method and implementations based on inversions of matrices are strongly discouraged in this context. Instead, it could be shown that by applying QR-decompositions, significantly more robust results could be achieved and therefore those approaches are to be preferred. In the last chapter, neural networks were presented as a method that has so far received little attention in the insurance sector. With a network consisting of less than 100 neurons, it is possible to replicate cash flows over a period of 60 years so well that the deviation is significantly below 2%. Even if these neuronal networks cannot yet be used to compress portfolios, the potential applications are interesting. For example, such trained networks could be used to quickly forecast estimated cash flow patterns for entire portfolios. The training effort could be carried out in advance in order to receive first results within minutes after the delivery of the monthly policy data. A more complex use case of neural networks would be that in which neural networks are used to estimate important parameters in the context of solvency calculations. Initial attempts in this direction are promising and require further in-depth analysis [11].

Appendix A.

Tables

month	pop_15	pop_70	prem_15	prem_70	prem_diff	claims_15	claims_70	claims_diff
0	1	1						
9	0.999834	0.987259	385.60	1128.40	742.80	1.70	130.61	128.91
21	0.977140	0.947577	379.73	1092.63	712.90	4.73	185.70	180.97
33	0.952445	0.905554	370.17	1045.64	675.47	10.08	201.42	191.34
45	0.928273	0.863652	360.80	998.80	638.00	15.22	216.81	201.60
57	0.904661	0.821842	351.64	952.09	600.45	19.87	232.12	212.25
69	0.881638	0.780093	342.69	905.48	562.79	24.11	247.29	223.18
93	0.837332	0.696652	325.47	812.40	486.93	32.45	277.16	244.71
105	0.816021	0.654913	317.19	765.86	448.68	36.70	291.89	255.19
117	0.795253	0.613146	309.11	719.31	410.19	41.02	306.32	265.29
129	0.775013	0.571358	301.25	672.72	371.47	45.41	320.21	274.81
141	0.755288	0.529577	293.58	626.11	332.54	49.85	333.38	283.53
165	0.717332	0.446430	278.82	533.03	254.20	58.90	354.70	295.80
177	0.699076	0.405693	271.73	486.94	215.21	63.50	359.06	295.56
189	0.681284	0.365984	264.81	441.73	176.92	68.15	359.15	291.01
201	0.663944	0.327583	258.07	397.76	139.68	72.83	355.49	282.66
213	0.647046	0.290722	251.51	355.31	103.81	77.54	348.61	271.07
237	0.614523	0.222339	238.87	275.96	37.09	87.12	326.91	239.79
249	0.598856	0.191115	232.78	239.40	6.62	91.79	312.50	220.71
261	0.583562	0.162081	226.85	205.15	-21.70	96.52	295.62	199.10
273	0.568621	0.135426	221.05	173.38	-47.67	101.35	276.01	174.66
285	0.554015	0.111340	215.39	144.32	-71.07	106.29	253.63	147.35
297	0.539728	0.089980	209.85	118.16	-91.69	111.28	228.95	117.68
309	0.000000	0.000000	0.00	0.00	0.00	5983.32	991.18	-4992.14

Table A.1.: Yearly outputs for PVFP-sensitivity analysis based on an interst rate of 2%.

Appendix B.

Code

```
# The pseudocode for the nnls-algoritm can be found at
# https://en.wikipedia.org/wiki/Non-negative_least_squares
nnls_VMF <- function(A, b, Modellpunkte = 0, Fehlertol = 1e-10, MainloopMax =
    10000, InnerloopMax = 1000, Ausgabe = 0) {

  N <- dim(A)[2]
  # Count the number of zeros per row; if for one row there are N zeros all
      policies
  # produce zeros for that cashflow --> not a relevant information.
  AnzNull <- rowCounts(A, value = 0)
  # find those rows which have non-zero values in A and b simultaneously
  # Those rows have relevant information
  Zeilen <- (AnzNull != N) & (b != 0)
  # Reduce A and b to those rows with relevant information
  A <- A[Zeilen,]
  b <- b[Zeilen]

  # Initialize
  P <- NULL
  R <- 1:N
  x <- rep(0, N)

  # Log the number of non-zero entries in x by counting the elements in P
  nonZeroEntries <- length(P)

  # Log the development of the residuals
  fitted <- A %*% x
  residual_tmp <-  (b - fitted)^2 %>% sum()

  # Calculate initial Gradient "t(A) %*% (b - A %*% x)"
  # Since in first iteration x is zero this simplifies to "t(A) %*% b"
  w <- crossprod(b, A)
  # Find highest descent
  wmax <- max(w)
  Stellen <- -log(Fehlertol, 10)
  # initialize counter for Main loop
  Mainloop <- 0

  # initialise variables for the exit-reason
  jprev <- 0
  Stabiler_Zustand <- 0
  Abbruch_Grund <- 0

```

```
42  # define the tolerance based on the machine precision
43  TestTol <- .Machine$double.eps * 1000
44
45  # When the user doesn't restrict the number of modelpoints then
46  # we set the number of maximal modelpoints to the number of varibles we
47  # want to match.
48  if (Modellpunkte == 0) {
49    Modellpunkte <- dim(A)[1]
50  }
51  ModellpunkteMax <- min(2*Modellpunkte, dim(A)[1])
52
53  # initialize progress bar
54  pb <- winProgressBar(title = "Progress",
55                       min = 0,
56                       max = Modellpunkte,
57                       width = 600)
58
59  # Mainloop
60  while ((length(P) < ModellpunkteMax) &
61         (Mainloop < MainloopMax) &
62         (Stabiler_Zustand == 0) &
63         (wmax > Fehlertol)) {
64
65    # log development of residuals
66    fitted <- A %*% x
67    resid <-  b - fitted
68
69
70    Stabiltest <- 0
71    while (Stabiltest == 0) {
72      j <- which.max(w[R])
73      Kandidat <- R[j]
74      Ptemp <- unique(sort(rbind(P, Kandidat)))
75      # check if det below tolerance, if this is the case then we can't invert
76      # t(A) %*% A restricted to the columns of P
77      if (det(crossprod(A[,Ptemp])) <= TestTol) {
78        w[Kandidat] <- 0
79      } else {
80        Stabiltest <- 1
81      }
82    }
83
84    # if the new candidate is the same as in the last iteration of the loop
85    # a stable state is reached and the optimization is finished
86    if (R[j] == jprev) {
87      #Stabiler_Zustand <- 1
88    }
89
90    #if (Stabiler_Zustand == 0) {
91    if (0 == 0) {
92      jprev <- R[j]
93
94      # move index form R to P
95      P <- unique(sort(rbind(P, R[j])))
96
97      # define restricted matrix A^P
98      AP <- A[, P]
99      s <- rep(0, N)
100     # compute the restricted least squres problem
101     sP <- solve( crossprod(AP), crossprod(AP, b))
```

```
      #       sP <- solve( t(AP) %*% AP, t(AP) %*% b)
      #       sP <- as.matrix(lsfit( AP, b, intercept = FALSE)[[1]])
      # assign the results of the restriced least squres problem to the
      # entries of s indexed in P.
      s[P] <- sP
      Innerloop <- 0

      # Innerloop
      while ((min(sP) <= 0) & (Innerloop < InnerloopMax)) {
        # find indices for negative values
        neg <- P[sP <= 0]
        alpha <- min (x[neg] / (x[neg] - s[neg]), na.rm = TRUE)
        x[P] <- x[P] + alpha * (s[P] - x[P])

        temp <- (round(x[P], Stellen) != 0)
        AP <- AP[,temp]
        P <- P[temp]
        sP <- solve( crossprod(AP), crossprod(AP, b))
        #         sP <- solve( t(AP) %*% AP, t(AP) %*% b)
        #         sP <- as.matrix(lsfit( AP, b, intercept = FALSE)[[1]])
        s <- rep(0, N)
        s[P] <- sP
        Innerloop <- Innerloop + 1
      } # End Innerloop

      R <- (1:N)[-P]
      x <- s
      w <- crossprod(b - AP %*% sP, A)
      wmax <- max(w)
      Mainloop <- Mainloop + 1


      # logging
      nonZeroEntries <- rbind(nonZeroEntries, length(P))
      # Log the development of the residuals
      fitted <- A %*% x
      residual_tmp <-  c(residual_tmp, (b - fitted)^2 %>% sum())



      setWinProgressBar(
        pb,
        length(P),
        title = paste0("Policies ", length(P), " of max. ",Modellpunkte,
                       " Mainloop: ", Mainloop, " max(w): ", round(wmax, 4)
        )
      )
    } # End Stabiler_Zustand
  } # End Mainloop

  # close progress bar
  close(pb)

  # Apply fit and calculate residuals
  fitted <- A %*% x
  resid <-  b - fitted

  if (Stabiler_Zustand == 1) {
    Abbruch_Grund <- "stable state"
  }
```

```
162   else if (Mainloop >= MainloopMax) {
163     Abbruch_Grund <- "main loop max reached"
164   }
165   else if (max(w) <= Fehlertol) {
166     Abbruch_Grund <- "gradient too small"
167   }
168
169   xepsilon <- x
170   Pepsilon <- (1:N)[xepsilon > Fehlertol]
171   xepsilon[xepsilon <= Fehlertol] <- 0
172
173   nnls_VMF.out <- list(
174     x = x,
175     xepsilon = xepsilon,
176     deviance = sum(resid ^ 2),
177     residuals = resid,
178     fitted = fitted,
179     mw = max(w),
180     w = w,
181     passive = P,
182     Pepsilon = Pepsilon,
183     loops = Mainloop,
184     nsetp = length(P),
185     Abbruch = Abbruch_Grund,
186     Entwicklung = nonZeroEntries,
187     Residual_Entwicklung = residual_tmp
188   )
189
190   class(nnls_VMF.out) <- "nnls"
191   nnls_VMF.out
192 } # End nnls_VMF
```

Listing B.1: Implementation of Lawson and Hanson NNLS algorithm in R

Bibliography

[1] Joseph J Allaire. *Deep Learning with R* -. Birmingham: Manning Publications Company, 2018. ISBN: 978-1-617-29554-6 (cit. on pp. 64, 66).

[2] E. Anderson et al. *LAPACK Users' Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-447-8 (paperback) (cit. on p. 54).

[3] David Arthur and Sergei Vassilvitskii. *How slow is the k-means method?* ACM, 2006 (cit. on p. 25).

[4] Richard E Bellman. *Adaptive control processes: a guided tour*. Princeton university press, 1961 (cit. on p. 34).

[5] Kevin Beyer et al. "When is "nearest neighbor" meaningful?" In: *International conference on database theory*. 1999, pp. 217–235 (cit. on p. 36).

[6] CEIOPS. *CEIOPS' Advice for Level 2 Implementing Measures on Solvency II: Technical provisions*. Oct. 2009. URL: https://eiopa.europa.eu/CEIOPS-Archive/Documents/Advices/CEIOPS-L2-Final-Advice-on-TP-Best-Estimate.pdf (cit. on p. 6).

[7] European Commission. *Official Journal of the European Union*. Vol. 58. L. Jan. 2015 (cit. on p. 7).

[8] European Commission. *Regulatory process in financial services*. URL: https://ec.europa.eu/info/business-economy-euro/banking-and-finance/financial-reforms-and-their-progress/regulatory-process-financial-services/regulatory-process-financial-services_en (cit. on p. 6).

[9] Wikipedia contributors. *Gradient descent*. [Online; accessed 2020-01-05]. 2020. URL: https://en.wikipedia.org/wiki/Gradient_descent (cit. on p. 77).

[10] Wikipedia contributors. *Rekurrentes neuronales Netz*. [Online; accessed 2020-01-01]. 2020. URL: https://de.wikipedia.org/wiki/Rekurrentes_neuronales_Netz (cit. on p. 71).

[11] Götz Cypra, Mario Hörig, and Daniel Hohmann. "DEEP LEARNING TECHNIKEN IM EINSATZ, Anwendungen im Kontext von Economic Capital und Cash Flow Projektionsmodellen." 2019 (cit. on p. 91).

[12] Pedro Domingos. "A few useful things to know about machine learning." In: *Communications of the ACM* 55.10 (2012), pp. 78–87 (cit. on p. 35).

[13] Jack J Dongarra et al. *LINPACK users' guide*. SIAM, 1979 (cit. on p. 55).

[14] EIOPA. *Final Report on Public Consultation No. 14/036 on Guidelines on valuation of technical provisions*. Feb. 2015. URL: https://eiopa.europa.eu/Publications/Guidelines/Final_Report_Val_tech_prov_GLs.pdf (cit. on p. 7).

[15] EIOPA. *Guidelines on valuation of technical provisions*. Feb. 2015. URL: https://eiopa.europa.eu/Publications/Guidelines/TP_Final_document_EN.pdf (cit. on p. 7).

[16] EIOPA. *Solvency II*. URL: https://eiopa.europa.eu/regulation-supervision/insurance/solvency-ii (cit. on p. 6).

[17] FMA. *Fakten, Trends & Strategien 2016*. 2016. URL: https://www.fma.gv.at/publikationen/fakten-trends-strategien/ (cit. on p. 9).

[18] FMA. *Solvency II*. URL: https://www.fma.gv.at/en/insurance/solvency-ii/ (cit. on p. 6).

[19] Hans U. Gerber. *Life Insurance Mathematics* -. Berlin Heidelberg: Springer Science & Business Media, 2013. ISBN: 978-3-662-03460-6 (cit. on p. 10).

[20] Anthony Goldbloom. *We've passed 1 million members*. [Online; accessed 2019-12-07]. 2019. URL: http://blog.kaggle.com/2017/06/06/weve-passed-1-million-members/ (cit. on p. 63).

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cit. on pp. 81–83, 85).

[22] Anil K Jain. "Data clustering: 50 years beyond K-means." In: *Pattern recognition letters* 31.8 (2010), pp. 651–666 (cit. on p. 21).

[23] Gareth James et al. *An Introduction to Statistical Learning - with Applications in R*. 1st ed. 2013, Corr. 6th printing 2016. Berlin Heidelberg: Springer Science & Business Media, 2013. ISBN: 978-1-461-47138-7 (cit. on p. 25).

[24] Reinhold Kainhofer, Martin Predota, and Uwe Schmock. "The new Austrian annuity valuation table AVÖ 2005R." In: (2006) (cit. on p. 11).

[25] Michael Kaltenbäck. *Analysis 1*. 2014. URL: https://www.asc.tuwien.ac.at/~funkana/skripten/ANA_I.pdf (cit. on p. 22).

[26] Michael Kaltenbäck. *Analysis 2 für Technische Mathematik*. Feb. 2015 (cit. on p. 47).

[27] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on pp. 84, 86).

[28] David Kriesel. "A brief introduction on neural networks." In: (2007) (cit. on p. 72).

[29] Charles L Lawson and Richard J Hanson. *Solving least squares problems*. Vol. 15. Siam, 1995 (cit. on pp. 38, 41).

[30] Stuart Lloyd. "Least squares quantization in PCM." In: *IEEE transactions on information theory* 28.2 (1982), pp. 129–137 (cit. on p. 24).

[31] C.M. Lund, Lawrence Livermore Laboratory, and United States. Department of Energy. *HCT : a General Computer Program for Calculating Time-dependent Phenomena Involving One-dimensional Hydrodynamics, Transport, and Detailed Chemical Kinetics*. UCRL (Series). Lawrence Livermore Laboratory, University of California, 1978 (cit. on p. 59).

[32] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. "The Planar k-Means Problem is NP-Hard." In: *WALCOM: Algorithms and Computation: Third International Workshop, WALCOM 2009, Kolkata, India, February 18-20, 2009. Proceedings*. Ed. by Sandip Das and Ryuhei Uehara. Springer Berlin Heidelberg, 2009, pp. 274–285. ISBN: 978-3-642-00202-1. URL: http://dx.doi.org/10.1007/978-3-642-00202-1_24 (cit. on p. 24).

[33] Dominik Maruszczak and Fabian Pribahsnik. "Binary Classification - A Model Comparison With Application In Email Spam Analysis." Bachelor's thesis. Vienna University of Technology, 2019 (cit. on pp. 70, 73).

[34] Glenn W Milligan and Martha C Cooper. "An examination of procedures for determining the number of clusters in a data set." In: *Psychometrika* 50.2 (1985), pp. 159–179 (cit. on p. 26).

[35] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines." In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814 (cit. on p. 70).

[36] Kasey Panetta. *Hype cycle for Emerging Technologies 2017*. Gartner, Inc. 2017. URL: https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/ (visited on 12/07/2019) (cit. on p. 63).

[37] Kasey Panetta. *Hype cycle for Emerging Technologies 2018*. Gartner, Inc. 2018. URL: https://www.gartner.com/smarterwithgartner/5-trends-emerge-in-gartner-hype-cycle-for-emerging-technologies-2018/ (visited on 12/07/2019) (cit. on p. 63).

[38] Kasey Panetta. *Hype cycle for Emerging Technologies 2019*. Gartner, Inc. 2019. URL: https://www.gartner.com/smarterwithgartner/5-trends-appear-on-the-gartner-hype-cycle-for-emerging-technologies-2019/ (visited on 12/07/2019) (cit. on p. 63).

[39] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003 (cit. on p. 24).

[40] EUROPEAN PARLIAMENT and THE COUNCIL. *Directive 2009/138/EC on the taking-up and pursuit of the business of Insurance and Reinsurance (Solvency II)*. Nov. 2009. URL: http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32009L0138&from=EN (cit. on pp. 6, 7).

[41] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: http://www.R-project.org/ (cit. on p. 56).

[42] Prajit Ramachandran, Barret Zoph, and Quoc V Le. "Searching for activation functions." In: *arXiv preprint arXiv:1710.05941* (2017) (cit. on p. 70).

[43] "REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)." In: *OJ* 59 (May 2016) (cit. on p. 2).

[44] Peter J Rousseeuw. "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis." In: *Journal of computational and applied mathematics* 20 (1987), pp. 53–65 (cit. on p. 30).

[45] Tracey Lien Steven Borowiec. *AlphaGo beats human Go champ in milestone for artificial intelligence*. [Online; accessed 2019-12-07]. 2016. URL: https://www.latimes.com/world/asia/la-fg-korea-alphago-20160312-story.html (cit. on p. 63).

[46] Gilbert W Stewart. *Matrix Algorithms: Volume 1: Basic Decompositions*. Vol. 1. Siam, 1998 (cit. on p. 46).

[47] The AlphaStar team. *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II*. [Online; accessed 2019-12-07]. 2019. URL: https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii (cit. on p. 63).

[48] Robert Tibshirani, Guenther Walther, and Trevor Hastie. "Estimating the number of clusters in a data set via the gap statistic." In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001), pp. 411–423 (cit. on pp. 28, 29).

[49] Andrew Trask. *Grokking Deep Learning*. 1st. USA: Manning Publications Co., 2019. ISBN: 1617293709 (cit. on p. 76).

[50] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*. Vol. 50. Siam, 1997 (cit. on p. 49).

[51] Wikipedia contributors. *AlphaGo versus Lee Sedol*. [Online; accessed 2019-12-07]. 2019. URL: https://en.wikipedia.org/wiki/AlphaGo_versus_Lee_Sedol (cit. on p. 63).