University of Tartu
Faculty of Science and Technology
Institute of Mathematics and Statistics

Artur Tuttar

Extending generalized linear models in insurance with machine learning techniques

Actuarial and Financial Engineering
Master's thesis (30 ECTS)

Supervisors: Meelis Käärik (PhD)
Julius Pau (MSc)

Tartu 2023

Extending generalized linear models in insurance with machine learning techniques

Masters's thesis

Artur Tuttar

Abstract. Machine learning models have shown promising results regarding their predictive power. However, little to no information about their use of variables is available. The aim of this thesis is to introduce and put into practice two ways of extracting this insight about variable use. This insight is applied to produce interpretable models that predict in a similar way to underlying machine learning models. The first three chapters give a theoretical overview of methods used to build models and extract insight, and the last two chapters focus on applying these methods to predict claim frequency using real-life insurance data.

Keywords: motor vehicle insurance, generalized linear models, interpretable machine learning.

CERSC research specification: P160 Statistics, operations research, programming, actuarial mathematics.

Üldistatud lineaarsete mudelite edasiarendus kindlustusandmetel masinõppe meetodite abil

Magistritöö

Artur Tuttar

Lühikokkuvõte. Masinõppe mudelid on viimasel ajal silma paistnud oma ennustusvõime poolest. Paraku ei võimalda masinõppe mudelite ülesehitus aru saada, kuidas need mudelid erinevaid tunnuseid kasutavad. See magistritöö tutvustab ja rakendab reaalelu andmetel kaht meetodit, mis püüavad luua masinõppe mudelist interpreteeritavaid mudeleid. Töö kolmes esimeses peatükis antakse teoreetiline ülevaade mudelitest ja meetoditest ning viimases kahes peatükis rakendatakse tuvustatud meetodeid kahjusageduse hindamiseks reaalelu kindlustusandmetel.

Võtmesõnad: sõidukikindlustus, üldistatud lineaarsed mudelid, interpreteeritav masinõpe.

CERCS teaduseriala: P160 Statistika, operatsioonianalüüs, programmeerimine, finants- ja kindlustusmatemaatika.

Contents

Introduction

Insurance companies focus on evaluating risks and providing coverage for them. The price of the coverage should be fair and correspond to the underlying risk. The fair price is usually given through rates applied to a given client. Currently, an interpretable statistical model is built and used to estimate the rates. However, this approach is being challenged by machine learning.

Risk evaluation is usually split into two parts: estimating the number of claims (or claim frequency) and estimating the claim amounts. This thesis focuses on the former. Currently, the main tool for this task is an interpretable generalized linear model (GLM). However, several machine learning algorithms have been shown to outperform this classical approach (Henckaerts et al., 2019; Wüthrich, 2019). These machine learning models are inherently opaque and thus bring value only in their accuracy. Thus, no insight applicable to the ratemaking can easily be extracted from these models.

This thesis aims to introduce and put into practice two ways to extract insights from machine learning models and produce interpretable counterparts to these models. Classic generalized linear models will be compared to machine learning models and corresponding interpretable models by modelling claim frequency for motor third party liability insurance. This thesis is split into five chapters.

The first chapter focuses on the model setup and model structure for generalized linear models. An overview of parameter estimation and claim frequency modelling using Poisson distribution is also provided. The second chapter introduces decision trees and tree-based boosting ensemble methods like gradient boosting machines (gradient tree boosting) and XGBoost. The third chapter focuses on model metrics and introduces two ways to extract insight from machine learning models: maidrr and rule ensemble.

In the fourth chapter, several models are used to predict the claim frequency for Latvian motor third-party liability data, provided by If P&C Insurance AS, including GLMs, gradient boosting machine (GBM) and XGBoost models. General model training procedures are also given. The last chapter introduces models using the insights extracted from the machine learning models developed in Chapter 4, and all models are compared using Poisson deviance and AIC. A small discussion about working on this thesis is also given.

All data manipulation and model training is done using corresponding packages for statistical computation software R (R Core Team, 2022). This thesis was written using Overleaf, an online compiler for the LaTeX typesetting system (Lamport, 1994).

The author would like to thank Julian Trufin and Roel Henckaerts for their correspondence regarding references and suggestions. The author is also extremely grateful for the advice and expertise provided by supervisors Meelis Käärik and Julius Pau. Lastly, the author is grateful for the help of his peers: Joseph Haske, Mihkel Lepson and Nicholas Lupul.

Additionally, this version of the thesis is made publicly available in the spirit of sharing research and showcasing the application of methods developed. However, appendices B, C, D, E, F contain information and results that are considered a trade secret for If P&C Insurance AS and could be used by other businesses besides them to adjust and improve ratemaking and pricing processes. This version of the thesis does not include these appendices; any references and text hyperlinks linking to these appendices have been altered to plain text.

1 Generalized linear models

One of the simplest ways to model a relation between independent variables $(X_1, X_2, \ldots, X_p)$ and response variable (Y) is to assume a linear relation (in terms of parameters) between the independent variables and response variable and fit an ordinary linear regression in the form of

$$\mu_i = \mathbf{E}\left(Y_i\right) \coloneqq \mathbf{E}\left(Y | X_1 = x_{i,1}, X_2 = x_{i,2}, \ldots, X_p = x_{i,p}\right) = \beta_0 + \sum_{j=1}^{p} \beta_j x_{i,j},$$

where $x_{i,j}$ is the realisation of the corresponding independent variable X_j for i-th observation.

In addition to the linearity assumption, ordinary linear regression assumes the normal distribution for the residuals of the model and constant variance for those residuals. Therefore, we have

$$Y_i - \mu_i = \varepsilon_i \sim N(0, \sigma),$$

where σ is constant.

When looking at insurance data, we seldom observe the normal distribution. For example, the number of claims is a positive integer or claim size can only be positive and have heavy tails. Thus normal distribution assumption does not apply and other ways of fitting a relationship between independent and dependent variables are needed.

A step up from ordinary linear regression was proposed in 1972. The generalized linear model (GLM) introduced by Nelder and Wedderburn showed a way to compute maximum likelihood estimates for parameters β_j, $j \in \{0, 1, 2, \ldots, p\}$ for observations conditionally distributed according to some exponential family distributions (Nelder and Wedderburn, 1972). These models remain an insurance industry staple tool to this day because they are simple to understand and easy to interpret.

1.1 Model structure

This subchapter is based on (de Jong and Heller, 2008).

A random variable Y is from the exponential family if the probability density function $f_Y(x)$ is of the form

$$f_Y(x) = c(x, \phi) \exp\left(\frac{x\theta - a(\theta)}{\phi}\right),$$

where θ and ϕ are the canonical and dispersion parameter of the exponential family, respectively. For these distributions, it holds that

$$\begin{aligned} \mathbf{E}(Y) &= a'(\theta), \\ \mathbf{D}(Y) &= \phi a''(\theta), \end{aligned} \tag{1.1}$$

where $\mathbf{E}(\cdot)$ and $\mathbf{D}(\cdot)$ denote the mean and the variance of a random variable, respectively.

The exponential family contains several distributions that are prevalent in insurance, including the exponential distribution, gamma distribution, inverse Gaussian distribution, Poisson distribution, binomial and negative binomial distributions.

The aim of the generalized linear model is the same as the ordinary linear regression model – to describe the response variable (Y) in terms of independent variables $(X_1, X_2, \ldots, X_p)$ and coefficients $(\beta_0, \beta_1, \ldots, \beta_p)$.

However, the models have some key differences. For GLMs, we allow the response variable Y to be from any exponential family, whilst for ordinary linear regression, the response is assumed to be normally distributed. Secondly, for ordinary linear regression, a linear relationship between the independent variables and conditional mean μ_i of the response is modelled, but for GLMs, the transformed conditional mean $g(\mu_i)$ is modelled, where $g(\cdot)$ is called the link function. So for GLMs, we have that

$$g(\mu_i) = \beta_0 + \sum_{j=1}^{p} \beta_j x_{i,j} =: \eta_i,$$

where η_i is called the linear predictor for the i-th observation.

Note that in the next two paragraphs, the i-th index is omitted since a general discussion about the model structure is given.

The link function $g(\cdot)$ acts as the mediator for the linear predictor η and the response variable Y. The choice of appropriate link function is not concrete for every exponential family distribution. Rather, it is dictated by the data and the problem at hand. However, for every exponential family distribution, a canonical link function can be found.

The canonical link function is a link function for which it holds that $g(\mu) = \theta$, where θ is the canonical parameter of the exponential family function. The most common link functions are

- **identity link**: $g(\mu) = \mu$, which is the canonical link of the normal distribution,
- **log-link**: $g(\mu) = \ln(\mu)$, which is the canonical link of the Poisson distribution,

- **power-link** $g(\mu) = \mu^p$, which is the canonical link for gamma distribution if $p = -1$ and inverse Gaussian if $p = -2$,
- **logit-link** $g(\mu) = \ln\left(\frac{\mu}{1-\mu}\right)$, which is the canonical link for binomial distribution.

Usually, the canonical link function is equal to the canonical parameter with respect to some constant. For example, for gamma distribution, we have that $\theta = -\frac{1}{\mu}$, but the canonical link function for this distribution is $g(\mu) = \frac{1}{\mu}$, so the -1 constant is omitted (de Jong and Heller, 2008).

However, sometimes when modelling claim size or frequency, a need to adjust for group size or time period arises. For example, in the case of the number of claims: To estimate the average number of claims in a period of time – claim frequency – the number of claims should be offset (divided) by the exposure time of the policyholder since longer exposure to risk means more time to have claims. In this case, using a log-link function would yield that

$$g\left(\frac{\mu}{n}\right) = \ln\left(\frac{\mu}{n}\right) = \eta \iff \ln(\mu) = \ln(n) + \eta,$$

where variable n is called exposure and $\ln(n)$ is called the offset.

1.2 Modelling using GLM

This subchapter is based on (de Jong and Heller, 2008).

The following steps are done when fitting a GLM:

1. Choose a response distribution with probability density/mass function $f_Y(\cdot)$ from the exponential family. The aim is to choose a response distribution tailored to the situation or modelling problem at hand.
2. Choose a link function $g(\mu)$. As discussed in the previous subchapter, no concrete link function can be given to any single situation as the data and problem at hand dictate the appropriate link function, but a canonical link is a good starting point.
3. Choose independent variables $X_1, X_2, \ldots, X_p$. The choice stems from the problem at hand and can vary based on domain knowledge.
4. Collect the data of the observed values of the response variable $\mathbf{y} = (y_1, y_2, \ldots)^T$ and independent variables $(x_{1,1}, x_{2,1}, \ldots)^T$, $(x_{1,2}, x_{2,2}, \ldots)^T$, $\ldots$, $(x_{1,p}, x_{2,p}, \ldots)^T$ (here $x_{i,j}$ refers to the value of variable X_j for ith observation).

5. Estimate $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_p)^T$ and ϕ (if unknown prior) through maximum likelihood or its variant. A more detailed explanation of this is given in the next subchapter.

6. Assess the fit of the model by examining the predictions of the model and other model diagnostics. The estimates of coefficients also show whether the independent variable helps to determine the value of the conditional mean of the response. This step usually restarts the fitting process as more information on the relation between the response and independent variables is gained.

Usually, the data is already collected and the modelling is aimed at selecting the appropriate distribution, link function and explanatory variables. Fitting the model is done through maximum likelihood methods, which are implemented in most statistical software.

1.3 Parameter estimation

This subchapter is based on (de Jong and Heller, 2008) and (Hardin and Hilbe, 2018). The author has filled in some of the details missing from these references.

The maximum likelihood estimate (MLE) of a parameter θ is such a value of θ that the likelihood of observing a given data is the biggest. We will now show how to get maximum likelihood estimates for the coefficients $(\beta_0, \beta_1, \ldots, \beta_p)$ of the generalized linear model.

Suppose we have a random variable Y belonging to the exponential family. Let $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ be an independant sample from random variable Y. Our aim is to find parameters $\boldsymbol{\beta} = (\beta_0, \beta_1, \beta_2, \ldots, \beta_p)^T$, such that the likelihood of seeing this sample $\mathbf{y}$ is highest. As the realisation of Y in the sample are independent, then the likelihood $L(\boldsymbol{\beta}, \phi, \mathbf{y})$ of getting this sample can be written as the product of the values of the probability density or mass function evaluated at sample observations

$$\mathcal{L} := L(\boldsymbol{\beta}, \phi, \mathbf{y}) = \prod_{i=1}^{n} f(\theta_i, \phi, y_i) = \prod_{i=1}^{n} c(y_i, \phi) \exp\left(\frac{y_i\theta_i - a(\theta_i)}{\phi}\right),$$

here $\theta_i = \theta(\mu_i)$, $\mu_i = \mu(\theta_i)$ and $g(\mu_i) = g(\mu(\theta_i)) = \eta_i$, where $\mu(\cdot)$ and $\theta(\cdot)$ are functions fixed by the distribution (how the mean of distribution relates to the parameter of the distribution), and η_i is the linear predictor for the ith observation. Using this and the fact the logarithm is a monotonically increasing function (then the maximum of the function remains in the same place), we can get the log-likelihood of the sample

$$\ell := l(\boldsymbol{\beta}, \phi, \mathbf{y}) = \ln(L(\boldsymbol{\beta}, \phi, \mathbf{y})) = \sum_{i=1}^{n} \left(\frac{y_i\theta_i - a(\theta_i)}{\phi} + \ln(c(y_i, \phi))\right).$$

Now, to find the set of parameters that maximise the log-likelihood, we have to take the derivative in terms of parameters $\boldsymbol{\beta}$. From this, we get that

$$\frac{\partial \ell}{\partial \boldsymbol{\beta}} = \left(\frac{\partial \ell}{\partial \beta_0}, \frac{\partial \ell}{\partial \beta_1}, \ldots, \frac{\partial \ell}{\partial \beta_p} \right)^T.$$

Now, using the chain rule for derivatives, we have the following expression for each β_j:

$$\frac{\partial \ell}{\partial \beta_j} = \sum_{i=1}^{n} \left(\frac{\partial \ell}{\partial \theta_i} \right) \left(\frac{\partial \theta(\mu_i)}{\partial \mu_i} \right) \left(\frac{\partial g^{-1}(\eta_i)}{\partial \eta_i} \right) \left(\frac{\partial \eta_i}{\partial \beta_j} \right). \tag{1.2}$$

Taking the partial derivatives of ℓ with respect to θ_i, we get

$$\frac{\partial \ell}{\partial \theta_i} = \frac{y_i - a'(\theta_i)}{\phi}, \tag{1.3}$$

for all observations $i = 1, \ldots, n$.

Now, since $\mu_i = \mathbf{E}(Y_i)$, then taking derivative of (1.1) with respect to the θ_i, we get

$$a''(\theta_i) = \frac{\partial \mu_i}{\partial \theta_i}. \tag{1.4}$$

Using Formula (1.4) and the fact that $\theta_i = \theta(\mu_i)$ we get that

$$\frac{\partial \theta(\mu_i)}{\partial \mu_i} = \frac{1}{a''(\theta_i)}. \tag{1.5}$$

Lastly, for a monotone link function $g(\mu) = \eta$, there exists inverse function $g^{-1}(\cdot)$ and then, we get that

$$\begin{aligned} \mu_i &= g^{-1}(\eta_i) \\ \frac{\partial \eta_i}{\partial \beta_j} &= x_{i,j}, \end{aligned} \tag{1.6}$$

for all observations $(i = 1, \ldots, n)$ in the sample.

Using Formulas (1.3), (1.1), (1.5) and (1.6) in Formula (1.2), we get that

$$\begin{aligned} \frac{\partial \ell}{\partial \beta_j} &= \sum_{i=1}^{n} \left(\frac{y_i - a'(\theta_i)}{\phi} \right) \left(\frac{1}{a''(\theta_i)} \right) \left(\frac{\partial g^{-1}(\eta_i)}{\partial \eta_i} \right) x_{i,j} \\ &= \sum_{i=1}^{n} \left(\frac{y_i - \mu_i}{\phi a''(\theta_i)} \right) \left(\frac{\partial g^{-1}(\eta_i)}{\partial \eta_i} \right) x_{i,j}, \end{aligned}$$

for all $j = 0, \ldots, p$. Additionally, if the model includes the intercept term (β_0) then $x_{i,0} = 1$ for all $i = 1, \ldots, n$, otherwise $x_{i,0} = 0$. Note that here $\frac{\partial g^{-1}(\eta_i)}{\partial \eta_i}$ is the derivative of the inverse of the link function with respect to linear predictor η_i.

Now solving this set of derivative equations

$$\frac{\partial \ell}{\partial \boldsymbol{\beta}} = \mathbf{0},$$

gives us an estimate for the unknown coefficients $\hat{\boldsymbol{\beta}}$. Since an analytical solution is difficult to find, an approximation method, such as the Newton-Rapshon method, can be used.

1.4 Count and frequency data modelling

This subchapter is based on (Hardin and Hilbe, 2018).

In this work, the practical part will focus on estimating claim frequency from the data. This will, in part, be done using GLM models with Poisson distribution. A short overview and details to keep in mind when dealing with Poisson distribution are provided below.

The Poisson distribution is a discrete distribution which is often used to describe counts. The probability mass function for Poisson distribution is

$$f(y, \xi) = \frac{e^{-\xi}\xi^y}{y!} = \frac{1}{y!}\exp\left(y\ln(\xi) - \xi\right),$$

where ξ is a parameter with an intuitive meaning of frequency at which the event that is being tracked happens.

From this form, we can clearly see that it is a part of the exponential family since we can choose $\theta = \ln(\xi)$, $a(\theta) = \xi = \exp(\theta)$, $\phi = 1$ according to the definition of the exponential family. A distinct property of Poisson distribution is that the expected value $\mathbf{E}(Y)$ and variance $\mathbf{D}(Y)$ are equal ($\mathbf{E}(Y) = a'(\theta) = \xi$, $\mathbf{D}(Y) = \phi a''(\theta) = \xi$). This is one of the most important properties to check when modelling with the Poisson distribution. If $\mathbf{D}(Y) < \mathbf{E}(Y)$, then there is underdispersion in the data. Although not ideal, this is rarely taken into account when modelling. If $\mathbf{D}(Y) > \mathbf{E}(Y)$, then there is overdispersion in the data and usage of regular Poisson distribution might lead to wrong conclusions. Other models like negative binomial or quasi Poisson can be used in this case.

2 Tree models

Most machine learning techniques are non-parametric and usually quite complex "black-box" methods, where data goes in and predictions come out. However, decision trees are not a black-box technique. In fact, they produce an easily interpretable model.

A decision tree is a non-parametric supervised learning method which tries to predict the response variable through a set of splits. These splits can be easily interpreted since they try to segment the data into homogeneous subsets. A chain of these splits is called a rule and can easily showcase a collection of data points that have somewhat similar independent variables and the response variable (Molnar, 2022).

Decision trees are used as the key method for several more advanced machine learning methods. These methods aim to use the predictive power of a tree but lower the variance of the trees through ensembling. The ensembles use the decision trees as base learners and build a structure containing these trees to augment the predicting power of a single tree. These tree-based ensemble methods can be classified into two groups: boosting and bagging tree ensembles (Hastie, Tibshirani, and Friedman, 2009).

In this chapter, we will first introduce a decision tree building method CART (Classification And Regression Trees), and then showcase a way these trees can be used as an ensemble to fix some of the shortcomings of a single decision tree. This work focuses on boosting ensembles of trees through gradient boosting and its further development XGBoost (eXtreme Gradient Boosting).

2.1 Decision trees

This subchapter is based on (Hastie, Tibshirani, and Friedman, 2009).

To understand how decision trees work, we will first start with a tree-growing algorithm with two independent variables. Then, we will generalise the algorithm to any number of variables. The splitting for both categorical and numeric variables will be discussed, but only the regression problem is explained, as this is the core problem of this work.

2.1.1 Data partitioning

Suppose we have a continuous response variable Y and two independent variables X_1 and X_2 taking values in the unit interval. Our aim is to partition the space $\mathbf{X} := \mathbf{X}_{\{1\}} \times \mathbf{X}_{\{2\}}$ (where $\mathbf{X}_{\{j\}}$ is the set of possible values X_j can take; in this case, $\mathbf{X}_{\{1\}} = \mathbf{X}_{\{2\}} = [0, 1]$) into regions,

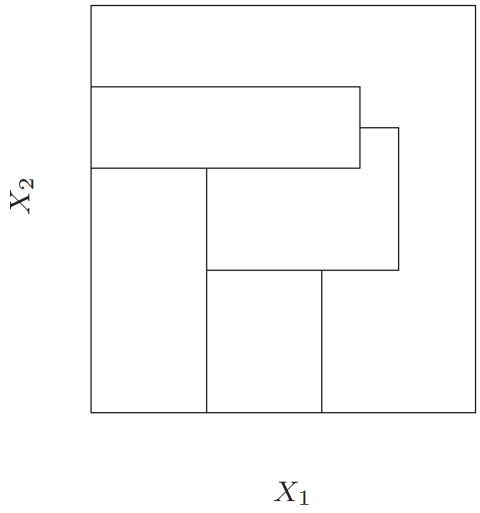

Figure 1: CART "general" partitioning using straight line boundaries. Taken from (Hastie, Tibshirani, and Friedman, 2009), Figure 9.2.

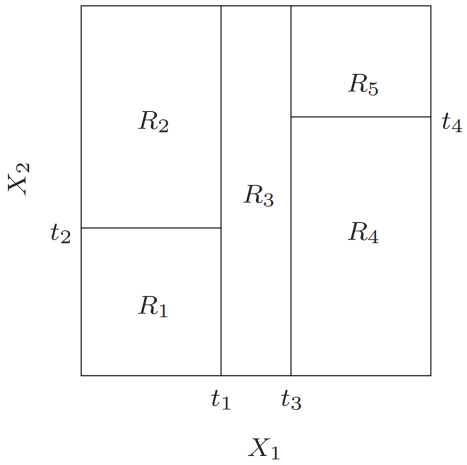

Figure 2: CART recursive binary partitioning. Taken from (Hastie, Tibshirani, and Friedman, 2009), Figure 9.2.

so the predicted response $\hat{y}_i$ in a given region is close to the actual response value y_i. There are infinitely many and infinitely complex ways of splitting the space $\mathbf{X}$ up into such regions. Decision trees use splits done using straight lines like $x_j = c$. However, even using straight lines, the partitioned regions can be hard to describe. An example of a "general" partitioning is given in Figure 1.

To solve this, a restriction is put in place. Each split has to divide the space or subspace into two distinct parts minimising the loss with respect to the response. This way of splitting is called recursive binary split. The algorithm for recursive binary splitting is quite simple. At the start, all possible splits of space $\mathbf{X}$ are considered. Since each region predicts a constant value – usually region average – as the prediction for the response, the best split minimising the loss to the response value Y is adopted, and the space is divided along this split. Now two separate regions (subspaces) of $\mathbf{X}$ are formed, usually denoted as R_1 and R_2. However, as more splits are done, the indices are renumbered for all regions. The previous step is repeated for the new regions, and again the "best" split is chosen. The splitting takes place until some stopping rule is triggered. For example, the maximum number of regions is reached.

In Figure 2, an example of a two-dimensional binary recursive split is given. First, a split along X_1 at value $x_1 = t_1$ is done. This results in two regions $R_1 = \{(x_1, x_2) \in \mathbf{X} | x_1 \leq t_1\}$ and $R_2 = \{(x_1, x_2) \in \mathbf{X} | x_1 > t_1\}$. The next split is done in region R_1 along variable X_2 at value $x_2 = t_2$ producing two new regions $R_1 = \{(x_1, x_2) \in \mathbf{X} | x_1 \leq t_1, x_2 \leq t_2\}$ and

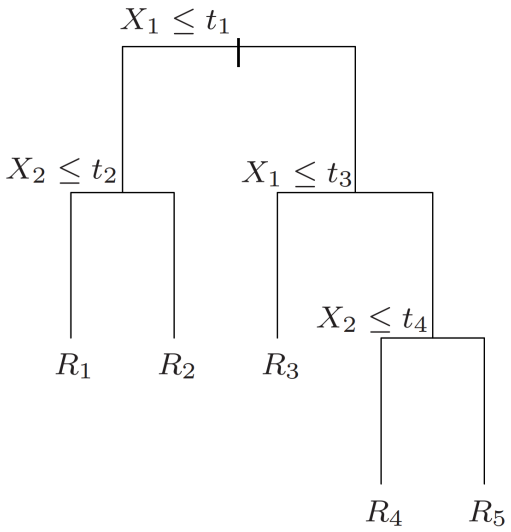

Figure 3: CART partitioning as a recursive binary tree. Taken from (Hastie, Tibshirani, and Friedman, 2009), Figure 9.2.

$R_2 = \{(x_1, x_2) \in \mathbf{X} | x_1 \leq t_1, x_2 > t_2\}$ and previous R_2 region becomes R_3. After this, region R_3 is split along X_1 at $x_1 = t_3$ resulting in regions $R_3 = \{(x_1, x_2) \in \mathbf{X} | x_1 > t_1, x_1 \leq t_3\}$ and $R_4 = \{(x_1, x_2) \in \mathbf{X} | x_1 > t_1, x_1 > t_3\}$. Lastly, region R_4 is split along variable X_2 at value $x_2 = t_4$, where resulting regions are $R_4 = \{(x_1, x_2) \in \mathbf{X} | x_1 > t_1, x_1 > t_3, x_2 \leq t_4\}$ and $R_5 = \{(x_1, x_2) \in \mathbf{X} | x_1 > t_1, x_1 > t_3, x_2 > t_4\}$.

The resulting regression model for Y can be written as

$$\hat{y} = f_{\text{tree}}(\mathbf{x}) = \sum_{m=1}^{5} c_m I[\mathbf{x} \in R_m], \tag{2.1}$$

where c_m denotes the predicted value for the region R_m and $\mathbf{x} = (x_1, x_2)$ or the vector of realised values of variables X_1 and X_2.

The dimensional partitioning plots work well for one and two-dimensional data. However, for higher dimensional data, the space partitioning is not very informative. Another way to describe this partitioning is by using a binary tree. The same two-dimensional example as for Figure 2 is represented as a binary tree in Figure 3. A splitting condition is displayed at each tree node; if the observations abide by this condition, they go to the left child node. Otherwise, the observation goes to the right child node. The tree plot also generalises well to higher dimensions since, usually, at each node, only one variable is inspected.

2.1.2 Regression trees

In the previous subchapter, we looked into how decision trees form the regression model by partitioning the data space into regions. However, the choice of best split and, in general, the growing of trees was not considered in detail. In this subchapter, we will remedy that.

We are considering a regression problem for the response variable Y. Let now our data contain p independent variables $X_1, X_2, \ldots, X_p$ for n observations. The partitioning algorithm described in the previous subchapter should be able to decide what splitting variable and split point to use automatically.

One way to do this would be to consider the regression model with M regions $R_1, R_2, \ldots, R_M$ with constant predictions c_m in each region

$$f(\mathbf{x}) = \sum_{m=1}^{M} c_m I(\mathbf{x} \in R_m),$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_p)$. Using this regression, we choose a loss function and minimise it. The choice of this function should stem from the problem at hand. When dealing with count data, the Poisson deviance or Poisson log loss function should be minimised. As a simple example, take the loss function to be the sum of squares, $\sum(y_i - f(\mathbf{x}_i))^2$. Using this loss function, it's easy to see that for any region R_m, the best prediction is the average of the response variable in the region R_m, $\hat{c}_m = \text{avg}(y_i|\mathbf{x}_i \in R_m)$.

But now we still have the issue of choosing the regions, as searching for the best binary partition with minimum possible loss is generally computationally infeasible. A greedy algorithm is used to choose the splitting variable X_j and splitting value s. We want to partition space $\mathbf{X} = \mathbf{X}_{\{1\}} \times \mathbf{X}_{\{2\}} \times \cdots \times \mathbf{X}_{\{p\}}$, where $\mathbf{X}_{\{j\}}$ is the set of values variable X_j can have. Let us denote the regions corresponding to split along variable X_j at splitting value s as $R_{m_1}(j, s)$ and $R_{m_2}(j, s)$ (at the start of fitting, region indices m_1 and m_2 are not fully determined as further splits might change them). Then we can write that

$$R_{m_1}(j, s) = \{(x_1, \ldots, x_p) \in \mathbf{X} | x_j \leq s\} \text{ and } R_{m_2}(j, s) = \{(x_1, \ldots, x_p) \in \mathbf{X} | x_j > s\},$$

if variable X_j is a numeric variable and then $s \in \mathbb{R}$ or

$$R_{m_1}(j, s) = \{(x_1, \ldots, x_p) \in \mathbf{X} | x_j \in s\} \text{ and } R_{m_2}(j, s) = \{(x_1, \ldots, x_p) \in \mathbf{X} | x_j \notin s\},$$

if variable X_j is categorical then s is a subset of values categorical variable X_j can take or, in

other words, $s \subset \mathbf{X}_{\{j\}}$. In both cases, we solve a minimisation problem

$$\min_{j,s} \left[\sum_{i,\mathbf{x}_i \in R_{m_1}(j,s)} L(y_i, \hat{c}_{m_1}) + \sum_{i,\mathbf{x}_i \in R_{m_2}(j,s)} L(y_i, \hat{c}_{m_2}) \right],$$

where $L(y, c)$ is a loss function and $\hat{c}_m$ is the solution to minimisation problem

$$\hat{c}_m = \arg\min_c \sum_{i,\mathbf{x}_i \in R_m} L(y_i, c), \tag{2.2}$$

where $m \in \{m_1, m_2\}$. In case of sum of squares, $L(y, c) = (y - c)^2$ and $\hat{c}_m = \text{avg}(y_i | \mathbf{x}_i \in R_m)$.

Finding the best splitting variable X_j and splitting point s, we adopt this split on the data and get two new regions from this. The same procedure is repeated on the resulting regions. This process, if left unchecked, can lead to severe overfitting since as the tree grows in size (depth), the tree relies on smaller and smaller subsets of the data. Overfitting leads to making predictions that most likely capture the local effect of the independent variables on the response. The tree structure, in that case, is too granular.

One way to combat overfitting is by introducing a complexity parameter. This parameter aims to find the optimal tree size based on observed data. One of the possible complexity parameters is the minimum decrease in the loss for a split. So if no split lowers the loss by the amount fixed by this parameter, no further splits are done. This parameter has the downside of being short-sighted since sometimes a seemingly useless split might lead to a good split down the line.

Another possible complexity parameter is cost complexity. In this case, we build the decision tree to its highest depth, stopping when we reach a stopping rule like minimum node size. This deep tree T_0 is then pruned (internal nodes are collapsed) based on the value of this cost complexity. Let T be a subtree of T_0 obtainable through pruning. Then define

$$N_m(T) = \sum_{i,\mathbf{x}_i \in R_m} 1 \qquad \text{(node support)},$$

$$Q_m(T) = \frac{1}{N_m(T)} \sum_{i,\mathbf{x}_i \in R_m} L(y_i, \hat{c}_m) \qquad \text{(node complexity)}.$$

Now we can define the cost complexity of tree T as

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m(T) Q_m(T) + \alpha |T|,$$

where $|T|$ is the number of leafs in the tree T. Now the parameter α acts as a tradeoff between the tree size and its associated goodness of fit. It can be shown that through pruning T_0, a unique

small subtree T_α that minimises $C_\alpha(T)$ exists and can be found through weakest link pruning. That is, collapsing a node that produces the smallest per-node increase in $\sum_m N_m(T)Q_m(T)$.

2.1.3 Advantages and disadvantages of trees

This subchapter is based on (Hastie, Tibshirani, and Friedman, 2009), (Molnar, 2022) and a scikit-learn post (*1.10. Decision Trees* 2023).

In the previous two subchapters, a brief overview of the CART decision tree building algorithm was given. Like with any model, this model also has its advantages and disadvantages.

In the case of the CART model, the decision trees are easy to visualise and, thanks to the building process, also easy to interpret. For CART models, the categorical variables can be used without any encoding (like dummy encoding) since CART uses a subset of variable levels to build the splitting condition. Modelling the data set with rules allows for easy interaction modelling.

However, decision trees without proper pruning can easily overfit to the training data. Decision trees are also highly unstable since a small change in the dataset might lead to a different tree altogether. Although the piecewise constant approximation can, in theory, approximate any function, to approximate linare relation a lot of splits are required. Finding a single globally optimal tree can be computationally infeasible.

2.2 Boosting

The previous chapter showcased a powerful predictive method, but it had some shortcomings. To combat some of the disadvantages present in the decision tree algorithm – chiefly the instability – and to augment the predictive power of a single predictor, an ensemble of predictors can be used. An ensemble is a collection of weak learners, such as low depth decision trees, all trained and arranged in a structure to lower the variance of predictions.

Although an ensemble of any methods is possible, we will look at decision tree ensembles. Two strategies are commonly employed for creating decision tree ensembles: bagging and boosting. In the case of bagging, a lot of decision trees are built on bootstrapped samples of the training data and the predictions of each tree are averaged to get an overall prediction. In boosting, weak learners are used to learn from the predictions of the previous iteration sequentially. This work focuses on boosting methods for decision trees.

Boosting can be done in several ways, like learning from prediction errors (Adaboost), gradient descent or gradient boosting. In general, we can say that for boosting, we are searching for prediction function $F(\mathbf{x})$ in the form of an additive expansion

$$F(\mathbf{x}) = \beta_0 + \sum_{t=1}^{\mathcal{T}} \beta_t f_t(\mathbf{x}),$$

where $f_t(\mathbf{x})$ is called the base learner, β_t is called the expansion coefficient and $\mathcal{T}$ is the number of learners or iterations. Ultimately, we want to minimise the within-sample loss and find all of the base learners such that the sum of losses in the sample is minimal. In other words, we want to find

$$F^*(\mathbf{x}) = \arg\min_{F(\mathbf{x})} \mathcal{L}(F),$$

where $\mathcal{L}(F) = \sum_{i=1}^{n} L(y_i, F(\mathbf{x}_i))$. This can seldom be done all at once so usually an iterative process is used instead. The following subchapters will describe a boosting technique called gradient boosting in greater detail.

2.2.1 Gradient descent

This subchapter is based on (Friedman, 2001).

In order to understand gradient boosting, we first need to have an overview of gradient descent. Since machine learning algorithms are made to optimise some loss or error function, then a way to find a global minimum is needed. Since minimisation requires us to find partial derivatives, set them to 0 and solve the equation, then usually, the equations tend to be quite complex. Gradient descent aims to approximate the solution to this kind of problem.

The idea of gradient descent is simple. First, we take a random guess of what a parameter, in terms of which we want to optimise, is. Next, calculate the negative gradient and update the chosen parameter values so the function value is smaller, and hopefully closer to the global minimum.

In our case, consider the single observation loss function $L(y_i, \hat{y}_i)$, where $\hat{y}_i = F(\mathbf{x}_i)$ is the prediction for ith observation ($i = 1, \ldots, n$). Denote the sum of loss for the whole dataset as

$$\mathcal{L}(F) \coloneqq \sum_{i=1}^{n} L(y_i, F(\mathbf{x}_i)).$$

We want to find the function F^* that minimises the sum of losses $\mathcal{L}$. Note that in the upcoming discussion the subscript, i from $\mathbf{x}_i$, is omitted since the gradient is calculated and adjusted for each data point separately.

Let us search for the function F^* in the form of

$$F^*(\mathbf{x}) = \sum_{t=0}^{\mathcal{T}} f_t(\mathbf{x}),$$

where $f_0(\mathbf{x})$ is a fixed initial guess and $f_t(\mathbf{x}),\ 0 < t \leq \mathcal{T}$ are the incremental improvements. Let $F_\tau(\mathbf{x}) = \sum_{t=0}^{\tau} f_t(\mathbf{x})$ be the incremental prediction function at step τ. Define then

$$f_t(\mathbf{x}) = -\rho_t g_t(\mathbf{x}),$$

where $\rho_t \in \mathbb{R}$ is the learning rate and

$$g_t(\mathbf{x}) = \frac{\partial L}{\partial \hat{y}}(y, F_{\tau-1}(\mathbf{x}))$$

is the gradient of the loss function evaluated at the previous incremental step $F_{\tau-1}$.

The learning rate ρ_t can be given in several forms. A constant learning rate $\rho_t = \rho \in \mathbb{R}, \forall t \geq 0$ is one of the most widely adopted forms in software, but setting the constant value too low might lead to very slow convergence while setting it too high might lead to a method that does not converge or does not converge to the global minimum.

Another way to fix the learning rate is to reduce it exponentially with each iteration. In this case, you would start with a constant learning rate $\rho_0 = \rho$ and at each step t you reduce it by some coefficient $\alpha \in (0, 1)$. So we get the generic formula $\rho_t = \rho\alpha^t$.

Overall, we can write that the resulting approximation to the solution function $F^*(\mathbf{x})$ is in the form of

$$F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) - \rho_t g_t(\mathbf{x}).$$

As each step reduces the gradient by some amount, a way to stop the process is required. The usual stopping rules are a maximum number of boosting iteration $\mathcal{T}$ or a threshold for the size of incremental step $f_t(\mathbf{x})$.

2.2.2 Gradient Boosting

This subchapter is based on (Friedman, 2002).

The setup for gradient boosting is the same as for gradient descent. Namely we want to find a function $F^*(\mathbf{x})$ such that sum of loss functions $\mathcal{L}(F^*) = \sum_{i=1}^{n} L(y_i, \hat{y}_i^*)$ is minimised, where $\hat{y}_i^* = F^*(\mathbf{x}_i)$. So, in other words

$$F^*(\mathbf{x}) = \arg\min_{F(\mathbf{x})} \mathcal{L}(F).$$

For gradient descent, the loss function for each observation was sequentially optimised by the gradient produced on the loss function. Gradient boosting takes another approach again using an additive expansion, but in this case, we search for the function $F(\mathbf{x})$ in the form of

$$F(\mathbf{x}) = \sum_{t=0}^{\mathcal{T}} \beta_t h(\mathbf{x}, \mathbf{a}_t),$$

where $h(\mathbf{x}, \mathbf{a})$, is a simple, weak learner function of independent variables $\mathbf{x}$ with parameters $\mathbf{a} = (a_1, a_2, \dots)$. The expansion coefficients β_t and function parameters $\mathbf{a}_t$ are derived from the training data in a stage-wise manner. Again an initial guess for the parameters is given at step $t = 0$ and then, at each iteration step $t = 1, 2, \dots, \mathcal{T}$, the next set of parameters is found as a solution to

$$(\beta_t, \mathbf{a}_t) = \arg\min_{\beta, \mathbf{a}} \mathcal{L}(F_{t-1}(\mathbf{x}) + \beta h(\mathbf{x}, \mathbf{a})). \tag{2.3}$$

The solution to this function is found through a two-step procedure. First, a least squares approach is used to estimate the parameters of the weak learner,

$$\mathbf{a}_t = \arg\min_{\mathbf{a}, \rho} \sum_{i=1}^{n} [\tilde{y}_{i,t} - \rho h(\mathbf{x}_i, \mathbf{a})]^2,$$

where $\tilde{y}_{i,t}$ is the negative gradient evaluated at the previous iteration,

$$\tilde{y}_{i,t} = -\frac{\partial L}{\partial \hat{y}}(y_i, F_{t-1}(\mathbf{x})),$$

and $\rho \in \mathbb{R}$ is the weight of the given base learner in the ensemble.

Secondly, the resulting optimal $\mathbf{a}_t$ is then used in (2.3) to form a one parameter optimisation problem

$$\beta_t = \arg\min_{\beta} \mathcal{L}(F_{t-1}(\mathbf{x}) + \beta h(\mathbf{x}, \mathbf{a}_t)). \tag{2.4}$$

Gradient tree boosting (gradient boosting machine) is a special case of the method described above. It fixes the structure of a weak learner to be a tree with M terminal nodes splitting the data into regions $R_1, R_2, \dots, R_M$. As specified in the previous subchapter, the regression tree predicts a constant $\hat{c}_{m,t}$ for the whole region that is the solution to (2.2). Then we can write

$$h(\mathbf{x}, \{R_{m,t}\}_{m=1}^{M}) = \sum_{m=1}^{M} \hat{c}_{m,t} I(\mathbf{x} \in R_{m,t}),$$

where $R_{m,t}$ is the m-th region of the t-th decision tree (our weak learner). Note that in this case, the model parameters would be the set of splitting variables with indices $\mathbf{j} \subset \{1, 2, \dots, p\}$ and

their split points $\mathbf{s}$, which can be calculated in the same manner as discussed previously. These parameters are enough to determine the corresponding regions $R_{m,t}$ for t-th iteration.

As the regions are disjointed, then the solution to (2.4) simplifies to a location estimation problem within each region $R_{m,t}$ for loss L

$$\gamma_{m,t} = \arg\min_{\gamma} \sum_{i,\mathbf{x}_i \in R_{m,t}} L(y_i, F_{t-1}(\mathbf{x}_i) + \gamma).$$

The updated prediction function for the t-th iteration in region $R_{m,t}$ then becomes

$$F_{m,t}(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \nu \cdot \gamma_{m,t} I(\mathbf{x} \in R_{m,t}),$$

where $\nu \in (0, 1]$ acts similarly as the learning rate ρ from gradient descent, fixing the amount a given tree can learn from its previous errors. Putting all of these steps together gives us the algorithm for generalized boosting with decision trees (Algorithm 1).

Algorithm 1 Generalized gradient tree boosting

$F_0(\mathbf{x}) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$
for $t = 1$ to $\mathcal{T}$ **do**
  $\tilde{y}_{i,t} = -\frac{\partial L}{\partial \hat{y}}(y_i, F_{t-1}(\mathbf{x})),\ i = 1, \ldots, n$
  Find regions $R_{1,t}, \ldots, R_{M,t}$ for decision tree with M terminal nodes using
  $\tilde{\mathbf{y}}_t = (\tilde{y}_{1,t}, \ldots, \tilde{y}_{n,t})$ as the vector of response values
  $\gamma_{m,t} = \arg\min_{\gamma} \sum_{i,\mathbf{x}_i \in R_{m,t}} L(y_i, F_{t-1}(\mathbf{x}_i) + \gamma)$, for $m = 1, \ldots, M$
  $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \nu \cdot \sum_{m=1}^{M} \gamma_{m,t} I(\mathbf{x} \in R_{m,t})$
end for

This algorithm was first introduced by Friedman in 1999 and has since been used and studied heavily (Friedman, 2001). The original algorithm was somewhat computationally and memory intensive, depending on the selected decision tree. But over the years, several updates to the algorithm have been made.

One of these updates is inspired by the work of Breiman on "bagging" (Breiman, 1996). Breiman proposed a mix of bagging and boosting in 1999 (Breiman, 1999). This would include a random sample of full training data as the starting point of the decision tree building and out-of-bag residuals for boosting steps. Friedman adopted these ideas and updated the gradient boosting algorithm (Algorithm 1) accordingly, calling the new algorithm "stochastic gradient boosting". The full algorithm is available in (Friedman, 2002). This allowed for a reduction in computation steps and memory usage proportional to the sampling fraction used.

2.2.3 XGBoost

This subchapter is based on (Chen and Guestrin, 2016).

A bigger set of upgrades to gradient boosting was introduced in (Chen and Guestrin, 2016), with the introduction of XGBoost. XGBoost can be considered an overhaul of the idea and algorithm proposed by Friedman. Chen and Guestrin modernised the method by optimising the tree-building process. The proposed upgrades stem from two sides, the underlying mathematical and algorithmic changes and hardware-related optimisation.

The notation for this chapter closely follows the notation presented in the previous chapters, where $F_t(\mathbf{x})$ is the prediction function for gradient tree boosting, c_m is the prediction for the region R_m, $L(y, \hat{y})$ is the chosen loss function and M is the number of terminal nodes.

A list of mathematical and algorithmic improvements proposed is given below.

- **The regularisation of the weak learner in the objective function.** When building trees, in addition to a differentiable convex loss function $L(y, \hat{y})$, a penalty term is included. The objective function, in this case, is

$$\mathcal{L}(F(\mathbf{x})) = \sum_{i=1}^{n} L(y_i, \hat{y}_i) + \sum_{t=1}^{\mathcal{T}} \Omega(F_t),$$

  where $\hat{y}_i = F(\mathbf{x}_i)$ and the penalty function is defined as

$$\Omega(F_t) = \kappa M + \frac{1}{2}\Lambda \sum_{m=1}^{M} \hat{c}_m^2,$$

  with penalty parameters κ and Λ .

- **Using a Taylor series approximation for the objective function**. The objective function for the tree for iteration t (denoted T_t) is defined as

$$\mathcal{L}_t = \sum_{i=1}^{n} L(y_i, F_{t-1}(\mathbf{x}_i) + F_t(\mathbf{x}_i)) + \Omega(F_t),$$

  but a second-order Taylor approximation can be used to optimise the new objective quickly

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} [L(y_i, F_{t-1}(\mathbf{x}_i)) + g_i F_t(\mathbf{x}_i) + \frac{1}{2} h_i F_t^2(\mathbf{x}_i)] + \Omega(F_t),$$

  where

$$g_i = \frac{\partial L}{\partial \hat{y}}(y_i, F_{t-1}(\mathbf{x}_i)) \text{ and } h_i = \frac{\partial^2 L}{(\partial \hat{y})^2}(y_i, F_{t-1}(\mathbf{x}_i)). \tag{2.5}$$

Since $L(y_i, F_{t-1}(\mathbf{x}_i))$ is a constant in terms of $F_t(\mathbf{x})$, it can be omitted from the objective function. Additionally, substituting $\Omega(F_t)$ and rearranging the sums we then get

$$\tilde{\mathcal{L}}^{(t)} = \sum_{m=1}^{M} \left[\left(\sum_{i,\mathbf{x}_i \in R_{m,t}} g_i \right) \hat{c}_{m,t} + \frac{1}{2} \left(\sum_{i,\mathbf{x}_i \in R_{m,t}} h_i + \Lambda \right) \hat{c}_{m,t}^2 \right] + \kappa M,$$

where $R_{m,t}$ is the m-th region from t-th iteration tree.

From this, we can get that the optimal leaf prediction $\hat{c}^*_{m,t}$ should approximately be

$$\hat{c}^*_{m,t} \approx -\frac{\sum\limits_{i,\mathbf{x}_i \in R_{m,t}} g_i}{\sum\limits_{i,\mathbf{x}_i \in R_{m,t}} h_i + \Lambda}.$$

From this, a new scoring function for a tree T_t is proposed using the approximated objective function $\tilde{\mathcal{L}}^{(t)}$ and the optimal leaf prediction $\hat{c}^*_{m,t}$

$$\mathcal{L}^{(t)}(T_t) = -\frac{1}{2} \sum_{m=1}^{M} \frac{\left(\sum\limits_{i,\mathbf{x}_i \in R_{m,t}} g_i \right)^2}{\sum\limits_{i,\mathbf{x}_i \in R_{m,t}} h_i + \Lambda} + \kappa M.$$

To evaluate a split, a greedy algorithm starts from a single leaf and adds branches to the tree. The objective function for a split can be written as

$$\mathcal{L}_{split} = \frac{1}{2} \left[\frac{\left(\sum\limits_{i,\mathbf{x}_i \in R_R} g_i \right)^2}{\sum\limits_{i,\mathbf{x}_i \in R_R} h_i + \Lambda} + \frac{\left(\sum\limits_{i,\mathbf{x}_i \in R_L} g_i \right)^2}{\sum\limits_{i,\mathbf{x}_i \in R_L} h_i + \Lambda} - \frac{\left(\sum\limits_{i,\mathbf{x}_i \in R_R \bigcup R_L} g_i \right)^2}{\sum\limits_{i,\mathbf{x}_i \in R_R \bigcup R_L} h_i + \Lambda} \right] - \kappa,$$

where R_R and R_L are the corresponding right and left regions for a considered split. $(R_R \bigcup R_L)$ is then the original leaf the split was considered for.

- **Column subsampling is implemented.** To combat overfitting, a given tree is allowed to choose only from a random subset of available columns to do a split with. This forces the model to sometimes build a tree with variables it would otherwise never use.

- **Approximate splitting algorithm.** For generic gradient boosting, to find the best splitting value s for the independent variable X_j, all possible splits are considered. This

is a called an exact greedy algorithm, and it is computationally demanding as efficiency is tied to the available memory. Instead of the exact search, an approximation – percentiles of variable values – of the possible candidate values is produced and evaluated. XGBoost has two variants of this algorithm: the global variant makes all of the candidate splits at the start of fitting and the local variant that re-proposes the candidates after each split.

- **A weighted quantile sketch is used.** This allows the subsetting of the independent variable values using the second-order gradient values h_i from Formula (2.5) for each observation. Let $\mathbf{D}_j = \{(x_{1,j}, h_1), (x_{2,j}, h_2), \ldots, (x_{n,j}, h_n)\}$. This denotes the pair of values for the variable X_j and second-order gradient h_i in the training set. Now, define

$$r_j(z) = \frac{1}{\sum_{i,(x_{i,j},h_i)\in\mathbf{D}_j} h_i} \sum_{i,(x_{i,j},h_i)\in\mathbf{D}_j, x<z} h_i,$$

  which represents the proportion of the sum of second-order gradients h_i for instances where the value of variable X_j is less than z. The aim is to find such candidates $(s_{j,1}, s_{j,2}, \ldots, s_{j,l})$ that satisfy

$$|r_j(s_{j,k}) - r_j(s_{j,k+1})| < \varepsilon, \ \forall k = 1, \ldots, l-1,$$

  where $\frac{1}{\varepsilon} \approx l$ (maximum number of splits considered). Note that $s_{j,1} = \min_i x_{i,j}$ and $s_{j,l} = \max_i x_{i,j}$.

  This complements the objective function since the values of h_i can then be considered as weights

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \frac{1}{2} h_i \left(F_t(\mathbf{x}_i) - \frac{g_i}{h_i} \right)^2 + \Omega(F_t) + \text{constant},$$

  giving us the weighted squared loss with the label $\frac{g_i}{h_i}$ and weights h_i

- **The fitting procedure is sparsity aware.** Real-world data is often sparse due to missing data, frequent zero entries or variable encoding. This kind of missing data is not useful when building trees but still has to be assigned to a node in a tree. To do this, a default direction for sparse data is used where the optimal direction is learnt from the data.

In addition to the mathematical and algorithmic improvements, some hardware optimisations are also implemented.

- **Column blocks are used.** Column value sorting and learning is one of the most memory-intense parts of the algorithm. For an exact greedy search, all values of the column have

to be read in memory, potentially requiring a lot of memory in the case of large datasets. To alleviate this, the column data is stored and processed in "blocks" since the augmented weighted quantile function allows for the processing of the column in parts. The usage of column blocks also allows for parallel computation of these blocks, speeding up the search for optimal splits.

- **The algorithm is cache-aware.** The column blocks allowed for parallel computation for split finding, but this results in non-continuous memory access to retrieve row-specific gradient statistics. Non-continuous memory access slows down the program since the read/write cycle of other computations would need to be queued. The solution to this is easy – using an appropriately sized column block that allows the gradient statistics to be fit in the CPU cache.

- **Blocks for Out-of-core computations to effectively use the machine's resources.** A block reading and writing system is used when the data does not fit into the main memory. All the data is stored on disk in blocks and accessed sequentially. This is, however, limited by the disk read and write speeds. To speed this up, the stored data blocks are compressed and decompressed on the fly, exchanging the disk usage for CPU usage. Additionally, the data blocks are stored on multiple disks (where available), and different threads of the CPU are assigned to access the data allowing for more streamlined read-and-write sequencing.

All of these augmentations combine to make the methods an order of magnitude faster, allowing for excellent out-of-core and distributed system performance with basically no predictive performance drawbacks.

3 Machine learning insights

In this work, we want to showcase a way to leverage the good predictive power of machine learning models in augmenting a standard GLM. Machine learning has been able to uncover a deeper understanding of the underlying trends in the dataset being modelled. These deeper trends and the relationships between independent variables and response variables are most often obfuscated and there is no clear way to showcase how a prediction from a machine learning method comes about. Recently, a need to understand the underlying decision-making of machine learning methods has become relevant, and research into interpretable machine learning has taken off (Murdoch et al., 2019).

In this chapter, we will discuss ways to learn from machine learning methods by visualising the use of variables and applying this knowledge in a semi-automatic way to augment the underlying GLM. Two methods are discussed: one focuses on insight extraction through partial dependence, and the other leverages the underlying structure of decision trees, extracting the splitting rules and modelling the response using those rules.

3.1 Measures and statistics

This subchapter is based on (Molnar, 2022) and (Hastie, Tibshirani, and Friedman, 2009).

This subchapter discusses several measures and metrics used for the insight methods and model comparison.

3.1.1 Model performance metrics

In machine learning, many different models can solve a single problem. Many hyper parameters can and should be tuned inside these machine learning models to find the best fit for the data. There is a need for some metric to quantify how well a model is performing.

One of the most popular metrics used is the model's *test set loss or deviance*. When training a machine learning model, a common approach is splitting the data into two distinct parts: training and test data. The aim is to have a previously unseen bit of data, the test set, that acts as an equal playing field for all models. First, the model prediction function $F(\mathbf{x})$ is estimated based on the training data, and then an appropriate (based on the modelling problem) loss function $L(y, F(\mathbf{x}))$ is used to calculate the total loss for the test set, $\sum_{i=1}^{n_t}(L(y_i, F(\mathbf{x}_i))$. Note that for some problems, the total loss is not the most appropriate metric to minimise, so deviance can

be used instead. For example for a Poisson distributed response variable, Poisson deviance for an i.i.d. sample of observations $\{(y_i, \mathbf{x}_i)\}_1^n$ can be calculated as

$$\mathrm{Dev}_{\mathrm{P}}(F) = 2\sum_{i=1}^{n}\left[y_i \ln\left(\frac{y_i}{F(\mathbf{x}_i)}\right) - (y_i - F(\mathbf{x}_i))\right]. \tag{3.1}$$

Total loss or deviance based model comparison has one main issue. Namely, as the complexity of the underlying model increases, the total loss or deviance of that model will stay the same or decrease. This is similar to the case where increasing the number of linear terms always lowers or keeps the deviance of the linear model. For machine learning models, this is not a significant issue since most of the models have basically an infinite (saturated) number of parameters (splitting variables, splitting values, etc.). However, this issue persists for parametric models like generalized linear regression.

A way to get around this for parametric models is to introduce a cost for the number of parameters in the model. For in-sample error prediction, the *Akaike information criterion* (AIC) can be used. For parametric models using likelihood, we have

$$AIC = -2 \cdot l(\boldsymbol{\beta}, \phi, \mathbf{y}) + 2p. \tag{3.2}$$

The best model can be again chosen by having the lowest AIC. As the number of model parameters is penalised, the lower AIC ensures that predictive power of the model increased more than just the improvement due to the increased complexity.

3.1.2 Variable importance

For models with p independent predictor variables, seldom are all variables equally relevant for predicting. For generalized linear models, an estimate of a variable's coefficient β_i and its variance $\mathbf{D}(\beta_i)$ is enough to tell if that variable is statistically significant for predicting the response variable through the use of a test such as the likelihood ratio test. However, for machine learning methods, this kind of setup is missing.

One possible solution to determine what predictors are relevant to predicting the response is called *variable importance*. There are several types of variable importance, but in this work, we look at relative variable importance and permutation variable importance.

In the case of relative variable importance, we want to know how "relevant" different values of variable X_p are in predicting the response variable. For a single decision tree, the squared

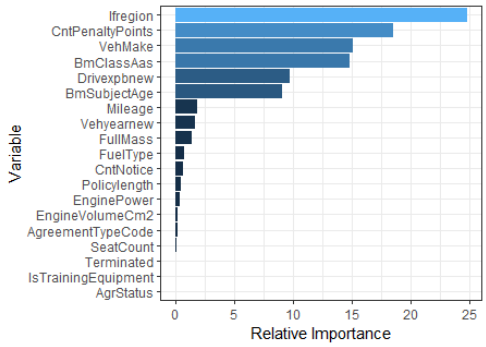

Figure 4: Relative variable importance for gradient boosting machine computed on training data.

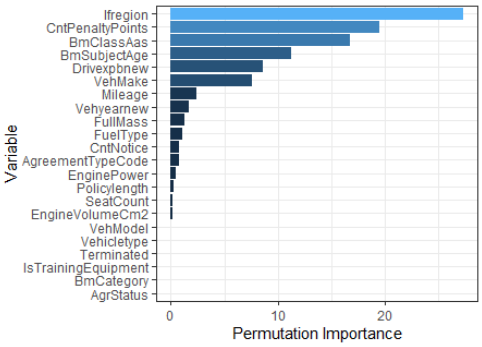

Figure 5: Permutation variable importance for gradient boosting machine computed on training data.

relative relevance of variable X_p can be expressed as the sum of squared improvements in the loss for all splits in a tree using this variable. In other words, we can write

$$\mathrm{IR}_p^2(T) = \sum_{j=1}^{J-1} i_j^2 I(v(j) = p),$$

where J is the number of internal nodes in the tree, $v(j)$ is the index of the splitting variable for node j, and i_j^2 is the squared improvement in the loss for adopting a given split. This single decision tree relevance generalises well to additive tree expansion models since an average across all trees in the expansion shows the same phenomena. Then we can write that for tree ensembles

$$\mathrm{IR}_p^2 = \frac{1}{\mathcal{T}} \sum_{t=1}^{\mathcal{T}} \mathrm{IR}_p^2(T_t),$$

where $\mathcal{T}$ is the number of decision trees in the additive expansion. Relevance is simply the square root of the corresponding squared relevance. As this measure is relative to the data it was fitted on, it is usually given as proportion (sums to 100). An example of scaled relative variable importance for a gradient boosting machine is shown in Figure 4.

Permutation variable importance is a little different from relative variable importance even though they try to quantify the same thing. For permutation variable importance, we take the variable X_p, permutate the realised values $\mathbf{x}_p = (x_{1,p}, x_{2,p}, \ldots, x_{n,p})^T$ in the dataset and see how much the loss of the dataset changed. Based on the increase in loss, we can see how important the given variable value was in making the prediction for a given observation.

To calculate permutation variable importance, we take the prediction function $F(\mathbf{x})$ and the loss function $L(y, F(\mathbf{x}))$ and compute the sum of losses on the dataset $\mathcal{L}_{orig} = \sum_{i=1}^{n} L(y_i, F(\mathbf{x}_i))$.

Then permutate the variable X_p in the dataset to get $X_{p_{perm}}$ and compute the sum of losses with the permuted variable $\mathcal{L}_p = \sum_{i=1}^{n} L(y_i, F(\mathbf{x}_{i_{perm}}))$, $\mathbf{x}_{i_{perm}}$ denotes the ith observation with the permutated variable $X_{p_{perm}}$. Permutation importance is then their ratio

$$\text{IP}_p = \frac{\mathcal{L}_p}{\mathcal{L}_{orig}},$$

or their difference

$$\text{IP}_p = \mathcal{L}_p - \mathcal{L}_{orig}.$$

Similarly to relative importance, this value is usually presented as a proportion. An example of permutation importance for a gradient boosting machine is shown in Figure 5. Note that the order of variable importance is different for different measures, but overall the same variables are considered important.

3.1.3 Partial dependence

Now that we have tools to choose a good model and to see what variables the model relies on to make predictions, we still need a way to visualise how the different values of a given variable or even a collection of variables affect the prediction. For a linear regression model, this boils down to interpreting the different regression coefficients β_j.

For example, in the case of modelling claim frequency using GLM with Poisson distribution and log-link function, we could say something along the lines: "Having two subjects for which all variables expect X_j are the same and numeric variable X_j differ by 1 unit, then the estimated claim frequencies differ by a factor of $\exp(\beta_j)$". For categorical variable X_k, the estimated claim frequency would differ by a factor of $\exp(\beta_k)$ when compared to the baseline categorical level.

As evident from previous subchapters, here again, machine learning does not have such simple interpretable tools. However, one somewhat interpretable tool exists - *partial dependence*. Partial dependence shows the average prediction for different values or unique combinations of values of fixed variables where the effect of not fixed variables is averaged out.

Take a vector of indices for fixed variables $S \subset \{1, 2, \ldots, p\}$. Let C be its complement set, so that $S \bigcup C = \{1, 2, \ldots, p\}$. Let $\mathbf{X}_S$ denote the variable space defined by the variables with indices S and let $\mathbf{X}_C$ be the complement variable space such that they can be combined to make original variable space $\mathbf{X}$. The prediction function can be written as $F(\mathbf{x}) = F(\mathbf{x}_S, \mathbf{x}_C)$, where $\mathbf{x} \in \mathbf{X}$, $\mathbf{x}_S \in \mathbf{X}_S$ and $\mathbf{x}_C \in \mathbf{X}_C$. Partial dependence for values $\mathbf{x}_S \in \mathbf{X}_S$ can then be defined as

$$PD(\mathbf{x}_S) = \mathbf{E}\left(F(\mathbf{x}_S, X_C)\right),$$

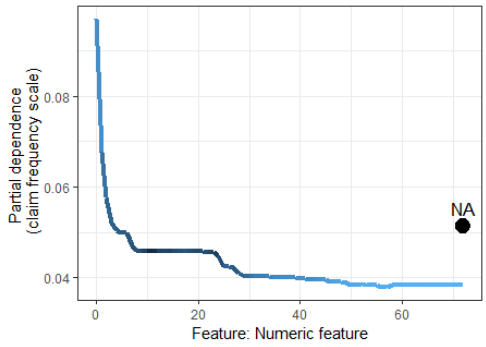

Figure 6: GBM partial dependence for a numeric variable estimated from a sample of 10^5 rows of training data

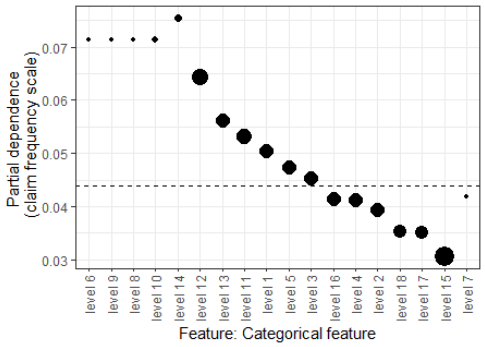

Figure 7: GBM partial dependence for ordered categorical variable estimated from a sample of 10^5 rows of training data

where X_C is a random vector from complement space $\mathbf{X}_C$. In other words, partial dependence of fixed variable values $\mathbf{x}_S$ is the expected value of the prediction function $F(\mathbf{x}_S, X_C)$ in the probability space defined by complement variables $\mathbf{X}_C$. Using real data, partial dependence can easily be estimated by

$$\hat{PD}(\mathbf{x}_S) = \frac{1}{n}\sum_{i=1}^{n} F(\mathbf{x}_S, \mathbf{x}_{i,C}),$$

where $\mathbf{x}_S$ is an element of observed combinations from space $\mathbf{X}_S$ and $\mathbf{x}_{i,C}$ denotes the values of the complement set variables for the i-th observation. Note that partial dependence is usually calculated sequentially over unique values of the fixed variables present in the data, then the particular values of partial dependence can be aggregated into a vector or easily plotted.

Single variable partial dependence uses all unique values of that variable present in the data (grid of values) and calculates the corresponding average prediction over all other variables. For numeric variables, plotting partial dependence produces a trailing plot showcasing the change in partial dependence value based on values of numeric variables. For categorical variables, plotting partial dependence produces a scatter plot showcasing the partial dependence value for each level of categorical variable. For two variable partial dependence, a heat map (if both are numeric) or multiple line plot (if one is numeric and the other is categorical) or level faceted point plots (both are categorical) can be used. There is no easy or interpretable way to visualise higher dimensional partial dependence. Examples of one variable partial dependence plots are presented in Figures 6 and 7.

From the averaging, we can see that if we fix only one variable X_j ($j \in \{1, 2 \ldots, p\}$) then, for all unique values of variable X_j, we go through all values from the dataset therefore, the partial dependence calculation is at least $O(n^2)$ complexity. Considering more than one variable, the

number of possible combinations grows by up to another factor of n, so in total complexity would be $O(n^3)$. In general, the complexity for partial dependence of s variables is $O(n^s)$. This is very computationally intensive, even for moderately sized datasets.

3.1.4 Friedman's H-statistic

This subchapter is based on (Friedman and Popescu, 2008).

One of the key ways to model non-linear relations in linear regression is through the use of interaction. Interaction allows the value of one variable to affect the effect of another variable or even variables. In predictive modelling, interactions add a lot of complexity to a model but can also significantly improve the predictive power of a model.

The decision trees are able to model interaction effects through consecutive splits on a single tree, thus leading to a structure that inherently models interactions. This property carries over to any other tree-based models (but not exclusively).

In their paper (Friedman and Popescu, 2008), the authors defined a statistic that quantifies the presence and/or strength of an interaction between variables used in predictive function $F(\mathbf{x})$. Let all partial dependence be centred (mean removed). They showed that the partial dependence of two variables X_j and X_k could be decomposed into

$$PD(\mathbf{x}_{\{j,k\}}) = PD(\mathbf{x}_{\{j\}}) + PD(\mathbf{x}_{\{k\}}),$$

if variables do not interact. Here $\mathbf{x}_{\{j,k\}} \in \mathbf{X}_{\{j,k\}}, \mathbf{x}_{\{j\}} \in \mathbf{X}_{\{j\}}$ and $\mathbf{x}_{\{k\}} \in \mathbf{X}_{\{k\}}$. Then, a statistic can be defined and empirically estimated using the empirical estimates of partial dependence on the dataset,

$$H^2_{\{j,k\}} = \frac{\sum_{i=1}^{n} \left[\hat{PD}(\mathbf{x}_{i,\{j,k\}}) - \hat{PD}(\mathbf{x}_{i,\{j\}}) - \hat{PD}(\mathbf{x}_{i,\{k\}})\right]}{\sum_{i=1}^{n} (\hat{PD}(\mathbf{x}_{i,\{j,k\}}))^2}, \tag{3.3}$$

where $\hat{PD}(\mathbf{x}_{i,\{j,k\}})$ is the partial dependence value for interaction combination present for observation $\mathbf{x}_i$ and $\hat{PD}(\mathbf{x}_{i,\{j\}})$ and $\hat{PD}(\mathbf{x}_{i,\{k\}})$ are single variable partial dependence values for variable values present for observation $\mathbf{x}_i$. This H-statistic showcases the fraction of variance not explained by a single variable partial dependencies $PD(\mathbf{x}_{\{j\}})$ and $PD(\mathbf{x}_{\{k\}})$. If the value is 0, then no interaction between variables X_j and X_k is present in prediction function $F(\mathbf{x})$. Otherwise, larger values correspond to a "stronger" interaction between these variables.

For a single variable X_j, we can also check if any interactions with this variable are present in

$F(\mathbf{x})$ by using another version of H-statistic (3.3),

$$H_j^2 = \frac{\sum_{i=1}^{n} \left[F(\mathbf{x}_i) - \hat{PD}(\mathbf{x}_{i,\{j\}}) - \hat{PD}(\mathbf{x}_{i,\setminus j}) \right]}{\sum_{i=1}^{n} (F(\mathbf{x}_i))^2}, \tag{3.4}$$

where $\hat{PD}(\mathbf{x}_{i,\setminus j})$ denotes the interaction partial dependence value for all variables expect jth variable for combination present in observation $\mathbf{x}_i$.

Friedman and Popescu also proposed to combine the numerator of (3.3) and denominator of (3.4) to get a version of H-statistic showcasing the importance of a particular interaction in the prediction model. Additionally, the authors showed that this statistic could easily be expanded to higher dimension interactions. In this work, we only focus on two variable interactions.

3.2 Model-Agnostic Interpretable Data-driven suRRogates (maidrr)

This subchapter is based on (Henckaerts, Antonio, and Côté, 2020).

Partial dependence is one way to quantify the way the machine learning model uses one variable. In 2020, Roel Henckaerts et al. proposed a method using partial dependence, segmenting and clustering it, to produce a surrogate model capable of mimicking the prediction function $F(\mathbf{x})$ of any machine learning model, thus being a model-agnostic approach. A workflow of the maidrr process can be seen in Figure 8.

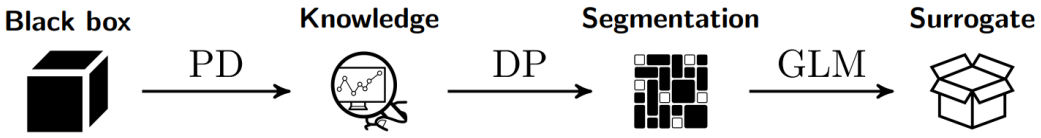

Figure 8: maidrr process for transforming black box methods into interpretable GLM. Taken from (Henckaerts, Antonio, and Côté, 2020) Figure 2.

Assuming a satisfactory machine learning "black box" method is fit to the data, we first extract the partial dependence for independent variables in the data. The same calculation is used as described previously, but since the algorithm is computationally very demanding, this is usually done over a smaller sample of training data. Note that since we want to average out the effect of other variables, the Monte-Carlo approach of oversampling for smaller initial datasets might lead to a better estimation of the distribution of complement variables, thus reducing the variance of partial dependence.

After partial dependence is calculated, a dynamic programming clustering K-means algorithm is used to make optimal (for a given K) and reproducible clusters of partial dependence effects (Wang and Song, 2011). These clusters are bins of values for numeric variables or sets of categorical levels for factor variables. Note that only sequential factor variable levels can be grouped for ordered factor variables. The clustering produces a segmentation of independent variables allowing for a constant to be fit to a given segment, representing its effect on the response variable.

To find an optimal number of groupings for all independent variables would be a p-dimensional problem. However, maidrr authors proposed the use of penalised loss function to find the optimal number of groups $\hat{k}_{\{j\}}$, $j = 1, \ldots, p$. The process is simple and starts by taking a random guess for the number of groups for a variable.

Suppose variable X_j was grouped into $k_{\{j\}}$ groups. Let $z_{i^*,\{j\}} = \hat{PD}(\mathbf{x}_{i^*,\{j\}})$ denote the partial dependence value of i^*-th unique value of variable X_j, $i^* \in \{1, \ldots, m_{\{j\}}\}$, where $m_{\{j\}}$ is the number of unique values, and let $\tilde{z}_{i^*,\{j\}}$ be the average partial dependence value of the group the unique value $\mathbf{x}_{i^*,\{j\}}$ belongs to.

The optimal number of groups $\hat{k}_{\{j\}}$ can be found by solving

$$\arg\min_{k_{\{j\}}} \sum_{i^*=1}^{m_{\{j\}}} w_{i^*,\{j\}}(z_{i^*,\{j\}} - \tilde{z}_{i^*,\{j\}})^2 + \lambda \ln(k_{\{j\}}), \tag{3.5}$$

where $w_{i^*,\{j\}}$ is the proportion of observations that have value $\mathbf{x}_{i^*,\{j\}}$. Using the weighted mean squared error in Formula (3.5) encourages good groupings for frequently occurring feature values. Low (high) values of penalty parameter λ allow for more (less) groups for a given feature, resulting in a smoother (coarser) approximation of the underlying partial dependence effect.

Examples of possible groupings for a numeric variable are presented in Figure 9, and for ordered categorical variable is presented in Figure 10. The figures indicate grouping by the vertical lines for numeric variables and as different symbols for categorical variables. Additionally, the weight of given feature values is displayed as the line colour for the numeric variable (darker means more) and the size of the symbol for the categorical variable (bigger means more).

Through grouping of partial dependence, we find a possible segmentation of independent variables in space $\mathbf{X}$, thus turning all variables into categorical form by introducing additional parameters for numeric variables and taking away some "not needed" parameters for categorical variables. After grouping, a surrogate model (GLM) is fit. Since all variables end up as categorical levels, the fitted GLM can easily be transformed into a fixed-size decision table which is very transparent and interpretable.

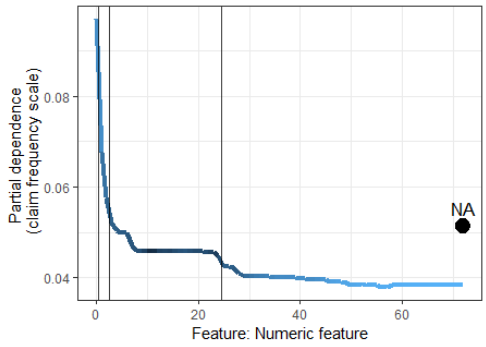

Figure 9: Example of grouping numeric variable partial dependence into 5 groups (NA values as a separate group).

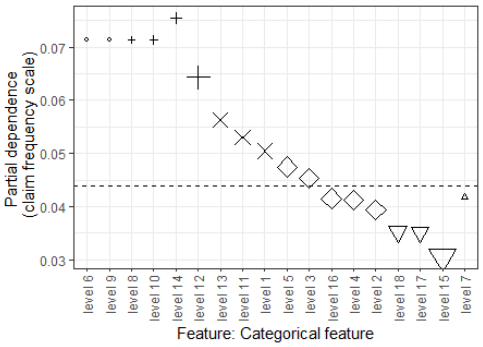

Figure 10: Example of grouping ordered categorical variable partial dependence into 6 groups.

In addition to main effect modelling, interaction can also be modelled using H-statistic described in the previous subchapter. H-statistic is used to find possible interactions, and partial dependence of interactions with suitable strength (above a value defined by the modeller) are grouped using the same procedure as above. The effect of the corresponding interaction variable is then interpreted as an additional effect on top of the main effects (interaction effect is centred at 0, which indicates no effect). Note that in this case, the main effect penalty and interaction effect penalty can be different and will be denoted by λ_{main} and λ_{intr}.

To find the best possible surrogate, maidrr focuses on 4 hyperparameters: penalties λ_{main} and λ_{intr}, maximum number of groups k and interaction strength cut-off value h. Penalties are tuned by a grid search using K-fold cross-validation (data is split into K parts, one part is left out as the validation part, and the model is trained on the rest of the data) by finding a GLM model minimising the desired loss function with respect to the original response variable (not machine learning predictions, like other surrogate techniques). First tuning of λ_{main} is done, and depending on the features selected (if $\hat{k}_{\{j\}} = 1$, the feature is excluded from the surrogate model) λ_{intr} is then tuned. Hyperparameters k and h focus on the complexity of the surrogate and depend on the desired outcome. If a more complex model is suitable, then a high value of k and a low value of h allow for smoother main effects and more interactions. Opposite values of k and h allow for a coarser surrogate. The algorithm for maidrr surrogate is present in Appendix (A.1) and for penalty tuning in Appendix (A.2).

3.3 Rule ensemble

This subchapter is based on (Friedman and Popescu, 2008).

There are almost countless ways to model a response variable, stemming from the choice of methods and parameters for these methods. Seldom are these methods easily interpretable. In some industries, the ability to interpret the models and make general business decisions based on those models is needed. Thus an interpretable machine learning method is needed.

Decision trees, as shown in Chapter 2, are rule based methods which, on their own, are very interpretable but also highly unstable. To fix this, tree ensembles can be used at the cost of interpretability. In 2008, Friedman and Popescu proposed a way to leverage these highly interpretable rules of tree ensembles to make a model with predictive power comparable to those ensemble methods.

When working with ensembles, we are searching for a predictive function $F(\mathbf{x})$ as an additive expansion

$$F(\mathbf{x}) = \beta_0 + \sum_{t=1}^{\mathcal{T}} \beta_t h(\mathbf{x}, \mathbf{a}_t), \tag{3.6}$$

where $h(\mathbf{x})$ is the "base learner" prediction function with parameters $\mathbf{a}_t$ and combination parameter β_t. When working with tree ensembles, "base learner" $h(\mathbf{x}, \mathbf{a}_t)$ is a decision tree with M terminal nodes and parameters $\mathbf{a}_t$ are the splitting variables with indices $\mathbf{j} \subset \{1, \ldots, p\}$, and splitting points $\mathbf{s}$, which are enough to define the M terminal regions.

In their paper (Friedman and Popescu, 2008), the authors proposed that letting go of the underlying decision tree branching structure and just using the rules produced from decision trees as the "base learners" could produce a highly interpretable method. Denote $\mathbf{X}_{\{j\}}$ as the set of all possible values of variable X_j and $v_{j,k}$ be a subset of these values, $v_{j,k} \subset \mathbf{X}_{\{j\}}$. A rule base learner is then

$$r_k(\mathbf{x}) = \prod_{j=1}^{p} I(x_j \in v_{j,k}),$$

where $I(\cdot)$ is the indicator function and k is index of the rule used. Using product over all variables results in a two-valued base learner ($r_k(\mathbf{x}) \in \{0, 1\}$), taking non-zero value only if all variables X_j, $j = 1, \ldots, p$ belong to their specified subset of values $v_{j,k}$. For orderable variables (numeric and ordered categorical variables), the subset of values is an interval of values

$$v_{j,k} = (t_{j,k}, u_{j,k}],$$

where $t_{j,k}$ and $u_{j,k}$ are the lower and upper limit values (categorical levels), respectively. For unorderable categorical variables, $v_{j,k}$ is an explicit subset of possible categorical levels.

Note that if $v_{j,k} = \mathbf{X}_{\{j\}}$, then variables X_j can be omitted from the rule since $I(x_j \in \mathbf{X}_{\{j\}}) = 1$ for all X_j values. In practice, "simple" rules are desirable as this leaves most of the variables X_j

out and focuses on a few "important" variable segments. An example of a rule generated from a CART decision tree on Figure 3 corresponding to region R_4 of the tree is

$$r_4(\mathbf{x}) = I(x_1 \in (t_3, 1]) \cdot I(x_2 \in [0, t_4]).$$

As shown above, consecutive decision tree splits fit the desired base learner structure and thus can easily be extracted and used on their own as base learners. Note that, not only terminal node rules can be used, but any combination of splits leading to any node in the tree can be used.

Suppose now, we have a $\mathcal{T}$ binary trees $\{T\}_1^{\mathcal{T}}$. This results in $K = \sum_{t=1}^{\mathcal{T}} 2(|T_t| - 1)$ rules $\{r_k(\mathbf{x})\}_1^K$, where $|T_t|$ is the number of terminal nodes for t-th tree. Then we can define rule ensemble as

$$F(\mathbf{x}) = a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x}), \tag{3.7}$$

where rules $\{r_k(\mathbf{x})\}_1^K$ serve as the base learners and a_k, $k \in \{0, \ldots, K\}$ serve as combination parameters from Formula (3.6).

Using now importance sampled learning ensemble (ISLE) methodology as described in (Friedman and Popescu, 2003), one possible way to estimate the combination parameters $\{\hat{a_k}\}_0^K$, is using regularised linear regression on the training data

$$\{\hat{a}_k\}_0^K = \underset{\{a_k\}_0^K}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y_i, a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x})\right) + \Lambda \cdot \sum_{k=1}^{K} |a_k|,$$

where $L(\cdot)$ is the loss function we want to minimise. The regularisation used here is Lasso regression, which uses the prediction risk $\sum_{i=1}^{n} L\left(y_i, a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x})\right)$ with an additional constraint on the absolute size of the parameter, $\Lambda \cdot \sum_{k=1}^{K} |a_k|$. It can be shown that when using Lasso regression, larger penalty values Λ produce shrinkage, often setting many "unimportant" parameters $\{a_k\}_0^K$ to zero, effectively excluding them from regression (Tibshirani, 1996). This is favourable since Lasso regression helps us to perform off-hand feature selection on many possible rules used in the ensemble.

In their research, Friedman and Popescu showed that rule ensemble generated from trees with random tree size performed well when compared to other ensemble methods. However, an additional augmentation was proposed. Friedman and Popescu argued that the linear function is among the most difficult functions to approximate using rules (and decision trees), requiring a

large number of iterations and rules to estimate accurately. They suggested that using additional linear components in the additive expansion can help deal with linear dependence without sacrificing much predictive power, thanks to Lasso regression allowing to eliminate unnecessary linear components. The linear augmented rule ensemble is then

$$F(\mathbf{x}) = a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x}) + \sum_{j=1}^{p} \gamma_j x_j,$$

where rule ensemble additive expansion from Formula (3.7) has additional linear terms as base learners with corresponding combination parameters γ_j.

Again using ISLE approach, we get that combination parameters $\{\hat{a}_k\}_0^K$ and $\{\hat{\gamma}_j\}_1^p$ can be estimated using Lasso regression

$$(\{\hat{a}_k\}_0^K, \{\hat{\gamma}_j\}_1^p) = \underset{\{a_k\}_0^K, \{\gamma_j\}_1^p}{\operatorname{argmin}} \sum_{i=1}^{n} L\left(y_i, a_0 + \sum_{k=1}^{K} a_k r_k(\mathbf{x}_i) + \sum_{j=1}^{p} \gamma_j x_{i,j}\right) + \Lambda\left(\sum_{k=1}^{K} |a_k| + \sum_{j=1}^{p} |\gamma_j|\right). \tag{3.8}$$

It is important to keep in mind that Lasso regression is very sensitive to the scale of predictors. Thus, the normalisation of predictors should be performed prior to fitting the model. Authors suggested to regularise variables X_j, $j = 1, \ldots, p$ using

$$x_j \leftarrow 0.4 \frac{x_j}{\operatorname{std}(x_j)},$$

where $\operatorname{std}(x_j)$ is the standard deviation of the variable X_j in the data. The coefficient 0.4 is used to scale the variable to the same influence as an average rule with uniform support (number of observations with a given rule) on the unit interval. Authors note that scaling the rules can be done but is seldom required since rules with very large or very small support are ultimately defined by a small number of training observations, which is undesirable.

Additionally, since linear terms might have outlier issues, a "Winsorized" version of the linear term should be used

$$W(x_j) = \min(\delta_j^+, \max(\delta_j^-, x_j)),$$

where δ_j^- and δ_j^+ are $\alpha \in (0, 0.5)$ and $1 - \alpha$ quantiles of the distribution of X_j, respectively.

4 Claim frequency modelling

This chapter focuses on modelling motor third-party liability (MTPL) claim frequency using generalized linear models, gradient boosting machine and XGBoost. The modelling is done on Latvian MTPL data provided by If P&C Insurance AS. Note that in Latvia, MTPL policies are a part of a shared market, thus, the same information is available to all insurance providers in the area at all times.

Motor third-party liability is compulsory insurance required for all vehicles registered with the Latvian Motor Vehicle Register. In case of an accident, this insurance covers the cost of damages done to a third party's property or health. Claim frequency modelling should, in theory, be the only approach capable of quantifying risk for the insurance company since claim amounts should have little to no relation with the policyholder or vehicle specified, as only third-party damages are compensated. Predicting claim frequency right allows the insurance company to capture better risk from the population by giving better prices and to drive away unwanted risk by setting a more appropriate price of insurance.

The full dataset contains 12 847 035 policies issued in $2012-2018$ to private clients. The dataset has 84 columns that can be split into 3 categories:

- Policy information columns: agreement type, agreement status, policy start and end dates, policy duration, estimated policy issue region, number of claims and claim amounts linked to the policy.
- Policy owner information: policy subject birth date, subject age and driving experience, national driving penalty points and penalty notices, start and end dates for license by vehicle category, bonus-malus class, previous bonus-malus class and the number of previous claims.
- Policy vehicle information: vehicle age, type, seat count, mass, make, model, body type, fuel type, mileage, engine power and volume, date of first registration and date of last inspection.

Several columns were either duplicate columns or information stored in them was unusable or missing. From these 84 columns, 45 were selected for further analysis and preprocessing. The full table of selected columns, column types and column descriptions can be seen in Appendix B.1.

4.1 Data and preprocessing

The initial dataset provided by If had several issues with regard to data quality. This is a common issue with real-life data as data aggregation is done by combining data from several sources that seldom share structure. To use this data for any kind of analysis, it needs to be preprocessed.

This dataset was quite big, just barely fitting into the memory on the machine (laptop provided by If) used. This meant that any sort of out-of-the-box data manipulation could not be done, and another software capable of dealing with big datasets needed to be installed. To solve this, a local computation cluster using Apache SPARK backend and Apache Hadoop distributed file system was set up (Apache Software Foundation, 2020; Apache Software Foundation, 2018). To communicate to the cluster, RStudio package sparklyr (Luraschi et al., 2022) was used as the frontend to send R commands to the cluster. This system allowed to use pipelines from package dplyr (Wickham et al., 2022) to select, filter and modify all of the data at once, doing it quite fast.

Preprocessing was done on column-by-column bases. Key preprocessing steps are listed below:

- All of the dates were converted from text type in format YYYY/MM/DD (but not always) to date objects in R.
- Columns VehMake, VehModel had some of their levels aggregated into level "Other" based on the count of observations with that level (less than 5000 were aggregated). VehModel was dropped due to too many categorical levels.
- VehMake and Ifregion were encoded to numeric levels to ease the visualisation and model printouts.
- Categorical columns with missing values were merged to corresponding "not available" levels ("n/z", "n/a", etc.).
- Some of the systemic errors were fixed. For example, sometimes, Mileage was calculated in the wrong way (StartMileage - LastMileage).
- Date-based values like age, driving experience, vehicle age, etc., were calculated and compared against values already present in the data.
- Ranges of most numeric variables (excluding response) were cut to 0.995 quantiles to exclude very large outliers.

- Testing scheme for date logic was developed. For example, check if the birth date is before all other dates.
- Categorical columns were turned to factor values, where factor levels were ordered by their count in the data (ordering by exposure was considered, but showed no difference in ordering of levels)
- Group ID was constructed since some policies had the same information aside from claim-related data. Group ID was used to make training and testing sets to avoid data leakage between splits.

Note that missing values were present for most numeric variables, and for some of the categorical variables missing values were not altered. This was intentional since gradient boosting machine implementation in R package gbm (Greenwell et al., 2022) allows for a separate direction for missing values, thus allowing to model even missing data.

Although some machine learning techniques are available in SPARK, the implementations available were not up to par in terms of possible loss functions, weighting etc. Implementations in R packages gbm, xgboost (Chen et al., 2022) were used instead. This meant that in-machine computation had to be done, so the whole data set training was impossible. Based on proposals from If and prior modelling papers like (Henckaerts et al., 2017) or (Wüthrich, 2019), a single financial year (2018) policies were chosen as data to be modelled. The latest financial year was chosen since less data seemed to be missing for later years.

The final data that was modelled and analysed contained 1 644 800 rows with 19 explanatory variables, exposure variable, number of claims as the response variable and Group ID column. The data was split based on unique values of Group ID column into three parts: training with proportion 0.6, validation with proportion 0.2 and testing with proportion 0.2 of the data.

4.2 Baseline models

To compare different approaches, a baseline model is needed. One such baseline is the trivial model predicting historic average response for new observations. However, for the insurance industry, a GLM model is widely used; thus, it is a good baseline to compare against.

This subchapter focuses on building the GLM models used as baseline models to compare different models to. The models are fitted using the structure specified in Subchapter 1.1. Parameters

of the models are estimated using maximum likelihood estimation following the setup given in Subchapter 1.3.

Note that GLM is not able to work with missing values. This means that missing data should either be imputed or deleted. As discussed at the end of the previous subchapter, the gradient boosting model is able to work with missing data, thus imputation of the data, although industry standard, is forgone to showcase one of the possible advantages that can be gained from machine learning methods.

This means that GLM models were fit on data that had all of the observations with missing values removed. Since GLM can easily be fitted using the likelihood of the data, there is no need for validation split to assess the fit of the model, and a bigger dataset allows for a better likelihood fit of the model parameters. GLM models were fitted using 1 117 001 observation (about 10.39% observation less compared to full training and validation set (1 315 720)).

A bidirectional step-wise search based on AIC (defined in Formula (3.2)) was used to find the model. The rationale for the step-wise search was that optimising AIC allows us to find the model best generalising to training data and hopefully generalising well in general. Additionally, the step-wise search can be considered an automatic modelling way, similar to machine learning models.

The model with the best AIC used these variables: FullMass, BmSubjectAge, Vehyearnew, Mileage, Drivexpbnew, AgreementTypeCode, CntNotice, CntPenaltyPoints, FuelType, BmClassAas, Ifregion, Policylength. Variables Vehyearnew and Drivexpbnew are transformations of variables VehicleAge and DrivExpB with corrections from date information. This model will be called "AIC model". The description of variables can be seen in Table 3 in Appendix B.1.

Note that no interactions were considered. This was done for three reasons. Firstly, a simple model to compare is desirable, and interactions introduce a lot of complexity. Secondly, aside from an exhaustive search, there are no easy ways to automate the search for interactions using just the GLM framework. Lastly, interaction computation for some combinations of the variables was not feasible (or in some cases possible) on the hardware used.

A second baseline model was also proposed, using the previously described GLM model as the starting point. Following some historical findings about driver age and experience, a polynomial relation for age and experience was considered (Valecký, 2016). In a backward step-wise search (based on AIC), starting at the sixth-order polynomial term for each variable was considered. The addition of polynomial terms decreased AIC further, where the lowest AIC was achieved by the model with both variables with polynomial terms up to the fifth order. This model will

be called "AIC model + poly".

Both of these GLM models will be used as benchmarks to compare the performance of machine learning models and improvement methods described in Chapter 3.

Appendix C gives an overview of both models. Model structure, coefficients and interpretation are presented.

4.3 Modelling with GBM and XGBoost

The gathered insight can only be as good and insightful as the underlying machine learning model is at predicting the response variable. This subchapter gives an overview of the tuning procedure for gradient boosted machine (GBM) and XGBoost models. The models follow the corresponding frameworks described in chapters 2.2.2 and 2.2.3. Note that these models share the structure; however, XGBoost can be considered the more advanced and modern algorithm.

GBM model is used to showcase maidrr methods since models from R package gbm work naively with maidrr implementation in package maidrr (Henckaerts, 2020). XGBoost model is trained using R package xgboost. The rule ensemble implementation in R package xrf (Holub, 2022) expects a XGBoost model as the underlying tree model.

For both models, the procedure for hyperparameter tuning was similar. First, a grid for appropriate hyperparameters was created. This was done using R package dials (Kuhn and Frick, 2022). The dials package allows to specify the ranges for hyperparameters and then apply a complete grid selection, random selection or maximal entropy selection. Maximal entropy selects parameters in such a way that the whole space specified by the ranges is best covered. This was used to generate 20 possible combinations of hyperparameters. Then, models were fit using these parameters on the training portion of the data, and their performance was validated on the validation portion of the data, using (3.1) as the error metric.

For the GBM model, 4 hyperparameters were tuned using the grid: maximum depth of each tree in the ensemble (range 3 to 6), ensemble learning rate (range 0.001 to 0.2), minimum number of observations in leaf node (range 1 to 987) and observation sampling rate for trees in the ensemble (range 0.5 to 0.8). The optimal number of trees in the ensemble was found using a fraction (20%) of training data as an additional validation data where the best number of trees had the smallest validation error.

Using this tuning procedure, I found that the GBM model with a tree depth of 4, an ensemble learning rate of 0.03311877, a minimum number of observations in a leaf node of 339, an

Table 1: Baseline models and machine learning models performance comparison. The test set had all rows with missing values dropped to make model performance comparable (GBM could also predict with missing values). The trivial model predicts training data average frequency for all observations.

Model	*AIC*	Poisson Deviance (Test)	Number of parameters	Deviance proportion
Trivial model	-	0.11695364	-	104.3842%
AIC model	167150.8	0.11204148	91	100%
AIC model + poly	166756	0.11159231	99	99.5991%
GBM	-	0.11146033	-	99.4813%
XGBoost	-	0.11158860	-	99.5958%

observation sampling rate of 0.6757881 and 691 trees gave the smallest validation error on the validation portion of the data.

XGBoost is the more advanced version of gradient tree boosting and thus allows for more hyperparameters to be tuned. For XGBoost, 7 hyperparameters were tuned: number of variables used to fit a tree (range $\lceil\sqrt{108}\rceil$ to $\frac{108}{3}$), maximum depth of trees in the ensemble (range 3 to 6), ensemble learning rate (range 0.1 to 0.3), number of trees (range 1 to 100), the minimum number of observations in leaf node (range 1 to 987), reduction in loss to allow further splits (range 0 to 0.2) and observation sampling rate for trees in the ensemble (range 0.5 to 0.75).

It is important to note that the XGBoost model, as implemented in package `xgboost`, did not allow for missing values in data and all data has to be numerical, thus one-hot encoding (splitting all levels of categorical variables into separate binary variables) needed to be applied. Doing so takes the number of variables to 108. Note also that only a small number of trees (up to 100) were considered since more trees produce more rules, thus increasing the computation intensity of rule ensemble methods.

The optimal XGBoost model had 35 variables for each tree, depth of 4, a learning rate of 0.28255206, 91 trees in the ensemble, at least 496 observations in the leaf nodes, reduction in loss of at least 0.06351081 and a sampling rate of 0.68568611 for each tree.

A comparison between the resulting models can be seen in Table 1. Since all models use variables and data differently, a unifying test set needed to be created. This meant removing all missing values from the testing set since both GLM and XGBoost models can not work with missing values. Based on the test set performance (using again (3.1)), we can see that the GBM model is clearly best, followed by the XGBoost model and *AIC* model with polynomial terms. The base *AIC* model is worse compared to other non-trivial models. Taking now the *AIC* model

Poisson deviance as 100%, then the polynomial terms improve the Poisson deviance by 0.4009%, XGBoost by 0.4042% and GBM by 0.5187%. The trivial model is about 4.3842% worse than the AIC model, showing that variables are able to describe the response in some way.

The accuracy improvements machine learning methods provided were much smaller than expected. However, it is important to remember that a better choice and tuning of hyperparameters and some restrictions on the model structure can further improve these models. This is not the main aim of the thesis, and no further search for better models is done.

5 Machine learning applications

In the previous chapter, we fitted machine learning models which were able to predict the response variable better than baseline models, based on Poisson deviance in Formula (3.1). This chapter focuses on extracting insight from machine learning models and making interpretable models based on this insight. We will be using the maidrr approach and rule ensemble for this.

5.1 maidrr modelling

We will apply the maidrr method to GBM model produced in the previous chapter. The GBM model from package gbm works natively with package maidrr since the authors of the package developed it with gbm package in mind. Although maidrr is model agnostic and any machine learning model can, in theory, be used.

Using the GBM model, we first extract the partial dependence for all variables with non-zero importance. The relative variable importance in Figure 4 corresponds to the GBM model trained in the previous chapter. Based on this, all variables besides AgrStatus, IsTrainingEquipment and Terminated had non-zero variable importance and thus, partial dependence for them was computed. Partial dependence was computed using a sample of 100 000 observations from the training set due to computational intensity. This sample is, however, 10 times bigger than the sample used by maidrr package by default.

Then, using the implementation of Algorithm A.2, the optimal grouping penalties were selected. Optimal penalties are the solution to the problem, as seen in Formula (3.5), where the optimal number of groups k_j^* is unknown. This problem is separately solved for main effects (single variable) and interaction variable effects.

This involves taking a set of potential grouping penalties, applying the penalty and seeing how well the surrogate GLM performs using cross-validation. Several runs with different sets of potential grouping penalties were done. For main effects (single variable), the optimal penalty was $\lambda_{\text{main}} = 5 \cdot 10^{-7}$ and for interaction effects, the optimal penalty was $\lambda_{\text{intr}} = 9 \cdot 10^{-6}$. The search range for both was 10^{-12} to 10^{-3}, so both penalties were around the middle of the search range. Note that smaller penalties mean more grouping levels.

One key advantage of maidrr is that it can automatically perform feature and interaction selection. For optimal penalties λ_{main} and λ_{intr}, 15 variables were selected, and in addition, 6 interactions using these variables were selected. The selected variables with optimal groupings were Ifregion (17 groups), CntPenaltyPoints (9), VehMake (15), BmClassAas (14), Drivexpbnew

(11), BmSubjectAge (24), Mileage (8), Vehyearnew (7), FullMass (6), FuelType (3), CntNotice (3), Policylength (2), EnginePower (3), AgreementTypeCode (2), SeatCount(2). The interactions selected for the surrogate model were CntPenaltyPoints and BmSubjectAge (13 groups), Ifregion and CntPenaltyPoints (11), Ifregion and BmSubjectAge (13), VehMake and BmSubjectAge (4), Drivexpbnew and BmSubjectAge (5), Ifregion and BmClassAas (7). This model will be called maidrr surrogate (encoded as "Surrogate" in tables). Appendix D has plots of grouped partial dependence for all of these variables.

It is important to note that the underlying GBM model was able to deal with missing values for variables, and this ability is also present in the surrogate. Namely, the surrogate model has a separate group containing missing values where appropriate or missing values are grouped with some group of values. Sometimes bigger values of variables are also attached to this group with missing values, indicating that missing values act similarly to those bigger values. One such example is FullMass variable, where group $[NA, NA]$ also contains vehicles with 3450 or greater full mass.

Detailed output and interpretation of the surrogate model can be found in Appendix E.1.

Now that we know what kind of groups can be used to imitate the machine learning model, augmentation for the underlying AIC model can be tested. To start out with, a forward search based on AIC was done starting from the AIC model with polynomial terms - for each grouping augmentation (for example, Ifregion variable grouping into 17 groups), AIC model equivalent variable was replaced by grouped variable coming from the maidrr surrogate model and model AIC was calculated. All of the different groupings were tested, and the augmentation providing the best gain in AIC was adopted. Then the process was repeated until there was no gain in AIC.

With this procedure, these groupings were adopted in this order: grouping for Mileage, then Drivexpbnew, then interaction between CntPenaltyPoints and BmSubjectAge, then VehMake grouping, then interaction between Ifregion and CntPenaltyPoints, then Vehyearnew, then CntNotice, then FullMass, then interaction between Ifregion and BmClassAas, then interaction between Ifregion and BmSubjectAge, then interaction between VehMake and BmSubjectAge, then EnginePower grouping, then FuelType grouping and lastly SeatCount grouping. This model will be called maidrr grouping augmented AIC model (encoded as "AIC model + group").

In addition to maidrr grouping, a second augmentation was proposed and tested. In some cases, fitting of constant to grouped value does not seem appropriate. For example, for Drivexpbnew variable (Figure 6), fitting a constant value for a group formed between 0 and 4 is not appropriate

since there is a clear non-constant drop in partial dependence for this range. To fix this, a piecewise linear approximation can be used. Using maidrr grouping augmented AIC model, each of the grouped ordered and numeric variables were replaced by their piecewise linear alternative one by one. If the piecewise alternative improved the AIC of the model, it was adopted. In the end, only one piecewise alternative improved AIC: Drivexpbnew. This model will be called maidrr grouping and spline augmented AIC model (encoded as "AIC model + spline").

A more detailed look into both maidrr grouping augmented, grouping and spline augmented AIC models are presented in Appendix E.2 and E.3, respectively.

5.2 Rule ensemble modelling

For rule ensemble, an implementation available in R package xrf was used. The procedure is carried out as described in Subchapter 3.3; however, the author of package xrf built the package focusing on fitting and extracting rules from the XGBoost model. Additionally, the author implemented rule duplication removal and rule deoverlapping.

Rule deoverlapping means fixing the structure of rules to have non or minimal overlap in segmented data produced by these rules. In practice, this means introducing additional rules and altering the original extracted rules to make disjointed segments of data. In this case, deoverlapping of rules proved to be computationally infeasible and thus was not used.

It is important to note that the out-of-the-box package xrf was not able to produce a model with satisfactory assumptions. To fix this, three things were done: a way to add XGBoost model trained outside of the package was implemented, Lasso regression model options like modelling family being Poisson was implemented, and linear term normalisation was added. Extracted rules will remain on the original scale since those rules can more easily be interpreted.

The XGBoost model trained in the previous chapter was used to make the rule ensemble. From this model, 1173 non-duplicate rules were extracted. Using these rules and the linear terms, a penalty parameter search was conducted. The penalty parameters considered were default values from glmnet package (Lasso regression backend package) (Friedman, Hastie, and Tibshirani, 2010), which were 100 logarithmically uniform values from $\Lambda_{\max}$ to $\Lambda_{\min} = 0.0001 \cdot \Lambda_{\max}$ where $\Lambda_{\max}$ is such penalty value for which all coefficients are 0. For this data we got that $\Lambda_{\max} = 1.510447 \cdot 10^{-2}$. These penalty parameter values correspond to the penalty Λ in Formula 3.8.

The fitted model finds two "optimal" penalty parameters $\Lambda_{\min} = 0.00040119$ and $\Lambda_{1\text{se}} = 0.00111633$. First corresponds to the penalty parameter achieving the lowest cross-validation er-

ror, while Λ_{1se} corresponds to the larger (in value) penalty parameter achieving cross-validation error 1 standard error (about 0.00107538 or 0.3% of the smallest cross-validation Poisson deviance (Formula (3.1))) away from the minimum. These models will be called rule fit model with minimum parameter (encoded as "RuleFit min") and rule fit model with 1se parameter (encoded as "RuleFit 1se"), respectively.

Rule fit model with minimum parameter had 377 non-zero coefficients for terms in regression. Out of these terms, 341 were for rules and 35 for original terms. For rule fit model with 1se parameter 146 terms had non-zero coefficients with 137 for rules and 8 for original terms. We can therefore say that about half of the additional non-zero terms improve the deviance by only 1 standard error amount, showing that adding more terms gives a small gain in deviance. It is also important to note that the normalisation of original numeric terms, as suggested in paper (Friedman and Popescu, 2008), was done.

A more detailed overview and a sample of rules and their coefficients of both rule fit models are available in Appendix F.1 and F.2.

5.3 Model comparison

This subchapter focuses on showcasing the difference in model accuracy metrics. Two metrics will be used to compare the models: Poisson deviance from Formula (3.1) on the test set as the performance measure and AIC as a goodness of fit (GOF) measure.

Comparison of accuracy measures based on previously unseen data is a standard approach for machine learning models. However, since one of the points of interest is to see if proposed model improvements improve the underlying model, AIC as the goodness of fit metric is used. For this, all the different GLM models will be trained on the same data, and their AIC value will be computed to asses their future performance.

To compare the models, a unified test set is necessary. Since all of the models need variables in different ways, two additional copies of the test set needed to be made. All of these test sets are identical in terms of observation ordering etc., but their underlying structure differs, like having one-hot encoding, normalisation of terms, etc. The resulting test data sets have 294 777 observations.

Additionally, it is important to note that maidrr surrogate model interaction terms are prone to producing additional missing values due to previously unseen combinations of interaction variables present in test data. This fact can not be avoided; thus, the observations producing these

Table 2: Model comparison based on *AIC* on the training set and Poisson deviance on the test set. The test set was unified to be fair for all models, as different models require different data structures. The test set contains 241 020 observations. *AIC* was found with all models retrained on training data where all missing values were omitted.

Model name	*AIC* (training)	Poisson Deviance (Test)	Number of parameters
Trivial model	-	0.11695364	-
AIC model	167150.8	0.11204148	91
AIC model + poly	166756	0.11159231	99
GBM	-	0.11146036	-
Surrogate	166279.7	0.11148752	156
Surrogate No interaction	166432.0	0.11143346	109
AIC model + group	166300.3	0.11145320	175
AIC model + spline	166294.6	0.11144887	181
XGBoost	-	0.11158860	-
RuleFit min	165945.1	0.11143408	377
RuleFit 1se	166479.5	0.11169946	146

errors were omitted. Final test data sets had 241 020 observations. This data was acceptable for all models; thus, deviance comparison was possible and fair.

In Table 2, a comparison between models is presented. As discussed earlier, the test set was made fair and comparable for all models. *AIC* value for likelihood-based models was found by retraining all models to the training data with all missing data removed. This makes the *AIC* values comparable. The number of parameters for rule fit models were chosen to be the number of non-zero coefficients since Lasso is used for feature selection in this model.

Based on the results from Table 2, we can see that all considered "augmented" models perform better than the basic *AIC* model. This is good to see since we are considering somewhat more complex models.

Moving deeper, we can see that the surrogate model containing only categorical variables is much better in terms of Poisson deviance compared to *AIC* model with polynomial terms. Based on *AIC*, the surrogate model is clearly better (by 476) compared to the *AIC* model with polynomial terms.

However, something rather odd is also happening. We can see that the surrogate model without interactions is the best model based on Poisson deviance. This is odd since the added interactions should, in theory, improve the fit of the model. There is no clear explanation why this model performs best on the test data. The fact that this model performs better than the underlying machine learning model is also strange since the underlying structure for both of these models is similar (constant prediction for a given segment of data). This might hint that the GBM model could be further improved.

Another thing that is strange is the fact that the AIC model with maidrr grouping augmentation and the AIC model with grouping and spline augmentation perform better than the underlying machine learning model, GBM, they were built on. One possible explanation for this is one of the disadvantages mentioned in Subchapter 2.1.3, related to the approximation of linear relations using splits. Both GBM and XGBoost models are very shallow models (depth of 4 for both) and have quite a low number of iterations (for GBM 691, for XGBoost 91). With a low number of iterations, it is hard to have enough splits related to linear terms; thus, only a rough sense of linear relation is captured.

I believe this is also supported by the fact that some of the terms are kept linear for both grouping augmented and grouping and spline augmented models.

A similar case holds up for XGBoost and corresponding rule fit models. Rule fit models again outperform the machine learning model they are based on, but they also contain the linear terms, thus capturing the linear relation, not captured by the tree ensemble method.

Rule fit model with a minimum parameter has similar deviance compared to the surrogate model with no interactions, showing that rules are able to describe the data in a similar way. However, we can see that rule fit model is not that stable since 1 standard error of cross-validation Poisson deviance is enough to make a model worse than the AIC model with polynomial terms (based on test deviance). This indicates that the underlying structure, although good in some cases, is very reliant on the complexity allowed by the penalty parameter. I believe that in order to make this model more stable, a better machine learning model and rules are needed.

However, looking into the rules that have non-zero coefficients has proven very insightful, showcasing which possible combination of variables might be used in ratemaking. More on this is discussed in the Subchapter 5.4.

Since all non-machine-learning models are ultimately likelihood based GLM models, then taking into account the number of parameters, we get that actually rule fit model with minimum parameter is the best, followed by the maidrr surrogate model. The difference in AIC between

these two models is 334.6, which is quite a significant decrease. The fact that the rules with some linear terms beat out piecewise constant fit is interesting. Rules are able to describe deeper interactions, however, at the cost of losing sight of simple constant or non-linear relations.

It is important to note that the best model based on deviance is not the best model based on AIC. The surrogate model without interactions was just barely able to beat rule fit model with minimum parameter based on test deviance. It might be that the interactions were not so present in the test data, but in training data it is clear that interactions are important and thus should be included if the best fit based on likelihood is desired.

Comparing the two approaches of gathering insight, we can see that maidrr is able to produce a more stable model based on purely categorical variables. However, the interpretation of interaction is much better for rule fit models. Rules give you a concrete idea how a combination of different variables affects the claim frequency, while with surrogates, you have to be very precise when interpreting the interaction for a particular observation. Overall it appears that we were able to gather some insight from the way machine learning models use variables and produce meaningful and, most importantly, interpretable augmentations for industry standard models.

5.4 Discussion

In this subchapter, a general discussion about machine learning modelling and the results of this thesis are discussed. This subchapter is based on the author's experience working with this data and the problem at hand.

To start out with, all of this work has been plagued by long waiting times. Starting out, I knew machine learning would take considerably more time than classic statistical models like GLM; however, I severely underestimated the time required to run some of the procedures and methods.

As evident in the method and model description, most of the algorithms are quite computationally intense (at least $O(n^2)$ for most), and it does not help that their implementations seldom allow for parallelisations running on one core of the machine by default. With my limited knowledge and skills, I was, however, able to improve some of the underlying code for maidrr package allowing me to run some of the code in parallel. This showcases well that working with machine learning requires additional skills in writing code and understanding the underlying structure of the method being used.

Continuing on this topic of computational difficulties, I was able to leverage the high-performance computational cluster available to the University of Tartu, thus allowing me to run several model

training sessions at the same time (University of Tartu, 2018). The cluster also allowed to more easily run the parallel computation since parallelisation in Windows (Bott and Stinson, 2019) is not allowed by default. Overall, even though all of the code could be run on a single core on a machine with only 16 GB of ram, it would have taken significantly longer since every job would need to be carefully planned and executed.

In general, working with machine learning on insurance data has proven difficult since most of the methods and implementations I found did not have the capabilities to model insurance data. They did not allow for the use of Poisson or custom objective (loss) function, thus, count or claim frequency modelling would not be optimal. The same goes for claim amount modelling using Gamma or other heavy-tailed distributions.

However, simply having a custom or Poisson objective was also not enough since modelling claim frequency requires us to use either offset for the response (given the right link between response and prediction function) or the ability to use weights for model fitting. Having both of these requirements proved to be very rare. In my experience, working in R helped with this since more implementations with both requirements were available (compared to Python libraries).

This shows that machine learning, although powerful in cases where normal or binomial distribution is used, is still not ripe enough to be freely applied to the insurance field and problems. There were no out-of-the-box solutions, and every approach and method needed a lot of additional work and prior knowledge to put to work predicting insurance data.

Regarding the results of the applied methods, I feel quite hopeful. I was able to discover similar groupings for variables used in ratemaking at the company presently, and I did this in a few months while they have been doing and developing their pricing structure for years. This is one of the key developments I achieved with maidrr and presented to the pricing unit at If.

In addition to grouping structure, If was also very interested in the rules found through rule ensembles. Rules allow us to find a segment of data historically performing abnormally compared to the rest of the observations, thus giving a reason to put additional restrictions in place for pricing this part of the population. With additional analysis, these combinations of variables might allow to gain an edge in insurance pricing in a market where all of the data is shared. Something different and more advanced has to be done to gain an edge in such a competitive environment, and I and my supervisor at If feel that this might be one of these things.

Lastly, I would like to comment that this research and thesis is by no means perfect, but it is a proof of concept to showcase what possible alternatives and additions machine learning can provide in the age of computing. It is clear that the machine learning field will keep on evolving.

However, I do not see the restriction of model and result interpretability being lifted any time soon in the insurance field. This way, a need for tools to make machine learning interpretable and, in general, interpretable machine learning will only grow in time. Machine learning is the future, but interpretable machine learning will help us to get there and understand it.

I would like to finish this discussion with a quote by Christoph Molnar: "When opaque machine learning models are used in research, scientific findings remain completely hidden if the model only gives predictions without explanations. To facilitate learning and satisfy curiosity as to why certain predictions or behaviours are created by machines, interpretability and explanations are crucial. Of course, humans do not need explanations for everything that happens. For most people, it is okay that they do not understand how a computer works. Unexpected events make us curious." (Molnar, 2022).

Conclusion

The purpose of this thesis was to introduce and showcase two ways to extract insight from machine learning models trained to evaluate risk in insurance pricing. To do this, the first 2 chapters of the thesis focused on introducing the current main statistical model, the generalized linear model, giving an overview of decision trees and tree-based boosting ensembles like gradient boosting machine and XGBoost. In Chapter 3, model metrics and insight statistics were discussed. After that, model agnostic data-driven surrogate models (maidrr) and rule ensemble methods were introduced and explained. In the last two chapters, all models and methods were applied to motor third party liability data coming from Latvia. The models were trained on the training split of the data, the resulting models were briefly explained, and their performance was assessed using the testing split of the data.

First, 4 models were fit to the training data: a GLM model based on step-wise AIC search, the AIC search model with additional polynomial terms (baseline models), gradient boosted machine and XGBoost as the machine learning models. The resulting machine learning models were better in terms of test set deviance compared to the baseline models. This gave reason to extract insight from these machine learning models and an additional 6 models were proposed: surrogate model, surrogate model without interactions, grouping augmented AIC search model, grouping and spline augmented AIC search model and two rule ensemble models with different penalty parameters.

All of the proposed machine learning insight augmented models ended up being better or comparable in accuracy to baseline generalized linear models. Based on likelihood metrics, the proposed augmentations proved to produce models that are better able to capture the likelihood of the data. Some strange behaviour was also observed, where some augmentations performed better on the test set compared to the original machine learning models. However, both ways of using machine learning models proved to be useful in different cases: maidrr is good for feature selection and grouping of these features and rule ensemble is good for searching for combinations of variables that can be priced differently.

This thesis was able to showcase two possible ways machine learning could be used to augment current practices of ratemaking. Both of the methods used show potential in their respective strengths of feature grouping for maidrr and combination search for rule ensemble. In the case of maidrr, the approach can be used for any machine learning model, and thus advancements in machine learning models applicable to the insurance field will also improve the surrogate model that can be evaluated. Rule ensembles help to gather ideas for further analysis and strategies.

References

1.10. Decision Trees (Jan. 2023). [Online; accessed 12. Jan. 2023]. URL: https://scikit-learn.org/stable/modules/tree.html.

Apache Software Foundation (July 20, 2018). *Hadoop*. Version 2.7.7. URL: https://hadoop.apache.org.

– (June 18, 2020). *Spark*. Version 2.3.4, 3.0.3. URL: https://spark.apache.org.

Bott, Ed and Craig Stinson (2019). *Windows 10 inside out*. Microsoft Press.

Breiman, Leo (Aug. 1996). "Bagging predictors". In: *Mach. Learn.* 24.2, pp. 123–140. ISSN: 1573-0565. DOI: 10.1007/BF00058655.

– (Feb. 1999). *Using Adaptive Bagging to Debias Regressions*. Technical Report 547. Berkeley, CA 94720: University of California at Berkeley, Statistics Department.

Chen, Tianqi and Carlos Guestrin (Mar. 2016). "XGBoost: A Scalable Tree Boosting System". In: *arXiv*. DOI: 10.1145/2939672.2939785. eprint: 1603.02754.

Chen, Tianqi, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, Tianyi Zhou, Mu Li, Junyuan Xie, Min Lin, Yifeng Geng, Yutian Li, and Jiaming Yuan (2022). *xgboost: Extreme Gradient Boosting*. R package version 1.6.0.1. URL: https://CRAN.R-project.org/package=xgboost.

de Jong, Piet and Gillian Z. Heller (Feb. 2008). *Generalized Linear Models for Insurance Data*. Cambridge, England, UK: Cambridge University Press. ISBN: 978-0-52187914-9. URL: https://www.cambridge.org/lv/academic/subjects/statistics-probability/statistics-econometrics-finance-and-insurance/generalized-linear-models-insurance-data?format=HB\&isbn=9780521879149.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent". In: *Journal of Statistical Software* 33.1, pp. 1–22. DOI: 10.18637/jss.v033.i01. URL: https://www.jstatsoft.org/v33/i01/.

Friedman, Jerome H. (2001). "Greedy function approximation: A gradient boosting machine." In: *The Annals of Statistics* 29.5, pp. 1189 –1232. DOI: 10.1214/aos/1013203451. URL: https://doi.org/10.1214/aos/1013203451.

– (2002). "Stochastic gradient boosting". In: *Computational Statistics & Data Analysis* 38.4. Nonlinear Methods and Data Mining, pp. 367–378. ISSN: 0167-9473. DOI: https:

//doi.org/10.1016/S0167-9473(01)00065-2. URL: https://www.sciencedirect.com/science/article/pii/S0167947301000652.

Friedman, Jerome H. and Bogdan E. Popescu (Oct. 2003). "Importance Sampled Learning Ensembles". In: *ResearchGate*. URL: https://www.researchgate.net/publication/2888930_Importance_Sampled_Learning_Ensembles.

– (Sept. 2008). "Predictive learning via rule ensembles". In: *The Annals of Applied Statistics* 2.3. DOI: 10.1214/07-aoas148. URL: https://doi.org/10.1214\%2F07-aoas148.

Greenwell, Brandon, Bradley Boehmke, Jay Cunningham, and GBM Developers (2022). *gbm: Generalized Boosted Regression Models*. R package version 2.1.8.1. URL: https://CRAN.R-project.org/package=gbm.

Hardin, James W. and Joseph M. Hilbe (Apr. 2018). *Generalized Linear Models and Extensions: Fourth Edition*. Stata Press. ISBN: 978-1-59718225-6. URL: https://www.amazon.com/Generalized-Linear-Models-Extensions-Fourth/dp/1597182257.

Hastie, Trevor, Robert Tibshirani, and Jerome H. Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY, USA: Springer. ISBN: 978-0-38784884-6. URL: https://hastie.su.domains/ElemStatLearn/.

Henckaerts, Roel (2020). *maidrr: Model-Agnostic Interpretable Data-driven suRRogate*. https://henckr.github.io/maidrr/, https://github.com/henckr/maidrr.

Henckaerts, Roel, Katrien Antonio, Maxime Clijsters, and Verbelen Roel (Jan. 2017). "A Data Driven Binning Strategy for the Construction of Insurance Tariff Classes". In: *SSRN Electronic Journal*. ISSN: 1556-5068. DOI: 10.2139/ssrn.3052174.

Henckaerts, Roel, Katrien Antonio, and Marie-Pier Côté (2020). *When stakes are high: balancing accuracy and transparency with Model-Agnostic Interpretable Data-driven suRRogates*. DOI: 10.48550/ARXIV.2007.06894. URL: https://arxiv.org/abs/2007.06894.

Henckaerts, Roel, Marie-Pier Côté, Katrien Antonio, and Roel Verbelen (Apr. 2019). "Boosting insights in insurance tariff plans with tree-based machine learning methods". In: *arXiv*. DOI: 10.48550/arXiv.1904.10890. eprint: 1904.10890.

Holub, Karl (2022). *xrf: eXtreme RuleFit*. R package version 0.2.2. URL: https://CRAN.R-project.org/package=xrf.

Kuhn, Max and Hannah Frick (2022). *dials: Tools for Creating Tuning Parameter Values*. R package version 1.1.0. URL: https://CRAN.R-project.org/package=dials.

Lamport, Leslie (1994). *LaTeX: a Document Preparation System*. 2nd ed. Massachusetts: Addison Wesley.

Luraschi, Javier, Kevin Kuo, Kevin Ushey, JJ Allaire, Hossein Falaki, Lu Wang, Andy Zhang, Yitao Li, Edgar Ruiz, and The Apache Software Foundation (2022). *sparklyr: R Interface to Apache Spark*. R package version 1.7.8. URL: https://CRAN.R-project.org/package=sparklyr.

Molnar, Christoph (2022). *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. URL: https://christophm.github.io/interpretable-ml-book.

Murdoch, W. James, Chandan Singh, Karl Kumbier, Reza Abbasi-Asl, and Bin Yu (Oct. 2019). "Definitions, methods, and applications in interpretable machine learning". In: *Proc. Natl. Acad. Sci. U.S.A.* 116.44, pp. 22071–22080. DOI: 10.1073/pnas.1900654116.

Nelder, J. A. and R. W. Wedderburn (1972). "Generalized linear models". In: *Journal of the Royal Statistical Society. Series A (General)* 135.3, 370–384. DOI: 10.2307/2344614.

R Core Team (2022). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria. URL: https://www.R-project.org/.

Tibshirani, Robert (1996). "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1, pp. 267–288. ISSN: 00359246. URL: http://www.jstor.org/stable/2346178 (visited on 01/31/2023).

University of Tartu (2018). *UT Rocket*. DOI: 10.23673/PH6N-0144.

Valecký, Jiří (May 2016). "Modelling Claim Frequency in Vehicle Insurance". In: *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis* 64.2, pp. 683–689. ISSN: 1211-8516. DOI: 10.11118/actaun201664020683.

Wang, Haizhou and Mingzhou Song (Dec. 2011). "Ckmeans.1d.dp: Optimal k-means Clustering in One Dimension by Dynamic Programming". In: *R Journal* 3.2, pp. 29–33. ISSN: 2073-4859. DOI: 10.32614/RJ-2011-015.

Wickham, Hadley, Romain François, Lionel Henry, and Kirill Müller (2022). *dplyr: A Grammar of Data Manipulation*. R package version 1.0.10. URL: https://CRAN.R-project.org/package=dplyr.

Wüthrich, Mario V. (Jan. 2019). "From Generalized Linear Models to Neural Networks, and Back". In: *SSRN Electronic Journal* 1. ISSN: 1556-5068. DOI: 10.2139/ssrn.3491790.

Appendix

Appendix A maidrr algorithms

In this appendix, two of the key algorithms for the maidrr method are given. The first algorithm focuses on generating a suitable surrogate model if penalty parameters are given and the second algorithm shows how to find optimal penalty parameters.

A.1 maidrr surrogate model algorithm

In this section, a copy of the maidrr algorithm as seen in (Henckaerts, Antonio, and Côté, 2020) is presented. Note that here we assume optimal penalty values λ_{main}, λ_{intr} are already found.

Algorithm 2 maidrr surrogate algorithm

Input: data, λ_{main}, λ_{intr}, k, h

// Main effect loop

for $j = 1$ to p **do**

Calculate $\hat{PD}(\mathbf{x}_{\{j\}})$ for all unique values of variable X_j in the data.

Apply DP algorithm to group values of X_j using $k^*_{\{j\}} = \underset{k_{\{j\}} \in \{1,\dots k\}}{\text{argmin}}$ Eq. (3.5) for $\lambda = \lambda_{\text{main}}$

Define X^c_j as the the grouped version of X_j with $k^*_{\{j\}}$ groups

end for

Feature selection: $F_{eat} = \{j | k^*_{\{j\}} > 1\}$

// Interaction effect loop

Interaction selection: $I = \{(l,m) | l, m \in F_{eat}, \text{ and } H^2_{\{l,m\}} \geq h\}$

for (a,b) in I **do**

Calculate $\hat{PD}(\mathbf{x}_{\{a,b\}})$ for all unique combinations of variables X_a and X_b in the data.

Apply DP algorithm to group interactions of $(\mathbf{X}_{\{a,b\}})$ using

$k^*_{\{a,b\}} = \underset{k_{\{a,b\}} \in \{1,\dots k\}}{\text{argmin}}$ Eq. (3.5) for $\lambda = \lambda_{\text{intr}}$

Define $X^c_{a:b}$ as the the grouped version of interaction variable $X_{a:b}$ with $k^*_{\{a,b\}}$ groups

end for

Interaction selection $I_{eat} = I \backslash \{(l,m) | k^*_{\{l,m\}} = 1\}$

Fit GLM to response using features X^c_j for $j \in F_{eat}$ and interaction $X^c_{a:b}$ for $(a,b) \in I_{eat}$

Output: Surrogate GLM

A.2 maidrr penalty tuning algorithm

In this section, an algorithm for maidrr penalty tuning is presented. This algorithm is not explicitly stated in (Henckaerts, Antonio, and Côté, 2020) but is implemented in the corresponding R package maidrr (Henckaerts, 2020). Denote $\vec{\lambda}_{\text{main}}$ and $\vec{\lambda}_{\text{intr}}$ as grids of potential values for corresponding penalty parameters.

Algorithm 3 maidrr penalty tuning algorithm

Input: data, $\vec{\lambda}_{\text{main}}$, $\vec{\lambda}_{\text{intr}}$, k, h, k_{fold}

Split data randomly into k_{fold} parts.
// Main penalty tuning
for λ in $\vec{\lambda}_{\text{main}}$ **do**
  for i in 1 to k_{fold} **do**
    Use part i of the data as validation split and rest as training split
    Run main effect loop of maidrr surrogate Algorithm 2 with $\lambda_{\text{main}} = \lambda$
    Fit a surrogate GLM using variables X_j^c for $j \in F_{eat}$ on the training split
    Calculate the validation split loss Val_i
    Save Val_i corresponding to penalty λ and fold i
  end for
end for
$\forall \lambda \in \vec{\lambda}_{\text{main}}$ calculate $\text{cv}_{err_i} = \frac{1}{k_{fold}} \sum_{i=1}^{k_{fold}} \text{Val}_i$.
Select $\lambda^*_{\text{main}} = \min_{\text{cv}_{err_i}} \vec{\lambda}_{\text{main}}$.
// Interaction penalty tuning
Using F^*_{eat} and groupings $k^*_{\{j\}}$, $j = 1, \ldots, p$ belonging to λ^*_{main}
for λ in $\vec{\lambda}_{\text{intr}}$ **do**
  for i=1 to k_{fold} **do**
    Use part i of the data as validation split and rest as training split
    Run interaction effect loop of maidrr surrogate Algorithm 2 with $\lambda_{\text{intr}} = \lambda$
    Fit a surrogate GLM using variables X_j^c for $j \in F^*_{eat}$ and interactions $X^c_{a:b}$ for $(a, b) \in I_{eat}$ on the training split
    Calculate the validation split loss Val_i
    Save Val_i corresponding to penalty λ and fold i
  end for
end for
$\forall \lambda \in \vec{\lambda}_{\text{intr}}$ calculate $\text{cv}_{err_i} = \frac{1}{k_{fold}} \sum_{i=1}^{k_fold} \text{Val}_i$.
Select $\lambda^*_{\text{intr}} = \min_{\text{cv}_{err_i}} \vec{\lambda}_{\text{intr}}$.
Fit GLM to response using features X_j^c for $j \in F^*_{eat}$ and interactions $X^c_{a:b}$ for $(a, b) \in I^*_{eat}$ corresponding to penalties λ^*_{main} and λ^*_{intr} using all of the data.

Output: Surrogate GLM with tuned values of penalties λ^*_{main} and λ^*_{intr}

Non-exclusive licence to reproduce thesis and make thesis public

I, Artur Tuttar,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, Extending generalized linear models in insurance with machine learning techniques, supervised by Meelis Käärik.

2. I grant the University of Tartu a permit to make the thesis specified in point 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 4.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in points 1 and 2.

4. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Artur Tuttar
16.05.2023