DEGREE PROJECT IN MATHEMATICS,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Claims Reserving using Gradient Boosting and Generalized Linear Models

MARCUS AHLGREN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ENGINEERING SCIENCES

Claims Reserving using Gradient Boosting and Generalized Linear Models

MARCUS AHLGREN

Degree Projects in Mathematical Statistics (30 ECTS credits)
Degree Programme in Applied and Computational Mathematics (120 credits)
KTH Royal Institute of Technology year 2018
Supervisor at Länsförsäkringar AB: Kathrin Vorwerk
Supervisor at KTH: Håkan Andersson
Examiner at KTH: Boualem Djehiche

TRITA-SCI-GRU 2018:234
MAT-E 2018:42

Royal Institute of Technology
School of Engineering Sciences
KTH SCI
SE-100 44 Stockholm, Sweden
URL: www.kth.se/sci

Claims Reserving using Gradient Boosting and Generalized Linear Models

Abstract

One fundamental function of an insurance company revolves around calculating the expected claims costs for which the insurer has to compensate its policyholders for. This is the process of claims reserving which is practised by actuaries using statistical methods. Over the last few decades statistical learning methods have become increasingly popular due to their ability to find complex patterns in any type of data. However, they have not been widely adapted within the insurance sector. In this thesis we evaluate the capability of claims reserving with the method of gradient boosting, a non-parametric statistical learning method that has proven to be successful within multiple other disciplines which has made it very popular. The gradient boosting technique is compared with the generalized linear model(GLM) which is widely used for modelling claims. We compare the models by using a claims data set provided by Länsförsäkringar AB which allows us to train the models and evaluate their performance on data not yet seen by the models. The models were implemented using R. The results show that the GLM has a lower prediction error. Also, the gradient boosting method requires more fine tuning to handle claims data properly while the GLM already possesses certain features that makes it suitable for claims reserving without making as many adjustments in the model implementation. The advantage of capturing complex dependencies in data is not fully utilized in this thesis since we only work with 6 predictor variables. It is more likely that gradient boosting can compete with GLM when predicting more complicated claims.

Reservsättning med Gradient Boosting och Generaliserade Linjära Modeller

Sammanfattning

En av de centrala verksamheterna ett försäkringsbolag arbetar med handlar om att uppskatta skadekostnader för att kunna ersätta försäkringstagarna. Denna procedur kallas reservsättning och utförs av aktuarier med hjälp av statistiska metoder. Under de senaste årtiondena har statistiska inlärningsmetoder blivit mer och mer populära tack vare deras förmåga att hitta komplexa mönster i alla typer av data. Dock har intresset för dessa varit relativt lågt inom försäkringsbranschen till förmån för mer traditionella försäkringsmatematiska metoder. I den här masteruppsatsen undersöker vi förmågan att reservsätta med metoden *gradient boosting*, en icke-parametrisk statistisk inlärningsmetod som har visat sig fungera mycket väl inom en rad andra områden vilket har gjort metoden mycket populär. Vi jämför denna metod med generaliserade linjära modeller(GLM) som är en av de vanliga metoderna vid reservsättning. Vi jämför modellerna med hjälp av ett dataset tillhandahålls av Länsförsäkringar AB. Modellerna implementerades med R. 80% av detta dataset används för att träna modellerna och resterande 20% används för att evaluera modellernas prediktionsförmåga på okänd data. Resultaten visar att GLM har ett lägre prediktionsfel. Gradient boosting kräver att ett antal hyperparametrar justeras manuellt för att få en välfungerande modell medan GLM inte kräver lika mycket korrigeringar varför den är mer praktiskt lämpad. Fördelen med att kunna modellerna komplexa förhållanden i data utnyttjas inte till fullo i denna uppsats då vi endast arbetar med sex prediktionsvariabler. Det är sannolikt att gradient boosting skulle ge bättre resultat med mer komplicerade datastrukturer.

Acknowledgements

I would like to express my gratitude to my supervisor at Länsförsäkringar AB, Kathrin Vorwerk, for guiding me through the project. I would also like to thank Håkan Andersson, my supervisor at KTH, for continuous valuable feedback.

Contents

1 Introduction

The interest in statistical learning methods has grown immensely over the last few years due to a combination of a drastic increase in both data availability and computing power. The insurance sector is fundamentally reliant on statistical methods and the vast amount of available data gives actuaries a great opportunity to take advantage of statistical learning methods. Actuaries have several different responsibilities. One central task for actuaries is *claims reserving*. This originates from the fact that the insurer must have sufficient capital to cover claims costs. The claims are usually divided into two groups, incurred but not yet reported claims(IBNR) and incurred claims that have been reported but are not yet settled(RBNS). For a given date the total loss reserve refers to the amount that is needed to settle all costs for claims that have already incurred i.e. both IBNR and RBNS claims. These are handled separately by the insurer and in this thesis we will only consider the latter. For RBNS claims there is a delay between the report date and the settle date. Thus, the insurer must make an estimate of the future claim costs in order to setup a capital reserve to cover these liabilities. In this thesis we will restrict the modelling to RBNS reserves.

In order to ensure that insurance companies can cover their liabilities there are certain requirements that need to be fulfilled. Within EU insurers must follow the Solvency 2 directive. This directive can be categorized into three different pillars and the capital reserves requirements goes under Pillar 1. More specifically, the solvency capital requirement(SCR) requires the insurers to reserve capital such that the probability of not being able to cover their losses is less than 0.5%. It is therefore essential to have well-functioning methods for claims reserving.

Since the cost of a claim at settlement is not deterministic one needs to estimate this amount in order to reserve a sufficient amount of capital. This is typically done using a set of relevant predictor variables together with a suitable model. A traditional method used for modelling claims is the *generalized linear model(GLM)*, a generalization of linear regression. GLM requires certain assumptions of the variable dependencies to be fulfilled in order to provide a valid model. Therefore, it is vital to have a good understanding of the data structure. As the complexity and dimensionality of the data grows this task becomes increasingly difficult. In this thesis we compare claims reserving with GLM and a statistical learning method, *decision trees*. Decision trees has its main advantage in interpretability but in its simplest form the model performance can be rather limited. However, in combination with ensemble methods such as *random forests* and *gradient boosting* the performance can be improved remarkably. Additionally, decision trees require no prior knowledge of the data structure.

This thesis is written for Länsförsäkringar AB who also provided the data set used to analyze the aforementioned models. Due to integrity some of the information in this data set can not be disclosed.

1.1 Previous Work

There has been several studies where statistical learning methods such as decision trees have been compared to GLM in claims modelling. In 2017 the study *Case Reserving in Non-Life Practice using Individual Data and Machine Learning* was conducted[2]. It tested the ability of several statistical learning methods including gradient boosting and random forests. The study was performed using vehicle claims. The results suggest that both tree based methods have great potential compared to more traditional methods such as GLM, especially when it comes to find intricate dependencies in the data. This is crucial to predict rare claims such as very severe claims. However, for regular claims the GLM performed better and thus performed better overall. It is therefore emphasized that statistical learning methods are not always better because they are more flexible. For data which fulfills the assumptions for traditional regression these are often superior. They are also more convenient to implement since they don't need as much tuning. More flexible methods such as decision trees should therefore be used in cases where the data set is very large and the dependencies are complex. This is where decision trees shine as they can naturally handle interactions between predictor variables, something that has to be implemented manually with GLMs.

1.2 Disposition

In section 2 we will begin with introducing some basic concepts within statistical learning theory to give a better understanding of what factors need to be taken into consideration when working with statistical models using any type of data. We first introduce GLM and then, in section 3 move on to decision trees and tree-based ensemble methods such as bagging, random forests and gradient boosting. In section 4 we present the the data set and the methodology. Finally, we present the results in section 5. A summary concludes the thesis.

2 Theory

2.1 Statistical Learning Theory

The framework of statistical learning has developed in order to build models and algorithms based on data in order to make predictions. A data point $(\boldsymbol{X}, Y) \in \mathbb{R}^p \times \mathbb{R}$ consists of the random independent variables $X_1, ..., X_p$ and the dependent variable Y. The independent variables are also commonly referred to as *predictors* or *features* while the dependent variable is referred to as *response variable.* Statistical learning can be divided into two main branches, supervised learning and unsupervised learning. The difference is that for the latter the dependent variable associated with a data point is not known. Only supervised learning methods will be used in this thesis why we from now on only consider labeled data i.e. data where the response variable is known. It is assumed that there is a relationship between the *predictors* and the *response* which we write as:

$$Y = f(\boldsymbol{X}) + \epsilon. \tag{1}$$

The function f is deterministic but unknown and the aim of supervised learning methods is to estimate this function. ϵ is an error term s.t.

$$E[\epsilon] = 0, \quad Var(\epsilon) = \sigma^2.$$

ϵ and $\boldsymbol{X}$ are also assumed to be independent. The error term can be interpreted as random noise that accounts for the effects on Y which are not captured by the predictors.

The data points used to estimate f is comprised in a training set $\boldsymbol{\mathcal{L}} = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^n$. Thus, the problem boils down to finding a function $\hat{f}(\boldsymbol{x}, \mathcal{L})$ that approximates $f(\boldsymbol{X})$. Random variables such as $\boldsymbol{X}$ will be denoted in upper case while realizations of random variables will be written in lower case. Approximations of functions f will furthermore be denoted as $\hat{f}$. For notational convenience the $\mathcal{L}$ can be dropped as $\hat{f}$ is a function of the training set by implication.

2.2 Model Selection

In order to decide which model to use one needs to take different factors into account such as accuracy and interpretability. These will be discussed in this section.

2.2.1 Loss Function

To choose a model that is well suited for prediction it is central to evaluate the predictive ability of the model. This leads us to introduce a *loss function.* The loss function $L(y, \hat{f}(\boldsymbol{x}))$ quantifies the prediction error i.e. the discrepancy between the prediction $\hat{f}(\boldsymbol{x})$ and the actual value y. The loss functions could in theory be of any form but they're commonly restricted to certain features. Typically the loss function is real-valued(often restricted to the non-negative reals) and convex. It is designed to capture that higher discrepancy leads to higher loss. The two arguably most common loss functions for regression and classification are respectively:

$$L(y, \hat{f}(\boldsymbol{x})) = (y - \hat{f}(\boldsymbol{x}))^2 \tag{2}$$

$$L(y, \hat{f}(\boldsymbol{x})) = \begin{cases} 1 & y \neq \hat{f}(\boldsymbol{x}) \\ 0 & y = \hat{f}(\boldsymbol{x}). \end{cases}$$

2.2.2 Risk Function

The ultimate goal in supervised learning is to find a model that is trained on past observations i.e training data and is able to generalize and make accurate predictions on previously unseen data. That is, we want to minimize the expected loss with respect to $\hat{f} \in \mathcal{F}$ where $\mathcal{F}$ is a class of functions, often referred to as the *function space.* We call the expected loss the *risk function*[11](or simply *risk*) and define it and its minimizer $\hat{f}^*$ as:

$$\begin{aligned} R(\hat{f}) &= E[L(Y, \hat{f}(\boldsymbol{X}))] \\ \hat{f}^* &= \underset{\hat{f} \in \mathcal{F}}{\arg\min}\, R(\hat{f}(\boldsymbol{X})) \end{aligned} \tag{3}$$

for some loss function $L(Y, \hat{f}(\boldsymbol{X}))$.

The risk measures how much loss we can expect on average but since the distribution of Y is unknown neither $R(\hat{f})$ nor $\hat{f}^*$ can be directly computed. However, at our hand we have a training set $\mathcal{L} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ of n i.i.d samples from Y. Thus, we can compute the *empirical risk* $\hat{R}_n(\hat{f})$ which is defined by:

$$R_n(\hat{f}(\boldsymbol{x})) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, \hat{f}_n(\boldsymbol{x_i})). \tag{4}$$

All quantities involved are known why we can compute the minimizer $\hat{f}^*$ defined as:

$$\hat{f}_n^* = \arg\min_{\hat{f}\in\mathcal{F}} R_n(\hat{f}). \tag{5}$$

It remains to establish how the empirical risk and true risk are related. Since $\hat{f}^*$ and $\hat{f}_n^*$ are minimizers of R and R_n respectively it must hold that:

$$\begin{aligned} R(\hat{f}_n^*) - R(\hat{f}^*) \geq& 0 \\ R_n(\hat{f}^*) - R_n(\hat{f}_n^*) \geq& 0. \end{aligned} \tag{6}$$

If we add these inequalities together we get:

$$\begin{aligned} 0 \leq& R(\hat{f}_n^*) - R(\hat{f}^*) + R_n(\hat{f}^*) - R_n(\hat{f}_n^*) \\ =& R(\hat{f}_n^*) - R_n(\hat{f}_n^*) + R_n(\hat{f}^*) - R(\hat{f}^*) \\ \leq& \sup_{\hat{f}\in\mathcal{F}}(R(\hat{f}) - R_n(\hat{f}) + R_n(\hat{f}^*) - R(\hat{f}^*)) \end{aligned} \tag{7}$$

From the law of large numbers it follows that for any fixed function $\hat{f}$:

$$R_n(\hat{f}) \xrightarrow{P} R(\hat{f}) \quad \text{as} \quad n \to \infty. \tag{8}$$

Pick $\hat{f}$ as $\hat{f}^*$ and we note that the second half of the last line of (7) will converge to 0 in probability. Furthermore, it would be neat if we also could show that the empirical risk converges uniformly to the true risk i.e.:

$$\sup_{\hat{f}\in\mathcal{F}}(R(\hat{f}) - R_n(\hat{f})) \xrightarrow{P} 0 \quad \text{as} \quad n \to \infty \tag{9}$$

since if this holds true it would imply that the left hand sides of (6) also converges to zero which would motivate the use of $R_n(\hat{f})$ and $\hat{f}_n^*$ as replacements for $R(\hat{f})$

and $\hat{f}^*$. It turns out that this holds true for certain classes of $\mathcal{F}$, decision trees being one of these. This was shown by Vapnik[12] in 1982.

One might ask why it's necessary to restrict the optimization of the risk w.r.t a certain class of functions. Let us remove this condition and allow all possible functions to be fitted to the training data. Then we could in theory choose a model that fits the training data in every single point and we would get empirical error equal to zero. However, this is not ideal since it implies that the model captures the noise in the data and is thus too sensitive. This will likely cause the model to not predict new unseen data very well. The phenomenon of low error on training data but high error on test data is called *overfitting*. Avoiding overfitting is essential when working with statistical learning methods. Therefore, one should restrict the minimization of the risk to a class of functions $\mathcal{F}$ that can generalize to unseen data. Some examples of such methods are *support vector machines(SVM)*, *neural networks* and *decision trees*[12].

2.2.3 Cross-Validation

When working with statistical learning methods one should separate the data into training set, test set and validation set in order to not evaluate the model on the same data as was used in the learning process. This is usually done by simply setting aside a piece of the available data for model evaluation. This is called the *holdout method*. However, sometimes the available data is limited why it can be problematic to both have enough data for training and evaluating the model. This is where the method of *cross-validation* can be used. An alternative to the *holdout method* is to repeatedly partition the observations into *training sets* and *test sets* and compute the model accuracy for each partition, thus, *cross-validating* the model performance. This approach is especially suitable when the number of observations is low. Validation and test sets are similar in the sense that they are not used to fit the model. However, validation sets are used to evaluate model performance in order to tweak the model. This could for instance be tuning of hyperparameters such as the depth in a decision tree. Test sets however are used for assessing performance of a final model that will not be altered.

As it is often not computationally reasonable to consider all possible partitions some partitioning procedures have become more popular than others. One of them is *k-fold cross-validation* where the data is randomly divided into k different sets of equal size, $\{\mathcal{D}\}_{i=1}^{k}$. For a given iteration, i, $\mathcal{D}_i$ is set aside as test set while the remaining observations are used as training set.

The model is then fitted on the training set and the empirical risk is then evaluated on the i:th set and can thus be written as:

$$R^{(i)}(\hat{f}_i(\boldsymbol{x})) = \frac{1}{|\mathcal{D}_i|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}_i} L(y, \hat{f}_i(\boldsymbol{x})) \tag{10}$$

where $|\mathcal{D}_i|$ denotes the number of elements in the i:th set. Below we see a an illustration of how the observations are partitioned into different formations of training and test sets for 3-fold cross-validation, each iteration yielding a different model and a different loss calculated on the test set.

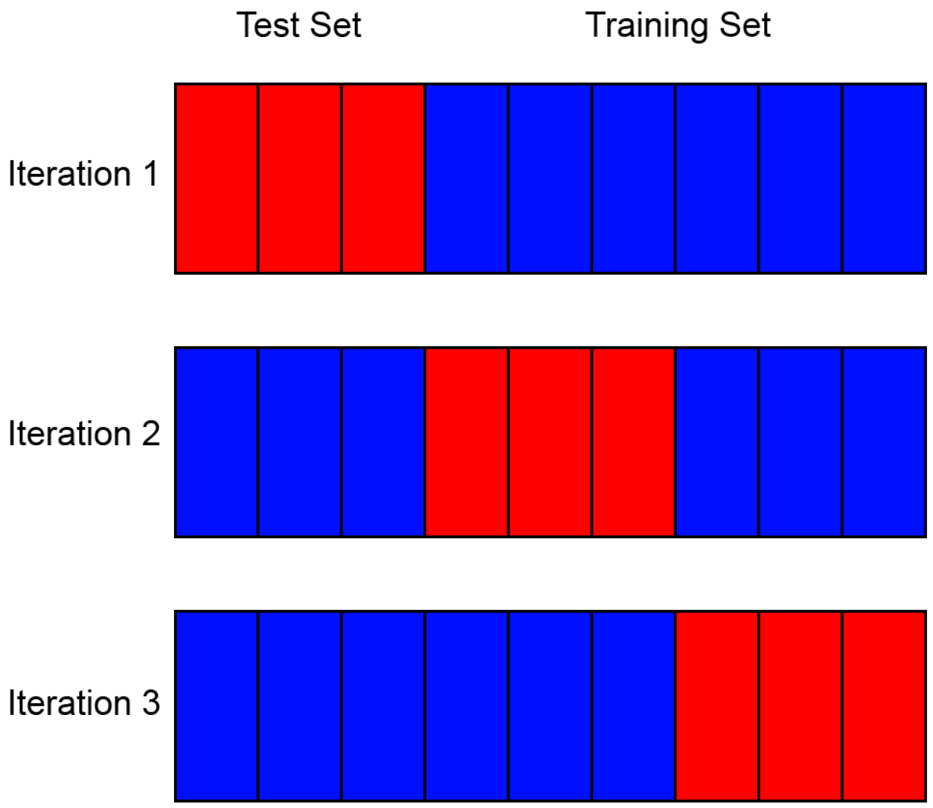

Figure 1: Example of 3-fold cross-validation with 9 observations.

Often the squared error is used as loss function. Once the empirical risk has been evaluated for all k sets we can finally compute the cross-validation error defined as:

$$CV_k = \frac{1}{k}\sum_{i=1}^{k} R^{(i)}(\hat{f}_i(\boldsymbol{x})). \tag{11}$$

The special case where $k = n$ is often used and has been given its own name, *leave-one-out-cross-validation*(LOOCV).

2.2.4 Bias-Variance Tradeoff

When choosing a model one adjusts the model to fit the training data. However, the ultimate goal is to find a model that generalizes to perform well on unseen data. In other words, we seek models which yield small loss. Let us here consider the most common loss function *squared loss*, applied on a fixed set $\mathbf{X}$ The expected loss the model will yield can be decomposed into one reducible part and one irreducible part. This decomposition can be written as[9]:

$$E[(Y - \hat{f}(\boldsymbol{X}))^2] = \underbrace{Bias(\hat{f}(\boldsymbol{X}))^2 + Var(\hat{f}(\boldsymbol{X}))}_{\text{reducible error}} + \underbrace{\sigma^2}_{\text{irreducible error}} \tag{12}$$

where

$$\begin{aligned} Bias(\hat{f}(\boldsymbol{X})) &= E[\hat{f}(\boldsymbol{X}) - f(\boldsymbol{X})] \\ Var(\hat{f}(\boldsymbol{X})) &= E[\hat{f}(\boldsymbol{X})^2] - E[\hat{f}(\boldsymbol{X})]^2. \end{aligned}$$

Proof.

$$\begin{aligned}
E[(Y-\hat{f}(\boldsymbol{X}))^2] &= E[Y^2 - 2Y\hat{f}(\boldsymbol{X}) + \hat{f}(\boldsymbol{X})^2] \\
&= E[Y^2] - 2E[Y\hat{f}(\boldsymbol{X})] + E[\hat{f}(\boldsymbol{X})^2] \\
&= E[Y^2] - 2E[Y]E[\hat{f}(\boldsymbol{X})] + E[\hat{f}(\boldsymbol{X})^2] \\
&= Var(Y) + E[Y]^2 - 2E[f(\boldsymbol{X})+\epsilon]E[\hat{f}(\boldsymbol{X})] + E[\hat{f}(\boldsymbol{X})^2] \\
&= \sigma^2 + E[f(\boldsymbol{X})]^2 - 2f(\boldsymbol{X})E[\hat{f}(\boldsymbol{X})] + E[\hat{f}(\boldsymbol{X})^2] \\
&= \sigma^2 + f(\boldsymbol{X})^2 - 2f(\boldsymbol{X})E[\hat{f}(\boldsymbol{X})] + E[\hat{f}(\boldsymbol{X})^2] - E[\hat{f}(\boldsymbol{X})]^2 + E[\hat{f}(\boldsymbol{X})]^2 \\
&= \sigma^2 + (f(\boldsymbol{X}) - E[\hat{f}(\boldsymbol{X})])^2 + E[\hat{f}(\boldsymbol{X})^2] - E[\hat{f}(\boldsymbol{X})]^2 \\
&= \sigma^2 + Bias(\hat{f}(\boldsymbol{X}))^2 + Var(\hat{f}(\boldsymbol{X}))
\end{aligned}$$

where we used the facts that Y and $\hat{f}(\boldsymbol{X})$ are independent and $E[f(\boldsymbol{X})] = f(\boldsymbol{X})$ since $f(\boldsymbol{X})$ is deterministic. $\square$

From equation (12) it is seen that the model should be chosen such that the bias and variance in the model is minimized. Often it is not possible to minimize them simultaneously and we refer to this issue as the bias-variance tradeoff. The error term σ^2 can not be reduced since it arises from noise in the data which the model doesn't account for. The error due to *bias* arises when one uses a simple model that is not able to capture more complex structures in the data. Approximating a non-linear function with linear regression exemplifies this.

The expected error due to *variance* stems from the fact the model yields different predictions based on what training data was used in the learning process. This error can be minimized by choosing a robust model that doesn't change much when using different training sets. The dilemma when minimizing the expected test error is that generally a flexible model with low *bias* tends to yield a higher error due to *variance* and vice versa. Therefore, the tradeoff between low *variance* and low *bias* must be considered when selecting the model.

2.3 Generalized Linear Models

Generalized linear models(GLM) are an extension from the classically used linear models. Recall that in the scope of linear regression we use a model of the form:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon} \tag{13}$$

where the dependent variable $\boldsymbol{Y}$ is an $n \times 1$ vector, $\boldsymbol{X}$ is an $n \times p$ matrix and $\boldsymbol{\beta}$ is a $p \times 1$ vector. Furthermore, it is assumed that the mean of the error term $\boldsymbol{\epsilon}$

is zero and the components of $\boldsymbol{Y}$ are independent normally distributed random variables with equal variance i.e.

$$\begin{aligned} Y_i &= N(\mu_i, \sigma^2) \forall i \\ E[\boldsymbol{Y}] &= \boldsymbol{\mu} = \boldsymbol{X\beta} \end{aligned}$$

Let us now introduce a new variable $\boldsymbol{X\beta} = \boldsymbol{\eta}$ which is called the *linear predictor* of the model[7]. For the classical linear model the relationship between the linear prediction and the mean is modeled as:

$$\boldsymbol{\mu} = \boldsymbol{\eta}.$$

GLM generalizes the linear model by allowing a more complex relationship between μ and η. We introduce the *link function* $g(\cdot)$ that links them together through:

$$\eta_i = g(\mu_i) \tag{14}$$

g may be any monotone and differentiable function. The linear model in (13) for instance uses the identity link. Moreover, GLM allows the components of $\boldsymbol{Y}$ to take on more distributions than just the normal distribution. We now allow any probability distribution of the form:

$$f_{Y_i}(y_i; \theta_i, \phi) = exp\left(\frac{y_i \cdot \theta_i - b(\theta_i)}{a(\phi)} + c(y_i, \phi)\right) \tag{15}$$

for some functions a, b and c and parameters θ, ϕ. θ is specifically called the *canonical parameter*. The mean and variance related to this distribution are:

$$\begin{aligned} E[Y_i] &= b'(\theta_i) = \mu_i \\ Var(Y_i) &= b''(\theta_i)a(\phi) \end{aligned} \tag{16}$$

where $b''(\theta_i) = \mu' = V(\mu)$ is called the *variance function.* This function plays an important role which is summarized[6] in the following theorem:

Theorem 2.1. *Within the class of distributions of the form (15), the distribution is uniquely determined by its variance function.*

Thus, characterizing model is equivalent to selecting the variance function. For instance, $V(\mu) = 1$ yields the normal distribution.

Different choices of link functions can be made but some are more suitable than others. Often times the *canonical link* is used which is defined by:

$$
\begin{aligned}
g(\mu_i) &= b'(\theta_i)^{-1} \\
\Rightarrow g(\mu_i) &= \theta_i = \sum_{i=1}^{p} x_{ij}\beta_i.
\end{aligned}
$$

This choice of link function ensures that μ_i stays within the range of the outcome variable y_i[10].

2.3.1 Tweedie Distribution

The Tweedie distributions are a family of distributions which are characterized by the variance function:

$$V(\mu) = \mu^p. \tag{17}$$

This family of distributions include distributions such as the normal distribution($p = 0$), Poisson distribution($p = 1$) and Gamma distribution($p = 2$)[6]. The Tweedie distributions are commonly used in the context of insurance claims since they have proven to provide good fits for certain choices of p.

2.4 Maximum Likelihood Estimation

The method of maximum likelihood estimation is a method used to choose a model from a family of distributions $f(\boldsymbol{x}|\boldsymbol{\theta}), \quad \boldsymbol{\theta} \in \boldsymbol{\Theta}$ by maximizing the likelihood function[1]. Given a set of i.i.d observations $\{\boldsymbol{x}_i\}_{i=1}^n$ we define the likelihood function as:

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} f(\boldsymbol{x}_i|\boldsymbol{\theta}). \tag{18}$$

Furthermore, we define the *maximum likelihood estimate(MLE)* as:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\ \in \boldsymbol{\Theta}}{\arg\max}\, L(\boldsymbol{\theta}). \tag{19}$$

Maximum likelihood estimation can for instance be used with GLMs to derive the parameter values.

2.5 Regularization

Regularization is a technique used to prevent overfitting by penalizing the loss function with a shrinking quantity. The principle is the same for all loss functions but we use the maximum likelihood described above for demonstration purposes. The regularized objective function would be written as:

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} f(\boldsymbol{x}_i|\boldsymbol{\theta}) + \lambda \|\boldsymbol{\theta}\|_p \tag{20}$$

where λ is the shrinking parameter and $\|\boldsymbol{\theta}\|_p$ is the p-norm. By introducing this penalizing term features that are not significant will have their coefficients forced to zero. The bigger the value of λ the more we penalize insignificant features. The special case of the p-1 norm is called *LASSO regularization* and for the p-2 norm we refer to *ridge regularization*[9].

3 Tree-Based Models

3.1 Introduction to Decision Trees

In this section we present the idea behind decision trees and how they are generated from a training set of observations in order to be used for prediction. The type of decision trees to be introduced are *classification and regression trees(CART)* which are two of the most commonly used tree-based models. Classification trees are used to predict categorical response variables while regression trees predict their continuous counterpart. The ideas presented in this section are based on the content on decision trees in the book *The Elements of Statistical Learning*[9].

Consider now the case where we are interested in predicting a response variable $Y \in \mathbb{R}$ which is a function of $\boldsymbol{X} \in \mathbb{R}^p$ consisting of p predictors which can be a mixture of both categorical and continuous variables. We denote the predictor space i.e. the set of all possible predictors by $\mathbb{X} \subseteq \mathbb{R}^p$. At our hand we have a training set $\mathcal{L} = \{(\boldsymbol{x_i}, y_i)\}_{i=1}^N$ consisting of N i.i.d samples from $(\boldsymbol{X}, Y)$. The idea is now to form regions in $\mathbb{X}$ such that homogeneous groups of observations are divided into these regions. New observations will then be predicted based on which region in the predictor space they fall into.

These groups are formed by partitioning the predictor space into non-overlapping regions $R_1, ..., R_J$ covering the whole predictor space. In theory these regions could be of any shape but only multidimensional boxes(hypercuboids) will be considered. The partitioning is done in a sequence of binary splits which means that each step splits a region into two new regions. This yields a tree where each node has exactly two branches except the terminal nodes which has zero branches. The idea is that each split should yield two new regions that are more *pure* than their composition. Thus, a new observation $\boldsymbol{x} \in R_j$ can be predicted using only characteristics of the observations in the j:th region.

The model derived from the tree is a piecewise constant function which can be expressed as:

$$f(\boldsymbol{x}) = \sum_{j=1}^{J} \hat{c}_j \mathbb{1}_{\{\boldsymbol{x} \in R_j\}} \tag{21}$$

where $\mathbb{1}$ is the indicator function which is equal to 1 for observations in the j:th region and 0 otherwise. Thus, $\hat{c}_j$ is the value predicted for observations falling into the j:th region. In the regression setting c_j is simply the mean of the observations in the j:th region and conversely c_j is set as the mode of the j:th region for a classification tree. To give a more precise algorithm of constructing the model we must first define some technicalities such as *impurity measure* that measures the impurity in the regions as well as splitting rules for partitioning the predictor space. However, first we will introduce some terminology used with decision trees.

3.2 Tree Terminology

A decision tree is comprised of structures called *nodes*. Each node is connected to other nodes via *branches*. Trees are grown downwards graphically by convention. Nodes directly connected to a node downwards are called its *child nodes* and nodes connected upwards are called its *parent nodes*. For a binary tree a split at a node yields branches to two new child nodes. Thus, for this type of tree each node has exactly zero or two child nodes. Nodes that have no children are referred to as *external nodes(terminal nodes)* or *leaves.* These nodes are located at the bottom of the tree while the top node of the tree is called the *root.* Nodes with at least one child are called *inner nodes.*

Each inner node is associated with a splitting rule that tells us whether we should choose the left or right branch from the node given an observation. The splitting is based on one variable at each node which is called the *splitting variable*, x_i. The splitting rule at the k:th node can be formulated as:

$$\begin{aligned} x_{jk} < t_k &\Rightarrow left\ \ branch \\ x_{jk} \geq t_k &\Rightarrow right\ \ branch \end{aligned}$$

t is the splitting point and x_{jk} indicates that the j:th predictor is used at k:th node to split. Following these splitting rules down from the root each observation lands at a terminal node which corresponds to a region(hypercuboid) in the predictor space. At this stage no more splitting is done and each observation is given a prediction $\hat{c}_j$ corresponding to the j:th region/terminal node. For a regression tree $\hat{c_j}$ is defined to be the mean of all n_j observations in the j:th region i.e.

$$\hat{c}_j = \frac{1}{n_j} \sum_{\boldsymbol{x}_i \in R_j} y_i.$$

For a classification tree $\hat{c_j}$ is the mode, i.e. the value that appears most often, of the observations in the region instead of the mean.

3.3 Constructing the Tree

In order to construct the tree one must choose the splitting points t_k as well as when to stop the splitting and declare a node as terminal. The aim is to select the splitting points such that the set of training observations are divided into homogeneous groups. Quantitatively we express this as minimizing a loss function. We here consider the squared loss:

$$L(y, \hat{f}(\boldsymbol{x})) = \sum_{j=1}^{J} \sum_{\boldsymbol{x}_i \in R_j} (y_i - \hat{f}(\boldsymbol{x}_i))^2 \overset{\text{not.}}{=} \sum_{j=1}^{J} Q_{R_j}. \tag{22}$$

By minimizing the loss we find regions such that the responses y_i are close to the region mean, thus forming homogeneous groups. Minimizing the loss by considering all possible partitions of the predictor space is computationally unfeasible. Therefore, an algorithm for systematic minimization is needed. The procedure of *binary splitting* is commonly used for this task. This algorithm can be described as following:

1. Select one region $\tilde{R}_i$ in $\mathbb{X}$ and consider the split into two new regions R_1 and R_2 defined as follows:

$$R_1(j,t) = \{\boldsymbol{x} \in \tilde{R}_i : x_j < t\}, \quad R_2(j,t) = \{\boldsymbol{x} \in \tilde{R}_i : x_j \geq t\}$$

2. Minimize

$$Q_{R_1}(j,t) + Q_{R_2}(j,t)$$

w.r.t. j and t. That is, select the splitting variable j and the splitting point t such that the sum of the loss in R_1 and R_2 is minimized.

3. Repeat step 1. and 2. for each region in $\mathbb{X}$. The region that decreases the overall loss the most is chosen to be split.

4. Repeat the above procedure until a stopping rule is fulfilled. A typical stopping rule is to stop splitting once the number of observations in each region is below a pre-set threshold value.

It is clear that the algorithm of binary splitting is *greedy* in the sense that it only searches locally for optimal splits and does not guarantee that the tree reaches a global minimum of the loss. Procedures for improving the tree construction are discussed in section 3.4. Below is an example of a decision tree and its corresponding partitioning of the predictor space.

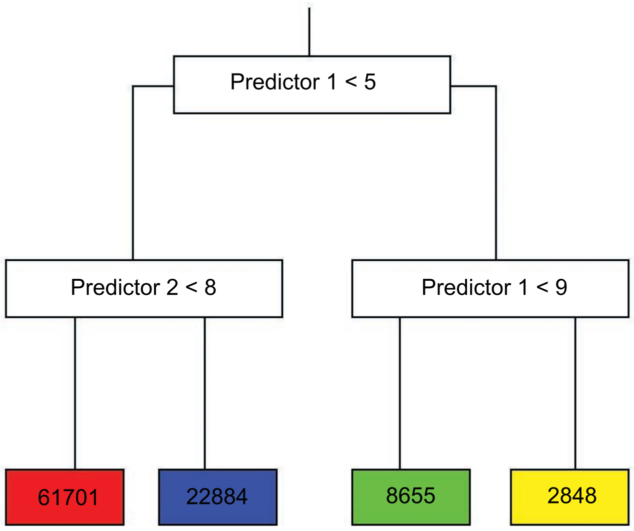

Figure 2: Decision tree example.

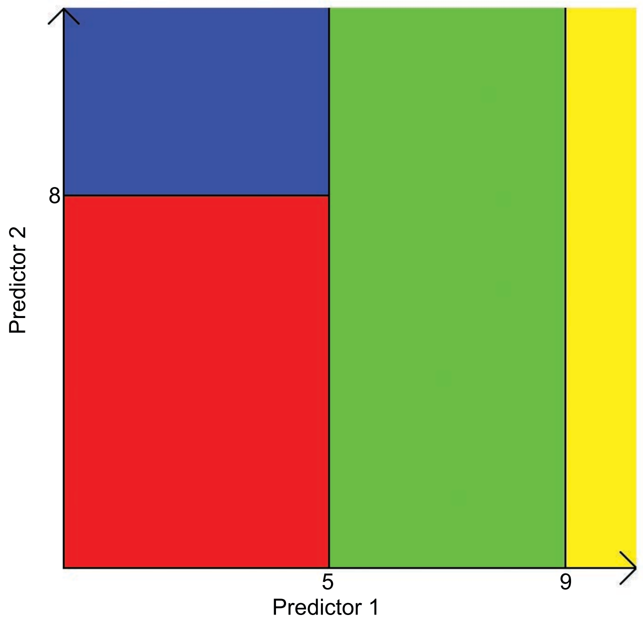

Figure 3: Partitioning of predictor space corresponding to the tree in figure 2.

In figure 2 we see a decision tree with three splits resulting in four leaves. By using the splitting rules and following the branches an observation can be put into one out of the four leaves. The splitting could in theory continue until there is exactly one leaf per observation which would yield a total loss equal to zero! However, this is not desired since an overfit tree doesn't generalize well to unseen observations which we want to predict.

3.4 Tree Pruning

The procedure described above does not take model complexity into consideration and could therefore suffer from overfitting. A model too complex will not perform well on unseen data due to high variance, recall the variance term in equation (12). A natural way to tackle the problem would be to penalize the complexity of the tree to find a good balance between variance and bias in the model. Thus, instead of only considering the the squared loss when fitting the model we modify the loss L to be:

$$L = \sum_{j=1}^{|T|} \sum_{x_j \in R_j} (y_j - f(\boldsymbol{x}_j))^2 + \alpha|T| \tag{23}$$

where α is a tuning parameter and $|T|$ is the number of leaves in the tree, thus penalizing more complex trees. For $\alpha = 0$ we get the full tree but as α increases more and more nodes will be forced out eventually leaving us only with the root. Having only the root would of course yield a model with very high bias. It is therefore necessary to choose a suitable value for α to get a balanced model. This is typically done using K-fold cross-validation. The process can be described as follows:

1. Split the training data into K-folds.

2. For $k = 1, ..., K$ using every fold except the kth:

a) Construct a sequence of trees $\{T\}_{i=1}^{n}$ as a function of α.

b) Compute the loss on the kth fold. The loss will in turn depend on α.

c) Compute the average loss for each value of α using the K trees. Pick the α that minimizes the loss.

3.5 Ensemble Methods

There are numerous ways to enhance the performance of a tree-model. The most common way is to combine several models and use the aggregated information stored in these models to get better predictions than what would be possible using a single model. The method of combining multiple models is known as *ensembling* and is commonly used with decision trees. In this section we will go over a few different ensembling methods. The method of gradient boosting which will be used in the case study is described in 3.5.3.

3.5.1 Bagging

Recall from section 2.2.4 that the expected prediction error one can expect from a model can be decomposed as:

$$E[(Y - \hat{f}(\boldsymbol{X}))^2] = \sigma^2 + Bias(\hat{f}(\boldsymbol{X}))^2 + Var(\hat{f}(\boldsymbol{X})).$$

In order to reduce the variance term the method of *bagging*(**b**ootstrap **agg**regat**ing**) can be used. It can be applied to a variety of statistical learning methods, and it is particularly useful in the context of trees. The method is a way of utilizing multiple training sets to refine the model. Assume we are given a set of training sets $\{\mathcal{L}_b\}_{b=1}^{B}$ where each set is independently drawn from the same probability distribution. For each set $\mathcal{L}_b$ a model $\hat{f}_b(\boldsymbol{x})$ is fit. Having several models instead of a single one it is natural to form the aggregated predictor as the average of these. By the law of large numbers the average of the models converge to the expectation of $\hat{f}(\boldsymbol{x})$ as b approaches infinity[3]. We write this as such:

$$\hat{f}_{agg}(\boldsymbol{x}) = \lim_{b\to\infty} \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(\boldsymbol{x}) = E[\hat{f}(\boldsymbol{x})] \tag{24}$$

The use of the aggregated predictor can be motivated as follows. The mean-squared prediction of errors of the aggregated and non-aggregated predictors are defined as:

$$\begin{aligned} e &= E[(Y - \hat{f}(\boldsymbol{x}))^2] \\ e_{agg} &= E[(Y - \hat{f}_{agg}(\boldsymbol{x}))^2] \end{aligned}$$

We will now show that $e \geq e_{agg}$.

Proof.

$$\begin{aligned} e &= E[(Y - \hat{f}(\boldsymbol{x}))^2] \\ &= E[Y^2] - 2E[Y]E[\hat{f}(\boldsymbol{x})] + E[\hat{f}(\boldsymbol{x})^2] \\ &= E[Y^2] - 2E[Y]E[\hat{f}_{agg}(\boldsymbol{x})] + E[\hat{f}(\boldsymbol{x})^2] \\ &\geq E[Y^2] - 2E[Y]E[\hat{f}_{agg}(\boldsymbol{x})] + E[\hat{f}(\boldsymbol{x})]^2 \\ &= E[(Y - \hat{f}_{agg}(\boldsymbol{x})^2] = e_{agg} \end{aligned}$$

where we used the facts that Y and $\hat{f}(\boldsymbol{x})$ are independent and the inequality follows from Jensen's inequality. □

Thus, we can conclude that the aggregated predictor produces predictions at least as accurate as the non-aggregated predictor. In reality one doesn't have access to an infinite amount of training sets $\mathcal{L}_b$. This is where the method of *bootstrap* comes in. From our single training set we draw samples with replacements which we form into B different training sets. The b:th bootstrapped training set then yields a model $\hat{f}_b(\boldsymbol{x})$. We now form the function $\hat{f}_{bagg}(\boldsymbol{x})$ to approximate $\hat{f}_{agg}(\boldsymbol{x})$. We define it as:

$$\hat{f}_{bagg}(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(\boldsymbol{x}) \tag{25}$$

There is no clear rule for how many bootstrap sets should be used but the more the better since:

$$\lim_{b \to \infty} \hat{f}_{bagg}(\boldsymbol{x}) = \hat{f}_{agg}(\boldsymbol{x}) = E[\hat{f}(\boldsymbol{x})].$$

3.5.2 Random Forests

The *random forests* approach of building a tree model is very similar to bagging but it improves the model by reducing the correlation between the trees in the ensemble $\{\hat{f}_b(\boldsymbol{x})\}_{b=1}^{B}$[4]. We will first establish how the correlation between the trees affects the prediction error, specifically the variance term.

$$\begin{aligned}
Var(\hat{f}_{bagg}(\boldsymbol{x})) &= Var\left(\frac{1}{B}\sum_{b=1}^{B}\hat{f}_b(\boldsymbol{x})\right) \\
&= \frac{1}{B^2}\sum_{i=1}^{B}\sum_{j=1}^{B} Cov(\hat{f}_i(\boldsymbol{x}), \hat{f}_j(\boldsymbol{x})) \\
&= \frac{1}{B^2}\sum_{i=1}^{B}\left(\sum_{j\neq i}^{B} Cov(\hat{f}_i(\boldsymbol{x}), \hat{f}_j(\boldsymbol{x})) + Var(\hat{f}_i(\boldsymbol{x}))\right) \\
&= \frac{1}{B^2}\sum_{i=1}^{B}\big((B-1)\rho\sigma^2 + \sigma^2\big) \\
&= \frac{B(B-1)\rho\sigma^2 + B\sigma^2}{B^2} \\
&= \rho\sigma^2 + \sigma^2\frac{1-\rho}{B}
\end{aligned} \tag{26}$$

where we have assumed that all combination of trees share the same correlation ρ and the variance σ^2 of the individual trees $\hat{f}(\boldsymbol{x})$ are equal since the bootstrapped training sets are identically distributed. By increasing the number of trees B we can reduce the second term in (26). The first term can be decreased by decorrelation of the trees. The difference from the bagging method is that for random forests one does not consider all p predictor variables when choosing which variable to split. Instead $m \leq p$ randomly selected predictors are considered at each split. The motivation for this can be described as follows. Since binary splitting searches locally for the best splitting variable the most influential variables will be at the top of the tree thus making all trees very similar. If we instead only are allowed to choose from a subset of predictors at each split the trees will be less correlated. m is a tuning parameter but it is often set to $m = \sqrt{p}$. In the special case where $m = p$ we get the bagging process which we described earlier.

3.5.3 Gradient Boosting

Gradient boosting/gradient boosting machine(GBM) is the method of combining multiple relatively weak learning models into an ensemble that can produce accurate predictions. In order to derive the gradient boosting method we first need to introduce some concepts in optimization on which GBM is founded.

Parametric Numerical Optimization

When working with optimization problems it is common to optimize a parametrized function $F(\boldsymbol{x};\boldsymbol{\theta})$ with respect to the parameters in $\boldsymbol{\theta}$. For a minimization problem we write this as:

$$\min_{\boldsymbol{\theta}} \quad F(\boldsymbol{x};\boldsymbol{\theta}). \tag{27}$$

In practice these problems rarely have analytical solutions. Hence, we need to find an approximate solution using numerical methods. Next we introduce the method of *gradient descent* which can be used to numerically find minima or maxima of multivariate objective functions.

Gradient Descent

Gradient descent utilizes the fact that a function $F(\boldsymbol{\theta})$ differentiable in a neighborhood of a point $\boldsymbol{a}$ decreases the fastest by going in the direction of the negative gradient evaluated at the point $\boldsymbol{a}$, $-\nabla F(\boldsymbol{a})$[8]. By iteratively taking sufficiently small steps ρ in the direction of the negative gradient we can make $F(\boldsymbol{x};\boldsymbol{\theta})$ approach local minima. The estimates of $\boldsymbol{\theta}$ are updated as:

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \rho_n \nabla F(\boldsymbol{\theta}_n) = \boldsymbol{\theta}_n - \rho_n g_n(\boldsymbol{\theta}_n)$$

where the algorithm is initialized with the starting guess $\boldsymbol{\theta}_0$. The step sizes ρ_n can be found by:

$$\rho_n = \arg\min_{\rho} F(\boldsymbol{\theta_{n-1}} - \rho \nabla F(\boldsymbol{\theta}_{n-1})). \tag{28}$$

This is called *line search* and chooses the step size such that $F(\boldsymbol{\theta})$ is minimized in the direction of the negative gradient.

The iterations continue until a stopping criterion is fulfilled.The following is commonly used:

$$|\nabla F(\boldsymbol{\theta_n})| \leq tolerance.$$

The resulting solution after n iterations can thus be written as:

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_n \tag{29}$$

Numerical Optimization in Function Space

Recall that the aim of any statistical learning method is to find a function that minimizes the risk function i.e. we want to solve the problem:

$$\min_{F \in \mathcal{F}} \quad R(F(\boldsymbol{X})) = E[L(y, F(\boldsymbol{X}))] = E\big[E[L(y, F(\boldsymbol{x}))|\boldsymbol{X} = \boldsymbol{x}]\big] \tag{30}$$

or equivalently:

$$\min_{F \in \mathcal{F}} \quad R(F(\boldsymbol{x})) = E[L(y, F(\boldsymbol{x}))|\boldsymbol{X} = \boldsymbol{x}] \tag{31}$$

for each $\boldsymbol{x}$.

This is analogous to the parametric optimization problem but the "parameter" is now the function $F(\boldsymbol{x})$[5]. We can search for solutions in the same way using gradient descent and iteratively update our estimate as:

$$\begin{aligned} F_{n+1}(\boldsymbol{x}) &= F_n(\boldsymbol{x}) - \rho_n \frac{\partial E[L(y, F_n(\boldsymbol{x}))|\boldsymbol{x}]}{\partial F_n(\boldsymbol{x})} = F_n(\boldsymbol{x}) - \rho_n E\left[\frac{\partial L(y, F_n(\boldsymbol{x})}{\partial F_n(\boldsymbol{x}))}|\boldsymbol{x}\right] \\ &= F_n(\boldsymbol{x}) - \rho_n g_n(\boldsymbol{x}) \end{aligned}$$

(32)

where we have assumed sufficient regularity so that differentiation and integration can be interchanged.

The step sizes are found from:

$$\rho_n = \arg\min_{\rho} E\left[[L(y, F_{n-1}(\boldsymbol{x})) - \rho E\left[\frac{\partial L(y, F_{n-1}(\boldsymbol{x}))}{\partial F_{n-1}(\boldsymbol{x})}|\boldsymbol{x}\right]\right]. \tag{33}$$

When the stopping criterion is fulfilled after n steps we obtain the solution:

$$F^*(\boldsymbol{x}) = F_n(\boldsymbol{x}). \tag{34}$$

The problem that we are faced with in practice is the fact that the negative gradient $-g_n(\boldsymbol{x})$ can only be computed at the points $\{(\boldsymbol{x_j}, y_j)\}_{j=1}^m$ in our training set $\mathcal{L}$. We attack this problem by approximating $g_n(\boldsymbol{x})$ using functions $h(\boldsymbol{x})$ from the class of functions $\mathcal{H}$. We call these *base learners.* The base learners are fitted to $g_n(\boldsymbol{x})$ as:

$$h_n(\boldsymbol{x}) = \arg\min_{h\in\mathcal{H},\beta} \sum_{j=1}^{m}(-g_n(\boldsymbol{x_j}) - \beta h(\boldsymbol{x_j}))^2. \tag{35}$$

These base learners are often chosen as decision trees which is also the base learner of choice in this thesis. The gradient boosting method for fitting functions can thus be summarized as follows:

Algorithm 1 Gradient Boosting

```
1: $\hat{f}_0(\boldsymbol{x}) = \arg\min_\rho \sum_{j=1}^m L(y_j, \rho)$
2: for $i = 1, ..., n$ do
3:     $g_i(\boldsymbol{x}_j) = \frac{\partial L(y, \hat{f}_{i-1}(\boldsymbol{x}_j))}{\partial \hat{f}_{i-1}(\boldsymbol{x}_j)}$, $j = 1, ..., m$
4:     $h_i(\boldsymbol{x}) = \arg\min_{h\in\mathcal{H},\beta} \sum_{j=1}^m(-g_n(\boldsymbol{x_j}) - \beta h(\boldsymbol{x_j}))^2$
5:     $\rho_i = \arg\min_\rho \sum_{j=1}^m L(y_j, \hat{f}_{n-1}(\boldsymbol{x}_j) + \rho h_i(\boldsymbol{x}_j))$
6:     $\hat{f}_i(\boldsymbol{x}) = \hat{f}_{i-1}(\boldsymbol{x}) + \rho_i h_i(\boldsymbol{x})$
7: return $\hat{f}(\boldsymbol{x}) = \hat{f}_n(\boldsymbol{x})$
```

Often one introduces an extra factor η in the algorithm such that:

$$F_n(\boldsymbol{x}) - \rho_n g_n(\boldsymbol{x}) \to F_n(\boldsymbol{x}) - \eta \rho_n g_n(\boldsymbol{x}), \quad 0 < \eta \leq 1. \tag{36}$$

This is called *shrinkage* and η refers to the learning rate of the model. By decreasing the learning rate we get a more conservative fitting algorithm that is less prone to overfit.

4 Case Study

In this section we will discuss topics on how the analysis was performed in practice such as data preprocessing and feature selection. The data preprocessing and modeling was performed using R.

4.1 Data Preprocessing

The unprocessed claims data set consists of roughly 375 000 observations. Out of these observations approximately 2000 observations had a negative value of the response variable *claims reserve*. These cannot be handled by the GLM with logarithmic link function why they were removed from the dataset. We also remove a few hundred rows that contained empty values in one or multiple columns. GBM is able to handle observations with empty categories but GLM is not why we decide to dismiss these faulty rows. Since the percentage of rows with missing values is so small we drop these instead of filling in the missing values.

For training 80% of the data was used while the remaining 20% were used for final evaluation of the models. In the models there are certain features that can be modified such as the depth of the decision trees or the number of decision trees to use in the ensemble. These features are the hyperparameters of the model. These have a great impact on the performance of the model why they should be optimized. This leaves us with two choices. Either we could partition our full data set into three parts such that we have a training, test and validation set or we can discard the validation set and use cross-validation on the training set. In this study we use the latter alternative.

From the data set five different predictor variables were chosen as they were known to be correlated with the response variable from previous experience. Four out of these five variables are numeric and one is a multilevel categorical variable with 36 level. The distribution of the categorical variable is very skewed where a very large fraction of the observations is contained in a small number of categories. If the number of observations in a category is very small it can be difficult to make accurate prediction. We therefore create a new variable where we aggregate categories into a new of the categorical variable with only 6 categories. We aggregate the categories such that each new category contains observations that share certain properties. The figures below are based on the training data.

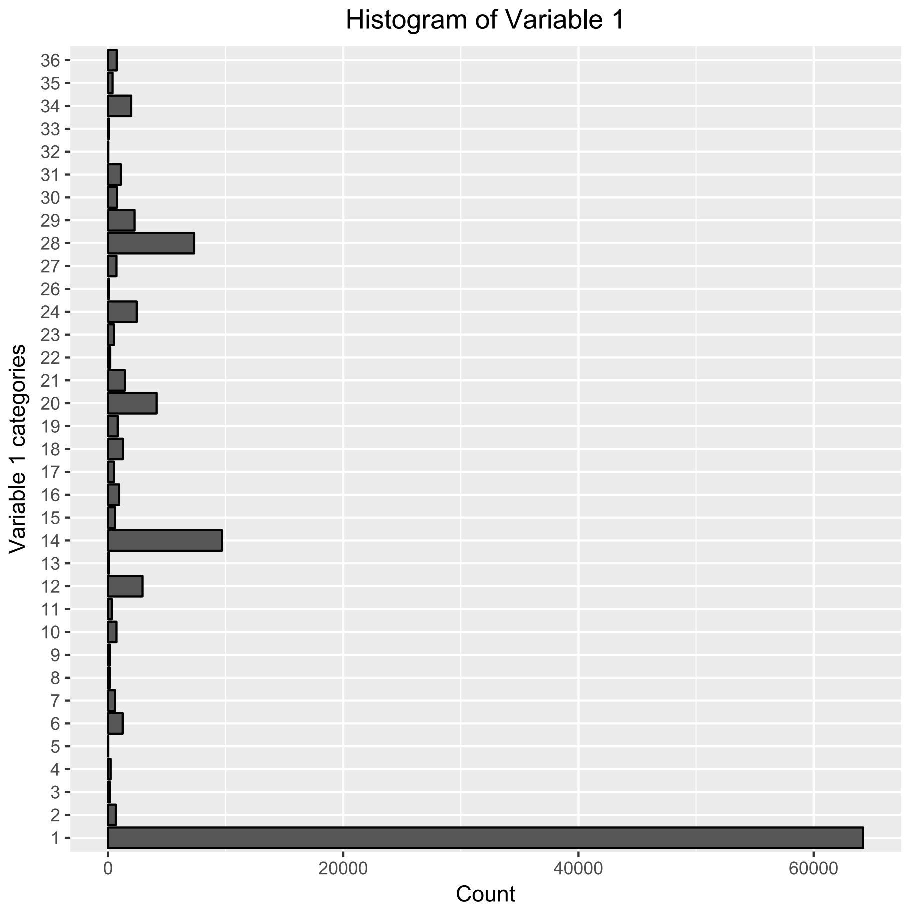

Figure 4: Number of observations per category.

As shown in figure 4 the vast majority of observations can be fit into a small number of levels. However, many categories only contain observations in the lower hundreds, some even less than 100. The variance within these categories can therefore be very high. In order to make this problem less severe we aggregate levels into 6 categories where each new level contains aggregated levels from our old categorical. By doing so we can make predictions not only using the finer division of levels but also the new variable with fewer levels. A histogram of these are shown in figure 5.

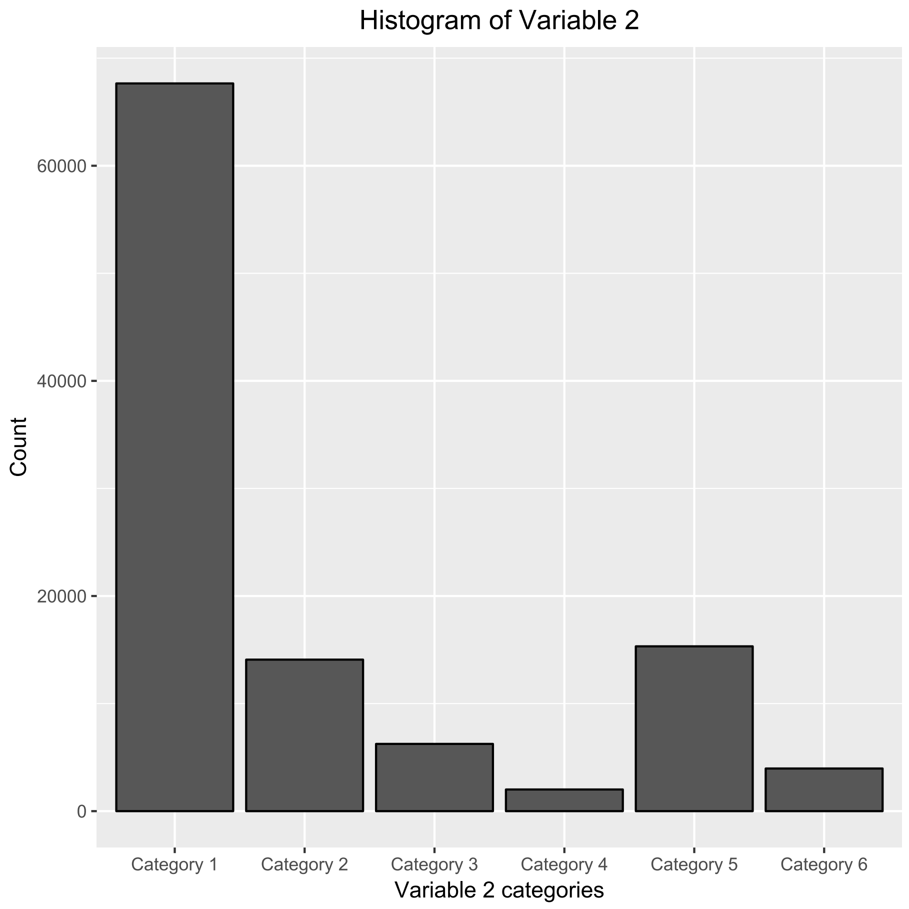

Figure 5: Number of observations per category.

After we aggregated the categories we have 6 levels where the smallest one contains approximately 5600 observations. Lastly the data was split into training and test set. The test set is not used until the final models for GLM and GBM are found using the training set. We make no further selections in the data before using it as input to the models. The predictor variables are summarized in table 1.

Table 1: Predictor Variables

Variable Name	Type	Notes
Variable 1	Categorical	36 levels
Variable 2	Categorical	6 levels. Aggregated categories of Variable 1
Variable 3	Numeric	-
Variable 4	Numeric	-
Variable 5	Numeric	-
Variable 6	Numeric	Represents time since claim occured. Measured in arbitrary time units.

4.2 Feature Selection

From the raw dataset six different variables were picked as they have been proven to be essential for this type of modelling from previous experience. In order to penalize less significant features we include a LASSO penalty when fitting the GLM, see section 2.5. LASSO regression performs feature selection in the sense that non-significant features will be biased towards zero and hence not have a big impact in the model.

For the gradient boosting method the same features are used to give a fair comparison to the GLM alternative. Tree models are more robust to non-significant features since the trees are always split by the feature that yields the highest decrease in the loss function. Thus, the most important feature will always be split first in the tree. Tree based algorithms therefore performs feature selection implicitly. The features used in the modelling process are summarized in the following table.

4.3 GLM

For the GLM we have two choices to make. We need to choose link and variance function. The variance function is limited to the Tweedie distributions i.e. $V(\mu) = \mu^p$. The typical values used to fit claims data is $p \geq 1$. However, for $p = 2$ we get the gamma distribution which only has support on the positive reals and can therefore not be used with the data since it contains zeroes. We therefore restrict p to the following values.

$$p = \{1, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9\}.$$

We use the logarithmic link function as it is the canonical link .The different values for p are evaluated using 10-fold cross validation and the lowest scoring model is used for final assessment. The coefficients are calculated using the maximum likelihood method with a LASSO penalty.

4.4 Gradient Boosting

For the gradient boosting method there are several hyperparameters that should be tuned in order to get a good approximation and prevent overfitting. The following parameters were tuned:

- **nrounds**: This parameter sets the number of trees to be used in the additive tree model. See algorithm 1 in section 3.5.3. The more trees are used the more complex structures can be captured in the data. However, too many trees will cause overfitting.
- **learning_rate(η)**: This parameter controls the learning rate of the model. See equation (36). In practise it is always better to use smaller values of η to prevent overfitting but it also requires more computing time. Thus, in practice one should use a small value that can fit a model in a reasonable amount of time. Smaller learning rate requires more trees why **nrounds** and η should be tuned simultaneously.
- **max_depth**: Sets depth in the tree. The deeper the tree the more complex structures in the data can be captured. However, it also makes the model more vulnerable to overfitting.
- **min_child_weight**: This parameter sets the minimum number of observations needed in each new node at a split. The lower value we use the finer splits can be made. Higher values yields a more conservative model.

- **gamma**: Each time a split is considered the resulting decrease in the loss is calculated. If the loss is very small it is likely that the model is just adjusting to noise. Only splits that yield a decrease in loss bigger than **gamma** will be considered.
- **subsample**: Denotes the fraction of features considered at each iteration. See section 3.5.2. The features are selected randomly at each iteration. By not considering all features in every iteration of the fitting process we can decrease the correlation between the trees in the sequence. By decreasing the subsample size we can get a fitting algorithm more robust to noise.
- **colsample**: Adds randomness to the fitting process just as the **subsample** parameter by only considering a fraction of the rows used to fit a new tree. Usually tuned simultaneously with *subsample*.
- **early_stopping_rounds**: Another parameter that is used to prevent overfitting. If the loss doesn't decrease within this number of iterations the procedure is stopped.

The gradient boosting model needs more tweaking than the GLM since there are more parameters to choose from. The optimal strategy would be to test combinations of all parameters at once but this would be too time consuming. Therefore, we will test them sequentially instead. We will start by tuning the parameters that have the largest influence on the model behaviour and then tweak the less prominent parameters. The parameter testing can be divided into the following steps.

1. **Fix learning_rate and nrounds.**

   In general the model performs better the lower η we use. However, this requires more trees which in turn requires more computing time. Typical values for η are 0.01 and 0.001 but the latter alternative turned out to be too time consuming why it could not be used. For the **nrounds** parameter we should set a value that allows the the cross-validation error to reach a minimum or at least stabilize. From observations in initial testing we find that the following values for the parameters were suitable:

   learning_rate $= 0.1$

   nrounds $= 500$

   The initial values for **gamma**, **subsample** and **colsample** are 0, 1, 1 respectively. These are the default values which will later be tuned as well.

2. **Tune max_depth and min_child_weight.**

   These two parameters affects the models behaviour the most why we tune them first. Since we only have six predictor variables we will only consider smaller values of **max_depth**. This is also in line with the philosophy behind gradient boosting that an ensemble of weak learners should be used together for better predictive capacity. The combinations of the following values were tested:

   max_depth = {1, 2, 3, 4, 5}

   min_child_weight {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}.

3. **Tune gamma.**

   Using the optimal combination of **max_depth** and **min_child_weight** we move on to tune **gamma**. We restrict ourselves to values in the the interval $[0, 1]$ which is sufficient for most problems. The following values were tested:

   gamma = {0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1}.

4. **Lower learning_rate and increase nrounds** for the previously found parameters in order to get better precision in the model and to reduce the likelihood of overfitting.

The evaluation of the hyperparameters will be done using 5-fold cross-validation instead of 10-fold as was done with GLM since it's considered too time consuming. We will use root mean squared error(RMSE) as error metric to compare the performance of using different combinations of parameters. RMSE is defined as the square root of the expression in equation (10). When evaluating the results of the models we will also use *relative approximation error* or just *relative error*. We introduce this metric since it's more easily interpretable than the RMSE. This can give us a better feel for how the models behave. The relative error is defined as:

$$relative\ error = \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)}{\sum_{i=1}^{n} y_i}. \tag{37}$$

The relative error allows us to interpret by what percentage our aggregated estimations differ from the aggregated actual values.

5 Results

5.1 Cross-Validation GLM

The best value for the variance power p in the Tweedie distribution was chosen using 10-fold cross-validation. We evaluate the models based on their RMSE.

Table 2: Cross-Validation of Tweedie Distributions

p	RMSE
1	34832
1.1	35140
1.2	35392
1.3	35595
1.4	35761
1.5	35901
1.6	36030
1.7	36212
1.8	36730
1.9	38485

Summary of 10-fold cross-validation using different values of p characterizing the variance function in the Tweedie models.

The trend we notice in table 2 is that higher values of p seems to decrease the model score. $p = 1$ which corresponds to the Poisson distribution yields the smallest error why this value was chosen for the final GLM model.

5.2 Cross-Validation Gradient Boosting

Table 3: Hyperparameter Cross-Validation: **max_depth, min_child_weight**

max_depth	**min_child_weight**	**gamma**	**subsample**	**colsample**	RMSE
5	9	1	1	1	31934
5	3	1	1	1	32046
5	4	1	1	1	32155
5	10	1	1	1	32186
5	5	1	1	1	32291

Summary of 5-fold cross-validations using different combinations of **max_depth** and **min_child_weight**. Only the five lowest scoring models are shown in the table.

The result shows that the optimal choice for **max_depth** is 5. The RMSE using different values of **min_child_weight** are very close but ultimately we go with 9 for further analysis of hyperparameters.

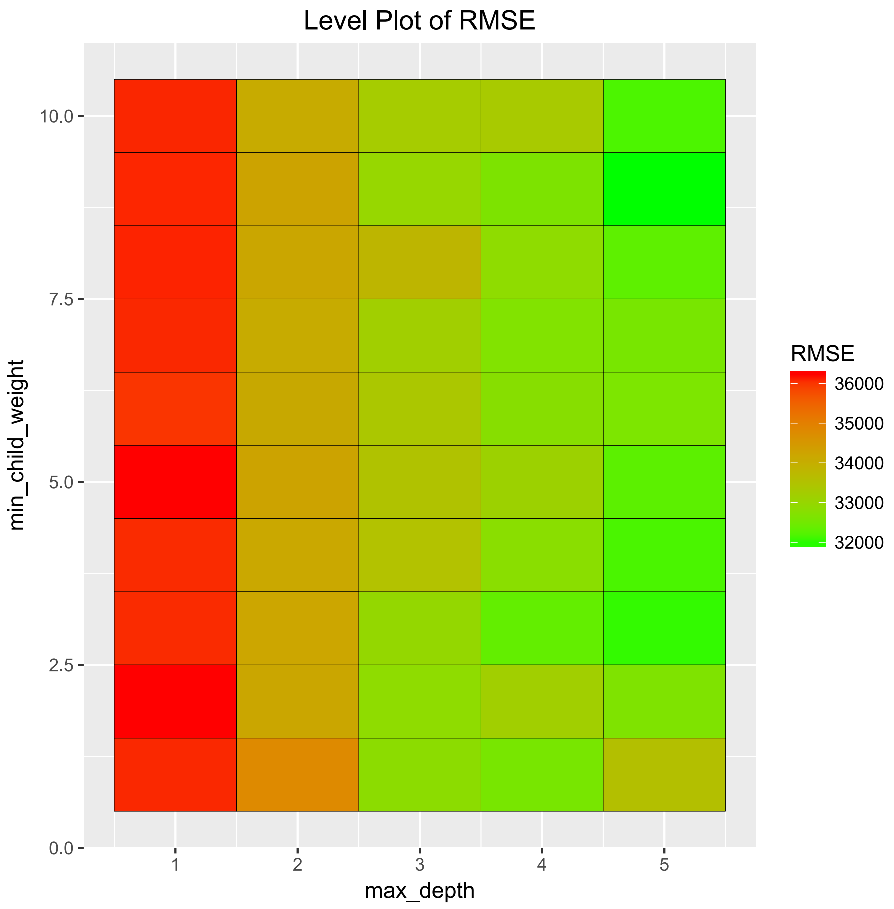

Figure 6: Level plot of all tested combinations of **max_depth** and **min_child_weight** using 5-fold cross-validation.

Figure 6 suggests that too small values for **max_depth** produces very biased models which can't utilize all the information provided by the data. As we grow the trees deeper the models become more complex and can make more accurate predictions resulting in lower values of the RMSE. Next we interrogate how the predictive accuracy depends on **gamma**.

Table 4: Hyperparameter Cross-Validation: **gamma**

max_depth	**min_child_weight**	**gamma**	**subsample**	**colsample**	RMSE
5	9	0	1	1	32483
5	9	0.1	1	1	32849
5	9	0.2	1	1	32431
5	9	0.3	1	1	32292
5	9	0.4	1	1	32758
5	9	0.5	1	1	32326
5	9	0.6	1	1	32475
5	9	0.7	1	1	32522
5	9	0.8	1	1	32225
5	9	0.9	1	1	32475
5	9	1	1	1	32841

Summary of 5-fold cross-validations using different values of **gamma**.

The results for **gamma** are also very close but the lowest RMSE is produced when using 0.8 why it will be used in the final model. The observant reader might notice that the values of RMSE are actually higher than in table 4 and that the same set of parameters yields different values of RMSE. This is due to the fact that multithreading was used to compute the models faster. The threads don't sync the parameters updates in order to provide the highest efficiency. Therefore, the results can vary between different trials. This problem doesn't arise when only using one thread but will require much more computing time which is not feasible when testing this many parameters. The setup used is not optimal but it should guide us in the right direction when choosing values for the hyperparameters.

Table 5: Hyperparameter Cross-Validation: **colsample**, **subsample**

max_depth	**min_child_weight**	**gamma**	**subsample**	**colsample**	RMSE
5	9	1	1	0.9	32306
5	9	1	1	0.7	32487
5	9	1	1	0.6	32535
5	9	1	1	0.5	32544
5	9	1	1	1	32786

Summary of 5-fold cross-validations using different combinations of **colsample** and **subsample**. Only the five lowest scoring models are shown in the table.

Table 5 shows that the five lowest scoring models used **subsample**=1 which suggests that we should always consider all possible features when making a split. The result also suggests that it is not optimal to consider all observations in the training set when making splits. The optimal value is found to be **colsample**= 0.9 which corresponds to using 90% of the observations.

As we have tuned all parameters we finally need to tweak **nrounds** for this set of parameters that we have found. We use early stopping of 10 rounds to prevent overfitting and find the optimum value of trees which turned out to be 433. Thus, the final values for the hyperparameters can be summarized as follows.

max_depth	**min_child_weight**	**gamma**	**subsample**	**colsample**	RMSE
5	9	0.8	1	0.9	32483

5.3 Overall Performance

Model	RMSE	Relative Error
GLM	42473	-0.13
GBM	44371	0.04

Table 6: Model performance on the whole test set

The final assessment of the model performances is summarized in table 6. It is worth noting that the GLM had the lower RMSE but gradient boosting only overestimated the total claims reserves by 4% whereas the GLM yielded an underestimation of 13%. However, this does not mean that gradient boosting should be preferred over GLM since terms can cancel out in the relative error. The relative errors should rather be used to give us an intuition of how well the model performs which can be rather difficult from the RMSE alone. The fact neither models produces relative errors that deviates too far from 0 suggests that they can both yield feasible predictions.

5.4 Performance Per Predictor Variable

In this section we look more into detail how the models perform for certain subsets within each predictor variable. This helps us identify if the models are able to yield reasonable predictions for all types of policyholders. The data is aggregated to understand how the models predict in certain subsets of the test set.

Table 7: Model performance for predictor variable *2*.

Group	RMSE GLM	RMSE GBM	Relative Error GLM	Relative Error GBM	Observations
Category 1	13154	12586	0.12	0.02	42123
Category 2	15786	23616	-0.24	0.20	346
Category 3	33136	36394	0.01	0.02	141
Category 4	90853	84497	-0.93	-0.73	82
Category 5	-	-	-	-	0
Category 6	8783	9729	0.22	0.11	788
Category 7	5809	3247	6.79	0.76	291
Category 8	23535	29939	0.24	0.60	225
Category 9	4759	27677	29.23	7.76	181
Category 10	26122	36394	-0.13	-0.23	692
Category 11	12279	31938	11.47	32.35	95
Category 12	7951	7218	0.07	-0.12	1628
Category 13	22619	27638	0.69	1.98	64
Category 14	13421	12801	0.16	0.08	6835
Category 15	99178	92213	-0.42	-0.50	606
Category 16	58778	58396	0.31	0.53	591
Category 17	265313	268302	-0.79	-0.82	523
Category 18	82864	83422	-0.26	-0.24	904
Category 19	100534	108534	-0.48	-0.22	577
Category 20	15474	15332	-0.14	-0.30	2749
Category 21	46184	48912	-0.37	-0.38	699
Category 22	12042	12129	0.11	0.64	134
Category 23	43398	44086	0.24	0.93	417
Category 24	17372	20480	1.25	0.92	1549
Category 25	1975	5135	Inf	Inf	41
Category 26	22688	19234	Inf	Inf	52
Category 27	27554	217126	0.72	13.90	496
Category 28	26761	26647	0.30	0.18	4836
Category 29	67192	66461	-0.39	-0.43	2563
Category 30	17755	18819	1.01	0.60	324
Category 31	148451	160174	-0.40	0.39	910
Category 32	802338	195269	-0.65	-0.21	25
Category 33	15221	65400	0.35	1.97	53
Category 34	68010	68281	-0.13	-0.20	1406
Category 35	14860	6742	4.58	1.54	91
Category 36	86525	86793	-0.24	-0.15	527

The models produce similar results overall. In some categories we find very high values of RMSE and relative errors but at we also note that many of these categories have relatively few observations. Therefore, the models don't have very much information to base their predictions on which could explain these high errors.

Table 8: Model performance for predictor variable *2*.

Group	RMSE GLM	RMSE GBM	Relative Error GLM	Relative Error GBM	Observations
Category 1	13542	13352	0.19	0.09	44436
Category 2	75994	76549	-0.36	-0.34	10181
Category 3	7700	7914	0.73	0.19	4040
Category 4	27041	27922	-0.16	-0.11	1260
Category 5	74385	80792	-0.19	0.31	11231
Category 6	15623	17168	0.05	0.04	2415

The variations in RMSE are very drastic over different categories for variable 2. At the same time, both models produce consistent results over all categories which suggests that it is not necessarily the models that are ill-tuned. It is more likely that these observations are difficult to predict solely based on which category they belong to.

Table 9: Model performance for *Variable 3*.

Group	RMSE GLM	RMSE GBM	Relative Error GLM	Relative Error GBM	Observations
Interval 1	60045	61223	0.31	-0.01	21142
Interval 2	14853	16209	-0.08	0.65	6884
Interval 3	40340	40428	0.08	1.07	8078
Interval 4	27669	34706	0.30	0.42	11318
Interval 5	35611	37590	0.35	-0.56	26141

The performance for both models are comparable overall. The only group where the models differ substantially is in interval 3 where GLM is able to provide much more accurate predictions. In interval 3 we see that GBM predicts claims reserves that are more than twice of the actual value even though both models score almost identical RMSEs. However, as noted earlier we shouldn't base the model validity on the relative error.

Table 10: Model performance for predictor variable *Variable 4*.

Group	RMSE GLM	RMSE GBM	Relative Error GLM	Relative Error GBM	Observations
Interval 1	44960	48828	-0.16	0.19	14296
Interval 2	54953	58057	-0.23	0.12	23200
Interval 3	36230	35350	0.14	-0.21	19253
Interval 4	20598	20643	0.56	-0.04	8026
Interval 5	30039	29816	- 0.37	-0.52	3470
Interval 6	19097	19302	-0.33	-0.02	5318

The discrepancy in RMSE is very small between the models for all intervals. There seems to be a trend where the RMSE decreases in the higher intervals but

Table 11: Model performance for predictor variable *variable 5*.

Group	RMSE GLM	RMSE GBM	Relative Error GLM	Relative Error GBM	Observations
Interval 1	24187	24756	-0.31	-0.19	8026
Interval 2	42129	48929	-0.15	0.04	38472
Interval 3	58278	49589	-0.14	0.05	14619
Interval 4	31018	34481	0.13	0.30	8229
Interval 5	22629	22162	0.49	0.04	4217

Once again we see similar results between the two models and the relative errors show no severe over/under-estimations in any categories.

Table 12: Model performance for *Variable 6.*

Group	RMSE GLM	RMSE GBM	Relative Error GLM	Relative Error GBM	Observations
Interval 1	124029	123708	-0.34	-0.01	2578
Interval 2	97168	97280	-0.14	-0.04	2750
Interval 3	70550	66254	0.12	-0.13	2868
Interval 4	67935	60899	0.15	-0.24	2860
Interval 5	64120	54320	0.16	-0.26	2807
Interval 6	64514	55441	0.02	-0.21	2781
Interval 7	38527	41673	0.15	-0.26	2698
Interval 9	33422	37785	0.20	-0.09	2655
Interval 10	31976	36543	0.11	-0.01	2567
Interval 11	25292	30624	0.04	0.09	2529
Interval 12	19172	25731	-0.08	0.16	2447
Interval 13	17875	24953	-0.23	0.18	2376
Interval 14	15417	23440	-0.27	0.48	2303
Interval 15	11615	22352	-0.66	1.26	22597
Interval 16	1413	22624	-0.87	42.20	12749
Interval 17	0.16	26788	Inf	Inf	3998

The intervals are ordered such that the first interval is closest to the occurence date of the claims and the last interval contain claims furthest from the occurence date. From table 7 we notice a clear pattern that the RMSE decreases the more the claim matures. We can also see this in the figure 5 below. This is a natural behaviour of the models since it is generally known that the major portion of the total loss of a claim is paid out early on while smaller amounts might be paid closer to the settlement date. Bigger losses are of course more difficult to predict than the smaller sums paid closer to the settlement date which explains why this pattern is detected in the results. This is also shown in figure 7.

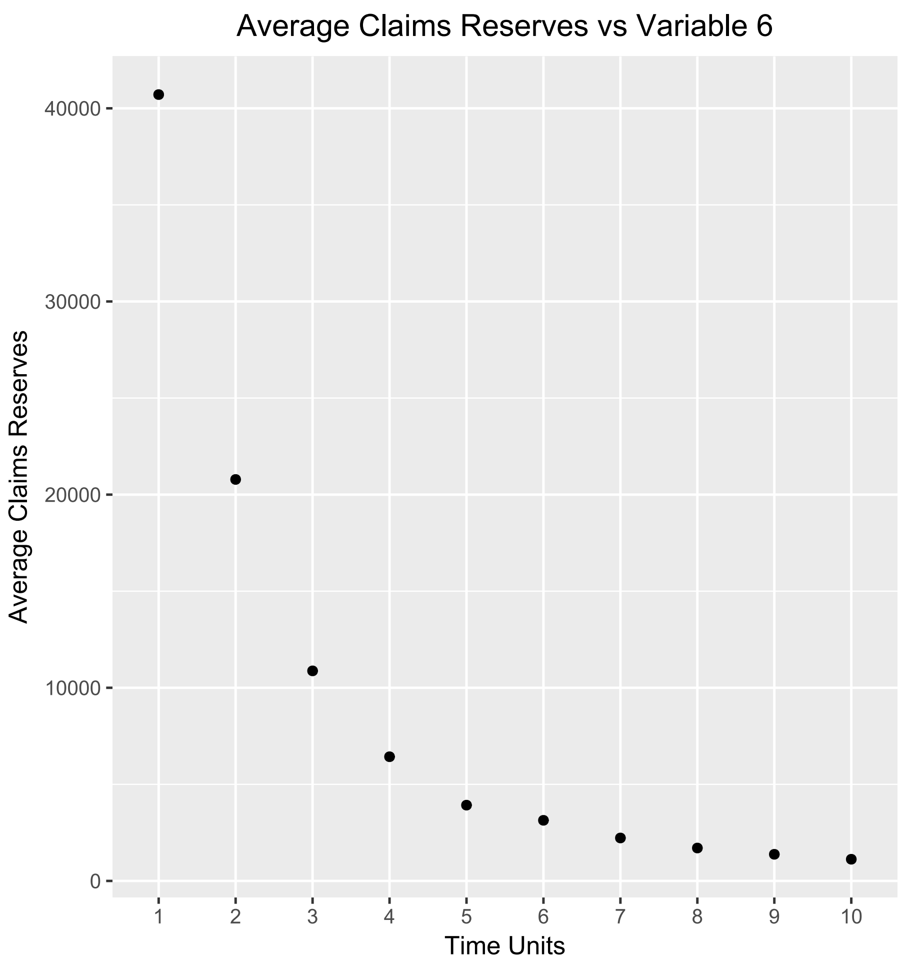

Figure 7: Time dependence of average claims reserves.

The value that stands out in table 11 is the second to last element of relative error for gradient boosting. However, it is worth noting that the average actual reserve for this development period is 29 SEK while the average reserve for all observations is 4194 SEK. Such a big relative error does therefore not have a very big impact on the total claims reserve. If we would find this value in one of the first intervals we should be more worried since it would imply that the model can't predict bigger amounts very well. This in turn would set the claims reserves very high or very low which is not ideal.

6 Summary and Conclusions

In this thesis we have investigated if the method of gradient boosting can compete with the more traditionally used generalized linear model for claims reserving. To evaluate this we used a claims data set for supervised learning and then measured their performance on a holdout test set. We have used root mean squared error as the metric of error since it is suitable for assessing regression models. The results show that the GLM scores an RMSE of 42473 while gradient boosting yields a value of 44371 which suggests that GLM holds a slight advantage in terms of pure predictive capacity. Since it can be difficult to get an intuition of model performance based solely on RMSE we also calculated the relative error i.e. the ratio of the aggregated predicted loss and the actual loss. The relative errors were found as 4% and -13% for gradient boosting and GLM respectively. From this information we can conclude that both models can provide acceptable levels of claim reserves in a real case scenario.

Since the predictive capability of gradient boosting has a proven track record for a wide variety of subjects it would not be unreasonable to expect it to outperform GLM on modelling claims data. However, one possible explanation that the GLM performed better is the relatively small amount of predictor variables used. One of the main drawbacks of GLM is that interaction terms have to be manually included in the model. For more complex models with more features this becomes increasingly difficult but the gradient boosting algorithm handles this automatically. Here we used small number of features that are previously known to have good predictive abilities. Therefore, the strengths of gradient boosting can not be fully utilized. Modelling complex claims where more features are needed is where gradient boosting has a better chance to compete.

Since the discrepancy between the model performances is relatively small we should further evaluate what desirable properties each model holds. One important aspect is how easily the models are implemented. In this respect GLM is clearly favoured. We have shown that a GLM with distribution from the Tweedie family can provide a valuable model after only tuning one hyperparameter. Fitting a satisfying gradient boosting model on the other hand is much more time consuming process. The time spent on fitting the GLM is a matter of minutes while the search for optimal hyperparameters for gradient boosting takes many hours. Hence, from a practical point of view the GLM has a clear advantage, especially if the process has to be iterated for other types of claims as well.

Lastly, we should note that the interpretability of the models are vastly different. To describe how the gradient boosting model goes from input to output is not an easy task why it is often described as a black-box. This is a recurring theme for many statistical learning methods. Neural networks is another popular model which also suffers from this issue. Thus, one should decide carefully if the interpretability problem can be outweighed by other factors. GLM on the other hand provides a model that could easily be described. Since we used a log-link in the fitting process of the GLM we could use exponentiation to get a model where the response is described by multiplicative factors for each predictor variable. This way one can easily interpret how changing the inputs would affect the response variable.

6.1 Future Work

A natural next step for further studies would be consider other types of claims than the ones used in this thesis. This way one could see if gradient boosting would be able to outperform GLM on other types of data. It would also be interesting to consider a richer data set with a large variety of predictor variables to validate whether gradient boosting can handle complex dependencies better than GLM.

In a more extensive study on the subject the focus should be put into improving the fitting of the gradient boosting model, specifically the process of tuning the hyperparameters. In this thesis we used a naive approach by sequentially tuning the hyperparameters. There are more sophisticated methods for optimizing hyperparameters in the model such as *random search* and *Bayesian Optimization* as this would probably yield better models in significantly less time which is a major drawback of the fitting process used for GBM in this thesis. This is a rather comprehensive subject in itself why it was not considered in this study.

References

[1] Gunnar Blom et. al. *Sannolikhetsteori och statistikteori med tillämpningar*. Studentlitteratur AB, 2014.

[2] Marco Aleandri. *Case Reserving in Non-Life Practice using Individual Data and Machine Learning*. URL: `http://www.dss.uniroma1.it/it/system/files/pubblicazioni/NLRes_v3.pdf`. (accessed: 22.05.2018).

[3] Leo Breiman. *Bagging Predictors*. 1994. URL: `https://www.stat.berkeley.edu/~breiman/bagging.pdf`.

[4] Leo Breiman. *Random Forests*. 2001. URL: `https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf`.

[5] Jerome Friedman. *Greedy Function Approximation: A Gradient Boosting Machine*. 1999. URL: `https://statweb.stanford.edu/~jhf/ftp/trebst.pdf`.

[6] Esbjörn Ohlsson Björn Johansson. *Non-Life Insurance Pricing with Generalized Linear Models*. Springer-Verlag, 2010.

[7] John Nelder Peter McCullagh. *Generalized Linear Models*. 1989. URL: `http://www.utstat.toronto.edu/~brunner/oldclass/2201s11/readings/glmbook.pdf`.

[8] Yaron Singer. *Advanced Optimization*. 2016. URL: `https://people.seas.harvard.edu/~yaron/AM221-S16/lecture_notes/AM221_lecture9.pdf`.

[9] Jerome Friedman Trevor Hastie Robert Tibshirani. *The Elements of Statistical Learning*. 2008. URL: `https://web.stanford.edu/~hastie/Papers/ESLII.pdf`.

[10] Heather Turner. *Introduction to Generalized Linear Models*. 2008.

[11] Vladimir Vapnik. *An Overview of Statistical Learning Theory*. 1982. URL: `http://math.arizona.edu/~hzhang/math574m/Read/vapnik.pdf`.

[12] Vladimir Vapnik. *Estimation of Dependencies Based on Empirical Data*. Springer-Verlag, 1982.

TRITA -SCI-GRU 2018:234