ulm university universität ulm

Fakultät für Mathematik und Wirtschafts-wissenschaften
Institut für Versicherungs-wissenschaften

Efficient valuation of a large portfolio of variable annuity contracts

Masterarbeit an der Universität Ulm

Vorgelegt von:
Stefanie Burkart
stefanie.burkart@uni-ulm.de
868233

Gutachter:
Prof. Dr. An Chen
Prof. Dr. Hans-Joachim Zwiesler

November 2020

Contents

1 Introduction

A variable annuity (VA) is a type of life insurance contract that allows the policyholder to invest in the financial market and provides guarantees at the same time. The premium is typically collected through a lump-sum payment and is invested in one or several mutual funds. Later, the insurance company returns these investment through a lump-sum payment or a series of contractually specified payments. An attractive feature of VAs is the embedded guarantees, which serve as protection of the investment against negative market fluctuations and mortality risk.

In this thesis, we aim to evaluate the insurance benefits of these guarantees for a large portfolio of VA contracts. Further, we want to calculate a risk measure, called solvency capital requirement (SCR), which specifies the amount an insurance company must hold to cover unexpected losses within the next year. In Chapter 3, we will see that the SCR calculation mainly consists of valuating the benefits given different financial market evolutions.

Since the payoff of the guarantees is often complex and path-dependent, there is generally no closed form solution for the valuation problem. In practice, insurance companies often rely on Monte Carlo (MC) simulations. A stochastic model is used to simulate future economic scenarios, and given one scenario a realization of the future benefits for one VA contract is calculated. The mean of the discounted samples is the MC estimate of the fair market value. However, using this simulation method on a large portfolio of VA contracts, is computationally demanding because every contract has to be projected over many scenarios for a long time horizon.
Within this work, we follow the objective to make the calculation more efficient in terms of runtime, while preserving the accuracy of the results as much as possible. Similar to Gan (2013), the idea is to apply machine learning methods to speed

up the estimation: First, a small subset of VA contracts is selected from the large portfolio, and MC simulation is performed on this set of representatives to calculate the quantity of interest. Then a regression model is build based on the representatives and their values. Lastly, this model is used to predict the fair values for the entire portfolio.
This approach is able to significantly decrease the computational time because it only requires a reduced number of expensive MC simulations.

The remaining of this thesis is structured as follows. Chapter 2 gives a brief description of VA products and their basic guarantees. It proceeds with a universal framework to determine the today's fair value of guaranteed benefits and presents the traditional MC simulation. Chapter 3 introduces the methodology of the SCR and describes how it can be estimated via a nested MC simulation. Chapter 4 introduces three common machine learning algorithms. In Chapter 5, numerical results of the application of the machine learning methods to the valuation and determination of the SCR are presented. Chapter 6 concludes the work.

2 Variable annuities

Variable annuities (VA) are unit-linked deferred annuities that are designed to provide a post-retirement income. At contract inception the premium is invested in a reference portfolio until a specific time T in the future. This period is called accumulation or deferment phase. The composition of the portfolio reflects the risk preferences of the policyholder. At retirement time T, also known as maturity, the payout phase starts. The insured can decide whether to take the account value of the portfolio as lump sum payment at time T or to receive it as a whole life annuity. The latter means that, according to current market conditions, the amount is converted into annual annuity payments that are paid until death of the insured. A main feature of variable annuities is that one can purchase one or more guarantees to protect the income in case of death and/or a bad fund performance. These guarantees are often referred to as GMxB, a guaranteed minimum benefit, where 'x' specifies the type. For example, 'x' can stand for death (D), accumulation (A), income (I) and withdrawal (W).
The GMDB ensures a minimum benefit in case of death during the accumulation phase. The GMAB guarantees a minimum account value at time T whereas the GMIB is only relevant when the policyholder decides for annuitization. The GMWB gives the possibility to withdraw money within certain limits.

To sum up, a variable annuity is a dynamic investment opportunity that can include protection against financial risks and early death at the same time. In this chapter, first the guarantees are explained in more detail. Next, I present a universal pricing framework to determine the today's value of the insurance benefits and show how Monte Carlo simulation is used to estimate this value.

This chapter is largely based on Bauer et al. (2008).

2.1 Guarantee types

The guarantees embedded in variable annuities can be classified in guaranteed minimum death benefits (GMDB) and guaranteed minimum living benefits (GMLB). The GMLB options consist of the guaranteed minimum accumulation benefit (GMAB), the guaranteed minimum withdrawal benefit (GMWB) and the guaranteed minimum income benefit (GMIB).

For each guarantee, we need to define a benefit base, which is used to determine the guaranteed amount. We consider three common types of benefit bases: The most basic one is the *return of premium*. The benefit base equals the invested premium and ensures that the policyholder gets back at least the initial investment. In the *(annual) roll-up* case, the benefit base is initialized with the premium and is then compounded annually with a constant interest rate i, where i is called the roll-up rate. Furthermore, the *(annual) rachet* benefit base is specified as the maximum of the account value at all past policy anniversary dates.

The GMDB guarantees a minimum payment if the insured dies before retirement time T. The dependants receive the greater of the current account value and the death benefit base. If the option is not included, they receive the account value only.
The GMAB is the simplest form of guaranteed minimum living benefits within variable annuities. If the policyholder survives the maturity T, the insurer guarantees a minimum account value. Hence, at time T there are four possibilities to choose from: The policyholder can either take or annuitize the current account value or take or annuitize the guaranteed amount. The rate, at which the amounts can be annuitized, is determined at retirement time T.
The option GMIB is similar to the GMAB, but the guarantee can only be taken if its amount is annuitized at the annuitization rate that has been specified at contract inception. A lump-sum payment of the guarantee is not possible. The GMIB can be interpreted as a guaranteed annuity that starts at time T, where the payments have already been specified at time 0.
Lastly, the GMWB rider gives policyholders the possibility to withdraw money during the contract life as long as they are alive. The benefit base of the guarantee specifies the maximum amount that can be withdrawn in total. Also the annual withdrawal amount is limited, and its maximum is typically defined by a portion

of the initial premium. At each withdrawal, the account value will be reduced by the amount of that withdrawal. If an amount higher than the current account value is withdrawn, the account value is set to zero. That is, it may happen that withdrawal is possible even if the account value already equals zero. At maturity time T, the insured receives the remaining account value if there is any. It should be noted that often there is a certain waiting period and only after that period withdrawals are possible. However, in this thesis we follow Bauer et al. (2008) and Milevsky and Salisbury (2006) and assume that withdrawals are possible from the contract inception.

2.2 Valuation framework

In this section, I present a framework to determine the fair value of one single contract according to risk-neutral valuation. Within this universal model any combination of the guarantees introduced in Section 2.1 can be evaluated. We consider a variable annuity contract with a finite, integer-valued maturity T and inception at time $t = 0$.

It is essential to understand which part of the benefit payments will be evaluated below. We are only interested in the part of the payment that is not covered by the policyholder's account, i.e. the amount that the insurance company has to pay with its reserves. This equals zero if the account value exceeds the guaranteed amount, otherwise it is defined as the difference between the guaranteed amount and the account value. In the following, we will refer to this part of the benefit as *guaranteed benefit*. The objective is to determine the today's value of the guaranteed benefits, denoted by V_0.

We start with a brief introduction to the risk-neutral valuation framework followed by assumptions and notation. The risk-neutral approach allows to determine V_0 as expected discounted cash flows. In order to determine the future payments of the guaranteed benefits, the section proceeds with a framework that projects a contract along the development of financial markets and biometric risks until maturity T.

2.2.1 Risk-neural valuation

For variable annuities, the benefit payments depend on both biometric risks and financial factors. Under certain assumptions regarding these two risks, a risk-neutral pricing formula can be applied.

In this approach, we assume that mortality follows some best estimates. Given a large portfolio of VA contracts, the mortality risk is diversified and hence it is plausible to assume that mortality follows its expectation. Further, we suppose that death can only happen at anniversary dates $t = 1, 2, \ldots$ of the contract. Let x_0 be the age of the policyholder at inception time $t = 0$. The term ${}_tp_{x_0}$ denotes the probability for an x_0-year old to survive t years, whereas q_{x_0+t} denotes the probability for an (x_0+t)-year old to die within the next year. Hence, ${}_tp_{x_0} \cdot q_{x_0+t}$ is the probability that the policyholder dies at time $t+1$. Let further ω represent the limiting age of the mortality table. This means that nobody can get older than age ω.

For the financial market, we assume the existence of a probability space equipped with a risk-neutral probability measure Q under which payment streams can be valuated as expected discounted values. Note that the existence of such a measure also implies the arbitrage-freeness of the financial market. Further, we suppose that there is a bank account $(B_t)_{t\in[0,T]}$ with $B_0 = 1$, $B_t = e^{rt}$ and constant short rate of interest r. We choose this bank account as numéraire. The underlying mutual fund of the variable annuity is denoted by S_t.

We also follow the common assumption that financial markets and biometric events are independent. This allows us to apply risk-neutral pricing and express the today's value of the guaranteed benefits from a VA contract by

$$V_0 = \sum_{t=1}^{\omega-x_0} {}_{t-1}p_{x_0} \cdot q_{x_0+t-1} \mathbb{E}_Q \left[e^{-rT} GB_T(t) \right] \tag{2.1}$$

where the expectation is taken with respect to the risk-neutral measure Q of the financial market, and $GB_T(t)$ reflects the total maturity-value of all guaranteed benefits for this contract when the policyholder dies at time t. That is, for each possible time of death t, all guaranteed benefits are accumulated and compounded until maturity T to obtain the time-T value $GB_T(t)$.

In the following, we aim to create a framework, where the value $GB_T(t)$ of the guaranteed benefits can be derived depending on a stock evolution and the occurrence of death at time t. In particular, the total value $GB_T(t)$ will be decomposed into the death benefit $D_T(t)$, the withdrawal benefit $W_T(t)$ and the survival benefit $L_T(t)$, which result from the guarantees GMDB, GMWB and GMAB and/or GMIB, respectively.

2.2.2 Contract assumptions and notation

In this part, we present further assumptions regarding the insurance contract and introduce the corresponding notation.

First, we assume a rational policyholder. That is, if there is a choice, he or she will always take the option with the higher value. Hence, we can say that the payment to the insured is always the maximum of all possible payment streams.

The premium P is a single, up-front payment and is solely invested in the underlying mutual fund S_t. The term A_t denotes the account value of the corresponding policyholder's individual portfolio.
For simplicity, we exclude all up-front fees. This yields $A_0 = P$. The fee for the guarantees, represented by φ, is charged proportionally to the account value and is deducted continuously from A_t every year.

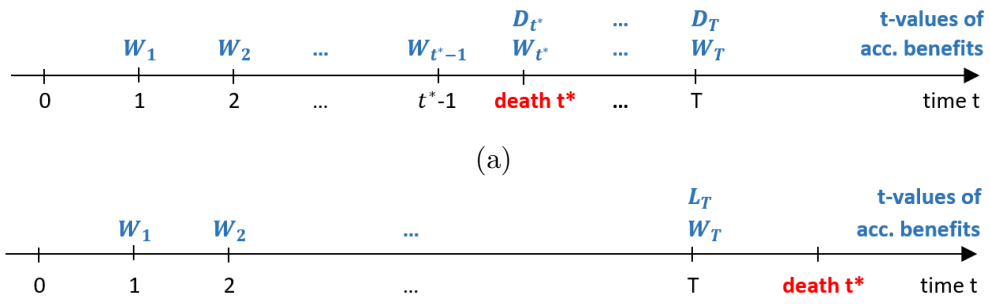
Next, we introduce notations to incorporate the benefit base of each guarantee. For the GMDB option, G_t^D denotes the guaranteed minimum death payment at time t. Thus, the payment to the dependants in case of death at time t is given by $\max\{A_t, G_t^D\} = A_t + \max\{0, G_t^D - A_t\}$. As mentioned previously, the second part of the decomposition equals the guaranteed benefit and is our value of interest. If the GMDB option is included in the contract, we set $G_0^D = A_0$, otherwise $G_0^D = 0$.
G_T^A is the minimum account value guaranteed to the policyholder if he or she is still alive at time T. To account for possible changes of this guarantee, e.g. in the rachet or roll-up benefit base, we consider the development G_t^A, $t = 1, \ldots, T$. If the GMAB option is included, we set $G_0^A = A_0$, otherwise $G_0^A = 0$.
Analogously, G_T^I specifies the guaranteed amount that is annuitized at maturity T with pre-specified conversion rate g. This yields annual annuity payments of $G_T^I g$. The evolution of the guaranteed value is denoted by G_t^I. If the contract contains

a GMIB guarantee, $G_0^I = A_0$, otherwise $G_0^I = 0$.
In the GMWB option, G_t^W represents the remaining total amount that can be withdrawn after time t and is initialized with $G_0^W = A_0$. In addition, G^E denotes the maximum amount that can be withdrawn annually and is typically set to a portion $x_w \in (0,1)$ of the premium, i.e. $G^E = x_w A_0$. This amount is assumed to be constant over time. In reality, each year the insured can choose the amount, denoted by E_t, he or she wants to withdraw within the limits, and clearly this behavior is not known to the insurance company in advance. Here, we consider a deterministic withdrawal strategy and in particular suppose that the policyholder takes the maximum withdrawal amount that is possible every year. If the contract is without a GMWB, let $G_0^W = G^E = 0$.

In order to valuate the guaranteed benefits, we define two virtual accounts W_t and D_t, which reflect the time-t value of all guaranteed benefits paid until time t and arising from the GMWB and GMDB option, respectively.
The withdrawal account W_t incorporates the withdrawals up to time t. Every withdrawal benefit is credited to the account, and at every time step the account is compounded with the risk-free interest rate r. Similarly, the death benefit account D_t is the compounded value of a death benefit paid up to time t. Both accounts are initialized with zero, i.e. $W_0 = 0$ and $D_0 = 0$.
In the end, the accumulated and compounded maturity-values W_T and D_T will be used in the risk-neutral pricing formula shown in Equation (2.1). The third one, the living or survival benefit L_T, is the guaranteed amount resulting from the options GMAB and GMIB. It is only paid in case of survival of the full contract life and will be defined in Section 2.2.5.

A visualization of the evolution of the guaranteed benefits can be seen in Figure 2.1, where we consider two cases: death up to time T and death after time T. Whenever a benefit value is not depicted in the time line, it equals 0. That is, in case of death at $1,\dots,T$ (see 2.1a) there is no maturity benefit L_T, whereas if death happens after T (see 2.1b) we have $D_T = 0$.
Further, it should be mentioned that in case of death at time t^*, no new withdrawals are credited to the withdrawal benefit account during $[t^*,\dots,T]$. The evolution in this time interval only consists of compounding the value W_{t^*-1}. Similarly, the guaranteed death benefit D_{t^*} is compounded up to maturity T.

Altogether, we have introduced the variables $\{A_t, W_t, D_t, G_t^D, G_t^A, G_t^I, G_t^W\}$. This set



(a)

(b)

Figure 2.1: Time line for the evolution of the guaranteed benefit values with a distinction between (a) time of death up to maturity T and (b) death after maturity.

is also called state variables as it fully describes the evolution of the contract.

Lastly, we exclude lapses of the insurance contracts. Therefore, the only two events that can occur during the contract life are:

- death of the policyholder
- withdrawal within a GMWB option

We assume that these events only happen at policy anniversary dates, i.e. at times $t = 1, ...T$.

In order to obtain the accumulated and compounded time-T values W_T, D_T and L_T of the guaranteed benefits, the next two parts explain how the state variables of the contract are projected along a development of the stock price and depending on the time of death. We distinguish between developments during the contract year and changes at anniversary dates. The notation $(\cdot)_t^-$ and $(\cdot)_t^+$ refers to the values of the state variables immediately before and after a policy anniversary date t, respectively.

2.2.3 Development between t^+ and $(t+1)^-$

For $t = 0$, the initial values of all variables are defined in Section 2.2.2. Now, we examine the development between two anniversary dates. That is, the development from t^+ to $(t+1)^-$ for $t = 0, \ldots, T-1$.

The account value changes according to the evolution of the underlying fund and is reduced proportionally by the guarantee fee. Hence, we have

$$A_{t+1}^{-} = A_t^{+} \cdot \frac{S_{t+1}}{S_t} \cdot e^{-\varphi}. \tag{2.2}$$

In order to maintain time-t values of the virtual withdrawal and death benefit accounts, we need to compound them every year with the risk-free interest rate, i.e. $W_{t+1}^{-} = W_t^{+} e^{r}$ and $D_{t+1}^{-} = D_t^{+} e^{r}$.
Further, the benefit base of each guarantee is adjusted according to

$$G_{t+1}^{D/A/I-} = \begin{cases} G_t^{D/A/I+} & \text{, if return of premium or rachet} \\ G_t^{D/A/I+} \cdot (1+i) & \text{, if roll-up} \end{cases} \tag{2.3}$$

Updating the return of premium and roll-up bases is intuitive. With the ratchet type, before taking the maximum of account values, we must first consider a possible reduction of the account value due to withdrawal at the anniversary date $t+1$. Hence, this benefit base will be adjusted in Section 2.2.4. If an option is not included in the contract, we set $G_{t+1}^{D/A/I-} = G_t^{D/A/I+}$.
Since withdrawals are not allowed during the contract year, the process G_t^{W} remains unchanged here, i.e. $G_{t+1}^{W-} = G_t^{W+}$.

2.2.4 Transition from $(t+1)^{-}$ to $(t+1)^{+}$

At policy anniversary dates $t = 0, \ldots, T-1$, we distinguish between the following cases in order to update all state variables.

(a) **Death at time $t+1$**
The guaranteed death benefit at time $t+1$ is given by

$$D_{t+1}^{+} = \max\{G_{t+1}^{D-} - A_{t+1}^{-}, 0\} \tag{2.4}$$

In case of death there are no future benefits, and thus the account value and all remaining benefit bases will be zero. It is not necessary to continue with the projection framework step by step. Instead, we can directly set $A_T^{+} = 0$ and $G_T^{A/I/W/D} = 0$. Due to the occurrence of death, no withdrawal is possible

at time $t+1$ and we get $W_{t+1}^{+} = W_{t+1}^{-}$.
Lastly, the benefit accounts need to be compounded until maturity T by $D_T^{+} = D_{t+1}^{+} e^{r(T-(t+1))}$ and $W_T^{+} = W_{t+1}^{+} e^{r(T-(t+1))}$.

(b) **Survival of $(t,t+1]$ and no withdrawal**
When no death or withdrawal benefits are paid, the benefit accounts do not change. Thus, it holds $D_{t+1}^{+} = D_{t+1}^{-}$, $W_{t+1}^{+} = W_{t+1}^{-}$ and also $A_{t+1}^{+} = A_{t+1}^{-}$.
It only remains to update the ratchet type guarantee base by

$$G_{t+1}^{D/A/I+} = \begin{cases} \max\{G_{t+1}^{D/A/I-}, A_{t+1}^{+}\} & \text{, if rachet} \\ G_{t+1}^{D/A/I-} & \text{,if return of premium or roll-up} \end{cases} \tag{2.5}$$

and we have $G_{t+1}^{W+} = G_{t+1}^{W-}$. Again, if the guarantees are not included, we set $G_{t+1}^{D/A/I+} = G_{t+1}^{D/A/I-}$.

(c) **Survival of $(t,t+1]$ and withdrawal**
By assumption, the withdrawal amount E_{t+1} at time $t+1$, is given by the maximum possible amount within the limits, which equals $E_{t+1} = \min\{G^E, G_{t+1}^{W-}\}$.
The remaining guaranteed withdrawal amount is reduced exactly by this withdrawal, i.e. $G_{t+1}^{W+} = G_{t+1}^{W-} - E_{t+1}$. The account value must also be reduced by this amount. But if the allowed withdrawal is greater than the account value, the account value is set to zero. This leads to $A_{t+1}^{+} = \max\{0, A_{t+1}^{-} - E_{t+1}\}$.
The withdrawal benefit is credited to the withdrawal account by

$$W_{t+1}^{+} = W_{t+1}^{-} + \max\{E_{t+1} - A_{t+1}^{-}, 0\} \tag{2.6}$$

while the death benefit account remains the same, i.e. $D_{t+1}^{+} = D_{t+1}^{-}$.
In order to take the withdrawal into consideration for the other guarantees, we also need to adjust the benefit bases of the death and survival guarantees. Usually, they are reduced by the same rate as the account value, which is called *pro rata* adjustment. For the ratchet types, we additionally need to

consider the maximum with respect to the current account value. We set

$$G_{t+1}^{D/A/I+} = \begin{cases} \frac{A_{t+1}^{+}}{A_{t+1}^{-}} \cdot G_{t+1}^{D/A/I-} & \text{, if return of premium or roll-up} \\ \max\{\frac{A_{t+1}^{+}}{A_{t+1}^{-}} \cdot G_{t+1}^{D/A/I-}, A_{t+1}^{+}\} & \text{, if rachtet} \end{cases} \tag{2.7}$$

and if an option is not included, we can keep $G_{t+1}^{D/A/I+} = G_{t+1}^{D/A/I-}$.

If the withdrawal amount exceeded the account value, we have set $A_{t+1}^{+} = 0$. This event is also known as ruin of the fund. Consequently, the pro rata adjustment also sets the benefit bases, with the exception of the withdrawal base, to 0. Note that G_{t+1}^{W+} can be greater than zero and thus withdrawals are still possible in the future. Hence, to deal with an account value being zero, we define $G_{t+1}^{D/A/I+} = 0$ if $A_{t+1}^{-} = 0$.

2.2.5 Survival benefit

By projecting the contract from time 0 to T^{+} according to the Sections 2.2.3 and 2.2.4, we have constructed the accumulated and compounded withdrawal and death benefits W_T and D_T, respectively.

Additionally, after surviving the entire contract the policyholder receives a guaranteed living benefit with maturity value L_T. If the contract only includes a GMDB and/or a GMWB option, simply the (remaining) account value at maturity is paid back and hence there is no guaranteed amount. For a contract including a GMAB or a GMIB, we now define the minimum guaranteed survival benefits.

The amount of the guaranteed benefit for a GMAB option equals $\max\{G_T^{A+} - A_T^{+}, 0\}$ and is denoted by L_T^A. Although the insured person can choose between a lump-sum payment and a whole-life annuity, the time-T value of the guaranteed benefit equals L_T^A in both cases.
To see this, we introduce the actuarial notation $\ddot{a}_{x_T} = \sum_{k=0}^{\omega - x_T} {}_k p_{x_T} e^{-rk}$, which indicates the value of an annuity paying one unit of money every year and starting at age x_T. When the policyholder chooses to receive the annuity, the amount G_T^{A+} is annuitized at current rates yielding annual payments of $G_T^{A+} \frac{1}{\ddot{a}_{x_T}}$ until death.

Therefore, the time-T value of the whole-life annuity is given by

$$\sum_{k=0}^{\omega-x_T} {}_kp_{x_T} e^{-rk} G_T^{A+} \frac{1}{\ddot{a}_{x_T}} = G_T^{A+} \frac{1}{\ddot{a}_{x_T}} \sum_{k=0}^{\omega-x_T} {}_kp_{x_T} e^{-rk} = G_T^{A+} \frac{1}{\ddot{a}_{x_T}} \ddot{a}_{x_T} = G_T^{A+} \tag{2.8}$$

and coincides with the value of the lump-sum payment. Consequently, the guaranteed benefit in this case also equals L_T^A.

With a GMIB, the policyholders can only take the guarantee if they annuitize the guaranteed amount G_T^{I+} at guaranteed annutization rate g, which has been specified at contract inception. In this case, the policyholder receives yearly payments of $G_T^{I+} g$ as long as he or she is alive. The time-T value of these annuity payments is then given by

$$\sum_{k=0}^{\omega-x_T} {}_kp_{x_T} e^{-rk} G_T^{I+} g = G_T^{I+} g \, \ddot{a}_{x_T} \tag{2.9}$$

Typically, the rate g is chosen based on conservative assumptions and is supposed to lead to $\ddot{a}_{x_T} g < 1$. Finally, the guaranteed income benefit is specified by $L_T^I = \max\{G_T^{I+} g \, \ddot{a}_{x_T} - A_T^+, 0\}$.

When the contract contains both, GMAB and GMIB, the survival benefit is simply set to the maximum of their guarantee values.
In total, we obtain

$$L_T = \begin{cases} 0 & \text{,if GMDB and/or GMWB only} \\ \max\{G_T^{A+} - A_T^+, 0\} =: L_T^A & \text{, if GMAB included} \\ \max\{G_T^{I+} g \, \ddot{a}_{x_T} - A_T^+, 0\} =: L_T^I & \text{, if GMIB included} \\ \max\{L_T^A, L_T^I\} & \text{,if GMAB and GMIB included} \end{cases} \tag{2.10}$$

Note that in case of death before time T, the account value and guarantee bases have been set to zero. Hence, the survival benefit L_T equals zero and is well defined by (2.10).

Finally, given the time of death t, the guaranteed benefits $L_T(t)$, $W_T(t)$ and $D_T(t)$ are specified for a development of the underlying fund S_t.

Overall, according to Equation (2.1) the today's value of all guaranteed benefits

for one contract can be written by

$$\begin{aligned} V_0 &= \sum_{t=1}^{\omega-x_0} {}_{t-1}p_{x_0} q_{x_0+t-1} \mathbb{E}_Q \left[e^{-rT} \left(L_T(t) + W_T(t) + D_T(t) \right) \right] \\ &= \sum_{t=1}^{T} {}_{t-1}p_{x_0} q_{x_0+t} \mathbb{E}_Q \left[e^{-rT} \left(L_T(t) + W_T(t) + D_T(t) \right) \right] \\ &\quad + {}_T p_{x_0} \mathbb{E}_Q \left[e^{-rT} \left(L_T(T+1) + W_T(T+1) + D_T(T+1) \right) \right] \end{aligned} \tag{2.11}$$

To understand the second equation, let us consider $t \in \{T+1\ldots,\omega\}$. In this case, the insured survived the contract, and it is clear that for different t all the withdrawal and death benefits coincide. Hence, we have $W_T(t) = W_T(T+1)$ and $D_T(t) = D_T(T+1)$. Due to the fact that we have deterministic mortality probabilities, the time-T value of the survival benefit is fixed at time T and we can also write $L_T(t) = L_T(T+1)$. The probability of surviving the entire contract is given by ${}_T p_{x_0}$, and together this yields the second part of (2.11).

Note that by construction we have $L_T(t) = 0 \; \forall t \leq T$ and $D_T(t) = 0 \; \forall t > T$.

2.3 Monte Carlo simulation

Even though the fair value in (2.11) is deterministic, there is in general no closed-form solution to calculate the expected value due to the complex dependency on the underlying mutual fund. Therefore, one has to apply numerical methods. In this section I present how Monte Carlo simulation can be used in order to approximate the today's value V_0.

The basic idea is that, if we knew the future stock evolution during $[0,T]$, all guaranteed benefits would be deterministic for each time of death t, and hence the today's fair value could be directly obtained. That is, after randomly generating a large number, say J, of independent stock realizations $s^{(j)}$, $j = 1,\ldots,J$, we compute the corresponding benefits and obtain a fair value $v_0^{(j)}$ for each stock evolution $s^{(j)}$. Averaging these fair value samples yields an estimate for our value of interest. This approach is depicted in Algorithm 1.

Algorithm 1: Traditional Monte Carlo Simulation

Output: fair value V_0

Sample J risk-neutral stock paths $s^{(1)},\dots,s^{(J)}$ for the time interval $[0,T]$;
for $j = 1,\dots,J$ **do**
  Determine $v_0^{(j)}$ via Equation (2.15) w.r.t. evolution $s^{(j)}$;
end
$V_0 = \frac{1}{J}\sum_{j=1}^{J} v_0^{(j)}$;

2.3.1 Simulation of stock paths

In order to determine market values, we need to generate J stock paths under the risk-neutral probability measure Q. There exist several equity models to describe a fund evolution. As common in this context, we assume that the fund evolves according to a geometric Brownian motion. That is, the dynamics of S_t is described by

$$dS_t = rS_t dt + \sigma S_t dW_t^Q \tag{2.12}$$

where W_t^Q is a Brownian motion under the risk-neutral measure Q. The drift r and volatility σ are constants. Note that under Q-dynamics the drift of the stock process equals the risk-free rate.

By Itô's lemma, the closed-form solution of the differential equation is given by

$$S_t = S_{t-1} \cdot \exp\left(r - \frac{1}{2}\sigma^2 + \sigma(W_t^Q - W_{t-1}^Q)\right) \quad t = 1,2,\dots \tag{2.13}$$

with initial value S_0. Without loss of generality we assume $S_0 = 1$.
According to the definition of a standard Brownian motion, the increments $W_t^Q - W_{t-1}^Q$ are independent and standard normally distributed under measure Q.
Thus, in order to construct J realizations of the process $(S_t)_{t=1,\dots,T}$, we generate independent samples of a standard normal random variable $z_t^{(j)}$ for $j = 1,\dots,J$ and $t = 1,\dots,T$.
For each $j = 1,\dots,J$, a sample path $s^{(j)} = (s_0^{(j)},\dots,s_T^{(j)})$ is obtained by

$$\begin{aligned} s_0^{(j)} &= 1 \\ s_t^{(j)} &= s_{t-1}^{(j)} \cdot \exp\left(r - \frac{1}{2}\sigma^2 + \sigma \cdot z_t^{(j)}\right), \quad t = 1,\dots,T \end{aligned} \tag{2.14}$$

2.3.2 Monte Carlo estimation

Given an evolution $s^{(j)}$, we can generate samples of the accumulated and compounded maturity benefits $\ell_T^{(j)}(t)$, $w_T^{(j)}(t)$ and $d_T^{(j)}(t)$ for one contract and all times of death t, as described in Section 2.2.
The time zero value of the guaranteed benefits linked to path $s^{(j)}$ is then given by

$$\begin{aligned} v_0^{(j)} = \sum_{t=1}^{T} {}_{t-1}p_{x_0} q_{x_0+t} e^{-rT} \left(\ell_T^{(j)}(t) + w_T^{(j)}(t) + d_T^{(j)}(t) \right) \\ + {}_T p_{x_0} e^{-rT} \left(\ell_T^{(j)}(T+1) + w_T^{(j)}(T+1) + d_T^{(j)}(T+1) \right) \end{aligned} \tag{2.15}$$

By averaging the independent realizations $v_0^{(1)}, \ldots, v_0^{(J)}$, we obtain the Monte Carlo estimate

$$V_0 = \frac{1}{J} \sum_{j=1}^{J} v_0^{(j)} \tag{2.16}$$

which is, by the law of large numbers, an approximation of (2.11).
It is important to understand that within this simulation framework, for each contract we have to calculate the benefits

- for all times of death $t = 1, \ldots, T+1$ and
- for all paths $s^{(j)}, j = 1, \ldots, J$.

The maturity T is fixed, but characteristically quite long for a life insurance contract. On the other hand, the number J of simulations is defined by the user. Clearly, the more realizations we simulate, the better will be the estimate. So when applying Monte Carlo simulation one has to find a balance between accuracy and computational costs.

The goal of this thesis is to evaluate a large portfolio of n variable annuity contracts, represented by $X_1, \ldots, X_n$. Let $v_0(X_i)^{(j)}$, $j = 1 \ldots, J$ denote the corresponding samples, which yield the MC estimate $V_0(X_i)$.
As n is large, one could already suspect that a MC simulation on all contracts takes quite long. Let us say we have $n = 10000$ contracts, $J = 1000$ paths and a maturity of $T = 25$. The number of annual projections that have to be done for valuation equals $n \cdot N \cdot T = 2.5 \cdot 10^8$. Even though one can apply parallel implementation and valuate distinct contracts simultaneously, the computational time

is still high.

This downside encourages to make use of machine learning methods in order to design a more efficient valuation algorithm. These methods will be presented in Chapter 4.

3 SCR calculation via nested simulation

The Solvency II Directive is a regulatory framework for insurance companies within the European Union and has been in force since January 2016. An important part of the directive is the calculation of the so-called solvency capital requirement (SCR). It is a risk capital that the insurer must hold to cover unexpected losses within the next year.
To calculate the required capital, insurance companies are allowed to use a standard formula as straightforward approximation. Alternatively, they can develop a company-specific internal model based on the market-consistent valuation of assets and liabilities to accurately reflect their risk situation.

In the following, we aim to do the latter. Section 1 introduces a mathematical framework, in which the SCR of one period is determined for a company offering variable annuities. We focus solely on market risks. In Section 2, a nested approach is described to estimate the risk capital based on Monte Carlo simulations.

This chapter is largely based on Bauer et al. (2012).

3.1 The solvency capital requirement

Under the regulatory framework Solvency II, the solvency capital requirement for one period is defined as the amount of capital that a company needs to survive the next year with probability of at least 99.5%.

Mathematically, this is expressed by

$$SCR = \arg\min_x \{\mathbb{P}(AC_1 \geq 0 | AC_0 = x) \geq 99.5\%\} \tag{3.1}$$

where the *available capital* AC_t is defined by

$$AC_t = MVA_t - MVL_t, \qquad t = 0, 1 \tag{3.2}$$

The terms MVA_t and MVL_t denote the market value of the assets and liabilities at time t, respectively. The available capital represents the amount of available financial resources that can serve as buffer against risks. Under Solvency II, it is also called *own funds*.

The implicit definition in (3.1) is difficult to evaluate in practice. Therefore, Bauer et al. (2012) introduce a simpler and approximately equivalent definition based on the one-year loss function Δ, evaluated at time zero,

$$\Delta = AC_0 - \frac{AC_1}{1+r} \tag{3.3}$$

where r is the one-year risk-free rate.

The SCR is then given by

$$SCR = \arg\min_x \{\mathbb{P}(\Delta > x) \leq 0.05\%\} \tag{3.4}$$

We will always use definition (3.4) in the following. Note that the SCR simply equals the α-quantile of the loss variable Δ at confidence level $\alpha = 99.5\%$. That is, the SCR corresponds to the value of risk. The probability that the loss over the next year exceeds the SCR is at most 0.05%.

In order to determine the SCR, the quantity of interest is the available capital for time $t = 0$ and $t = 1$. From Equation (3.2), we see that the calculation of the available capital requires a market consistent valuation of both assets and liabilities. The valuation of assets is usually straightforward, whereas the valuation of life insurance liabilities is challenging due to the complex financial structure as we have seen previously. Thus, the difficulty in determining the SCR largely originates in the difficulty of valuating the liabilities. Within this thesis it is not possible to provide a realistic framework for both assets and liabilities. Therefore, similarly to Hejazi and Jackson (2017), we assume a simple asset structure and restrict the insurance business to the portfolio of variable annuities. For $t = 0$ and $t = 1$, we consider the balance sheet depicted in Table 3.1.

Assets	Liabilites
M_t	MVL_t
A_t	AC_t
MVA_t	MVA_t

Table 3.1: Simplified balance sheet

We assume that the insurance company does not hedge any risks and invests the shareholders' total money, denoted by M_0 at time 0, at risk-free interest rate r. Further, we have the initial account value of the reference portfolio from all variable annuity contracts, denoted by A_0. Note that this amount does actually not belong to the insurer's assets. It will be fully paid back to the policyholders. Hence, exactly the same amount A_0 also occurs on the liability side as part of the insurance liabilities MVL_0. MVL_0 also consists of the time zero value of the guaranteed benefits for the whole portfolio, denoted by V_0. Therefore, we can rewrite $MVL_0 = A_0 + V_0$.

The same holds for time $t = 1$. The only value known at time 0 is the shareholders' money $M_1 = M_0(1+r)$. All other time-1 market values are random at time 0.

In total, under this setting we obtain

$$\begin{aligned} AC_0 &= MVA_0 - MVL_0 = (M_0 + A_0) - (A_0 + V_0) = M_0 - V_0 \\ AC_1 &= MVA_1 - MVL_1 = (M_1 + A_1) - (A_1 + V_1) = M_1 - V_1 = M_0(1+r) - V_1 \end{aligned} \tag{3.5}$$

Together with (3.3) this yields

$$\Delta = AC_0 - \frac{AC_1}{1+r} = (M_0 - V_0) - \frac{M_0 \cdot (1+r) - V_1}{1+r} = -V_0 + \frac{V_1}{1+r} \tag{3.6}$$

and we see that the loss does not even depend on the asset amount M_0. The randomness of the loss variable Δ comes from the randomness of the time-1 value of the guaranteed benefits V_1.

Therefore, in order to obtain the SCR by assessing the one-year loss distribution, we need to calculate the current market value of all guaranteed benefits and the distribution of their market value in one year.
In the next section, it is explained how this task can be solved within the nested simulation approach.

3.2 Nested Monte Carlo simulation

Given Equation (3.4), we can calculate the SCR by assessing the distribution of the one-year loss Δ. Since the true loss distribution is not known, we need to rely on its empirical distribution and construct realizations of the random variable Δ by a nested MC simulation approach.

First, we need to set the mathematical framework. Let T be the maturity of the longest-term policy in our portfolio. For the financial market, we assume the existence of a complete probability space (Ω, F, P) equipped with a filtration $F = (\mathscr{F}_t)_{t\in[0,T]}$. Ω is the set of all possible outcomes of the financial market while P is the real-world probability measure. $\mathscr{F}_t$ represents all information about the financial market up to time t. Further, in order to perform market-consistent valuation of the guaranteed benefits from VAs, we also follow the assumptions introduced in Section 2.2. In particular, we assume the existence of the risk-neutral probability measure Q to perform risk-neutral valuation.

By definition (3.4) the SCR is the 99.5%-quantile of the one-year loss distribution with respect to probability measure P.
That is, we need to investigate the real-world distribution of the random variable $\Delta : \Omega \to \mathbb{R}$. For a state $\omega \in \Omega$, Equation (3.6) yields

$$\Delta(\omega) = -V_0 + \frac{V_1(\omega)}{1+r} \tag{3.7}$$

As V_0 and r are deterministic, assessing the distribution of Δ corresponds to assessing the real-world distribution of the $\mathscr{F}_1$-measurable random variable V_1.
We will estimate this distribution via an empirical distribution, which is specified by N real-world samples $\omega^{(1)}, \ldots, \omega^{(N)}$ for the development of the financial market over the first year. Since we have assumed that the risk-free rate r is constant, the uncertainty of the financial market is fully captured by the evolution $(S_t)_{t\in[0,1]}$ of the underlying stock from the VA contracts.

Although the market value V_1 conditional on scenario $\omega^{(\ell)}$, denoted by $V_1^{(\ell)}$, is deterministic, there exists no closed-form solution to determine this value and again we need to rely on Monte Carlo simulation. To do so, for each scenario $\ell = 1, \ldots, N$, we estimate $V_1^{(\ell)}$ via an MC simulation by considering J risk-neutral paths starting at the endpoint of $\omega^{(\ell)}$. Due to the nested structure of stock market

simulations this approach is called nested simulation and is illustrated in Figure 3.1.

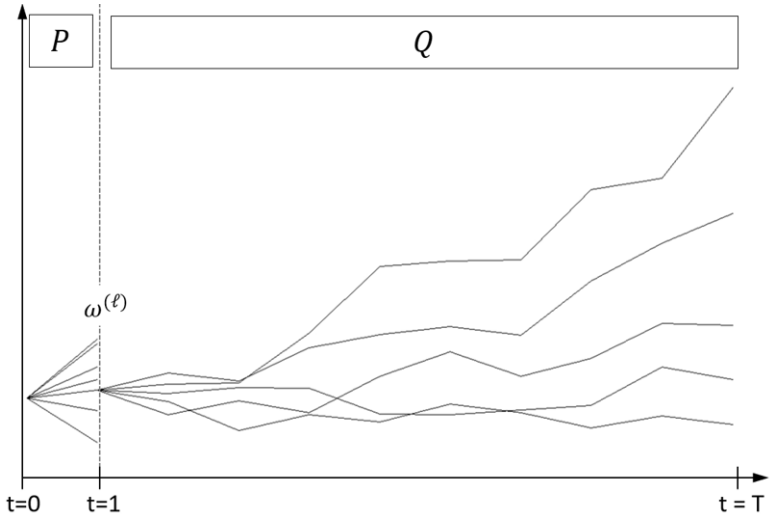


Figure 3.1: Nested simulation approach

To avoid confusion, we will always refer to the outer (real-world) samples as *scenarios* and to the the inner (risk-neutral) samples as *paths*.

Altogether, the approach is summarized in Algorithm 2. In the following parts every step is explained in detail.

Algorithm 2: Nested Monte Carlo Simulation

Output: SCR of the overall portfolio

Estimate $V_0 = V_0(X_1) + \cdots + V_0(X_n)$ using Algorithm 1;
Sample N real-world scenarios $\boldsymbol{\omega}^{(1)}, \ldots, \boldsymbol{\omega}^{(N)}$ of the underlying fund for the time interval $[0,1]$;
for $\ell = 1, \ldots, N$ **do**
    for $i = 1, \ldots, n$ **do**
        Estimate $V_1(X_i)^{(\ell)}$ via MC simulation given evolution $\boldsymbol{\omega}^{(\ell)}$;
    end
    $V_1^{(\ell)} = V_1(X_1)^{(\ell)} + \cdots + V_1(X_n)^{(\ell)}$;
    $\Delta^{(\ell)} = -V_0 + \frac{V_1^{(\ell)}}{1+r}$;
end
Approximate the SCR by the empirical 99.5%-quantile of $\Delta^{(\ell)}$;

3.2.1 Determining V_0

The term V_0 equals the sum of the today's benefit values from all contracts of the portfolio. As presented in Chapter 2, each of them is deterministic at $t = 0$ and can be computed by risk-neutral valuation. In case of variable annuities, we do not have an analytical solution of Equation (2.1) and thus have to rely on numerical methods like MC simulation.
Via the procedure described in Algorithm 1 we approximate the today's values $V_0(X_1), \ldots, V_0(X_n)$ of all contracts, and hence we obtain the today's liabilities of the guaranteed benefits from the entire portfolio by $V_0 = \sum_{i=1}^{n} V_0(X_i)$.

3.2.2 Generating stock scenarios

The solvency calculation aims to assess the risk of an insurance company depending on future developments of the financial market. For risk management purposes, we consider possible realistic evolutions of the market in the future, i.e. under the real-world measure P. In our setting, the development of the underlying stock S_t captures the entire uncertainty of the financial market. Hence, in order to assess the distribution of V_1, we need to generate N stock scenarios $\omega^{(1)}, \ldots, \omega^{(N)}$ over the first year under the measure P.
In the valuation framework of variable annuities (see Section 2.2) we have assumed that the events of death and withdrawal can only happen at policy anniversary dates. Thus, it is sufficient to model the evolution in the interval $[0,1]$ by one single realization $\omega^{(\ell)}$ at time $t = 1$.

Similar to Section 2.3.1, we assume that the dynamics of S_t is described by

$$dS_t = \mu S_t dt + \sigma S_t dW_t^P \tag{3.8}$$

but now with W_t^P being a Brownian motion under real-world measure P and drift μ being larger than the risk-free rate r. As initial value we take $S_0 = 1$.

Analogously, we obtain N scenarios $\omega^{(\ell)}$, $\ell = 1, \ldots, N$, of S_1 by generating N standard normal random variables $z^{(\ell)}$, $\ell = 1, \ldots, N$. For each $\ell = 1, \ldots, N$, we then set

$$\omega^{(\ell)} = \exp\left(\mu - \frac{1}{2}\sigma^2 + \sigma \cdot z^{(\ell)}\right) \tag{3.9}$$

as $S_0 = 1$ is assumed to hold.

3.2.3 Determining $V_1^{(\ell)}$

Given scenarios of the stock value at time 1, in this section, we aim to draw realizations of the guaranteed benefits V_1.
In general, using risk-neutral pricing, the fair value at $t = 1$ of all future guaranteed benefits for one single contract X_i is given by

$$V_1(X_i) = \sum_{t=2}^{\omega - x_0} {}_{t-1}p_{x_0} \cdot q_{x_0+t-1} \mathbb{E}_Q \left[e^{-r(T-1)} GB_T(t, X_i) | \mathscr{F}_1 \right] \tag{3.10}$$

Of course, $\mathscr{F}_1$ is not available at $t = 0$ and thus $V_1(X_i)$ is random from today's point of view.
Now, conditional on a scenario $\omega^{(\ell)}$, $\ell = 1, \ldots, N$, of the stock market at time $t = 1$, we obtain a realization of the fair value for contract X_i by

$$\begin{aligned} V_1^{(\ell)}(X_i) = &\sum_{t=2}^{T} {}_{t-1}p_{x_0} \cdot q_{x_0+t} \mathbb{E}_Q \left[e^{-r(T-1)} \left(L_T(t, X_i) + W_T(t, X_i) + D_T(t, X_i) \right) | \omega^{(\ell)} \right] \\ &+ {}_T p_{x_0} \mathbb{E}_Q \left[e^{-r(T-1)} \left(L_T(T+1, X_i) + W_T(T+1, X_i) + D_T(T+1, X_i) \right) | \omega^{(\ell)} \right] \end{aligned} \tag{3.11}$$

Again, there is no analytical solution for the calculation of $V_1^{(\ell)}(X_i)$ and we need to use Monte Carlo simulation.

Before we can perform the time-1 valuation of future guaranteed benefits, we need to project the contracts from time $t = 0$ to time $t = 1$ along each scenario $\omega^{(\ell)}$. This process is often referred to as *aging* and reflects what happens actually to the annuity contract, i.e. according to the real-world scenario.
Independent of whether real-world or risk-neutral scenarios are given, the cash flow projection model described in Section 2.2 is valid. Therefore, for each scenario we age each contract by projecting the state variables from time 0 to 1 according to the presented framework and given scenario $\omega^{(\ell)}$ for S_1 . Now, the only adjustment is that, because we do not want to include withdrawal and death benefits paid at time $t = 1$ in the valuation of future benefits, we reset the corresponding benefit

accounts to zero, i.e. $W_1 = D_1 = 0$.

Afterwards, we can apply MC simulation to determine the fair value at time 1 conditional on a stock scenario $\omega^{(\ell)}$.
As we aim to determine a market-consistent value, for each $\ell = 1,\dots,N$, we need to generate risk-neutral paths of the stock evolution starting at scenario $\omega^{(\ell)}$ at time 1. Thus, we set $s_1^{(\ell)} = \omega^{(\ell)}$ as starting value and then generate J sample paths $s^{(\ell,j)}$, $j = 1,\dots,J$ for the development in $t \in [1,T]$ according to the risk-neutral generator (see Section 2.3.1).

Now, by projecting the contracts along these risk-neutral paths, as described in Section 2.2, we obtain maturity values for the guaranteed benefits. For each contract $i = 1,\dots,n$, for each outer scenario $\ell = 1,\dots,N$, for each inner stock path $j = 1,\dots,J$ and for each time of death $t = 2,\dots,T+1$, we get realizations of the values $L_T^{(\ell,j)}(t,X_i)$, $W_T^{(\ell,j)}(t,X_i)$ and $D_T^{(\ell,j)}(t,X_i)$.
In total, Equation (3.11) together with the observed maturity values yields samples $v_1^{(\ell,j)}(X_i)$ of the fair value of guaranteed benefits at time 1.

Finally, the Monte Carlo estimate for contract X_i of the time-1 value conditional on scenario ℓ is given by

$$V_1^{(\ell)}(X_i) = \frac{1}{N}\sum_{j=1}^{N} v_1^{(\ell,j)}(X_i) \tag{3.12}$$

Adding up all contract values yields the desired estimate of the value for the entire portfolio

$$V_1^{(\ell)} = \sum_{i=1}^{n} V_1^{(\ell)}(X_i) \tag{3.13}$$

3.2.4 Estimation of SCR

In the last step, for each scenario $\omega^{(\ell)}$, we define an observation of the loss variable by

$$\Delta^{(\ell)} = -V_0 + \frac{V_1^{(\ell)}}{1+r} \tag{3.14}$$

Note that the estimates $V_1^{(\ell)}$, $\ell = 1,\dots N$, are independent and identically distributed as direct MC realizations and hence this also holds for $\Delta^{(\ell)}$ as linear

transformation of it. Therefore, the loss samples can be used to construct an empirical distribution.

By definition (3.4), the SCR is the smallest x such that $\mathbb{P}(\Delta > x) \leq 1-\alpha$ where α equals 99.5%. This is equivalent to $\mathbb{P}(\Delta \leq x) \leq \alpha$. Now, having observations of the random variable Δ, we can estimate this probability via its empirical distribution:

$$\mathbb{P}(\Delta \leq x) \approx \frac{|\{\Delta^{(\ell)} : \Delta^{(\ell)} \leq x\}|}{N} \tag{3.15}$$

Choosing x minimal such that the approximated probability is at most α, results in x being the empirical α-quantile. After ordering the N loss realizations in ascending order, yielding $\Delta_{(1)} \leq \cdots \leq \Delta_{(N)}$, the empirical α-quantile of Δ is given by the $\lfloor \alpha N \rfloor$-th element of the ordered losses. Therefore, the SCR can be approximated by

$$SCR = \Delta_{(\lfloor \alpha N \rfloor)} \tag{3.16}$$

Clearly, the more observations N we have, the better will be the SCR approximation, especially because we are interested in the tail of the loss distribution. On the other hand, increasing the number N significantly increases the running time of the nested MC simulation. For each outer scenario, we need to project J paths in the inner simulation due to the complex valuation of variable annuities.

If we have a large portfolio of VA contracts, the computational costs get extremely high and an acceptably accurate estimation of the capital requirement via nested simulation becomes impractical. Not even parallel implementation can reduce the runtime satisfactorily.
For this reason, it is very obvious that we want to make use of machine learning methods in order to speed up the estimation.

4 Introduction to machine learning methods

The downside of the Monte Carlo simulation schemes, presented in Sections 2.3 and 3.2, is the high computational time, in particular for a large number of contracts. Therefore, the aim is to apply machine learning methods to speed up the valuation and SCR determination.

The basic idea is to use the computationally expensive MC simulation only for a subset of contracts, or in case of SCR a subset of contracts and scenarios, called representatives. A prediction model is fit on the representatives and their MC values, and lastly the values of the remaining samples can be estimated via the fitted model.

In this chapter, machine learning methods are introduced that can serve as prediction models in the outlined approach.
Generally, machine learning methods can be categorized into *supervised* and *unsupervised* algorithms. In supervised learning, observations of the outcome can be used to guide the learning process, while in unsupervised learning there is no outcome measure, and the goal is to find patterns among input observations.
We deal with supervised methods as we can estimate the value of interest via MC simulation. In this thesis, three very common machine learning methods are presented: Generalized linear models, regression trees and neural networks.

Since machine learning is very popular and has many areas of application, there exist different variants and extensions of these methods. It is not possible to give a complete overview within this thesis, and thus we limit ourselves to one version for each machine learning method. The methods of this chapter will be applied in Chapter 5 on variable annuity contracts.

In the following, we assume that we have k observations $y_1, \ldots, y_k$ of the value of interest, which is referred to as *response* or *dependent variable*. For each observation i, x_{ij} denotes the value of the j-th variable, $j = 1, \ldots, p$, and we write $\boldsymbol{x_i} = (x_{i1}, \ldots, x_{ip})$. Since these p variables are used to predict the response, they are often called *predictors* or *explanatory variables*.

Our insurance portfolio is said to be a data of mixed type. That is, we have both numerical and categorical variables. The numerical variables consist of quantitative attributes like age, premium and roll-up rate. Categorical variables, also referred to as *factors*, take a fixed number of non-numerical values and there is no logical order between the categories. Each possible value of a factor is referred to as *level* or *category*. For example, the attribute gender is a categorical variable and has the levels male and female.
Some machine learning algorithms cannot handle categorical variables. In this case, these variables must be converted into binary variables, so-called *dummy variables*. For each level a column is created, where the i-th entry equals 1 if the level of observation i corresponds to the category of the column, otherwise 0.
Clearly, there is some redundancy in this dummy coding. For instance, suppose that we have a variable gender with two levels 'male' and 'female'. If we know that the policyholder is female, then the policyholder is not male, and vice versa. Hence, one column can be omitted. In general, the number of dummy-coded variables needed for one categorical variable is one less than the number of its levels. The level that is excluded in the dummy variables is called *reference* or *baseline*. After converting all factors into non-redundant binary variables, the total number of variables used in our data set is denoted by p^*.

4.1 Generalized linear models

In a classical linear model, the observations are assumed to be normally distributed around the mean that is a linear function of coefficients and explanatory variables. A generalized linear model (GLM) extends this concept in two directions: The random variables involved do not need be normal, and additionally a transformed linear relationship between means and explanatory variables is possible. GLMs were introduced by McCullagh and Nelder (1989).

The values $y_1, \ldots, y_k$ are assumed to be observations af a sequence of random variables $Y_1, \ldots, Y_k$, where the components Y_i are independently distributed with means $E(Y_i) = \mu_i$. Further, they are assumed to belong to the same member of the exponential family. The probability distribution of the response is said to determine the random component of the model.
The predictors need to be of numerical type and hence a conversion of categorical variables into dummy variables is necessary. The resulting columns are summarized in the so-called design matrix $\boldsymbol{X}$. It consists of the p^* numerical columns and a vector of ones as first column. An observation i of explanatory variables corresponds to a row $\boldsymbol{x_i}$ of $\boldsymbol{X}$. For a more compact notation, we also write $\boldsymbol{y} = (y_1, \ldots, y_k)^T$, $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_k)^T$.

The systematic component of the model describes how the explanatory variables are related to the mean of the responses. The *linear predictor* $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_k)^T$ is defined by

$$\eta_i = \beta_0 + \sum_{j=1}^{p^*} x_{ij}\beta_j \tag{4.1}$$

where $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_{p^*})^T$ are the regression coefficients, and β_0 is particularly called intercept. The parameters are unknown and have to be estimated from data. Clearly, $\boldsymbol{\eta}$ is linear in the regression coefficients.
In matrix notation, the linear predictor can be written as $\boldsymbol{\eta} = \boldsymbol{X\beta}$.

The link between the random and systematic component is captured by the so-called *link function* $g(\cdot)$. It holds

$$\boldsymbol{\eta} = g(\boldsymbol{\mu}) = g(E[\boldsymbol{Y}]) \tag{4.2}$$

In total, for estimates $\hat{\boldsymbol{\beta}}$ of the regression coefficients, a response value can be estimated by

$$\hat{y} = g^{-1}(\hat{\beta}_0 + \sum_{j=1}^{p^*} \hat{\beta}_j x_j) \tag{4.3}$$

given explanatory values $\boldsymbol{x}$, and where g^{-1} is the inverse of the link function.

4.1.1 The exponential family

In GLMs we assume that the responses come from a distribution that belongs to the exponential family. It is a large class of distributions and includes, for example, discrete distributions as Bernoulli, Binomial and Poisson or continuous distributions as Normal, Gamma or Inverse Gaussian.

A distribution is a member of the exponential family if its probability mass function (if Y discrete) or its density function (if Y continuous) has the form

$$f(y;\theta,\phi) = exp\left(\frac{y\theta - b(\theta)}{a(\phi)} + c(y,\phi)\right) \tag{4.4}$$

for some specific functions $a(\cdot)$, $b(\cdot)$ and $c(\cdot)$. θ is called the canonical parameter, and ϕ is the dispersion parameter.

For instance, the Normal distribution can be written as

$$f(y;\theta,\phi) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2\right) = \exp\left(\frac{y\mu - \mu^2/2}{\sigma^2} - \frac{1}{2}(\frac{y^2}{\sigma^2} + \log(2\pi\sigma^2))\right)$$

so that $\theta = \mu$, $\phi = \sigma^2$, and $a(\phi) = \phi$, $b(\theta) = \theta^2/2$ and $c(y,\phi) = -\frac{1}{2}(\frac{y^2}{\phi} + \log(2\pi\phi))$.

For all members of the exponential family, it can be shown that

$$\begin{aligned} E(Y_i) &= \mu_i = b'(\theta_i) \\ Var(Y_i) &= b''(\theta_i)a(\phi) = V(\mu_i)a(\phi) \end{aligned} \tag{4.5}$$

where the variance function is defined by $V(\mu_i) = b''(\theta_i)$.

The choice of the probability distribution may be suggested by the type of the response data or knowledge of how the variance changes with the mean.
The family of exponential distributions enables the GLMs to be fitted to a wide range of data types, such as binary data, counts, proportions, and continuous data.

4.1.2 The link function

The link function relates the linear predictor $\boldsymbol{\eta}$ to the expected value $\boldsymbol{\mu}$ so that $g(\boldsymbol{\eta}) = \boldsymbol{\mu}$, where $g(\cdot)$ can be any monotonic differentiable function. The *canonical link function* is a special link function satisfying $\boldsymbol{\theta} = g(\boldsymbol{\mu})$.

Which link function should be used depends on the relationship and distribution of the dependent variable. When both $\boldsymbol{\mu}$ and $\boldsymbol{\eta}$ take any value on the real line, it might be plausible to use the identity link and simply set $\boldsymbol{\mu} = \boldsymbol{\eta}$. In contrast, if we are dealing with e.g. Gamma or Poisson distribution, we must ensure $\boldsymbol{\mu} > 0$ while $\boldsymbol{\eta}$ might be negative. A reasonable choice in this case could be the log link $\boldsymbol{\eta} = \log(\boldsymbol{\mu})$ with its inverse $\boldsymbol{\mu} = e^{\boldsymbol{\eta}}$.

The choice of the link function also influences the interpretation of the parameters. For instance, with a log link function additive effects contributing to $\boldsymbol{\eta}$ become multiplicative effects contributing to $\boldsymbol{\mu}$.

4.1.3 Parameter estimation

By assuming that the distribution of Y belongs to the exponential family, maximum likelihood estimates can be derived for the regression coefficients $\beta_0, \beta_1, \ldots, \beta_{p^*}$.

The log-likelihood is defined by $\ell(y_i, \theta_i, \phi) = \log f(y_i, \theta_i, \phi)$. Since the random variables Y_i are independent and belong to the exponential family, the sample log-likelihood of vector $\boldsymbol{y}$ is given by

$$\ell(\boldsymbol{y}, \boldsymbol{\mu}, \phi) = \sum_{i=1}^{k} \ell(y_i, \theta_i, \phi) = \sum_{i=1}^{k} \frac{y_i \theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi) \tag{4.6}$$

where θ_i is a function of $\mu_i = \boldsymbol{x_i}\boldsymbol{\beta}$.

Maximum likelihood estimates solve the *score equations*

$$U(\beta_j) = \frac{\partial \ell(\boldsymbol{y}, \boldsymbol{\mu}, \phi)}{\partial \beta_j} = 0, \quad j = 1, \ldots, p^*. \tag{4.7}$$

In matrix notation, we write $U(\boldsymbol{\beta}) = 0$. In general, simultaneously solving the score equations is not trivial, and an iterative solution has to be computed numerically.

We apply a variant of the Newton-Raphson algorithm, called *Fisher scoring algorithm*. The *Fisher information matrix* is defined as the negative expectation of the matrix of second derivatives, i.e. $\boldsymbol{I}(\boldsymbol{\beta}) = -E\left[\frac{\partial^2 \ell(\boldsymbol{y},\boldsymbol{\mu},\phi)}{\partial \boldsymbol{\beta}\partial \boldsymbol{\beta}^T}\right]$.
One iteration follows

$$\boldsymbol{\beta}^{(r+1)} = \boldsymbol{\beta}^{(r)} + \left\{\boldsymbol{I}(\boldsymbol{\beta}^{(r)})\right\}^{-1} U(\boldsymbol{\beta}^{(r)}) \tag{4.8}$$

where $\boldsymbol{\beta}^{(r)}$ is the estimate of $\boldsymbol{\beta}$ at iteration r.

Applying the chain rule and using (4.5) yields

$$U(\beta_j) = \frac{\partial \ell(\boldsymbol{y},\boldsymbol{\mu},\phi)}{\partial \beta_j} = \frac{1}{a(\phi)} \sum_{i=1}^{k} W_i (y_i - \mu_i) \frac{d\eta_i}{d\mu_i} x_{ji} \tag{4.9}$$

for $j = 1, \ldots, p^*$, when we define $W_i^{-1} = V(\mu_i)\left(\frac{d\eta_i}{d\mu_i}\right)^2$.

Further, it holds that the Fisher scoring matrix has elements

$$I_{js}(\boldsymbol{\beta}) = -E\left[\frac{\partial^2 \ell(\boldsymbol{y},\boldsymbol{\mu},\phi)}{\partial \beta_j \partial \beta_s}\right] = -\frac{1}{a(\phi)} \sum_{i=1}^{k} \frac{x_{ji} x_{si}}{V(\mu_i)(d\eta_i/d\mu_i)^2} = -\frac{1}{a(\phi)} \sum_{i=1}^{k} W_i x_{ji} x_{si}. \tag{4.10}$$

In total, we have $U(\boldsymbol{\beta}) = \frac{1}{a(\phi)} \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{M} (\boldsymbol{y} - \boldsymbol{\mu})$ and $\boldsymbol{I}(\boldsymbol{\beta}) = \frac{1}{a(\phi)} \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X}$, where $\boldsymbol{W}$ is the diagonal matrix of W_i and $\boldsymbol{M}$ is the diagonal matrix of link derivatives $\frac{d\eta_i}{d\mu_i}$.
Hence, we can write (4.8) as

$$\boldsymbol{\beta}^{(r+1)} = \boldsymbol{\beta}^{(r)} + (\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{M} (\boldsymbol{y} - \boldsymbol{\mu}) \tag{4.11}$$

where all quantities at the right hand side are evaluated at $\boldsymbol{\beta}^{(r)}$.
Note that $a(\phi)$ cancels out in (4.11), and hence ϕ does not need to be known in order to estimate $\boldsymbol{\beta}$.

It can be shown that each iteration step of the Fisher scoring algorithm equals the result of a weighted least squares regression of an adjusted dependent variable z_i on explanatory variables x_{ij}. For $\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{M}(\boldsymbol{y} - \boldsymbol{\mu})$, iteration (4.11) is equivalent to

$$\boldsymbol{\beta}^{(r+1)} = (\boldsymbol{X}^T \boldsymbol{W} \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{W} \boldsymbol{z}. \tag{4.12}$$

Therefore, the procedure is also called *iteratively re-weighted least squares* (IRLS)

algorithm because in each iteration the weight matrix is updated.

Furthermore, a positive aspect of the algorithm is that the weights and adjusted responses depend on the coefficients only via the fitted values μ_i. Since the objective of the model fitting is to produce $\hat{\mu}_i$ that are as close as possible to the original y_i, it is natural to start with the initial value $\boldsymbol{\mu}^{(0)} = \boldsymbol{y}$.

The algorithm stops when the parameter estimates do not change significantly anymore. The final estimate is denoted by $\hat{\boldsymbol{\beta}}$.

4.1.4 Statistical inference

Considering a statistical model such as the GLM allows for statistical inference, see Dunn and Smyth (2018). Here, we introduce the Wald test.
This inference method tests the null hypothesis $H_0 : \beta_j = \beta_j^0$ for any $j = 0, \dots, p^*$, where β_j^0 is some specific value. One can show that the regression coefficients $\hat{\beta}_j$ are approximately normally distributed for a sufficiently large size of the training sample. Furthermore, the standard error of the estimated parameters can be directly calculated from the inverse of the information matrix. It holds $se(\hat{\beta}_j) = \sqrt{\phi} v_j$, where v_j are the square-root diagonal element of $(X^T W X)^{-1}$.
Hence, if H_0 is true, the test statistic $Z = \frac{\hat{\beta}_j - \beta_j^0}{se(\hat{\beta}_j)}$ is approximately standard normally distributed. One should reject H_0 at significance level α if $|Z| > z_{\alpha/2}$, where $z_{\alpha/2}$ denotes the $\alpha/2$-quantile of the standard normal distribution.

When ϕ is unknown, an estimator s^2 of ϕ must be used to compute the standard errors $se(\hat{\beta}_j) = s v_j$. For details of s^2, see Dunn and Smyth (2018). The Wald statistic is then defined by $T = \frac{\hat{\beta}_j - \beta_j^0}{se(\hat{\beta}_j)}$. And now, as the estimate s^2 is used, T follows approximately a t-distribution with $k - (p^* + 1)$ degrees of freedom under H_0. The hypothesis should be rejected at significance level α if $|T| > t_{\alpha/2, k-p^*-1}$.

4.1.5 Extensions

The GLM is already a generalization of the classical linear model. Nevertheless, various extensions exist that are not considered in this thesis, but are worth mentioning.
For instance, a model could include

- Interaction terms: Sometimes two factors, or a factor and a numerical, have an *interaction effect*. For example, when gender and age are explanatory variables, but the age effect for males and females is different. Gender and age can then be combined into one variable that describes the combined effect of these variables and is called their interaction.
- Semi-parametric components: E.g. generalized additive models (GAMs) are able to handle non-parametric components in the linear predictor η by smooth functions of the explanatory variables.

4.2 Regression trees

We follow the classification and regression tree (CART) methodology introduced by Breiman et al. (1984), and that is also presented in James et al. (2013). Classification trees are used to predict categorical response, whereas regression trees deal with continuous responses. We focus on regression trees only.

The idea of tree-based methods is to divide the space of explanatory variables into non-overlapping *regions* that are homogeneous with respect to the response values.
A tree can be easily visualized in a tree chart. Starting at the *root*, an observation is passed down the tree through multiple splits, called *internal nodes*, where rule-based decisions are made until it reaches a *terminal node* or *region*.
In regression trees, the prediction for each region is a constant that equals the sample mean of the response values within the region. That is, given regions $R_1, \dots, R_H$ and an observation $\boldsymbol{x}$ of explanatory variables, the prediction is given by

$$\hat{y} = f(\boldsymbol{x}) = \sum_{m=1}^{H} \mu_m \mathbb{1}_{\boldsymbol{x} \in R_m} \tag{4.13}$$

where μ_m denotes the mean of response values in region R_m.

An exemplary regression tree is shown in Figure 4.1. For each node, the sample mean and the proportion of observations in the corresponding subtree is depicted. The tree consists of eight terminal nodes and hence the number of distinct prediction values is restricted to eight.

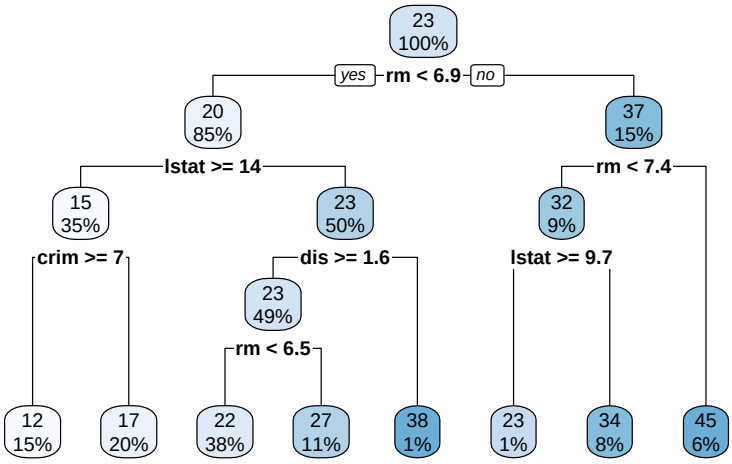


Figure 4.1: A single regression tree build on the Boston Housing data. The value of interest is the price of built houses in 1000 USD. The prediction is based on the average number of rooms per dwelling (rm), the percentage of lower status of the population ($lstat$), the per capita crime rate by town ($crim$), and the weighted distances to five Boston employment centers (dis).

Note that trees can easily handle categorical predictors without converting them into dummy variables.

4.2.1 Tree construction

The goal is to find an appropriate number H and regions $R_1, \ldots, R_H$ that minimize the sum of squared errors (SSE) when an observation in region R_m is predicted by the region's mean μ_m. The SSE is defined by

$$SSE = \sum_{m=1}^{H} \sum_{x_i \in R_m} (y_i - \mu_m)^2 \tag{4.14}$$

It is infeasible to consider every possible partition of the predictor space, and typically a *recursive binary splitting* is used to find an approximately optimal division. Binary refers to the fact that each parent node is always split into two child nodes. Recursive implies that the predictor space is split successively and partitions do not change based on later partitions in the tree.

The fitting process starts with the entire data set and selects a predictor and a cut point to partition the data into two groups such that the resulting split minimizes the sum of squared errors. At each further step, the parent node, the predictor and the split point have to be found such that this additional split leads to the greatest reduction in the *SSE*.
For continuous predictors the process of finding the optimal location of the split is straightforward since the data can be ordered in a natural way. However, when a variable has more than two categories, the search becomes more complex because there are multiple possibilities to combine the categories in two subgroups.

The splitting process is repeated until a certain stopping criteria. Typically, the minimum number of samples in a terminal node and/or the maximum tree depth are specified.

4.2.2 Pruning

A fully grown tree is often quite large and could have single observations in the terminal nodes. This could cause overfitting and poor prediction performance on new data sets. On the other hand, if the training process is stopped too early, the constructed tree might be too small and not be able to capture the existing relationship.

A widely used strategy, called *pruning*, is to grow a very large tree T_0 and then prune it back to a subtree of appropriate size. Intuitively, the optimal subtree is the one with minimum test error. Given a subtree, one can estimate its test error using cross-validation. However, estimating the cross-validation error for every possible subtree is computationally expensive.

Instead, we follow a process called *cost-complexity tuning*. The complexity parameter $\alpha > 0$ penalizes the tree size, which is defined as the number of terminal nodes. Given a large tree T_0 and fixed parameter α, an optimal subtree T_α of T_0 is found as

$$T_\alpha = \arg\min_{T \subset T_0} \left\{ \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \mu_m)^2 + \alpha |T| \right\} \tag{4.15}$$

where $|T|$ denotes the number of terminal nodes in T. For a range of parameters α, a sequence of optimal subtrees T_α can be determined via (4.15).

In the next step, we aim to find the best subtree within this sequence with respect to the test error. The K-fold cross-validation consists of the following steps:

1. Divide the data into K folds.
2. For each $k = 1, \ldots, K$ and for each α: Use all but the k-th fold to grow a large tree and apply cost-complexity tuning with parameter α. Given this tree, determine the mean squared error (MSE) on the k-th fold.
3. Average the error for each α over all K folds, yielding MSE_α.
4. Take α^* with minimum MSE_α.

Lastly, we find T_{α^*} as optimal subtree according to (4.15).

4.2.3 Ensemble methods

Although pruning is applied in the tree construction, single regression trees still suffer from high variance and appear very unstable to small changes in the data. By aggregating many regression trees, the predictive performance can be substantially improved. This approach is called *ensemble* technique. Of course, ensembles can be helpful in any other prediction method as well. However, since the technique is particularly powerful for tree models, we introduce it in this section.

Bootstrap aggregation (bagging), proposed by Breiman (1996), is one method to create ensembles. The basic idea is that averaging multiple predictions that are based on independent data sets reduces the variance. In practice, we generally do not have multiple separate data samples and instead we need to rely on bootstrapping. A bootstrap sample is drawn by randomly taking observations from the data with replacement until the sample has the same size as the original data set. With replacement means that the same observation can occur multiple times in the bootstrap data set.
Therefore, the first step is to generate a number, say B, of bootstrap samples independently from the data set. For each bootstrap sample $b = 1, \ldots, B$, we then grow a single, unpruned tree yielding the prediction $f^b(\boldsymbol{x})$.
The overall prediction value is given by the average of the individual predictions:

$$\hat{y} = f_{bag}(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} f^b(\boldsymbol{x}) \tag{4.16}$$

Note that each (unpruned) tree has high variance, but low bias. Averaging these B trees reduces the overall prediction variance.

One drawback of bagging is that we cannot fully exploit variance reduction since the single trees are not completely independent. As all predictors are considered at any time in all trees, the grown trees may have very similar structures due to the underlying relationship. This is called *tree correlation*. We now want to *de-correlate* the trees by adding randomness to the construction process.

Breiman (2001) suggested an algorithm called *random forests*. It works very similar to bagging and we also make use of bootstrap samples. But when the trees are built, only a limited number $m \leq p$ of explanatory variables are considered at random as candidates to find the best split. Each time a split is constructed, another subset of predictors is selected randomly. Again, the trees are left unpruned and the individual predictions are averaged equivalently to (4.16) to give the final prediction.
In summary, random forests de-correlate trees by forcing each split to consider only a subset of the explanatory variables. Clearly, if a random forest is built based on $m = p$ predictors in each steps, the method simply corresponds to bagging.

There exist also other ensemble methods that are not covered in this thesis. For instance:

- Boosting: Boosting is similar to bagging and differs only in the way the data is resampled. In bagging all observations have the same probability of being selected into the next bootstrap sample. Whereas in boosting, observations that have been frequently misclassified in previous trees have a higher probability to be drawn.
- Stochastic Gradient Boosting: Many small regression trees are built *sequentially* based on the residuals from the previous tree. In each step, a shrunk version of the prediction term from the current tree is added to the model.

4.3 Neural networks

Artificial neural networks, or simply neural networks, are a non-linear regression technique inspired by models of the human brain. Here, we present a *single-layer feed-forward network* and follow Kuhn and Johnson (2013).

The architecture of a neural network can be visualized by a diagram as depicted in Figure 4.2. It contains nodes, so-called *neurons*, and one-way connections between them. The input layer consists of all explanatory variables, where the categorical ones have been replaced by dummy variables. The elements of the hidden layer are referred to as *hidden units* since they are not directly observable from the data sample. H denotes the number of hidden units. The output neuron produces the value of interest.

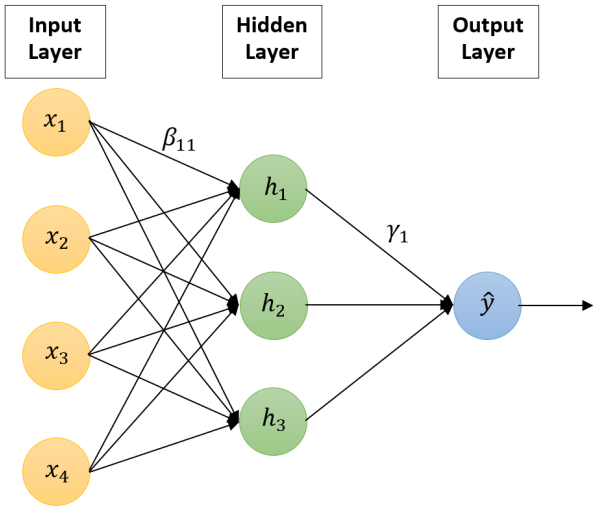


Figure 4.2: A single-layer neural net with three hidden units

Each hidden unit receives a linear combination of the predictors transformed by a non-linear function $g(\cdot)$, which is called *activation function*. Usually, the logistic function $g(u) = \frac{1}{1+e^{-u}}$ is used.
Given an input $\boldsymbol{x}$ with p^* explanatory variables $x_1, \ldots, x_{p^*}$, the value of the m-th hidden unit can be expressed by

$$h_m(\boldsymbol{x}) = g\left(\beta_{m0} + \sum_{j=1}^{p^*} x_j \beta_{mj}\right), \quad m = 1, \ldots, H \tag{4.17}$$

The outcome of the network is then calculated as linear combination of all hidden units:

$$f(\boldsymbol{x}) = \gamma_0 + \sum_{m=1}^{H} \gamma_m h_m(x) \tag{4.18}$$

In total, we obtain the predicted value by

$$\hat{y} = f(\boldsymbol{x}) = \gamma_0 + \sum_{m=1}^{H} \gamma_m \, g\left(\beta_{0m} + \sum_{j=1}^{p^*} x_j \beta_{mj}\right) \tag{4.19}$$

4.3.1 Training process

In Equation (4.19), we need to estimate $H(p^*+1)+H+1$ unknown parameters. Often the parameters are also called *weights* since they form a weighted average of values from the previous layer. The intercepts $\gamma_0, \beta_{10}, \ldots, \beta_{H0}$ can be interpreted as bias to the neurons.

As measure of fit, the sum of squared errors (SSE) is chosen to be minimized. It is given by

$$SSE = \sum_{i=1}^{k} (y_i - f(\boldsymbol{x_i}))^2 \tag{4.20}$$

Since there are no constraints on the parameters, optimization is challenging. Usually, the parameters are initialized randomly around zero and then a numerical algorithm is applied to iteratively update the weights.

We present the *back-propagation* algorithm that uses gradient descent for minimization (see Hastie et al. (2009)). The name comes from the fact that the gradient is derived by traversing the network forwards and backwards.
The weights are updated in the opposite direction of the gradients and estimates of iteration $r+1$ are given by

$$\begin{aligned} \gamma_m^{(r+1)} &= \gamma_m^{(r)} - \eta \frac{\partial SSE}{\partial \gamma_m^{(r)}} \\ \beta_{mj}^{(r+1)} &= \beta_{mj}^{(r)} - \eta \frac{\partial SSE}{\partial \beta_{mj}^{(r)}} \end{aligned} \tag{4.21}$$

where η is the learning rate. Therefore, we need to compute the partial derivative

of SSE with respect to each weight in the network.

First, we write $SSE = \sum_{i=1}^{k} SSE_i$ where SSE_i is the error arising from observation i. Clearly, the partial derivative of SSE equals the sum of partial derivatives of SSE_i. We exploit the structure of the neural network to obtain alternative expressions of these partial derivatives.

Applying the chain rule, yields

$$\begin{aligned} \frac{\partial SSE_i}{\partial \gamma_m} &= -2\,(y_i - f(\boldsymbol{x_i}))\, h_m(\boldsymbol{x_i}) \\ \frac{\partial SSE_i}{\partial \beta_{mj}} &= -2\,(y_i - f(\boldsymbol{x_i}))\, \gamma_m\, g'\left(\beta_{0m} + \sum_{j=1}^{p^*} x_{ij}\beta_{mj}\right) x_{ij} \end{aligned} \tag{4.22}$$

for $m = 1,\ldots,M$ and $j = 1,\ldots,p^*$. When we set $h_0(\boldsymbol{x_i}) = 1$ and $\boldsymbol{x}_{i0} = 0$, the above equations also hold for $m = 0$ and $j = 0$.

Now, we define

$$\begin{aligned} \delta_i &= -2\,(y_i - f(\boldsymbol{x_i})) \\ \varepsilon_{mi} &= -2\,(y_i - f(\boldsymbol{x_i}))\, \gamma_m\, g'\left(\beta_{0m} + \sum_{j=1}^{p^*} x_{ij}\beta_{mj}\right) \end{aligned} \tag{4.23}$$

and hence we can write (4.22) as $\frac{\partial SSE_i}{\partial \gamma_m} = \delta_i h_m(\boldsymbol{x_i})$ and $\frac{\partial SSE_i}{\partial \beta_{mj}} = \varepsilon_{mi} x_{ij}$.

The quantities δ_i and ε_{mi} can be interpreted as errors for the current model and observation i at the output unit and the hidden units, respectively.
Plugging δ_i into the definition of ε_{mi}, results in

$$\varepsilon_{mi} = \delta_i \gamma_m g'(\beta_{0m} + \sum_{j=1}^{p^*} x_j \beta_{mj}) \tag{4.24}$$

This is often called *back-propagation equations* as it shows how the errors can be computed in backwards direction. Hence, we first compute $\delta_i = -2(y_i - f(\boldsymbol{x_i}))$ and then obtain ε_{mi} via Equation (4.24).

Overall, following (4.21), we can update the weights according to

$$\begin{aligned} \gamma_m^{(r+1)} &= \gamma_m^{(r)} - \eta \sum_{i=1}^{k} \delta_i^{(r)} h_m(\boldsymbol{x_i})^{(r)} \\ \beta_{mj}^{(r+1)} &= \beta_{mj}^{(r)} - \eta \sum_{i=1}^{k} \varepsilon_{mi}^{(r)} x_{ij} \end{aligned} \tag{4.25}$$

In summary, the process to update the weights consists of two main steps: In the forward process, the predicted outputs are calculated given the current weights. In the backward pass, we compute the partial derivatives in terms of the values δ_i and ε_{mi} and update the weights accordingly.
The algorithm stops if the fitting criterion does not change significantly anymore.

One issue of this optimization is that due to the non-convexity of the error function, it cannot be ensured that a global optimum is found. That is, for different initial parameters the training could lead to different parameter estimates and might result in a local optimum. To obtain a more stable model, one could initialize the weights with multiple distinct starting values and pick the best fit according to an error function or use all resulting networks and take the average output as prediction value.

4.3.2 Weight decay

Since we have a lot of parameters, the model is extremely flexible but at the same time it also easily leads to overfitting.
One way to regularize the model is the method of *weight decay*. Here, a penalty for large regression coefficients is added to the error function. This can be written as

$$SSE + \lambda \left(\sum_{m=0}^{H} \gamma_k^2 + \sum_{m=1}^{H} \sum_{j=0}^{p^*} \beta_{mj}^2 \right) \tag{4.26}$$

for a given value of weight decay λ. Larger λ will shrink to weights towards 0.

To ensure that all inputs are treated equally in the regulation process, it makes sense to scale all inputs to the same range $[0, 1]$.

5 Numerical results

In the last part, we apply the machine learning techniques from Chapter 4 to a large portfolio of variable annuity contracts in order to speed up the determination of the today's value and of the solvency capital requirement.

First, we need to generate the portfolio of VA contracts. Then we predict the today's value of the guaranteed benefits applying machine learning methods. As described in Chapter 3, the main task in calculating the SCR consists of generating the empirical distribution of the fair values at time 1. Therefore, we dedicate an extra section to the estimation of V_1, and finally the SCR is estimated.

All codes are written in the programming language R.

5.1 Synthetic portfolio and parameter setting

Unfortunately, we have no real data records of variable annuity contracts, and hence we need to work with a synthetic dataset. The portfolio is drawn uniformly by randomly sampling values for each attribute from the ranges specified in Table 5.1. We generate $n = 10000$ contracts.

Although the framework in Chapter 2 allows for the evaluation of any combination of guarantees, we restrict the synthetic portfolio to the possibilities indicated in Table 5.1. If we have a death guarantee in addition to one of the guarantees GMWB, GMAB or GMIB, denoted by GMWBwD, GMABwD or GMIBwD, we suppose that the guaranteed death benefit is specified as return of premium.
The other attributes correspond to the variables as defined in Section 2.2.2. Note that the premium P always refers to the up-front investment of the VA contract and hence specifies the initial account value. We do not expect that the premium equals the today's fair value of the contract.

Attribute	**Values**
Guarantee type	GMDB, GMWB, GMWBwD, GMAB, GMABwD, GMIB, GMIBwD
Product type	returnP, rollup, ratchet
Withdrawal rate x_w	$0.04, 0.05, 0.06, 0.06, 0.08$
Income ratio $g \cdot \ddot{a}_T$	$0.75, 0.8, 0.85, 0.9, 0.95$
Roll-up rate i	$0.01, 0.02, 0.03, 0.04, 0.05$
Premium P	$[10000, 500000]$
Maturity T	$\mathbb{N} \cap [10, 25]$
Age x_0	$\mathbb{N} \cap [20, 60]$
Gender	Male, Female

Table 5.1: Possible values for generation of the synthetic portfolio

The age x_0 of the policyholder at contract inception and the gender are necessary to identify the associated best estimate mortality rates. We follow the mortality tables of the German society of actuaries (DAV 2004 R).

We exclude all guarantee fees and set $\varphi = 0$. Clearly, this is not a realistic assumption. However, since we are looking at a portfolio with various products, it is not possible to determine fair guarantee fees for all these contracts. In practice, the fee should be listed as additional attribute.

Further, it should be emphasized that all contracts in the portfolio have inception date $t = 0$ and hence are concluded at current time point. We do not take into account any old policies or future business.

To get an insight into the generated portfolio, a few contracts are printed here:

```
      guarantee  p.type maturity gender age   premium w.rate i.rate
9995       GMAB  rollup       25 Female  29 364023.28     NA     NA
9996       GMDB  ratchet       16   Male  44  16182.02     NA     NA
9997     GMWBwD    <NA>       25   Male  43 240424.51   0.05     NA
9998       GMAB returnP       18   Male  49 116690.42     NA     NA
9999     GMIBwD  ratchet       10   Male  49 232991.91     NA    0.8
10000    GMABwD  ratchet       20 Female  59 335231.74     NA     NA
      rollup.rate
9995         0.04
9996           NA
9997           NA
```

```
9998            NA
9999            NA
10000           NA
```

To obtain risk-neutral paths of the stock evolution for the MC simulation of V_0, the scenario generator in Section 2.3.1 is applied with risk-free rate $r = 0.03$ and volatility $\sigma = 0.15$. We consider $J = 1000$ risk-neutral paths in the simulation. Furthermore, for the nested simulation approach in the case of SCR estimation, we also need realizations of the real-world development during the first year. Here, we use the algorithm in 3.2.2 with drift $\mu = 0.07$ and volatility $\sigma = 0.15$ to generate $N = 1000$ real-world scenarios.

5.2 Today's fair value

In this section, we aim to calculate the value V_0. We start with the results of the traditional MC simulation. In addition, we want to use machine learning methods to determine the fair value more efficiently.
Remember that the approach consists of the following basic steps:

1. Choose a set of contracts as representatives.
2. Evaluate the representatives via MC simulation.
3. Fit a prediction model based on the representatives and their calculated fair values.
4. Estimate the fair values of the remaining contracts via the prediction model.

The section proceeds with a suggestion how to select the representatives. Then, after a small adjustment of the data, the machine learning algorithms from Chapter 4 are applied. At first, we consider the results of each method individually and then conclude the section with a comparison of the accuracy and risk drivers for V_0 of the models.

5.2.1 Results from traditional MC simulation

The Monte Carlo estimation of the fair value at time 0 according to Section 2.2 with $n = 10000$ contracts and $J = 1000$ risk-neutral stock paths took `525.13` seconds, i.e. roughly 8 minutes, using my computer and two cores in parallel.

A histogram of the calculated V_0 is depicted in Figure 5.1 and a summary of the values is given by

```
Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
2.4   3515.3  12322.2  24540.5  28450.7 326703.3
```

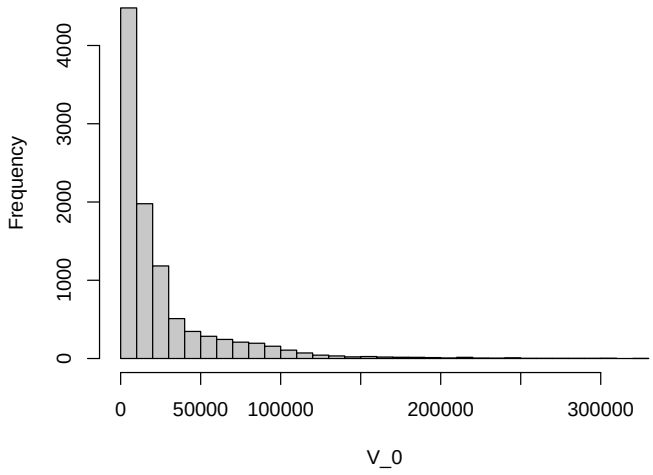


Figure 5.1: Histogram of today's fair value calculated via traditional MC.

We conclude that our value of interest is positive and right-skewed.

Further, to get a first impression of the relation between the fair value and all explanatory variables, the univariate impact of each variable on the today's value is plotted in Figure 5.2.

From the univariate perspective, we observe the following dependencies within our synthetic portfolio:

- The `guarantee` clearly has a great influence on the value as it determines the type of contract. The highest guaranteed benefits are observed for

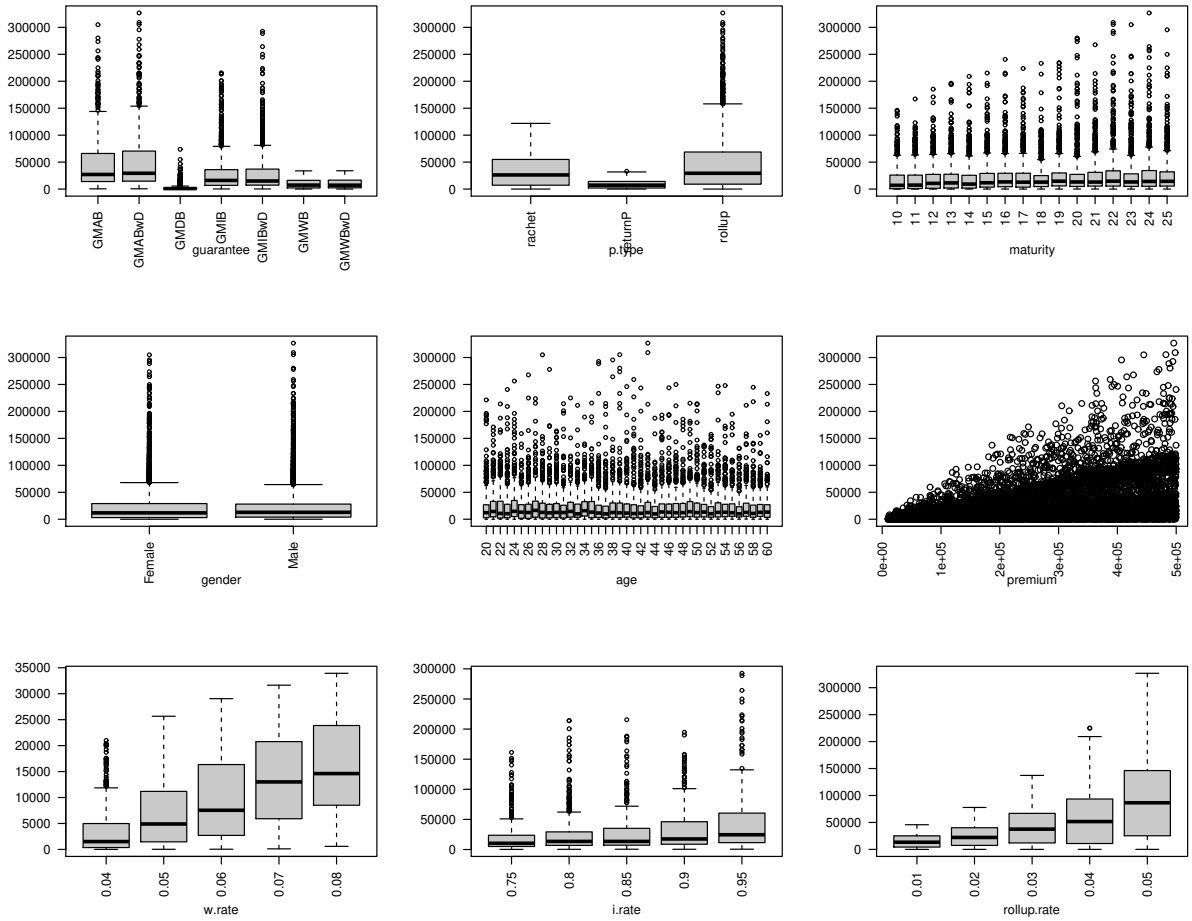


Figure 5.2: Univariate impact of explanatory variables on the fair value V_0.

the GMAB options followed by contracts including a GMIB. Remember that the guaranteed accumulation and income benefits are defined by $L_T^A = \max\{G_T^{A+} - A_T^+, 0\}$ and $L_T^I = \max\{G_T^{I+} g\,\ddot{a}_{x_T} - A_T^+, 0\}$, respectively. Since the ratio $g\,\ddot{a}_{x_T}$ is smaller than 1 within our portfolio, it is reasonable that a GMIB yields smaller V_0. The GMWB provides the lowest fair values. This is plausible since you profit from the guarantee only, if the fund performs poorly and if the maturity is long enough to withdrawal (nearly) the entire benefit base within the annual limits. Also, for the GMDB we observe relatively low V_0 because it is rather unlikely to die during the contract. As expected, a death benefit on top of other guarantees increases the fair value slightly.

- The attribute `p.type` specifies the evolution of the guaranteed benefit base. Compounding the benefit base every year by the roll-up rate yields the highest values. Of course, the benefit of premium return is smaller because returning the initial investment corresponds to compounding it with rate 0. In the ratchet type, the maximum of annual account values is taken as benefit base and this clearly equals at least the initial investment.
- For different `maturity` values, the majority of the V_0 are at the same height. However, we see that with longer durations the maximum fair values become higher. This is presumably due to contracts inclduing a GMAB or GMIB with a roll-up benefit base, where a long maturity implies that the initial investment is compounded many times. It is also expected that in the case of a GMWB, the V_0 generally increases with maturity because longer terms allow for a larger number of withdrawals and hence a possibly higher value. However, the GMWB fair values are so small that the influence cannot be seen here.
- The attributes `gender` and `age` seem to have hardly any influence on the fair value since they only affect the death probabilities. Most contracts have a retirement age of at most 70 years, and the best estimates do not vary much by gender and age within the interval $[20, 70]$.
- Furthermore, it seems that there is a strong positive correlation between V_0 and the `premium`. It is absolutely plausible that the larger the investment, the higher the corresponding guarantees.
- Lastly, increasing `w.rate, i.rate` and `rollup.rate` yields an increasing fair value since the rates specify the evolution of the benefit bases directly. Particularly the roll-up rate, which annually compounds the benefit base, has a large impact.

5.2.2 Data preparation and sampling

The printed contracts in Section 5.1 have `NA` entries. In particular, NA occurs within the columns `p.type, w.rate, i.rate` and `rollup.rate` due to distinct contract specifications for different guarantees. For example, in contracts without a GMWB no withdrawal rates are specified, and in general every contract has at

least one NA entry at some point. Since machine learning methods cannot work with data that contains NA values in each row, the portfolio must be adjusted in a way to get rid of these values.

One possibility would be to split the data according to the basic four types of guarantees. For each of the datasets we then could fit a machine learning method and combine the resulting models for prediction. However, the goal of this thesis is to apply a single prediction model for the entire data set. Therefore, we now have to prepare the data accordingly.
To deal with the unspecified entries in the column `p.type`, we combine the columns `guarantee` and `p.type` into one column named `product`.

```
> levels(df$product)
```

```
 [1] "GMAB ratchet"    "GMAB returnP"   "GMAB rollup"    "GMABwD ratchet"
 [5] "GMABwD returnP" "GMABwD rollup"  "GMDB ratchet"    "GMDB returnP"
 [9] "GMDB rollup"    "GMIB ratchet"    "GMIB returnP"   "GMIB rollup"
[13] "GMIBwD ratchet"  "GMIBwD returnP" "GMIBwD rollup"  "GMWBwD"
[17] "GMWB"
```

For the rates `w.rate, i.rate` and `rollup.rate`, we replace `NA` entries by `0`. This is the most reasonable choice when we need to define a specific number.

In the next step, we need to select a subset `Z` of the portfolio as representative contracts that are fed into the machine learning methods. Since we use a uniformly generated portfolio, it is sufficient to randomly select the representatives in this setting. In practice, given a heterogeneous portfolio, methods like clustering and Latin Hypercube sampling are recommended in order to span the predictor space most appropriately.
An important question is how to choose the number of representatives. Clearly, this depends on the prediction model and the data. Loeppky et al. (2009) suggests to start with $k = 10p^*$ as a rule of thumb, where p^* is the number of explanatory variables counting dummies. For our dataset, we have $p^* = 23$ and hence $k = 230$ might be an appropriate choice.
In order to investigate the model performance, we will compare predicted values with their MC estimates. Here, all contracts that are not used for the model fitting serve as `test` set.

```
> k <-  230
> ind <- sample(nrow(df), k)
```

```
> Z <- df[ind, ]
> test <- df[-ind,]
```

Now, different machine learning methods can be applied to explain the variation of V_0 by the explanatory variables, which specify each contract. In the remaining part of the section, we always aim to model the behavior of the fair value based on the training set `Z`. The resulting predictions should be as accurate as possible.

5.2.3 Generalized linear models

We start with two GLMs to predict the today's fair value. Since the response values are positive and continuous, the Gamma and the Gaussian distributions are reasonable choices. To ensure that positive values are obtained in the prediction, a log link function is applied in both cases.
In R, the models are fit on the set of representatives `Z` by

```
> glm.1 <- glm(v0 ~ ., data=Z, family=Gamma(link = "log"))
> glm.2 <- glm(v0 ~ ., data=Z, family=gaussian(link = "log"))
```

where the term `v0 ~ .` indicates that all explanatory variables are included in the model.
R automatically converts all categorical variables into dummies, and for each factor the alphabetically first level is set as reference level. The predictor `product` uses the baseline `GMAB ratchet`, while `Female` serves as reference for `gender`. For the baselines, the height of V_0 is captured by the intercept β_0.

Exemplarily, the summary of the estimated coefficients from `glm.2` is given by:

```
                        Estimate Std. Error t value Pr(>|t|)
(Intercept)             9.097e+00  1.054e-01  86.302  < 2e-16 ***
productGMAB returnP    -1.337e+00  1.244e-01 -10.743  < 2e-16 ***
productGMAB rollup     -1.128e+00  8.953e-02 -12.601  < 2e-16 ***
productGMABwD ratchet   9.288e-02  6.165e-02   1.506    0.133
productGMABwD returnP -1.401e+00  2.071e-01  -6.764 1.36e-10 ***
productGMABwD rollup  -1.109e+00  7.323e-02 -15.149  < 2e-16 ***
productGMDB ratchet    -3.389e+00  6.467e-01  -5.241 3.95e-07 ***
productGMDB returnP    -4.654e+00  4.533e+00  -1.027    0.306
productGMDB rollup     -4.230e+00  5.695e-01  -7.426 2.90e-12 ***
productGMIB ratchet    -3.102e+00  3.209e-01  -9.667  < 2e-16 ***
```

```
productGMIB returnP   -4.170e+00  4.463e-01  -9.345  < 2e-16 ***
productGMIB rollup    -3.953e+00  3.305e-01 -11.958  < 2e-16 ***
productGMIBwD ratchet  -3.079e+00  3.390e-01  -9.082  < 2e-16 ***
productGMIBwD returnP -4.494e+00  3.721e-01 -12.076  < 2e-16 ***
productGMIBwD rollup  -3.935e+00  3.481e-01 -11.304  < 2e-16 ***
productGMWB           -3.567e+00  3.726e-01  -9.571  < 2e-16 ***
productGMWBwD         -3.484e+00  3.880e-01  -8.979  < 2e-16 ***
maturity               3.924e-02  3.468e-03  11.316  < 2e-16 ***
genderMale             5.915e-02  2.724e-02   2.172    0.031 *
age                   -1.298e-03  1.268e-03  -1.023    0.307
premium                3.903e-06  1.232e-07  31.667  < 2e-16 ***
w.rate                 2.922e+01  5.176e+00   5.646 5.41e-08 ***
i.rate                 2.857e+00  3.713e-01   7.695 5.78e-13 ***
rollup.rate            3.929e+01  1.832e+00  21.449  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

In the column 'Estimate' the maximum likelikhood estimates are printed. Additionally, the summary also shows the 't value', which is the Wald statistic to test the null hypothesis $H_0 : \beta_j = 0$ for each coefficient. Large absolute t-values indiciate statistical significance. Given the Gaussian `glm.2`, the evidence suggests that e.g. the `premium` parameter is non-zero whereas no evidence exists to reject $\beta_{age} = 0$. Similarly to the univariate impact, we conclude that the variables `age` and `gender` do not have a great influence on the fair value at time 0.

Since we keep the data in original scale, it is not possible to directly see how heavily a predictor influences the expected fair value compared to others. However, one can generally say that a positive coefficient increases the predicted value with increasing explanatory value. Except from age, all numerical attributes have an increasing effect. This corresponds to the results in the univariate analysis.
Moreover, the structure of a GLM allows to assess the relative impact of one explanatory variable when all other variables remain unchanged. Recall, that according to (4.3), given the explanatory values $x_1, \ldots, x_{p^*}$, the response is predicted by $\hat{y} = \exp(\hat{\beta}_0 + \sum_{j=1}^{p^*} \hat{\beta}_j x_j)$.
Therefore, $\exp(\hat{\beta}_j)$ gives the relative change in y when x_j is increased by 1, i.e. replaced by $x_j + 1$, for $j = 1, \ldots, p^*$. More generally, $\exp(h \cdot \hat{\beta}_j)$ denotes the relative change in y when x_j is increased by $h \in \mathbb{R}$.
This allows us to interprete the impact of numerical variables. We have:

```
> exp(glm.2$coefficients['premium'] * 10000)

1.039798

> exp(glm.2$coefficients['rollup.rate'] * 0.01)

1.481235

> exp(glm.2$coefficients['maturity'] * 1)

1.04002226
```

Hence, when we look at two policies, where one has a `premium` that is 10000 higher than the other one and which are otherwise identical, the today's value for the policy with higher initial investment is expected to be 4% larger. For instance, let us consider two identical policies but one has a initial investment of 10000 and the other contract invests a premium of 20000. Intuitively, we would expect to double the payoff when doubling the investment for exactly the same contract. Since we assume a log link between the linear predictor and the expected response, the model fails to represent the true relation and expects to increase the benefits only by 4% instead of 100%. So we see that the structure of the GLM offers helpful and direct interpretations, but the assumptions also limit the model very much.
Further, for two identical contracts which only differ in the `rollup.rate` by 0.01, the contract with higher roll-up rate is expected to have a 48% higher value. This suggests a huge importance of the predictor.
When one increases the maturity of any contract by one year, the predicted fair value is expected to increase by 4%

Similarly, we can interpret the relative impact of categorical variables with respect to their baseline level. Remember that the levels were encoded by 0-1 entries, where 1 indicates the occurrence of the level. Hence, for factors, $\exp(\hat{\beta}_j)$ reflects the relative change in y when level x_j occurs compared to the same contract but with the baseline level.
In Figure 5.3, we see the relative impact of the product levels with their 95% confidence interval with respect to the baseline `GMAB ratchet`.
It indicates, that the baseline and `GMABwD ratchet` have a similar impact, whereas `GMAB returnP`, `GMABwD returnP`, `GMAB ratchet` and `GMABwD ratchet` are expected to have a significantly lower fair value of medium size. All remaining levels

lead to a relatively low fair value when all other predictors remain the same. This is not intuitive and an explanation for this behavior will be given in Section 5.2.6.

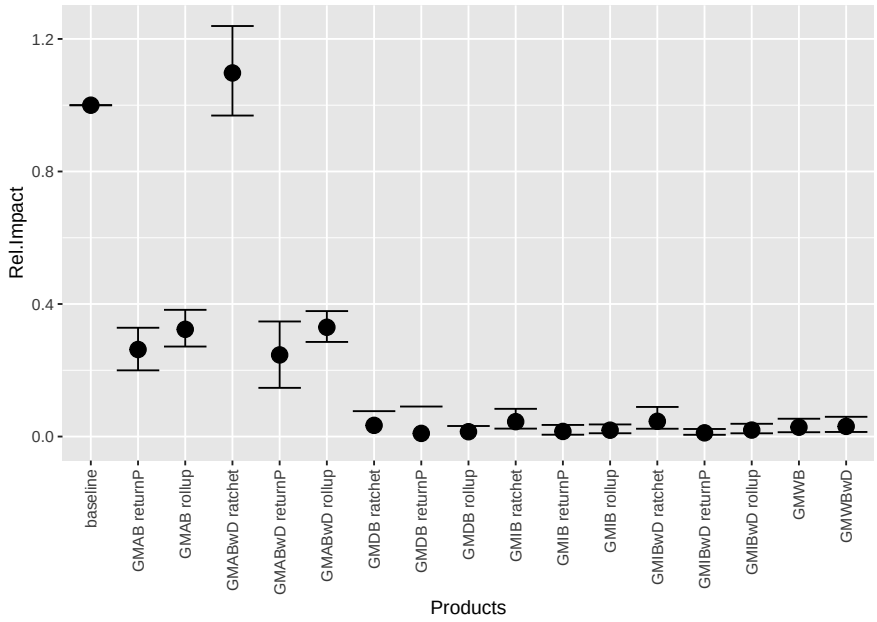


Figure 5.3: Relative impact of product levels in the Gaussian model `glm.2` with baseline `GMAB ratchet`.

To predict response values for the test data set, we call

```
> pred.1 <- predict(glm.1, test, type = "response")
> pred.2 <- predict(glm.2, test, type = "response")
```

In Figure 5.4 these predicted fair values are plotted against their Monte Carlo estimates. You can see a scatter plot (left) and a QQ plot (right). The line represents a coincidence of the predicted values with the MC values. At the top, we see that the Gamma model substantially overestimates the today's fair value, particularly in the higher range of the value. In contrast, assuming a Gaussian distribution provides a better fit.

The poor prediction quality of the Gamma model could be caused by a violation of the assumption that the response data follows a Gamma distribution.
To check this, one can examine the quantile residuals of the model (for details, see Dunn and Smyth (2018)). If the correct distribution of the exponential family is chosen, the quantile residuals should be normally distributed. In Figure 5.5 the quantile residuals of both models are plotted against the theoretical quantiles of

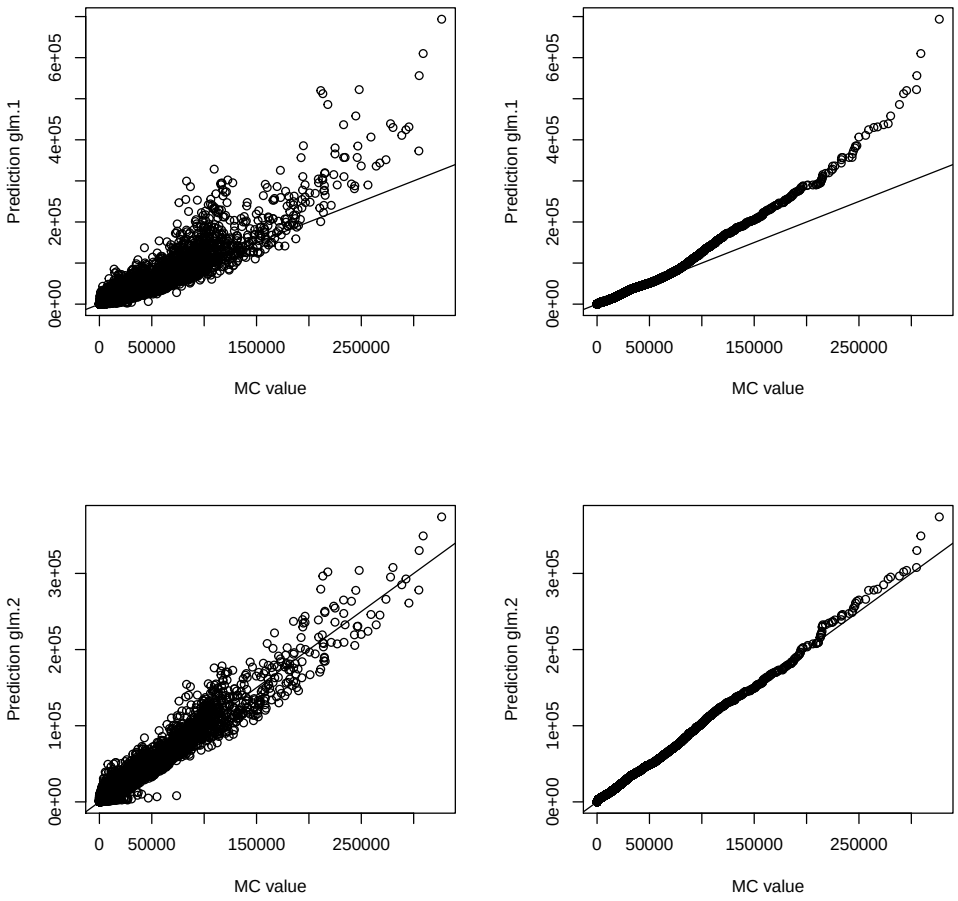


Figure 5.4: Prediction accuracy on test data for (top) Gamma `glm.1` and (bottom) Gaussian `glm.2`.

the normal distribution. The residuals of the Gamma model `glm.1` show a higher deviation from the diagonal than the Gaussian model `glm.2`.

All together, one can say that the second model, which assumes a Gaussian distribution with a log link, seems to be the better choice in this context.

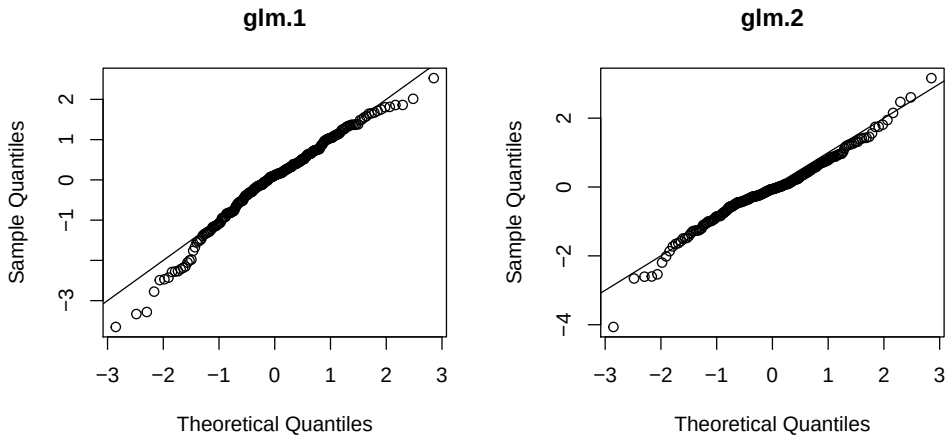


Figure 5.5: Quantile residuals from models `glm.1` and `glm.2` vs. theoretical normal quantiles.

5.2.4 Regression trees

Single trees

To predict V_0 with a single regression tree, we use the package `rpart` in R. A model with default setting is fit by

```
> single.tree <- rpart(v0 ~ ., Z, method = "anova")
```

The resulting tree is visualized in Figure 5.6. The top split is made by `product`, and in intermediate splits `product` and `premium` occur multiple time. This indicates that these two variables have a large impact on the response value and therefore are often referred to as *strong predictors*. Additionally, one split is defined by the `w.rate`.
For all numeric variables, the sample mean of the response is always larger on the right-hand side of the split, i.e. in the regions above the cut point. That is, the higher e.g. the premium the higher the predicted fair value.
In summary, based on a small set of representatives, the highest response values in the model are obtained by the products `GMAB ratchet`, `GMABwD ratchet`, `GMAB rollup`, `GMABwD rollup`, `GMIBwD ratchet` and `GMIB rollup` with a large `premium` and a high `rollup.rate`. Note that the last split `rollup.rate >= 0.035` particularly excludes contracts with zero roll-up rate, i.e. the ratchet type products.

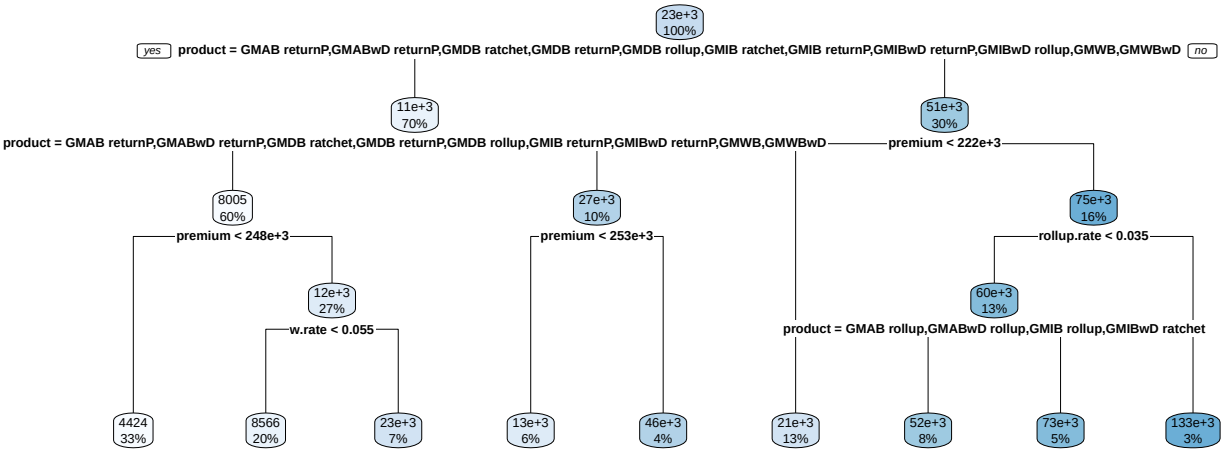


Figure 5.6: Tree chart of model `single.tree`.

As explained in Section 4.2.2, the single tree is obtained by pruning a large tree. The package `rpart` automatically performs cost-complexity tuning for a range of parameters α, which penalize the SSE for the number of terminal nodes, see (4.15). 10-fold cross-validation is used to compare the error for each α-value.
For the model `single.tree` the results are illustrated in Figure 5.7. The y-axis depicts the relative cross-validation error averaged over the 10 folds. The lower x-axis is the cost-complexity parameter α, while the upper x-axis shows the number of terminal nodes $|T_\alpha|$ of the optimal subtree obtained from pruning with the corresponding α-value. Intuitively, large values of α result in less complex models and hence smaller trees T_α, and vice versa for smaller values of α. You can also see that the larger the tree, the smaller the test error. This is reasonable because the tree model tries to reflect complex dependencies of variable annuity contracts. The result of the tuning is an optimal tree size of $|T| = 9$.
Additionally, there is a dashed line that passes above the point $|T| = 5$. This represents the so-called *1-SE rule* from Breiman et al. (1984), which suggests that instead one could choose the smallest tree within 1 standard error (SE) of the minimum cross-validation error.

Fair values for the test data are predicted by

```
> pred.st <- predict(single.tree, test)
```

For each data point, the algorithm finds the corresponding terminal node in the

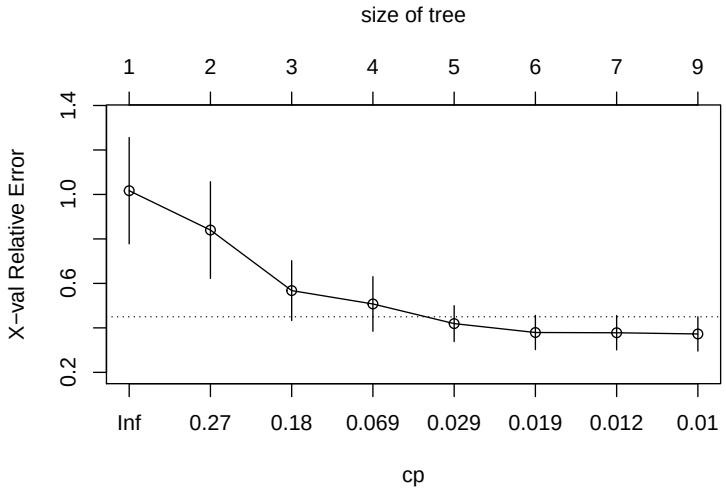


Figure 5.7: Cross-validation results of cost-complexity tuning.

tree model and takes the constant value of this region as prediction. Clearly, as we only have nine terminal nodes, there are only nine distinct possible response values, and consequently the predictive performance is very poor. The results are shown in Figure 5.8.

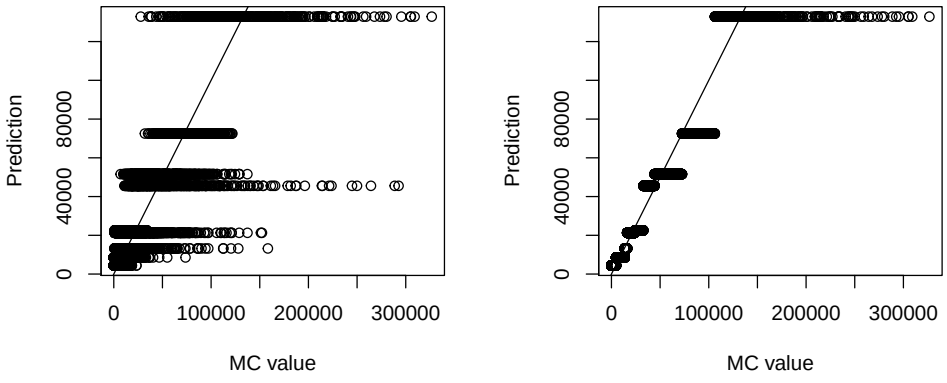


Figure 5.8: Prediction accuracy on test data via model `single.tree`.

In the default model `single.tree`, the control parameter `minsplit`, which specifies the minimum number of observations that must exist in a node for a split, is

set to 20. As we only have $k = 230$ observations in the training set, it is reasonable to reduce `minsplit` to e.g. 5. However, given the same set of representatives, calling `rpart` with `minsplit = 5` and leaving everything else unchanged, results in exactly the same pruned tree.

Bagging

In this section, we use the package `ipred` to work with bagged trees. A model is obtained by

```
> bag.tree <- bagging(v0 ~., data = Z, control = list(minsplit=2, cp=0))
```

To fit each tree in the ensemble, `bagging` calls the function `rpart`. To ensure that the trees are unpruned, we have to set `control = list(minsplit=2, cp=0)`. Here, by default `nbagg = 25` trees are aggregated.

By averaging the predictions of several trees, we aim to get a more stable prediction, but at the same time we lose the ability to interpret the results. It is no longer possible to visualize the bagged model like the single tree before.

Response values are predicted by

```
> pred.bag <- predict(bag.tree, test)
```

and the result is depicted in Figure 5.9.

Large MC values are extremely underestimated by the bagged tree model. The maximum fair value of the chosen representatives equals `200932.39`. By construction of the model, the maximum possible predicted response is `200932.39` as well. Hence, tree models are not able to perform extrapolation to obtain larger values. In addition, the property that the fair values are positively skewed means that it is very unlikely that extremely large MC values occur within only a small subset of the portfolio. This explains the poor prediction accuracy.
Note that an alternative sampling method is not necessarily helpful because the selection is based on the explanatory variables and at this time the MC values are not yet known.

Now, we also want to investigate the effect of the number of bagged trees on the model quality. To measure the performance for different values of `nbagg`, we

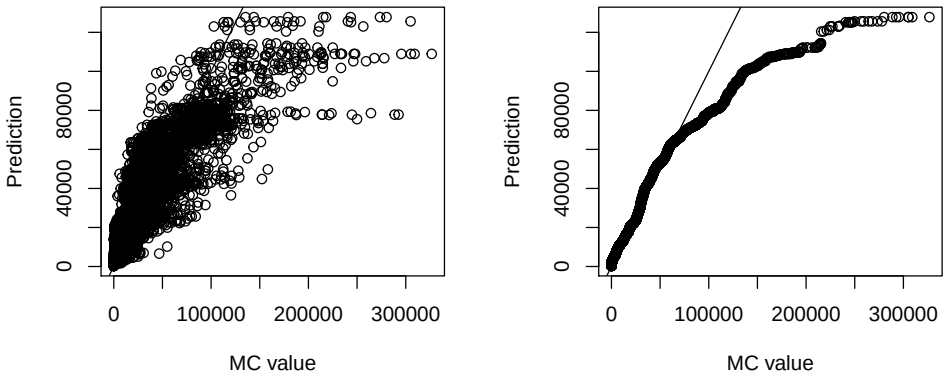


Figure 5.9: Prediction accuracy on test data via model `bag.tree`.

determine the root mean squared error (RMSE) of the fitted and observed values from the test set. The RMSE is defined by

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$$

and can be calculated by the function `rmse()`, which is provided in the package `caret`. Increasing the number of bagged trees, see Figure 5.10, makes the prediction slighlty more stable. A suitable choice might be `nbagg = 200`. Note that a large number of trees does not lead to overfitting, it only increases the computational time needed for model fitting.

Random forests

We use the package `randomForest` to model V_0 via random forests. A default model is fit by

```
> rf <- randomForest(v0 ~., Z, nodesize = 1)
```

where `nodesize` specifies the minimum size of a terminal node, and setting `nodesize = 1` ensures that unpruned trees are obtained in the ensemble.

The predicted values

```
> pred.rf <- predict(rf, test)
```

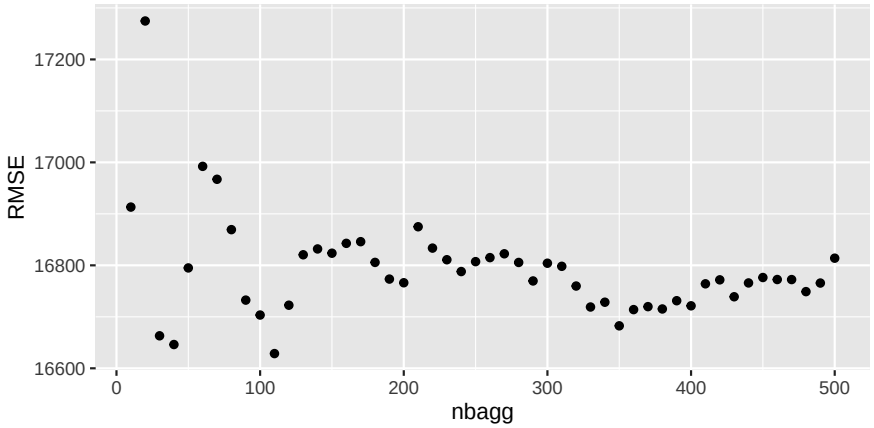


Figure 5.10: Test error vs. number of bagged trees.

are displayed in Figure 5.11.

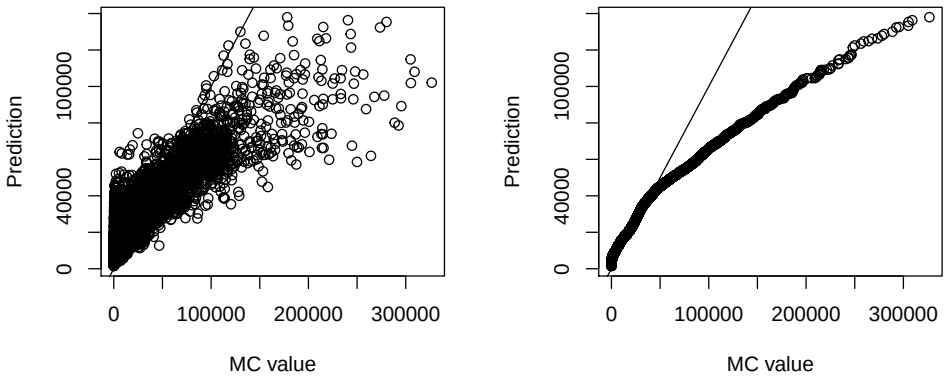


Figure 5.11: Prediction accuracy on test data via model `rf`.

Again, we are confronted with the issue that this model underestimates large values. As in bagging, this comes from the fact that tree models do not have the ability to extrapolate large values.
Furthermore, the number of variables to be considered in each splits, denoted by `mtry`, is set to $\lfloor \frac{p}{3} \rfloor = \lfloor \frac{8}{3} \rfloor = 2$ by default. If so few candidates are randomly selected at each split, it is likely that no relevant predictor is taken, and therefore the split will contribute very little to improve the fit.

To find a suitable choice for the model parameter `mtry`, we examine its impact

on the test error simultaneously with the impact of the number of trees, which is defined by `ntree = 500` in the default setting. The result can be seen in Figure 5.12.

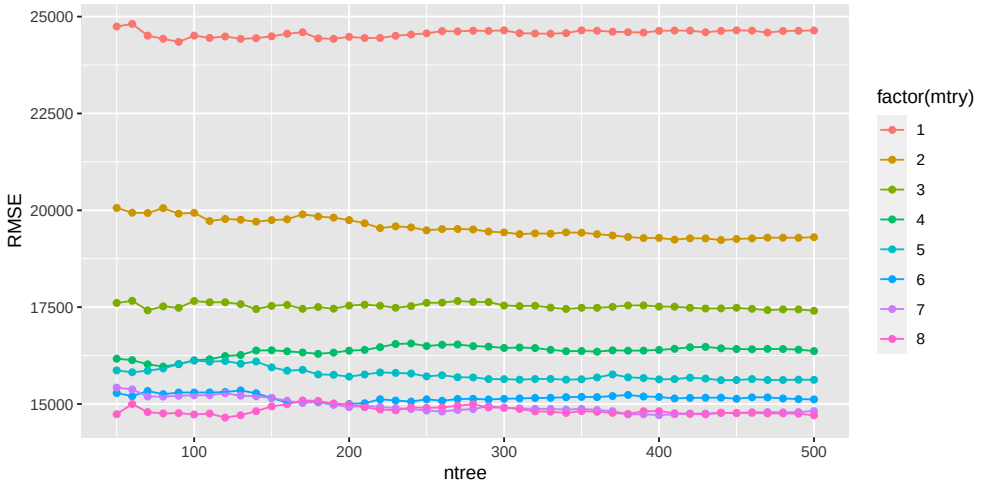


Figure 5.12: Test error vs. number of trees by number of variables considered in each split.

We see that the more variables we consider, the lower is the test error, and seven or eight predictors yield the best result. This means that we do not benefit from the randomness added at the tree construction, which indicates the existence of very few strong predictors. The number of aggregated trees has no strong impact, but again we do not have to worry about overfitting. In conclusion, an appropriate choice could be `mtry = 7` and `ntree = 300`.

5.2.5 Neural networks

Lastly, we implement a neural network using the package `nnet` to model V_0.

Before we can apply this method, min-max normalization is necessary to scale the numerical variables of the training data to the interval $[0, 1]$:

```
> is.num <- sapply(df, is.numeric)
> min.Z <- apply(Z[,is.num], 2, min)
> max.Z <- apply(Z[,is.num], 2, max)
> Z.nn <- Z
```

```
> Z.nn[,is.num] <- t(apply(Z.nn[,is.num], 1,
        function(x) {(x-min.Z)/(max.Z-min.Z)}))
```

A model is fit by

```
> net1 <- nnet(v0 ~ ., data = Z.nn, size = 1, decay = 5e-4, maxit = 1000,
linout = T)
```

where `size` denotes the number of hidden units, `decay` is the weight decay parameter used for regularization, `maxit` is the maximum number of iterations in the training process, and `lineout = T` is always needed for a continuous response. The function automatically converts categorical variables into binary dummies and selects the same reference levels as the GLM, i.e. `GMAB ratchet` and `Female`.

The fitted neural net is visualized in Figure 5.13. Black connections represent positive weights whereas gray connections represent negative weights. The thicker the line the larger is the absolute value of the fitted parameter.

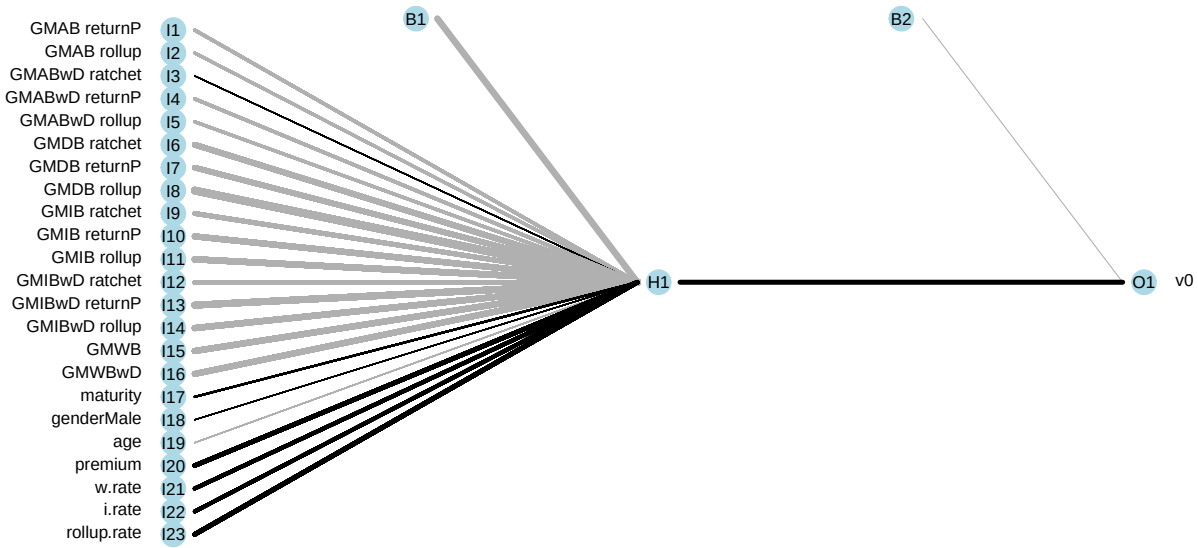


Figure 5.13: The neural network from model `net1`.

With only one hidden unit in the network, the architecture allows us to understand the impact of the predictors. Since the logistic function $g(u) = \frac{1}{1+e^{-u}}$, which transforms the weighted average of the input, is an increasing function and the weight from `H1` to `O1` is positive, increasing predictors with positive weights to the hidden unit leads to a larger output of the network. If we increase the variables with outgoing negative weights, the response value decreases. So a network with only one neuron shows similarity to a GLM. However, since we do not assume any distribution, the parameter estimates are not restricted.
Furthermore, since we work with a scaled input, we can infer directly from the absolute value of the weights, i.e. from the thickness of the lines, how strongly the predictors influence the response variable.
We observe that most of the `product` weights are relatively large and hence indicate a high influence on the predicted value. Moreover, all product parameters except the one of `GMABwD ratchet` have a negative weight and thus yield a lower fair value than the baseline `GMAB ratchet`. Generally, the signs of all predictor coefficients coincide exactly with the ones in the Gaussian GLM `glm.2`. That is, except for age, all numerical predictors increase the output of the neural net with increasing explanatory value. The weights of `gender` and `age` are nearly 0 and hence they have hardly any impact on the prediction. In contrast, the variables `premium`, `w.rate`, `i.rate` and `rollup.rate` have large parameters implying a great influence on the fair value within the model.

For prediction, the test data also needs to be scaled and afterwards we rescale the estimated responses to the original scale:

```
> test.nn <- test
> test.nn[,is.num] <- t(apply(test.nn[,is.num], 1,
         function(x) {(x-min.Z)/(max.Z-min.Z)}))
> pred.nn <- predict(net1, testdf.nn)
> pred <- pred.nn * (max.Z['v0'] - min.Z['v0']) + min.Z['v0']
```

The results are plotted in Figure 5.14. We see that large MC values are slightly underestimated by the model `net1`.

So far we have used only one neuron in the hidden layer, and now we examine if it is possible to improve the prediction quality with more hidden units.
Figure 5.15 shows that increasing the number first reduces the test error, but using more than four hidden units results in worse predictive accuracy. Remember that,

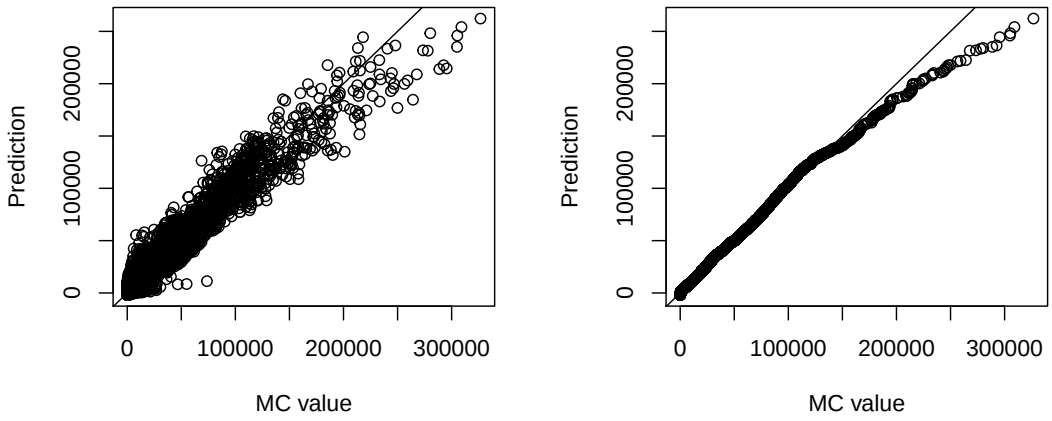


Figure 5.14: Prediction accuracy on test data via model `net1`.

given H hidden units, the number of parameters to be estimated is given by $H(p^*+1)+H+1$. With $p^*=23$, we have

H	1	2	3	4	5	6	7	8	9	10
parameters	26	51	76	101	126	151	176	201	226	251

With only $k=230$ observations in the training set, it is reasonable that models with at least 126 parameters perform worse than the simplest model with only one hidden unit.

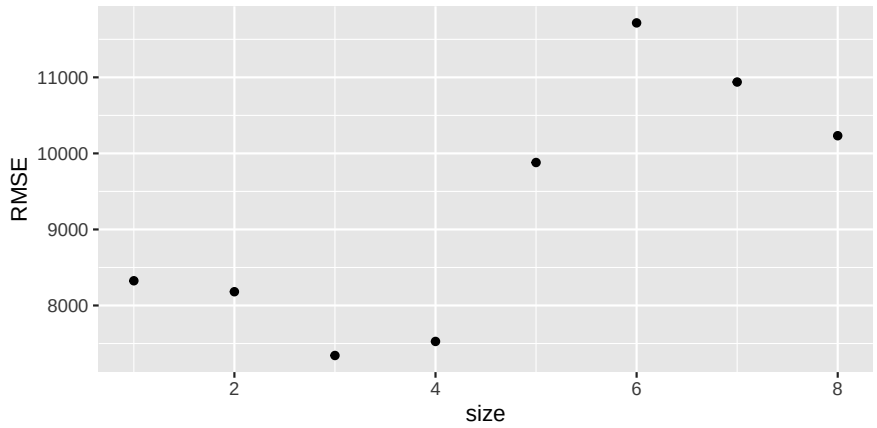


Figure 5.15: Test error vs. number of hidden units.

Therefore, with multiple (but not too many) neurons in the hidden layer it is possible to improve the model. Unfortunately, with a more complex network, it is no longer possible to interpret the influence of the individual parameters.

Next, we want to analyze the effect of the parameter `decay`, which is used for weight regularization. A larger decay parameter forces the fitting process to produce a less complex model. In Figure 5.16, we simultaneously tune `size` and `decay`.

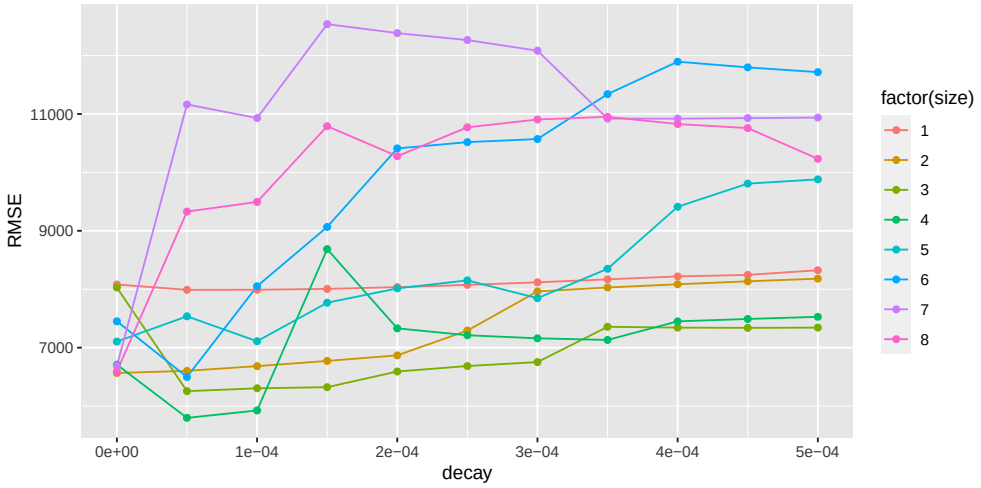


Figure 5.16: Test error vs. weight decay by number of hidden units.

In general, the test error increases with increasing decay, and hence a very small decay value is recommended. Further, we observe that the test error is more volatile for lager networks. This comes from the random initialization of the weights and the fact that we do not have many training samples compared to the number of parameters that have to be estimated. Hence, the fitting process converges to different weight estimates. Although the network of size four leads to the lowest test error, an appropriate choice could be `size=3` because this model seems to be more stable.

Note that if we increased the number of samples in the training set, larger networks might also be able to improve the prediction accuracy.

5.2.6 Comparison

In this part, we directly compare the previous methods to model V_0 by examining the predictive accuracy on a test set. Furthermore, we summarize the impact of the explanatory variables on the fair value.

Predictive performance

For a comprehensive analysis of the prediction quality, we do not limit the number of representatives to $k = 230$ anymore, but also consider sets with $k = 460$ and $k = 920$ contracts. Further, for each k we draw ten sets of representatives, so that we do not rely on model performances that are based on one single dataset only. For validation reasons, half of the entire portfolio is randomly taken as test set and from the other half the representatives are selected.

In total, we fit each of the following models to 30 sets of representatives `Z[[m]]`, for $m = 1, \ldots, 30$:

```
> Gam.glm[[m]] <- glm(v0 ~ ., data = Z[[m]],
         family = Gamma(link = "log"))
> Gauss.glm[[m]] <- glm(v0 ~ ., data = Z[[m]],
         family = gaussian(link = "log"))
> single.tree[[m]] <- rpart(v0 ~ ., data = Z[[m]], method = "anova",
         control = list(minsplit = 5))
> bag.tree[[m]] <- bagging(v0 ~ ., data = Z[[m]], nbagg = 200,
         control = list(minsplit = 2, cp = 0))
> rand.forest[[m]] <- randomForest(v0 ~ ., data = Z[[m]],  mtry = 7,
         ntree = 300, nodesize = 1)
> H3.net[[m]] <- nnet(v0 ~ ., data = Z.nn[[m]], size = 3, decay = 1e-05,
         maxit = 1000, linout = T)
> H6.net[[m]] <- nnet(v0 ~ ., data = Z.nn[[m]], size = 6, decay = 1e-05,
         maxit = 1000, linout = T)
```

where `Z.nn` denotes the scaled version of the representatives.

To examine the performances, the root mean squared error (RMSE) on the test data is averaged for each model and each k. The result is depicted in Figure 5.17.

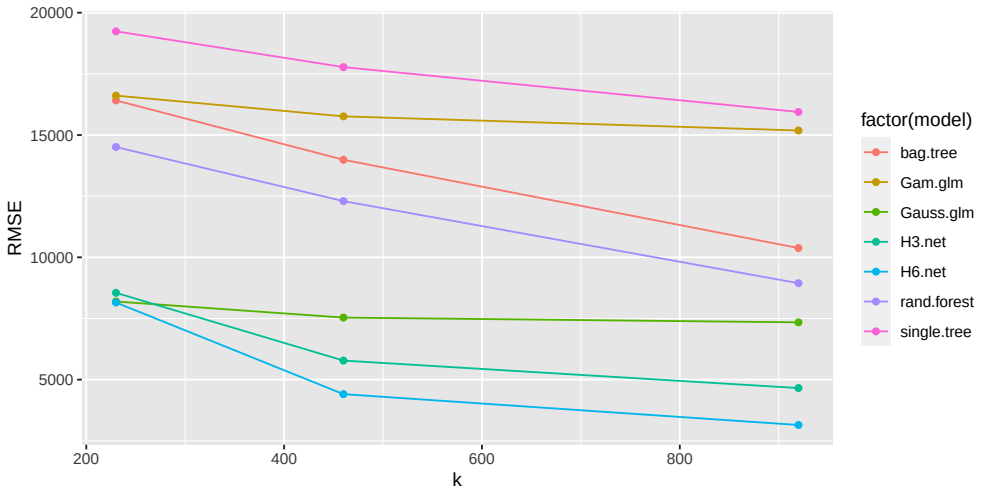


Figure 5.17: Average of RMSE over multiple sets of representatives for each model and each number of representatives.

The tree models appear to have the worst prediction accuracy, but get better for larger k. As seen before, they probably underestimate high fair values, and a larger set of representatives helps to cover a wider range of the response values. The errors of the Gamma GLM are only slightly lower than those of the single tree, while the Gaussian GLM shows a much better performance. It is also interesting that the accuracy of the GLMs seems not to depend on the size of the training set. That is, apparently $k = 230$ representatives are already sufficient to fit the GLMs. The best results are achieved by the two neural networks, especially with at least $k = 460$ representatives and six hidden units.

We are not only interested in the averaged errors, but also in examining how volatile the model performance is due to different training sets. This can be visualized by a boxplot, see Figure 5.18.
Again, one can see that the Gaussian GLM is very stable for all k. In contrast, the nets have more volatile test errors, particularly for $k = 230$. Of course, the single tree turns out to be the worst method. But remembering that the model can only use a few single values as predicted responses, it has a surprisingly acceptable quality and is remarkably stable, especially for $k = 920$.
Generally, we can say that the more representatives we have, the less volatile is the resulting prediction error with respect to distinct data sets.

In the next Figure 5.19, the percentage error (PE) of each model and each k is

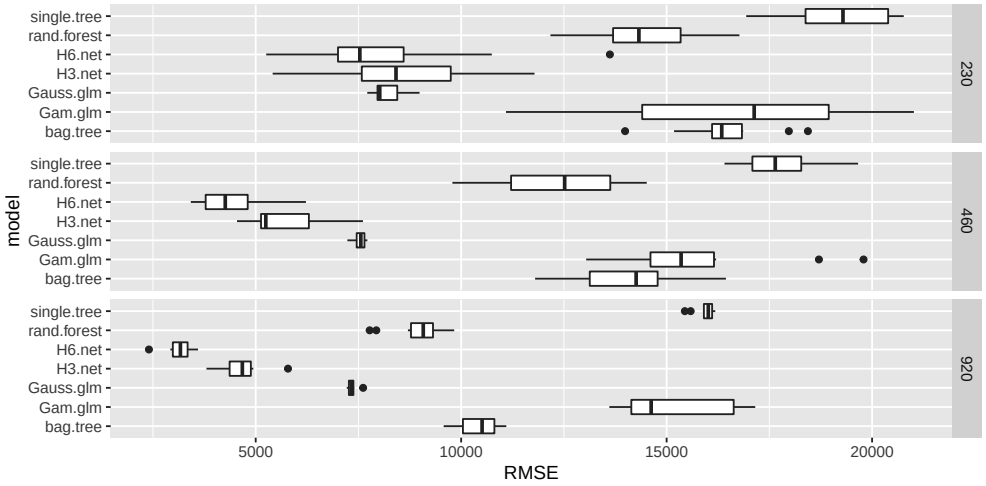


Figure 5.18: Boxplot of RMSE based on multiple sets of representatives

depicted. The percentage error for estimates $\hat{y}_1, \ldots, \hat{y}_n$ with actual response values $y_1, \ldots, y_n$ is given by

$$PE = \sum_{i=1}^{n} \frac{\hat{y}_i - y_i}{\sum_{i=1}^{n} y_i}$$

Clearly, the PE is not a measure of fit for single observations because over- and underestimated values cancel each other out. However, it is useful if we are interested in the sum of the today's fair values over all contracts, as in the solvency calculation. The error can thus be interpreted as a measure of fit on the portfolio level.

The diagram shows that with a Gamma GLM the fair values are on average extremely overestimated and it therefore fails to predict V_0. For $k = 230$ and $k = 920$, the test errors on the portfolio level of the remaining methods are very close to 0. Only in case of 460 representatives, all tree models underestimate the value on average. Although we have already drawn several sets of representatives, it may be the case that the maxima of fair values of the sets with $k = 460$ observations are lower than corresponding maxima of the sets for other k. This would explain the larger negative PE of the tree models for $k = 460$.

Lastly, we would like to consider how stable the methods are with respect to randomness during the fitting process. This behavior is examined by fitting the models multiple times on the same set of representatives, but using different seeds for the random processes. The function `set.seed()` specifies the starting number,

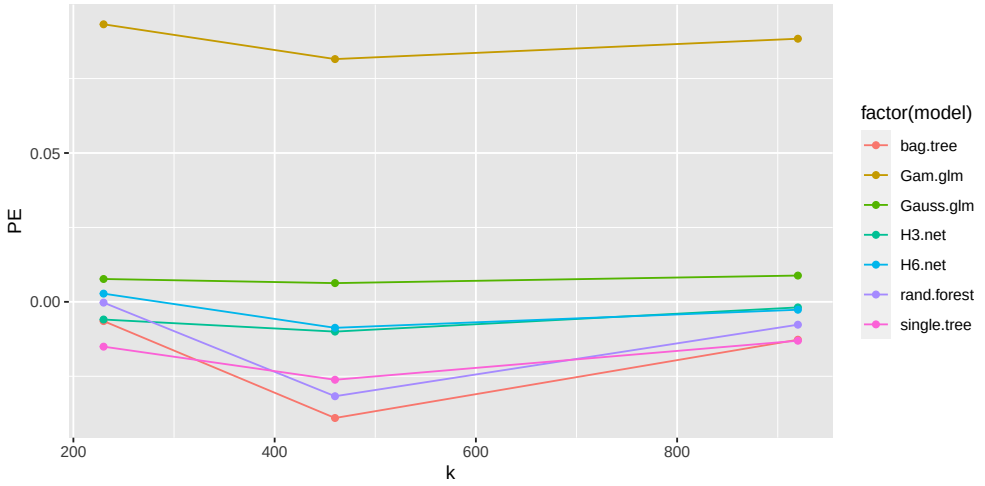


Figure 5.19: Average of PE over multiple sets of representatives for each model and each number of representatives.

which is used to generate a sequence of random numbers, and ensures reproducible results when you start with that seed each time you run the same process. The RMSE results are visualized in a boxplot, see Figure 5.20.

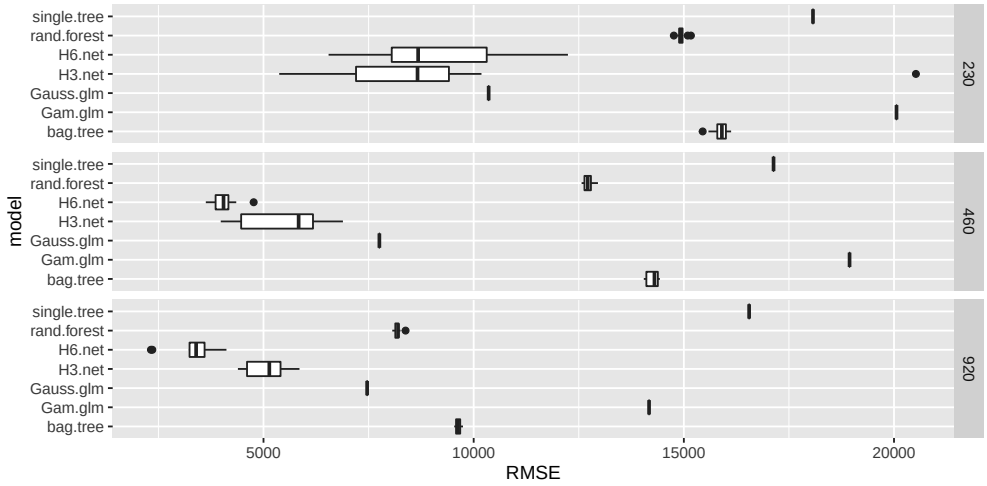


Figure 5.20: Boxplot of RMSE based on distinct seeds specifying the random processes during fitting.

It is obvious that the test errors of neural networks are quite unstable, especially for small k. This is plausible because the starting parameters are chosen randomly in this method. Therefore, distinct initializations could lead to different prediction

results. However, more samples help to reduce this effect. All other methods appear to be very stable. GLMs and single trees do not contain any randomness in their fitting. In bagging only the choice of boosting samples is random, and in random forests additionally the splitting candidates are selected at random.

To get an impression of the time, which is needed for model training, Table 5.2 shows the average runtime to fit one model for a given k. The single tree and the GLMs are extremely fast. But even the bagged tree, which is the slowest of all models, only takes a few seconds.

Model	k=230	k=460	k=920
single.tree	0.005	0.007	0.009
Gauss.glm	0.006	0.007	0.009
Gam.glm	0.009	0.010	0.016
H3.net	0.163	0.286	0.502
rand.forest	0.271	0.736	2.272
H6.net	0.340	0.606	1.253
bag.tree	1.188	2.065	3.950

Table 5.2: Average runtime for model training in seconds.

In summary, if we are restricted to only $k = 230$ representatives, the Gaussian GLM is recommended within this framework because it leads to very stable and good results. Otherwise one should use $k = 920$ and the neural net with six hidden units as this methods yields the best predictive accuracy within this comparison.

Risk drivers for V_0

First of all, it should be remembered that the calculation of the (estimated) fair values is actually possible via MC simulation, but only computationally expensive. This means that the actual dependencies are known. Thus, we also know the parameters that determine the value V_0 and our data set contains exactly these explanatory variables. So in the following analysis we only examine how strongly the predictors influence the response value.

The explanatory variable `product` seems to have the biggest influence on V_0 in all considered models. In both the Gaussian GLM and the neural net with one hidden unit, we have seen the following effect (see Figure 5.3): `GMAB ratchet` and

`GMABwD ratchet` yield the highest values, while medium large values are obtained by `GMAB returnP`, `GMABwD returnP`, `GMAB ratchet` and `GMABwD ratchet`. All remaining products lead to a relatively low fair value, and we generally see that within each guarantee the ratchet type yields the highest values, followed by the roll-up benefit base and the lowest is the return of premium.
The model fits are based on a limited number of representatives and hence small deviations to our expectation are possible. However, it makes absolutely no sense that e.g. the values of the roll-up benefits are always lower than the values of the ratchet benefit bases. This is due to the fact that the parametric models consider the influence of all weights simultaneously and that NA entries in the rates are replaced with 0 during data preparation.
Exemplarily, we explain the consequences of setting `rollup.rate = 0`. Originally, the policies with a roll-up benefit base have been created with roll-up rates in the range $[0.01, 0.05]$. Further, we have seen that all previous models show a great influence of the roll-up rate on V_0. Hence, if we consider e.g. a ratchet type policy, the large impact of the roll-up rate and `rollup.rate = 0` cause the fair value to be predicted extremely low in the parametric models. To compensate for this effect, the models increase the coefficients for ratchet type and premium back products. Since the guaranteed benefits of ratchet type contracts are larger than the guaranteed part of the repayment of the initial investment, the ratchet type products appear to have the highest fair values within each guarantee.
Similarly, setting `i.rate = 0` causes all products without a GMIB to have much higher values compared to GMIB levels than actually expected. Therefore, the GMAB options are the levels with the highest fair values in the parametric models. The same applies to `w.rate = 0`, even though we do not see a strong effect here. In summary, this effect does not make the models bad. Only the interpretation of the individual product levels becomes more difficult.

Additionally to the `rollup.rate`, the `premium` is also a strong driver for the today's fair value. This is absolutely plausible, since both of the predictors directly determine the benefit bases. In the parametric models, also `i.rate` and `w.rate` appear to be important. The income rate directly affects the survival benefit, while a large withdrawal rate enables to withdraw larger amounts every year and hence contributes to benefit from the guarantee. As expected, all of these four numerical predictors increase V_0 with increasing explanatory values.

The variables `gender` and `age` have hardly any influence on the today's value of guaranteed benefits.

5.3 Fair value at time 1

To determine the SCR as a quantile of the loss distribution, we first need to create the empirical distribution of the fair values at time 1 under the real-world measure P. That is, for a variety of real-world stock scenarios at time 1, the corresponding time-1 values of the entire portfolio have to be calculated. Although we are only interested in the aggregated values over the whole portfolio, we must first calculate the values for each contract individually, as it is not possible to directly determine the total value.
We start with some results of the nested Monte Carlo simulation and continue with the application of machine learning methods to make the calculation feasible for a large portfolio in terms of time. In the last part of this section, a comparison of the model performance and the influence of the predictors is provided.

5.3.1 Results from nested MC simulation

First of all, the scenarios for the stock value at time 1 are sampled via the real-world generator in Section 3.2.2. Figure 5.21 shows the histogram of the resulting $N = 1000$ scenarios. By construction, the scenarios are log-normally distributed.

Now, in the nested MC approach, for each of these scenarios and each contract the corresponding fair values at time 1 have to be calculated. For $n = 10000$ contracts and $N = 1000$ scenarios this adds up to 10 million combinations, where each has to be evaluated via an inner risk-neutral simulation. It is not possible to run this nested simulation on my computer.

Nevertheless, we would like to see some results, and therefore we apply the simulation only on the first 100 contracts and reduce the number of inner paths to 100. This took approximately 24 minutes. Note that for the entire portfolio and $J = 1000$ inner paths, the simulation is expected to take 1000 times longer, which equals 394 hours.

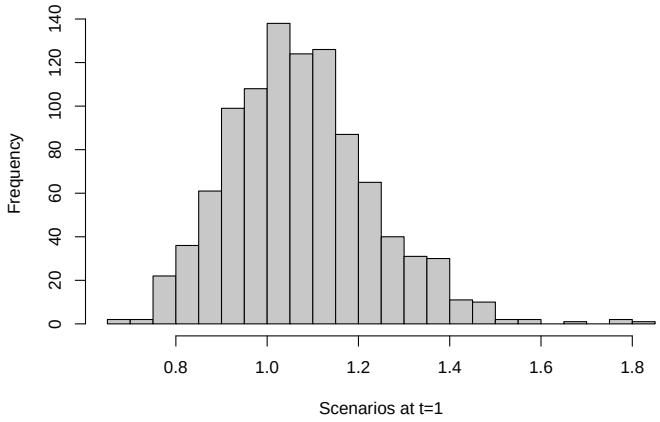


Figure 5.21: Histogram of generated scenarios at time 1.

In Figure 5.22, for each stock scenario the resulting time-1 value aggregated over the 100 contracts is plotted against its scenario value.

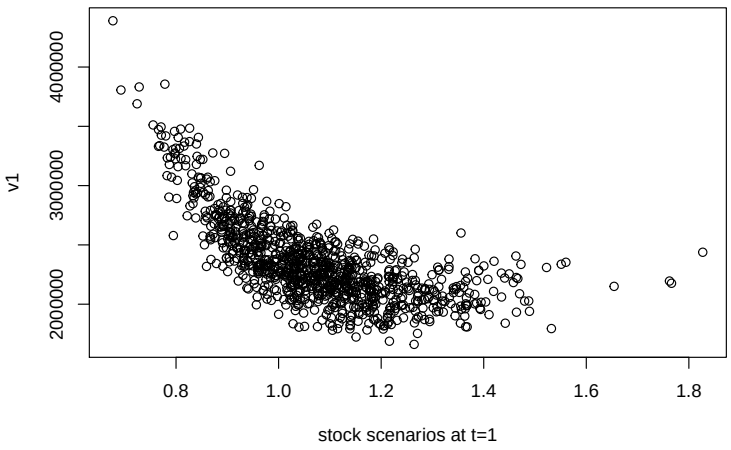


Figure 5.22: The stock scenarios at time 1 vs. the total fair values for 100 contracts simulated with a reduced number of inner paths.

We see that the value decreases for increasing stock values and then remains on the same level from a certain point on. This is absolutely plausible because the insurance company must pay guaranteed benefits if the account value is low, and the account value is directly affected by the stock values. According to (2.2), the

account value at time 1 before a possible withdrawal is given by $A_1^- = A_0 \frac{S_1}{S_0} = P \cdot S_1$ in our setting since we have $A_0 = P$, $S_0 = 1$, and no guarantee fees that reduce the account value.
Of course, a low account value at time 1 does not necessarily imply a low value at the time of the guarantee payment, but it does make it more likely.

5.3.2 Data preparation and sampling

The data preparation for the insurance portfolio is conducted equivalently to 5.2.2. Additionally, we now have the set of stock scenarios at time 1, and therefore we create a combined data set `comb`, which consists of the combinations of all contracts and all scenarios, i.e. has 10 million observations.

Next, we need to select a subset of the combinations that will serve as input to the machine learning methods.
This time, we do not randomly choose the representatives because the scenarios are not uniformly distributed, see Figure 5.21. In particular, we know that by construction they are log-normally distributed. Hence, if we selected the representatives randomly from `comb`, we would get many scenarios of average stock value but hardly any extreme values. On the other hand, calculating the SCR means that we are interested in extreme quantiles of the one-year losses. Figure 5.22 suggests that we will have higher losses for very low scenarios and therefore, the goal is to estimate the fair value well even at the tails of the scenario distribution. For this reason, we would like to select the representatives in such a way that the included scenarios are equidistantly spread over the range of possible scenarios and hence also contain extreme values.

The strategy is to separately choose from the set of contracts and scenarios. As before, a subset of contracts is randomly drawn from the entire portfolio. To obtain scenario samples, we define equidistant points on the range of the given scenarios and then select the scenarios that are closest to these points. Again, we take $k = 10p^*$ as number of representatives, but now p^* equals 24 as the stock value at time 1 is also considered as an explanatory variable now.

```
> k <- 240
> Z.Policies <- df[sample(nrow(df), k),]
> equ.points <-seq(min(scenarios),max(scenarios),
```

```
        (max(scenarios)-min(scenarios))/(k-1))
> Z.scenarios <- sapply(equ.points,
        function(x) { scenarios[which.min(abs(scenarios-x))] } )
> Z <- cbind(Z.Policies, S1 = Z.scenarios)
```

To check the model performance we also need a test set. We randomly select 5000 samples from all combinations by

```
> test <- comb[sample(nrow(comb), 5000),]
```

In the following, we apply the machine learning methods to predict the fair value at time 1 for each scenario and each contract. In all methods, the time-1 values are modeled by the contract attributes and the stock scenario at time 1, using the representative subset `Z`.
For the contract-specific explanatory variables, we expect an influence on V_1 similar to the one on V_0. Therefore, in the following we will focus more on the relation of V_1 with respect to scenario values S_1.

5.3.3 Generalized linear models

Again, we start with one Gamma GLM and one Gaussian GLM, both with a log link function. At $t = 1$ it could happen that the fair value of one contract at a certain (high) scenario already equals 0. To prevent that the algorithm has to deal with zeros, we slightly shift the response values in positive direction. However, since we generally have very large values, this small shift has no effect on the prediction.

```
> glm.1 <- glm(v1 + 1e-03 ~ ., data=Z, family=Gamma(link = "log"))
> glm.2 <- glm(v1 + 1e-03 ~ ., data=Z, family=gaussian(link = "log"))
```

Exemplarily, the summary of the coefficients from `glm.2` is given by:

```
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)             1.009e+01  1.894e-01  53.251  < 2e-16 ***
productGMAB returnP    -1.910e+00  3.552e-01  -5.378 1.96e-07 ***
productGMAB rollup     -2.444e+00  2.262e-01 -10.807  < 2e-16 ***
productGMABwD ratchet   -1.737e-01  9.696e-02  -1.791 0.074640 .
productGMABwD returnP -1.509e+00  2.266e-01  -6.661 2.24e-10 ***
productGMABwD rollup   -2.237e+00  2.081e-01 -10.753  < 2e-16 ***
```

```
productGMDB ratchet    -2.675e+00  8.124e-01  -3.293 0.001159 **
productGMDB returnP   -4.918e+00  6.483e+00  -0.758 0.448986
productGMDB rollup    -6.469e+00  2.344e+00  -2.760 0.006272 **
productGMIB ratchet    -5.771e+00  7.331e-01  -7.872 1.69e-13 ***
productGMIB returnP   -6.905e+00  8.935e-01  -7.727 4.15e-13 ***
productGMIB rollup    -7.827e+00  8.058e-01  -9.713  < 2e-16 ***
productGMIBwD ratchet  -5.089e+00  7.314e-01  -6.957 4.12e-11 ***
productGMIBwD returnP -7.778e+00  1.549e+00  -5.022 1.07e-06 ***
productGMIBwD rollup  -7.541e+00  7.917e-01  -9.525  < 2e-16 ***
productGMWB           -4.064e+00  1.264e+00  -3.215 0.001505 **
productGMWBwD         -3.683e+00  1.097e+00  -3.358 0.000928 ***
maturity               4.239e-02  6.457e-03   6.565 3.84e-10 ***
genderMale             1.463e-01  5.566e-02   2.628 0.009219 **
age                   -9.102e-03  2.341e-03  -3.888 0.000135 ***
premium                3.695e-06  2.521e-07  14.657  < 2e-16 ***
w.rate                 2.852e+01  1.654e+01   1.725 0.086006 .
i.rate                 5.470e+00  8.031e-01   6.811 9.55e-11 ***
rollup.rate            6.186e+01  4.658e+00  13.280  < 2e-16 ***
S1                    -4.201e-01  8.046e-02  -5.222 4.17e-07 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

As expected, the product levels have an influence similar to the one of the previous Gaussian GLM for V_0. Small deviations are natural since the fit is based on only a small number of representatives. The same holds for most of the other coefficients. Only `gender` and `age` now appear to be significant within this model, and in addition age has a stronger negative impact on the fair value. There seems to be no explanation for this, which could indicate that the model has problems to reflect the true relationships.

The estimated coefficient of `S1` is negative and hence the response is expected to decrease with increasing stock value. According to the model, the relative impact of the scenario at time 1 is given by

```
> exp(glm.2$coefficients['S1'] * 0.1)

0.9588565
```

This implies that if we consider the same contract at two time-1 scenarios, which

differ by 0.1, the fair value at the higher stock value is expected to be 4.1% lower. Moreover, when we know the time-1 value of one contract for a certain scenario, this relative impact allows us to directly determine the fair values for all other scenarios. This is a nice property, but it also means that the shape of the V_1-distribution is solely determined by the coefficient β_{S_1}.

In Figure 5.23, predictions for the test data are plotted against their Monte Carlo estimates. The Gamma model extremely overestimates the response values, even for small MC values. We conclude that this model fails to predict the true relationship. The Gaussian distribution yields a better fit, but the predicted values still differ in the higher range.
This again encourages the presumption that a GLM struggles to capture the true behavior of V_1 with respect to different scenario values.

5.3.4 Regression trees

Single trees

An implementation of a single tree to model the value V_1 is given by

```
> single.tree <- rpart(v1 ~ ., Z, method = "anova",
        control = list(minsplit = 5))
```

Figure 5.24 visualizes the resulting tree, which consists of eight terminal nodes. Again, the product and the premium are important predictors and also the roll-up rate defines a split. Now, additionally the stock value at time 1 is used twice for splitting. We notice that regions above the cutting points of `S1` always have lower response values. In contrast, higher realizations of both the premium and roll-up rate lead to higher fair values.
The region with the highest predicted response contains samples of the products `GMAB ratchet`, `GMABwD ratchet`, `GMAB rollup`, `GMABwD rollup`, `GMIBwD ratchet` and `GMIBwD rollup` with large `premium` and relatively low stock scenario `S1`.

In Figure 5.25, we face again a poor predictive performance because the single tree only uses eight discrete response values for prediction.

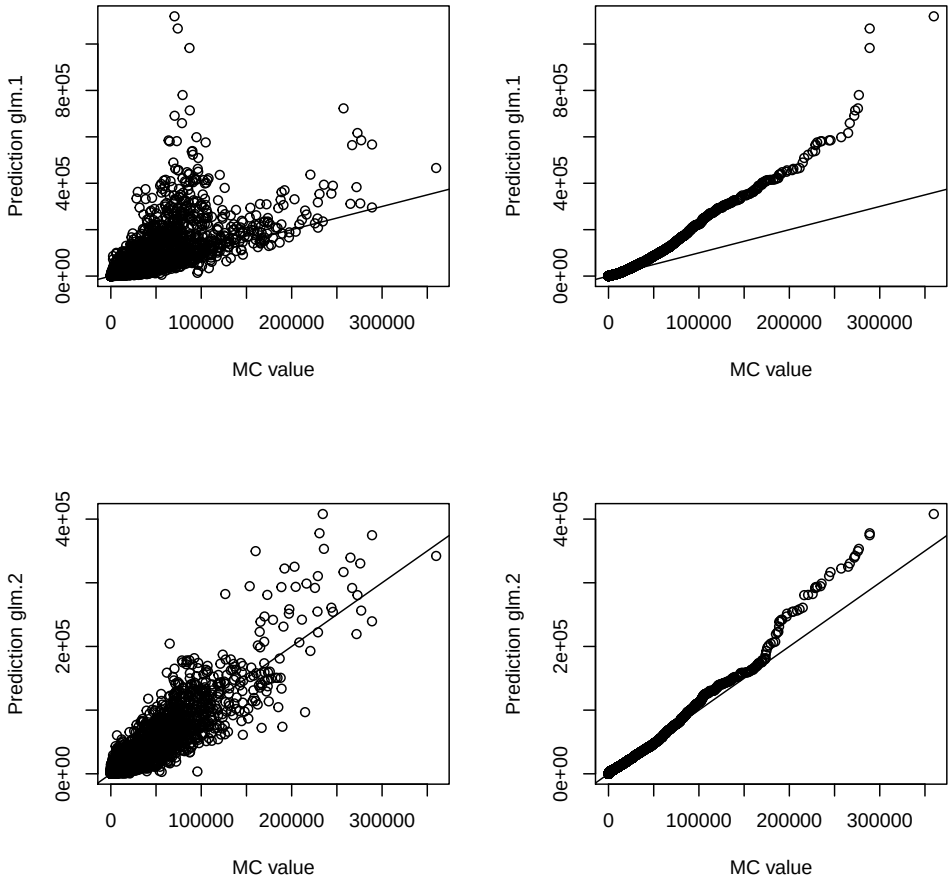


Figure 5.23: Prediction accuracy on test data for (top) Gamma `glm.1` and (bottom) Gaussian `glm.2`.

Bagging

We obtain the bagged model to explain the variation of V_1 by

```
> bag.tree <- bagging(v1 ~., data = Z, nbagg = 200,
          control = list(minsplit=2, cp=0))
```

and the corresponding prediction is shown in Figure 5.26.

As before, the model significantly underestimates large MC values, as it cannot perform extrapolation and the maximum response value of the set of representatives is much lower than the maximum value of the test set.

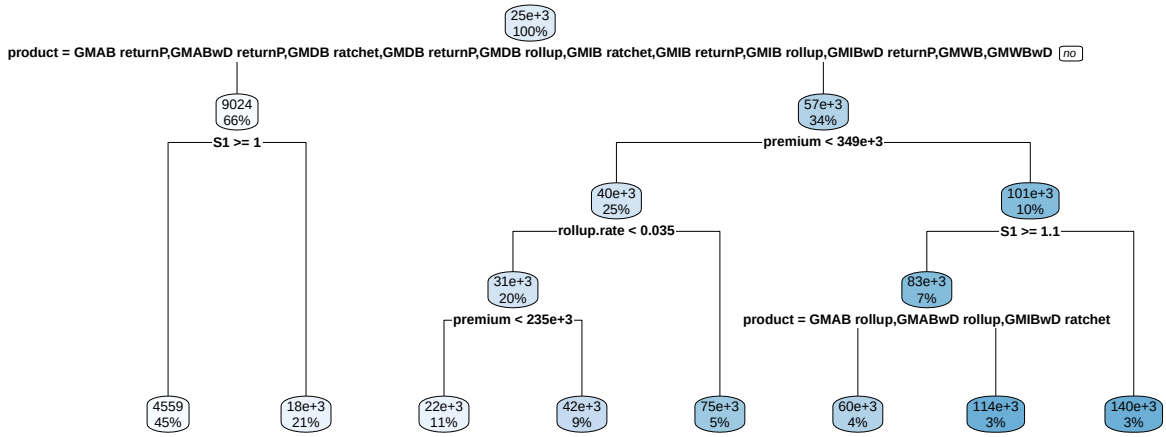


Figure 5.24: Tree chart of model `single.tree`

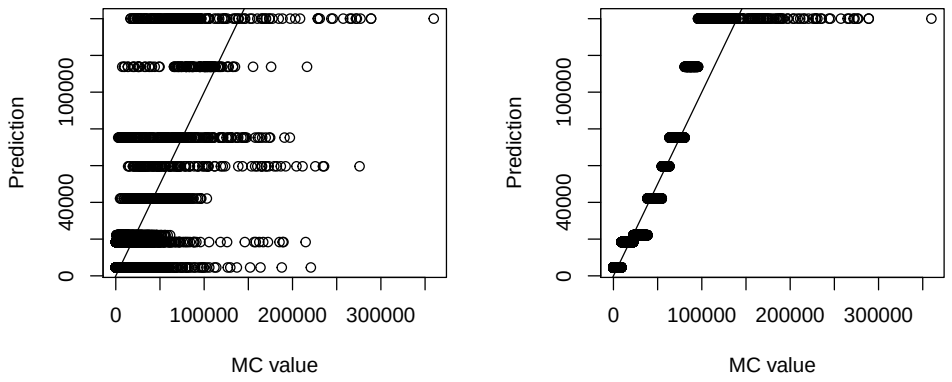


Figure 5.25: Prediction accuracy on test data via model `single.tree`.

To check if the number of bagged trees is sufficient, we examine the test error for different values of `nbagg`. According to Figure 5.27, we see that `nbagg = 200` has already been a suitable choice.

Random forests

Next, we consider a model of random forests. It is given by

```
> rf <- randomForest(v1 ~., Z, nodesize = 1)
```

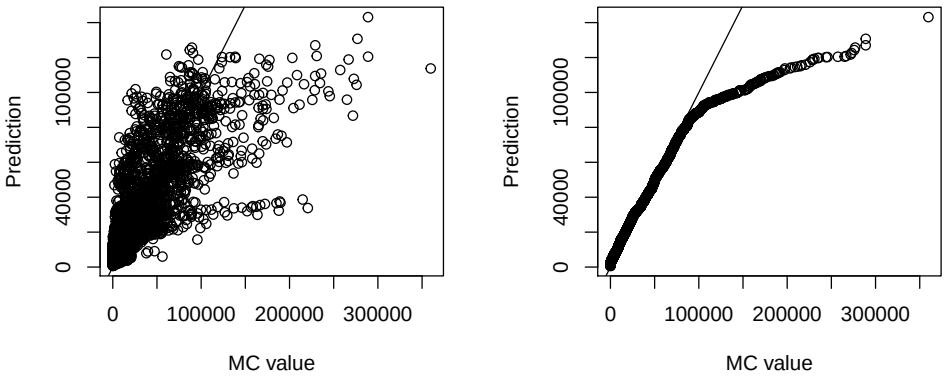


Figure 5.26: Prediction accuracy on test data via model `bag.tree`.

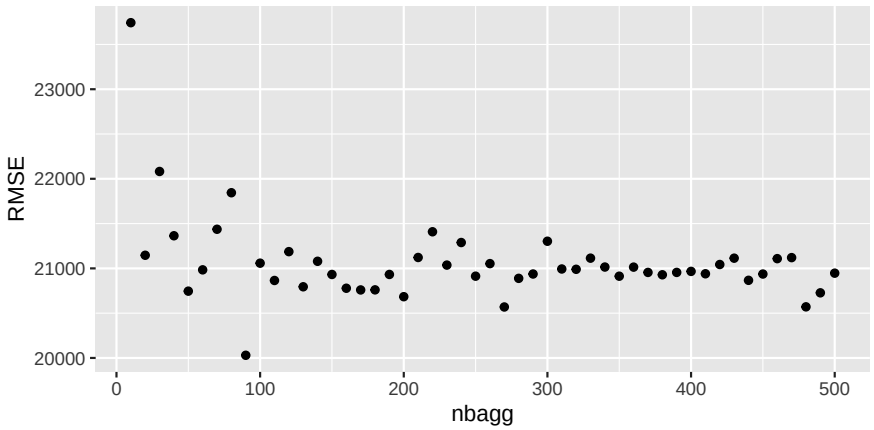


Figure 5.27: Test error vs number of bagged trees.

where the default number of candidates at each split, denoted by `mtry`, now equals $\lfloor \frac{p}{3} \rfloor = \lfloor \frac{9}{3} \rfloor = 3$.

The prediction is displayed in Figure 5.28, and we face the same problem of underestimating large responses due to the incapability of the model to predict response values outside the range of the representatives.

To make a proper choice of the parameters `mtry` and `ntree`, we simultaneously consider their impact on the test error, see Figure 5.29.
The results are quite stable for different numbers of trees and one could say that `ntree = 300` is sufficient. On the other hand, the choice of `mtry` makes a sig-

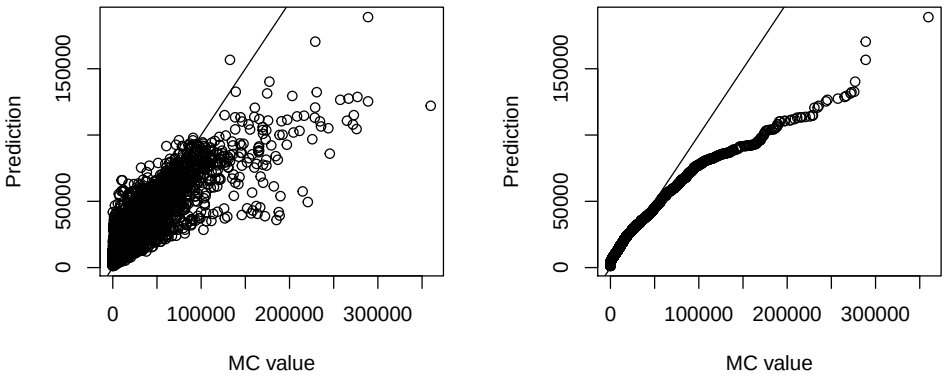


Figure 5.28: Prediction accuracy on test data via model `rf`.

nificant difference. Randomly taking one predictor at each split yields the worst results. However, considering all or all but one variables in each split does not lead to the best model performances either. The lowest test error is obtained by `mtry = 5`. That is, in this setting we indeed profit from the randomness within random forests, and the quality of the prediction can be improved by de-correlated trees.

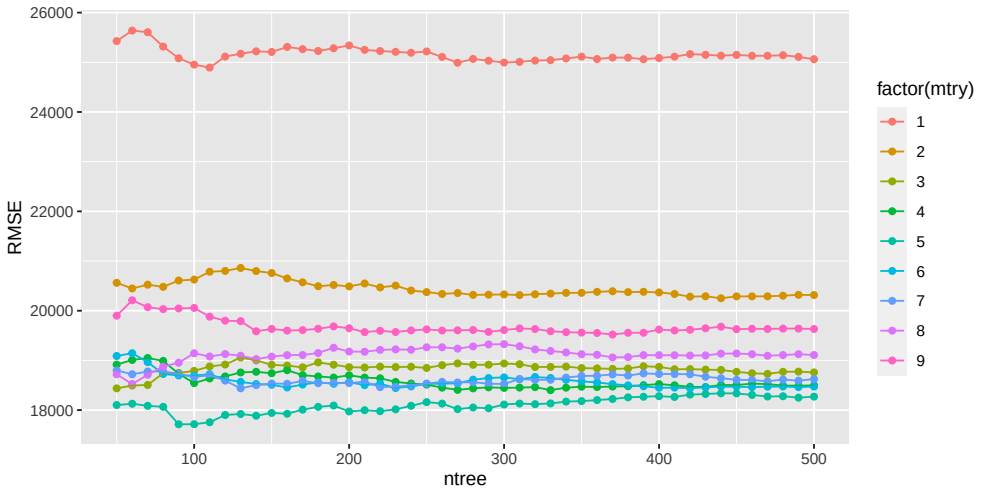


Figure 5.29: Test error vs. number of trees by number of variable candidates at each split.

5.3.5 Neural networks

Lastly, we fit a neural network with one hidden unit on the scaled dataset `Z.nn` by

```
> net1 <- net1 <- nnet(v1 ~ ., data = Z.nn, size = 1, decay = 5e-4,
        maxit = 1000, linout = T)
```

and again `GMAB ratchet` and `Female` are automatically chosen as baseline levels.

The trained net is shown in Figure 5.30. The structure of the neural net with only one neuron in the hidden layer again allows for an interpretation of the weight coefficients. The predictor `S1` has a light negative weight. That is, by the model, the fair value decreases with increasing stock scenarios. We also see that the `product` levels have a large influence of the prediction. In contrast, `gender` and `age` have weights close to 0 and hence hardly have any impact.
It is interesting that in this network the `premium` and `w.rate` are not that important anymore compared to the neural net model of V_0. This could be due to the fact that we are working with a different set of representative contracts, or it could mean that the neural network with one hidden unit also has difficulties to reflect the dependencies of V_1 because of the additional variation due to distinct scenario values.

The predicted response values are plotted in Figure 5.31, and we conclude that the model `net1` with only one hidden neuron is able to indeed reproduce the behavior quite well.

To check whether we can make the model even better, we simultaneously analyze the impact of the parameters `size` and `decay` in Figure 5.32. In general, you can see that for small weight decays the prediction seems a bit unstable. For sufficiently large decay, the test error is first decreasing with the size until `size = 4` and then increasing again. Further, for a size of least six the test error is very volatile. This is absolutely reasonable because for a net with six hidden units we need to estimate 175 parameters.
Overall, one can say that setting `decay = 3e-04` and `size = 4` is appropriate, given the current set of representatives. If we increase the number of representatives, more hidden units might help to learn the relationship better.

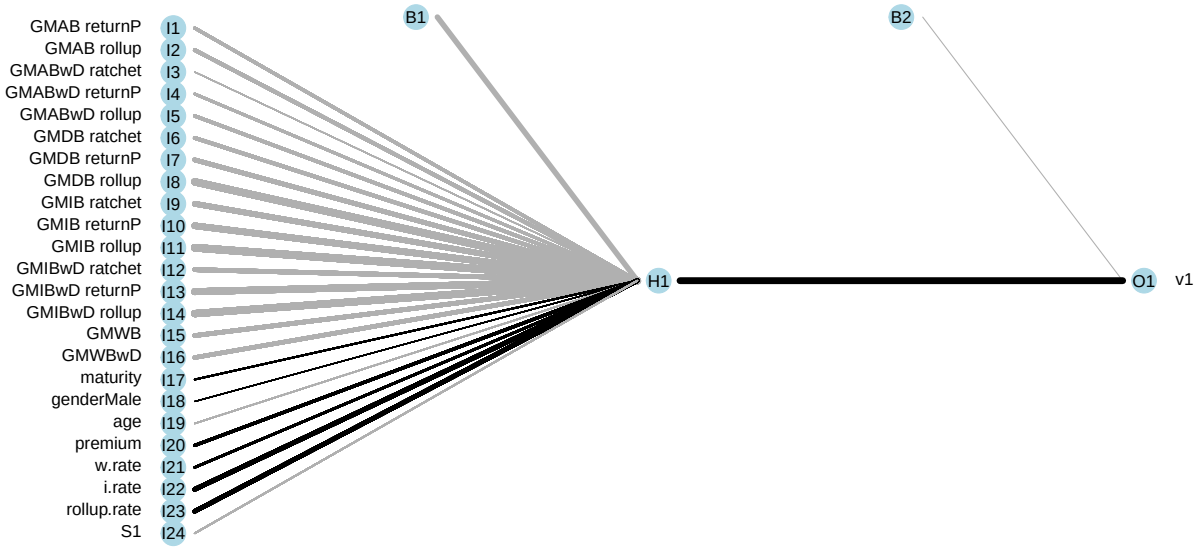


Figure 5.30: The neural network from model `net1`.

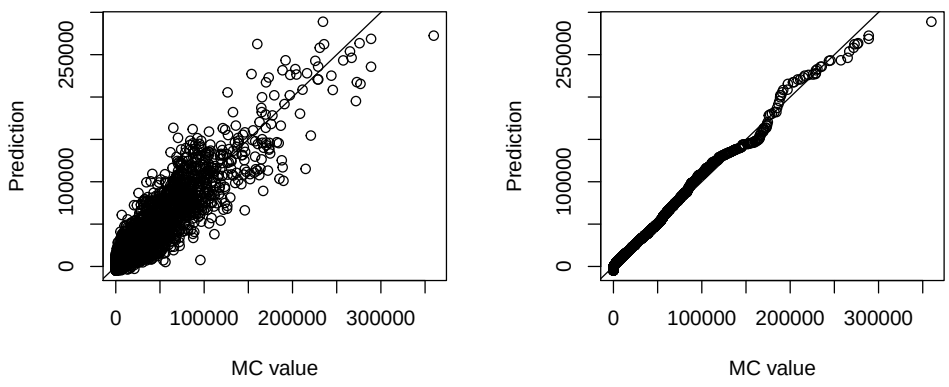


Figure 5.31: Prediction accuracy on test data via model `net1`.

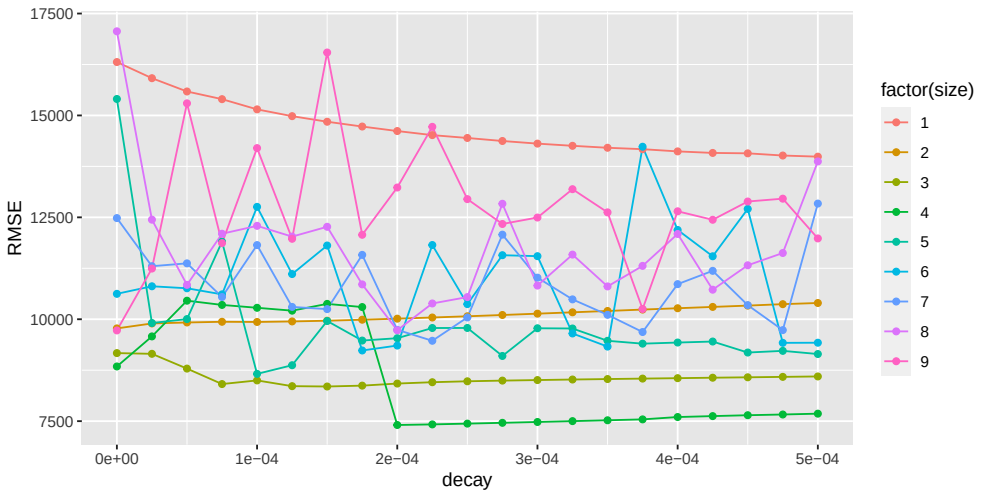


Figure 5.32: Test error vs weight decay by number of hidden units.

5.3.6 Comparison

Lastly, we compare the results of the previously introduced machine learning methods, which attempt to model the behavior of V_1 using the insurance attributes and the stock scenarios as explanatory variables. We analyze both the quality of the models in terms of predictive accuracy and the impact of the individual predictors.

Predictive performance

Again, we examine the predictive accuracy of the models based on multiple sets of representatives of different sizes. According to the description in 5.3.2, we randomly draw a test set of size 5000 and for each $k = 240, 480, 960$ we create ten sets of representatives.

For every set $m = 1, \ldots, 30$ of representatives, we fit the following models:

```
> Gam.glm[[m]] <- glm(v1 + 1e-03 ~ ., data = Z[[m]],
          family = Gamma(link = "log"))
> Gauss.glm[[m]] <- glm(v1 + 1e-03 ~ ., data = Z[[m]],
          family = gaussian(link = "log"))
> single.tree[[m]] <- rpart(v1 ~ ., data = Z[[m]], method = "anova",
          control = list(minsplit = 5))
```

```
> bag.tree[[m]] <- bagging(v1 ~ ., data = Z[[m]], nbagg = 200,
          control = list(minsplit = 2, cp = 0))
> rand.forest[[m]] <- randomForest(v1 ~ ., data = Z[[m]],  mtry = 5,
          ntree = 300, nodesize = 1)
> H4.net[[m]] <- nnet(v1 ~ ., data = Z.nn[[m]], size = 4, decay = 3e-04,
          maxit = 1000, linout = T, trace = F)
> H6.net[[m]] <- nnet(v1 ~ ., data = Z.nn[[m]], size = 6, decay = 3e-04,
          maxit = 1000, linout = T, trace = F)
```

The Gamma GLM does not converge for one training sample, and as the results of this method are generally very poor, we exclude the Gamma GLM in further analysis. It is not recommended to apply the model in this context.

The averaged RMSE on the test data for each method and each number of representatives are depicted in Figure 5.33. The tree models provide the highest test errors, especially the single regression tree. But also the Gaussian GLM seems to perform only a little bit better. The best prediction accuracy is achieved by the neural networks. This is plausible and reflects the impressions that we have gained in the previous parts of this section.
For all methods the model quality improves with increasing size of the training data. In this context, it is recommended to select at least 480 representatives.

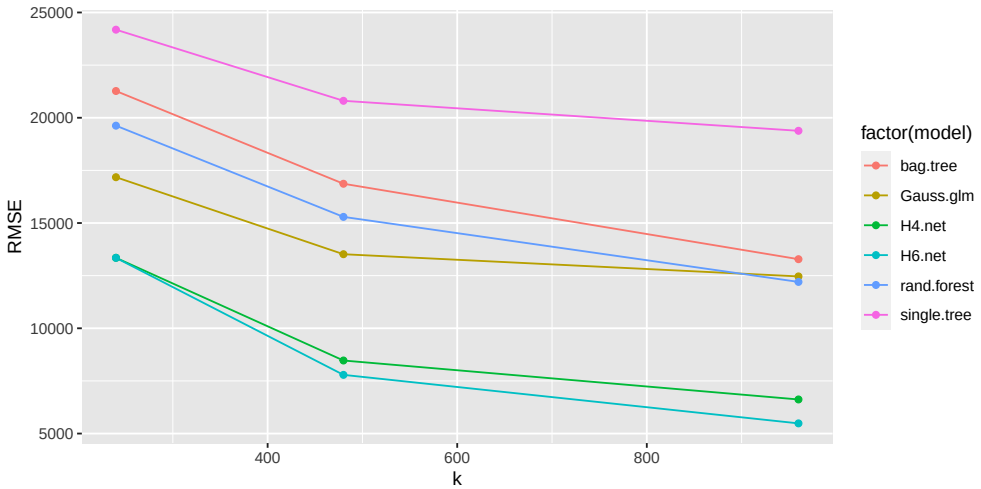


Figure 5.33: Average of RMSE over multiple sets of representatives for each model and each number of representatives.

The corresponding boxplot of the previous graph is shown in Figure 5.34. For all

methods the RSME gets less volatile when the size of the training set is increased. The most stable results with respect to the choice of representatives is achieved by the neural nets at $k = 980$.

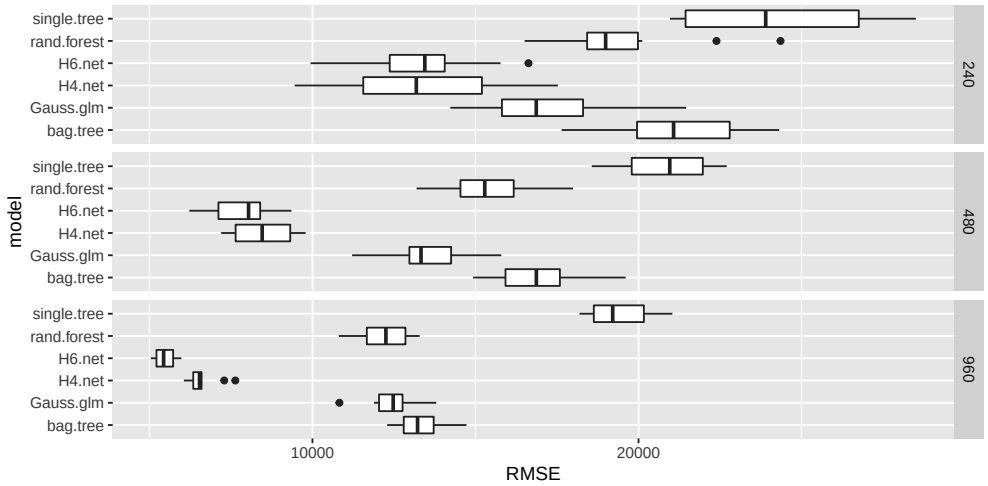


Figure 5.34: Boxplot of RMSE based on multiple sets of representatives

When we consider the averaged percentage errors in Figure 5.35, we see that the Gaussian GLM predicts responses which are on average too large, whereas tree models generally underestimate the values, particularly for smaller k. Only the neural nets generally achieve percentage errors close to zero and on average slightly overestimate the MC values.

Exactly as depicted previously in Figure 5.20 for the estimation of V_0, also here the neural nets have volatile test errors depending on random initialization of the parameters when we use only 240 representatives. Otherwise, the methods are very stable.

In conclusion, a neural net with six hidden units and a set of $k = 980$ representatives is recommended to predict the behavior of the time-1 value in this context.

Risk drivers for V_1

In all models, we have plausibly seen a large impact of the `product` on the fair value. And again, we face the difficulty of interpreting the influence of the individual product levels in the parametric models due to the modification of the roll-up,

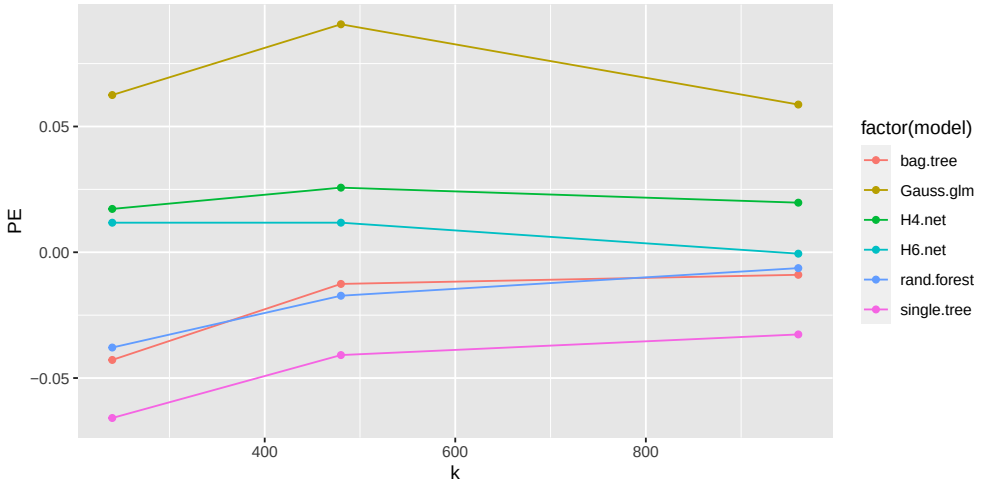


Figure 5.35: Average of PE over multiple sets of representatives for each model and each number of representatives.

withdrawal and income rate. Additionally, the predictor `rollup.rate` appears to be a strong driver for V_1.
The single tree model suggests that both the `premium` and the scenario value `S1` also have a decisive role in the prediction of V_1. Similarly, the Gaussian GLM considers both parameters to be significant within the model. In contrast, the neural net with one hidden unit does not weight the inputs of the premium and the stock value that much. However, in all models we have an increasing effect of the premium and a decreasing effect of the scenario on the fair value V_1.

Overall, one might conclude that due to the additional variation of the fair value with respect to the time-1 scenarios, the models that allow for interpretation, might have small difficulties in reproducing the dependencies on the contract attributes. Note that only within the simplest models - the GLM, the single tree and the neural net with one hidden unit - the impact of the predictors can be interpreted. Since the neural networks with four and six neurons provide good predictions on the test data, these two more complex models seem to be affected only slightly or not at all by this issue.

5.4 Estimation of SCR

Finally, we can put all parts together and estimate the SCR according to the framework presented in Chapter 3. Remember that the approach consists of the following basic steps

1. Estimate V_0 for the entire portfolio.
2. Estimate $V_1^{(\ell)}$ for the entire portfolio given scenario samples, $\ell = 1, \ldots, N$.
3. Calculate the losses $\Delta^{(\ell)} = -V_0 + \frac{V_1^{(\ell)}}{1+r}$.
4. Estimate the SCR by the empirical 99.5%-quantile of the losses.

Note that here the terms V_0 and $V_1^{(\ell)}$ always refer to the aggregated fair values over all contracts. We speed up the nested simulation by applying machine learning methods that predict the fair values for both times based on a small number of representatives.

We use the Gaussian GLM, the random forest and the neural net of size six from 5.2.6 with $k = 920$ and the corresponding models from 5.3.6 with $k = 960$ because they achieve the most promising results under each of the fundamental machine learning methods, respectively.

First, the today's fair value of the whole portfolio is estimated. The partial steps and their runtimes are listed in Table 5.3. The time for selecting the representative contracts and for the MC simulation is of course the same within all three methods. Note that the fitting step also includes data preparation and, in case of a neural net, scaling. For the neural net we also have to re-scale the values in the prediction step.

Model	GLM	Random forest	Neural net
Select representatives	0.12	0.12	0.12
MC simulation	41.93	41.93	41.93
Model fitting	0.54	2.97	2.17
Prediction	0.08	0.39	0.29
Total	42.67	45.41	44.51

Table 5.3: Runtimes for the V_0 estimation in seconds.

The runtimes do not differ significantly and in total all models need about 45

seconds. Remember, that the traditional MC simulation took 525 seconds, which is more than ten times as long.

When we consider the results of the predicted V_0 in Table 5.4, we see that the estimates are very similar. The fair value that was obtained by the traditional MC simulation is given by $V_0 = 245\,404\,851$. Hence, for all models the deviation from the MC value is quite small, particularly for the prediction via the neural net. Of course, the MC result is also just an estimate of the true today's value.
It is indeed surprising that the tree model does not underestimate V_0. Apparently, among the selected representatives there is at least one contract with a very high response value, so that the model works well without extrapolation.

Model	GLM	Random forest	Neural net
V_0 estimate	251 256 473	250 031 683	245 122 825
Relative error wrt. MC value	2.384 %	1.885 %	-0.115 %

Table 5.4: Estimated V_0 and relative error.

It can be seen as a success that within such a short time a value, whose actual determination is very complex, can be predicted with such a good quality.

The next step is to create the empirical distribution of V_1. Therefore, we first need to generate real-world realizations of the stock value at time 1. The remaining steps are similar as before, see Table 5.5. Now, we have 10 million combinations of contracts and scenarios, which must be predicted via the fitted models. The GLM is very fast and even the neural network, in spite of the required scaling, can cope well with the prediction of this large amount of data. Only the random forest needs roughly four minutes for the prediction. However, considering that with the nested MC simulation it was not even possible to determine the fair values at time 1 given multiple scenarios for such a large portfolio, a total runtime of a bit more than five minutes is still very fast.

Ultimately, the one-year losses $\Delta^{(\ell)}$ are calculated very quickly as linear transformation of $V_1^{(\ell)}$. The resulting empirical distribution is shown on the left side of Figure 5.36. A negative loss naturally corresponds to a profit.
We see three different shapes of the distribution: The GLM yields a symmetric loss distribution with both few small and few large losses. In contrast, the losses of the random forest are nearly equally distributed until a loss of approximatly

Model	GLM	Random forest	Neural net
Sample scenarios	0.27	0.27	0.27
Select representatives	0.28	0.28	0.28
MC simulation	57.92	57.92	57.92
Model fitting	0.90	3.84	2.83
Prediction	7.05	263.25	24.77
Total	66.42	325.56	86.07

Table 5.5: Runtimes for the estimation of all $V_1^{(\ell)}$ in seconds.

`x = 2e+07`. Above this point there are only very few observations spread over a large range. Lastly, the neural net seems to produce a smoothed version of the losses from the random forest. All three models have in common that there are very few extremely high losses and that the support of the distributions is about the same.

From the empirical loss distribution the SCR is estimated as 99.5%-quantile. That is, given $N = 1000$ scenarios, the SCR equals the 995-th element of the losses sorted in ascending order.
In Figure 5.36, the estimated SCR is marked with a red dot. The numerical results are depicted in Table 5.6. The outcomes of the individual methods are all on the same scale, but differ a bit. The GLM predicts the lowest SCR, the neural net the highest one.

Model	GLM	Random forest	Neural net
SCR	49 335 616	84 619 970	102 560 755

Table 5.6: Estimated SCR from machine learning methods

As the estimates of the today's fair values V_0 of all three models are approximately equivalent, the difference in the SCR must arise from the prediction of $V_1^{(\ell)}$. On the right hand side of Figure 5.36, the estimated time-1 values are plotted versus their stock scenarios at time 1. In all models we see a decreasing curve, i.e. low scenarios lead to large fair values and vice versa.

The GLM shows almost a straight line for the dependence of V_1 on the given scenarios. The structure of the GLM allows for a direct interpretation of the impact of the stock scenarios. If we know that, for instance, the predicted fair value for scenario $\tilde{S}_1 = 1.0326392$ is given by $\tilde{V}_1 = 266675233$, then the value for any other

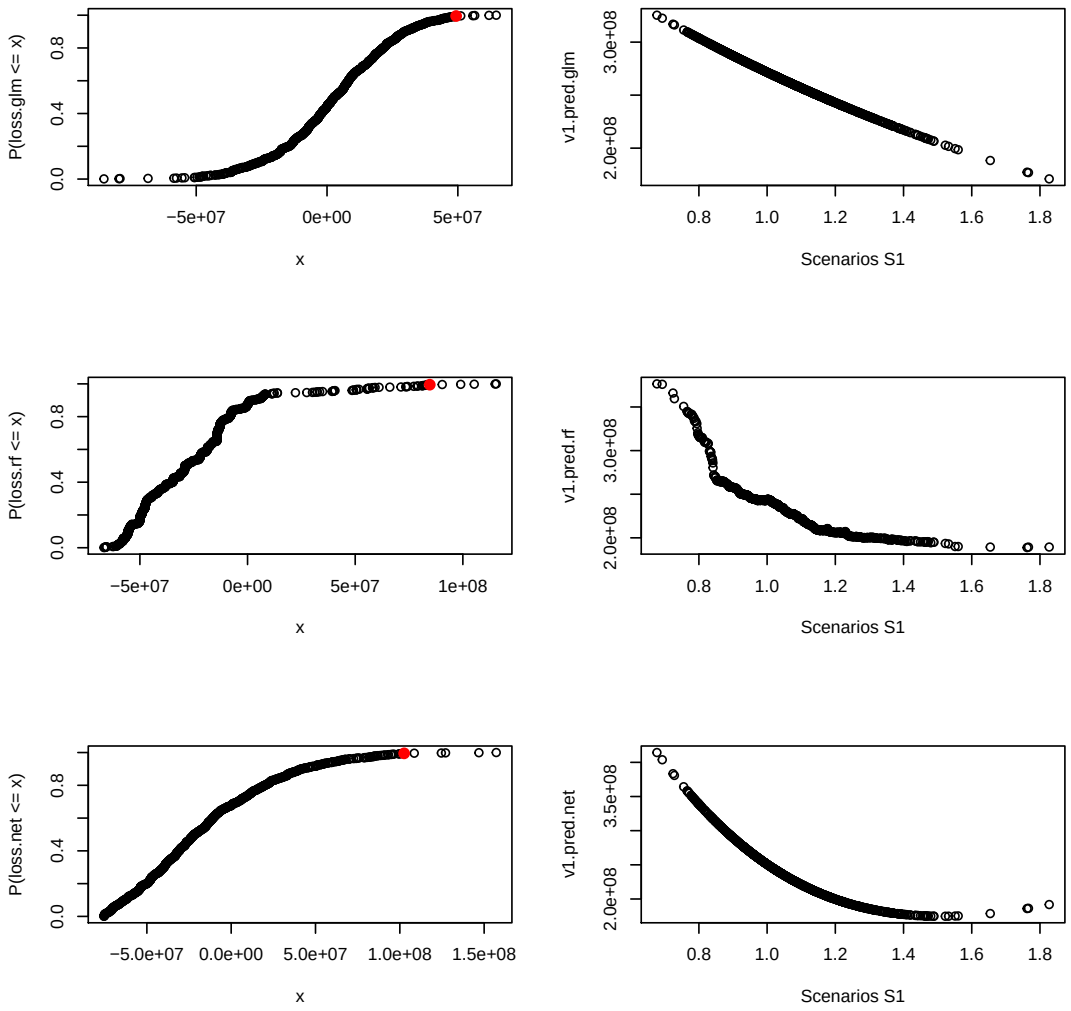


Figure 5.36: (left) Empirical loss distribution.
(right) Predicted time-1 values vs the given stock scenarios at time 1 for each model.

scenario S_1 can be obtained by $V_1 = \tilde{V}_1 \cdot \exp(\hat{\beta}_{S_1}(S_1 - \tilde{S}_1))$, where the stock parameter $\hat{\beta}_{S_1}$ equals -0.5595 in the fitted GLM. This holds due to the assumed model relationship with a log link, and because we only consider the fair values aggregated over all contracts. That means that we have a log-linear relationship between the fair value and the scenario values, expressed by $\log(V_1) = \log(\tilde{V}_1) + \hat{\beta}_{S_1}(S_1 - \tilde{S}_1)$. It is important to understand that when we apply a GLM with a log link function, we assume exactly this relationship in this context. This expression might

be useful, but at the same time it is a very strong assumption and it is doubtful whether this relation is valid. Hence, the model is not very flexible and eventually fails to describe the shape of V_1 and hence also of the loss distribution.
Note that the remaining model parameters, which specify the impact of the contract attributes, have no influence on the behavior of the aggregated fair values V_1, but only serve to determine the base level, i.e. can be considered as intercept.

Random forests are non-parametric algorithms and are based on simple yes-or-no rules that create non-smooth response regions. Hence, by construction of the model, the impact of the stock scenario at time 1 can only be captured by splitting a set of observations into two subgroups, where a certain scenario value is used as a cut point. Aggregating over multiple unpruned trees helps to smooth out the prediction surface. The performance of trees is limited because they cannot extrapolate. However, since they do not have any parametric assumptions, they are flexible and are able to reflect even complex relationships.
Figure 5.36 shows that the predicted time-1 values decrease rapidly at the beginning and then at a scenario value of approximatly 0.85 a kink appears. Presumably, in multiple trees of the ensemble an important split is made with this scenario value as cutpoint.

Neural nets are parametric models, but because of multiple hidden neurons they are much more flexible than GLMs and allow for a more complex relationship between scenario values and responses. When we now consider the behavior of the time-1 values produced by the neural network, we see that the the curve is actually a smoothed version of the random forest. We know that within tree models no assumptions are taken on any relation, and therefore they are free to model the fair value. So if we see a similar shape for the neural net, this suggests that the net can reproduce the true behavior of V_1 well, at least for the set of representatives. Additionally, the neural net has the ability to extrapolate the responses into areas outside the values seen within the representatives.

Since no reference SCR is available from the nested MC simulation for the entire portfolio, it is difficult to judge which of the three estimated SCRs clearly gives the best prediction. In the previous Section 5.3.6, we have seen that the neural network is very promising and yields the smallest test errors. Since we use exactly the same models, we can expect that this still applies.
Further, in Figure 5.37, we compare the MC values V_1 of the representatives, which

serve as input for the training, with the fitted values of the models. Note that the these values are no aggregated fair values, but the ones arising from one sample of a single contract and a single scenario.

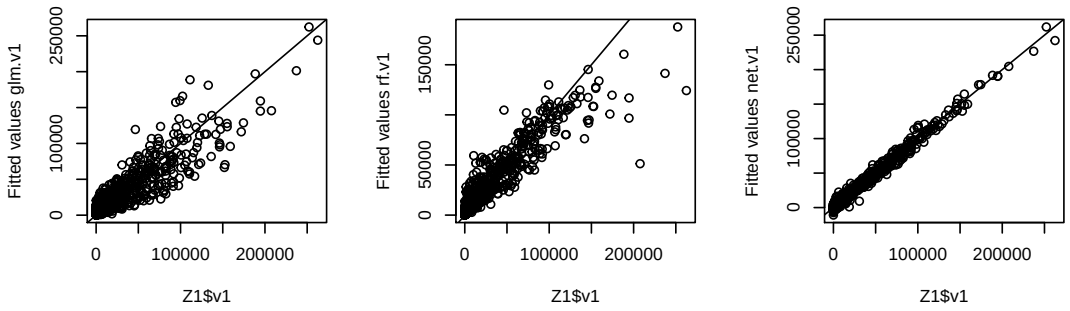


Figure 5.37: Fitted fair values vs. MC values of the representatives.

We see that the neural net provides the best fit on the set of representatives. An overfit of the data is not expected since the neural network is regularized by weight decay and the model also achieves a good fit on the test data in Section 5.3.6.

In summary, it seems that under this framework the neural network offers the best chance to estimate the SCR as well as possible.

6 Conclusion

Variable annuities are insurance products that contain complex guarantees. The fair market value of the guaranteed benefits cannot generally be determined in a closed form and hence insurance companies often rely on Monte Carlo simulation. However, MC simulation is computationally expensive when applied to a large portfolio of variable annuity contracts. An approach to address this runtime-related problem involves machine learning techniques and makes the calculation more efficient by reducing the number of required MC simulations.
In this thesis, we empirically investigated the performance of three machine learning methods on a synthetic dataset with respect to estimating the today's fair value and determining the SCR.

Tree-based methods are considered as non-parametric models and they can be applied to fit complex non-linear problems. However, tree models are unable to discover trends and hence the models fail to perform extrapolation of the response value outside the range of the training data. Here, trees tend to underestimate values when applied to extrapolation domains.
The generalized linear models are parametric models that assume a transformed linear relationship. This allows for an intuitive interpretation of the relationship, but at the same time makes the model quite inflexible. When calculating the time-1 values for different scenarios, it seems that the GLM cannot capture the behavior of the distribution with respect to the stock values correctly. In future research, one might consider transformations of explanatory variables or include interaction terms between multiple variables in order to possibly improve the model.
The neural net seems to be a very promising technique in this context. Due to its many parameters the model is more flexible than a GLM and is also able to perform extrapolation. Overfitting can be prevented by applying weight decay for regularization. Depending on the random initialization of the parameters, the training process could converge in different local maxima. To make the model

more stable, ensemble techniques could be incorporated in future.
In addition, we have seen that the data adjustments make the interpretation of the product influence difficult within parametric models. Alternatively, one could split the portfolio into four sets by the guarantee and build a separate model for each part. This would avoid the interpretation issue and might even provide a better prediction accuracy since the influence of attributes can be specified differently for distinct guarantees.

In conclusion, the numerical results have demonstrated that in terms of predictive accuracy the neural net performs best among the methods considered in this context. With a sufficient number of neurons, it is capable of very precisely predicting the behavior of the today's values and the values at time 1. All methods were able to drastically reduce the runtime.

Within this thesis many assumptions and simplifications are made to create a feasible valuation framework and therefore the results are only valid within this context. The projection onto realistic circumstances and real data sets must be carried out very carefully.

Bibliography

Bauer, Daniel, Alexander Kling, and Jochen Russ (2008). "A universal pricing framework for guaranteed minimum benefits in variable annuities". In: *ASTIN Bulletin: The Journal of the IAA* 38(2), pp. 621–651.

Bauer, Daniel, Andreas Reuss, and Daniela Singer (2012). "On the calculation of the solvency capital requirement based on nested simulations". In: *ASTIN Bulletin: The Journal of the IAA* 42(2), pp. 453–499.

Breiman, Leo (1996). "Bagging predictors". In: *Machine learning* 24(2), pp. 123–140.

Breiman, Leo (2001). "Random forests". In: *Machine learning* 45(1), pp. 5–32.

Breiman, Leo et al. (1984). *Classification and regression trees*. CRC press.

Dunn, Peter K and Gordon K Smyth (2018). *Generalized linear models with examples in R*. Springer.

Gan, Guojun (2013). "Application of data clustering and machine learning in variable annuity valuation". In: *Insurance: Mathematics and Economics* 53(3), pp. 795–801.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Hejazi, Seyed Amir and Kenneth R Jackson (2017). "Efficient valuation of SCR via a neural network approach". In: *Journal of Computational and Applied Mathematics* 313, pp. 427–439.

James, Gareth et al. (2013). *An introduction to statistical learning*. Vol. 112. Springer.

Kuhn, Max, Kjell Johnson, et al. (2013). *Applied predictive modeling*. Vol. 26. Springer.

Loeppky, Jason L, Jerome Sacks, and William J Welch (2009). "Choosing the sample size of a computer experiment: A practical guide". In: *Technometrics* 51(4), pp. 366–376.

McCullagh, Peter and John A Nelder (1989). *Generalized linear models*. 2nd ed. Vol. 37. Chapman and Hall, London.

Milevsky, Moshe A and Thomas S Salisbury (2006). "Financial valuation of guaranteed minimum withdrawal benefits". In: *Insurance: Mathematics and Economics* 38(1), pp. 21–38.

Name: Stefanie Burkart Matrikelnummer: 868233

Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben wird.

Ulm, den 02.11.2020 S. Burkart

Stefanie Burkart