ERGO Center of Excellence in Insurance

Eine Einrichtung der TUM gefördert von der ERGO Group

Technische Universität München

Department of Mathematics

Customer Churn Prediction In The Insurance Sector Using Machine Learning Methods

Implementation in SAS

Master Thesis

by

Bassant Abed

Supervisor:	Prof. Dr. Matthias Scherer
Advisor:	Gabriela Zeller
Submission Date:	18th of June 2020

I hereby declare that this thesis is my own work and that no other sources have been used except those clearly indicated and referenced.

Munich, 18/06/2020

Abstract

To prevent customer churn in a highly competitive sector, insurance companies can leverage the potential of machine learning automation and the availability of various data sources, to enhance customer understanding. Due to the advances in standard data-mining techniques, recent studies in search of the best churn prediction models have been increasingly applying more sophisticated methods that deliver higher prediction accuracy but often lack interpretability. The aim of this thesis is to take traditional churn prediction models one step further by combining class imbalance reduction and predictive power maximisation with a perspective on interpretability. The study is conducted in collaboration with the German insurance group ERGO, with the final objective of validating and improving an existing churn prediction model and identifying churn drivers for clients with a likelihood of cancelling their active contract in the upcoming quarters. An extensive feature set characterizing clients' data is systematically investigated using data-mining techniques to identify relevant churn drivers. In the advanced analytics software SAS Enterprise Guide and SAS Enterprise Miner, five prediction models are implemented, namely logistic regression, decision trees, random forests, gradient boosting, and neural networks. Through data pre-processing, class imbalance and dimensionality reduction, models with higher Precision are achieved. Supported by the results of unbiased evaluation metrics, a fine-tuned random forest model delivers the most promising results for churn identification. On the provided dataset, machine learning models outperform classic regression by delivering higher Sensitivity, Precision, and Cumulative Lift values. Beyond building prediction models, an insight into extractable actions and interpretability methods is provided to allow marketeers to take post-modelling effective action. The final determined models could help boost customer retention by applying the suggested actions and providing the desired customer understanding.

Zusammenfassung

Um den Verlust von Kunden in einem hoch kompetitiven Sektor zu verhindern, können Versicherungsunternehmen das Potenzial der Automatisierung durch modernes maschinelles Lernen und diverse Datenquellen nutzen, um ihr Kundenverständnis zu verbessern. Aufgrund der Fortschritte bei den gängigen Data-Mining Techniken wenden neuere Studien auf der Suche nach den besten Storno-Vorhersagemodellen zunehmend komplexere Methoden an, die zwar eine höhere Vorhersagegenauigkeit liefern, deren Interpretation jedoch häufig schwierig ist. Das Ziel dieser Arbeit ist es, traditionelle Storno-Vorhersagemodelle weiterzuentwickeln, um unter Berücksichtigung des Klassenungleichgewichts hohe Vorhersagekraft und potenziell verbesserte Interpretierbarkeit zu kombinieren. Die Studie wird in Zusammenarbeit mit der deutschen Versicherungsgruppe ERGO durchgeführt, um einerseits ein bestehendes Storno-Vorhersagemodell zu validieren und zu erweitern und andererseits Stornotreiber für Kunden, die in den kommenden Quartalen eine hohe Kündigungswahrscheinlichkeit für ihren aktiven Vertrag aufweisen, zu identifizieren. Der in dieser Arbeit bereitgestellte Datensatz umfasst zahlreiche Eingangsvariablen zur Charakterisierung jedes Kunden, welche mithilfe von Data-Mining Techniken systematisch untersucht werden, um relevante Stornotreiber zu identifizieren. In der Software SAS Enterprise Guide und SAS Enterprise Miner wird zunächst eine geeignete Vorverarbeitung durchgeführt (Bereinigung des Datensatzes, Dimensionsreduktion, Reduzierung des Klassenungleichgewichts), bevor die fünf Vorhersagemodelle logistische Regression, Entscheidungsbäume, Random Forest, Gradient Boosting und neuronale Netze implementiert werden. Die Modelle werden anhand geeigneter Performance-Metriken verglichen, wobei ein Random Forest Modell die vielversprechendsten Ergebnisse für die Identifizierung von Storno liefert. In dem in dieser Arbeit verwendeten Datensatz übertreffen Machine Learning Modelle die klassische logistische Regression, insofern, dass sie höhere Werte für Sensitivität, Präzision und kumulativen Lift liefern. Über die Erstellung von Vorhersagemodellen hinaus wird ein Einblick in einsetzbare Maßnahmen und Interpretierbarkeitsmethoden gegeben, sodass Versicherer aufbauend auf der Modellierung effektive Maßnahmen zur Storno-Reduktion ergreifen können. So können die vorgestellten Modelle zur Kundenbindung beitragen, indem relevante Storno-Treiber rechtzeitig identifiziert und anschließend die vorgeschlagenen Maßnahmen angewendet werden, um die Abwanderung von Kunden zu verhindern.

Acknowledgements

I sincerely want to thank Prof. Dr. Matthias Scherer for allowing me to write this thesis under his supervision, enabling many scientifically enriching discussions and his patient guidance, encouragement, and advice. I would also like to express my deepest gratitude to my advisor Gabriela Zeller for her grateful assistance, suggestions, and continuous motivation. Moreover, I would like to give thanks to all my supervisors from ERGO for their project management support, insights, and coaching. Finally, I am very grateful to my family and friends for their endless support and patience.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the age of artificial intelligence, high competition levels in the insurance sector have driven forward-thinking companies to employ advanced machine learning methods, providing them a competitive advantage. The potential of machine learning automation and various data sources, enhancing customer understanding, can be leveraged by insurance companies to reduce churn in times where high customer retention is crucial. Most often, the profitability of a business is directly related to the growth of its customer base, especially in the insurance sector, and can be more accurately predicted by forecasting the expected churn rate. Churn rate is the percentage at which a customer or several customers terminate service provided by a business in the future. Forecasting churn would enable insurance companies to dynamically adjust current short and long-term business decisions, seizing the opportunity to be one step ahead of their competitors. Since the cost of acquiring new customers is proven to be usually higher than keeping existing customers, see Kaya et al. (2018), companies should consider clients as a long-term asset and initially maximise the potential in the area of customer retention. Knowing the clients' preferences and expectations is thus part of the long-term success of a business.

The main objective of this thesis is to construct a model and determine the techniques that best identify the highest possible number of clients intending to cancel their outstanding insurance contract while preserving appropriate Precision and Accuracy levels. Moreover, an understanding of the customer characteristics vital to predicting customer churn behaviour should be provided by identifying relevant churn drivers. An identification of contracts with a higher cancellation probability, i.e. a ranked list of at-churn-risk customers based on assigned churn score and identified churn behaviour is of further interest. The motivation is reducing customer churn by identifying those potentially valuable candidates beforehand and taking targeted proactive action by creating customer profiling. The study estimates the frequency of churn with respect

to suitable time-period-cohorts of the historical data and the repsective months before contract end, during which customers are more likely to churn. The above research questions are addressed by conducting a case study in collaboration with the insurance group ERGO, on an available real dataset and by validating the proposed model on out-of-sample data.

The framework to reach these objectives includes several subtasks. First, an insight into background sources of the study is given, accompanied by introductory machine learning foundations in Chapter 2.1. The methodology of machine learning models is hardly separable for proper analysis and practical implementation; thus, it is addressed in detail in Chapter 3.1. Available literature such as Hastie, Tibshirani, and Friedman (2001) and Shalev-Shwartz and Ben-David (2014) present the previous works by J.R. Quinlan, J.H. Friedman, and L. Breiman, providing a base of the implemented methodology. In consideration of the dataset's binary response, namely a labelling variable of churn ($Y = 1$) or non-churn ($Y = 0$), classification methods are applied. First, the classic logistic regression method, which serves as a benchmark to measure the performance of the more complex methods is presented. As a base for the more sophisticated tree-based methods, decision trees are subsequently introduced. Tree-pruning strategies and overfitting issues are further analysed. Based on the decision tree algorithm, the random forest approach is regarded to incorporate an ensemble technique in our analysis. Next, gradient boosting is considered to represent an alternative ensemble family of machine learning methods. Finally, the predictive power of neural networks is examined by analysing the architecture and possible optimization procedures.

The dataset is pre-processed, cleaned, and mined before practical models are implemented, as described in Chapter 4 by applying the presented data-mining methods in Section 3.2. Tackled challenges include data partition, dimensionality reduction, as well as the transformation of biased and incomplete data. A further data-mining issue, namely underlying class imbalance, is solved. Class imbalance and resulting possible misclassification problems are overcome by applying sampling procedures on the data. Subsequently, a feature selection is performed to identify the variables contributing the most to the prediction variable. The feature subset is generated by considering wrapper methods such as stepwise selection and variants of decision trees. Next, the five models are implemented on the given case study following the same sequence as they are introduced. Thereafter, the proposed models of the use case are cross-validated, its results evaluated by examining several appropriate measures in Chapter 5 and its findings discussed in Chapter 6. The advanced analytics data-mining tools SAS Enterprise Miner and SAS Enterprise Guide are used to carry out the above methods. Applied procedures and macros are mentioned afterwards at the corresponding step and in Appendix A.2.3.

Chapter 2

Background and Notation

2.1 Background and Existing Literature

For churn-exposed industries, we find a variety of implemented methods in many fields if we consider current literature related to customer churn and different data-mining techniques used to predict churn. These include banking and insurance, telecommunication, retail markets, vacation rentals, and others. The most common goal is to identify relevant churn drivers and predict the probability that a given customer will churn, since customer attrition is one of the biggest threats to any industry. Furthermore, it would be vital to know in which of the churn-prone cases, an application of potentially costly measures is worthwhile. We carry out a short review of some of the presently available literature on the topic to gain some insight into possible methods and enhancement opportunities we can contribute to.

In order to tackle and reduce possible churn cases, the common reasons for churn need to be understood. Shah, Shah, and Rahevar (2018) provide a good overview of the process and steps to specify a churn prediction model and the different causes and types of churn. According to this research, churn types can be divided into voluntary and non-voluntary categories from a customer perspective and into contractual and non-contractual categories from a business perspective. Voluntary churn occurs when the customers themselves decide to end the contract, whether deliberately or not, whereas non-voluntary churn represents the case where the company itself decides to no longer provide the service for its customer. The recommended model specification steps include data acquisition and assessment, goal definition, data exploration and adjustments, dimensionality reduction through feature selection, modelling, scoring and comparing, and eventually deploying the outbound model. We consider this as a benchmark for our

process and include or enhance all appropriate steps. The study by Shah, Shah, and Rahevar (2018) lists some of the common causes of contract cancellation, which intuitively include general client dissatisfaction with regards to the provided service, possible high premiums in contrast to competitors, security issues, or new administrations. Bearing these causes in mind can give us a hint about possible variable importance in our models; for example, a client's previous complaints behaviour could be a possible factor since an overall dissatisfaction can lead the customer to churn.

Taking into account the provided background and introduction to the churn framework, the available machine learning models need to be compared and selected for the application. Vafeiadis et al. (2015) provide a comparative study of the most common machine learning techniques used to predict customer churn in the telecommunication industry. The five methods compared in their study include support vector machine, neural networks, decision tree, regression analysis, and Naive Bayes. The Monte Carlo simulation shows that neural networks and decision tree are the two best classifiers on the given churn use case dataset. In general, Boosting was proven to improve the accuracy of all methods. A further takeaway from this study is to apply one type of boosting techniques such as gradient boosting on our case study.

To have an understanding of possible outcomes and conventional evaluation methods used, we also take a look at a case study predicting churn on imbalanced datasets like the use case on hand. Burez and Van den Poel (2009) highlight that the issue of class imbalance has not been addressed a lot in rare event prediction research, and draws attention to its equal importance in the churn modelling procedure. It is shown that non-standard metrics improve predictive performance, as well as sampling and modelling techniques such as gradient boosting, which provide excellent results on an imbalanced dataset. These remarks are taken into consideration as we would include a section on possible resolutions of rare event prediction challenges.

It is necessary to preliminarily introduce important terms and frameworks which apply to the research's objectives and case study's scope.

Supervised Learning

Machine learning algorithms can be divided into two learning categories, namely supervised and unsupervised learning (see Gareth et al. (2013), pp. 26-27). Given a dataset including both input variables and the respective paired output, supervised learning is the procedure of inferring a function mapping the input to the output from these instances. Unsupervised learning, in contrast, detects previously non-deducted patterns in the data with no pre-existing labels and with minimal human intervention. Regarding the use case

investigated in this research, the provided dataset includes both input variables and a labelling target variable shifting the main focus on supervised learning techniques.

Classification

One instance of supervised learning known as 'classification' is a kind of categorization (see Gareth et al. (2013), pp. 27-28). In machine learning, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs. The labelling target of the underlying case study is often a variable of binary type, i.e. requiring classification of the observations into one of the two available classes. Identifying one of the qualitative events can be achieved by applying classification methods, which are the main focus of this research.

Predictive Accuracy - Model Interpretability Trade-Off

In general, as the flexibility of a method increases, its interpretability decreases (see Gareth et al. (2013), pp. 24-26). A model's flexibility refers to its level of restrictions concerning input and output specifics. It is essential to keep in mind the desired goal throughout the study to determine the optimal trade-off for this unique dataset and research goal.

Bias-Variance Trade-Off

The applied statistical learning methods in this research will always show competing properties, specifically bias versus variance. Variance refers to the generalisability error of the model on a different dataset, and bias measures the deviation of the predicted value from the real value. It is important to note that as we use more flexible methods, the variance will increase, and the bias will decrease (see Gareth et al. (2013), pp. 33-36).

Considering the already carried out methods by previous studies and the standard machine learning techniques, we apply models like decision trees, random forests, neural networks, logistic regression, and gradient boosting to compare the performance on the use case and dataset on hand to these previous findings. Throughout the analysis of the results, the trade-offs introduced above are additionally taken into account.

2.2 Mathematical Notation

This section serves the reader with the main conventions in this research. If an object is not defined as standard in Table 2.1, it is defined on the corresponding spot.

Symbol	Meaning
$\mathbb{R}$	the set of real numbers.
$\mathbb{R}^d$	the set of d-dimensional vectors over $\mathbb{R}$.
$\mathbb{N}$	the set of natural numbers.
$\mathbb{Z}$	the set of integers.
$\mathbb{X} \in \mathbb{R}^{N \times M}$	$N \times M$-dimensional matrix over $\mathbb{R}$.
X	univariate random variable.
$\boldsymbol{X}$	multivariate random variable.
$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$	vectors.
x	scalar.
x_{ij}	element of matrix $\mathbb{X}$ in the i-th row and j-th column.
$\langle x, y \rangle$	$= \sum_{i \in \mathbb{N}} x_i y_i$ space - inner product.
$x^\top$	transpose of x.
log	the natural logarithm.
exp	the exponential function.
$\mathcal{X}$	instances domain (set).
$\mathcal{Y}$	labels domain (set).
$\mathcal{H}$	hypothesis class (set).
$\mathcal{L}$	loss function.
$\mathbb{1}_{[\text{expression}]}$	indicator function equals 1 if expression is true and 0 otherwise.

Table 2.1: Mathematical Notation

Abbreviation	Meaning
LR	Logistic Regression.
RF	Random Forest.
GB	Gradient Boosting.
NN	Neural Network.
CV	Cross Validation.
CVE	Cross Validation Error.

Table 2.2: Abbreviations

Chapter 3

Modelling Process and Methods

3.1 Theoretical Methodology

3.1.1 Logistic Regression

The process of estimating and modelling the relationship between a dependent variable (also 'response' or 'outcome') and one or more independent variables (also 'features', 'predictors' or 'covariates') has become an essential statistical task in all disciplines. The classic linear regression model is used to explain the relationship between the response and features by assuming the underlying association is linear. Sometimes the response variable is not a numerical value but merely a selection of one of two possible outcomes, i.e. a binary response. For example, will a customer churn or not? Will a client test positive or not? Will a fraud case occur or not? These instances can be represented by a binary variable with only two outcomes, namely 1 if the event occurs or 0 if the event does not occur, respective to the individual event definition. Modelling a binary response requires performing some adjustments to the assumptions attributed to the linear regression method, which led to the rise of the logistic regression model. The goal of this model is to classify individual cases by predicting the respective discrete outcome correctly. Furthermore, a display of the input-output relationship strength or extent of association can be derived from this model. However, logistic regression still belongs to the linear methods for classification, i.e. linear classifiers. A hypothesis class includes all possible forms of hypotheses that are being examined. Similar hypothesis classes to logistic regression are, for instance half-spaces and linear regression (see Shalev-Shwartz and Ben-David (2014), pp. 126-127). This section examines the analysis possibility of binary response data through the logistic regression method.

First of all, to determine the model's dimensions, the given dataset needs to be examined. Given M input variables and N data observations, we consider the feature matrix $\mathbb{X}$,

$$\mathbb{X} = \begin{pmatrix} x_{11} & \dots & x_{1M} \\ x_{21} & \dots & x_{2M} \\ \vdots & \dots & \vdots \\ x_{N1} & \dots & x_{NM} \end{pmatrix} \in \mathbb{R}^{N \times M},$$

$$\text{i.e. } \mathbb{X} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_N})^\top, \boldsymbol{x_i} \in \mathbb{R}^M =: \boldsymbol{\mathcal{X}},$$

to represent our independent variables in a summarised form. The input variables can be of any form, such as categorical, discrete, or continuous, since the model does not postulate restrictions or assumptions about their distribution. Nevertheless, input variables should not be highly correlated as it might affect the quality of the estimation (see Ranganathan, Pramesh, and Aggarwal (2018)). Additionally, it is required that all given observations are independent and identically distributed (i.i.d.) (see Shalev-Shwartz and Ben-David (2014), p. 38).

Compared to the classic linear regression, which predicts a numerical outcome for the given input values, the logistic regression model aims at predicting a discrete outcome by forecasting the occurrence probability of the different possible classes. This means, our response variable values y_i have to belong to one of C predetermined labelling classes. $\mathcal{C}$ is a finite set of integers.

$$\boldsymbol{y} \in \{1, \dots, C\}^N =: \mathcal{C}^N,$$
$$\boldsymbol{y} = (y_1, \dots, y_N)^\top, y_i \in \mathcal{C}.$$

Given a target-labelled training set of a case study, the goal is to model the input-output relationship by finding a mapping function (or classifier) $f : \mathbb{R}^M \to \mathcal{C}$ such that the vectors of feature variables are mapped to a corresponding predicted event label which can be expressed by the following equation:

$$\widehat{y_i} = f(\boldsymbol{x_i}),\ \forall i \in \{1, \dots, N\},$$
$$\widehat{\boldsymbol{y}} = (\widehat{y_1}, \dots, \widehat{y_N}).$$

This results in a finite partition of the feature space such that

$$\mathcal{X} = \bigcup_{\widehat{y_i} \in \mathcal{C}} \mathcal{X}_{\widehat{y_i}} \text{ where } \mathcal{X}_{\widehat{y_i}} = \{\boldsymbol{x_i} \in \mathcal{X} : f(\boldsymbol{x_i}) = \widehat{y_i}\}.$$

In order to validate the used approach, the mathematical framework will be initially described. First, since logistic regression is part of the family of linear classifiers, its components can be expressed by some set of linear functions (see Shalev-Shwartz and

Ben-David (2014), pp. 126-127). We analyse the definition of the set A_M of affine functions (see Shalev-Shwartz and Ben-David (2014), p. 117):

$$\begin{aligned} A_M &= \{h_{b_0,\boldsymbol{b}}(\boldsymbol{x_i}) : b_0 \in \mathbb{R}, \boldsymbol{b} \in \mathbb{R}^M, i \in \{1,\dots,N\}\} \\ &= \{\boldsymbol{x_i} \mapsto b_0 + \langle \boldsymbol{b}, \boldsymbol{x_i}\rangle : b_0 \in \mathbb{R}, \boldsymbol{b} \in \mathbb{R}^M, i \in \{1,\dots,N\}\}, \text{ where} \\ h_{b_0,\boldsymbol{b}}(\boldsymbol{x_i}) &= b_0 + \langle \boldsymbol{b}, \boldsymbol{x_i}\rangle = b_0 + \sum_{j=1}^{M} b_j x_{ij}, \quad \forall i \in \{1,\dots,N\}. \end{aligned}$$

This set expresses all mapping functions using the parameters $b_0 \in \mathbb{R}$, $\boldsymbol{b} \in \mathbb{R}^M$ where each mapping returns a classifying scalar $b_0 + \langle \boldsymbol{b}, \boldsymbol{x_i}\rangle$, given the input vectors $\boldsymbol{x_i}$. Through a composition of a function $\phi : \mathbb{R} \to \mathcal{Y}$ on the set A_M one receives the corresponding hypothesis class. As logistic regression includes the prediction of the probability of an event occurring, a mapping function to the interval $[0, 1]$ is needed. Hence, the S-shaped sigmoid function $\phi = \text{sig}(t) = \frac{1}{1+e^{-t}}$ is often chosen (see Figure 3.1).

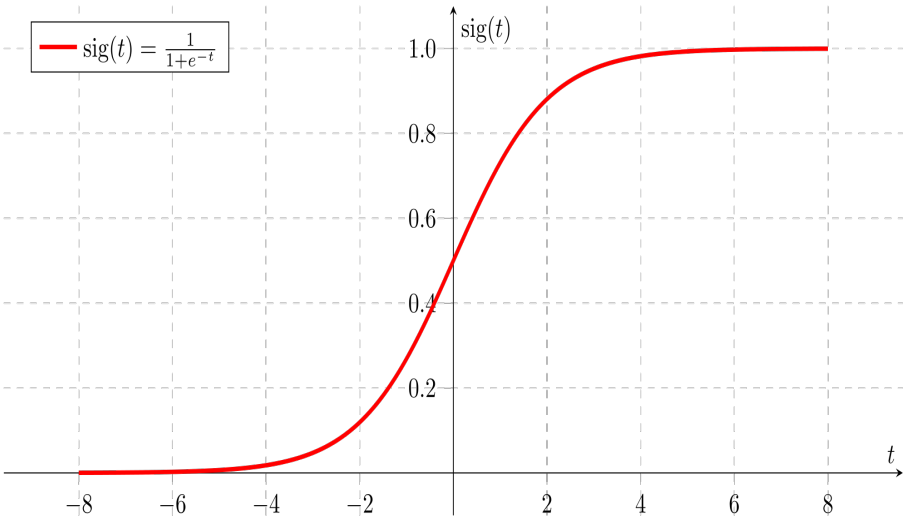


Figure 3.1: Sigmoid Function By Arunava (2018)

To take a more in-depth look at the origin of the chosen function, one must think of the prediction goal as follows. Since one wants to predict the likelihood that an event occurs i.e. $p = \mathbb{P}(y_i = 1)$, it can be derived from the odds expression, that is the quotient of the probability of an event occurrence p and its complement $1 - p = \mathbb{P}(y_i = 0)$, see left side of Equation (3.1). This is also called 'odds in favour of' the event.

$$\frac{p}{1-p} = b_0 + \langle b, \boldsymbol{x_i}\rangle. \tag{3.1}$$

The right side of Equation (3.1) can take any real value in $]-\infty, \infty[$, which is unreasonable in the logistic regression model as mentioned before. We choose p and the respective link

function in a way to restrict their values to $[0, 1]$. A transformation to the log odds (Logit) by taking the logarithm of the odds also makes the quotient easier to interpret and analyse. Logit is symmetric around 0, which results in better interpretability and adaptivity. Hence, the regression is performed against the Logit of the response and not the response variable itself. Since the parameter b_0 accounts for the intercept term of the model, we can express the right side of Equation (3.1) through $\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle$ with $\bar{\boldsymbol{b}} = (b_0, \boldsymbol{b})^\top$ and $\bar{\boldsymbol{x}}_{\boldsymbol{i}} = (1, \boldsymbol{x}_{\boldsymbol{i}})^\top$.

$$\log\left(\frac{p}{1-p}\right) = \langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle$$
$$\frac{p}{1-p} = \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)$$
$$p = \frac{\exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)}{1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)} = \frac{1}{1 + \exp(-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)} = h_{\bar{\boldsymbol{b}}}(\boldsymbol{x}_{\boldsymbol{i}}).$$

Since $e^x > 0 \; \forall x \in \mathbb{R}$ and $\frac{x}{1+x} < 1 \; \forall x \geq 0$, $0 < p < 1$ holds as desired.

A second aspect to be examined are the corresponding distributions of the variables and events. Observing the basic linear mapping expression

$$Y(\bar{\boldsymbol{x}}_{\boldsymbol{i}}) = \langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle + \epsilon \; , \; \epsilon \sim \mathcal{N}(0, \sigma^2),$$

it is assumed that the response variable of the linear regression model, given the observations and model parameters, has the following probability distribution:

$$Y \mid \bar{\boldsymbol{x}}_{\boldsymbol{i}}, \theta \sim \mathcal{N}(\mu(\bar{\boldsymbol{x}}_{\boldsymbol{i}}, \bar{\boldsymbol{b}}), \sigma^2(\bar{\boldsymbol{x}}_{\boldsymbol{i}})) \text{ with } \theta = (\bar{\boldsymbol{b}}, \sigma^2) \text{ model parameters.}$$

Considering a two - class problem (class 0 vs. class 1), hence $\mathcal{C} = \{1, 2\}$ and the given input variables $\mathbb{X}$, we can derive the probabilistic classification model, see Equations (3.2) - (3.5), where the response is evaluated as an indicator variable (see Murphy (2012), p. 245). We also view the response variable values as realisations of the random variables Y_i with $\boldsymbol{Y} = (Y_1, \ldots, Y_N)$:

$$Y_i \mid \bar{\boldsymbol{x}}_{\boldsymbol{i}}, \bar{\boldsymbol{b}} \sim Ber(Y_i \mid \mu(\bar{\boldsymbol{x}}_{\boldsymbol{i}})) \tag{3.2}$$
$$\mathbb{P}(Y_i = 1 \mid \bar{\boldsymbol{x}}_{\boldsymbol{i}}) + \mathbb{P}(Y_i = 0 \mid \bar{\boldsymbol{x}}_{\boldsymbol{i}}) = 1 \tag{3.3}$$
$$E[Y_i \mid \bar{\boldsymbol{x}}_{\boldsymbol{i}}] = \mathbb{P}(Y_i = 1 \mid \bar{\boldsymbol{x}}_{\boldsymbol{i}}) = \mu(\bar{\boldsymbol{x}}_{\boldsymbol{i}}) = \text{sig}(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle) \tag{3.4}$$
$$\mathbb{P}(Y_i \mid \bar{\boldsymbol{x}}_{\boldsymbol{i}}, \bar{\boldsymbol{b}}) = Ber(Y_i \mid \text{sig}(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)) \tag{3.5}$$

Therefore, if all cases of the probability function are considered, the logistic model can be represented by:

$$\mathbb{P}(Y_i = y_i) = \begin{cases} \frac{\exp\left(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle\right)}{1+\exp\left(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle\right)} = p_i & \text{if } y_i = 1, \\ \frac{1}{1+\exp\left(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle\right)} = 1 - p_i & \text{if } y_i = 0. \end{cases} \tag{3.6}$$

A transformation of this estimated probability into a binary variable yields the predicted model response. To proceed with the transformation all samples in the training set $\mathcal{D}$, $(\boldsymbol{x_i}, y_i) \in \mathcal{D}$ are assumed to be i.i.d. The conditional likelihood function can then be expressed as:

$$\begin{aligned}
L(\bar{\boldsymbol{b}}, \mathcal{D}) = \mathbb{P}(\boldsymbol{Y} \mid \mathbb{X}, \bar{\boldsymbol{b}}) &\overset{i.i.d.}{=} \prod_{i=1}^{N} \mathbb{P}(Y_i = y_i \mid X_i = \bar{\boldsymbol{x}}_{\boldsymbol{i}}) \\
&= \prod_{i=1}^{N} \mathbb{P}(Y_i = 1 \mid \bar{\boldsymbol{x}}_{\boldsymbol{i}}, \bar{\boldsymbol{b}})^{y_i} (1 - \mathbb{P}(Y_i = 1 \mid \bar{\boldsymbol{x}}_{\boldsymbol{i}}, \bar{\boldsymbol{b}}))^{1-y_i} \\
&\overset{(3.6)}{=} \prod_{i=1}^{N} {p_i}^{y_i} (1 - p_i)^{1-y_i} \\
&= \prod_{i=1}^{N} \left(\frac{\exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)}{1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)} \right)^{y_i} \left(\frac{1}{1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)} \right)^{1-y_i} \\
&= \prod_{i=1}^{N} \frac{\exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)^{y_i}}{1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)}.
\end{aligned} \tag{3.7}$$

It is clear from Equation (3.7) that it is cumbersome to find $\bar{\boldsymbol{b}}$ such that the likelihood function $L(\bar{\boldsymbol{b}}, \mathcal{D})$ is maximised. An alternative approach is to maximise the log-likelihood which in turn maximises the likelihood since the log function is strictly monotonic.
Applying the log function on Equation (3.7) results in:

$$\begin{aligned}
\log(L(\bar{\boldsymbol{b}}, \mathcal{D})) &\overset{(3.7)}{=} \log \left(\prod_{i=1}^{N} \frac{\exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)^{y_i}}{1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)} \right) \\
&= \sum_{i=1}^{N} y_i \langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle - \sum_{i=1}^{N} \log[1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)].
\end{aligned} \tag{3.8}$$

Moreover, to maximise the log-likelihood (3.8) we need to set the first derivative with respect to each $\bar{b}_j$ equal to 0 and solve the $M + 1$ resulting equations:

$$\begin{aligned}
\frac{\partial \log(L(\bar{\boldsymbol{b}}, \mathcal{D}))}{\partial \bar{b}_j} &= \frac{\partial}{\partial \bar{b}_j} \left(\sum_{i=1}^{N} y_i (\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle) \right) - \frac{\partial}{\partial \bar{b}_j} \left(\sum_{i=1}^{N} \log[1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)] \right) \\
&= \sum_{i=1}^{N} x_{ij} y_i - \left(\frac{1}{1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)} \right) \cdot \frac{\partial}{\partial \bar{b}_j} \left(1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle) \right) \\
&= \sum_{i=1}^{N} x_{ij} y_i - \left(\frac{1}{1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)} \right) \cdot \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle) \cdot x_{ij} \\
&\overset{(3.6)}{=} \sum_{i=1}^{N} x_{ij} \left(y_i - p_i \right).
\end{aligned} \tag{3.9}$$

The resulting equations to be solved for the optimal vector of parameter estimates $\bar{\boldsymbol{b}}$ are non-linear in $\bar{b}_j$. There exists no closed form solution for the equations and only an

iteration procedure can be used to reach the optimal solution. Different optimisation techniques are used to compute the optimal $\bar{\boldsymbol{b}}$. Most commonly, to solve Equations (3.9) the Newton-Raphson method is applied (see Murphy (2012), pp. 249-250). After numerically estimating the vector $\bar{\boldsymbol{b}}$, proving that the Hessian matrix is negative definite and that $\bar{\boldsymbol{b}}$ is a global maximum, one can use $\bar{\boldsymbol{b}}$ to model the relationship between the response and features.

Another approach to compute the parameter estimates is by defining a loss function that accounts for the cost of misclassification. Similar to least squared error in logistic regression that gives a convex function, a convex loss function is needed to optimise the parameters through standard methods by easily providing the global optimum. The function should assign a proper penalty when predicting 1 while the actual target is 0, for instance and vice versa. A suitable cost or loss function can be expressed by (see Shalev-Shwartz and Ben-David (2014), p. 127):

$$\mathcal{L}(h_{\bar{\boldsymbol{b}}}(x), y_i) = \begin{cases} -\log\left(h_{\bar{\boldsymbol{b}}}(x)\right) & \text{if } y_i = 1, \\ -\log(1 - h_{\bar{\boldsymbol{b}}}(x)) & \text{if } y_i = 0. \end{cases} \tag{3.10}$$

Hence, we can summarise the loss function (3.10) in the following manner:

$$\begin{aligned}
\mathcal{L}\left(h_{\bar{\boldsymbol{b}}(x)}, y_i\right) &= -[y_i \log(h_{\bar{\boldsymbol{b}}}(x)) + (1 - y_i)\log(1 - h_{\bar{\boldsymbol{b}}}(x))] \\
&= -\left[y_i \log\left(\frac{1}{1 + \exp(-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)}\right) + (1 - y_i)\log\left(1 - \frac{1}{1 + \exp(-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)}\right)\right] \\
&= -\left[-y_i \log(1 + \exp(-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)) + (1 - y_i)\log\left(\frac{\exp(-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)}{1 + \exp(-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)}\right)\right] \\
&= -[-y_i \log(1 + \exp(-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle)) + (1 - y_i)[-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle - \log(1 + \exp(-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle))]] \\
&= -[-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle + y_i \langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle - \log(1 + \exp(-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle))] \\
&= -[y_i \langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle - \log(1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle))].
\end{aligned}$$

This loss function is monotonic w.r.t. y_i and convex w.r.t. $\bar{\boldsymbol{b}}$. (See Proof in Appendix A.1.1)

The aim is to minimise the loss function as it penalises $h_{\bar{\boldsymbol{b}}}$ based on the log of the expression:

$$\underset{\bar{\boldsymbol{b}} \in \mathbb{R}^{M+1}}{\arg\min} \sum_{i=1}^{N} -[y_i \langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle - \log(1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle))].$$

Minimising the loss function is equivalent to the maximum likelihod estimation (MLE) (3.9), as clearly $\mathcal{L}(h_{\bar{\boldsymbol{b}}}(x), y_i)$ is equal to the negative log-likelihood derived before in Equation (3.8) (see Shalev-Shwartz and Ben-David (2014), p. 127).

Now that it is comprehensive how to reach the parameter estimates needed for the model, it is just as important to understand which features should be considered in the model.

Feature Selection

While Section 4.3 will explain in detail how each of these selections works, an overview of the possible variations is provided to compare the different techniques of logistic regression. As previously mentioned, highly correlated features can affect the performance of logistic regression. Therefore, running selection procedures to choose the most important variables is a vital step before estimating the parameters. One can determine the best combination of relevant features through a forward, backward, or stepwise selection procedure.
According to Hastie, Tibshirani, and Friedman (2001) (pp. 58-59), the forward selection technique begins with a model including only the intercept and sequentially adds predictors improving the model fit. In comparison, backward selection begins with the full model, including all possible features, and sequentially eliminates input variables with the least impact on the model's quality (see Figure 3.2). Stepwise selection is a combination of both previous selection methods. This is done by testing the effect of the features' inclusion or elimination on the model fit at each step.

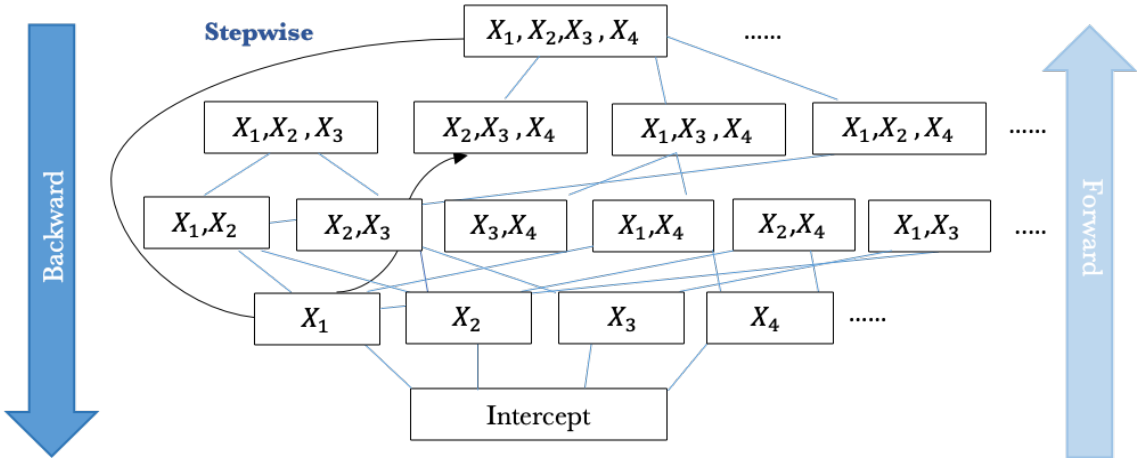


Figure 3.2: Wrapper Selection Techniques

Inspired by Rüping (2006)

Further model variations come in the form of a wide range of significance measures. The importance of the features can be assessed by performing statistical tests on the estimated coefficients. Two possible tests are the Wald statistic and the likelihood ratio test. The goodness-of-fit for the logistic model is then measured by performing either the χ^2 -test or R^2 test. These tests are explained extensively in Sections 3.2.3 and 3.2.5.

Regarding the scope of logistic regression, one needs to mention the dependence of its efficiency on the extent of association between the covariates and the response variable. Feature engineering, such as variables selection and transformation, influences the performance significantly.

On the one hand, the logistic regression technique is very efficient and does not require many computational resources in comparison to more sophisticated methods. The simplicity of logistic regression makes it highly interpretable, which is an excellent asset when specific measures need to be derived. It can additionally serve as a benchmark for more complex models.
On the other hand, the model works poorly with highly correlated variables. The model is also limited to specific data types, such as non-linearly separable data. Data is said to be linearly separable if it is possible to draw a linear separating line between different classes. In the case of linearly separable data, the sigmoid function gets sharper, driving the model to a possible overfitting (see Adams (2018)). A possible solution, for instance is a regularisation of the large weights that cause this overfitting. To overcome these dependencies and correlations, certain measures are considered in Section 4.3.

To make the model more intuitive, we consider the insurance case study examined in this research as an example. Since logistic regression is one of the most often used basic statistical learning models and is implemented when the response variable is categorical, we consider it a reasonable technique for a churn-labelled dataset (target variable is either 1 in case of a churn event or 0 in a non-churn event) to serve as a benchmark for our more complex models. In that scenario, the probability function represents these cases:

$$\mathbb{P}(Y_i = y_i) = \begin{cases} \frac{\exp\left(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle\right)}{1+\exp\left(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle\right)} & \text{if } y_i = 1, \text{ i.e. if client i has a churn label of 1,} \\ \frac{1}{1+\exp\left(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle\right)} & \text{if } y_i = 0, \text{ i.e. if client i has a non-churn label of 0.} \end{cases}$$

The modelling of churn behaviour can require some extra pre-processing as many explanatory variables have to be taken into account, including factors reflecting deviating human behaviour. Hence, to reduce the dimensionality of the M input variables, the most important variables can be selected by computing selection metrics and ranking the variables accordingly. Some general pre-processing of the features is usually necessary, for instance the imputation of missing data and removal of highly correlated variables. Subsequently, forward, backward, or stepwise selection procedure should be performed to respectively select the most valuable combination of covariates for the model. The parameter estimates are then computed for the subset of these variables by choosing the Logit link function and running the iterative Newton-Raphson procedure, which yields $\bar{\boldsymbol{b}}$. The model's equation can be interpreted by taking the exponential of the coefficients and observing that the odds of churn increase or decrease by $\exp(\bar{b_j})$ if the characteristic $\bar{b_j}$ is underlying. Each of the steps mentioned above is further explained in the case study Sections 4 and 6. In the next sections we start investigating the modern 'tree'-based family of machine learning models.

3.1.2 Decision Tree

The first 'tree'-based method considered in this study is the classic decision tree used for both classification and regression purposes. The approach consists of a sequential decision-making process in the form of a flow-chart with a series of test questions.

Decision trees can be used with various objectives. One of the primary goals of this method is the prediction of future records by using the tree-model derived from past data. The observations have to be correctly classified by testing requirements on a subset of the input variables. Similarly, in regression frameworks, the relationship to the continuous response of which the values are to be predicted is modelled by running a series of criteria on the features subset. A further vital use of this method is for variable selection. Like the selection procedures introduced in logistic regression, decision tree serves as a technique to select the most relevant variables for tree-based methods such as random forest and gradient boosting. This technique is not only limited to a deduction of a variable selection but also to establish relative variable importance. The variables' rankings are based on an enhanced model accuracy or reduced inhomogeneity of nodes (see below for further elaboration). Hence, one can determine how important a variable is for the model by looking at the results of the decision tree. Last but not least, this approach can be used for data manipulation and handling of missing values. Since decision tree assists in describing, generalising and categorising a dataset, it is often used in data-mining tasks.

An initial introduction of some preliminaries from graph theory and tree-based definitions (see Diestel (2016), pp. 2-8) is necessary for a better model understanding:

- A **graph** is an ordered pair $G = (V, E)$ (i.e. the order is relevant $G \neq (E, V)$), where V is a non-empty set of nodes or vertices and E a set of edges, disjoint from V.
- A **directed graph** is a graph $G = (V, E)$ with a non-empty set of nodes V and a set of **directed** edges which connect unordered pairs of nodes.
- A **cycle** is a finite sequence of distinct edges connecting a series of nodes, with only the first and last edge repeated.
- An **acyclic graph** is a graph with no cycles.
- A **tree** is a directed acyclic graph (V, E), where two nodes are connected by exactly one unique path.
- A **root** node is the vertex at which there are no incoming edges but zero or more outgoing edges. A **rooted tree** is therefore a directed acyclic graph, where all edges are directed away from the root.

- A **decision tree** is a tree-structured model, where the model $f : \mathcal{X} \to \mathcal{Y}$ is defined by a rooted tree.
- A **parent** node is a node divided into two sub-nodes. An **internal** node is a node receiving exactly one incoming edge and two or more outgoing edges. A **leaf** or **terminal** node has only one incoming edge and no outgoing edges (see Figure 3.3).

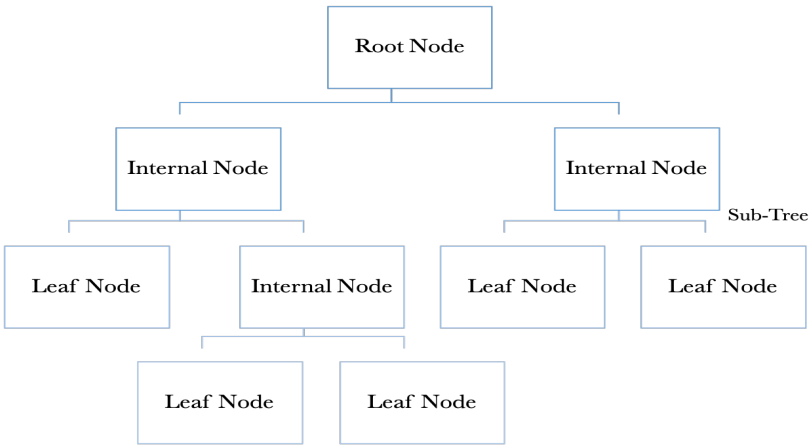


Figure 3.3: Decision Tree Structure

Inspired by Chauhan (2020).

One of the advantages of a decision tree is that there are very few theoretical assumptions the training data has to fulfil for the method to be applicable. There are no distributional, independence, or variance assumptions on the features. All input variables require a finite domain, however. The maximum dimensions of the decision tree are 2^M leaves and tree depth of $M + 1$ (see Shalev-Shwartz and Ben-David (2014), p. 251) if splitting w.r.t. an attribute is only allowed once. That is, the response is assumed to be of the discrete binary type in the classification setting and numeric in the regression setting. Nodes represent the different attributes $\boldsymbol{x_1}, \ldots, \boldsymbol{x_m}$ of the feature matrix $\mathbb{X}$ and edges the decisions based on these attributes. Several internal nodes in the tree can represent the same attribute if the algorithm specifications enable it.

Like any classification model, the tree has to be represented by a non-linear mapping function $f : \mathcal{X} \mapsto \mathcal{Y}$ of the input variables estimated by the conditional expectations on the different partitions of the feature space. This is possible despite the main difference that it can not be represented by applying a function of the family of affine function as in Section 3.1.1. The general categorising approach will be explained in the next section, along with its possible variants. Given an input space $\mathcal{X}$, the goal is to partition the space in a way that defines unique isolating areas, including the different labels (see Figure 3.4). The areas can be expressed in the following manner. Considering R_{p_k} to be a parent

region at step k, we define the two resulting regions from partitioning by splitting using the input variable $\boldsymbol{x_j}$ as

$$R_s = \{\boldsymbol{x_i} \mid x_{ij} \leq t \text{ where } \boldsymbol{x_i} \in R_{p_k}, t \in \mathbb{R}, j \in \{1, \ldots, M\}\}, \tag{3.11}$$

$$R_{s'} = \{\boldsymbol{x_i} \mid x_{ij} > t \text{ where } \boldsymbol{x_i} \in R_{p_k}, t \in \mathbb{R}, j \in \{1, \ldots, M\}\}. \tag{3.12}$$

The feature space is partitioned such that the union over all distinct, non-overlapping regions created by the definitions above is equal to the feature space. The enumeration of the regions is arbitrary.

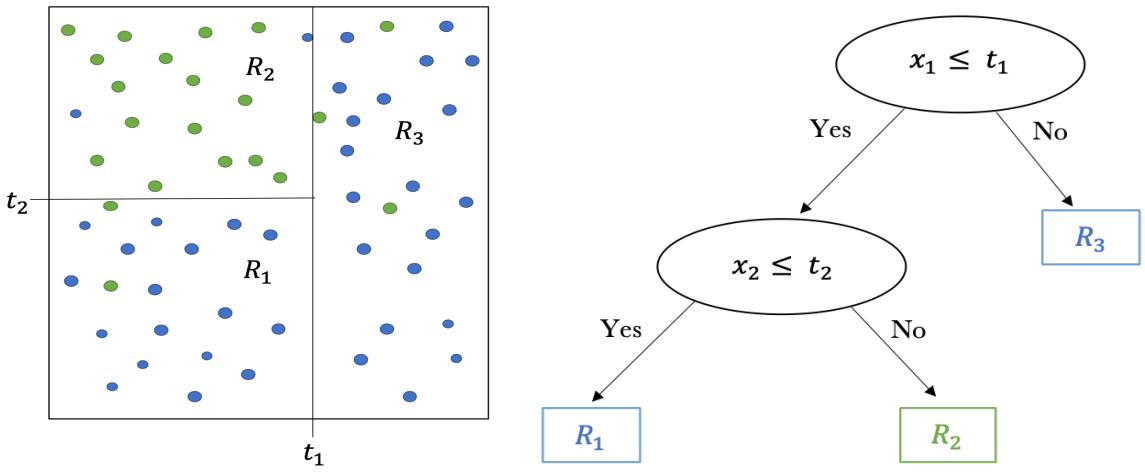
$$\mathcal{X} = \bigcup_{s=1}^{S} R_s \text{ where } S \in \mathbb{N} \text{ with}$$
$$R_s \cap R_r = \emptyset \quad \text{for } r \neq s.$$

We consider the following example where we have $M = 2$ features. In the root node, i.e. in step $k = 0$, we decide to split $\mathcal{X} = R_{p_0}$ using attribute $\boldsymbol{x_1}$ so we obtain the two regions:

$$R_s = \{\boldsymbol{x_i} \mid x_{i1} \leq t_1 \text{ where } \boldsymbol{x_i} \in R_{p_0}, t_1 \in \mathbb{R}, j = 1\},$$
$$R_{s'} = \{\boldsymbol{x_i} \mid x_{i1} > t_1 \text{ where } \boldsymbol{x_i} \in R_{p_0}, t_1 \in \mathbb{R}, j = 1\} =: R_3.$$

In step $k = 1$ we consider the parent region $R_s = R_{p_1}$ and split it such that we get the two regions:

$$R_s = \{x_{i2} \leq t_2 \text{ where } \boldsymbol{x_i} \in R_{p_1}, t_2 \in \mathbb{R}, j = 2\} =: R_1,$$
$$R_{s'} = \{x_{i2} > t_2 \text{ where } \boldsymbol{x_i} \in R_{p_1}, t_2 \in \mathbb{R}, j = 2\} =: R_2.$$



(a) Feature Space Partition. (b) Decisions Flowchart.

Figure 3.4: Example: Decision Tree Partition

Inspired by Hastie, Tibshirani, and Friedman (2001).

Figure 3.4(a) shows a partition of a two-dimensional feature space by recursive binary splitting on fake data including two classes indicated by the colours green and blue. The space is stratified such that the individual regions contain as much as uniquely possible only the green or the blue class. It can however occur that some observations lie in regions with a different majority label.
Using these definitions, we can express our predictor for regression trees as the sum over all mean responses of sample observations assigned to the individual regions (see Murphy (2012), p. 544):

$$\begin{aligned} & f : \mathcal{X} \to \mathcal{Y}, \\ & f(\boldsymbol{x}) = E[y_i \mid x_i \in R_s] = \sum_{s=1}^{S} c_s \mathbb{1}_{\{\boldsymbol{x} \in R_s\}}, \ \boldsymbol{x} \in \mathbb{X}, \ S \in \mathbb{N}, \\ & \widehat{y} = \hat{c}_s = \frac{1}{|R_s|} \sum_{i=1}^{N} y_i \mathbb{1}_{\{x_i \in R_s\}}, \end{aligned} \tag{3.13}$$

where $|R_s|$ is the number of observations that fall into region R_s and S the total number of regions. By travelling on a path from the root node and creating regions with each split, we reach the terminal node to predict the label, for instance $\boldsymbol{x}$, given that the observation lies in region R_s. The model involves stratifying the outstanding predictor space $\mathcal{X}$ into a number of distinct regions R_s that are branch-like with the constant c_s representing the response of the region in the case of regression trees. In the case of continuous features, the definitions (3.11) and (3.12) of R_s and $R_{s'}$ are reasonable. However, if we are considering a categorical variable, the definitions would be adjusted to the following splitting rules:

$$\begin{aligned} R_s &= \{\boldsymbol{x_i} \mid x_{ij} = c \text{ where } \ \boldsymbol{x_i} \in R_{p_k}, c \in \mathcal{C}', j \in \{1, \dots, M\}\}, \\ R_{s'} &= \{\boldsymbol{x_i} \mid x_{ij} \neq c \text{ where } \ \boldsymbol{x_i} \in R_{p_k}, c \in \mathcal{C}', j \in \{1, \dots, M\}\}. \end{aligned}$$

where $\mathcal{C}'$ represents the C' classes of the respective categorical feature. In comparison to statement (3.13), classification trees can not use the mean response as a representation for each region, so the most commonly occurring class in the region is predicted instead, which also leads to differences in the error measures.
We define the proportion p_{mc} and the respective predictor in the classification setting as

$$p_{sc} = \frac{1}{|R_s|} \sum_{\boldsymbol{x_i} \in R_s} \mathbb{1}_{\{y_i = c\}} \ \text{ s.t. } \sum_{c=1}^{C} p_{sc} = 1, \tag{3.14}$$

$$\widehat{y} = f(\boldsymbol{x}) = \arg\max_{c \in \mathcal{C}} \left(\frac{1}{|R_s|} \sum_{\boldsymbol{x} \in R_s} \mathbb{1}_{\{y_i = c\}} \right) = \arg\max_{c \in \mathcal{C}} p_{sc}, \tag{3.15}$$

$$p_{s\widehat{y}} = \frac{1}{|R_s|} \sum_{\boldsymbol{x_i} \in R_s} \mathbb{1}_{\{y_i = \widehat{y}\}},$$

with R_s being the region that contains $\boldsymbol{x_i}$ (see Hastie, Tibshirani, and Friedman (2001), pp. 308-310). While p_{sc} (see Equation (3.14)) represents the proportion of class c observations that fall in node s, representing region R_s, $\widehat{y}$ (see Equation (3.15)) stands for the most probable class label or majority class in node s.

The decision-tree-space complexity encountered to obtain the above defined final predictors raises the main question of how to grow the tree optimally. Most developed algorithms apply a greedy approach which optimises the tree only locally with respect to the features. The general model building process firstly selects the best attribute for splitting by using some of the popular measures introduced below. The attribute is then set as a decision node splitting the instance space. This process is repeated recursively for each subsequent node until a stopping criterion is met.

Univariate Splitting Criteria

Apart from the importance of choosing a proper growing and pruning algorithm, assessing the classification cost, which measures the quality of each split within the algorithms, is very crucial. It is important to note that the splits are done with the objective of optimising the following measures on the respective region R_s, e.g. comparing the purity of the parent region before and after splitting w.r.t. a particular attribute. The most common error measures or univariate splitting criteria for classification settings (see Hastie, Tibshirani, and Friedman (2001), pp. 308-310) are:

- Misclassification Rate:

$$\text{MR} = \frac{1}{|R_s|} \sum_{\boldsymbol{x_i} \in R_s} \mathbb{1}_{\{y_i \neq \widehat{y}\}} = 1 - p_{s\widehat{y}}.$$

  This error corresponds to the number of misclassified cases if we assign the most common class to be the label of this region. So the error represents the fraction of observations that do not belong to the most probable class. It was shown that this rate is generally not preferred due to the lack of sensitivity for tree-growth (see Gareth et al. (2013), pp. 311-314).

- Gini Index:

$$\text{Gini} = \sum_{c=1}^{C} p_{sc}(1 - p_{sc}) \overset{(3.14)}{=} 1 - \sum_{c=1}^{C} {p_{sc}}^2.$$

  The Gini index provides a measure of total variance across the different classes. It measures the node's 'purity' in the following sense. If the values of the proportions p_{sc} are close to zero or one and hence the Gini Index value is small,

it indicates that a node is 'pure' as it contains mainly instances from one class (see Gareth et al. (2013), pp. 311-314).

- Entropy or Deviance:
$$\mathbb{H}(\mathcal{D}) = -\sum_{c=1}^{C} p_{sc} \log p_{sc}. \tag{3.16}$$
Similar to the Gini Index, the value of Entropy is smaller the purer the s-th node and is very often used as a splitting criterion due to its enhanced sensitivity for tree-growth.

- Information Gain:
As defined in Bahety (2014), given the underlying training set $\mathcal{D}$ and $S_j(c) = \{\boldsymbol{x_i} \in \mathcal{D} | \, x_{ij} = c\}$ the set of instances $\boldsymbol{x_i}$ with attribute j taking the category c:
$$\mathrm{IG}(\mathcal{D}|j) = \mathbb{H}(\mathcal{D}) - \mathbb{H}(\mathcal{D}|\, j) \tag{3.17}$$
$$= \mathbb{H}(\mathcal{D}) - \sum_{c=1}^{C'} \frac{|S_j(c)|}{|\mathcal{D}|} \cdot \mathbb{H}(S_j(c)). \tag{3.18}$$
Information Gain represents the change in Entropy for a change of input variable. It also represents the reduction in the cost generated by going from a parent node to its successor as it measures the label's Entropy before and after the split (see Shalev-Shwartz and Ben-David (2014), p. 254). A high or maximal information gain implies that the feature optimally partitions the space.

- Gain Ratio:
$$\text{Gain Ratio} = \frac{\text{Information Gain}}{\text{Split Info}}, \tag{3.19}$$
$$\text{Split Info} = -\sum_{c=1}^{C'} \frac{|S_j(c)|}{|\mathcal{D}|} \log\left(\frac{|S_j(c)|}{|\mathcal{D}|}\right). \tag{3.20}$$
The Gain Ratio was introduced to measure the information generated by a split that is valuable for prediction (see Frank (2003)).

The numerous aspects to be considered when growing a tree are, for instance the outline of a specific tree structure, the choice of feature used as a splitting criterion at each node, as well as the specification of splitting criterion and the threshold parameter at which the tree stops to grow.
These dimensions are considered in the variety of possible algorithms to create decision trees. The contrast between the algorithms lies mainly in four aspects:

- What is my splitting criterion? How is the error or variability measured?

- Can the algorithm be used for regression and/or classification purposes?
- Does the algorithm apply a technique to avoid overfitting?
- Does the algorithm handle missing values? How?

Decision Tree Algorithms

One of the first decision tree algorithms was developed in the late 1980s by J.R. Quinlan and is called ID3 (Iterative Dichotomiser 3). In this approach, all possible decision trees are built, and the simplest one out of all is chosen (see Quinlan (1986)). To build the decision tree, that only supports categorical variables, minimum Entropy and maximum Information Gain are considered as test attributes of the respective node. The procedure works as follows (see Bahety (2014)):

1. Using the original training dataset $\mathcal{D}$ at the root node, recursively calculate the Entropy and Information Gain only of unused features at each iteration.
2. Check if any of the following stopping criteria are reached. If not, go to step 3.
   (a) If all observations are labelled by y_s then s is a terminal node or leaf labelled by y_s.
   (b) If there are no more unused features, this node is considered a terminal node and labelled by the majority occurring class of the prior node.
   (c) Similarly, if there are no more observations in the resulting node, the node is labelled by the majority class of the prior node.
3. Start the procedure of attribute selection. From the subset of unused attributes, select the attribute with the smallest Entropy or largest Information Gain to partition the current space.
   (a) Choose the best classifying feature and set it as an internal node.
   (b) Add two corresponding branches. One representing all observations where the feature dependent condition $x_s = c$ is fulfilled and one with $x_s \neq c$.
   (c) Go to step 2 for each resulting node.

This procedure is repeated for all subsequent nodes, does not guarantee to calculate the optimal solution, and can lead to overfitting. The algorithm also does not include a pruning method to avoid possible overfitting, nor the handling of missing values.

An extension of ID3 to support numerical variables and missing values was later introduced by Quinlan (1987), namely the C4.5 algorithm. This approach has become a benchmark for newer algorithms. The main difference to ID3 is the splitting criterion, since C4.5 uses the Gain Ratio as a measure of the chosen feature for decision making. A further difference lies in the possibility of applying a bottom-up pruning technique on the tree to avoid overfitting.

Pruning Algorithms

Most pruning procedures grow a large tree and apply a bottom-up approach to eliminate the nodes that do not provide additional information.
In the extension (see Quinlan (1987)), reduced error pruning (REP) and complex error pruning (CEP) were introduced. REP starts from the leaves up and changes the label of each internal node, i.e. (non-leaf) subtree with the most probable class if it does not affect the prediction accuracy. If there is no subtree with the same property and this adjustment reduces or maintains the error level, then the subtree is replaced by the leaf. CEP, in contrast, builds several trees starting from the original unpruned tree. Given that T_k is the unpruned tree and T_0 the rooted tree, firstly trees T_{i+1} are created by replacing subtrees in T_i with leaves if the error measure is small enough (see Rokach and Maimon (2015)). All created trees are then evaluated with respect to their generalisation error on the cross-validation sets. The best-pruned tree with minimal generalisation error is chosen.
A similar yet more complex pruning algorithm used in C4.5 is the error based pruning (EBP). The algorithm checks all nodes bottom-up, compares the three errors below and chooses the alternative with the smallest error (see Rokach and Maimon (2015)):

1. $E(\text{subtree}(T,t), S_t)$
2. $E(\text{pruned}(\text{subtree}(T,t),t), S_t)$
3. $E(\text{subtree}(T, \text{maxsuccesor}(T,t)), S_{\text{maxsuccesor}(T,t)})$ where

$$E(T,S) = MR(T,S) + Z_\alpha \sqrt{\frac{MR(T,S)(1-MR(T,S))}{|S|}}.$$

One further well-known approach is the CART Algorithm, which works for both classification and regression problems. It uses the Gini Index as a measure for splitting. The recursive binary splitting approach is applied from the root node to the terminal node, where a stopping criterion is reached. The tree is not forward-looking, i.e. it does not look for the optimum split that minimises further error measure at nodes later on but

rather locally minimises the error measure. The algorithm, however, includes a pruning phase after growing, which uses CEP.

One of the most notable advantages of decision tree is its interpretability, as the structure is not only intuitive but also easily tractable. The procedure additionally handles different data types such as categorical and numerical, whether ordered or unordered. Last but not least, a feature selection within the method provides efficiency in the prediction procedure. The model, on the other hand, is unstable with respect to small changes in data, i.e. any small change can lead to a substantial change in the structure of an optimal tree (see Murphy (2012), p. 550). The decision tree is also relatively inaccurate in comparison to other predictors such as a random forest. Information Gain is often biased for data with more categorical data levels, thus normalising the Gain Ratio.

Given a churn framework such as our case study we consider the feature space $\mathcal{X}$ as the space with M variables representing diverse useful information on the client such as previous churn behaviour, necessary general information such as age, marital status, regional information, and explanatory variables on the agent in charge. Another interesting perspective on the use of a decision tree for a churn-labelled dataset is the prediction of occurrence probabilities through regression trees. Not only can we classify instances, but we can predict their occurrence probability. We choose to apply classification trees in our setting where we consider a training dataset $\mathcal{D}$ with observations labelled by the binary churn target variable. If we have an underlying client instance along with M information variables, the goal is to be able to follow a specific decision path through the trained decision tree and classify whether the customer will churn or not. Additionally, we want to predict the probability indicating if the customer will churn or not using regression trees. We decide to apply the CART algorithm, since it includes both options of classification and regression. After training the tree on our dataset $\mathcal{D}$, we insert a previously unseen instance $\boldsymbol{x}$ into the tree, and receive the predicted churn behaviour $\hat{y}$ as a result by the respective assigned leaf node. Considering that the decision tree can potentially overfit, we investigate a tree-based resolution in the following section.

3.1.3 Random Forest

It was underlined in the previous section that decision tree suffers from high variance, tends to overfit the data, and leads to an accuracy drawback when predicting out-of-sample observations. It was also shown how 'greedy' decision tree algorithms can be. The motivation to tackle these matters gave rise to an extended and enhanced use of decision trees. The method resulting from trying to avoid these obstacles is the random forest technique. Random forest can be used for both classification and regression problems.

The essential idea of random forest is the construction of an ensemble of decision trees and receiving a prediction by averaging the predictions of the single trees involved. An ensemble should include several trees constructed by an algorithm, and using its individual predictions should result in the final prediction.

Random forest modelling requires little or no assumptions. Since it is based on the decision tree algorithms, it mainly requires the finiteness of the domains of features spaces for the construction of the individual trees. The model also works on small to medium-sized datasets and does not necessarily need large numbers of observations. The observations are logically required to be i.i.d.

We consider the labelled training dataset

$$\mathcal{D} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_N}, y_N)\},$$

with the feature matrix $\mathbb{X}$ and the binary classification set $\mathcal{Y} = \mathcal{C} = \{1, 2\}$ as the labels' domain in classification settings and $\mathcal{Y} = \mathbb{R}$ that of regression settings. The goal is to model the association between features and response through a function composition of $f : \mathcal{X} \mapsto \mathcal{Y}$ and decision tree classifying maps. Hence, we represent random forest by the following definition:

A random forest is a classifier f depending on a family of decision trees $\{T_b\}_{b=1}^{B}$ with B different bootstrap samples and i.i.d. parameters randomly chosen from a subset of the model parameters set. By performing a randomised variant of the decision tree algorithms, a forest of decision trees is built.

Bagging

In order to reduce possible variance or overfitting, specific methods were introduced. These techniques mainly create an enhanced composite of several predictors into one estimator. We introduce bagging (see Hastie, Tibshirani, and Friedman (2001), pp. 282-283), as a motivation for the random forest model:

1. Repeatedly draw B different bootstrap samples from training set $\mathcal{D}$.
2. Train the method on each of the B samples to obtain the predictors $f_b(x) = T_b$ for $b = 1, \ldots, B$.
3. All predictors are combined by averaging (in regression frameworks) or majority voting (in classification frameworks) and the labels are assigned accordingly.

$$f_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b \quad \text{in the regression framework.}$$

Computing M different trees on different subsets through bagging, however, can lead to highly correlated predictors. This calls for a further random element such as random feature selection. That is, not only the subset of observations should be randomly chosen but also the subset of input variables considered for splitting within the decision tree algorithm.

Random Forest Training

Based on extended bagging, the general random forest approach can be represented by the following algorithm (see Hastie, Tibshirani, and Friedman (2001), pp. 587-589):
For $b = 1$ to B:

1. Draw a bootstrap sample S with size N from the training dataset.
2. Considering independent decision trees on this sample, create a random forest by recursively repeating the following steps to the leaves until a minimum node size is reached.
   (a) Select a random subset of features with size k from the M input variables, where $k \leq M$.
   (b) Select the best attribute for splitting among the k inputs.
   (c) Split the node into two successive nodes respectively.
3. Construct ensembles of un-pruned trees $\{T_b\}_{b=1}^{B}$ using the decision tree inducers introduced in Section 3.1.2.

To calculate the final estimate, we distinguish between regression and classification purposes.
In classification, a majority vote is used, i.e. if c_b is the predicted class of the b^{th} random forest tree then $\widehat{c} = \arg\max_{c_b} \sum_{b=1}^{B} p_b \mathbb{1}_{T_b = c_b}$ is the most probable class. The prediction of the random forest is equal to the majority vote $f(x) = \widehat{c}$.
In regression we average over all available trees such that the random forest map is defined as $f(x) = \frac{1}{B} \sum_{b=1}^{B} \widehat{y}_b$.

Random forest has been proven to be very robust to noise, accurate, and easy to understand. Not only does it solve the overfitting problem of decision trees, but it also works with high-dimensional data very well. Random forest also works in both classification and regression frameworks. On the other hand, random forest is computationally expensive and inefficient, since it requires a lot of steps and time to

calculate each involved tree's predictions. While decision trees are tractable, large random forests are not easily interpretable nor tractable.

As we have decided that the decision tree algorithm is reasonable for a churn modelling purpose, we consider random forest as an adequate technique complementing our classification goals. Random forest supports both classification and regression simultaneously, which delivers easy implementation. Knowing that the case study deals with very correlated features in the dataset, as there are only minor differences in the numerous measures, we consider the robustness to noise of random forest a suitable property. However, we also know that we can not deduct a rule generating mechanism or segmentation due to the complexity of random forest, and hence, we should consider more classic models such as logistic regression to be complementary to the random forest results. Running the above algorithm on a churn-labelled training set $\mathcal{D}$ should result in the required classifications. After creating the diverse decision trees, as explained in Section 3.1.2, the most frequent class is chosen as a label for the random forest's final prediction. Incorporating such a version of ensemble techniques in our use case can allow us to tackle challenges such as class imbalance, overfitting, and inefficient computation and is therefore applied and analysed in Chapter 5.
As random forest represents one type of ensemble learning methods and was proven to enhance predictive power, we consider a further variant of ensemble modelling in the next section.

3.1.4 Gradient Boosting

The most common approach to data-driven modelling is to build a single robust predictive model. An alternative approach introduced in the previous section is by using an ensemble technique that generates one powerful learner simultaneously combining several base weak learners and averaging the models in the ensemble. Motivated by the unknown effect of a stagewise adaptive procedure adding the learners iteratively and allowing consideration of the error obtained in previous models, the first boosting algorithms were developed.
As described by Freund and Shapire (1999), *"Boosting is a general method for improving the accuracy of any given learning algorithm"*. Inspired by Kearns and Valiant (1994), the first to question whether a weak learner can be boosted into one powerful learner, Freund and Shapire (1999) proposed the first boosting algorithm 'AdaBoost'. Later on, Breiman (1997) introduced an AdaBoost version, including gradient descent (explained in this section), with a particular loss function. To enhance flexibility and optimise user-specific cost functions, Friedman (2001) generalized Adaboost to gradient boosting to handle various loss functions.

Gradient boosting trains the models in a gradual, additive, and sequential manner granting it the power to improve predictive accuracy. The mechanism is relying on two aspects, the gradient descent and boosting techniques, using gradients of the loss function to identify the classifier's shortcomings. This section starts by identifying the required mathematical assumptions for the gradient boosting framework and motivation build-up, then working through the gradient descent method used for model training. It is important to note that we mainly consider the tree-based gradient boosting technique in this study.

Given M input variables and N data observations, we consider the feature matrix $\mathbb{X} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_N})^\top, \boldsymbol{x_i} \in \mathcal{X}$, and labelling vector $\boldsymbol{y} = (y_1, \ldots, y_N)^\top, y_i \in \mathcal{C}$ in the dataset, with the goal to minimize a given objective loss function $\mathcal{L}$ as follows (see Friedman (2001)):

$$\begin{aligned} \tilde{f}(\boldsymbol{x_i}) &= y_i, \quad \forall i \in \{1, \ldots, N\}. \\ \tilde{f}(\boldsymbol{x_i}) &= \underset{f(\boldsymbol{x_i})}{\arg\min}\, \mathcal{L}(y_i, f(\boldsymbol{x_i})), \quad \forall i \in \{1, \ldots, N\}. \end{aligned} \tag{3.21}$$

Equation (3.21) represents the general supervised learning objective to obtain the unknown functional association $\tilde{f}$ mapping input $\mathbb{X}$ to output $\boldsymbol{y}$ through an estimate $\hat{f}(\boldsymbol{x_i})$ such that the specified loss function $\mathcal{L}(y_i, f(\boldsymbol{x_i}))$ is minimized. In terms of expectations the above objective can be expressed as the minimization of the expected loss function value over the joint distribution of the sample values ($\mathbb{X}$, $\boldsymbol{y}$) (see Friedman (2001)):

$$\tilde{f}(\boldsymbol{x_i}) = \underset{f(\boldsymbol{x_i})}{\arg\min}\, \mathbb{E}_{\mathbb{X}}[\mathbb{E}_y(\mathcal{L}(y_i, f(\boldsymbol{x_i})))|\boldsymbol{x_i}], \quad \forall i \in 1, \ldots, N.$$

Due to the unknown joint distribution $\mathcal{V}$, the finite training sample, with N' training observations generated by $\mathcal{V}$, is used instead in practice. Solving the approximated version

$$\hat{f}(\boldsymbol{x_i}) = \underset{f(\boldsymbol{x_i})}{\arg\min}\, \frac{1}{N'} \sum_{i=1}^{N'} \mathcal{L}(\boldsymbol{y_i}, f(\boldsymbol{x_i})),$$

enables the approximation of $\mathcal{V}$ via the training sample's empirical distribution.
The input variables can be of any form, such as categorical, discrete, or continuous since the underlying tree-based models do not postulate restrictions or assumptions about their distribution. Moreover, missing values are directly handled by the procedure and require no additional imputation. The labelling variable $\boldsymbol{y}$ can have any distribution, and the loss function is assumed to be a convex differentiable function of arbitrary choice. Common loss functions for the different target types are introduced in Table 3.1 (see Hastie, Tibshirani, and Friedman (2001), p. 360). This setup gives the gradient boosting model the advantage of being an efficient and flexible model.

Model	Loss Function $\mathcal{L}$	Gradient
Regression	$\frac{1}{2}[y_i - f(\boldsymbol{x_i})]^2$	$y_i - f(\boldsymbol{x_i})$
Regression	$\lvert y_i - f(\boldsymbol{x_i})\rvert$	$\text{sign}(y_i - f(\boldsymbol{x_i}))$
Classification (see (3.16))	$-\sum_{c\in\mathcal{C}} \mathbb{1}_{\{y_i=c\}} \log(p_c(\boldsymbol{x_i})$	$\mathbb{1}_{\{y_i=c\}} - p_c(\boldsymbol{x_i})$

Table 3.1: Gradient Boosting - Loss Functions

Additive Approach Motivation

Despite the similarity to the adaptive basis function models, gradient boosting's optimisation lies in a functional space w.r.t. the predictions rather than a parameter space. For parametrized models, the function search space can be restricted to the parameter search space using the family $f(\boldsymbol{x_i}, \theta)$ such that (see Friedman (2001)):

$$\tilde{\theta} = \arg\min_{\theta} \mathbb{E}_x[\mathbb{E}_y(\mathcal{L}(\boldsymbol{y_i}, f(\boldsymbol{x_i}, \theta)))|\boldsymbol{x_i}], \quad \forall i \in 1, \ldots, N,$$
$$\tilde{f}(\boldsymbol{x_i}) = f(\boldsymbol{x_i}, \tilde{\theta}).$$

Analogously to the parametric optimization problem, we consider the vector of functions $f(\boldsymbol{x_i})$ fixed at $\boldsymbol{x_i}$ $\forall i \in \lfloor N \rfloor$ to be the 'parameter' vector of interest. The primary difference in boosting lies in the manner of parametrizing f in the additive functional form rather than the θ additive parametric form to tackle the absence of a closed-form solution:

$$f(\boldsymbol{x_i}) = \sum_{n=0}^{K} f_n(\boldsymbol{x_i}),$$

with K representing the number of iterations, f_0 the initial guess and the individual f_n the so called incremental 'boosts'. By choosing base learners $h(\boldsymbol{x}, \theta)$ the ensemble estimates can be predicted in any boosting process by (Friedman (2001)):

$$f_n(\boldsymbol{x_i}, \rho, \theta) = \sum_{n=0}^{K} \rho h(\boldsymbol{x_i}, \theta_n).$$

Especially considered in this study is the case where each of the parametrized functions of inputs $h(\boldsymbol{x_i}, \theta_n)$ is an individual decision tree and the parameters are the splitting variables, split locations, and leaf nodes classifications.

A greedy forward stagewise additive approach of function incrementing with the base-learners can be defined in this framework to solve:

$$\min_{\{\rho_n, \theta_n\}_{n=1}^{K}} \sum_{i=1}^{N} \mathcal{L}(y_i, \sum_{n=1}^{K} \rho_n h(\boldsymbol{x_i}, \theta_n)), \tag{3.22}$$

with the optimal step-size ρ_n specified at each iteration as follows:

Algorithm: Forward Stagewise Additive Modelling (FSAM)
(see Hastie, Tibshirani, and Friedman (2001), pp. 341-342)

Initialisation: $f_0(\boldsymbol{x}) = 0$.
Recursion:
for $n = 1, \ldots, K$:
  Compute $(\rho_n, \theta_n) = \arg\min_{\rho,\theta} \sum_{i=1}^{N} \mathcal{L}(y_i, f_{n-1}(\boldsymbol{x_i}) + \rho h(\boldsymbol{x_i}, \theta))$.
  Update $f_n(\boldsymbol{x}) = f_{n-1}(\boldsymbol{x}) + \rho_n h(\boldsymbol{x}, \theta_n)$.
end
Result: $f_K(\boldsymbol{x_i})$.

FSAM considers the loss in total of interest, that is, the iteratively added learners and the respective coefficients are chosen such that the updated model as a whole exhibits minimum loss. Previously added parameters are not adjusted with each added optimal expansion basis function. The final obtained function expansion represents the solution to problem (3.22).
Considering a classification tree as a base learner:

$$T(\boldsymbol{x}; \Theta) = \sum_{s=1}^{S} \theta_s \mathbb{1}_{\{\boldsymbol{x} \in R_s\}}, \quad \Theta = \{R_s, \theta_s\}_{s=1}^{S},$$

with predictive rule $\boldsymbol{x} \in R_s \Rightarrow f(\boldsymbol{x}) = \theta_s$ and θ_s representing the modal class of observations in region R_s, one can implement FSAM to find the additive model (3.23) through solving (3.24) at each step n (see Hastie, Tibshirani, and Friedman (2001), pp. 356-357):

$$f_K(\boldsymbol{x}) = \sum_{n=1}^{K} T(\boldsymbol{x}; \Theta_n), \tag{3.23}$$

$$\widehat{\Theta}_n = \arg\min_{\theta_n} \sum_{i=1}^{N} \mathcal{L}(y_i, f_{n-1} + T(\boldsymbol{x_i}, \Theta_n)). \tag{3.24}$$

$$\widehat{\theta}_{sn} = \arg\min_{\theta_{sn}} \sum_{\boldsymbol{x_i} \in R_{sm}} \mathcal{L}(y_i, f_{n-1}(\boldsymbol{x_i}) + \theta_{sn}). \tag{3.25}$$

One approximation approach to solve (3.24) is by applying the commonly utilised steepest gradient descent for parameter estimation, which uses $f_n = -\rho_n g_n$. Considering the current gradient g_n of $\mathcal{L}(f)$ evaluated at $f = f_{(n-1)}$ such that the vector g_n's components are expressed as:

$$g_{in} = \left[\frac{\partial \mathcal{L}(y_i, f(\boldsymbol{x_i}))}{\partial f(\boldsymbol{x_i})}\right]_{f(x_i)=f_{n-1}(\boldsymbol{x_i})}, \ \forall\, i \ \in \ \{1, \ldots N\}, \tag{3.26}$$

the gradient descent technique can be described through the following algorithm:

Algorithm: Steepest Gradient Descent
(see Hastie, Tibshirani, and Friedman (2001), pp. 358-359)

Initialisation: $f_n = 0$.
Recursion:
for $n = 1$ to K
  Compute $\rho_n = \arg\min_\rho \mathcal{L}(f_{n-1} - \rho g_n)$.
  Update $f_n = f_{n-1} - \rho_n g_n$.
end
Result: $\hat{f} = \frac{1}{K}\sum_{n=1}^{K} f_n$.

The gradient descent is often used as a numerical minimization method. It is called the batch algorithm as it has to iterate through the entire dataset to compute the gradient before starting the following iteration, which can be computationally infeasible for large problems. The gradient descent algorithm initialises the function value at iteration $n = 0$ by setting it to an offset value such as $f_n = 0$, and with each iterative step, it updates the new loss function in the direction of the negative gradient of the current loss function. The negative sign of ρg_n applies the decrease in function value with respect to the highest rate of $\mathcal{L}$ increase and represents the 'line search' along that direction.

Gradient Boosting Training

To generalize the gradient descent approach to previously unseen data $f_K(\boldsymbol{x})$, least-square trees T_{cn} are constructed at each iteration for each class c with classified labels as close as possible to the negative gradient g_{icn} defined in the gradient boosting algorithm:

$$\Theta_{cn} = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{icn} - T_c(\boldsymbol{x_i}, \Theta))^2,$$

which is the main step of the gradient boosting algorithm, along with the computed constant in (3.25) (see Hastie, Tibshirani, and Friedman (2001), p. 359). Gradient boosting alternatively to FSAM considers the loss as an iterative numerical optimization problem. Each additional learner with its determined step size is an individual correction term to its previous model. Breaking down the gradient boosting algorithm, it is essential to note that the different model settings are reached by entering different loss functions. The algorithm here focuses on the binary classification case.

Algorithm: Gradient Boosting
(see Hastie, Tibshirani, and Friedman (2001), pp. 360-361)

Initialisation: $f_0(\boldsymbol{x}) = \arg\min_\theta \sum_{i=1}^{N} \mathcal{L}(y_i, \theta)$.
Recursion:
for $n = 1$ to K
  for $c = 1$ to 2
    for $i = 1$ to N
      Compute $g_{icn} = \left[\frac{\partial \mathcal{L}(y_i, f_1(\boldsymbol{x_i}), f_2(\boldsymbol{x_i}))}{\partial f_c(\boldsymbol{x_i})}\right]_{f_c(\boldsymbol{x_i}) = f_{c,n-1}(\boldsymbol{x_i})} = \mathbb{1}_{\{y_i = c\}} - p_c(\boldsymbol{x_i})$.
    end
    Fit a classification tree w.r.t. g_{cn} giving terminal regions R_{sn} for $s \in \{1, \ldots, S_n\}$.
    for $s = 1$ to S_n
      Compute $\theta_{scn} = \arg\min_\theta \sum_{\boldsymbol{x_i} \in R_{sn}} \mathcal{L}(y_i, f_{c,n-1}(\boldsymbol{x_i}) + \theta)$.
    end
    Update $f_{c,n}(\boldsymbol{x}) = f_{c,n-1}(\boldsymbol{x}) + \sum_{s=1}^{S_n} \theta_{scn} \mathbb{1}_{\{\boldsymbol{x} \in R_{sn}\}}$.
  end
end
Result: $\hat{f}_c(\boldsymbol{x}) = f_{c,N}(\boldsymbol{x})$ for c in $\{1, 2\}$.

The algorithm's first step initialises the model with the constant model representing a tree with one terminal node. The algorithm then recursively fits the base learner, i.e. a classification tree to the computed respective negative gradient vector. The optimal tree constant θ_{scn} is computed to update the estimate of the current iteration. For each class, we obtain the tree expansion $f_{c,N}(\boldsymbol{x})$ generating classifications or predicted probabilities.

One of gradient boosting's advantages is its high predictive accuracy. The technique offers high flexibility as it optimises on various loss functions and hyperparameters. As mentioned in the mathematical assumptions, gradient boosting models are computationally more efficient as they directly include missing values handling instead of an additional imputation. Drawbacks of this technique are, however also prevailing, such as an overfitting tendency caused by the continuous improvement to reduce all errors. The high model flexibility comes with a cost of sizeable hyperparameter search grids, increased time and memory exhaustion. In comparison to the decision tree method, for instance gradient boosting is less interpretable in nature, offering less trust for the user. As boosting methods have shown good performance in churn prediction frameworks, we consider their implementation on the study's use case. We study a further adaptive basis function model in the next section, however, with numerical optimisation in parameter space.

3.1.5 Neural Networks

With a wide range of applicability in many disciplines, neural networks, rooting back to the 1940s, have gained popularity and evolved into a broad family of approaches with frequent use across multiple areas. The network-based model, initially inspired by a brain's complex neural network structure, which originated from efforts to make information processing in biological systems mathematically representable (McCulloch and Pitts (1943); Rosenblatt (1961)), was shown to be able to carry out laborious computations. Over time, it was extended for statistical pattern recognition. Like any other supervised machine learning method, given a label vector and input variables, the network's central capability is to model a non-linear association by generating communication between the computation units so-called 'neurons' of the network. Commonly, classification mechanisms comprised of linear combinations of fixed basis functions are used. It is discovered that such models have useful analytical and computational properties but that the curse of dimensionality limits their practical applicability. To apply such models to significant scale problems, it is necessary to make the basis functions adaptable to the data on hand.

Neural networks possess this adaptability and can be described by several terms. Using the introduced terminology of graph-theory in Section 3.1.2, one can describe a neural network as a directed graph with its 'neurons' representing nodes and the links generating connections between the neurons representing the edges of the graph. One can also consider a network as a class of serial parametric non-linear functional transformations. There are three main types of neural networks, namely the feedforward neural network, generally referred to as multilayer perceptron, recurrent, and convolution neural network. Feedforward neural network processes input information by forward propagation and is usually applied to tabular structured data. Recurrent neural network, in comparison, contains a forward direction and additional recurrent connections in a looping manner, ensuring that sequential information patterns are captured through the inputs. This type is preferably used on time series, text, and audio data. Finally, convolutional neural network is mainly prevalent in image and video processing. Given that the considered use case in this research provides tabular data, the focus is mainly on feedforward neural networks in this section.

We start by defining the network's main mathematical framework and then introducing the different architecture and hyperparameter variants throughout the network training. First of all, it is essential to note that neural networks work for both classification and regression purposes. They are applicable to structured and unstructured data, i.e. for both supervised and unsupervised learning problems.

Considering the terminology of graph structures, one can define a feedforward neural

network as an acyclic directed graph $G = (V, E)$ with its directed edges linking the output from some neurons to the input of others (see Shalev-Shwartz and Ben-David (2014), p. 269). The acyclic property of the graph ensures that the outputs of the network are deterministic functions of the inputs.

Given M input variables and N data observations we consider the same feature matrix $\mathbb{X}$ introduced in Section 3.1.1, which can be expressed by vector $\boldsymbol{x} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_M}), \boldsymbol{x_j} \in \mathbb{R}^N$ to represent the inputs. Features considered can be of any type, however, different encoding can help improve performance. If the inputs' ranges are too big, a normalisation of the inputs is usually considered a best practice for training, as obtaining a mean close to 0 can lead to a quicker learning rate of the optimal parameters for each input node. Furthermore, the labelling vector $\boldsymbol{y}$

$$\boldsymbol{y} \in \{1, \ldots, C\}^N =: \mathcal{C}^N,$$

$$\boldsymbol{y} = (y_1, \ldots, y_N)^\top, y_i \in \mathcal{C},$$

is considered in the neural network framework with the goal of approximating a final non-linear mapping function $\sigma : \mathcal{X} \to \mathcal{C}$ such that the weighted sum of outputs of all final neurons are mapped to a corresponding predicted event label. The mapping function is called 'activation function' in the neural network case. This structure can be considered a natural extension of logistic regression as expressed by the following equation:

$$\widehat{y}_i = \sigma\left(\sum_{j=1}^{M} w_j o_j(\boldsymbol{x_i})\right) \ \forall i \in \lfloor N \rfloor, \text{ with } \widehat{\boldsymbol{y}} = (\widehat{y}_1, \ldots, \widehat{y}_N). \tag{3.27}$$

Neural Networks Architecture

We extend the general representation (3.27) by the following notations and definitions. Over the edges of the network's graph, the weight functions $w : E \mapsto \mathbb{R}$, indicating the influence an input will have on the output, can be defined for later estimation purposes. A neural network consists of one or more layers, where the neurons are organized in a stacked pattern. The set of neurons can be organized by the decomposition (see Shalev-Shwartz and Ben-David (2014), p. 269):

$$\begin{aligned} V &= \dot{\bigcup}_{l=0}^{L} V_l, \\ E &: V_{l-1} \mapsto V_l. \end{aligned} \tag{3.28}$$

The first layer, i.e. the input layer V_0, represents exactly the feature space $\mathcal{X}$ and respectively the output of the $M + 1$ neurons in the first layer is exactly $\boldsymbol{x_j}$, $\forall j \in M$ and a bias term accounting for output adjustments (see Figure 3.5). We denote by $v_{l,n}$

the n^{th} neuron of the l^{th} layer and n_l the number of total neurons in the l^{th} layer, thus $V_l = \{v_{l,1}, \ldots, v_{l,n_l}\}$. While each layer extracts features of the input for classification, the use of multiple hidden layers allows construction of hierarchical features at different levels of resolution. Each layer V_{l-1} is connected by edges E to layer V_l as expressed by the mapping (3.28) and as visible in Figure 3.5.

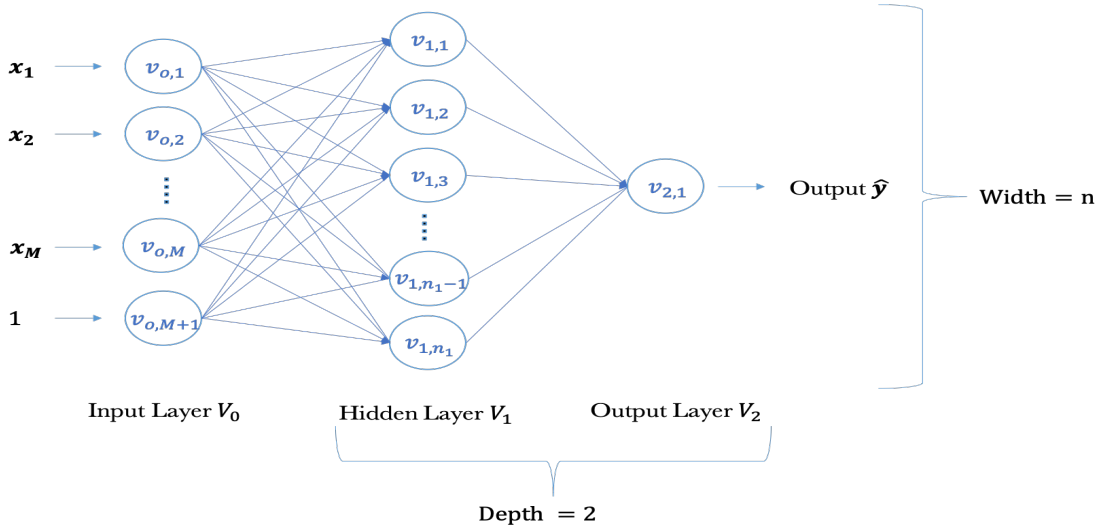


Figure 3.5: Feedforward Neural Network Architecture

Shalev-Shwartz and Ben-David (2014), (pp. 269-270) define the network's architecture (V, E, σ) by the following equations for a fixed neuron $v_{l+1,n} \in V_{l+1}$:

$$a_{l+1,n}(\boldsymbol{x}) = \sum_{r:(v_{t,r}, v_{t+1,n})} w((v_{t,r}, v_{t+1,n}))o_{t,r}(\boldsymbol{x}), \tag{3.29}$$

$$o_{l+1,n}(\boldsymbol{x}) = \sigma(a_{l+1,n}(\boldsymbol{x})). \tag{3.30}$$

The input into the n^{th} neuron in the l^{th} layer is denoted by $a_{l,n}(\boldsymbol{x})$ given the input vector $\boldsymbol{x}$ into the network. Similarly, the output of the n^{th} neuron of the l^{th} layer is denoted by $o_{l,n}(\boldsymbol{x})$. $a_{l+1,n}(\boldsymbol{x})$ defined by Equation (3.29), represents the weighted sum of output for all neurons connected between layer l and $l+1$. Each layer gets an activation function assigned by the user, which transforms the weighted sum into the neuron's output. The network's depth is equal to L, demonstrating the origination of the term 'deep learning', its width $\max_l |V_l|$ and its size equal to $|V|$ (see Shalev-Shwartz and Ben-David (2014), p. 270). Activation functions are defined to perform a certain mathematical operation. The commonly applied activation functions σ are:

- Sigmoid Function (see Figure 3.1):

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad x \in \mathbb{R}.$$

The non-linear sigmoid function returns a number between 0 and 1 for any given real value x, providing an easy interpretation. The choice of a sigmoid function for the activation correponds to the logistic regression model.

- Hyperbolic Tangent:

$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1, \quad x \in \mathbb{R}.$$

The tanh function reduces the real values x's range to the range -1 to 1 and centres the values around 0. The centring provides a comparative advantage to the sigmoid function.

- ReLU (Rectified Linear Unit):

$$\text{ReLU}(x) = \max(0, x), \quad x \in \mathbb{R}.$$

The piecewise linear rectified linear activation function is the function returning the provided real value x as long as $x > 0$ is fulfilled and otherwise returning the value 0. The limited sensitivity and saturation of the sigmoid and tanh function lead to challenges when adapting the network's weights. These are overcome by ReLU as it represents sparsity and preserves linearity.

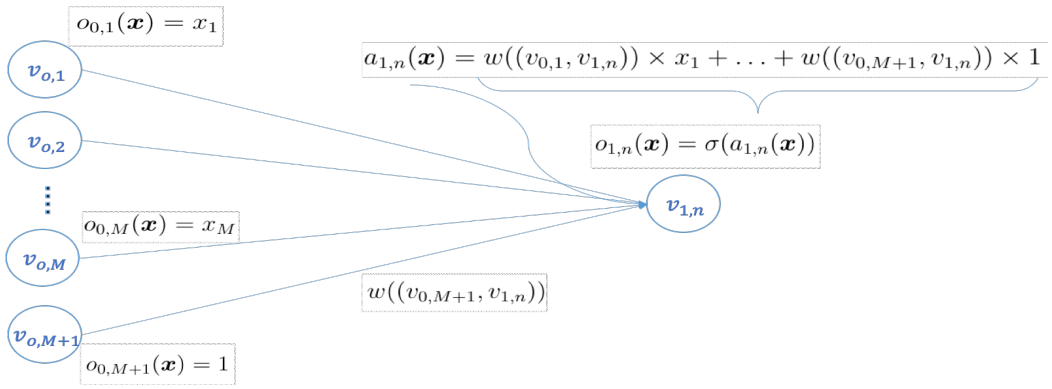


Figure 3.6: Neural Network Architecture Formulas

To provide an illustration to Equations (3.29) and (3.30), Figure 3.6 fixes the n^{th} neuron of the first hidden layer, i.e. $v_{1,n}$ and demonstrates the resulting neurons' input and outputs. Similar to the gradient boosting model, to estimate model parameters, loss functions are minimized in the training process. For classification, the deviance measure is often used as a loss function (see Table 3.1).

Neural Networks Training

Algorithm: Backpropagation (BackProp)
(see Shalev-Shwartz and Ben-David (2014), pp. 277-281)

Initialisation: $W_{l,r,s} = w((v_{l,r}, v_{l,s}))$, $W_{l,r,s} = 0$ for $(v_{l,r}, v_{l,s}) \notin E$,
$o_0 = \boldsymbol{x}$, $\delta_L = o_L - y$.
Recursion Forward:
for $l = 1$ to L
  for $n = 1$ to n_l
    Compute $a_{l,r} = \sum_{s=1}^{k_{l-1}} W_{l-1,r,s} o_{l-1,s}$.
    Compute $o_{l,r} = \sigma(a_{l,r})$.
  end
end
Recursion Backward:
for $l = L - 1$ to 1
  for $n = 1$ to n_l
    Compute $\delta_{l,r} = \sum_{s=1}^{k_{l-1}} W_{l,r,s} \delta_{l+1,s} \sigma'(a_{l+1,s})$.
  end
end
Result: $\forall (v_{l-1,s}, v_{l,r}) \in E$; partial derivative $v_k = \delta_{l,r} \sigma'(a_{l,r})) o_{l-1,s}$.

Algorithm: Stochastic Gradient Descent
(see Shalev-Shwartz and Ben-David (2014), pp. 277-281)

Initialisation: $w^{(1)} \in \mathbb{R}^{|E|}$ s.t. $w^{(1)}$ close to 0.
Recursion:
for $k = 1$ to K
  Compute gradient $v_k = \text{backpropagation}(\mathcal{D}, w, V, E, \sigma)$ for $\mathcal{D} = (x, y) \sim \mathcal{V}$.
  Update $w^{(k+1)} = w^{(k)} - \rho_n (v_k + \lambda w^{(k)})$.
end
Result: $\hat{w}$, the best performing weight vector $w^{(k)}$.

Neural networks training comprises of minimising the loss function w.r.t. the adaptive weights and bias, which is commonly performed by using the 'stochastic' gradient descent (SGD) as numeric parameter optimization method after determining the gradient via the 'backpropagation' method. The backpropagation algorithm calculates the gradient

w.r.t. all of the network's weights on a sample generated by the distribution $\mathcal{V}$ of the full dataset observations. The method includes two recursion directions as visible in the algorithm setup. The first recursion transfers the information in a forward manner, setting all neurons' inputs and outputs per the defined layer structure, sample, and activation functions. The second recursion starts at the last layer and works in a backward order to determine the gradient for each edge of the network.
Subsequently, the weight vector, optimising the loss function, is determined by using the SGD updated version of the gradient descent algorithm (see Section 3.1.4), which enhances the search of the global optimum for non-convex objective functions. To yield faster optimisation, the SGD also only iterates through subsets of the full batch by considering $\mathcal{D}$. The procedure initialises the weight vector's values at iteration $n = 1$ by setting $\boldsymbol{w}^{(1)}$ as a random vector close to zero, and with each iterative step, updates the new loss function in the direction of the negative gradient of the current loss function. The negative sign of ρg_n applies the decrease in function value with respect to the highest rate of $\mathcal{L}$ increase. It is also important to note that the variable step size choice is more significant in this context due to the non-convexity of the loss function. Sometimes, including a regularization parameter λ also known as 'weight decay', such that the objective becomes $\arg\min_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) + \frac{\lambda}{2} \|\boldsymbol{w}\|$, corresponding to the ridge regression constraint in Equation (3.37) (see Shalev-Shwartz and Ben-David (2014), p. 277). The larger the values for the penalty parameter, the more the network's weights will tend toward zero.

This developed and often used backpropagation algorithm was, however, proven to be a lengthy converging optimization technique and not well scalable on large networks, such that numerous other optimisation methods were developed. One of the developed families of such methods, Quasi-Newton, leads to more efficient network training. The Quasi-Newton family includes one of the alternative network training algorithms, namely the Quick-Propagation (QuickProp) method by Fahlman (1988). This local updating algorithm approximates the loss function using Taylor expansion and results in the following independent weight updates (see Fahlman (1988)):

$$\triangle W_{l,r,s}^{(k)} = \triangle W_{l,r,s}^{(k-1)} \frac{\triangle \mathcal{L}_{l,r,s}^{(k)}}{\triangle \mathcal{L}_{l,r,s}^{(k-1)} - \triangle \mathcal{L}_{l,r,s}^{(k)}}, \tag{3.31}$$

$$W_{l,r,s}^{(k)} = W_{l,r,s}^{(k)} + \triangle W_{l,r,s}^{(k)}. \tag{3.32}$$

The algorithm proposes a local approximation of the loss function and the computation only depends on the regarded neurons in the connected layers, thus QuickProp can speed up learning by only using the loss function curvature. Subsequent to retrieving the first gradient with backpropagation, the direct step (3.31) is performed to attempt jumping in one step from the particular point directly to the minimum of the parabola, assuming

that the loss function surface is locally quadratic. A further developed technique is the conjugate gradient, which builds a learning procedure combining gradient descent and Newton's method. As backpropagation does not ensure the fastest convergence, the conjugate gradient performs the search along with conjugate directions. We refer to (Johansson, Dowla, and Goodman (1991)) for a detailed explanation and derivation of the algorithm steps.

The advantages offered by neural networks includes the joint capability of modelling complex linear and non-linear relationships using flexible input types and delivering outputs of various forms. In comparison to standard regression models, neural networks are also less sensitive to noise. While neural networks exhibit many advantages, they are not exempt from drawbacks. First, specifying the optimal network architecture does not have an accompanying guideline but is instead achieved by experience, as well as trial and error, which makes it harder to implement for a given use case by the user. Specially added to the complexity of network optimisation, overfitting issues have to be taken into account. Furthermore, the lack of model interpretability reduces the trust of the user in the provided result. Not knowing why and exactly how the network predicts this outcome induces reduced flexibility in post-modelling extractable measures.

Considering the reliable power contained by neural networks, it can be useful to apply them in the churn prediction framework to detect existing relationships. As neural networks come in different architecture variants and optimisation possibilities, we consider them an interesting application on the case study, to observe the resulting performance w.r.t. the different settings. Before presenting the models implemented in the use case, we introduce the theory of pre-processing the underlying dataset in the next section.

3.2 Prediction Process

3.2.1 Data and Features Pre-Processing

Real-world data is most often imperfect in the sense that it can include improper data entries or errors, irrelevant data, missing values of attributes, or incompatible data formats. Hence, before any valuable knowledge extraction or data analysis process can start, there are standard data requirements that have to be fulfilled. Data pre-processing not only prepares the data for a more efficient analysis but can also be used with the objective of getting a grasp of the dataset and its nature. By doing this, the user can choose to reduce the levels of granularity or change the structure of the data as desired. In large enterprises of the insurance industry, for instance there are numerous insurance categories with diverse regulatory requirements and platforms. It is often a technical challenge to pre-process the data from such a heterogeneous data landscape. The different sources can have different degrees of complexity and quality, which can make integration into one flat-file database difficult.
Most of these challenges are solved through a form of ETL (Extract, Transform, Load) process, see (Gour et al. (2010)). The typical ETL procedure includes extracting the data from different sources into one data warehouse. Then, an appropriate transformation is applied by a series of rules. Lastly, the data is loaded into the end target most often as one flat-file database.
To enhance data reliability, three major categories of data problems are considered. One is often faced with either too little data, too much data or fractured data as described by Famili et al. (1997). First, data could be insufficient when a large number of the feature values are missing, general attributes are missing or when the dataset size is not large enough. Second, a great amount of data can also be disadvantageous if noisy or irrelevant data is encountered, or if there are too many diverse data types. Third, the data could be fractured if many data sources are integrated into one, with incompatible formats or diverse granularity levels.

Given a raw dataset with M input variables and N observations, the goal is to transform the data represented by $\mathbb{X}$ through functions T_i with $\mathbb{X}' = T_i(\mathbb{X})$ such that $\mathbb{X}'$ is more useful, does not exclude useful information and eliminates some of the above mentioned problems (see Famili et al. (1997)). $\mathbb{X}'$ would then contain $m < M$ features.

First, the data needs to be partitioned into a training set $\mathcal{D}$, validation set $\mathcal{V}$, and test set $\mathcal{T}$ if the given raw dataset is large enough. Smaller datasets are partitioned only into a training and a test dataset, and validation is performed on cross-validation samples from the training dataset to avoid overfitting, fine-tune hyperparameters, and get more

evaluation metrics for model evaluation. An overview of the method is provided here, and the validation approach is further introduced in Section 3.2.4.

Cross-Validation

Very widely used is the k-fold cross-validation technqiue to assess predictive performance. This procedure partitions the data into k equal parts with each a size of $\frac{N}{k}$; $k-1$ folds are used to train the model and the respective remaining k^{th} fold is used to test the model's predictive performance.

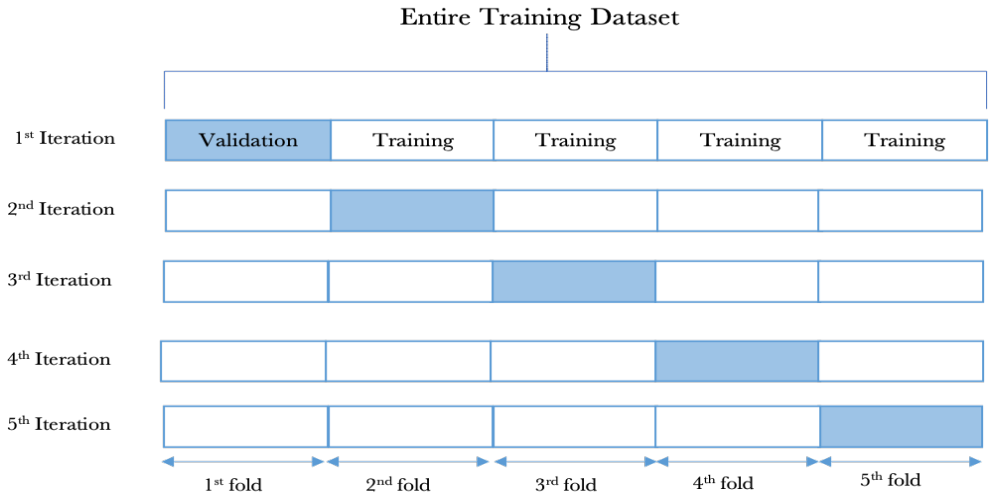


Figure 3.7: Cross-Validation Procedure

As described in Hastie, Tibshirani, and Friedman (2001) (pp. 241-243), after random partitioning, each observation in the dataset uniquely belongs to a certain fold which can be indexed by $\kappa : \{1, \ldots, N\} \mapsto \{1, \ldots, k\}$. Given each fitted function $\hat{f}^k$, determined by training the model on the training dataset without the k^{th} fold, one can define the cross-validation prediction error as the weighted average of all loss functions dependent on the respective function of the fitted models:

$$CV(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, \hat{f}^{\kappa(i)}(\boldsymbol{x_i})). \tag{3.33}$$

The case $k = N$, namely the leave-one-out cross-validation, is another possible variant where at each iteration, exactly one observation is left out for validation, and the rest of the data is used for model training. We consider the example of 5-fold cross-validation in Figure 3.7 as an illustration of the method. For each $k = 1, \ldots, 5$ fold, we iterate and create a model. In the first iteration, the model is fitted on folds $2, 3, 4, 5$ and then tested on the first fold. In the second iteration, the model is fitted on folds $1, 3, 4, 5$ and tested

on the second fold and so on. The evaluation of the model with respect to the CV error is further specified w.r.t. considered measures in Section 3.2.5.
To dive deeper into the possible appropriate data transformations, solutions for each of the pre-processing problem categories are introduced in the next paragraphs.

Data Transformation

When handling too much data, one encounters diverse data types and input variables with numerous levels or big ranges. It is usually the case that parametric predictive models, such as regression and neural networks, suffer from performance drawbacks if various categorical variables with many levels are included. Coding each categorical variable by indicator variables leads to a large number of model parameters. A common approach to tackle this issue is the combination or grouping of categorical input levels with similar outcomes (see Famili et al. (1997)). By consolidating the levels, the input variable's information would still be included yet without the need for encoding.

Often occurring are also data errors or outliers due to improper data entries, transmissions, or characteristics of the different systems from which the data is extracted. It is justifiable to remove outliers only if they do not represent actual natural variations but are rather errors, not reflecting reality. In that case, correct outlier detection is necessary to be able to remove or handle them. Standard solutions are data filtering or replacement.

Some datasets may have a large size with respect to the number of observations, while in other settings, the amount of information collected in the form of different attributes can be large. Nowadays, data is collected for all purposes, even if it does not assist in the currently required modelling purpose, since it could be useful information in the future. However, including too many features can increase model complexity and decrease performance. Hence, only the relevant features are chosen to reduce dimensionality and irrelevance. This approach is further explained in the upcoming Section 3.2.2.

Datasets with too little data, such as sets with attributes having missing values, can be tackled by variable imputation or complete elimination of the observations. On the one hand, elimination could reduce available useful information. On the other hand, unequal vector lengths could cause inaccurate evaluations or bias in the information value (see Famili et al. (1997)). Variable imputation is therefore preferred and refers to the substitution of missing values with a predefined replacement value. To reduce the largest amount of information loss possible, one can set a missing value threshold. If the threshold is set to $x\%$, variables with a missing value percentage exceeding $x\%$ are completely removed, and the rest is imputed with the pre-determined replacement values. However, the different variable types should be treated differently when imputing:

- Interval variables can be complemented with the mean, median, or mid-range of the variable values. Another alternative is a tree-based imputation (explained below).
- Class variables can either be complemented with the majority class occurring or a tree-based imputation as well.

The tree-based imputation relies on the introduced decision tree algorithm in Section 3.1.2 which automatically handles missing values within the modelling procedure. The method, as described by Rahman and Islam (2011), mainly divides the dataset into a dataset $\mathcal{D}_{\mathcal{F}}$ with observations having no missing values, i.e. full records and a dataset $\mathcal{D}_{\mathcal{M}}$ with observations having some missing records. A set of decision trees is built on $\mathcal{D}_{\mathcal{F}}$ using the features with missing values in $\mathcal{D}_{\mathcal{M}}$ and the C4.5 algorithm. Each missing record in $\mathcal{D}_{\mathcal{M}}$ is then inserted into the tree and is assigned the mean value (if numeric) or majority class (if categorical) of the respective leaf it falls into. We refer to (Rahman and Islam (2011)) for the detailed steps of imputation based on decision trees.

An improvement of generalisation and manageability is, in all ways, more profitable. However, a resulting limitation when imputing missing values is the omitting of model uncertainty by disregarding missing values in the fitted function. The imputed values are handled as real values, so this uncertainty is not accounted for in the model. Therefore, the distinction between structurally missing values and missing values with a 'real value' is important.

Especially in the churn setting where the features indicate deviating human behaviour we encounter a lot of missing values or features. The insurance sector offering a wide range of insurance categories has to process the respective data in distinct systems and define the possibility and paths of merging all sources. With the objective of reducing errors and enhancing the predictive power of the models, an extensive pre-processing is necessary to handle all of the above commonly occurring issues. Furthermore, a good understanding of the dataset can enable experts to determine whether the model is in line with the expectations of the churn's relevant drivers. This good grasp can enable marketing teams to use appropriate retention approaches.
Regarding our use case, it is important to consider partitioning the data in the right manner, using cross-validation, to avoid having small datasets and respectively smaller event frequencies. A proper variable imputation should also be performed as clients' data is often incomplete. Discretization of continuous variables or encoding categorical variables should be considered w.r.t. the model on hand. An implementation of these suggested data and features pre-processing methods is illustrated in Section 4.2 along with a necessary class imbalance reduction, introduced in the next Section.

3.2.2 Class Imbalance Reduction

In recent years, rare events modelling has become a critical part of the industry's essential business practices. Rare events are infrequent observations that might have a substantial impact on the company's profitability or general business position. Due to the low observed frequencies, datasets containing rare events are mostly imbalanced. An imbalanced dataset occurs when one target class is represented more than the others, i.e. when the class distribution is skewed. When the minority class, i.e. underrepresented class of interest, has a low frequency in the dataset, most predictive models' performance is degraded as it is not able to predict these rare events precisely. Most machine learning approaches assume that the data has an equal representation of the data; thus, several approaches were developed to help address imbalanced datasets and correctly handle them. In this section, the two common technique categories are introduced, namely data-level approaches and algorithm-level approaches (see He and Ma (2013)).

Data-level approaches

The data-level approaches include all data processing methods that are used for altering the number of observations for each class. One example is the widely used sampling technique, which generates a new dataset from the underlying set with respect to the needed class distributions or sample size. We discuss three possible variants in this section which include, random over- or under-sampling and SMOTE-NC (see below).

- Random Under-Sampling
  One of the classic sampling techniques that tackle the class imbalance problem is the random under-sampling approach. This consists of randomly removing majority class samples from the dataset. This approach might lead to a loss of information by reducing the available dataset size.

- Random Over-Sampling
  Random over-sampling addressees the imbalance by randomly duplicating minority class samples from the dataset for a random number of times. This approach might deliver biased results as it over-weighs the available rare events and be inefficient as it exhibits long computation time. Random over-sampling can be generated through the following steps (see Zhang and Chen (2019)):

  1. Set $y^{\text{new}} = y_{\text{MC}}$, where MC represents the minority class.
  2. Select $(\boldsymbol{x_i}, y_i)$ such that $y_i = y^{\text{new}}$ with probability $\frac{1}{N_{\text{MC}}}$.

3. Sample $\boldsymbol{x}^{\text{new}}$ from $\mathcal{V}_{Cov_{\text{MC}}}(., \boldsymbol{x_i})$ the probability distribution centered at $\boldsymbol{x_i}$ and covariance matrix Cov_{MC}.

The procedure first selects the minority class label and assigns this label to an instance-pair from the sample with probability $\frac{1}{N_{\text{MC}}}$, where N_{MC} represents the number of observations belonging to class MC. The last step is sampling the input values $\boldsymbol{x}^{\text{new}}$ from the given distribution.

- Synthetic Minority Over-sampling Technique - Nominal Continuous (SMOTE-NC): The SMOTE approach introduced, by Chawla et al. (2002), represents an advanced sampling method for over-sampling. As the name implies, the algorithm uses generated synthetic examples instead of randomly replicated observations. The basic SMOTE approach is, however, only applicable to continuous features. As the underlying use case dataset includes both nominal and continuous input variables, the SMOTE-NC version applicable for both types is considered. The algorithm by Chawla et al. (2002) takes smaller dataset samples of the minority class and generates synthetic samples along regions joining the k minority class nearest neighbours. Multiplying the difference between a feature vector in the sample and its nearest neighbour by a random number between 0 and 1 creates a random point along the line segment, providing more general examples. The algorithm can be described as follows:

  1. Penalize the difference of nominal features by computing the median of standard deviations of all the minority class's continuous features. This median should be used in the following algorithm steps to compute the euclidean distance of a nominal feature between a sample and the nearest neighbour.
  2. Compute the Euclidean distance between the considered feature vector and other feature vectors of the minority class sample, computing the nearest neighbour. The median computed should be included in the Euclidean distance for the nominal features deviating between the considered vector and its potential nearest-neighbour.
  3. Generate the synthetic sample using the SMOTE procedure described by the following pseudocode. While the values of the continuous features are deducted from the pseudocode steps, the nominal feature values are set by choosing the most occurring value of the k-nearest neighbours.

The following Pseudocode represents SMOTE for T number of minority class samples, SMOTE over-sampling percentage $N\%$, and number of nearest neighbours k:

Algorithm: k-Nearest-Neighbour
(see Shalev-Shwartz and Ben-David (2014), pp. 258-259)

Initialisation: Create sample $\mathcal{D} = \{(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_n}, y_n)\}$.
Recursion:
for $\boldsymbol{x} \in \mathcal{X}$
  Assign $\boldsymbol{x}$ the majority label in $\{y_{\pi_i} : i \leq k\}$, with $\pi_1, \ldots, \pi_N$ representing the reordering of instances $\{1, \ldots, N\}$ w.r.t. to their proximity to $\boldsymbol{x}$.
end
Result: $\mathbb{K} = (\boldsymbol{x}, y_{\pi_i})$.

Algorithm: SMOTE (T, N, k)
(see Chawla et al. (2002))

Initialisation: $N = \text{int} \cdot \frac{N}{100}$, $s = 0$.
Recursion:
If $N < 100$:
  Randomise T minority class samples :
  $T = \frac{N}{100} \cdot T$.
  $N = 100$.
end
for $i = 1, \ldots, T$:
  Compute k nearest neighbours for each minority class sample using $\mathbb{K}$.
  Generate synthetic samples:
  for $j = 1, \ldots, M$:
    if $N \neq 0$:
      Compute $D = \mathbb{MC}_{\mathbb{K}(nn)}(j) - \mathbb{MC}_i(j)$, for $nn \in \{1, \ldots, k\}$.
      Compute $\mathbb{S}_s(j) = \mathbb{MC}_i(j) + Gap \cdot D$ for Gap = random.uniform$(0, 1)$.
      $s = s + 1$, $N = N - 1$.
    end
  end
end
Result: $\frac{N}{100} \cdot T$ synthetic minority class samples represented by $\mathbb{S}$.

The SMOTE methodology, described by the above algorithm first randomly selects a proportion of the minority class. For each $\boldsymbol{x_i}$ in the sample the k nearest neighbours inside the subset are computed with the suggested algorithm using Euclidean distance. To generate each variable value of the synthetic observation the method progresses by

choosing a random instance nn from the k nearest neighbours computed. Subsequently, the difference D between instance nn and i for the j^{th} variable in matrix $\mathbb{MC}$, representing the original minority class sample, is determined. The computed difference multiplied by a random number between 0 and 1 is added to the original instance i, generating a synthetic value for variable j, which is stored in matrix $\mathbb{S}$, representing the synthetic samples. This is repeated for all variables on the minority class subset. We should note that s, initialised at 0, counts the number of synthetic samples generated. SMOTE overcomes the overfitting issue occurring in random over-sampling applications through the generation of artificial samples. Simultaneously, the procedure does not lead to a loss of information, in contrast to random under-sampling. However, SMOTE suffers from the possibility of additional noise created by neighbouring instances not representing the desired class. A further disadvantage is its impracticality for high-dimensional data, due to increased iterations accompanied by high computational complexity.

Algorithm-level approaches

- Cost-Sensitive Learning:
  Cost-sensitive learning assigns a misclassification cost for wrongly classified instances instead of directly trying to classify the observations. That is, instead of each instance being either correctly or incorrectly classified by the algorithm, each class (or instance) is assigned a misclassification cost. Thus, instead of trying to optimize the accuracy, the problem is then to minimize the total misclassification cost.

- Ensemble Modelling:
  One further popular approach to tackle class imbalance effects is to implement ensemble modelling. Whether bagging, boosting or stacking, the ensemble algorithms have showed enhanced performance on class-imbalanced datasets (see Burez and Van den Poel (2009)).

Taking into account that the underlying case study aims at predicting churn, class imbalance reduction techniques need to be applied. Churn can be viewed as a rare event in most industries, as it represents a small proportion of activity in the considered interval. For the underlying use case, it is also important to note that stratified sampling, stratified partition, and stratified cross-validation should be applied. Using the random variants of these approaches can lead to an unequal distribution of the available rare observations, which makes a withhold of an equal frequency necessary. These settings adjustments are defined on the corresponding spots in the following case study chapters.

3.2.3 Dimensionality Reduction

Along with the frequently faced pre-processing challenges such as nonstandard data structures or diverse data types, one often faces 'the curse of dimensionality' (Bellman (1961)) during the modelling cycle. High-dimensional feature spaces could provide useful features associated with the target, which improve the model fit. However, these spaces also require more data and add noisy non-informative features that are irrelevant to the classification, thus degrading the model's performance. The selection of appropriate input variables and the extraction of valuable information reduces a possible deterioration of potentially useful models. High dimensions of input spaces can lead to possible overfitting, since the number of configurations covered by an individual observation decreases. The computational and run-time cost of complex models can also be very high. Fast model implementation and easy interpretability can, therefore, deliver more desirable results.

Given a dataset with design matrix $\mathbb{X}$ representing M input variables, the goal is to find a matrix $\mathbb{X}'$ with $m < M$ features and better contribution to the model's predictive power.

In this section, different feature selection approaches that enable the construction of efficient and informative feature sets are addressed. Three main method categories are introduced, namely filter-based methods, wrapper methods, and embedded methods.

Filter based methods

An easily applicable approach is the selection of features based on their particular relevance or association with the response variable proven through statistical tests. That is, input variables are evaluated according to a pre-determined quality metric independent of the other inputs. The entire selection is independent of any modelling procedure as well and hence considered as a pre-modelling step (see Figure 3.8). The features can be filtered from the entire input space by ranking the metric values and setting a cutoff value. However, the techniques only consider the relationship of each feature with the target, but no interaction between the features, e.g. multicollinearity.

- A classic filter based method is the Chi-Square based selection. The Chi-Square test was initially created for independence and goodness-of-fit testing. In the framework of feature selection one can test the independence of a feature and class occurrence.

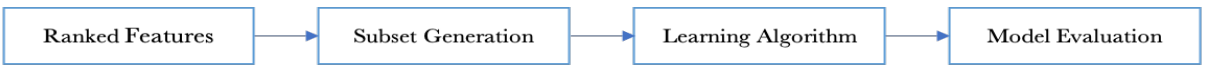


Figure 3.8: Filter Methods Flowchart

The method is applicable on both categorical and numeric interval variables. It requires, however a discretisation of the interval variables by splitting the range into equal-sized categories. In the binary case, the Chi-Square statistic is then defined as (Wu and Flach (2002)):

$$\chi^2 = \sum_{i=1}^{r} \frac{(f_{i0} - \mu_{i0})^2}{\mu_{i0}} + \frac{(f_{i1} - \mu_{i1})^2}{\mu_{i1}}. \tag{3.34}$$

f_{i*} represent the observed frequencies and μ_{i*} the expected frequencies w.r.t. each class deducted from the respective contingency Table 3.2, where $\mu_{ij} = \frac{f_{*j} f_{i*}}{f}$.

	$y_i = 0$	$y_i = 1$	$\sum$
Input category c'_1	$f_{10}(\mu_{10})$	$f_{11}(\mu_{11})$	f_{1*}
$\vdots$	$\vdots$	$\vdots$	$\vdots$
Input category c'_r	$f_{r0}(\mu_{r0})$	$f_{r1}(\mu_{r1})$	f_{r*}
$\sum$	f_{*0}	f_{*1}	f

Table 3.2: $r \times 2$ Contingency Table By Wu and Flach (2002)

The obtained χ^2-test statistic, see Equation (3.34), is compared against the critical value from the χ^2 distribution table with $(r - 1)$ degrees of freedom. If the corresponding p-value is smaller or equal than the pre-determined significance value, (i.e. the null hypothesis of independence has to be rejected), there is an association between the feature and response variable, and the feature should be considered an input. Independence would imply close observed and expected values, leading to a smaller Chi-Square statistic. One can conclude from the above formula that a higher χ^2 indicates a higher likelihood of correlation between the feature and event class. Hence, the top-ranked features are those with the largest Chi-Square values.

- The previously introduced information theory measures (Information Gain (3.17)) and (Gain Ratio (3.19)) have an additional usage compared to splitting measures for trees, namely as filter methods. In the context of the response variable, each variable's gain is evaluated as a measure of mutual information. A ranking provided through the information gain or gain ratio values provides the selected variables.

Wrapper methods

In regression-based classification models such as logistic regression, one can apply the well-known subset selection methods. Wrapper methods sequentially add or eliminate features from a subset of features by training the model on the subset and testing the

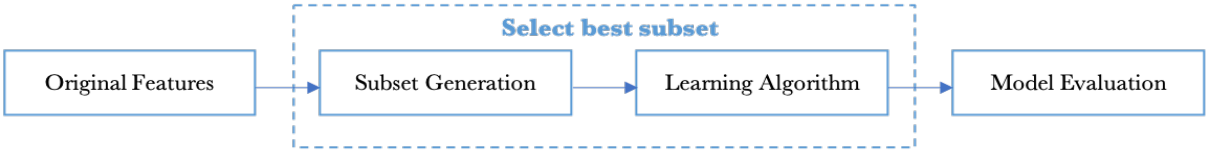


Figure 3.9: Wrapper Methods Flowchart

added predictive power to the classifier repeatedly (see Figure 3.9). Considering the 2^M possible models can be computationally infeasible and impractical for large M. The different versions of the subset selection search through smaller subsets from the entire models' space instead.

- **Forward selection** starts with the baseline model $\mathcal{M}_0$ containing only the intercept and no input variables, predicting the overall sample mean or majority class. Iteratively the feature with the best model fit improvement is added, creating a nested sequence of models with increasing complexity (see Gareth et al. (2013), pp. 207-210).

Algorithm: Forward Selection

Recursion:

for $j = 0, \ldots, M-1$:

> Construct all $M - j$ models supplementing $\mathcal{M}_j$ with one additional feature.
> Choose the model providing the best improvement among the
> $M - j$ models, i.e. the model with lowest p-value from the F-statistic and its respective alpha level, denote it by $\mathcal{M}_{j+1}$.

end

The loop is terminated if no significant improvement in the p-value can be made w.r.t. the pre-determined threshold.

Result: The best j models $\mathcal{M}_0, \ldots, \mathcal{M}_j$, $j \in \{0, \ldots, M\}$.

Among the at most $M+1$ candidate models $\mathcal{M}_0, \ldots, \mathcal{M}_j$, the model with the lowest AIC (3.35), highest BIC (3.36) or lowest cross-validation error (3.33) is chosen. Considering the above steps in the algorithm, one can conclude that only a maximum of $1 + \sum_{j=0}^{M-1}(M-j) \overset{\text{Gauss Formula}}{=} 1 + \frac{M(M+1)}{2}$ models are examined instead of the exhaustive construction of 2^M models.

The Akaike Information Criterion (AIC) defined as (see Hastie, Tibshirani, and Friedman (2001), p. 231):

$$AIC = -\frac{2}{N} \cdot \mathcal{L} + 2 \cdot \frac{K}{N}, \tag{3.35}$$

was raised to solve the problem of overfitting by taking into account the corresponding number of observations N, the number of model parameters K, and the log likelihood $\mathcal{L}$. The model with lowest AIC is considered to be best, as good fit is rewarded through the $-\frac{2}{N} \cdot \mathcal{L}$ term and over-parametrisation is penalised by the term $2\frac{K}{N}$, demonstrating the trade-off between fit and complexity.

The difference between the Bayesian Information Criterion BIC and AIC lies mainly in the penalty term. While AIC tends to choose more complex models to deliver a better model performance, the BIC is rather oriented towards simple models and penalises complexity (see Hastie, Tibshirani, and Friedman (2001), p. 233).

$$BIC = -2\mathcal{L} + \log(N)K. \tag{3.36}$$

It is important to note that both measures are relative, that is they can only be used for a model comparison and not as general goodness-of-fit or quality measures.

- **Backward selection** follows an opposite idea as forward selection, in which the approach starts with the full saturated model $\mathcal{M}_M$ containing all M predictors. At each step one predictor is removed and the respective models are considered (see Gareth et al. (2013), pp. 207-210).

Algorithm: Backward Selection

Recursion:

for $j = M, \ldots, 1$:

> Construct all j models each dropping one of the predictors from $\mathcal{M}_j$ for the $j-1$ predictors.
> Choose the model with least reduction in overall model fit among the j models, i.e. the model with lowest p-value from the F-statistic and its respective alpha level, denote it by $\mathcal{M}_{j-1}$.

end

The loop is terminated if the p-value of the remaining features is below a pre-determined threshold.

Result: The best j models $\mathcal{M}_0, \ldots, \mathcal{M}_j$, $j \in \{0, \ldots, M\}$.

From the at most $M+1$ models, the model with the highest AIC, lowest BIC, or lowest cross-validation error is chosen. Like forward selection, backward selection only examines $1 + \frac{M(M+1)}{2}$ models instead of 2^M. However, this selection can only be applied if the number of input variables M is larger than the number of observations N.

- **Stepwise selection** combines both aspects of the forward and backward selection approaches. The approach starts with the baseline model $\mathcal{M}_0$ and sequentially adds features improving the model fit. After each addition however a significance re-evaluation is performed as follows:

Algorithm: Stepwise Selection

Recursion:
for $j = 0, \ldots, M-1$:

> Construct all $M - j$ models supplementing $\mathcal{M}_j$ with one additional feature.
> Choose the model with the best additional improvement among the $M - j$ models.
> Re-evaluate the model w.r.t. the elimination threshold; if any of the current inputs exceed the F-statistic p-threshold, remove it and defined the model as $\mathcal{M}_{j+1}$.

end

The loop is terminated if no significant improvement in the p-value can be made w.r.t. the pre-determined remain or elimination threshold.
Result: The best j models $\mathcal{M}_0, \ldots, \mathcal{M}_j$, $j \in \{0, \ldots, M\}$.

It is important to note that stepwise selection usually delivers better results as it incorporates both aspects of the forward and backward selection, could however require longer computational time.

Embedded methods

Embedded methods represent techniques that have a built-in feature selection integrated within the model training procedure itself (see Figure 3.10). We first introduce two regularisation approaches for regression, which include shrinking the coefficient estimates towards zero. By setting some of the parameter estimates very close to zero, a variance reduction and simultaneous variable selection are performed. Then, decision tree variants are introduced, mostly used as a variable selection for the random forest and gradient boosting algorithms.

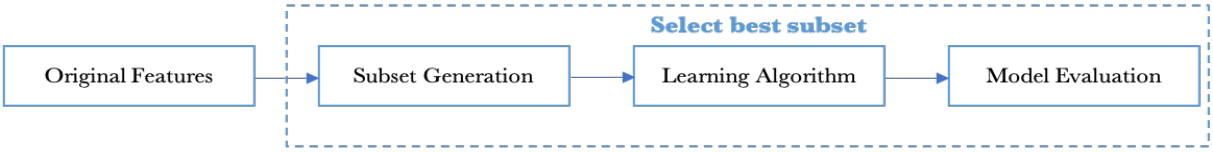


Figure 3.10: Embedded Methods Flowchart

Regularisation methods are used in cases with a large number of covariates and coefficient instability. By controlling how large the coefficients grow, a high potential variance can be controlled. The method consists mainly of shrinking the maximum likelihood estimates of model parameters towards zero. This is done through the addition of a penalty term to the respective loss function of the model, i.e.

$$\min_{b_0,\dots,b_M} [\mathcal{L}(\boldsymbol{x_i}, y, b) + \lambda P(b)].$$

- The **ridge regression model** in the logistic regression framework consists of an extra L_2-norm penalty parameter (sum of squared estimates, see Equation (3.37)) applied to all coefficients except the intercept. By choosing a complexity parameter $\lambda \geq 0$ the penalty is added to the function of the MLE. The ridge maximisation equation is thus expressed as follows (see Hastie, Tibshirani, and Friedman (2001), pp. 61-63):

$$l = \sum_{i=1}^{N} [y_i \langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle - \log(1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle))] - \lambda \sum_{j=1}^{M} {b_j}^2. \tag{3.37}$$

  As the shrinkage parameter λ and coefficients' magnitude increases, the coefficients tend to zero but are never completely eliminated. The drawback of this method is the inclusion of all M predictors in the model, which can lead to complex interpretability for large M. The above problem can also be expressed through the following constrained form (a proof is provided in Appendix A.1.2):

$$\min_{\bar{\boldsymbol{b}}} l = \min_{\bar{\boldsymbol{b}}} \sum_{i=1}^{N} [y_i (\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle) - \log(1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle))] \text{ s.t. } \sum_{j=1}^{M} {b_j}^2 \leq t. \tag{3.38}$$

- **LASSO** (Least Absolute Shrinkage and Selection Operator) overcomes the disadvantage of ridge regression by including a form of variable selection as well. Instead of adding an L_2 penalty term, LASSO adds an L_1 penalty, that is the sum of absolute values of the estimates (see Hastie, Tibshirani, and Friedman (2001), pp. 68-69):

$$l = \sum_{i=1}^{N} [y_i \langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle - \log(1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle))] - \lambda \sum_{j=1}^{M} |b_j|. \tag{3.39}$$

  By subtracting the absolute instead of squared values (see Equation (3.39)), some of the parameter estimates become equal to zero and are thus ignored in the final model. The above optimisation problem is equivalent to minimising the loss function subject to the L_1-norm of the parameter estimates being smaller than a certain threshold (a proof is provided in Appendix A.1.2):

$$\min_{\bar{\boldsymbol{b}}} l = \min_{\bar{\boldsymbol{b}}} \sum_{i=1}^{N} [y_i (\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle) - \log(1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle))] \text{ s.t. } \sum_{j=1}^{M} |b_j| \leq t. \tag{3.40}$$

- As explained in Section 3.1.2 a decision tree can not only be used as a classification model but also for feature selection purposes. Along with the explained different variants such as ID3, C4.5, and CART, CHAID represents a further decision-tree-based method, most often used for variable selection. While CART and C4.5 are mainly oriented towards classification by building nodes with maximum homogeneity, CHAID emphasises strong variable relationships.
  The chi-square automatic interaction detection (CHAID) was developed in 1980 by Gordon V. Kass to discover associations between a categorical target and the available predictors. The method aims at creating significant configurations from the features to detect potentially important interactions. After each feature is best split w.r.t. the Chi-Square statistic, all inputs are compared through an iterative algorithm, and the best variable is chosen for node splitting. Similar to most decision tree algorithms, the technique represents the outcome in a tree-like structure. As the name indicates, the CHAID method uses the Pearson Chi-Square statistic within the algorithm, as it is used as a criterion for the binary splitting and category grouping.

  Given a target variable $\boldsymbol{y}$ with $y_i \in \mathcal{C}$ where $|\mathcal{C}| \geq 2$ and a feature $\boldsymbol{x_j}$ with $x_{ij} \in \mathcal{C}'$ where $|\mathcal{C}'| \geq 2$, for all $i \in \{1, \ldots, N\}$, one goal is to reduce the $c \times c'$ frequency table by merging some of the feature's levels into one if the variable type permits it.

  In order to consider all possible ways of partitioning a nominal predictor into two groups one would have to compute, according to Kass (1980):

$$\sum_{g=1}^{C} \sum_{i=0}^{g-1} \frac{(-1)^i (g-i)^c}{i!(g-i)!}$$

  possible combinations, splitting c values into g groups and in the ordinal inputs' case $2^{(c-1)}$ possibilities. To avoid scanning through that many combinations, a built-in approach is suggested by Gordon V. Kass. We consider the binary case for simplicity: The first step of the technique involves creating inputs of categorical type from the underlying numeric interval variables by dividing the range into a predetermined number of categories or bins with an approximately equal number of observations. The algorithm then performs the following steps as defined by Kass (1980):

  1. For each predictor create the $c' \times 2$ contingency table as explained in Table 3.2 and apply the following steps.
     (a) Create the 2×2 frequency table using the predictor's categories pairs leading to no significant difference w.r.t. the target. If the table of Chi-Square statistics does not show any significance, combine these two categories into one compound category and repeat this step (find next

possible pair including the new compound category and apply the same method).

(b) To ensure the validity of the newly created mergers, using the created compound categories in the previous step, implement the most significant binary split (if it prevails) and go back to step (a).

DF	P-values				
	0.1	0.05	0.025	0.01	0.005
1	2.71	3.84	5.02	6.63	7.88

Table 3.3: Chi-Square Distribution Table

2. Among all predictor variables, including the merged variables, choose the predictor with the smallest p-value (see Table 3.3), i.e. the variable yielding the most significant segmentation to partition the dataset.
3. Return to step 1 for each resulting segment of the dataset if no stopping criterion is reached.

If a node meets any of the possible stopping criteria, terminate:

- The Chi-Square threshold p-value is exceeded.
- Maximum tree depth is exceeded.
- Minimum parent node or node size is not reached.

Although filter methods are easily and efficiently applicable, wrapper and embedded methods tend to perform better because the feature selection is chosen such that the classification is optimised. Filter methods also suffer from a lack of multicollinearity assessment. A feature can be irrelevant on its own but, when combined with others, show significant association to the target. Wrapper methods suffer from computational expensiveness in comparison to filter methods. Hence, the trade-off between computational time and accuracy needs to be considered. Embedded methods, in contrast, combine the efficiency of filter methods and accuracy of wrapper methods by considering feature interactions and feature subsets within the algorithm.

Especially in churn prediction, where determining a list of the relevant drivers is of high importance, dimensionality reduction is a vital step. By selecting a smaller, more powerful subset of features, classifications are more comfortable to comprehend and use for determining effective retention measures. To first ensure the validity of the classifier we evaluate quality measures introduced in the next section.

3.2.4 Modelling and Prediction

The churn prediction research history shows that some of the most used classification methods for churn forecasts include the random forest, gradient boosting, and neural network methods (see Vafeiadis et al. (2015)). Now that the theory behind these methods is introduced and the pre-processing steps of the dataset are outlined, it is crucial to comprehend the right modelling steps and sequence using these machine-learning algorithms to tackle the key problem optimally.

The supervised modelling process usually contains three phases, namely a training phase, validation phase, and test phase (see Shalev-Shwartz and Ben-David (2014), p. 150). Training and validating the model on the given dataset can be achieved in different forms. One approach to build up the training algorithm is by using the training set, then using the validation set for hyperparameter tuning and an initial performance check. Another manner is to carry out the training by only using the training set for both purposes and using cross-validation as an alternative to the classic validation procedure. To assess the fully-trained classifiers and determine the final candidate model, the test set is used for measuring performance objectively. The outcome of all on the introduced models should include an assigned forecasted probability for each observation. The last step before the overall model assessment includes determining the classification by transforming the assigned probabilities of churn to binary values. This is done for models that only learn calibrated probabilities throughout the training process. The transformation can be defined as follows:

$$\hat{y}_i = \begin{cases} 1 & \text{if } \hat{\mathbb{P}}(y_i = 1) \geq t \\ 0 & \text{else,} \end{cases}$$

where $\hat{\boldsymbol{y}}$ is the vector of binary predicted labels and $t \in [0, 1]$ the forecasted probability.

Methods such as decision-tree-based methods and neural networks which primarily deliver the forecasted classifications can vice versa be calibrated to eventually deliver posterior probabilities. When dealing with imbalanced datasets, it is crucial to accurately determine the threshold t such that the minority class can be accurately and sufficiently predicted. The choice of t can be optimised by optimising evaluation metrics such as the Precision-Recall curve or the adjusted F-measure introduced in Section 3.2.5.

The need for hyperparameter tuning and optimisation techniques is perceived as part of the validation procedure, reducing manual steps, increasing efficiency, and improving the performance of machine-learning methods. By tailoring the hyperparameters w.r.t. the dataset and modelling goal on hand, an enhancement of algorithm performance can be reached, as it was proven that different configurations work best for different datasets (see Hutter, Kotthoff, and Vanschoren (2019), pp. 3-5).

For the machine learning method denoted by $\mathcal{M}$ and the hyperparameter space $H = H_1 \times H_2 \times \ldots \times H_k$, for all k hyperparameters and the individual hyperparameter domains $H_1, H_2 \ldots, H_k$, this problem is defined by Hutter, Kotthoff, and Vanschoren (2019) (p. 5) as the fine tuning of the hyperparameters to find:

$$\hat{\lambda} = \arg\min_{\boldsymbol{\lambda} \in H} \mathbb{E}_{(D_{\text{train}}, D_{\text{val}})} \mathcal{L}(\mathcal{M}_{\boldsymbol{\lambda}}, D_{\text{train}}, D_{\text{Cross-val}}),$$

where $\mathcal{M}_{\boldsymbol{\lambda}}$ denotes the machine learning method $\mathcal{M}$ with its hyperparameters represented by $\boldsymbol{\lambda}$ and $\mathcal{L}$ the loss generated by the model using hyperparameters $\boldsymbol{\lambda}$. $\mathbb{E}$ denotes the approximate estimation, as in practice the user usually has access to a finite data subset. Once the tuning is performed and a hyperparameter set is chosen, the model with these optimal hyperparameters is finally built using the whole training dataset.

In the churn case study, we apply all of the introduced methodologies by first performing a hyperparameter tuning per k-fold-cross-validation and finally choosing the best model out of the resulting selections. We consider the different hyperparameter grids for the introduced methods as elaborated below through the typically used ranges with reference to (Probst, Boulesteix, and Bischl (2019)). The user can optimally search the hyperparameters grid systematically or randomly according to the software capabilities.

Decision trees can be fine-tuned by defining different node splitting criteria. Often increasing the maximum depth of the tree (1-50) is applied to receive a larger tree and more accurate predictions. However, increasing decision tree size can lead to overfitting. One can also define the minimum samples needed in a node for split application and the leaves (1-20). Tree-based methods like random forest and gradient boosting can be tuned similarly with a few extra tweaks. Random forests need to be assigned the maximum number of decision trees in the forest (1-2000), which can have higher scoring benefits but a much longer training time. The number of features considered for splitting can be adjusted to be between (1-$\sqrt{M}$). Additionally, all decision tree tuning can be made, as explained above. Gradient boosting has two hyperparameters sets, one for the boosting process and one for decision tree growth. The boosting process includes setting the number of iterations (1-5000). Increasing the number of iterations should be accompanied by a lower shrinkage parameter to offset the overfitting effect. The user should also set a relatively high training proportion to avoid a potential loss of information.

To summarise, the modelling procedure can be outlined in the following sequence. First, a machine learning method $\mathcal{M}$ needs to be determined along with the definition of the required hyperparameters grid of the respective model. Next, the model is trained only on the respective training folds and validated by evaluating the CVE. The final chosen hyperparameters are used to train the model on the entire training set. Lastly, the comparison of model performance is assessed via evaluation measures on the test set.

3.2.5 Model Evaluation

The final vital key to determine optimal classifiers is the choice of suitable evaluation metrics, allowing the user to determine the best possible model for the underlying prediction goal. Since the final model choice is based on the selected performance measure, a wrong choice w.r.t. the dataset characteristics and end goal can mislead the user about the actual model capacity and quality. Both goodness-of-fit measures suitable for the different methodologies and generally applicable classification performance measures can be considered for the overall assessment. Choosing an appropriate method for several objectives is challenging, especially for models trained on imbalanced datasets and skewed class distribution. In this section, we present, considering class imbalance conditions, the standard metrics in binary classification frameworks used to ensure validity, reliability, and generalisability of the discussed modelling techniques.

First, the familiarisation with the concept of a confusion matrix resulting from the final predicted labels of the individual observations in binary classification frameworks is necessary. By setting a certain probability cutoff threshold on the predicted probabilities, the frequencies can shift by leading to different counts in the confusion matrix. The matrix can be visualised through the following table layout.

		Classification Outcome	
		N′	**P′**
Actual Label	**N**	True Negative (TN)	False Positive (FP)
	P	False Negative (FN)	True Positive (TP)

Table 3.4: Confusion Matrix Composition

The true positives (TP) in Table 3.4 represent the number of correctly classified event observations. In the case study context, this is the number of clients labelled as churners and correctly classified by the trained model as churners. Similarly, (TN) represents the number of correctly classified non-event observations. In the case study context, this is the number of clients who are labelled as non-churners and are correctly classified by the trained model as non-churners. In contrast, false positives (FP) are the count of labelled non-churners, which are misclassified by the model as churners and vice versa for false negatives (FN). Columns of the confusion matrix consist of the predicted classes, while

the rows represent the actual classes. However, none of the above measures are solely adequate for a final model choice; thus, threshold metrics were introduced to unify their evaluation strengths. According to Hossin and Sulaiman (2015), evaluation metrics can be divided into the three main categories threshold, ranking, and probability metrics.

Threshold Metrics

Threshold metrics are based on the confusion matrix components and include the measures which change based on the chosen cutoff value, as reviewed by Hossin and Sulaiman (2015):

- Accuracy:
$$\frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{FP}+\mathrm{TN}+\mathrm{FN}}. \tag{3.41}$$
Accuracy is the fraction of correctly positive and negative classified events. In the case of highly imbalanced datasets, accuracy can be very misleading as the TN frequency increases the overall accuracy, hiding the true performance of positive event prediction, which is the primary goal of rare event modelling. Accuracy will therefore not be used in the upcoming churn case study.

- Misclassification Rate (MR):
$$\frac{\mathrm{FP}+\mathrm{FN}}{\mathrm{TP}+\mathrm{FP}+\mathrm{TN}+\mathrm{FN}}. \tag{3.42}$$
MR is the fraction of wrong positive and negative classified events. Similar to Accuracy, MR is not an adequate metric for models trained on imbalanced datasets.

- Precision:
$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}. \tag{3.43}$$
Precision represents the fraction of correctly classified events among all event classifications, e.g. actual true churners between all classified churners. That is, precision is the accuracy of event classifications and therefore of high relevance.

- Recall, True Positive Rate (TPR), Sensitivity or Hit Rate:
$$\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}. \tag{3.44}$$
Sensitivity represents the fraction of correctly model-classified events among all actual events, e.g. the proportion of correctly classified true churners. Sensitivity however, does not take into account false positives, thus high Sensitivity is only useful to rule out non-churners from the event category but not to detect them.

- True Negative Rate (TNR) or Specificity:

$$\frac{\text{TN}}{\text{FP} + \text{TN}}. \tag{3.45}$$

Specificity represents the fraction of correctly model-classified non-events among all actual events, e.g. the proportion of correctly classified true non-churners. In contrast to Sensitivity, high Specificity is useful to rule out churners from the non-event category rather than detecting them. Specificity can also be computed by 1 + FPR.

- False Positive Rate (FPR):

$$\frac{\text{FP}}{\text{FP} + \text{TN}}. \tag{3.46}$$

The FPR is the ratio of model-misclassified non-events as events among all true non-events.

- Geometric Mean (G-Mean) (see Akosa (2017)):

$$\sqrt{(\text{Sensitivity} \times \text{Specificity})}. \tag{3.47}$$

The G-Mean provides an evaluation measure on the balance of classified majority and minority classes. The lower the G-Mean, the poorer the classifier's performance in classifying events irrespective of the good non-events classification performance. This metric is a good indicator of the extent of underfitting of events.

- F-Measure, F-Score or F_1:

$$\frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{3.48}$$

The F-Measure combines both Precision and Recall aspects by determining the harmonic mean of both. The measure, however, is not powerful for imbalanced datasets as precision and recall have the same weights. Instead, we consider the adjusted F-Measure for the underlying case study.

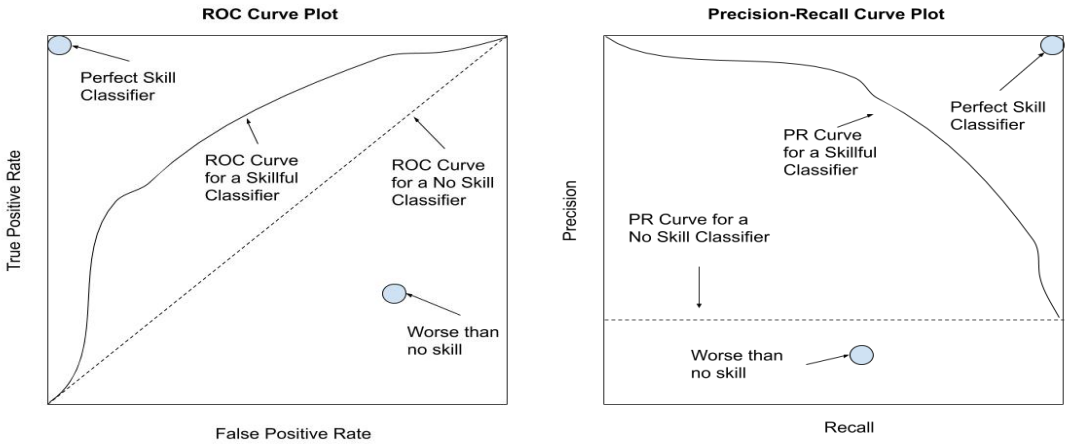
- Adjusted F-Measure or F_β (see Akosa (2017)):

$$\frac{(1 + \beta^2) \times \text{Precision} \times \text{Recall}}{(\beta^2 \times \text{Precision}) + \text{Recall}}. \tag{3.49}$$

By increasing beta above 1, the metric tends to favour increasing the Recall instead of equally balancing Precision and Recall. As in some cases, FN are more costly than FP, Recall should be considered rather than Precision. $0 < \beta < 1$ implies a higher importance of Precision, while $\beta > 1$ indicates a higher weight for Recall.

Ranking Metrics

- ROC and AUC:
  The ROC (Receiver Operating Characteristic) curve displays the classification model performance at different thresholds. It plots the TPR (Sensitivity) on the y-axis against the FPR (1 − Specificity) on the x-axis. The curve represents a trade-off between costs stemming from non-churners wrongly classified as churners vs. the potential benefits from the correctly classified churners. It shows that increasing the TP frequency comes with the cost of also increasing FP frequency.
  The AUC (Area under the ROC Curve) is used as a performance metric of the ROC curve measuring the ROC's area under the curve with values ranging from 0 to 1 and representing the model's TP ranking capabilities. The nearer the ROC curve to the top left corner, the larger the AUC and the better the classifier's performance, as the ROC curve is closer to the coordinates (0,1), i.e. closer to the model exhibiting 100% Sensitivity and Specificity (see Figure 3.11(a)). An AUC value of 1 therefore implies a perfect model, while a value of 0.5 conveys it is as poor as a random model.



(a) ROC Chart (b) PR Chart

Figure 3.11: ROC and PR Charts By Brownlee (2020)

- PR (Precision-Recall) Curve:
  The PR curve displays the model's precision and recall performance at different thresholds. It plots the TPR (Sensitivity/Recall) on the x-axis against the Precision on the y-axis, with the blue dot in Figure 3.11 representing a coordinate going through a classifier's curve. The curve represents a trade-off between trying to achieve high accuracy through high Precision while keeping a high event classification capability, i.e. a high Recall. Ideally, the user wants a model with many

event classifications, that are also labelled correctly. The height of the baseline curve in Chart 3.11(b) is equal to the proportion of events in the dataset.

To explain the upcoming ranking metrics Gain and Lift we need to use the model's predicted probabilities. Given the predicted probabilities, $\hat{p}_i = \hat{\mathbb{P}}(y_i = 1)$, we score our underlying observations by ranking their corresponding probabilities from the largest to the smallest value. As elaborated by Shmueli (2019), the probabilities are then ranked by applying the transformation $R(\hat{p}_i) = N - i + 1$ with the highest ranked observation fulfilling $R(\hat{p}_N) = 1$.

- Cumulative Gain and Lift:
  Cumulative Gains represent the number of true classified events among the top n ranked observations (Shmueli (2019)).

$$\mathrm{CG}(n) = \mathrm{TP}_n. \tag{3.50}$$

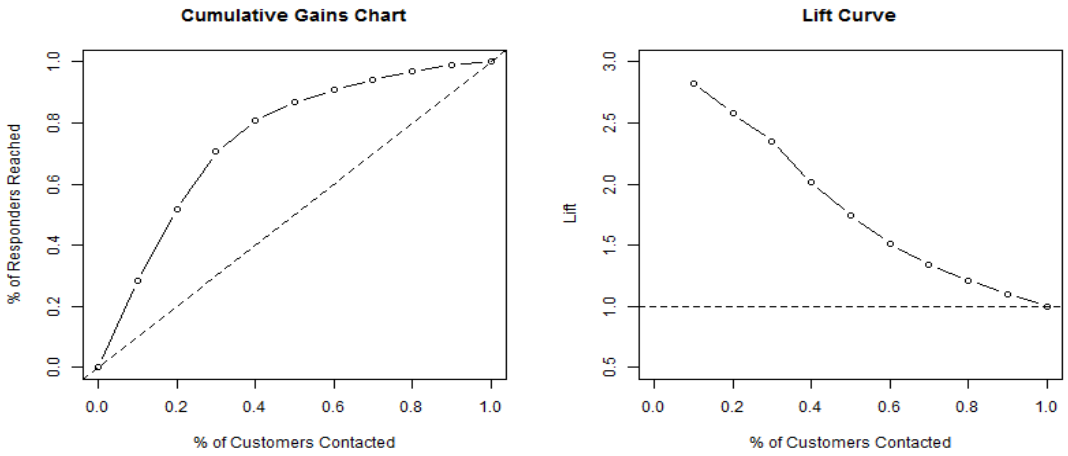
  Lift represents the proportion of Cumulative Gains vs. random targeting of a population from the test dataset. If the model is implemented, Lift is a general indicator of how much better it would capture response compared to random targeting of risk customers from the sample. The higher the Lift, the better the performance vs. random samples (Shmueli (2019)).

$$\mathrm{Lift}(n) = \frac{\mathrm{TP}_n}{\frac{n}{N}(\mathrm{TP} + \mathrm{FN})} = \frac{\frac{\mathrm{TP}_n}{n}}{\frac{(\mathrm{TP}+\mathrm{FN})}{N}}. \tag{3.51}$$

  An alternative to the use of classification frequencies and confusion matrices are Lift and Gain charts. Lift charts are commonly used in the evaluation of churn prediction models because most underlying datasets are imbalanced, leading to biased confusion matrices and hence misleading threshold performance measures.

- Cumulative Gain Chart and Cumulative Lift Chart:
  The Cumulative Gain chart can be visualised by plotting $\mathrm{CG}(n)$ as a function of the targeted observations n (see Figure 3.12(a)). A classic Lift chart similarly plots the ratio from Equation (3.51) as a function of n. A more powerful and commonly used visualisation is the Cumulative Lift chart which plots the Lift as $\mathrm{Lift}(n/N)$ (see Figure 3.12(b)). That is, the lift on the y-axis corresponds to the respective decile. This shows a clearer version of the targeting.

The Cumulative Lift chart can be interpreted by considering the different percentiles. Considering Figure 3.12(b), if the 10% percentile, i.e. the upper 0.1 proportion of scored customers are contacted, the model shows a 2.75 higher capture rate than a random 10% sample. That is, the model would capture 27$\dot{5}$



(a) Cumulative Gains Chart (b) Lift Chart

Figure 3.12: Gain & Lift Evaluation Charts By Littler (2020)

Probability Metrics

Probability metrics are mainly used for models delivering the likelihoods of each observation belonging to the respective class label. The metrics are based on the predicted probabilities rather than counts of each class label.

- Similar to the commonly used MSE in numerical target prediction, one can measure the deviation between true and predicted labels by computing the average square error (ASE) for a binary target, where $\hat{p}_{ic}, c \in \mathcal{C}$ represents the predicted probabilities of the classes for observation i as defined by SAS Institute Inc. (2017):

$$\text{ASE} = \sum_{i=1}^{N}\sum_{c=1}^{C}\frac{(\mathbb{1}_{\{y_i=c\}} - \hat{p}_{ic})^2}{CN}. \tag{3.52}$$

- Log Loss:
  Last but not least, we re-introduce the Log Loss measure used for quantifying the performance of the models' predicted probabilities. Log Loss measures the divergence of the observations' predicted probabilities of classification from the actual labels. Hence, a perfect model would have a Log Loss of 0 and a poorly performing model would deliver a high Log Loss. Log Loss and Cross-Entropy are equal when computing the error rates for models. We refer to Equation (3.10) and Equation (3.16), delivering the Log Loss measure:

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i). \tag{3.53}$$

As demonstrated, there are numerous evaluation metrics with different possible deployments regarding the business and research objectives. In the underlying churn use case, we mainly focus on less class-imbalance sensitive measures such as Recall, Event-Precision, Cumulative Lift, and AUC value. The practical implementation of the models and subsequent evaluation using these measures is presented in the next chapters.

Chapter 4

Case Study: Insurance Data

4.1 Data Description and Visualisation

ERGO is an insurance group offering a wide range of insurance products and operating in the following sectors: property and casualty, life, health, legal, and travel insurance. The confidentiality of its clients' insurance data leads to a separated pre-processing necessity by different teams. After the data is extracted from the data warehouse and imported into a separate environment, it is brought in a pseudonymous[1] form by the data engineering team and cannot be analysed by the same individuals. The pseudonymous data is then analysed by the data science team to train the respective models. Every three months (quarterly) an automatic data extraction and script calculation are performed to deliver a suitable ABT[2] for the models of choice. We are aiming at providing a model for each of the following four insurance categories: household, legal, liability, and accident insurance. For each insurance category, we create a different ABT with primarily the same features. The main differences lie in the target variable and values of variables with respect to clients' changes and perspectives, which is analysed next.

Household Insurance - Dataset Overview	
Number of clients observed	289,103
Features	2250
Churn Frequency	3.41 %
Median time insured	15 years

Table 4.1: Household Insurance Dataset Overview

[1] Individual identification characteristics, like name or address, are replaced by pseudonyms.

[2] An Analytical Base Table (ABT) is a dataset in which all data preparation steps are applied, including the generation of the target and all features.

The household insurance dataset being examined (see Table 4.1) contains 289,103 observations and 2250 predictor variables, the target variable, and one identification variable (client ID). This dataset is then stratified into a training set containing 80% of the data points, i.e. 231,280 observations, and a test set containing the other 20%. The test set remains aside until the end of the prediction process for the purpose of a final out-of-sample assessment.

Table 4.2 shows a summarised overview of the available variable types and their frequencies. It contains four different types of data: binary, nominal, unary (variable consists of one value only) and interval. We can see that we are handling a mixed dataset with a big segment of nominal data representing 43%, interval data 35%, binary 11%, and unary 11%. In order to reduce the number of variables, we set a missing values threshold of 99%, i.e. if at least 99% of the values in a column are missing or consist of one attribute, the variable is left out of the imported dataset. Likewise, nominal variables with a vast number of levels are left out. By setting the threshold of number of classes to 25, a nominal variable with more than 25 classes will be rejected. Furthermore, interval variables with less than 20 levels will be marked as nominal.

Variable Role	Variable Type	Frequency
ID (Client Number)	Interval	1
Target	Binary	1
Input (1428)	Binary	190
	Interval	626
	Nominal	612
Rejected (822)	Binary	32
	Interval	123
	Nominal	313
	Unary	206
	Time-ID	148

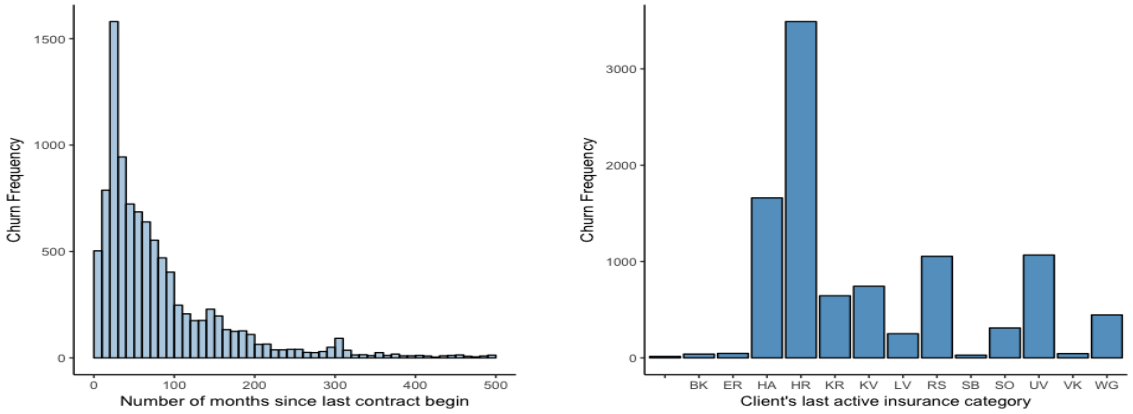
Table 4.2: Household Insurance Dataset Feature Overview

The data also includes ten general variables representing the different insurance categories that could provide compelling information about the client. One can observe that some variables are similarly considered in all insurance categories. The categories mainly represent the ten different insurance sectors: accident, construction loan, disability, property and casualty, life, health, legal, car, care, and dental insurance.

Clients with active contracts in the respective insurance category will be part of the table, along with gathered information from other insurance categories' tables. Not all the clients in the household insurance ABT are present in the accident insurance ABT. E.g., if the

clients hold both an active accident and household insurance, they will be represented by a unique observation row in both the accident-ABT and household-ABT. The main differences between the ABTs lie in the following variables. Firstly, the target variable differs with respect to the insurance product, since the creation of the target variable is based on the clients' previous churn behaviour and the date of their outstanding contract's end. The contract is always insurance category specific. The variables referring to contract end are linked to the respective insurance being observed, e.g. in the household dataset, we consider the contract end-date of the client's household contract, even if this client has another insurance product with a similar contract end-date. Hence, these variables also differ between the ABTs.

Next, we analyse the target variable, which we aim to model by using the presented methods in Chapter 3.1. Given a client's information, the model should be able to predict whether the customer will churn or not and with which probability during the observed time frame it will occur. Hence, the given target variable represents avoidable churn occurrence and is defined with respect to two aspects. First, the group of clients who have cancelled their outstanding contract 4-8 months before contract end, i.e. 1-5 months before the cancellation deadline (3 months before contract end) is selected. Secondly, all clients with an outstanding contract and only 4-8 months to go till contract end are further chosen. Labelling these cases by 1 and all others by 0 defines the target variable of the dataset. The target variable includes two classes; class 1, which includes 9848 observations, and class 0, which includes 279,255 observations.



(a) Time Elapsed

(b) Last Active Sector

Figure 4.1: Features' Univariate Exploratory Analysis

To have an illustration of the general interaction between features and the target variable, we analyse some intuitively important input variables in the dataset, such as churn behaviour with respect to regions, age, and contract specific characteristics in association

with the target variable (see also Figure A.1). In Figure 4.1(a) we can see a high churn frequency for clients with a shorter contract activity, i.e. clients who have been insured for 180 or more months, equivalent to 15 or more years, churned less often compared to individuals who have only started their first contract since 30 months, i.e. two and a half years. Figure 4.1(b) shows how clients who have an active contract of property (HR) and casualty (HA) insurance out of all insurance categories were more likely to churn.

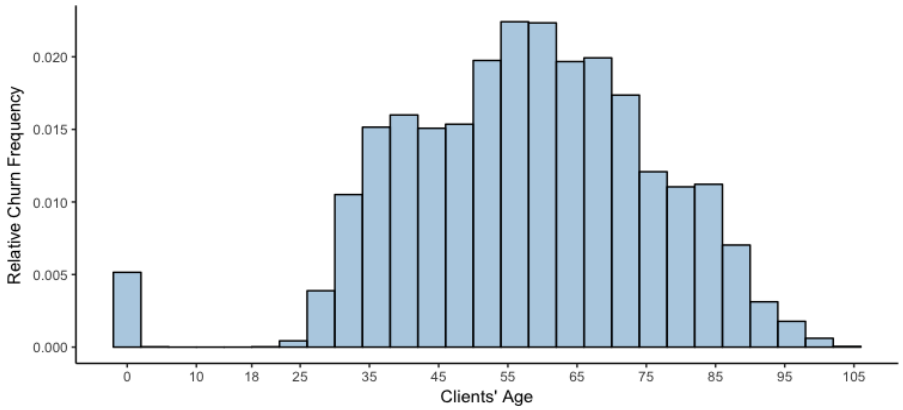


Figure 4.2: Relative Churn Frequency Per Age Group

In Figure 4.2, the churn behaviour is almost normally distributed along with all age groups. It is visible how the highest frequencies of churn occur between the ages of 48 to 53. This appears reasonable, as individuals in this age group might change their household in need of different insurance. We can also see a large number of 0-aged churners. This is due to missing values that are labelled by the age of 0. These examples show the necessity of exploratory analysis to detect outliers or inconsistencies. This is fixed in Section 4.2.

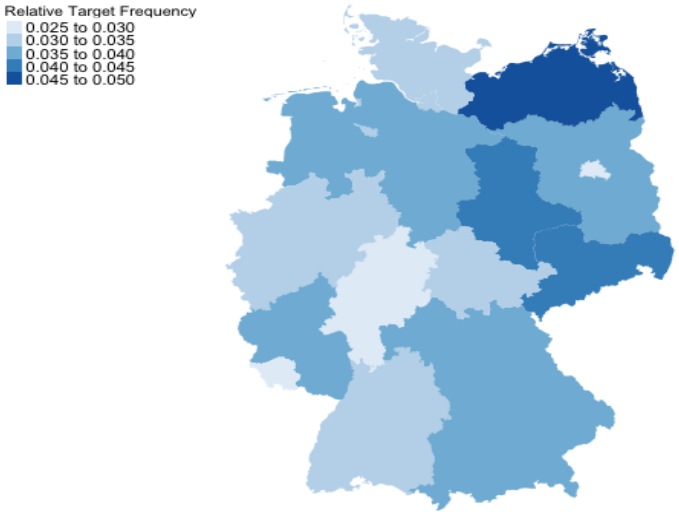


Figure 4.3: Relative Churn Frequency Per Region

In Figure 4.3, it is visible how the highest relative frequencies of churn occur in the

north-east regions of Mecklenburg-Vorpommern and Saxony, followed by Saxony-Anhalt and Bavaria. Considering that the second-largest activity location of ERGO's insurance services is provided in Nürnberg, along with many headquarters of rival German insurance companies, it is reasonable that this can lead to higher churn frequency in Bavaria due to higher competition levels. The eastern states' high frequency can be possibly caused by the distance factor to the main agents of ERGO primarily located in Nordrhein-Westfallen.

We can provide a more detailed overview of the input variables by categorising them into time-variant and time-invariant variables, as well as into diverse general categories.

Category	Specifications	Time Variance
Socio-demographic Characteristics		
Customer ID		
Age, Gender, Family Status	Basic Information	
Income, Job, Education Level	Socio-Economic	
Region, Location, ZIP-Code	Micro-Geographic	
Contract and Product Characteristics		
Agent's change in location		
Number of agents	Agent related	
Time elapsed since agent's contract begin		✓
Agent's Gender		
Premium price		
Payment method	Product	
Insurance type		
Time elapsed since last contract end	Contract	✓
Tine elapsed since last contract begin		✓
Company - Customer Relation		
Elapsed time since first/last cancellation		✓
Cancelation reason	History of churn behaviour	
Cancellation in other contract types		
Number of churns		
Cross-selling category of last churn		
Number of complaints		
Elapsed time since last complaint		✓
Elapsed time since start of relation		✓

Table 4.3: Features' Categories Overview

From the variables' overview in Table 4.3, we obtain a good understanding of the potential available and possibly influential variables in the model and can adjust our model settings to adapt to the dataset nature. Post-modelling, the user can detect the most important categories and further enhance or increase the available features lying in it.

4.2 Data and Features Pre-Processing

As introduced in Section 3.2.1, extensive data-mining and pre-processing is necessary for robust training of the considered models. We adaptively implement the presented approaches on the given raw dataset in our case study. The dataset is provided quarterly through the given server for processing by the models. Most of the pre-processing work is automated in the steps before our interaction with the dataset. So the raw dataset already includes merged data with one standard format. Since ERGO has several data sources due to its diverse insurance activities, the format of each needs to be taken into consideration. Hence, the raw dataset delivered includes the variables and their assigned names, the data format of each, as well as some features generated through variable interactions.

We start by giving a short introduction to the SAS Enterprise Miner (SEM) software we are using for the prediction. By creating a new diagram in SEM, we can illustrate the modelling process through a flowchart. SEM uses the so-called 'SEMMA' (Sample Explore Modify Model Assess) approach, which allows the user to first sample and explore, then model, modify, and assess the data. This approach already makes it easier for the user to include all appropriate steps in the modelling process required to obtain a sound model. After creating a new project and new diagram, the raw dataset is imported as-is from the particular server via the 'Data Source' button. During the import of the dataset, SEM requires the user to enter some pre-processing information, as mentioned in Section 4.1. This already includes rules settings pre-determinable for the handling of missing values and categorical values with numerous levels.

First, we conducted an exploratory analysis of the underlying dataset to have a proper visualisation to spot outliers and detect pattern. These results were already presented in Section 4.1.

Second, a partition of the data is necessary to implement all pre-processing steps on the training set and set the test dataset aside. We choose to split the dataset into a training dataset containing 80% of the data and a test dataset containing the other 20% of the data. As no distinct validation dataset is set aside due to data scarcity, validation has to be obtained through 5-fold cross-validation on the training set. To perform the 5-fold cross-validation in SEM, the 'Transformation' node is used. The data is randomly segmented by applying the formula $\text{int}(\text{ranuni}(1) * 5) + 1$ on the training set.

This random transformation (no stratification available in SEM) allocates a fold with an index from 1 to 5 to each observation. When we train the model and tune the hyperparameters, we consider the folds as groups in the SEM context, and each iteration corresponds to one loop through start group to end group.

An illustration of the cross validation procedure in SEM is provided in Figure A.5.

Finally, imperfections revealed through the analysis are removed by handling missing values, inconsistencies, and duplicates. We start by imputing the missing values through the 'Impute' node before applying logistic regression and neural network modelling steps. Since decision tree already handles missing values within the algorithm, there is no need to add this as a pre-step. A final data check is performed to avoid inconsistencies (see Figure 4.4).

When handling missing values, we can either remove the instances which have missing values or remove the missing values directly by imputing the variables. Since we have a large number of 1422 feature variables, we decide to implement both methods to reduce the challenge we are facing and adequately handle the resulting effect of missing values. First, we remove all variables with a 99% missing values quota when importing the dataset. Then we impute the continuous variables by filling their missing values with the mean and categorical variables by replacing the most commonly occurring class or count. It should be noted that count stands for modal value. We also set the missing cutoff value to 80%. This is the maximum percentage of missing values allowed for a variable to be imputed. Any variable exceeding this cutoff is rejected. Otherwise, inputs with a missing values percentage less or equal than 80% are replaced by an adjusted new input with its missing values replaced by the determined synthetic value. SEM thus rejects 718 variables, containing more than 80% missing values, and imputes 704 variables complying with the cutoff value, representing the new feature space.

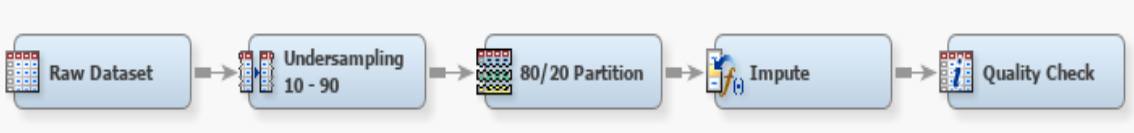


Figure 4.4: Pre-Processing Flowchart Overview

Further vital pre-processing steps to consider here are the grouping of excessive categorical input variable levels that might lead to overfitting and adjusting the roles of entries assigned by SEM. Variables can be grouped in SEM by using the 'Replacement' node or within the methods. We decide to do this at a later point if required. SEM can misinterpret variable notation; for instance, the role of a variable including the word 'segment' in its name is automatically set to 'Segment', which has to be readjusted. RS is also falsely interpreted by SEM as residuals.

Analysing the first implemented steps demonstrates the importance of initial data and features pre-processing to complete the subsequent modelling steps successfully.

4.3 Class Imbalance Reduction

In the considered use case, we handle a dataset containing only a 3.4% frequency of churn labels. Applying any of the class imbalance reduction techniques introduced in Section 3.2.2 can help to increase the predictive accuracy of event observations. Considering SEM's technical availabilities, random under-sampling and SMOTE for continuous features are applicable. Therefore, we consider only under-sampling in this case study but highly recommend considering the SMOTE-NC technique as the study done by Batista, Prati, and Monard (2004) shows that over-sampling techniques provide more accurate results than under-sampling w.r.t. the ROC Curve. Especially SMOTE provided good results on a considered dataset with not so many positive cases. The models based on un-sampled data are compared to those with under-sampled pre-training to obtain validation and comparative statistics. We can observe the following confusion matrices on the test data for different models when the data is un-sampled.

Model	TN	FP	FN	TP
Logistic Regression	55833	19	1956	15
Random Forest	55852	0	1971	0
Neural Network	55852	0	1971	0

Table 4.4: Un-sampled Model Evaluation

Table 4.4 shows how almost all models can not identify any churn event observations. Logistic regression solely identifies a total of 15 churn cases, which makes about 0.03% of the test set. It is important to determine the under-sampling ratio cautiously as information is being disregarded, making a large information loss disadvantageous. We consider an appropriate ratio with respect to the dataset size and target frequency, given that the total raw dataset has 230k observations and only 3.4% target frequency. Choosing a 50% for under-sampled representation, for instance would remove too many majority class observations leading to a high loss of information and possibly a high misclassification rate. For illustration, Table 4.5 confirms this statement by findings of the use case.

Model	TN	FP	FN	TP	MR	C-Lift
Logistic Regression	1189	635	1336	782	50%	1.88
Random Forest	1048	518	1453	923	50%	1.83
Gradient Boosting	1104	535	1436	867	50%	1.88
Neural Network	1178	670	1301	793	50%	1.72

Table 4.5: 50%-Under-Sampled Model Evaluation

Table 4.5 shows how all models exhibit an accuracy equal to 50 %, which is considered an only fair performance given that the dataset is now balanced. Misclassifying too many non-event observations can lead to a distraction from the true churners desired to be identified. It is also important to note how observing only 6.8% of the dataset on hand, i.e. 19716 observations as 50%-under-sampled model does, can consequently capture less of the at-risk customers. This conclusion is deducted by the small demonstrated Cumulative Lift of the 5% percentile, only achieving about 1.9 times higher response capture than a random 5% sample.

After considering commonly used ratios and using the training set to determine an adequate ratio, we implement a 10-90% stratified under-sampling in SEM, i.e. 10% is the proportion of the created sample containing the churn level. This is done by setting the Sample node method to Stratify, the Criterion Property to Level Based, Level Selection property to Event, and Sample Proportion to 10. By determining the Level Proportion to be 100%, i.e. all event observations should be included in the sample, the 100% would make up the 10% of the newly generated sample.

Table 4.6 shows how all models exhibit an overall Precision above 64%, which is considered a good performance since the dataset is still imbalanced. In comparison to the 50%-under-sampled Cumulative Lift, the models now show an almost overall C-Lift of 4, implying a higher response capture. It is also important to note that the FP count is substantially lower and the FN count comparable, leading to a MR of only 9.8% despite the bigger size of the examined set.

Overall, the enhanced model performance observed through the model evaluation statistics confirms the importance of incorporating any class imbalance reduction techniques in churn prediction.
It is important to note that any dataset transformation should be applied in a stratified manner. That is, dataset partition should be generated through stratified sampling as random sampling can lead to a disadvantageous distribution of the rarely available event observations. Similarly, cross-validation folds should be segmented through stratified sampling.

Model	TN	FP	FN	TP	MR	C-Lift
Logistic Regression	17661	68	1872	99	9.8%	3.89
Random Forest	17615	114	1767	204	9.5%	4.42
Gradient Boosting	17634	95	1776	195	9.4%	4.25
Neural Network	17564	165	1772	199	9.8%	4.08

Table 4.6: 10-90%-Under-Sampled Model Evaluation

4.4 Dimensionality Reduction

Considering the pre-processed non-imputed dataset of the case study, including 1422 variables, feature selection is a vital step to improving prediction accuracy and reducing the computational costs. As introduced in Section 3.2.3, there are numerous feature selection methods for such a dataset and churn classification problem that will be implemented in our case study in this section.

Variable Selection for Logistic Regression

We start by presenting the different implemented selection methods as a step within the logistic regression modelling procedure. The performance of logistic regression models increases when a thorough variable selection is applied. Since the underlying dataset includes a large number of features, several approaches are considered. After imputing the variables with a missing percentage of 80% and rejecting the rest, as mentioned in Section 4.2, we have an input space of 765 variables. First, the CHAID tree embedded method was performed to reduce computational time, followed by a stepwise wrapper method within the regression model. Next, ignoring computational time, all 765 variables were entered directly into the stepwise wrapper method. Last but not least, a LASSO regression was performed as an alternative to the classic logistic regression model. The selections were compared to the full saturated model with no selection applied.

	# Variables	# Coefficients	Run Time
CHAID	152	821	4min 01s
Stepwise Selection	30	112	31min 05s
CHAID + Stepwise Selection	30	135	4min 34s
LASSO	104	118	3min 57s
No Selection	765	2438	21min 51s

Table 4.7: Comparison Of Variable Selection Methods

Table 4.7 [3] provides an overview of the number of variables selected by each method and their resulting number of coefficients. Applying stepwise selection or a mixture of both CHAID and stepwise selection method results in 30 variables, the fewest number in total. LASSO, in contrast, has 104, a relatively higher number of variables since all inputs are considered, and only some coefficients are set equal to zero. The number of coefficients (118) in the LASSO model is, however less than 135 coefficients of the CHAID + Stepwise

[3]The run time was determined w.r.t. running the program on the SAS Server in the ERGO IT-Infrastructure and can not be further specified.

model, implying a simpler model complexity despite a larger number of overall variables. It also implies a smaller number of nominal inputs due to less generated scores for model parameters. Applying no selection, the all-features regression model delivers a model with 765 variables and is relatively inefficient.

We compare the top-ranked 20 variables of the CHAID and stepwise selection methods next to check significance (see Table A.1 for the LASSO selection).

Stepwise Selection	CHAID
HH_Ph_State_Key_A1	Churn_Last_No_Mon
Id_HH	Geo Cluster of kgs12
AG_Contr_Prop_A_Beg_Dat_1Y	Geo Cluster of kgs16
CL_Doc_Reas_A1	AG_Chg_Resp_OE_Pnr_1Y
HH_Ins_Cond_Year_A1	State
Kgs12_Prop_Hs	Life stage of kgs12
HH_Contr_Chg_Reas_A1	No_Mon_First_Contr_Beg
Churn_3Mon	UP_PERSTYP_A_1
Churn_3Mon_No_CI	Churn_Last_CS_Cat
AG_Contr_Prop_A_End_Dat_1Y	No_Mon_First_Contr_Beg_HH
Job_Pos	Initial_Product
Loc_Chg_3Mon	HH_Ins_Cond_Year_A1
Churn_Last_CS_Cat	UP_PERSTYP_S_1
Delta_No_Cat_3Mon	Churn_3Mon
Delta_Kompo_Contr_3Mon	Job_Pos
AG_Chg_Resp_OE_Pnr_1Y	ANZ_ERST_END_CS_SP
KKM_A_P_Quelle	VM_ANZ_VMT_P_KNR_AKT_V
Kgs16_Rentalscore	Age
No_Cars_HH	Rental score of kgs12
HH_EndDate_Cat	No_Mon_Last_Contr_Int_Beg_Dat

Table 4.8: Case Study - Selected Input Variables

The above abbreviations kgs12 and kgs16 refer to the municipality codes based on micro-market and street granularity, respectively. HH refers to household, C & L to Casualty & Liability and # to the number of a considered variable. It is important to note that most of the dataset features' abbreviations are explained in Tables A.2 and A.3. One can observe from Table 4.8 the similarities between the selected variables of each method. The variables *Churn_3Mon* and *AG_Chg_Resp_OE_Pnr_1Y* are ranked in the top 20 variables of each selection method. The client's existence, or lack of a recent churn respectively, can indicate the upcoming churn in another insurance category. A change of the responsible agent seems to affect the client's churn decision as well. Most variables proven to be significant are either client-related basic information like age, life stage, region, job position or contract, and agent-based features. It is also visible how CHAID

takes into account more micro-geographic variables such as geographical clusters and rental scores of the respective municipalities. In the following paragraphs, an explanation of the technical settings and implementation of the methodology in SEM is provided.
In the SEM node, 'Variable Selection' one can choose a variable selection model associated with the target variable. The possible choices are the CHAID tree, R-Square method, or both. Considering the high number of categorical and binary variables in the feature space, we consider the CHAID tree-like selection more suitable. CHAID also detects possible interactions between the variables. Constrained by the available selection methods in SEM, the CHAID tree might not provide the optimal features subset for regression purposes but can be used for a substantial reduction to keep an overview of the input variables.
The procedure works as explained in Section 3.2.3. In SEM, one can also specify the number of bins or categories in which the range of the numeric variables is divided for the tree-splits. One can also predetermine the minimum Chi-Square cutoff value for variable selection and the number of passes through the training data for performing the binary splits. The default number of bins is 25, cutoff value 3.48 corresponding to 5% significance (see Table 3.3), and the default number of passes is 6.

Target Model	Chi-Square
Chi-Square Options	
Number of Bins	25
Maximum Pass Number	10
Minimum Chi-Square	6.63

Table 4.9: CHAID Tree Settings

Model Selection	
Selection Model	Stepwise
Selection Criterion	AIC
Use Selection Defaults	No

Table 4.10: Wrapper Methods Settings

By increasing the number of passes, the process will require more computational time, which might improve the model performance. The number of passes controls the depth of the tree that is fit. By increasing the number of passes, the potential depth of the tree increases. The increase can impact which variables are ultimately selected since different numbers of splits on the same variable will have different predictive capabilities. The more trees are considered, the more accurate the method is at identifying the best predictors that can be identified using that method. The number of passes, however, does not impact the number of variables that are selected directly. The Minimum Chi-Square property, on the other hand, has a direct impact. The higher the Minimum Chi-Square value specified, the fewer the variables which will be retained and the higher their significance. The feature selection method is included in the cross-validation procedure of the model to determine optimal parameters. The resulting parameter suggestions, see Table 4.9, are 10 passes, 25 bins, and a cutoff value of 6.63 corresponding to a significance level of 1% (see Table 3.3).

In the SEM 'Regression' node, one can choose the direction of the wrapper methods

between Forward, Backward, or Stepwise and the final model selection criterion such as AIC, BIC, cross-validation error or Misclassification Rate, see Table 4.10. In order to optimise the choice of these three parameters and wrapper methods, a tuning is provided via 5-fold cross-validation. The combination of model parameters and wrapper direction, leading to the smallest cross-validation error, is chosen for use in the training cycle.

Variable Selection for Random Forest and Gradient Boosting

In comparison to logistic regression, the base feature selection for random forest and gradient boosting is automatically made through a decision tree. It is important to note that imputation is not performed for the RF and GB model, as it is already incorporated in the modelling. Instead of entering all 1422 variables right to the methods, it is preferred to pre-select the variables providing the best tree performance. The chosen tree selects the variables within 11 minutes, on average, more slowly than LASSO and CHAID.

We can see from Figure 4.5 that some of the top 20 ranked variables by the decision tree are also included by the logistic regression selection methods, which validates their significance. We note that the determined importance in Figure 4.5 is based on the reduction of the respective Gini Index over the sum of all nodes. These include the number of months since the last churn, churn in the last 3 months, and the number of months since the start of the first contract. We have seen from the exploratory analysis and the variable selection 4.8 that intuitively, clients who have had an insurance contract for a longer time churned less frequently. This information could be powerful in predicting future churn behaviour, as visible in Figure 4.5. This detailed understanding of influential variables will allow an effective measures extraction for potential churners.

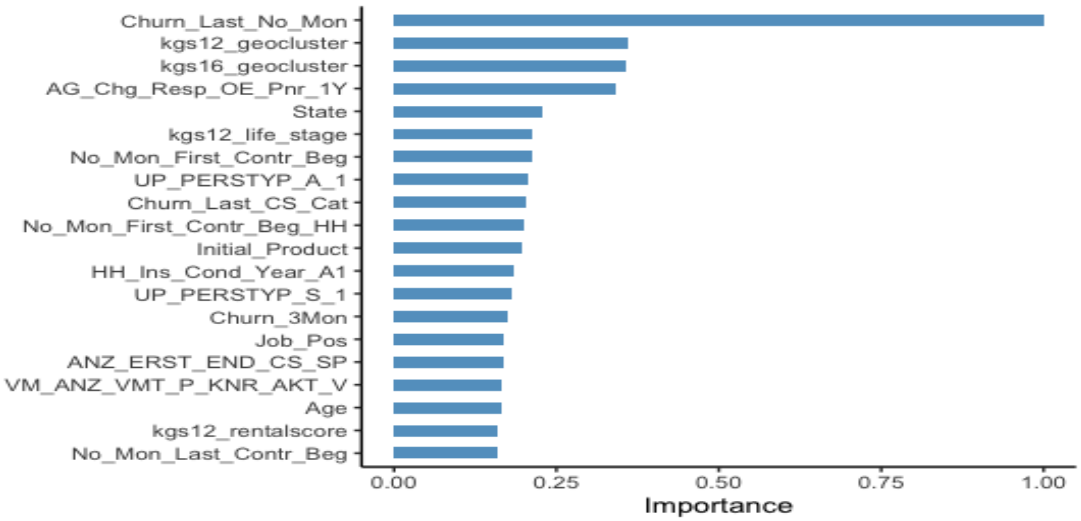


Figure 4.5: Implemented Decision Tree - Variable Importance

Chapter 5

Case Study: Prediction Results and Evaluation

In this chapter, we summarise the findings of modelling using the methods introduced in Chapter 3, and the insurance dataset provided by ERGO, described in Chapter 4. We first provide an overview of the results and a respective interpretation of the findings. The evaluation via the defined assessment measures is chosen considering the respective research goal and business objective. A final discussion via model comparison and final model selection will be further elaborated in the next chapter.

5.1 Logistic Regression

We start off by presenting the results of the classic logistic regression model. It has been illustrated in Chapter 4 how managing missing values, handling extreme values, and a proper feature selection lead to better prediction results. We present here the final results after implementing the proper pre-processing, class-imbalance, and dimensionality reduction procedures. To visualise the process of classification via logistic regression, the SEM modelling diagram is presented in Figure 5.1.

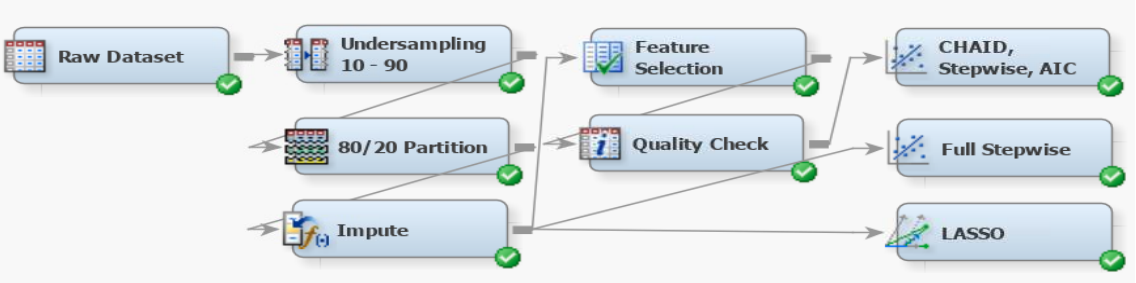


Figure 5.1: Flowchart Logistic Regression Modelling in SEM

After assigning the above settings of the under-sampling ratio to 10%-90%, 80%-20% training-test data partition, and a variable imputation with an 80% missing cutoff value, we start training the different model variants w.r.t. feature selection methods.

First, to find the optimal set of hyperparameters, including the choices for the model selection method, final model selection criterion, and right maximum number of effects, 5-fold cross-validation is applied. SEM does not offer an integrated complete hyperparameters search grid; hence all reasonable combinations are set manually and cross-validated.

We compare the models' hyperparameters and performance of each of the different feature selection procedures. Overall, the different wrapper directions are compared via random 5-fold cross-validation. Table 5.1 represents the resulting cross-validation metrics for each of the considered selection methods with the criteria in brackets referring to the model selection criterion of the respective wrapper technique.

Model Description	ASE	AUC	Gain	C-Lift
CHAID + Stepwise (AIC)	0.0838	0.691	187.89	2.88
CHAID + Forward (CVE)	0.0839	0.691	186.46	2.86
CHAID + Stepwise (CVE)	0.0840	0.684	184.97	2.85
CHAID + Backward (CVE)	0.0842	0.692	190.23	2.90
CHAID + Forward (AIC)	0.0844	0.683	181.04	2.81
CHAID + Backward (AIC)	0.0851	0.673	168.47	2.68

Table 5.1: Logistic Regression Cross-Validation Results

It is visible from Table 5.1 that the stepwise wrapper technique with model selection criterion setting 'AIC' delivers better model performance vs. 'CV-ASE', as it delivers a smaller CV-ASE and higher Cumulative Lift. This could be attributed to SEM's technical only availability of a hold-one-out cross-validation rather than stratified cross-validation, leading to an imbalanced distribution of the rare available churn labels between the datasets. The AIC stepwise selection model is favoured overall, confirming the advantages of combining both forward and backward selection elements. Based on the smallest cross-validation error, the hyperparameter combination of the model 'CHAID + Stepwise (AIC)' is chosen for comparison against other variants of logistic regression techniques on the test dataset. We denote this model by $\mathcal{M}_1$.

$\mathcal{M}_1$, using a combination of CHAID and stepwise selection, fits a model with 30 input variables and 141 model parameters predicting the churn labelling target. We only consider the results of the model in the last step, i.e. step 30 of the stepwise selection procedure for extensive analysis. Statistical significance $\chi^2(df = 135, N = 78790) = 4624.78,\ p < 0.001$ is demonstrated by the full model containing all 30 selected predictors, implying the

model's ability to distinguish between potential churners and non-churners.

To get a deeper understanding of the contributing drivers and identify features with no predictive power, we interpret the significant point estimates for the model's odds ratio. This can be understood as the effect of a one-unit change in the regarded variable on the overall predicted odds, while all the other variables are held constant at their default values.

Significant odd ratios of $\mathcal{M}_1$ to an α-level of 0.001 (see Table 5.2) include the *Churn_3Mon* odd ratio estimate of 3.657 for class 'Yes' vs. 'No'. This implies that clients who churned in the previous 3 months are 3.657 times more likely to churn than clients with *Churn_3Mon* set to class 'No'. The variable *CL_Doc_Reas_A1* with effects 60 and 62 also exhibits high estimated odd ratios. Cases with change reason 60 are 3.464, and reason 62 are 5.222 more likely to churn than cases with reason 70. This is intuitive, considering that reason 60 for changes in the C&L contract stands for cancellation and 62 for pre-cancellation, while reason 70 stand for initiation. Similarly, every additional unit (0.1) in the proportion of the variable *AG_Cont_Prop_A_End_Dat_1Y* (agent's contracts proportion vs. other agents w.r.t. end date within a year) leads to a 77.2% higher chance of churn. If there was no change in the responsible agent of the policyholder, then the policyholder is 42.1% less likely to churn. Similarly, no change in the location contributes to a 50% decreasing effect. An odds ratio estimate of 1 represents a neutral effect on the event likelihood. For example, *Last_Agcy_Active* has no significant increasing or decreasing effect.

Variable	Level	Odds Ratio Estimate
CL_Doc_Reas_A1	62 vs. 70	5.222
Churn_3Mon	Y vs. N	3.657
CL_Doc_Reas_A1	60 vs. 70	3.464
Churn_3Mon_No_CI		1.85
AG_Cont_Prop_A_End_Dat_1Y		1.772
Last_Agcy_Active		1
Resp_OE_Pnr		1
AG_Chg_Resp_OE_Pnr_1Y	0 vs. 1	0.579
Loc_Chng_3Mon	0 vs. 1	0.491
Int_Marketing	Bank vs. other	0.486
HH_Ins_Cond_Year_A1	1981 vs. 2017	0.16
Last_Adm_Sys_Active	IT vs. VK	0.007

Table 5.2: Odds Ratio Estimates - Logistic Regression Model 1

For the overall variable significance, the Wald Statistic and its corresponding significance levels are considered (see Table A.4). If the Wald Statistic is big enough, it will conform with the 0.001 significance level and be considered as an input variable.

To evaluate the model's overall performance and goodness-of-fit, one can consider diverse metrics. As resolved in Section 3.2.5, specific measures can be misleading for highly imbalanced datasets. The choice heavily relies on the predictions of interest and post-prediction intended use. We thus first investigate all key metrics for model comparison and consider only the suitable measures for model selection. Model $\mathcal{M}_1$'s fit statistics show an ASE of 0.0828 and MR of 0.0978 on the training set. The model performs similarly on the test set as it shows an ASE of 0.0831 and MR of 0.0981. The minor difference between the model's errors on training and test dataset shows that the model does not tend to deviate as it works comparably well with previously unseen data.

[%]	Cutoff $t =$		
	0.99	0.50	0.13
Accuracy	90	90.17	78.61
Overall Precision	95	76.14	57.86
Event Precision	100	61.90	22.47
Sensitivity (TPR)	0.05	4.62	46.42
Specificity (TNR)	100	99.68	82.19

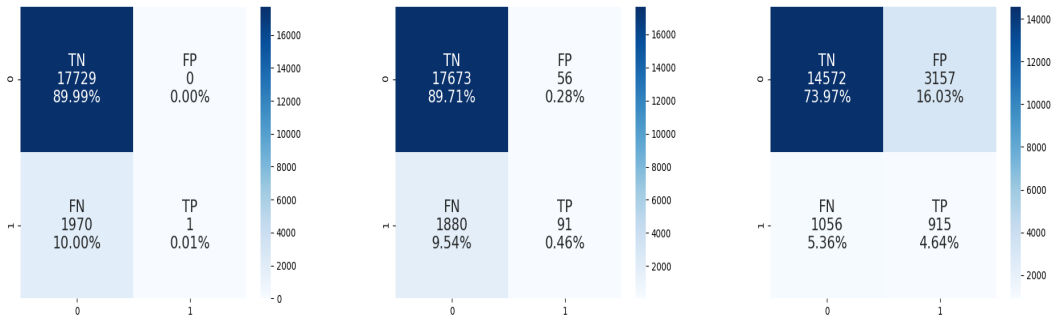
	Cutoff $t =$		
	0.99	0.50	0.13
G-Mean (3.47)	2.25	21.45	61.77
$F_{1.2}$-Score (3.49)	0.09	7.44	32.31
F_1-Score (3.48)	0.10	8.59	30.28
FPR [%]	0	0.32	17.81
FNR [%]	99.95	95.38	53.58

Table 5.3: Classification Statistics - Logistic Regression Model 1

Table 5.3 casts light on the threshold metrics of the test dataset for three alternative cutoffs. While the default cutoff 0.5 is considered by SEM automatically, we consider the cutoff corresponding to the highest $F_{1.2}$-Score (0.13 here) and cutoff with the highest Precision (0.99 here), to give an insight on how the evaluation measures would change with the adjustment of the cutoff. As expected, it is notable how with decreasing cutoffs, a decrease in Event Precision simultaneously with an increase in Sensitivity occurs. Despite an increase in TPR when the cutoff $t = 0.13$ is applied, the Event Precision decrease is of a disadvantage as we require a moderately precise model with a high TPR. In contrast, a Precision and Specificity increase accompanied by a Sensitivity decrease can be observed when the higher cutoff 0.99 is set. Table 5.3 also suggests an Accuracy of 90% for the logistic regression model with stepwise selection. It is however visible from the confusion matrix in Figure 5.2(b) that the TP frequency is 0.46%. This shows that our model selection w.r.t. our research goals should not be based on the Accuracy measure as it is misleading w.r.t. the imbalanced case study dataset.

The confusion matrices in Figure 5.2, based on the test dataset, demonstrate the high imbalance through the heat-map. While TNR shows the highest frequency of right classifications, we observe small TP counts. This suggests that capturing the particular churning clients is in practice challenging with the use of an imbalanced dataset. We will therefore use the ranking metrics independent of the cutoff scores and threshold metrics

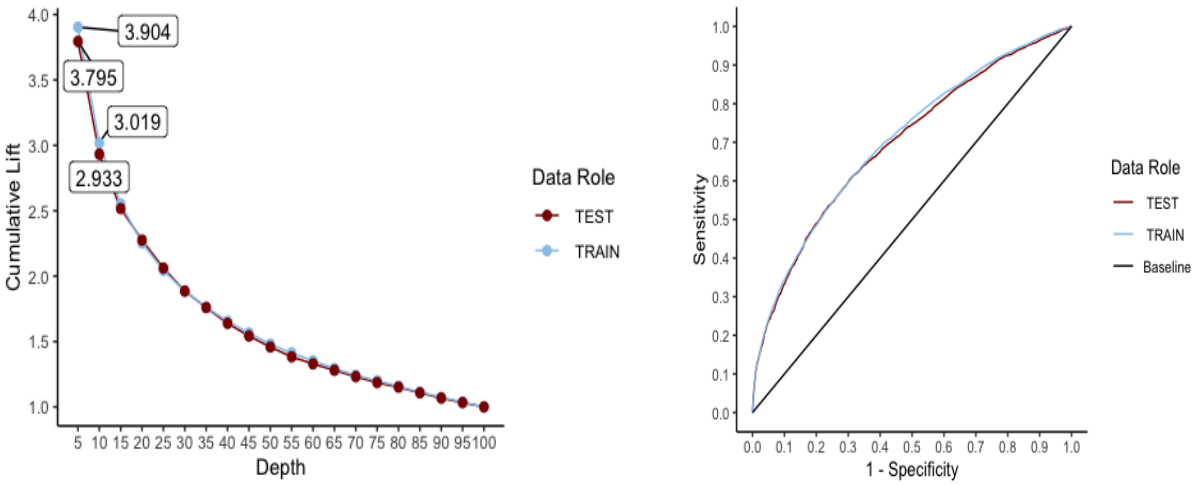
to help identify the best churn prediction model.



(a) Confusion Matrix $t = 0.99$ (b) Confusion Matrix $t = 0.5$ (c) Confusion Matrix $t = 0.13$

Figure 5.2: Confusion Matrices - Logistic Regression Model 1

The AUC value and Cumulative Lift chart are considered essential performance indicators that influence the final model choice. Figure 5.3(a) shows a C-Lift of 3.9 (3.8) for the 5% percentile and 3 (2.9) for the upper decile of the training (test) dataset. This means if the user implements the suggested model, it should be possible to capture 19-19.5% of the at-risk customers for intervention if the clients scored in the first 5% percentile are contacted. Contacting 5-10% suggest by the test scoring is 2.9-3.7 times higher than the rate achieved by contacting a random 5-10% of the underlying observations. Marginally worse model performance and a difference in the percentage captured by 0.09-0.11% is displayed through the test set curve. The model exhibits a good AUC value of 0.705 on the training set and similarly well 0.698 on the test set. Moreover, the small difference between training and test performance is verified through the ROC Index.



(a) Lift Chart Model 1. (b) ROC Chart Model 1.

Figure 5.3: Evaluation Charts - Logistic Regression Model 1

To compare a baseline logistic regression model without prior selection, only the stepwise wrapper is applied, resulting in model $\mathcal{M}_2$ with 30 input variables and 112 model parameters. A relationship between the probability of churn and the combination of 30 independent variables is verified, since the full model's Chi-Square $\chi^2(df = 112, N = 78790) = 4717.13$ indicates a 0.001 level of significance. The null hypothesis was accordingly rejected, and the existence of an association between the independent variables and the churn target is hence supported (see Table A.5).

The model $\mathcal{M}_2$ validates the odds ratio estimates of model $\mathcal{M}_1$, as its estimates are consistent with the level of effects predicted before (see Table 5.4). Comparably, clients who churned in the previous 3 months are 3.68 times more likely to churn than clients with *Churn_3Mon* set to class 'No'. Moreover, a higher unit in the proportion of an agent's contracts vs. other agents increases the churn likelihood by 81%. Cases of the variable *CL_Doc_Reas_A1* with reasons 62 (pre-cancellation) are 5.092 more likely to churn than cases with reason 70. For every additional churned car insurance contract *Churn_3Mon_No_CI*, there is a 90.6% increase in the likelihood of churn. Furthermore, clients with job label 6 (unknown) are 1.353 times more likely to churn than clients with label 'D' (unknown). As $\mathcal{M}_1$ predicts, *Last_Agcy_Active* does not change the odds of response by an amount captured by the model. Likewise, *HH_Ph_State_Key_A1* (policyholder's state key in household contract) with state key 01 (Schleswig-Holstein), does not have a significant effect. With every additional unit in Kgs12_Prop_Hs (kgs12's high school proportion), there is a subsequent 70% decreasing change in the odds.

Variable	Level	Odds Ratio Estimates
CL_Doc_Reas_A1	62 vs. 70	5.092
Churn_3Mon	Y vs. N	3.68
Churn_3Mon_No_CI		1.906
AG_Cont_Prop_A_End_Dat_1Y		1.817
Kgs16_Rentalscore	1 vs. 9	1.357
Job_Pos	6 vs. D	1.353
HH_Ph_State_Key_A1	01 vs. 16	1
Last_Agcy_Active		1
Delta_No_Cat_3Mon		0.733
Job_Pos	A vs. D	0.651
AG_Chg_Resp_OE_Pnr_1Y	0 vs. 1	0.589
Loc_Chng_3Mon	0 vs. 1	0.495
Kgs12_Prop_Hs		0.309
HH_Ins_Cond_Year_A1	1981 vs. 2017	0.105

Table 5.4: Odds Ratio Estimates - Logistic Regression Model 2

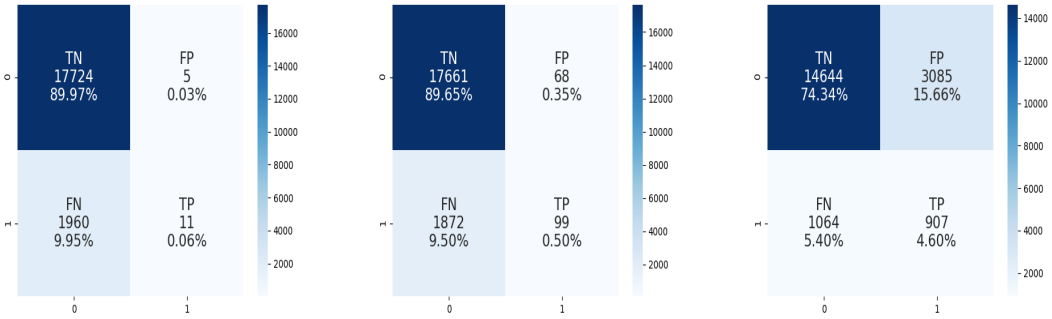
Table 5.5 portrays the threshold metrics for the three alternative cutoffs as defined for $\mathcal{M}_1$. The default cutoff of 0.5 delivers a TPR of 5.02%, which is relatively low compared

to the TNR. Reducing the cutoff value of this model will induce the inverse relationship between decreasing Precision and increasing Sensitivity. To obtain the balanced tradeoff in both measures and general comparability of all models, we remain with the model using the default cutoff value of 0.5. Table 5.5 also suggests an FNR of 94.98% for the logistic regression model with stepwise selection. It is visible from the confusion matrix in Figure 5.4(b) that the TP frequency is 0.5%. This shows that the model selection w.r.t. our research goals can not include the highest TP count.

[%]	Cutoff $t =$		
	0.84	0.50	0.13
Overall Precision	79.40	74.85	57.97
Event Precision	68.75	59.28	22.72
Sensitivity (TPR)	0.56	5.02	46.02
Specificity (TNR)	99.97	99.62	82.60

	Cutoff $t =$		
	0.84	0.50	0.13
G-Mean (3.47)	7.47	22.37	61.65
$F_{1.2}$-Score (3.49)	0.94	8.04	32.40
FPR [%]	0.03	0.38	17.40
FNR [%]	99.44	94.98	53.98

Table 5.5: Classification Statistics - Logistic Regression Model 2



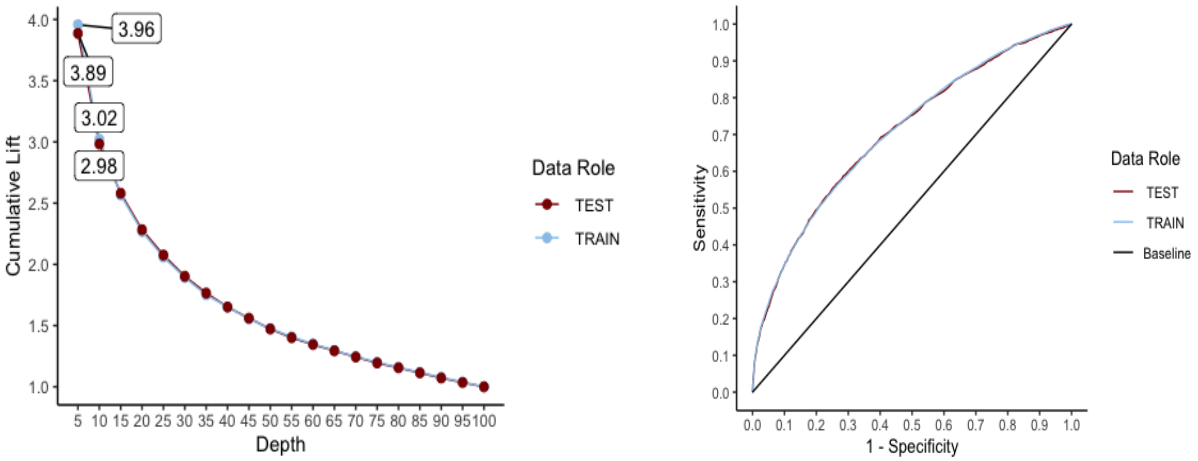
(a) Confusion Matrix $t = 0.84$ (b) Confusion Matrix $t = 0.5$ (c) Confusion Matrix $t = 0.13$

Figure 5.4: Confusion Matrices - Logistic Regression Model 2

The confusion matrix in Figure 5.4(c) demonstrates the highest TP count when the low 0.13 threshold, maximising the $F_{1.2}$-score, is applied. While a high TP frequency of 4.6% is reached, the model suffers from a very low Event Precision of 22%. For the default value we can conclude that $\mathcal{M}_2$ shows higher Sensitivity than $\mathcal{M}_1$, while $\mathcal{M}_1$ shows higher Event Precision.

Figure 5.5(a) shows that $\mathcal{M}_2$ exhibits a Cumulative Lift of 3.96 (3.89) i.e. 19.5-19.8% risk customers are captured in the 5% percentile, which is 0-0.3% higher than $\mathcal{M}_1$. The upper decile shows a Cumulative Lift resulting in a 29.8-30.2% response capture. Using model $\mathcal{M}_2$ will deliver a 3.89-3.96 times higher rate of risk capture than contacting a random 5% sample of the observations, if the 5% percentile of scored clients are contacted. Figure 5.5(b) shows very similar statistics on both training and test datasets implying high

stability and no signs of overfitting. $\mathcal{M}_2$ performs in a more stable manner on the test dataset than $\mathcal{M}_1$. Regarding the ROC chart, the AUC value of $\mathcal{M}_2$ on the training dataset is 0.705 and stays stable as 0.704 on the test dataset.



(a) Lift Chart Model 2.

(b) ROC Chart Model 2.

Figure 5.5: Evaluation Charts - Logistic Regression Model 2

Overall, the classic model $\mathcal{M}_2$ with no prior feature selection tends to be more robust than $\mathcal{M}_1$ as it performs more similar on both the training and test sets. Model $\mathcal{M}_2$ is also able to classify a higher frequency of True Positives than $\mathcal{M}_1$. Based on the higher Recall on the test set, we choose $\mathcal{M}_2$ as a logistic regression model candidate against the machine learning algorithms.

Last but not least, a comparison of the LASSO shrinkage method against the classic logistic regression models is conducted. The LASSO method is selected since it also serves as a further feature selection variant.

Model Description	ASE	AUC	Gain	C-Lift
LASSO (200 Effects)	0.0835	0.70	199.56	2.99
LASSO (300 Effects)	0.0835	0.70	199.56	2.99
LASSO (100 Effects)	0.0836	0.69	197.92	2.97

Table 5.6: LASSO Cross-Validation Results

The possible LASSO combinations include setting the different maximum number of effects 100, 200, or 300 in SEM. The cross-validation error of the model with 200 effects is the lowest between the three possibilities. It is clear from Table 5.6 that the model with an allowed maximum number of 300 effects stops the selection procedure at 200 effects since more effects worsen the model performance. Based on the lowest cross-validation error and simultaneously highest cross-validation Cumulative Lift, we chose the LASSO model, $\mathcal{M}_3$, with 200 maximum effects to compare against all the other models.

Since LASSO aims to shrink all the parameter estimates towards and some equal to zero, $\mathcal{M}_3$'s parameter estimates' range spans between -0.1 and 0.3, a much smaller range than that of the logistic regression model's parameter estimates, see Table 5.7, A.6 and A.7.

Variable	Level	Parameter Estimate
CL_Doc_Reas_A1	62	0.298
Churn_3Mon	Yes	0.227
Churn_3Mon_No_CI		0.098
AG_Cont_Prop_A_End_Dat_1Y		0.060
Last_Agcy_Active		0.001
Delta_No_Cat_3Mon		-0.036
AG_Chg_Resp_OE_Pnr_1Y	0	-0.048
Loc_Chg_3Mon	0	-0.070
Kgs12_Prop_Hs		-0.090

Table 5.7: Parameter Estimates - LASSO Model

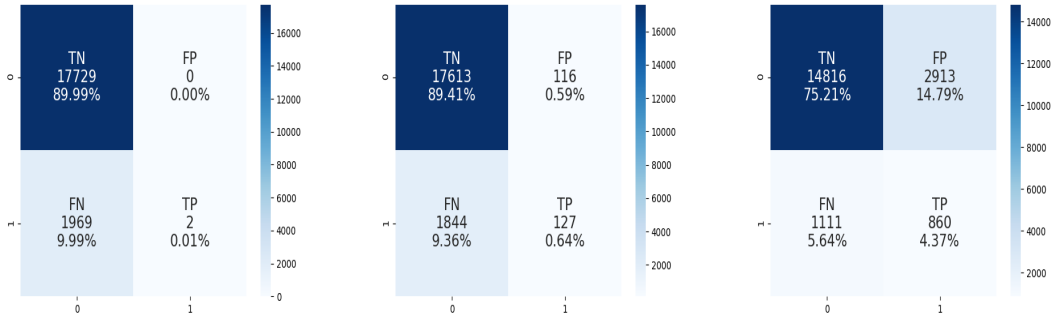
Similar to models $\mathcal{M}_1$ and $\mathcal{M}_2$, the LASSO model predicts the same ranking of variables' effects. For instance, *Churn_3Mon* and *CL_Doc_Reas_A1* have high estimates in contrast to other parameters shrunken very close to zero. The parameter estimate of *Last_Agcy_Active* is shrunk very close to zero, verifying the finding of no substantially changing effect. *Loc_Chg_3Mon* and *AG_Chg_Resp_OE_Pnr_1Y* lead to a decreasing effect on the odds. In line with model $\mathcal{M}_2$, LASSO's parameter estimate of Kgs12_Prop_Hs suggests a decrease in the odds for an increase in kgs12's high school proportion.

Next, the different classification statistics of the model demonstrated by Table 5.8 are elaborated. The LASSO model exhibits a relatively higher TPR of 6.44% than models $\mathcal{M}_1$ and $\mathcal{M}_2$ in the default mode and a fair Event Precision of 52%. For the default cutoff score, the LASSO model is able to classify more positive cases in comparison to $\mathcal{M}_2$, however, accompanied by a lower Event and Overall Precision, while for cutoff value 0.12 $\mathcal{M}_2$ has a higher TPR. This implies that the interpretation dependent on the cutoff values should be universal, i.e. all models compared should have the same cutoff.

[%]	Cutoff $t =$		
	0.99	0.50	0.12
Overall Precision	95	71.39	57.91
Event Precision	100	52	22.79
Sensitivity (TPR)	0.10	6.44	43.63
Specificity (TNR)	100	99.35	83.57

	Cutoff $t =$		
	0.99	0.50	0.12
G-Mean (3.47)	3.19	25.30	60.39
$F_{1.2}$-Score (3.49)	0.17	10.06	31.74
FPR [%]	0	0.65	16.43
FNR [%]	99.90	93.56	56.37

Table 5.8: Classification Statistics - LASSO Model

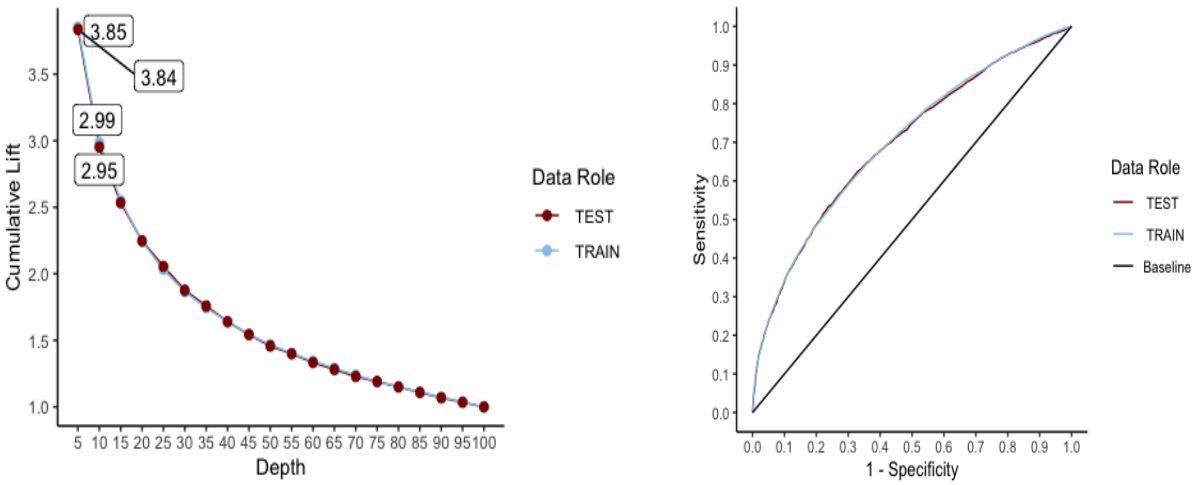


(a) Confusion Matrix $t = 0.99$ (b) Confusion Matrix $t = 0.5$ (c) Confusion Matrix $t = 0.12$

Figure 5.6: Confusion Matrices - LASSO Model

The confusion matrices of the model in Figure 5.6 show very similar frequencies as the other two models. The LASSO model however has the highest True Positives and False Positives count overall for the 0.5 cutoff value. The cutoff value achieving the highest $F_{1.2}$-Score is 0.12 in comparison to 0.13 of the other two models, demonstrating that the model reaches higher TP Counts in the lower predicted probabilities segments.

Figure 5.7(a) displays $\mathcal{M}_3$'s Cumulative Lift of 3.85 (3.84) i.e. 19.2-19.25% risk customers capture in the 5% percentile on the training (test) set, which is 0.2% lower than $\mathcal{M}_2$. Using model $\mathcal{M}_3$ will deliver a 2.95-2.99 times higher rate of risk capture than contacting a random 10% sample of the population if the 10% percentile of scored clients are contacted. The 5-10% percentiles exhibit a C-Lift resulting in a 19.2-29.9% response capture. Overall, the LASSO model delivers a smaller Cumulative Lift than $\mathcal{M}_2$.



(a) Lift Chart Model 3. (b) ROC Chart Model 3.

Figure 5.7: Evaluation Charts - LASSO Model

The ROC chart in Figure 5.7(b) also shows narrowly deviating curves on training and test datasets, implying high stability. $\mathcal{M}_3$ performs in a similarly robust manner on the test set like the $\mathcal{M}_2$ logistic regression model. The ROC curve exhibits a good AUC of 0.7 on the training dataset and a similarly well 0.698 AUC on the test dataset.

To conclude, it has been verified that using different logistic regression variants and shrinkage methods produce similar results in this churn prediction framework. Superior results are, however, perceived for $\mathcal{M}_2$ and $\mathcal{M}_3$ due to their high Sensitivities and Cumulative Lifts. We consider both model suggestions as candidates against the upcoming machine learning methods.

From the outlined model analysis, the following key findings emerged:
Consistent important drivers for household insurance churn prediction are the high relevance of previous churn history and agent-related changes. Moreover, the clients' churn behaviour in other contract types is included in the top indicators. Taking into account the research goals, logistic regression identifies a fair number of clients with cancellation intention. Furthermore, the technique is able to deliver relevant churn drivers and consequent effects, which partly helps create a type of consumer profiling.

5.2 Decision Tree

Next, we describe the results of the tree-based techniques by first illustrating its building blocks, i.e. decision trees.

The integrated decision tree as variable selection pre-step for random forest and gradient boosting can be used as a way of reducing an unnecessarily large number of redundant variables and increase model performance. The information-rich inputs can then be further evaluated by the tree-based modelling nodes. By considering the three variants, CART, CHAID tree, and Entropy as a node split criterion, we carry out a cross-validation procedure to fine-tune the hyperparameters of the decision tree and discuss the resulting facts below.

Model Description	ASE	AUC	Gain	C-Lift
CHAID Tree	0.0835	0.677	186.48	2.86
Entropy	0.0842	0.625	158.84	2.59
CART	0.0857	0.59	126.88	2.27

Table 5.9: Decision Tree Cross-Validation Results

Table 5.9 demonstrates that the CHAID Tree performs best overall as it delivers the

smallest CV-ASE and highest AUC and Cumulative Lift. CART is the weakest model between all three with a higher CV-ASE and lowest AUC and Cumulative Lift. Based on the smallest CV-ASE and overall best performance, we apply the CHAID Tree Settings in the SEM variable selection step before training the random forest and gradient boosting models.

In SEM, we set the nominal target criterion to 'ProbChisq' to build a CHAID tree. The inputs can be used several times at different nodes with only binary splits allowed, and a maximum depth of 6 can be achieved by the tree. Finally, the large full tree is chosen with cross-validated subtrees. The CHAID method yields a tree with the following classification statistics:

Statistic [%]	Train	Test
Overall Precision	82.31	70.98
Event Precision	73.17	51.02
Sensitivity (TPR)	16.48	11.42
Specificity (TNR)	99.33	98.78

Table 5.10: Classification Statistics - Decision Tree

A subtree of the full tree, included as an illustration on the next page (see Figure 5.8), shows the high importance of the variable *Churn_Last_No_Mon* (number of months since last churn) as it is already used twice for node splitting. While the top ten logistic regression features include churn related inputs, *Churn_Last_No_Mon* is not included in the tree's top variables. In accordance with the previous logistic regression models, observations with no change of responsible agent, indicated by the binary variable *AG_Chg_Resp_OE_Pnr_1Y* (change of responsible agent in the past year), lead to fewer observations with churn labels (7.82% vs. 12.99%) of the respective nodes' samples. The tree also suggests that clients younger than 73 years generate internal nodes with a higher churn label frequency than older clients.

Table 5.10 demonstrates that the tree's Sensitivity is in line with our research goal and with previous classifiers' performance. Considering that the tree is able to identify a relatively good number of potential churners, we consider the provided features' subset a good foundation for the tree-based models. However, the tendency of decision tree to overfit is underlined by the big discrepancy between training and test values. The tree's Precision is compromised while still providing a high number of correct event classifications proven by the high TPR. Overall, the decision tree exhibits similarities with logistic regression and is observed to be a reasonable feature selection base for random forest and gradient boosting.

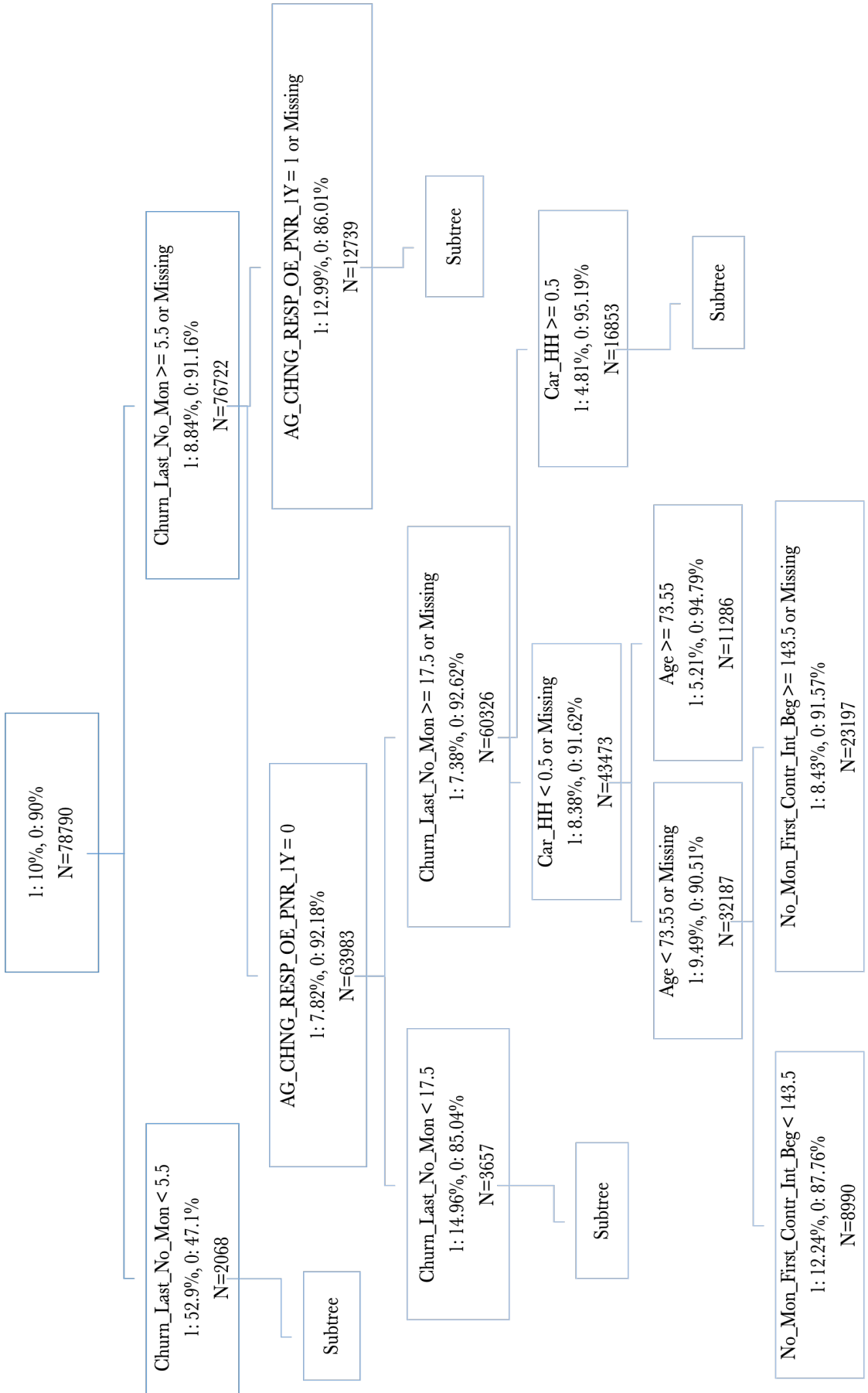


Figure 5.8: Decision Tree Structure - Churn Prediction Example

5.3 Random Forest

Considering the provided illustration of how the single trees in a churn-predicting random forest could look like, we train the random forest model on the chosen variables and allow the base decision trees to prioritise the important associated variables (see Flowchart A.2).

First, the extensive grid of hyperparameters requires a proper fine-tuning to achieve better forest performance and avoid potential overfitting by conducting an Out-of-Bag (OOB) assessment. Since SEM does not support cross-validation on the random forest node, the Out-of-Bag errors are compared as an alternative to the CV-ASE. For the hyperparameter tuning, we compare different combinations of numbers of trees, maximum leaf sizes, numbers of variables per split, and maximum tree depths. Based on the Out-of-Bag validation metrics, we choose two of the best performing models and consider them in the final comparison for the random forest candidate model.

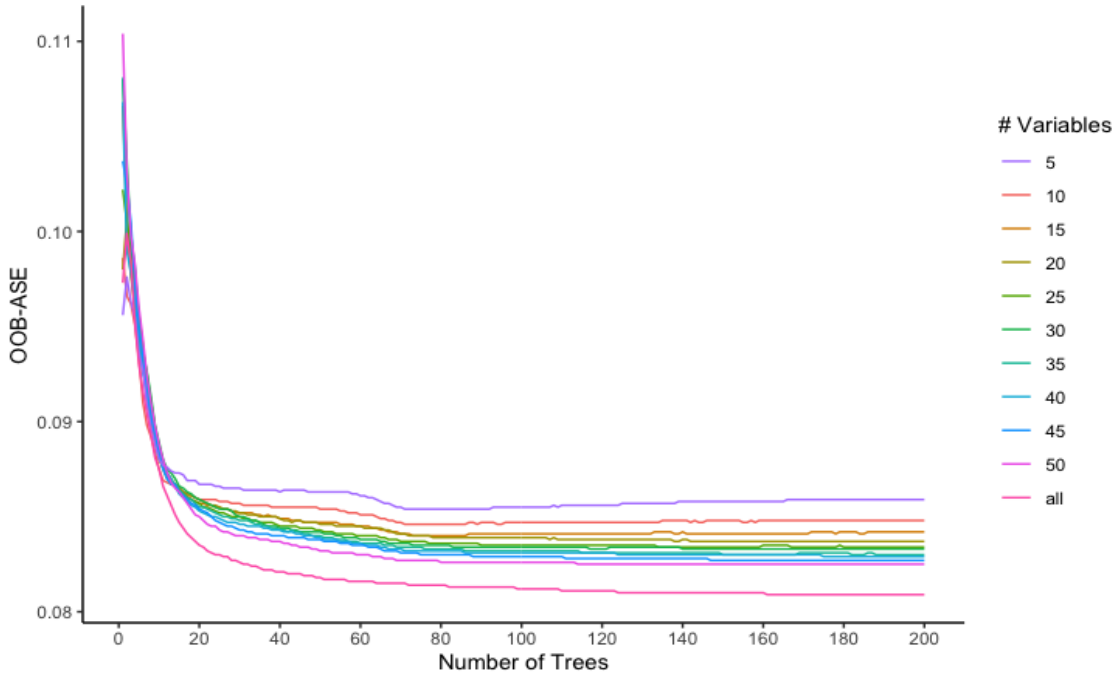


Figure 5.9: Random Forest Out-of-Bag ASE Development

Figure 5.9 portrays the development of the OOB average squared error (ASE) on the y-axis versus number of trees on the x-axis for each considered number of variables used for splitting the tree nodes. It is visible how an increase in the number of considered variables reduces the Out-of-bag ASE. Regarding all variables at each split considerably reduces the ASE, however, with an accompanying risk of overfitting the model, longer computation time, and increased complexity. Starting from 100-150 forest trees, there is only an incremental reduction in ASE visible, which makes the final performance indifferent to the number of trees.

# Variables	# Trees	# Leaves	OOB ASE	OOB MR	OOB Log-Loss
All	100	74961	0.081	0.095	0.292
50	150	171768	0.082	0.099	0.295
40	130	151988	0.083	0.099	0.297
25	200	255716	0.083	0.099	0.298

Table 5.11: Out-of-Bag Statistics - Random Forest Models

Table 5.11 demonstrates that using a higher number of trees with a smaller number of variables can deliver similar results compared to trees with a large number of variables. Choosing a large number of variables and trees can, however, also lead to an overfitted model. This is suggested by the hyperparameter tuning as the biggest difference between training ASE and OOB ASE is exhibited by the model with all variables. Based on the resulting out-of-bag errors and settings, we investigate the random forest with 150 trees, 50 variables denoted by $\mathcal{R}_1$ vs. 130 trees, 40 variables ($\mathcal{R}_2$), and an untuned forest ($\mathcal{R}_3$).

To compare the final chosen models, we first clarify the respective settings in SEM and characteristics of the random forest. Untuned forests with the default settings in SEM are set to only try 14 variables for node splitting, build the forest using 100 trees, training the forest on a 0.6 proportion of the training dataset, and using the rest for Out-of-Bag assessment. Furthermore, a maximum depth of 50 is considered, and the node splitting criterion is set to the Gini criterion. In comparison, we set the training proportions for $\mathcal{R}_1$ and $\mathcal{R}_2$ to 0.8 and allow only a 0.2 proportion of the training dataset to be used for Out-of-Bag assessment. This is done to avoid a higher imbalanced distribution of the target levels and to obtain the same validation proportion as in the 5-fold cross-validation. Moreover, $\mathcal{R}_1$ can choose out of 50 variables, the Gini criterion for node splitting uses 150 trees and has a maximum depth of 6. Similarly, $\mathcal{R}_2$ can choose from 40 Variables, the Gini criterion for node splitting uses 130 trees and has a maximum depth of 6.

The first chart in Figure 5.10 shows the Cumulative Lift curves of all three models on the training set. In the 5% percentile, the untuned forest $\mathcal{R}_3$ shows the highest Cumulative Lift of 5.11 vs. a Cumulative Lift of 4.68 for $\mathcal{R}_1$ and slightly lower 4.66 for $\mathcal{R}_2$. For the upper decile, all models decrease substantially, demonstrating the low frequency of the target observations as the datasets only include less than 10% churn labeled instances. The second chart shows the models' performance on the test set. The same models' ranking pattern of the training set chart prevails on the test set which implies some findings. The untuned default forest yields a considerably lower Cumulative Lift than on the training set, indicating a high discrepancy between in and out-of-sample performance despite being the best predictive forest. This highlights the underlying overfitting and the model's unstable generalisation on unseen data. $\mathcal{R}_1$ and $\mathcal{R}_2$ are comparable to their values on the training

set, verifying that increasing the depth of the forest can lead to overfitting.

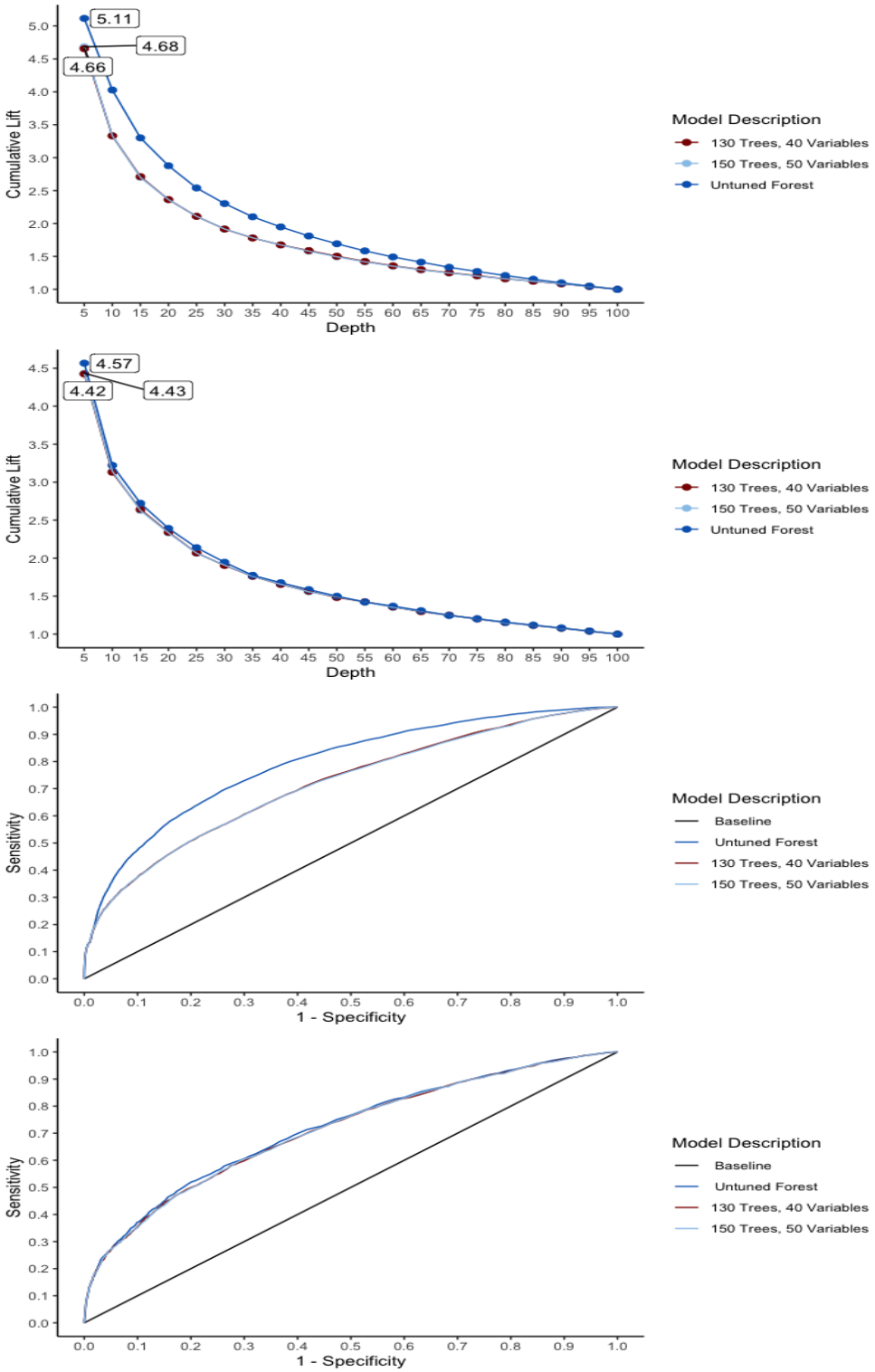


Figure 5.10: Random Forest Models' Evaluation Charts

$\mathcal{R}_1$ performs better overall than $\mathcal{R}_2$ as it shows higher Cumulative Lift on both the 5% and 10% percentiles. Overall these findings are in accordance with findings visible by the

ROC charts on training and test sets. Moreover, $\mathcal{R}_3$'s discrepancy between in-sample and out-of-sample performance is visible by the high training ROC curve vs. a less steep curve on the test set. Generally, all three ROC curves show moderate model performance on the test set and deliver AUC values between 0.7-0.72, with $\mathcal{R}_3$ performing best.

[%]	$\mathcal{R}_1$	$\mathcal{R}_2$	$\mathcal{R}_3$
Accuracy	90.44	**90.45**	90.41
Overall Precision	77.36	77.52	**79.41**
Event Precision	63.84	64.15	**68.16**
Sensitivity (TPR)	10.30	**10.35**	7.71
Specificity (TNR)	99.35	99.36	**99.60**
Cumulative Lift (5%)	4.43	4.42	**4.57**

	$\mathcal{R}_1$	$\mathcal{R}_2$	$\mathcal{R}_3$
G-Mean (3.47)	31.99	**32.07**	27.71
F_1-Score (3.48)	17.74	**17.82**	13.86
$F_{1.2}$-Score (3.49)	15.69	**15.77**	12.12
FPR [%]	0.65	0.64	**0.40**
FNR [%]	89.70	**89.65**	92.29
AUC	0.71	0.71	**0.72**

Table 5.12: Classification Statistics - Random Forest Models

For an extensive evaluation, Table 5.12 compares the test set evaluation measures of the random forest methods with tuned hyperparameters versus the default forest settings suggested by SEM. While the untuned $\mathcal{R}_3$ exhibits overall the highest Event Precision, $\mathcal{R}_1$ and $\mathcal{R}_2$ show considerably higher Sensitivity, especially $\mathcal{R}_2$'s relatively high 10.35% Sensitivity w.r.t. the underlying class imbalance. Model $\mathcal{R}_2$, built using 130 trees and 40 variables demonstrates the best balance between Precision and Recall, as it exhibits the highest F-scores. The adjusted $F_{1.2}$-Score, weighing Recall by just 0.2 times more than Precision, favours $\mathcal{R}_2$ with the highest score of 15.77, a relatively higher score than $\mathcal{R}_3$'s 12.12. One can also note that $\mathcal{R}_3$ shows the highest Specificity. In the underlying case study, False Negatives are more costly than False Positives; hence, models with a smaller FNR are preferably selected, for instance model $\mathcal{R}_2$ with the overall smallest FNR of 89.65%. It must be pointed out that models with the highest Cumulative Lift, such as model $\mathcal{R}_3$ here, do not guarantee the highest Recall rate. While Recall measures the correct event classifications between all event classifications, Cumulative Lift measures the increased event capture only for a targeted subset, e.g. the top proportion of the predicted probabilities ranked list. The statistics findings in Table 5.12 are directly in line with previous findings by the Cumulative Lift and ROC Chart regarding $\mathcal{R}_3$'s overfitting potential, as the test shows a similar performance compared to the other models in contrast to a much better-indicated assessment on the training set. Although the models deliver a seemingly high 90% Accuracy, we obtain in a similar pattern as logistic regression models very high 90% FNR. In the context of class imbalance, Accuracy remains a misleading evaluation metric for tree-based techniques.

Based on all of the above results and overview of different model properties, we select $\mathcal{R}_2$ as a final candidate for the random forest model. $\mathcal{R}_2$ provides the highest Recall along

with an adequate balance of Event Precision proven by the high F-scores.

The final chosen model $\mathcal{R}_2$ has the following variable importance structure:

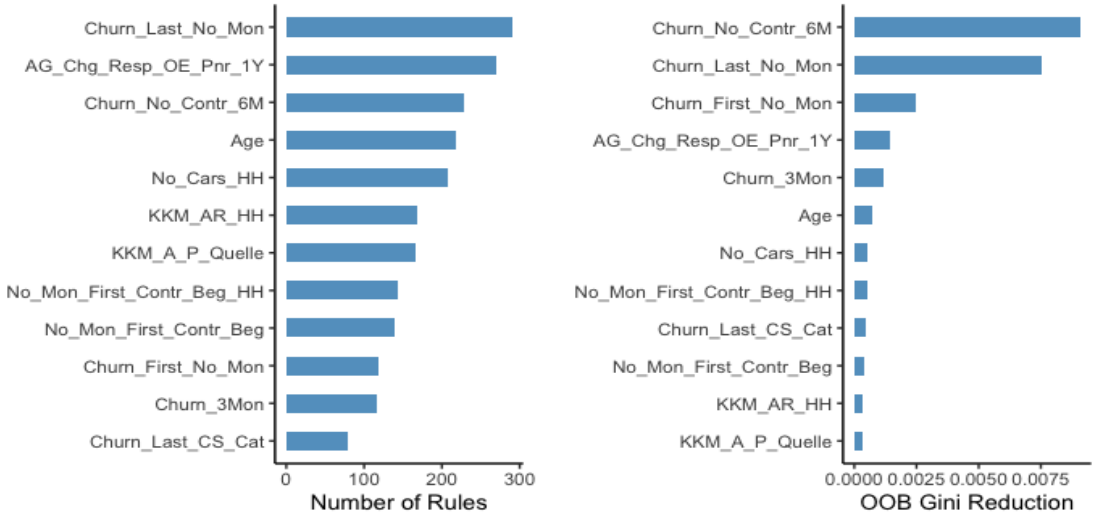
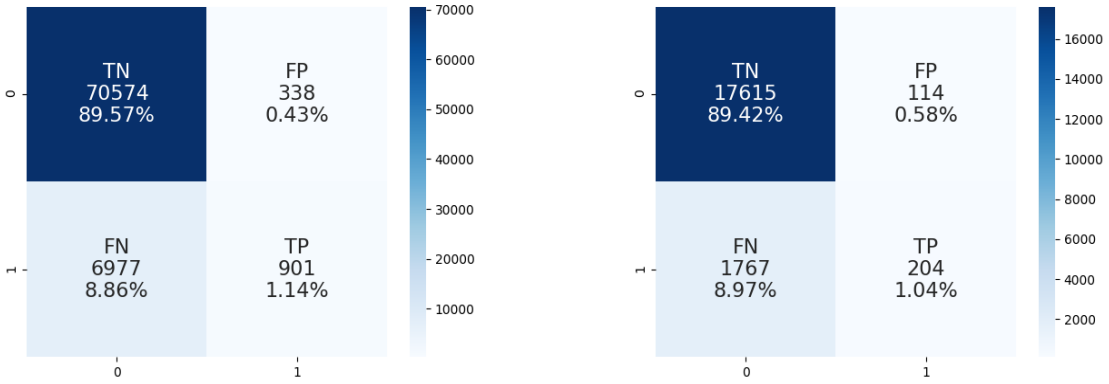


Figure 5.11: Random Forest Model $\mathcal{R}_2$ - Variable Importance

The right panel of Figure 5.11 displays the mean decrease in Gini, which resembles the mean of a variable's total influence on a decrease in node impurity, weighted by the samples arriving at that node in each of the forest's tree. The higher the mean decrease in Gini, the more important the variable. In accordance with the logistic regression models' feature importance, the random forest model tends to highly weigh some of LR's top-ranked variables. These are for example churn and agent behaviour related variables such as *Churn_3Mon*, *Churn_No_Contr_6Mon* (number of churned contracts in the past 6 months), *Churn_First_No_Mon* (number of months since first churn) and *AG_Chg_Resp_OE_Pnr_1Y*. The number of churned contracts in the past 6 months has the greatest Gini reduction influence and is, therefore, of high importance. Based on the number of times the variable was used for splitting, depicted in the left panel of Figure 5.11, we can also draw conclusions about the importance. The variables *Churn_No_Contr_6Mon* and *Churn_First_No_Mon* simultaneously show high Gini reduction and high number of splits. The association is highlighted as *Churn_First_No_Mon*, measuring the number of months since the last churn is used for 334 rules in the entire forest. Similarly, the variables *AG_Chg_Resp_OE_Pnr_1Y* and *Age* are used for 200-250 splits.

The train confusion matrix of $\mathcal{R}_2$ shows a True Positives count of 901, i.e. 1.14% of the classified observations by the random forest classifier (see Figure 5.12(a)), relatively high w.r.t. the analysed logistic regression models. Consequently, the False Negatives frequency of 8.96% is overall lower showing a less Negatives classifications oriented distribution in the heat-maps than LR. One can notice vice versa that the model is able to properly



(a) Train Confusion Matrix.

(b) Test Confusion Matrix.

Figure 5.12: Random Forest Model $\mathcal{R}_2$ - Confusion Matrix

classify TN, leading to a low frequency of 338 FP, 0.43% of the total classifications. The test confusion matrix shows similar total percentages for each component as all frequency proportions change by less than 0.16% (see Figure 5.12(b)). This is also indicated by the small 1% difference in train and test Sensitivity as visible in Table 5.13.

To further analyse the confusion matrix indications, we review the resulting evaluation metrics of $\mathcal{R}_2$ given by Table 5.13.

	AUC	C-Lift(5%)	Event Precision	TPR	FNR	FPR	$F_{1.2}$-Score
Train	0.72	4.66	72.72%	11.44%	88.56%	0.48%	17.47
Test	0.71	4.42	64.15%	10.35%	89.65%	0.64%	15.77

Table 5.13: Classification Statistics - Random Forest Model $\mathcal{R}_2$

Table 5.13 summarises the research-goal relevant classification measures of model $\mathcal{R}_2$ on both the training and test sets. $\mathcal{R}_2$ shows a high Event Precision on the training set of almost 73%, and in comparison, only 64% on the test dataset. This can be attributed to the smaller number of available cases in the test vs. training set, not allowing the forest to obtain a comparable event accuracy. The ROC index hardly changes, implying that the Sensitivity levels remain stable on out-of-sample observations. Furthermore, we analyse the relevant ranking metrics for the imbalanced dataset on hand. Model $\mathcal{R}_2$ exhibits a test Cumulative Lift of 4.42 in the upper 5% percentile (see Figure 5.10), i.e. 22.1% risk customers are captured, which is a relatively high Cumulative Lift. The upper 10% percentile shows a Cumulative Lift resulting in a 31-33% response capture. Using model $\mathcal{R}_2$ will deliver a 3.1-4.6 times higher rate of risk capture than contacting a random 5-10%

sample of the observations if the clients in the 5-10% percentile are contacted. The ROC charts in Figure 5.10 also show very similar curves on both sets implying high robustness and no signs of potentially disadvantageous overfitting.

In conclusion, the random forest modelling results display the potential differences that can arise when the algorithm's hyperparameters are adjusted. From the short review above, we can confirm that an increase in maximum depth generates a higher model Precision. However, high maximum depth, along with a low training set proportion, can negatively affect the model's generalisability and robustness. These essential findings are consistent with previous studies (Probst, Boulesteix, and Bischl (2019)). The results lead to similar conclusions as logistic regression, namely that important influencing features include churn-history and contract-related characteristics. It is important to note that some of the top-ranked variables by the random forest model, explicitly *KKM_AR_HH* and *KKM_A_P_Quelle* were included with no provided elaboration on the definition by ERGO. As the variables show relatively high importance in the tree-based methods, it is considered valuable to add the variables' definitions.
On the one hand, considerably superior results are achieved with random forest models compared to logistic regression with respect to Recall and Cumulative Lift. On the other hand, random forest models do not provide the possible odd ratio estimates interpretation to extract valuable churn drivers and effectively tackle them, e.g. through marketing measures, by understanding the effect direction. Nonetheless, it is well justified to use random forest models for high and precise targeted churn detection.

5.4 Gradient Boosting

Moving on to the last tree-based method, gradient boosting (see Flowchart A.3), we start by giving a short review of the cross-validation results with the objective of hyperparameter tuning and model optimisation.

Model Description	ASE	AUC	Gain	C-Lift
100 Itr. GB	0.0810	0.720	218.48	3.18
GB Default	0.0819	0.703	208.83	3.09
CHAID + GB	0.0821	0.699	204.27	3.04
Entropy + GB	0.0824	0.687	193.73	2.94

Table 5.14: Gradient Boosting Cross-Validation Results

In Table 5.14 the four top models of the gradient boosting parameter combinations are listed with their respective cross-validation statistics. The first model '100 Itr. GB',

denoted by $\mathcal{G}_1$, is a fine-tuned gradient boosting model. The model uses an increased number of 100 iterations and a decreased shrinkage parameter of 0.05 controlling the learning rate to offset potential overfitting due to the increased iterations number. $\mathcal{G}_1$'s decision tree parameters include a maximum depth of 4, a leaf fraction of 0.005, and a variable can be used by two splitting rules in the same path. We denote the default gradient boosting model as $\mathcal{G}_2$. The untuned default gradient boosting model in SEM has, in comparison, only 50 iterations, a shrinkage parameter of 0.1, maximum depth of 2, leaf fraction of 0.001, and variables can not be reused. The tuned gradient boosting model performs overall better on the cross-validation set as it shows a smaller cross-validated ASE, higher AUC of 0.72, and a higher Cumulative Lift than the default model. To compare classic gradient boosting models, we compare the model, including features pre-selection via a decision tree. We apply the best performing decision trees according to Table 5.9; these are the CHAID tree and the tree based on Entropy as a splitting criterion. As clear from Table 5.14 the classic gradient boosting models outperform both models with an extra feature selection as their cross-validated errors are slightly higher and Cumulative Lifts are lower. Based on the above observed lower CV-ASE, we compare the best two performing models on the cross-validation set, $\mathcal{G}_1$ and $\mathcal{G}_2$, for illustration and analysis of relative performance.

Figure 5.13 displays the relative variable importance of model $\mathcal{G}_1$, which resembles the variables' total contribution to the change in the residual sum of squares error values. The higher the reduction in error value, the higher the variable's relative importance measure. In comparison with the previous models' feature importance, the gradient boosting model highly weighs some different variables as the tree's top important variables. These are for example, rather the basic client-related variables such as *Age*, *Geo_Cluster_kgs16* and CL_Ph_State_Key_A1. The variables *Churn_Last_No_Mon* and *AG_Chg_Resp_OE_Pnr_1Y* remain highly ranked in accordance with the previous models.

Likewise, Figure 5.14 demonstrates the relative variable importance of model $\mathcal{G}_2$. In comparison with $\mathcal{G}_1$'s feature importance bar chart, the default gradient boosting model has a slightly different variable ranking. Whilst $\mathcal{G}_1$ only includes *Churn_Last_No_Mon* in the top ranks, $\mathcal{G}_2$ considers more variables indicating churn behaviour history is of the highest importance. We can justify $\mathcal{G}_2$'s reduced number of overall splitting rules with the smaller allowed maximum tree depth. It is important to note that some of the top-ranked variables by both gradient boosting models, explicitly *KKM_AR_Kunde* and *KKM_A_P_Quelle* were included in the random forest model with no provided elaboration on the definition. Generally, one can observe consistency with previous models implying association between the target and variables in churn history or agent related categories.

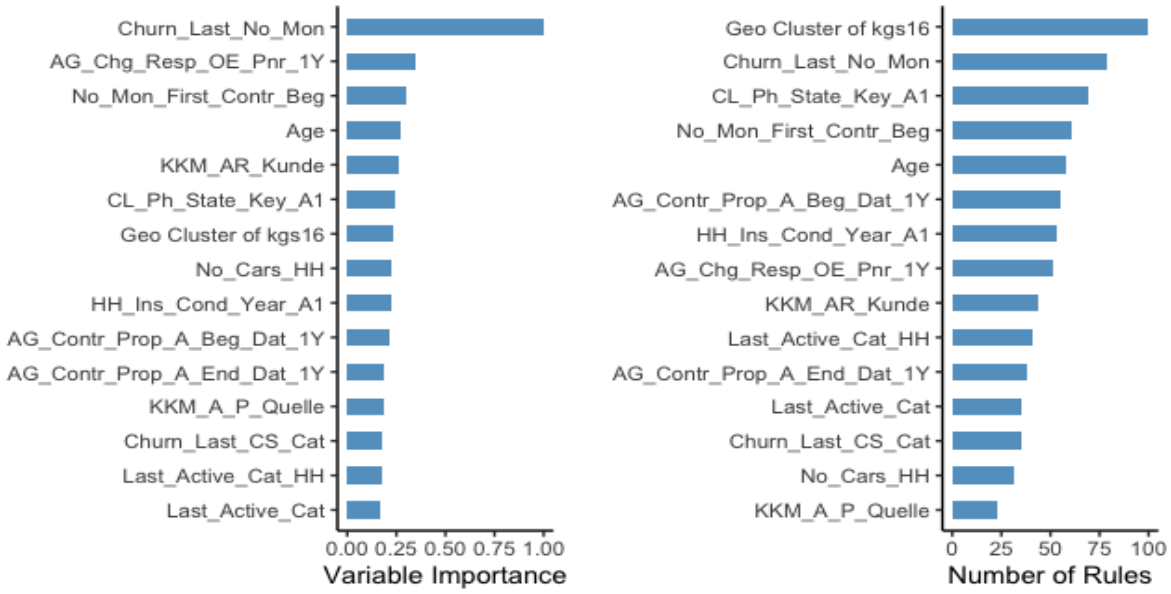


Figure 5.13: Gradient Boosting Model $\mathcal{G}_1$ - Variable Importance

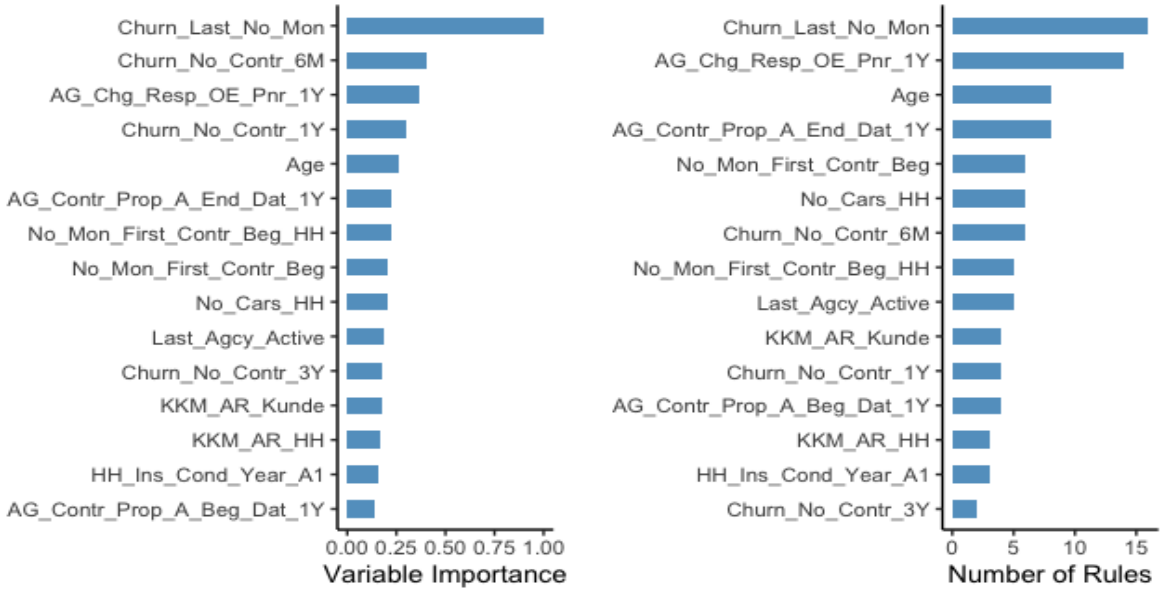
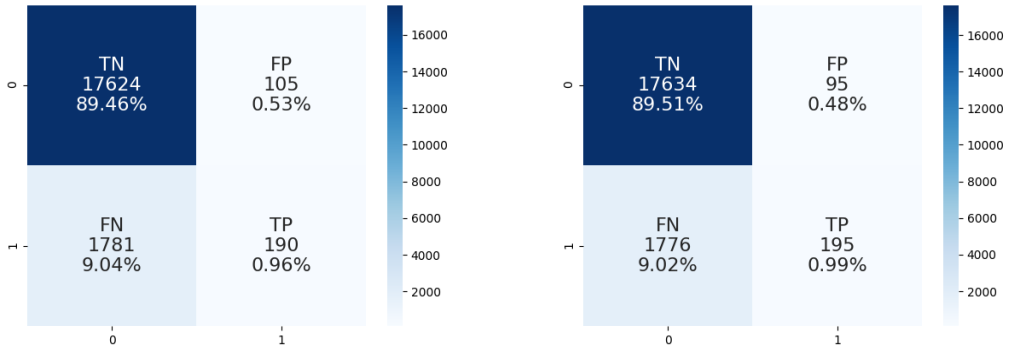


Figure 5.14: Gradient Boosting Model $\mathcal{G}_2$ - Variable Importance

Next, we analyse the classification frequencies of the models on the test set. The confusion matrix of $\mathcal{G}_1$ w.r.t. the default 0.5 cutoff value shows a moderately high True Positives count of 190, 0.96% of the classified observations by the model. $\mathcal{G}_2$ is able to correctly classify a slightly higher count of 195 True Positives, making up roughly 1% of model classifications. As a consequence, the $\mathcal{G}_1$'s False Negatives frequency of 1781 is lower vs. the 1776 count of $\mathcal{G}_2$. One can additionally observe that $\mathcal{G}_2$ is able to slightly classify more True Negatives than $\mathcal{G}_1$, leading to a lower count of 95 False Positives, 0.48% of the



(a) Confusion Matrix - $\mathcal{G}_1$.

(b) Confusion Matrix - $\mathcal{G}_2$.

Figure 5.15: Gradient Boosting Models - Confusion Matrices

total classified sample vs. 105, 0.53% of $\mathcal{G}_1$'s classifications. The heat-maps' colours show overall the same orientation of the classifications distribution.

To support the findings obtained by the confusion matrices, we consider the test evaluation metrics' values for both models displayed in Table 5.15. The table reveals almost overall superior results for model $\mathcal{G}_2$ on the test set. $\mathcal{G}_1$ outperforms only w.r.t. showing higher Cumulative Lift and ROC Index. This highlights that the model with a higher Cumulative Lift does not necessarily deliver the highest TP frequency nor Precision. In line with the confusion matrices, $\mathcal{G}_2$ exhibits almost a 10% Recall rate, slightly lower than random forest models, and twice as high as logistic regression. The Event Precision is moderate as it is lower than 70%, however in consistency with previous classifiers' performance. In conclusion, both models perform fairly with minor percentage differences.

	$\mathcal{G}_1$	$\mathcal{G}_2$
Accuracy [%]	90.43	**90.50**
Overall Precision [%]	77.61	**79.05**
Event Precision [%]	64.41	**67.24**
Sensitivity (TPR) [%]	9.64	**9.89**
Specificity (TNR) [%]	99.41	**99.46**
Cumulative Lift (5%)	**4.65**	4.25

	$\mathcal{G}_1$	$\mathcal{G}_2$
G-Mean (3.47)	30.96	**31.37**
F_1-Score (3.48)	16.77	**17.25**
$F_{1.2}$-Score (3.49)	14.80	**15.21**
FPR [%]	0.59	**0.54**
FNR [%]	90.36	**90.11**
AUC	**0.72**	0.71

Table 5.15: Classification Statistics - Gradient Boosting Models

The chart in Figure 5.16(a) shows the Cumulative Lift curves of the model $\mathcal{G}_1$ on both the training and test set. In the 5% percentile, the tuned gradient boosting model shows a high Cumulative Lift of 4.65 on the training set vs. the same Cumulative Lift of 4.65 on the test set. For the 10% decile, the model's Cumulative Lifts decrease substantially, demonstrating the underlying low frequency of the target observations as the datasets only include less than 10% churn labeled instances. On the 10% decile, $\mathcal{G}_1$ exhibits a trained Cumulative Lift of 3.45 and test Cumulative Lift of 3.27. The second chart 5.16(b) shows the models' ROC curve on the training and test sets. The same robustness pattern of the chart on the sets prevails. The tuned gradient boosting yields a marginally lower train ROC curve than on the test set, indicating some discrepancy between in and out-of-sample performance. This highlights the underlying overfitting for gradient boosting models with a high iterations number.

Analogously, Figure 5.17(a) displays the Cumulative Lift curves of the model $\mathcal{G}_2$ on the training vs. test set. The default gradient boosting shows a marginally higher Cumulative Lift on the training set equal to 4.31 vs. 4.25 on the test set. We can, therefore, conclude that using model $\mathcal{G}_2$ and contacting the clients in the 5% percentile, we should be able to capture 21.25-21.55% more at risk-customers than random-targeting which is an effective result in practice. In comparison to $\mathcal{G}_1$, a small deviation is visible on both the Cumulative Lift and ROC charts. In the upper retrieval area (5%-10%) we also note a sharper decrease in Lift for $\mathcal{G}_1$. The default model, however, generalises in both charts poorer than $\mathcal{G}_1$, supported by the lower C-Lift and AUC values of Table 5.15.

Based on the above review of different model properties, $\mathcal{G}_2$ is selected as a final candidate for the gradient boosting model. $\mathcal{G}_2$ provides the highest Recall and Precision between the considered models. The model, however, does not optimally reach the highest Cumulative Lift nor the highest AUC value. Regarding the $\mathcal{G}_2$'s metrics' discrepancies between test and train sets, visible in Table 5.16, implies strong model robustness and generalisability. It should be mentioned that the performance on the test set is partially better than on the training set demonstrated by measures like Precision and FPR. One possible explanation of this random occurence could be the features' different distribuation between test and training set.

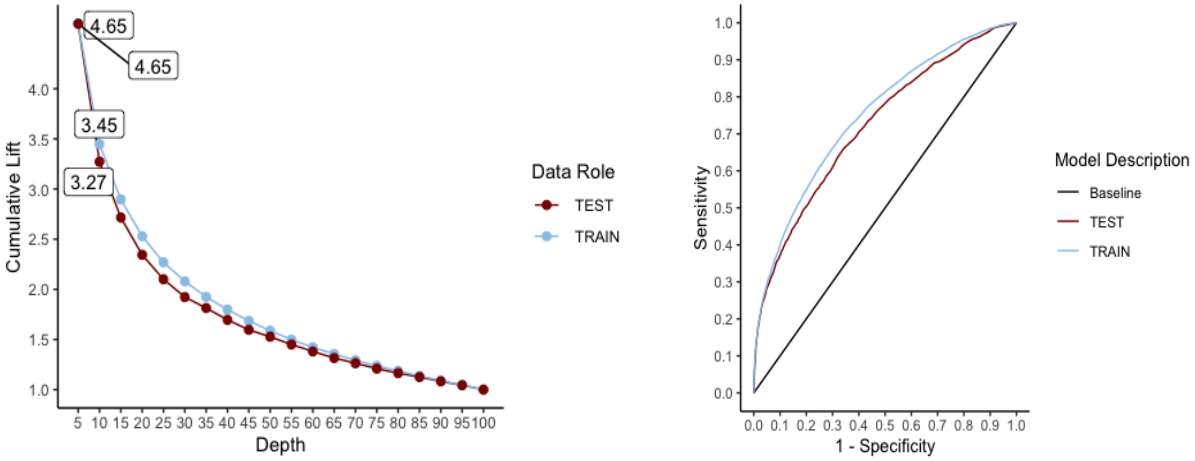
The gradient boosting technique findings can be summarised as follows:

The investigated versions of gradient boosting models are able to identify a good proportion of customers with cancellation intention along with good accuracy. Moreover, gradient boosting models with a moderate iterations number relative to the target frequency and dataset size perform in a stable manner on previously unseen data. Increasing the iterations number can increase model performance but confirms the necessity of defining an adequate bias-variance trade-off. Similar performance as other tree-based methods such as random forests, decision trees and superior results over

classic models i.e., logistic regression, are observed. Finally, identified representative variables include the number of months since the last churn, change of the responsible agent in the last year, age and client-company relationship recency implied by No_Mon_First_Contr_Beg (number of months since the start of the first contract).

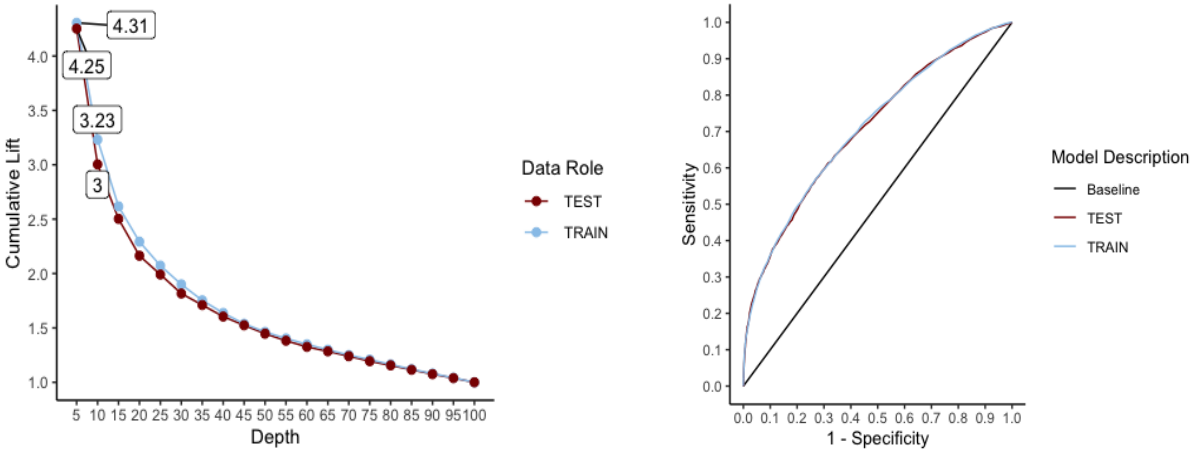
	AUC	Overall Precision	Event Precision	TPR	FNR	FPR
$\mathcal{G}_2$ Train	0.71	78.82 %	66.78%	10.05 %	89.95 %	0.56%
$\mathcal{G}_2$ Test	0.71	79.05 %	67.24%	9.89 %	90.11 %	0.54%

Table 5.16: Classification Statistics - Gradient Boosting Model $\mathcal{G}_2$



(a) Lift Chart - $\mathcal{G}_1$. (b) ROC Chart - $\mathcal{G}_1$.

Figure 5.16: GB Model $\mathcal{G}_1$ - Evaluation Charts



(a) Lift Chart - $\mathcal{G}_2$. (b) ROC Chart - $\mathcal{G}_2$.

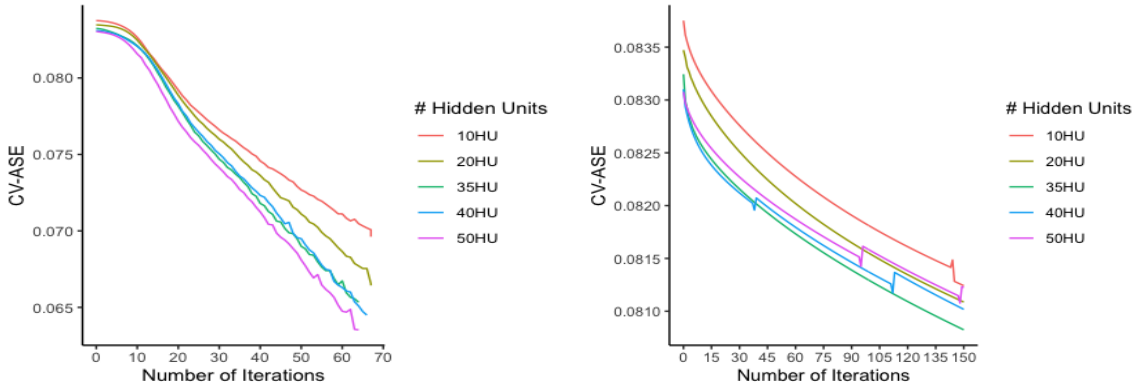
Figure 5.17: GB Model $\mathcal{G}_2$ - Evaluation Charts

5.5 Neural Networks

Finally, we implement a further family of machine learning algorithms, namely neural networks. Neural networks' hyperparameter tuning is commonly known to be a tedious task that can not be efficiently performed by using a complete grid search (see Hastie, Tibshirani, and Friedman (2001), p. 397). Therefore, we consider some of the important tuning properties and validate the candidate choices through 5-fold cross-validation. Considering the SEM remarks in Appendix A.2.3 and the available tuning options, the user can primarily set the number of hidden units (HU)/neurons, the number of hidden layers, and the total number of iterations (IT). Increasing the NN size should not hurt the performance (see Hastie, Tibshirani, and Friedman (2001), p. 400), so varying the number of neurons, and the number of iterations will be the main hyperparameter tuning base. We consider other aspects like weight decay adjustment as a manner to reduce potential overfitting generated by large weights. Similarly, increasing the number of iterations can reduce the error and is therefore considered. Additionally, a wide variety of optimization techniques are available in SEM, including standard BackProp, QuickProp, LM-BFGS (see Nocedal (1980)), and Conjugate Gradient. Finally, the learning rate adjustable to some of the algorithms is regarded as a tunable hyperparameter. Since the tuning of several parameters can not be performed simultaneously in SEM, we first consider a simple MLP with a single hidden layer, 35 HU, and a learning rate of 0.01 to tune the weight decay.
In Table 5.17 one can observe the statistics of the same architecture with different weight decays. The table does not show a consistent pattern w.r.t. weight decay size but implies that the smallest weight decay is favoured overall as its model exhibits the smallest CVE-ASE, highest AUC and Cumulative Lift. Considering the number of neurons and iterations is of interest next. To observe the different required iterations for each optimization technique we set the initial learning rate for QuickProp to 0.01 and that of BackBrop to 0.1. As the QuickProp optimizer learns faster it requires a smaller number of overall iterations with a smaller learning rate.
The iteration plots in Figure 5.18 of both QuickProp and BackProp show a decreasing pattern for an increasing number of iterations. It is also visible how QuickProp either requires fewer iterations for the model to achieve convergence quickly or reaches a possibly

NN's Weight Decay	ASE	AUC	C-Lift
0.0001	0.055	0.907	5.934
0.01	0.057	0.903	5.844
0.001	0.061	0.888	5.533

Table 5.17: NN Weight Decay Tuning Results



(a) Iterations Plot - QuickProp.

(b) Iterations Plot - BackProp.

Figure 5.18: Iteration Plots Of Different NN Optimizers

high overfitting rate and thus induces early stopping. The network with 50 HU shows the smallest CV-ASE for QuickProp, followed by a network with 35 or 40 HU. It is visible how the QuickProp optimizer favours networks with more neurons over smaller networks. Similarly, BackProp prefers overall mid-sized networks with a large number of neurons, yet not too many as the network with 50HU performs marginally worse. Regarding that the differences in CV-ASE are small, we consider further supporting statistics such as AUC values and Cumulative Lift of the models. In Table 5.18, the cross-validation metrics are listed for the different model properties. While the 50 HU QuickProp model shows the smallest CV-ASE, 40 HU provides the highest Cumulative Lift and a small difference in ASE. Considering that the AUC value is the same for the top three models, we choose the model with 40 HU as a candidate model for QuickProp and denote it by $\mathcal{N}_1$. For BackProp, the model $\mathcal{N}_2$ with 40 HU provides the highest Cumulative Lift as well, making it an interesting model for more investigation. [1] Based on the above hyperparameter tuning and deduced models we implement a comparison on the test set as as final assessment, including a model with 2 layers and LM-BGFS optimizer, denoted by $\mathcal{N}_3$.

QProp Properties	ASE	AUC	C-Lift
50HU, 67IT	**0.064**	0.681	2.68
40HU, 67IT	0.065	0.681	**2.77**
35HU, 67IT	0.065	0.681	2.75
20HU, 67IT	0.067	0.679	2.65
10HU, 67IT	0.069	0.675	2.69

BProp Properties	ASE	AUC	C-Lift
35HU, 150IT	**0.0808**	0.695	2.87
40HU, 150IT	0.0810	0.696	**2.91**
20HU, 150IT	0.0811	0.693	2.89
50HU, 150IT	0.0812	0.696	2.90
10HU, 150IT	0.0812	0.696	2.90

Table 5.18: NN Optimizers' Cross-Validation Results

[1] The minor differences in the measures' values, ranking some models with less neurons on the top, could be attributed to the random cross validation.

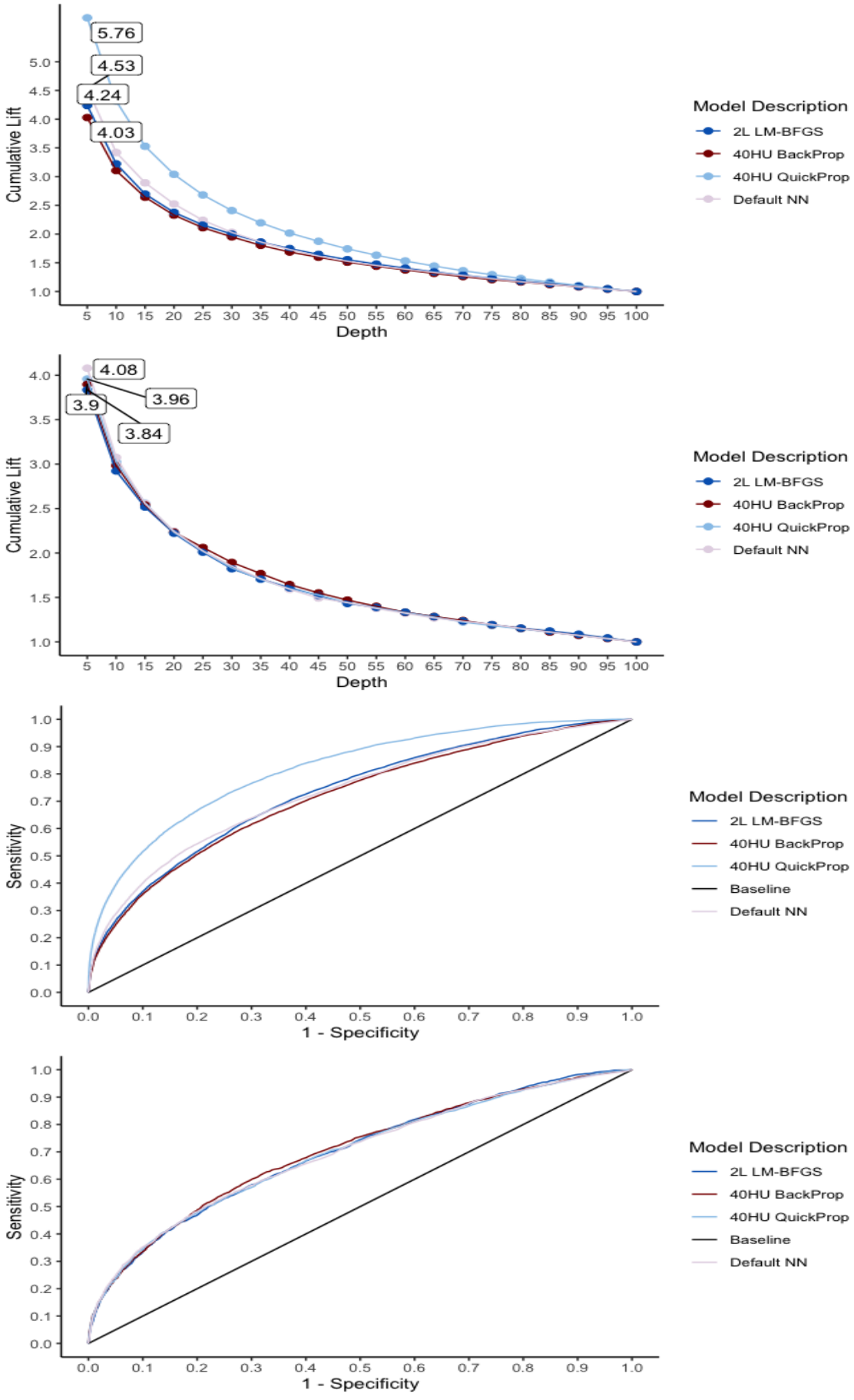


Figure 5.19: Neural Network Models Evaluation

In the comparison, we lastly include the default model $\mathcal{N}_4$ of SEM (see Flowchart A.4), creating an MLP of 1 layer and 3 hidden units using the Conjugate Gradient optimizer. The default learning rate is set to 0.1 and initial weight decay to 0.0001.

The top chart in Figure 5.19 shows the Cumulative Lift curves of all four candidate models on the training set. In the 5% percentile, the QuickProp MLP shows the highest Cumulative Lift of 5.76 vs. a Cumulative Lift of 4.53 for $\mathcal{N}_4$, slightly lower 4.24 for $\mathcal{N}_3$ and the lowest for the BackProp model $\mathcal{N}_2$ with a Cumulative Lift of 4.03. For the upper decile, all models decrease substantially, demonstrating the underlying low frequency of the target observations. The second chart shows the models' performance on the test set. The ranking pattern of the train chart changes implying some key aspects. The QuickProp network yields a considerably lower Cumulative Lift than the first curve, indicating a high discrepancy of almost 2 Lift points between in and out-of-sample performance. This highlights the underlying overfitting and the general model's poor generalisation on unseen data. Due to this performance, $\mathcal{N}_1$ no longer exhibits the highest Cumulative Lift on the test set as SEM's default neural network claims the first rank with a slightly higher Cumulative Lift of 4.08. $\mathcal{N}_2$ - $\mathcal{N}_4$ are comparable to their values on the training set, verifying that some neural network optimizers work well with the data on hand and others, such as QuickProp can lead to overfitting.

$\mathcal{N}_4$ performs overall better than $\mathcal{N}_1$ as it shows higher test Cumulative Lift on both the 5% and 10% percentiles. Overall these findings are in accordance with results visible by the ROC charts on training and test sets. Moreover, $\mathcal{N}_1$'s out-of-sample discrepancy is visible by the high training ROC curve vs. a less steep curve on the test set. The ROC curves of all for models show moderate performance on the test set and deliver an AUC value around 0.7, higher than the baseline model, with $\mathcal{N}_2$ outperforming overall.

For an extensive evaluation, Table 5.19 compares the test set evaluation measures of the neural network models with tuned hyperparameters versus the traditional default suggested by SEM. While the back-propagation MLP exhibits overall the highest Event Precision, $\mathcal{N}_1$ and $\mathcal{N}_4$ show considerably higher TPR, especially $\mathcal{N}_1$'s relatively high

[%]	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$
Overall Precision	73.28	**76.41**	72.84	72.75
Event Precision	55.67	**62.15**	55.06	54.67
Sensitivity (TPR)	**10.71**	7.91	7.46	10.10
Specificity (TNR)	99.05	**99.46**	99.32	99.07
FPR	0.95	**0.54**	0.68	0.93

	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$
G-Mean	**32.56**	28.06	27.22	31.63
$F_{1.2}$-Score	**16.00**	12.32	11.55	15.16
C-Lift (5%)	3.96	3.90	3.84	**4.08**
FNR [%]	**89.29**	92.09	92.54	89.90
AUC	0.695	**0.702**	0.697	0.695

Table 5.19: Classification Statistics - Neural Network Models

10.71% TPR w.r.t. the underlying class imbalance. Models $\mathcal{N}_1$ and $\mathcal{N}_4$ demonstrate the best balance between Precision and Recall as they hold the highest $F_{1.2}$-Score. The adjusted $F_{1.2}$-Score, weighing Recall by just 0.2 times more than Precision, favours $\mathcal{N}_1$ overall by portraying the highest score of 16, a relatively higher score than $\mathcal{N}_2$ and $\mathcal{N}_3$. One can also note that $\mathcal{N}_2$ shows the highest Specificity. In the underlying case study, False Negatives are more costly than False Positives; hence, models with a smaller FNR are preferably selected, for example either model $\mathcal{N}_1$ or $\mathcal{N}_4$ with overall smallest FNRs. It must be pointed out that models with the highest Cumulative Lift, such as model $\mathcal{N}_4$ here, do not guarantee the highest Recall rate. The statistics findings are directly in line with previous results by the Cumulative Lift and ROC Chart regarding $\mathcal{N}_1$'s overfitting potential, as the test shows a similar performance compared to the other models in contrast to a much better-indicated assessment on the training set. Although the models deliver a seemingly high accuracy, we obtain in a similar pattern as the previous machine learning models very high 90% FNR.

Based on all of the above results and overview of different model properties, we select $\mathcal{N}_4$ as a final candidate for the neural network model. $\mathcal{N}_4$ provides the second-highest Recall along with an adequate balance of Event Precision proven by the $F_{1.2}$-Score and highest Cumulative Lift. We therefore investigate this model in more detail. For illustration we include the structure of the MLP graph along some of its estimated weights. Figure 5.20 displays that some of the highest assigned weights between the input layer and hidden layer, belong to the variables *Churn_Last_No_Mon*, *Age* and *Loc_Chng_3Mon* implying more influence on the output. The color of the links imply the relative size of the weights.

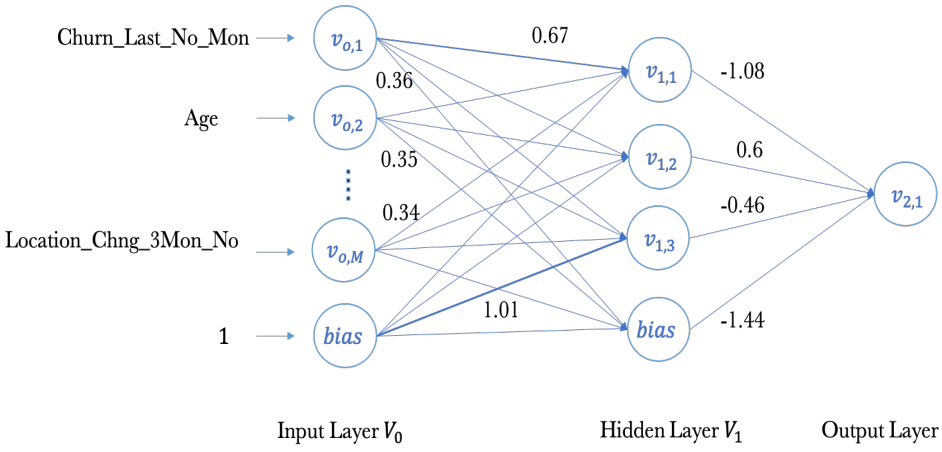


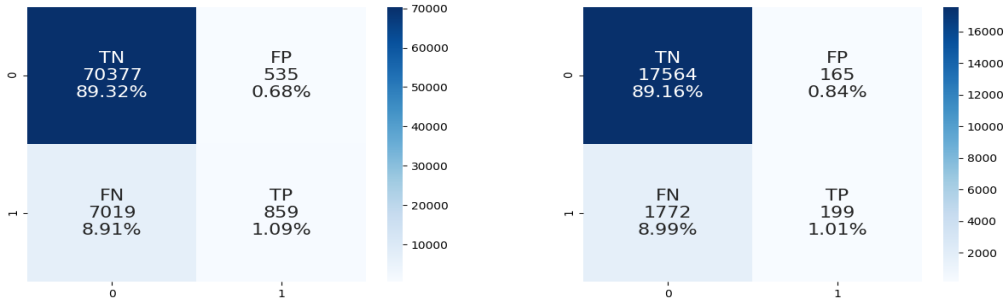
Figure 5.20: Default Neural Network Weights Illustration

	AUC	C-Lift(5%)	Event Precision	TPR	FNR	FPR	$F_{1.2}$-Score
Train	0.73	4.53	61.62%	10.90%	89.10%	0.75%	16.45
Test	0.70	4.08	54.67%	10.10%	89.90%	0.93%	15.16

Table 5.20: Classification Statistics - Neural Network Model $\mathcal{N}_4$

Table 5.20 summarises the research-goal relevant classification measures of model $\mathcal{N}_4$ on both the training and test sets. $\mathcal{N}_4$ shows a high Event Precision on the training set of almost 62% and, in comparison, only 55% on the test dataset. This can be attributed to the smaller number of available cases in the test vs. training set, not allowing the network to obtain a comparable event accuracy. The ROC index changes by a small rate, implying that the sensitivity levels remain relatively stable on out-of-sample observations. Furthermore, we analyse the relevant ranking metrics for the imbalanced dataset on hand. Model $\mathcal{N}_4$ exhibits a test Cumulative Lift of 4.08 (see Figure 5.19) in the 5% percentile, i.e. 20.4% risk customers are captured, which is relatively high. The 10% percentile shows a Cumulative Lift resulting in a 31-34% response capture. The ROC charts in Figure 5.19 also show very similar curves on both sets implying fair robustness and no signs of potentially disadvantageous overfitting. Furthermore, the confusion matrix of the final candidate model $\mathcal{N}_4$ shows on the training set a True Positives count of 859, i.e. 1.09% of the classified observations, relatively high w.r.t. the analysed logistic regression models and slightly lower than random forest models. Consequently, the False Negatives frequency of 8.91% is overall lower. One can notice vice versa that the model is able to properly classify True Negatives, leading to a low frequency of 535 False Positives, 0.68% of the total classifications.
The heat-map in Figure 5.21 shows a fewer Negatives classifications oriented distribution than that of logistic regression models. The test confusion matrix shows similar total percentages for each component as all frequency proportions change by less than 0.17%. This is also indicated by the less than 1% difference in train and test Sensitivity as visible in Table 5.20.

In conclusion, the neural network modelling results display the possible arising differences when the algorithm's hyperparameters are adjusted. From the short review above, we can confirm that different optimisers can generate a higher model Recall. However, high maximum iterations along with an inappropriate weight decay or learning rate can negatively affect the model's generalisability and robustness. These basic findings are consistent with previous studies (Probst, Boulesteix, and Bischl (2019)). The results lead to similar conclusions as previous models w.r.t. predictive capabilities and possible model enhancements. The model however demonstrates the black-box drawback, as the user does



(a) Train Confusion Matrix - $\mathcal{N}_4$.

(b) Test Confusion Matrix - $\mathcal{N}_4$.

Figure 5.21: Neural Network Model $\mathcal{N}_4$ - Confusion Matrices

not get any insights into the important covariates for the prediction.

On the one hand, considerably superior results are achieved with neural network models in comparison to logistic regression with respect to Recall and Cumulative Lift. On the other hand, the models do not provide the interpretability to extract valuable churn drivers and effectively tackle them, e.g. through marketing measures.

Nonetheless, it is well justified to use neural network models for a high and less precise targeted churn detection. To evaluate all implemented models simultaneously and evaluate their performance, the next chapter starts with a full model comparison, along with a discussion of the findings and eventually complementing it by a concluding section.

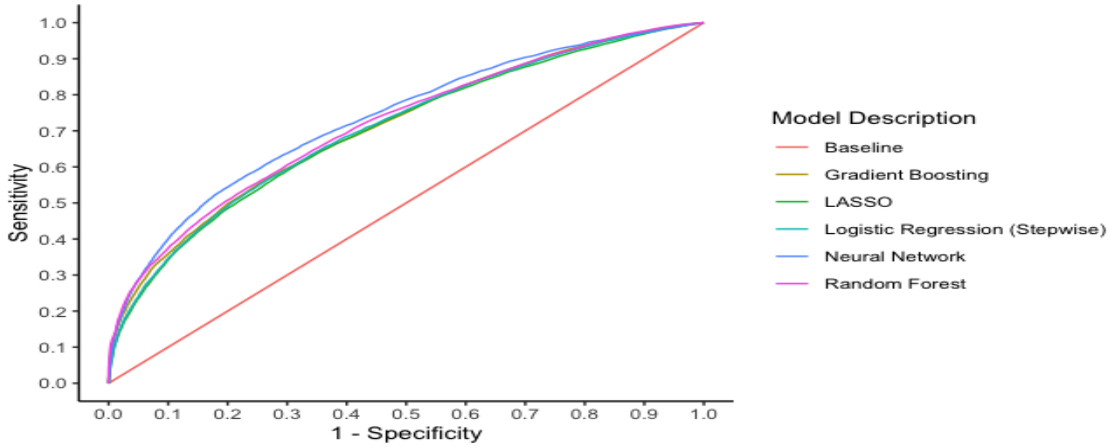
Chapter 6

Case Study: Discussion

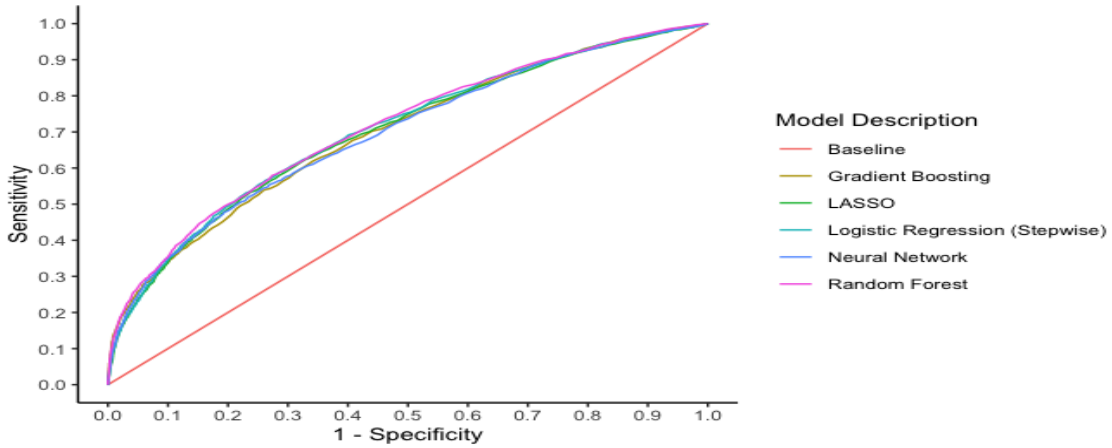
6.1 Model Comparison

In this section, we provide a summary and collective overview of all candidate models' performance on the training and test sets by considering diverse evaluation metrics (see Flowchart A.6). We focus on comparing the models' advantages and disadvantages with respect to general evaluation categories and the key research objectives. For simplicity, we denote the models by the abbreviations defined in Section 2.2.
First, the ROC curves of all models are considered in Figure 6.1. Overall, the charts imply that all models perform better on both sets than the random baseline model. On the training set, NN exhibits the highest ROC curve over the entire range. RF consistently shows a superior curve on all models on the test set, indicating a larger AUC value, which is associated with a better classifier ranking ability. It is interesting to note that the single classifiers LR and LASSO exhibit higher curve values for the FPR mid-range (0.2-0.6) vs. GB and NN visible on the test ROC chart. It appears to be that with increasing cutoffs, the simple classifiers are rather able to maintain an adequate balance between TPR and FPR than gradient boosting techniques.

Figure 6.2 and 6.3 are set against each other to contrast the models' robustness and response capture simultaneously. The random forest's Cumulative Lift of the 5% percentile clearly outperforms the other models on both sets. It is important to note that in comparison to the other models, NN shows the highest discrepancy between training and test Cumulative Lift. The second-best performing classifier is the gradient boosting model with only a small difference in the test set versus RF. Overall, the LASSO model exhibits the lowest 5% Cumulative Lift on both sets. The model rankings do not prevail throughout the upper deciles, e.g. on the 10% decile. On the 10% decile, NN performs better than



(a) Train ROC Charts.



(b) Test ROC Charts.

Figure 6.1: Model Comparison - ROC Charts

gradient boosting. Starting from the 60% decile, all models perform the same as the lower scored observations are assigned to negative classifications. The consistently close Specificity rates imply that all models can properly classify the non-churners. Generally, using the suggested models and contacting the upper ranked 5% of scored test clients will allow the user to capture 19.2-22.1% more of the at-risk customers than contacting a random 5% sample of clients.

Similarly, Figure 6.4 and 6.5 are compared to contrast the models' stability and Precision-Recall (PR) trade-off simultaneously. All in all, the classifiers yield high Recall only at low Precision values, as for 80-100% Recall, the classifiers merely achieve 15-10% Precision. This finding implies that the models' predicted scores are only marginally associated with the outcome. A substantial decrease in RF and GB PR curve steepness is observable on the test vs. train set. While ROC curves focus on both target classes, PR curves cover mainly the minority class implying a higher relevance for the final model

selection.

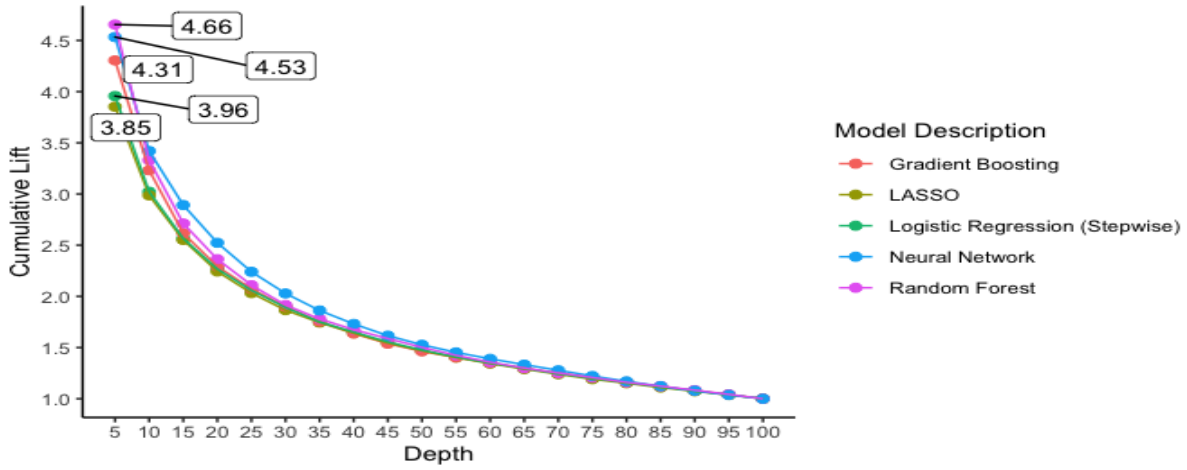


Figure 6.2: Model Comparison - Train Cumulative Lifts Chart

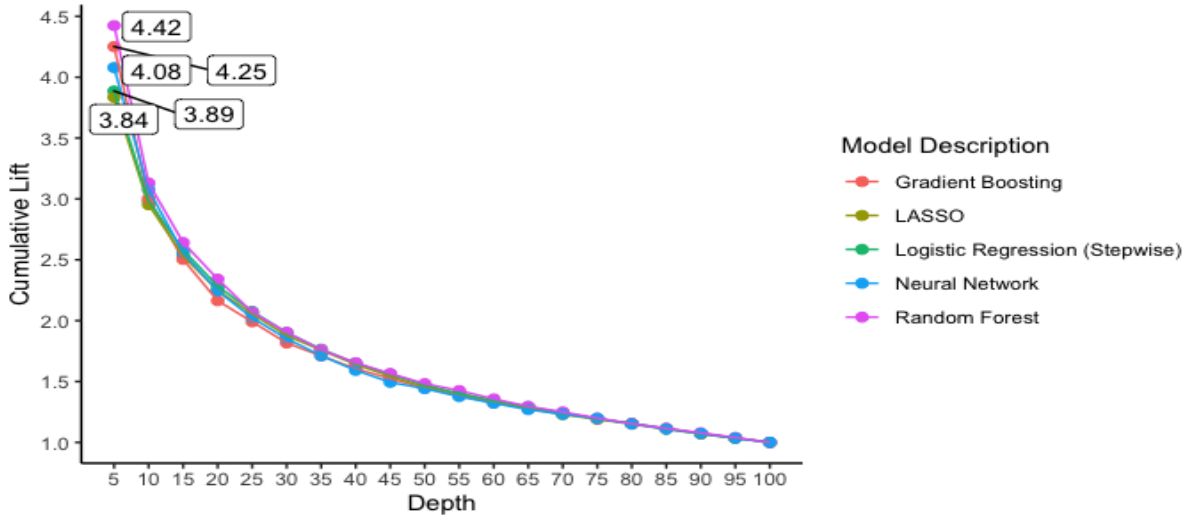


Figure 6.3: Model Comparison - Test Cumulative Lifts Chart

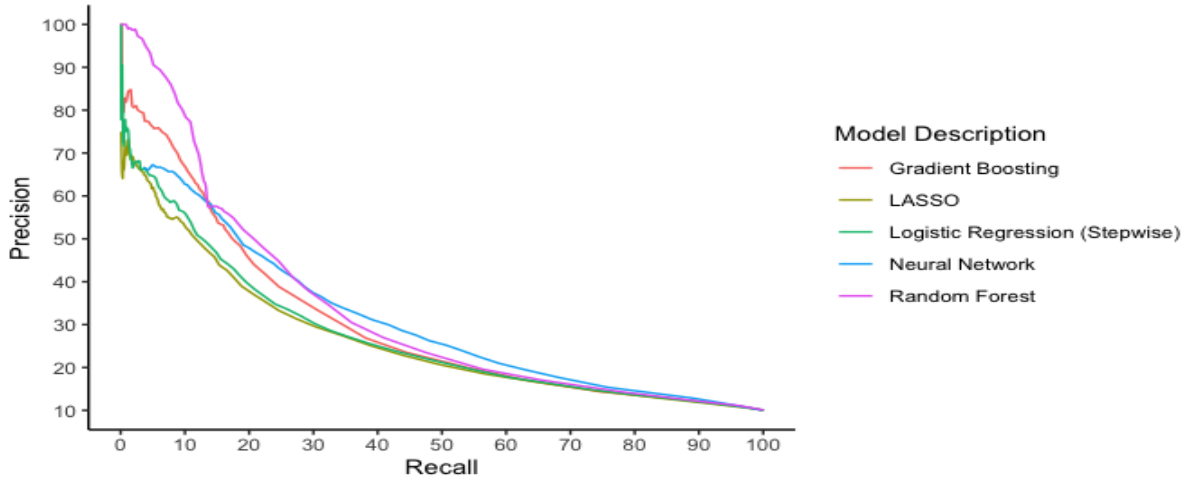


Figure 6.4: Model Comparison - Train Precision-Recall Chart (PRC)

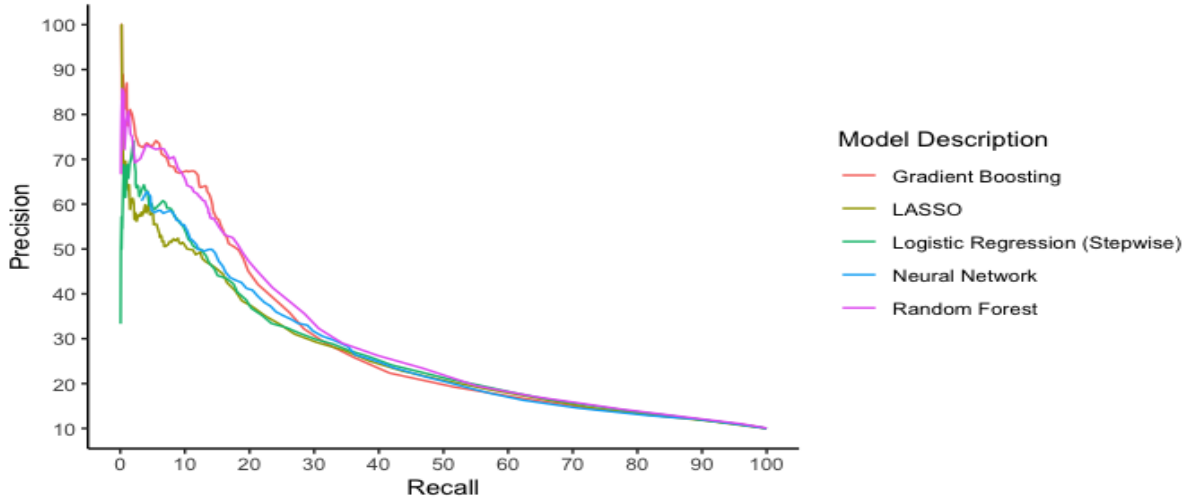


Figure 6.5: Model Comparison - Test Precision-Recall Chart (PRC)

Similar to the ROC chart, the PRC plot in Figure 6.4 indicates that RF has a superior prediction performance through all Recall rates, followed by gradient boosting. Figure 6.5 shows a slightly different picture, which has to be attributed to the differences in test data. The GB model is now clearly leading over the wide range [2,7] and [11,17] of Recall rates, although the random forest is stronger for the excluded rates. Regarding the mid-field of Recall rates (around 20%), random forest continuously outperforms with LASSO and LR additionally showing almost the same percentages. Overall the same primary finding remains, namely that tree-based methods can predict more churners with a higher Precision using the underlying case study dataset. It is also notable how the NN curve underperforms relative to RF and GB despite a high Recall. This can be attributed to low Event Precision below 60%. After determining the final candidate model in the model selection Section 6.2, the user can investigate the Precision-Recall curve to determine the cutoff, where a balance of Recall and Precision levels are acceptable.

We support the drawn conclusions by analysing the evaluation metrics for all models comparatively. ROC plots pose a general difficulty of judging practical performance, as a decision is required about which areas of FPR are relevant and acceptable. While the primary goal of this use case is achieving a high Sensitivity, being satisfied with the mid-FPR-field performance, and not taking into account the strong class imbalance could translate into large numbers of False Positive predictions. Therefore, an evaluation with respect to the AUC value can be more accurate and less cumbersome. Overall Table 6.1 demonstrates that random forest exhibits superior results on all train metrics except FPR and AUC. The lowest FPR is assigned to the logistic regression models, implying higher Specificity and better non-churners detection ability. The second-best overall performing model on the training set is the neural network as it shows the highest AUC value, second-highest Recall, Cumulative Lift, and $F_{1.2}$-Score.

	LR	LASSO	RF	GB	NN
AUC/ROC Index	0.71	0.7	0.72	0.71	**0.73**
Cumulative Lift (5%)	3.96	3.85	**4.66**	4.31	4.53
Event Precision [%]	64.21	56.21	**72.72**	66.78	61.62
Sensitivity (TPR) [%]	5.42	6.78	**11.44**	10.05	10.90
FNR [%]	94.58	93.22	**88.56**	89.95	89.10
FPR [%]	**0.34**	0.59	0.48	0.56	0.75
G-Mean (3.47)	23.24	25.96	**33.74**	31.62	32.90
$F_{1.2}$-Score (3.49)	8.68	10.59	**17.47**	15.42	16.45

Table 6.1: Train Classification Statistics - All Models

	LR	LASSO	RF	GB	NN
AUC/ROC Index	0.70	0.69	**0.71**	**0.71**	0.70
Cumulative Lift (5%)	3.84	3.89	**4.42**	4.25	4.08
Event Precision [%]	59.28	52.26	64.15	**67.24**	54.67
Sensitivity (TPR) [%]	5.02	6.44	**10.35**	9.89	10.10
FNR [%]	94.98	93.56	**89.65**	90.11	89.90
FPR [%]	**0.38**	0.65	0.64	0.54	0.93
G-Mean (3.47)	22.37	25.30	**32.07**	31.37	31.63
$F_{1.2}$-Score (3.49)	8.04	10.06	**15.77**	15.21	15.16

Table 6.2: Test Classification Statistics - All Models

Last but not least, we compare all models on the test set to obtain an unbiased assessment from which we can draw conclusions and understanding. The test set deteriorates the Precision performance of all models except gradient boosting. A gradient boosting's moderate 67.24% Precision is shown to be the highest of all models on the test set, additionally higher than the training value. Unlike Precision, Recall rates remain relatively close to train values. The random forest model generates the highest rate of TP classifications as its Sensitivity is 10.35% on the test set. In consistency with FPR train values, logistic regression exhibits the highest Specificity, which is, however, not of primary interest. As RF no longer displays both the highest Recall and Precision, it can be useful to consider the adjusted $F_{1.2}$-Score, offering an assessment through a Recall-weighted measure. The random forest provides evidence to its good balance as its $F_{1.2}$-Score of 15.77 is the highest. Gradient boosting, in contrast, has a slightly lower 15.22 score, provided by the higher Precision and a marginally lower Recall. With respect to

the models on hand, the model with highest Cumulative Lift, i.e. RF also obtains the highest Sensitivity. Overall, random forest's superior performance can be attributed to its built-in-functionality of minimising the overall error rate, such that majority classes of imbalanced datasets get assigned a low error rate. In contrast, the minority class obtains a larger error rate.

From the short review above, the following summarised key findings emerge with respect to each evaluation category:
Considering model robustness, not all top-ranked models under consideration of business objectives would be highly favoured. As shown through comparative statistics and charts on training and test sets, the almost overall best performing classifier RF comes with a cost of high deviation between training and test values. We consider the gradient boosting model to be the top choice for an optimal trade-off between bias and variance. The model, providing the second-highest Cumulative Lift and highest Precision, additionally provides the smallest deviation from training values and accompanying model stability. In contrast, the model providing the least robustness is the Conjugate Gradient optimised neural network as high out-of-sample deviance was demonstrated through the comparison of several metrics on train and test set.

Examining the interpretability of the models confirms the importance of investigating present machine learning advancements in model interpretability. As shown in Section 5, logistic regression and LASSO techniques are able to provide a quantitative interpretation of the variables' influence on churn likelihood. This is particularly important when investigating clients with a tendency to churn, as it might improve the analyst's and marketeer's understanding. Tackling specific client segments exhibiting particular characteristics is mainly attributable to perceiving the predicted increasing, decreasing, or neutral effects. While the tree-based methods RF and GB provide a variable importance ranking, they do not explicitly demonstrate the direction or type of effect.
Together, all models' findings consistently confirm that important drivers of churn prediction are previous churn history, agent-induced properties, client's behaviour in other insurance contract categories, and essential client characteristics. Meanwhile, NN shows an interpretability disadvantage as it is a complete black-box model.

Handling missing values can contribute to model efficiency. Considering the additional run time by pre-adjusting missing values for the logistic regression, LASSO and neural network models are indicative of a potential disadvantage. Meanwhile, tree-based techniques automatically process missing-values as decision trees include the built-in functionality saving extra pre-processing efforts.

Considering Precision as an indicative measure of the classifier's predictive power under

our use case's imbalanced dataset conditions, we conclude that gradient boosting provides the model with the highest churn predictive power.
Similarly, taking into consideration the key goal of churn identification, it is deduced that random forest provides the highest Recall. Together, the findings confirm the strength of tree-based machine learning techniques in churn prediction.

6.2 Model Selection

To summarise the results and obtain a conclusive overview, this section includes final remarks and reflective judgments on the key findings. The main research question and sub-questions shall be revisited and answered with respect to the obtained results. The goal is not to determine one ultimate champion model for all research goals but to meet existing business objectives by considering the appropriate model accordingly.

The first objective was to determine the techniques that best identify the highest possible number of clients intending to cancel their outstanding insurance contract while preserving appropriate Precision and Accuracy levels. To this end, a model with highest Sensitivity is most useful, while other characteristics such as Precision could be only moderate and interpretability less relevant. Following the findings above, if one wants to identify the highest number of at-churn-risk customers, a random forest with Senistivity 10.35 is preferable, followed by a feedforward neural network with Sensitivity 10.10.
Possibilities to create highly precise prediction models on imbalanced datasets and calculating the customers' churn probabilities can be achieved by applying class imbalance reduction techniques along the standard data-mining process. The methods worked very well by providing models with up to 67% Event Precision on the test set.
Random forest and gradient boosting models provide the best prediction models delivering both high Recall and Precision values. One approach of combining the different models' advantages is by comparing the client scoring or using new interpretability methods.

The next research goal considered is the identification of relevant churn drivers. To this end, a model with highest interpretability is most useful, while other characteristics such as Precision and Recall could be moderate. The models with intrinsic interpretability, such as logistic regression or LASSO, do provide an insight into why customers churn while not having highest predictive power. Hence, we conclude that the model with moderate Precision and highest interpretability is the logistic regression model with stepwise selection. Solutions to overcome this joint issue are therefore further investigated in Section 6.3.2.
Key customer characteristics to predicting relevant churn drivers were categorised in three

classes: basic information, product-related information, and agent-related features. The identified drivers describing customers churn behaviour are from these classes with churn behaviour over-represented. Customers are likely to churn if they have already churned other active contracts at the same insurance company. If the responsible agent changes, this increases the likelihood that the client will churn, e.g. to keep the agent at a new respective insurance company.

The last research goal considered is the generation of a ranked list of at-churn-risk customers based on assigned churn score and identified churn behaviour. Using a decision tree, it is possible to directly deduct churn probabilities based on the relevant churn drivers and create a list of the at-churn-risk customers. Taking into account that most clients churn 5-8 months before contract end and the relevant deducted churn drivers, the list can be further segmented. We give a more extensive insight into possible customer profiling and segmentation in the next section.

6.3 Actions and Insights

Taking into account the proposed models by Section 6.2, this section combines an intepretability aspect with the standard technqiues and provides insight into the customer retention actions in the insurance sector. Furthermore, some research limitations are described to make the users aware of possible enhancements.

6.3.1 Actions in the Insurance Sector

Actions Post-Modelling

Beyond merely identifying the potential churn customers and drivers, more importantly is determining early enough signals for customer intervention. The alerts should be against customers, who in the recent period show signs but still hold standing contracts, such that targeted proactive retention is applied. Optimising marketing actions and taking effective measures, consequently from churn prediction, can reduce the number of customers falling into the at-churn-risk category and increase revenues from the same base of existing customers. It is important to note that churn modelling is an iterative process; i.e. there has to be regular back-testing and development of the considered models to confirm prediction quality.

After identifying customers at risk of churning, the marketeers have to put the models into production. It is vital to know how exactly the marketeer can reach the customer by

considering individual marketing actions for the diverse customer base. Taking random proactive retention measures do not necessarily deliver lower churn rates, but instead applying targeted proactive retention to maximize chances that the specific customer will remain would be more effective. For instance, a client might respond more positvely to retention emails than calls and another may negatively react to any kind of marketing.
It needs to be taken into consideration that the behaviour and preferences of each customer are different. Consequently, not all high at-churn-risk customers are marketing-affine. Instead of generating marketing losses by addressing the wrong clients, marketeers can approach clients with a high predicted churn index and high marketing-affinity. One can, for instance determine this affinity by running marketing campaigns and storing the indicating repsonse of customers in the database over the years. Alternatively, clients with high response rates to retention campaigns can be selected as investigable control groups and be optimally considered in the following models. For illustration, the client clustering within the different matrix parts can be applicable post scoring in Figure 6.6.

Considering that human behaviour can deviate, marketing departments should rely on customer rankings rather than absolute predicted probability values. The rankings should be based on frequency and recency aspects. Recency is provided automatically by target definition, and frequency is provided through higher rankings of predicted probabilities. After ranking the customers, the user will have a clear perspective on the clients' risk level by considering a risk-based segmentation w.r.t. engagement levels and marketing-affinity.

Figure 6.6 illustrates a possible customer segmentation with respect to predicted churn probability and profitability provided by the customers. The first segment, identifying top priority clients, includes clients with the highest predicted churn probability and very high profitability. These clients are considered of the highest priority as they represent

Figure 6.6: Customer Segmented Targeted Retention

the most vital risk exposure by the identified at-churn-risk customers and ensure high profitability for the company if proactive measures are taken against their churn intention. The clients in this segment can be further partitioned, for instance by considering their marketing-affinity, demographic aspects, or behaviour history. The segmentation can be made more actionable by identifying the respective consequences of targeting the individual created groups.

Cost-Sensitive Evaluation

The approach of cost-sensitive learning and evaluation, already introduced in Section 3.2.2, can help improve a majority-class-biased model. Trying to minimize the overall costs or maximizing revenues as a business objective, the user can assign a higher misclassification cost for the minority class. These can be applied by using the exact costs provided by marketing teams exerted for the measures taken in customer retention along with the costs of customer acquisition. Another manner would involve considering the costs and benefits of respective classification post-training. For illustration, we consider a costs-savings analysis for the use case on hand post-training. If the insurance company engaged every single customer or even a high unnecessary number due to a high False Positives count, the costs would be too high. Nevertheless, the higher costs induced by customer acquisition should be taken into consideration. Focusing retention efforts on a small valuable subset of high-risk customers would, therefore, be a more effective strategy.
An approximation of actual business impact can be achieved by following the analysis of two of the most promising models generated, namely random forest and gradient boosting. As mentioned in the Results Section 5, the choice of a cutoff score for the confusion matrix can lead to better results according to the desired business objective. Assuming that we want to achieve one particular business goal, for instance cost-saving, we calculate the resulting costs for each threshold and choose the cutoff with minimum costs. For this purpose, we state the following assumptions with regard to costs as implied by experts at ERGO. The cost of retention strategies used per customer is assumed to be approximately €15, implying a TP and FP cost of €15. For each churn-prone client that we don't approach due to a FN, we assume the client leaves and we have to acquire one new client at €30 cost. Assuming that customer acquisition costs are twice as much as customer retention measures, we consider the following costs for threshold optimisation:

Observing the curves in Figure 6.7 leads to optimal cutoff and savings findings. The curve demonstrates that the random forest model achieves the lowest costs at the 0.54 cutoff, while gradient boosting reaches the optimal cost savings at the 0.44 cutoff. The random forest model would imply total costs of €57,705 versus slightly lower €57,375 costs if the

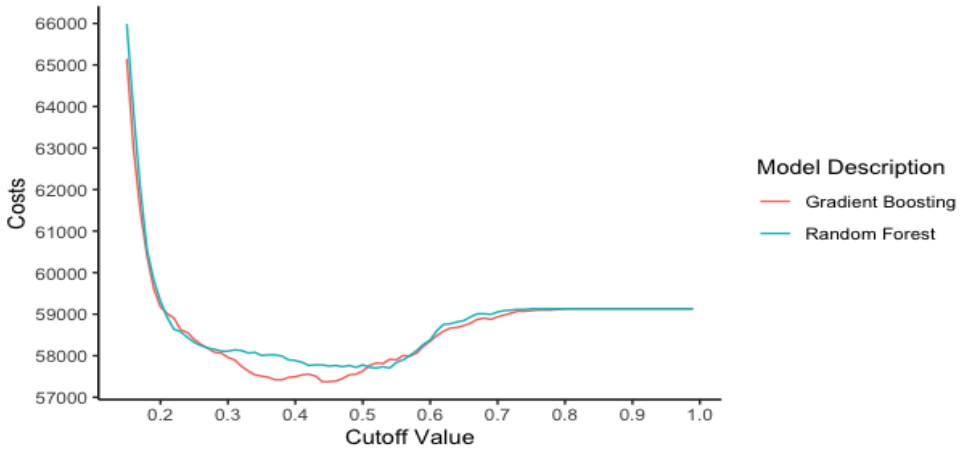


Figure 6.7: Cutoff Optimisation w.r.t. Retention & Acquisition Costs

gradient boosting model is used and all the TP & FP identified customers are approached. Considering the use of the default 0.5 cutoff would lead to total costs of €57,780 when random forest is used and €57,630, in case of gradient boosting. These imply respective minor cost savings but still provide optimal results w.r.t cost-sensitivity.

This example illustrates the added value of optimizing the generated machine learning models, by determining the optimal cutoff score for higher Precision, as well as a beneficial impact on the business. For accurate computations, we suggest an exact identification of costs and benefits to reach the more business-optimised churn prediction models.

6.3.2 Interpretable Machine Learning

Understanding why the machine learning models classify churners as they do can drive the marketing and analytics teams to achieve the desired profitability and customer satisfaction goals. As observed in Section 6.2, depending on the business objective, either accuracy or interpretability can be considered a priority. In some applications, the transparency provided by interpretable-models such as logistic regression is preferred over accuracy. While the random forest was shown to be the champion model w.r.t. our research goal, marketing teams can only extract the important contributing drivers, but not their effect on clients' churn behaviour. Like many other domains, interpretable models have thus gained importance in recent years in the churn prediction research. Newly introduced methods can help to overcome the obstacles of black-box models and simultaneously make use of the suggested highly sensitive models.
Considering that ERGO's suggested models do not include an interpretability perspective, we consider this a new important post-modelling orientation and a customer apprehension analysis that we can contribute to.

Interpretability methods can be categorised through several criteria. First of all, the methods can be classified into intrinsic and post-hoc interpretability categories (see Molnar (2019), p. 16). Interpretability is achieved by using simple structured models (intrinsic) or applying a model analysis post-training (post-hoc). Similarly, one can categorize the techniques into model-specific and model-agnostic procedures. Model-specific methods are only applicable to a specific machine learning class, while model-agnostic methods are post-hoc techniques practical for any machine learning model (see Molnar (2019), p. 17).

We consider two further general categories, namely global and local interpretability categories (see Molnar (2019), pp. 17-19). While global models use all observations in the dataset to explain the input-output relationship, local methods focus on explaining the prediction of a single instance. Sometimes, global interpretability does not give specific enough insights; thus, we apply the local methods examining specific instances or a group of predictions, i.e. clusters.

In this section, we give an insight into the newly implemented model-agnostic methods and consider a model-specific solution. Some of the model-agnostic methods were conducted for the final candidate models to help analyse the interpretability of the suggested black-box models such as random forest, neural networks, and gradient boosting. For the implementation of model interpretability techniques, the newest tool from SAS for machine learning, namely SAS Viya, is used. Similar to SEM, remarks are included in Appendix A.2.3 to explain the respective implementation steps and procedures.

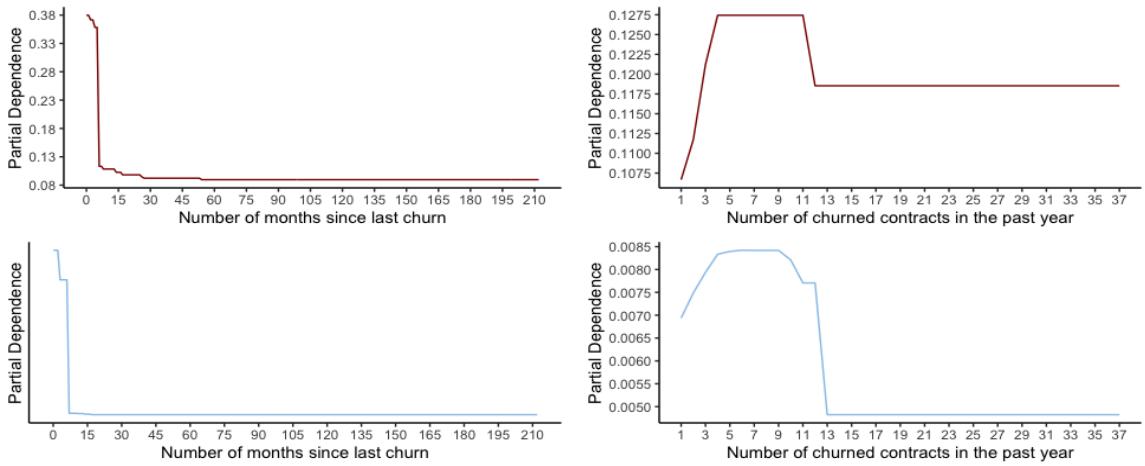
Partial Dependence Plots

One of the commonly utilised global interpretability methods to understand contributing churn drivers is the partial dependence plot. The partial dependence function uses all instances for plot generation and therefore demonstrates the global association between a feature and the target. The partial dependence plot displays the marginal effect on the classification result of the regarded trained model, i.e. the partial dependence of predictions on those observed features (see Molnar (2019), p. 81). The plot can reveal if the underlying association is linear, a step-function, or a more complex relationship. This reveals the direction of dependency, i.e. whether we should expect increasing, decreasing, or neutral effects when the value of an input variable changes. In classification settings, the plot displays the probability of the target occurring for different values of the considered feature. We state the definition in the binary case as

follows (see Hastie, Tibshirani, and Friedman (2001), p. 370):

$$f_k(\boldsymbol{x_j}) = \log p_k(\boldsymbol{x_j}) - \frac{1}{C}\sum_{c=1}^{C} \log p_c(\boldsymbol{x_j}).$$

For each class c, the log of predicted probability is computed for the separate $k \in \{1, \ldots, C\}$ created models and the respective predicted probabilities of each. Like the Logit model, the plots can reveal the effect on log-odds of that class occurrence by the considered feature as the scale is similar to that of Logit. For illustration, Figure 6.8 displays the partial dependence plots of two features in the gradient boosting and random forest candidate models chosen in Section 6.1.



(a) PDP No_Mon_Last_Churn (b) PDP No_Churn_Contr_1Y

Figure 6.8: Partial Dependence Plots For GB and RF Models

The partial dependence plots in Figure 6.8 illustrate interpretability. Figure 6.8(a) and 6.8(b) show two plots, the dark-red belonging to the gradient boosting model and the blue plots to the random forest model. Both plots in Figure 6.8(a) show a similar sharp decreasing interaction for an increasing number of months since the last churn occurred. The plots also show neutral effects for a higher number of months. In contrast, the partial dependence plot of the number of churned contracts in the past year first shows a sharply increasing interaction, followed by a decrease and neutralisation. This can be attributed to the fact that there are only very few instances with more than 13 churned contracts in the past year. It is important to mention that generally one has to be careful which feature ranges are considered, as for some, there might be no available data (such as here, ≥ 14 churned contracts). It is intuitive that with increasing units of churned contracts, the customer is more likely to churn. Likewise, recent churners are more likely to churn other active contracts, than customers with the last occurring churn more than 15-20 months ago.

LIME

We consider the LIME (Local Interpretable Model Agnostic Explanations) method in comparison to the above demonstrated global partial dependence plots. This local surrogate model is used to explain individual predictions of the respective black-box model, in contrast to the global models considering all instances. LIME does this by merely explaining these through fitting a linear regression to the original classifier's features using the predicted probabilities as the response variable. This is done by applying the following steps (see Ribeiro, Singh, and Guestrin (2016)(a)):

1. Select instance $\boldsymbol{x}$ for further investigation and consider it as the centroid of a cluster.
2. Generate more sample points around this instance by using the standard deviation from the overall training set. Alternatively, exponential smoothing kernels can be used to define the neighbourhood (see Ribeiro, Singh, and Guestrin (2016)(a)).
3. Enter the synthetic instances into the black-box model and create a new response variable using the computed predicted probabilities.
4. Weigh the new sample points w.r.t. their distance to the observed instance, i.e. by the weight factor $w = e^{\frac{x-s_i}{\sigma}}$ for σ a scaling factor.
5. Fit a LASSO model to the new labelled dataset to select the important features.
6. Use the selected features by the LASSO model to fit a linear regression model providing interpretability.

Implementing these steps in SAS Viya directly delivered the following local model statistics for two randomly chosen instances using the suggested gradient boosting model: As visible in Table 6.3 the local models fitted to each of the clusters only deviate by

Statistic	Centroid 1	Centroid 2
Adj. R^2	0.709	0.709
ASE	$9e^{-05}$	$9e^{-05}$
R^2	0.709	0.709
AIC	-82685	-82686

Table 6.3: Classification Statistics - Local Surrogate Model

amounts captured in more than three digits. Both models suggest an R^2 of 0.709 for the linear regression model which is considered a strong effect size. Moreover, both instances are assigned to a low-risk category as the predicted churn probability is only

0.084. Considering the first observed instance, Figure 6.9 shows the LIME explanations of the prediction. The coefficients in the LIME explanation show that for this instance all considered feature variables reduce the risk of churn, which is in line with the customer being assigned a low-risk predicted probability. Looking at the specific feature values for this observation can give the analysts intuition on how to create the customer profiling.

An alternative model-specific solution is the comparison of logistic regression and random forest customer scoring post-modelling. In efforts of making use of a simple model like logistic regression, one further approach we implemented is the comparison of scoring distributions. With respect to the business objective, for example we assumed that a considerable percentage of at-risk customers identification is sufficient, and compared the 1000 top-ranked (highest predicted churn probabilities) clients of both random forest and logistic regression models. Subsequently, we deducted segments of the top-ranked clients of each and checked if a high proportion is present in both models. Conformity of 70% showed that using a logistic regression model will help us identify 70% of the 1000 top-ranked clients by the random forest model. Considering aspects like marketing-affinity and retention response can further determine if these clients are worth approaching.

It is important to note that the methods introduced in this section provide a first insight into the black-box models, but have some drawbacks and limitations:
Partial dependence plots, for instance are easy to implement, displaying only the average marginal effects, but allowing heterogeneous effects to possibly stay undiscovered by assuming the features are independent (see Molnar (2019), p. 85). While the idea of LIME is also reasonable and practical as it is model-agnostic, one concern is the possible instability of the explanations, as two close observations could provide substantially different explanations (see Ribeiro, Singh, and Guestrin (2016)(b)), misleading the user.

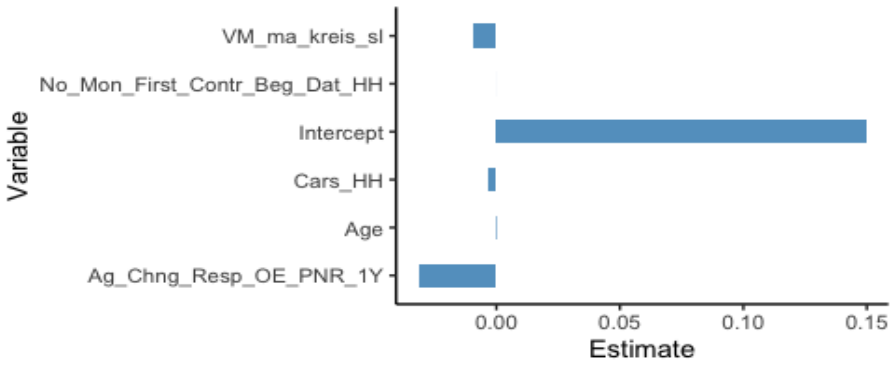


Figure 6.9: Estimates - Local Surrogate Model

Despite the possible drawbacks, the above findings imply that extending the models by an interpretability analysis could enhance customer understanding and help derive more

targeted measures. We, therefore, recommend extending the existing churn prediction model by such explainable methods.

6.3.3 Research Limitations

Limitations of the study, that the insurance company and users should be aware of, are shortly described below.

The findings concluded for the household insurance category do not automatically transfer to other insurance categories. Instead of predicting if an observed client will churn in any of the categories collectively, this study uses a dataset labelled by a household churn target. Thus, the models are mainly focused on predicting household insurance churn. The insurance company can, however, easily embed the modelling techniques to the whole insurance categories range by generating a merged dataset for all segments jointly. It is, however, important to note that difficulties in modelling will rise when customers only hold one active insurance contract or when trying to extract respectively suitable measures.

A further considerable limitation is missing information about past performed model-specified actions, implying how approachable the customers are. Not incorporating previously taken actions into the predictive model, hence recommending possibly the same actions and measures for the same customers once more in the following model implementation, can lead to costly consequences. Taking into account that reducing the number of taken actions per client is more effective and profitable, these clients should be eliminated in the post-modelling strategy or be addressed with new actions. This limitation rises because no particular model-extracted actions have been taken by ERGO in the past years to be included in our model generation.

The suggested churn prediction model is mainly based on historical data conveying comparable traits between the present and previous customers. According to the equivalence of the datasets' features, customers whose churn probability corresponds to clients whose churn behaviour is now known, are assigned a similar classification. The sole reliance on the assumption that previous clients' data corresponds to future churn behaviour can be disadvantageous when seasonal effects, extreme events, or deviating human behaviour occur.

Finally, the sutdy does not identify the optimal time window to take proactive actions against potential churners. That is, the model does not incorporate when to best apply the measures, e.g. by contacting the customer. If the model suggests a post-modelling action, the only guideline given is to reach out to the client as fast as the marketeers can to effectively guarantee the benefits are received right after costs are generated.

Chapter 7

Conclusion

Customer retention with the help of customer churn prediction remains a key business objective for all industries and is known to be a more valuable strategy than customer acquisition. Substantial challenges faced during churn prediction include class imbalance, black-box interpretability, and model specification. While a wide range of literature is already approaching these topics, companies are still in search of better models providing a combined offer of customer valuation, class imbalance, and high predictive power.

In this study, we developed a model to predict churn for our industry partner, ERGO, using well-known machine learning models. The following key findings emerged throughout our analysis of model results.
Classic regression models are outperformed by more sophisticated models such as random forest and gradient boosting. We found some encouraging results when we fine-tune the hyperparameters of the respective models and provide insights into the interpretability.

The creation of a highly precise prediction model on the imbalanced dataset was handled by applying random under-sampling along the standard data-mining process, which showed increased predictive accuracy and Cumulative Lift. Under-sampling the majority class worked well by providing models with up to 67% Event Precision on the test set.

The models with intrinsic interpretability, such as logistic regression or LASSO, did provide an insight into why customers churn but did not offer the most precise models. Hence, we conclude that the model with moderately high Precision and interpretability is the logistic regression model via stepwise selection. New solutions allowed higher predictive models to be implemented with an insight into churn reasons. Partial dependence plots and LIME provided the alternative of implementing an interpretable gradient boosting model instead of the less sensitive logistic regression model.

Key identified churn drivers were churn-behaviour variables such as the number of months

since the client's last churn, the number of churned contracts in the last 6 months, or 1 year, the last cross-selling category churned and if the client churned a contract in the last 3 months. Additionally, agent related features were demonstrated to have effects such as the agent's change of location, responsibility change, or proportion of supervised clients relative to other agents. Last but not least, some basic client data was shown to be informative such as age, high school diploma, and rental score.

Random forest delivered the overall best performance, gradient boosting the most robust, and logistic regression the most interpretable results. The drawbacks of each model are addressed by considering cost-saving analysis and interpretability methods. It was concluded that cost and benefit analysis is essential to predictive churn analytics and should not be overlooked.

Apart from the concluded findings and mentioned research limitations, several interesting perspectives for future research have emerged. The aim of future work should be to generalize further the theoretical analysis of highly predictive models suitable for imbalanced labelled datasets and apply them for churn prediction frameworks.
A primary takeaway from the models presented is the generalisability to the entire insurance type range. Any service-based company can gain an understanding and insight into the proposed actions, and predictive statistics by a similar model suggested. The analysis provided is thus relevant and can be extended to other organisations, products, and sectors.
Given that machine learning models often only rely on historical data, memorising a cause-and-effect structure, black swan events are not incorporated when they occur. We want to tailor and stay ahead of such infrequent events.
Churn prediction modelling could be extended with a focus on a time dimension. Databases can be enriched to include time-series data improving the weights of events and target labelling, as well as predicting when the user would churn, which can consequently provide an investment time window optimization.
Finally, taking ethical considerations into account, compliance with regulations for data protection and privacy concerns was considered in this research as the clients' sensitive data is analysed in a pseudonymous form. However, along with increasingly tight regulations, there is a rise in the need to quantify privacy and define privacy mathematically. While many heuristics were implemented for privacy perseverance such as anonymization (removal of identifiable attributes), they were proven to fail by all the linkage attacks conducted, calling for a need for a robust privacy definition. Some advances have been made by defining the rigorous mathematical definition 'differential privacy' (see Ji, Lipton, and Elkan (2014)), leaving space for more theoretical analysis of learning with differential privacy and its suitability for churn prediction frameworks.

Bibliography

Adams, Ryan P. (2018). *Linear Classification with Logistic Regression*. Tech. rep. Princeton University, pp. 1–9.

Akosa, Josephine S. (2017). "Predictive Accuracy : A Misleading Performance Measure for Highly Imbalanced Data". URL: https://support.sas.com/resources/papers/proceedings17/0942-2017.pdf.

Arunava. (2018). *Sigmoid Function*. URL: https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e.

Bahety, Anand. (2014). "Extension and Evaluation of ID3–Decision Tree Algorithm". In: pp. 1–8. URL: http://ssltest.cs.umd.edu/Grad/scholarlypapers/papers/Bahety.pdf.

Batista, Gustavo E.A.P.A, Prati, Ronaldo C., and Monard, Maria Carolina. (2004). "A study of the behavior of several methods for balancing machine learning training data". In: *ACM SIGKDD Explorations Newsletter*, pp. 1–20. ISSN: 19310145. DOI: 10.1145/1007730.1007735.

Bellman, Richard E. (1961). *Adaptive control processes: a guided tour*. Princeton, New Jersey.: Princeton University Press.

Breiman, Leo. (1997). *Prediction Games And Arcing Algorithms*. Tech. rep. Berkley, CA.: Statistics Department, University of California, pp. 1–33.

— (2001). "Random Forest". In: *Machine Learning* 45.1, pp. 5–32. ISSN: 1098-6596. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3.

Brownlee, Jason. (2020). *(URL)*. URL: https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/.

Burez, J. and Van den Poel, D. (2009). "Handling class imbalance in customer churn prediction". In: *Expert Systems with Applications* 36, pp. 4626–4636. ISSN: 09574174. DOI: 10.1016/j.eswa.2008.05.027. URL: http://dx.doi.org/10.1016/j.eswa.2008.05.027.

Chauhan, Nagesh Singh. (2020). *Decision Tree*. URL: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html.

Chawla, Nitesh V. et al. (2002). "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research* 16, pp. 321–357. ISSN: 10769757. DOI:

10.1613/jair.953. arXiv: 1106.1813. URL: https://arxiv.org/pdf/1106.1813.pdf{\%}0Ahttp://www.snopes.com/horrors/insects/telamonia.asp.

Diestel, Reinhard. (2016). *Graph Theory*. Springer, pp. 1–274. ISBN: 978-1-4612-9969-1. DOI: 10.1007/978-1-4612-9967-7. arXiv: arXiv:1011.1669v3.

Fahlman, Scott E. (1988). *An empirical study of learning speed in back-propagation networks*. Tech. rep. CMU-CS-88-162. Pittsburgh, USA: Carnegie Melon University, School of Computer Science, pp. 1–19. URL: http://scholar.google.com/scholar?cluster=3383018529262918047{\&}hl=en.

Famili, A. et al. (1997). "Data Preprocessing and Intelligent Data". In: *Intelligent Data Analysis Journal*.

Frank, Vanden Berghen. (2003). *Classification Trees : C4 . 5*. Tech. rep. Universit Libre de Bruxelles, pp. 1–5.

Freund, Yoav and Shapire, Robert E. (1999). "A short introduction to Boosting". In: *Journal of Japanese Society for Artificial Intelligence* 14.5, pp. 771–780. arXiv: 1508.01136. URL: http://arxiv.org/abs/1508.01136.

Friedman, Jerome H. (2001). *Greedy Function Approximation: A Gradient Boosting Machine*. Tech. rep. Stanford, CA: Sequoia Hall, Stanford University, pp. 1–39.

Gareth, James et al. (2013). *Introduction to Statistical Learning with Applications in R*. Springer, pp. 1–426. ISBN: 9781681735054. DOI: 10.2200/S00899ED1V01Y201902MAS024.

Gour, Vishal et al. (2010). "Improve Performance of Extract, Transform and Load ETL in Data Warehouse". In: *International Journal on Computer Science & Engineering* 2.3, pp. 786–789. ISSN: 09753397.

Hastie, Trevor, Tibshirani, Robert, and Friedman, Jerome. (2001). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York: Springer, pp. 1–745. ISBN: 9780387848570. DOI: 10.1007/b94608. URL: http://www.springerlink.com/index/D7X7KX6772HQ2135.pdf.

He, Haibo and Ma, Yunqian. (2013). *Imbalanced Learning*. John Wiley & Sons, Inc., pp. 1–210. DOI: 10.1002/9781118025604.ch3.

Hossin, Mohammad and Sulaiman, MN. (2015). "A Review on Evaluation Metrics for Data Classification Evaluations". In: *International Journal of Data Mining & Knowledge Management Process* 5.2, pp. 01–11. ISSN: 2231007X. DOI: 10.5121/ijdkp.2015.5201.

Hssina, Badr et al. (2014). "A comparative study of decision tree ID3 and C4.5". In: *International Journal of Advanced Computer Science and Applications* 4.2, pp. 13–19. ISSN: 2158107X. DOI: 10.14569/specialissue.2014.040203.

Hutter, Frank, Kotthoff, Lars, and Vanschoren, Joaquin. (2019). *Automated Machine Learning*. Vol. 498. Springer, pp. 233–317. ISBN: 9783319009599. DOI: 10.1007/978-3-319-00960-5_6.

Ji, Zhanglong, Lipton, Zachary C., and Elkan, Charles. (2014). "Differential Privacy and Machine Learning: a Survey and Review". In: pp. 1–30. arXiv: 1412.7584. URL: http://arxiv.org/abs/1412.7584.

Johansson, Erik M, Dowla, Farid U, and Goodman, Dennis M. (1991). "Backpropogation Learning For Multilayer Feed-Forward Neural Networks Using The Conjugate Gradient Method". In: *International Journal of Neural Systems* 2.4, pp. 291–301.

Kass, G. V. (1980). "An Exploratory Technique for Investigating Large Quantities of Categorical Data". In: *ournal of the Royal Statistical Society: Series C (Applied Statistics* 29.2, pp. 119–127. ISSN: 00359254. DOI: 10.2307/2986296.

Kaya, Erdem et al. (2018). "Behavioral attributes and financial churn prediction". In: *EPJ Data Science* 7.1. ISSN: 21931127. DOI: 10.1140/epjds/s13688-018-0165-5. URL: http://dx.doi.org/10.1140/epjds/s13688-018-0165-5.

Kearns, Michael and Valiant, Leslie. (1994). "Cryptographic Limitations on Learning Boolean Formulae and Finite Automata". In: *Journal of the ACM (JACM)* 41.1, pp. 67–95. ISSN: 1557735X. DOI: 10.1145/174644.174647.

Littler, Sarah. (2020). *Cumulative Charts*. URL: https://select-statistics.co.uk/blog/cumulative-gains-and-lift-curves-measuring-the-performance-of-a-marketing-campaign/.

McCulloch, Warren S. and Pitts, Walter H. (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5, pp. 115–133.

Molnar, Christoph. (2019). *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. Pp. 1–247. URL: https://christophm.github.io/interpretable-ml-book.

Murphy, Kevin P. (2012). *Machine Learning: A Probabilistic Perspective*, pp. 1–1067. ISBN: 9780262018029. DOI: 10.1007/978-94-011-3532-0_2. URL: https://mitpress.mit.edu/books/machine-learning-1.

Nocedal, Jorge. (1980). "Updating Quasi-Newton Matrices with Limited Storage". In: *Mathematics of Computation* 35.151, pp. 773–782. ISSN: 00255718. DOI: 10.2307/2006193.

Probst, Philipp, Boulesteix, Anne Laure, and Bischl, Bernd. (2019). "Tunability: Importance of hyperparameters of machine learning algorithms". In: *Journal of Machine Learning Research* 20, pp. 1–22. ISSN: 15337928. arXiv: 1802.09596.

Quinlan, J. Ross. (1986). "Induction of decision trees". In: *Machine Learning* 1.1, pp. 81–106. ISSN: 0885-6125. DOI: 10.1007/bf00116251.

— (1987). "Simplifying decision trees". In: *International Journal of Man-Machine Studies* 27.3, pp. 221–234. ISSN: 10715819. DOI: 10.1006/ijhc.1987.0321.

Rahman, Md Geaur and Islam, Md Zahidul. (2011). "A decision tree-based missing value imputation technique for data pre-processing". In: *Proceedings of the Ninth Australasian Data Mining Conference-Volume 121*, pp. 41–50. ISBN: 9781921770029.

Ranganathan, Priya, Pramesh, C.S., and Aggarwal, Rakesh. (2018). "Common pitfalls in statistical analysis: Logistic regression". In: *Perspectives in Clinical Research* 8.3, pp. 148–151. DOI: 10.4103/picr.PICR. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6176693/?report=printable.

Ribeiro, Marco Tulio, Singh, Sameer, and Guestrin, Carlos. (2016)(a). "Model-Agnostic Interpretability of Machine Learning". In: Whi. arXiv: 1606.05386. URL: http://arxiv.org/abs/1606.05386.

— (2016)(b). "Why Should I Trust You? Explaining the Predictions of Any Classifier". In: arXiv: 1710.10720. URL: http://arxiv.org/abs/1710.10720.

Rokach, Lior and Maimon, Oded. (2015). *Data Mining with Decision Trees: Theory and Applications*. 2nd Edition. Vol. 81. World Scientific Publishing Co. Pte. Ltd., pp. 1–305. ISBN: 964-7445-88-1.

Rosenblatt, Frank. (1961). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Washington, DC: Spartan Books, pp. 1–621. DOI: 10.2307/2312103.

Rüping, Stephan. (2006). "Learning Interpretable Models". PhD thesis. Universit¨at Dortmund, p. 209. DOI: 10.1086/508674. URL: http://www.stefan-rueping.de/publications/rueping{_}2006b.pdf.

SAS Institute Inc. (2012). *SAS Enterprise Miner and SAS Text Miner Procedures Reference for SAS 9.2*.

— (2017). "SAS Visual Analytics 8.1: Working with SAS Visual Data Mining and Machine Learning". In: pp. 1–22.

Shah, Jainam D., Shah, Fenil D., and Rahevar, Mrugendra. (2018). "Customer Churn Prediction Analysis". In: *International Journal of Computer Applications* 182.29, pp. 15–17. DOI: 10.5120/ijca2018918145.

Shalev-Shwartz, Shai and Ben-David, Shai. (2014). *Understanding Machine Learning: From Theory to Algorithms*. New York, USA: Cambridge University Press, pp. 1–449. ISBN: 9781107298019. DOI: 10.1017/CBO9781107298019.

Shmueli, Galit. (2019). "Lift Up and Act! Classifier Performance in Resource-Constrained Applications". In: arXiv: 1906.03374. URL: http://arxiv.org/abs/1906.03374.

Vafeiadis, T. et al. (2015). "A comparison of machine learning techniques for customer churn prediction". In: *Simulation Modelling Practice and Theory* 55, pp. 1–9. ISSN: 1569190X. DOI: 10.1016/j.simpat.2015.03.003. URL: http://dx.doi.org/10.1016/j.simpat.2015.03.003.

Wu, Shaomin and Flach, Peter A. (2002). "Feature selection with labelled and unlabelled data". In: *Proceedings of ECML/PKDD'02 Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning*, pp. 156–167.

Zhang, Jue and Chen, Li. (2019). "Clustering-based undersampling with random over sampling examples and support vector machine for imbalanced classification of breast cancer diagnosis". In: *Computer Assisted Surgery* 24.sup2. PMID: 31403330, pp. 62–72. DOI: 10.1080/24699322.2019.1649074. eprint: https://doi.org/10.1080/24699322.2019.1649074. URL: https://doi.org/10.1080/24699322.2019.1649074.

Appendix A

Supplementary Information

A.1 Proofs

A.1.1 Chapter 3.1.1

Proof (Convexity of Loss Function):

According to the second order condition; a function f which is twice continuously differentiable is convex $\iff$ its Hessian matrix is positive semi-definite, i.e. $\forall z : z' \nabla_x^2 f(x) z \geq 0$.
We take a look at the gradient of the first part of the loss function:

$$\begin{aligned}\nabla_{\bar{\boldsymbol{b}}}[-\log(h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}}))] &= \nabla_{\bar{\boldsymbol{b}}}[\log(1+e^{-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle})] \\ &= \left(\frac{-e^{\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle}}{1+e^{-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle}}\right)\bar{\boldsymbol{x}}_{\boldsymbol{i}} = \left(\frac{1}{1+e^{-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle}} - 1\right)\bar{\boldsymbol{x}}_{\boldsymbol{i}} = (h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}}) - 1)\bar{\boldsymbol{x}}_{\boldsymbol{i}}.\end{aligned}$$

And now the hessian:

$$\nabla_{\bar{\boldsymbol{b}}}[Grad] = \nabla_{\bar{\boldsymbol{b}}}[\nabla_{\bar{\boldsymbol{b}}}[-\log(h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}}))]] = \nabla_{\bar{\boldsymbol{b}}}((h_{\bar{\boldsymbol{b}}}(x) - 1)\bar{\boldsymbol{x}}_{\boldsymbol{i}}) = h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}})(1 - h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}}))\bar{\boldsymbol{x}}_{\boldsymbol{i}}\bar{\boldsymbol{x}}_{\boldsymbol{i}}{}'.$$

Hence $z'[h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}})(1 - h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}}))\bar{\boldsymbol{x}}_{\boldsymbol{i}}\bar{\boldsymbol{x}}_{\boldsymbol{i}}{}^{\top}]z = h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}})(1 - h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}}))(\bar{\boldsymbol{x}}_{\boldsymbol{i}}{}^{\top} z)^2 \geq 0$. We analyse the gradient of the second part of the loss function:

$$\begin{aligned}\nabla_{\bar{\boldsymbol{b}}}[-\log(1 - h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}}))] &= \nabla_{\bar{\boldsymbol{b}}}[(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle + \log(1+e^{-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle}))] \\ &= \bar{\boldsymbol{x}}_{\boldsymbol{i}} + \nabla_{\bar{\boldsymbol{b}}}[\log(1+e^{-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle})].\end{aligned}$$

$$\nabla_{\bar{\boldsymbol{b}}}[Grad] = \nabla_{\bar{\boldsymbol{b}}}[\nabla_{\bar{\boldsymbol{b}}}[-\log(h_{\bar{\boldsymbol{b}}}(\bar{\boldsymbol{x}}_{\boldsymbol{i}}))]] = \nabla_{\bar{\boldsymbol{b}}}[\bar{\boldsymbol{x}}_{\boldsymbol{i}} + \nabla_{\bar{\boldsymbol{b}}}[\log(1+e^{-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle})] = \nabla^2_{\bar{\boldsymbol{b}}}[\log(1+e^{-\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle})],$$

which was proven to be semi-definite above. Hence, the statement is proven since the positive linear combination of convex functions is convex.

A.1.2 Chapter 3.2.3

Proof (Equivalence of constrained problem for LASSO and Ridge regression):
The proof of constrained and unconstrained form equivalence can be achieved by showing that $\bar{\boldsymbol{b}}$ values of each are equal. Given the general optimisation problem

$$min_{x\in\mathbb{R}^n} f(x) \text{ s.t. } h_i(x) \leq 0,\ i = 1, \ldots, m,$$

the first and second KKT conditions needed for the equivalence proof are

$$\partial f(x) + \sum_{i=1}^{m} u_i \partial h_i(x) = 0 \quad \text{(Stationarity)}, \tag{A.1}$$

$$u_i h_i(x) = 0\ \forall i\ \in\ \lfloor m \rfloor \quad \text{(Complementarity)}. \tag{A.2}$$

Knowing that the KKT method minimises functions subject to inequality, we replace the constrained form by the following equivalent unconstrained form :

$$\min_{\bar{\boldsymbol{b}}} \sum_{i=1}^{N} [y_i(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle) - \log(1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle))] - \theta(\sum_{j=1}^{M} {b_j}^2 - t). \tag{A.3}$$

By showing that $\lambda = \theta$, we show the equivalence of $\bar{\boldsymbol{b}}$ values.

The stationary condition of the KKT method (see Equation (A.1)) gives us the information that the gradient of the Lagrangian equals 0 with respect to $\bar{\boldsymbol{b}}$.
We can show that the differentiation in both forms depends on the same terms by considering the extended form of Equation (A.3), namely:

$$\min_{\bar{\boldsymbol{b}}} \sum_{i=1}^{N} [y_i(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle) - \log(1 + \exp(\langle \bar{\boldsymbol{b}}, \bar{\boldsymbol{x}}_{\boldsymbol{i}} \rangle))] - \theta(\sum_{j=1}^{M} {b_j}^2) + \theta(t).$$

That is the first two terms depending on $\bar{\boldsymbol{b}}$ are equivalent to the terms of the unconstrained form and are those relevant for the gradient.

Secondly, the complementarity condition of KKT (see Equation (A.2)) provides the following statement:

$$\theta(\sum_{j=1}^{M} {b_j}^2 - t) = 0. \tag{A.4}$$

Equation (A.4) indicates that $\bar{\boldsymbol{b}}$ minimising the unconstrained form must be $t = \sum_{j=1}^{M} {b_j}^2$. The constrained form holds as the equivalence is proven.

The exact same proof steps hold for the LASSO method by replacing the norm by $\sum_{j=1}^{M} |b_j|$.

A.2 Case Study

A.2.1 Data Description

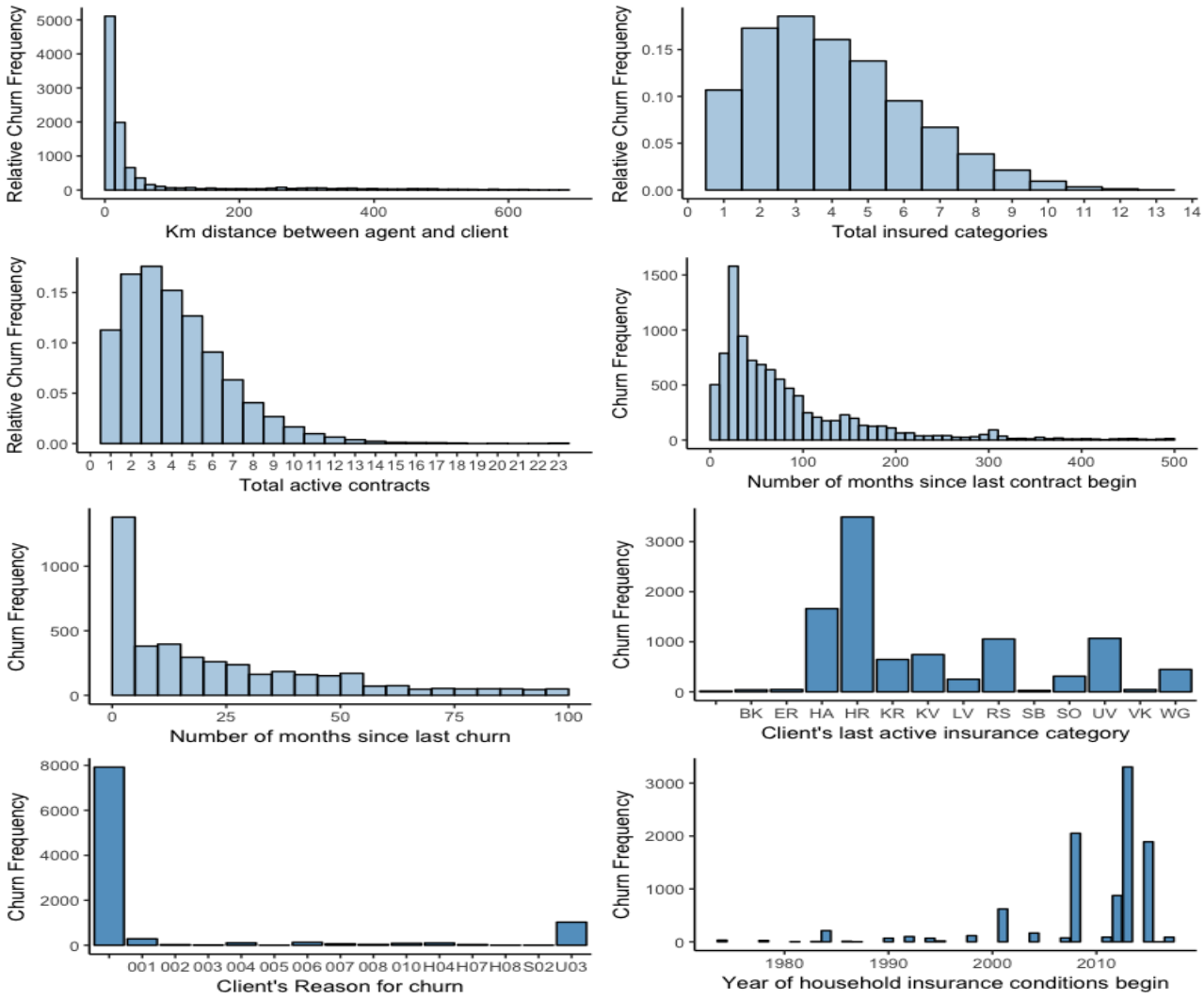


Figure A.1: Extended Features' Exploratory Analysis

A.2.2 Dimensionality Reduction

LASSO Selection
CL_Doc_Reas_A1
Churn_3Mon
Churn_3Mon_No_CI
UP_ZKHASUM_S_1
Kgs12_Prop_Hs
AG_Contr_Prop_A_Beg_Dat_1Y
HP_BAK_JAHR_A_1
ALLE_ROLLE_VERLPERS_ANZ
Churn_3Mon_HH
Loc_Chg_3Mon
KA
VM_AEND_VM_ZUSTAE_VERTR_1J
AG_Contr_Prop_A_End_Dat_1Y
Chur_3Mon_AI
HP_ORG_SL_A1
AG_Chg_Resp_OE_Pnr_1Y
SO
Churn_3Mon_HA
Delta_No_Cat_3Mon
Churn_Last_CS_Cat

Table A.1: LASSO - Selected Input Variables

Variable Abbreviation	Meaning
Active_Passive_24Mon_HH	Active or Passive household last 24 months.
Age	Age
AG_Chg_Resp_OE_Pnr_1Y	Change of responsible agent in last year.
AG_Contr_Prop_A_Beg_Dat_1Y	Agent's contracts proportion vs. other agent w.r.t. initial date within a year.
AG_Contr_Prop_A_End_Dat_1Y	Agent's contracts proportion vs. other agents w.r.t. end date within a year.
AG_No_Contr	Agent's number of resp. contracts.
AG_No_Contr_Beg_Dat_1Y_HH	Agent's number of HH contracts with start date in the last year.
Churn_3Mon	Customer churn occurence or non-occurence in last 3 months.
Churn_3Mon_No_CI	Number of cancelled car insurance contracts in last 3 months.
Churn_First_No_Mon	Number of months since first churn.
Churn_Last_No_Mon	Number of months since last churn.
Churn_Last_CS_Cat	Last churned cross-selling category.
Churn_No_Contr_6Mon	Number of churned contracts in the last 6 months.
Churn_No_Contr_1Y	Number of churned contracts in the last year.
Churn_No_Contr_3Y	Number of churned contracts in the last 3 years.
CL_Doc_Reas_A1	Reasons for C&L first active contract changes.
CL_PCL_Typ _A1	Private casualty & libility type of first active contract.
CL_Ph_Adm_Sys_Key_A1	Policy holder's adm.sytem key of first active C&L contract.
Delta_Kompo_Contr_3Mon	Delta of number of KOMPO contracts in last 3 months.
Delta_No_Cat_3Mon	Delta of number of insured categories in last 3 months.
HH_EndDate_Cat	End date category of HH contract.
HH_Ins_Cond_Year_A1	Year of household insurance conditions begin, first active contract.
HH_Contr_Chg_Reas_A1	Reasons for household first active contract changes.
HH_Ph_State_Key_A1	Policy holder's state code in first active household contract.

Table A.2: Meaning of features' abbreviations

Meaning	English Abbrev. VarIables
Id_HH	Client ID refers to household/family.
Int_Marketing	Initial marketing.
Job_Pos	Job position.
Kgs12_Geocluster	Geo Cluster of kgs12.
Kgs12_Prop_Hs	Proportion of kgs 12 with high school diploma.
Kgs12_Rentalscore	Rental score of kgs12.
Kgs16_Geocluster	Geo Cluster of kgs16.
Kgs16_Lstage	Life stage of kgs16.
Kgs16_Rentalscore	Rental score of kgs16.
Last_Agcy_Active	Last active agency.
Loc_Chng_3Mon	Location change occurence or non-occurence in last 3 months.
No_Active_Contr	Total number of active contracts.
No_Cars_HH	Number of household's cars.
No_Mon_First_Contr_Beg	Number of months since first contract start.
No_Mon_First_Contr_Beg_HH	Number of months since first HH contract start.
Resp_OE_Pnr	ID number of responsible Agent.

Table A.3: Meaning of features' abbreviations (continued)

A.2.3 SEM Implementation Remarks

As SAS Enterprise Miner delivers a user-friendly interface with no visible details of the exact procedure triggered behind the nodes, we list a few remarks here concerning the implemented settings and the procedures run with reference to SAS Institute Inc. (2012). Regarding the technical implementation of random forest in SEM, the node *'HP Random Forest'* was used for classification. Within SEM's node, the *'PROC HPForest'* procedure is triggered with the following adjustable settings in the statements. By setting the train proportion to $1/n$, the INBAGFRACTION $= 1/n$ option in the statement is set to specify the number of observations to sample without replacement into the bagged data. The procedure randomly selects m candidate input variables independently in every node, where m is the by the user predetermined value of the VARS_TO_TRY = option in the statement. The procedure computes the average square error measure of prediction error, the misclassification rate, and the log-loss. Given an observation, the procedure follows the tree-paths and assigns the observation to a single leaf in each decision tree. Using the generated posterior probability predictions, (i.e. the proportion of that class among the bagged training observations in that leaf) of the respective tree containing the leaf, the classification is forecasted by assigning the class with the highest posterior probability. The decision trees trained by the procedure are generated using recursive binary splitting, with splits seeking to maximize Gini reduction for a nominal target and variance reduction for an interval target.

Automated hyperparameter tuning of number of trees and variables, is applied through the following code:

```
%macro hpRFTuning (noVars = 10,maxTrees=200);

  %let noTries = %sysfunc(countw(& noVars.));

  %do k = 1 %to & noTries.;
    %let Try = %sysfunc(scan(& noVars.,&k));

    proc hpforest data=&EM_IMPORT_DATA maxtrees=&maxTrees.
    var_to_try=&Try.;
      input %EM_INTERVAL_INPUT /level=interval;
      input %EM_ORDINAL_INPUT /level=ordinal;
      input %EM_NOMINAL_INPUT /level=nominal;
      input %EM_BINARY_INPUT /level=binary;
      target %EM_TARGET / level=binary;
      ods output fitstatistics = fitstats_vars&Try.;
    run;

```

```
    data fitstats_vars&Try.;
      length varsToTry $ 8;
      set fitstats_vars&Try.;
      varToTry = " &Try. " ;
    run;

    proc append base=fitStats data=fitstats_vars&Try. force;
    run;
  %end;

%mend hpRFTuning;

% hpRFTuning(noVars =5 10 15 20 25 30 35 40 45 50 all,maxTrees=200);

%em_register(type=Data,key=fitStats);

data &em_user_fitStats;
  set fitStats;
run;

%em_report(viewType=data,key=fitStats,autodisplay=y);
%em_report(viewType=lineplot,key=fitStats,x=nTrees,y=miscOOB,group=varsToTry,
description=Out of Bag Misclassification Rate,autodisplay=y);
```

Using the 'Gradient Boosting' node in SEM similarly invokes the *'PROC TreeBoost'* procedure, which implements the algorithm described by Friedman (2001). Adjustable settings in the node include, for example the number of iterations for the boosting series. Considering that the triggered procedure uses decision trees as base learners for the boosting ensemble, the number of iterations is equal to the number of trees. The user can also set the MAXDEPTH representing the maximum tree depth allowed where the procedure can continue searching for new splitting rules (see SAS Institute Inc. (2012)):

```
PROC TREEBOOST <options>;
ASSESS <options>;
CODE <options>;
DECISION DECDATA=data-set-name <options>;
FREQ variable;
IMPORTANCE <options>;
INPUT list-of-variables <options>;
MAKEMACRO NTREES=macro-name;
PERFORMANCE <options>;
SAVE <options>;
SCORE <options>;
SUBSERIES <options>;
TARGET variable <options>;  RUN;
```

To implement a linear or logistic regression in SEM, the 'DMREG' procedure is invoked by the 'Regression' node to either predict the numerical value of the continuous variable of interest or to predict the probability that a categorical target will take on one of the possible classes. The procedure consists of several statements, each including different adjustable parameters and has the following outline (see SAS Institute Inc. (2012)):

```
PROC DMREG DATA=data-set-name DMDBCAT=catalog-name <options>;
CLASS list-of-variables;
CODE <options>;
DECISION DECDATA=data-set-name <options>;
FREQ variable;
MODEL target-variable=input-variables </ options>;
NLOPTIONS <options>;
PERFORMANCE <options>;
REMOTE <options>;
SCORE <options>;
QUIT;
```

The MODEL statement, most importantly, sets the target variable in terms of the required input variables, as well as the fitting characteristics. The optimization methods available for computing are chosen by specifying the TECHNIQUE to NEWRAP (Newton-Raphson Optimization with Line Search), NRRIDG (Newton-Raphson RidgeSyntax Optimization), or QUANEW (Quasi-Newton Optimization), and many more. Furthermore, SELECTION options assign, e.g. the model selection criterion, wrapper methods direction if desired, and a maximum number of steps in the selection process. At each step of the model construction, the procedure will choose the model with best-specified criterion values such as CHOOSE = AIC, CV-ASE, or VERROR. The different wrapper methods correspond to SELECTION= Backward, Forward, None or Stepwise.

The alternative LASSO method first computes the estimates by running a logistic regression through PROC LOGISTIC and feeding these estimates into the PROC GLMSELECT. A further logistic regression run is performed at the end to transform the output to a predicted probability between 0 and 1.

The 'Neural Network' node makes use of the PROC Neural procedure does not use all of its aspects however. It is important to note that the statement includes many optional settings that we do not apply and refer to SAS Institute Inc. (2012) for details. Most important components of the PROC Neural include the ARCHITECTURE, HIDDEN, and NETOPTIONS statements. The architecture statement specifies the required neural network architecture, for instance a GLM, MLP or Ordinary Radial Basis Function Network. The hidden statement sets the number of hidden units or neurons which are connected in a forward-manner by the CONNECT statement. NETOPTIONS include

adjustable hyperparameters like (DECAY) weight decay, (OBJECT) loss function used by the network and RANDIST for the choice of distribution to generate random initial weights, like the Cauchy, Normal or Uniform distributions.

The technical functionalities of the used neural network optimisers will also be shortly introduced here. The standard BackProp computes the weight update at each iteration as explained in Section 3.1.5. Using the PDETAIL option in PROC NEURAL procedure, one can assign accordingly QPROP or BPROP. The outputs of PDETAIL for BackProp are the quantities measuring previous weight change, current weight change, gradient, previous and current weight. As for QPROP, SEM computes Equation (A.5) by computing the following alternative, as it can become large or move away from the minimum:

$$\alpha_{l,r,s}^{k} = \frac{\triangle \mathcal{L}_{l,r,s}^{(k)}}{\triangle \mathcal{L}_{l,r,s}^{(k-1)} - \triangle \mathcal{L}_{l,r,s}^{(k)}}. \tag{A.5}$$

$$\alpha^{k} = \begin{cases} \alpha_{\max} & \text{for } |\alpha_{l,r,s}^{k}| > \alpha_{\max}, \\ \alpha_{\max} & \text{for } (\triangle \mathcal{L}_{l,r,s}^{(k-1)} - \triangle \mathcal{L}_{l,r,s}^{(k)}) \cdot \triangle \mathcal{W}_{l,r,s}^{(k-1)} > 0, \\ \alpha_{l,r,s}^{k} & \text{otherwise.} \end{cases}$$

Considering the above computation, the output by QPROP in SEM includes addditional to BPROP's standard outputs, the quantities of the modified gradient and α^k.

Finally, we include some remarks about the procedures used by SAS Viya for the generation of partial dependence plots and LIME models. The 'partialDependence' procedure generates a function averaging the model prediction for each value of a specific feature. Assigning a model, input dataset and predicted probabilities to the action 'partialDependence' produces the plot using PROC SGPLOT. The second available action in SAS Viya, is the 'linearExplainer'. The procedure contains PROC CAS code to run the adjusted settings. By assigning an input dataset to the statement and the trained black-box model to 'modelTable', we can run the procedure 'explainModel' to determine the LIME model. The 'preset' parameter specifies that the LIME explanation method should be used.

```
proc cas;
loadactionset "explainModel";
explainModel.linearExplainer / table = ""
query            = "QUERY"
modelTable       = ""
modelTableType   = "ASTORE"
predictedTarget  = "P_target"
seed             = 1234
```

```
preset              = "LIME"
inputs              = {}
nominals            = {}
;
run;
quit;
```

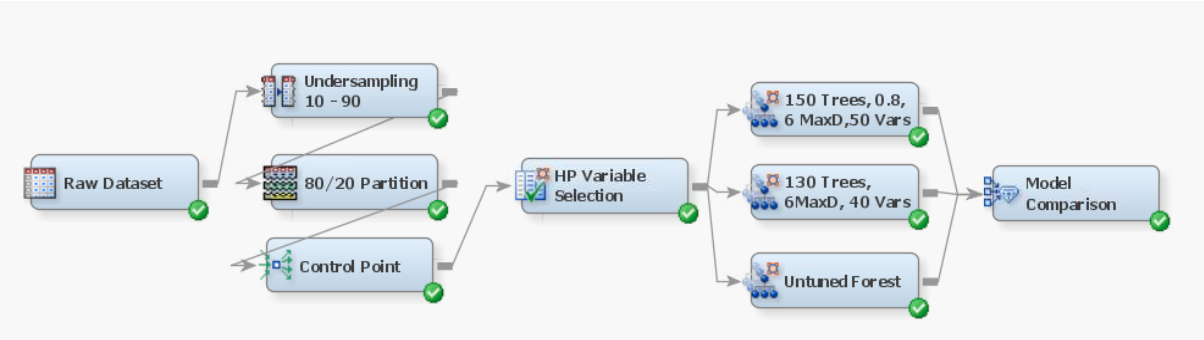


Figure A.2: Flowchart Of Random Forest Modelling In SEM

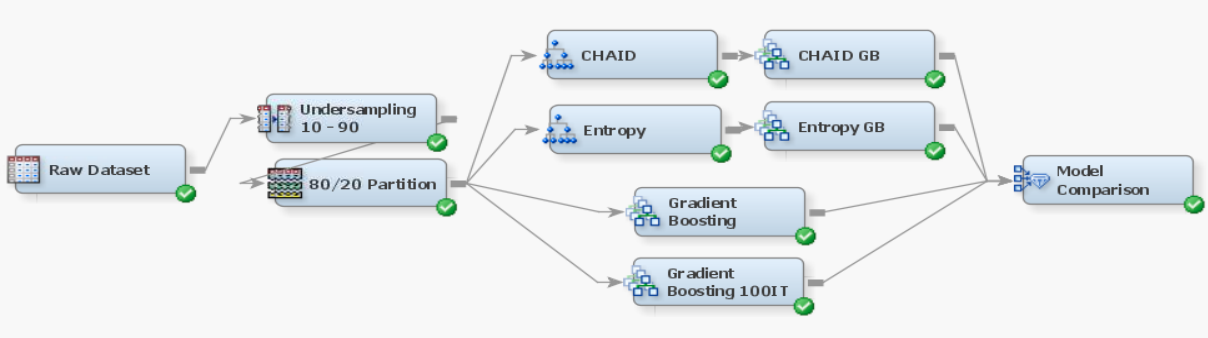


Figure A.3: Flowchart Of Gradient Boosting Modelling In SEM

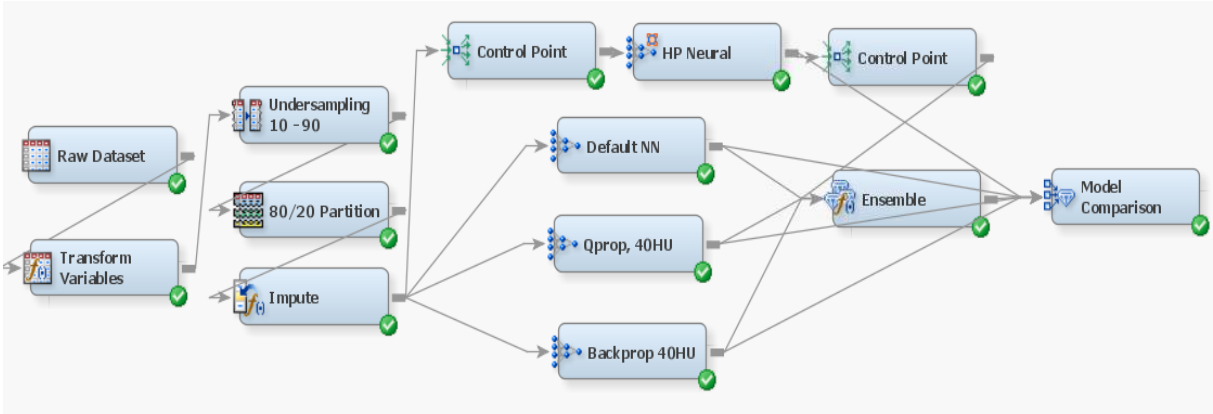


Figure A.4: Flowchart Of Neural Network Modelling In SEM

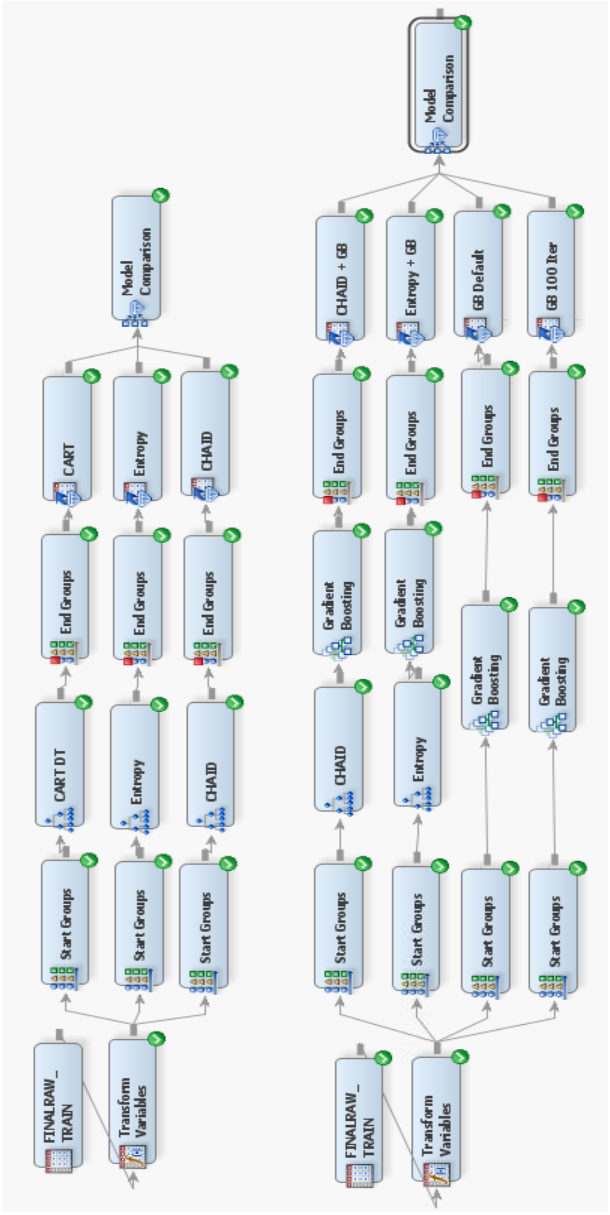


Figure A.5: Flowchart Of Cross Validation In SEM

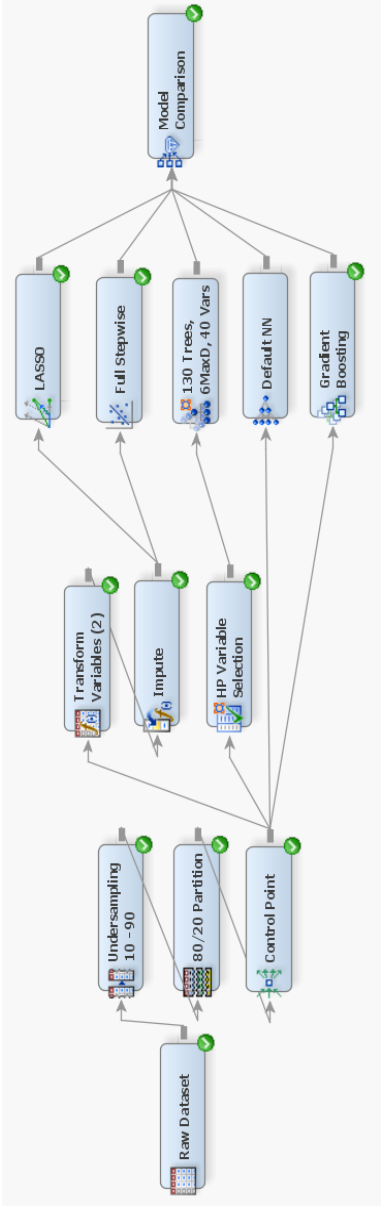


Figure A.6: Flowchart Of All Models In SEM

A.2.4 Results

Variable Entered	DF	Wald χ^2	Pr $>\chi^2$	AIC
Churn_3Mon	1	209.22	<.0001	50414.2
AG_Chg_Resp_OE_Pnr_1Y	1	299.45	<.0001	49848
Churn_Last_CS_Cat	10	182.72	<.0001	49270.9
No_Mon_First_Contr_Beg	1	48.91	<.0001	48755.7
Delta_Kompo_Contr_3Mon	1	178.38	<.0001	48455.9
AG_Contr_Prop_A_End_Dat_1Y	1	130.06	<.0001	48277.5
Churn_Last_No_Mon	1	162.46	<.0001	48102
SB_HH	1	60.60	<.0001	47931.9
Age	1	124.61	<.0001	47789.9
Kgs12_Prop_Hs	1	9.45	0.0021	47673.6
KKM_A_P_Quelle	2	90.78	<.0001	47575.1
CL_Doc_Reas_A1	7	97.52	<.0001	47483.9
Loc_Chg_3Mon	1	76.36	<.0001	47410.6
Job_Pos	13	72.68	<.0001	47328
Active_Passive_24Mon_HH	1	40.83	<.0001	47273
Int_Marketing	9	71.92	<.0001	47218.1
HH_Ins_Cond_Year_A1	21	113.30	<.0001	47172.2
HH_Ph_State_Key_A1	17	76.70	<.0001	47122.6
HH_EndDate_Cat	3	40.64	<.0001	47085.8
No_Active_Contr	1	34.85	<.0001	47055.3
Kgs16_Rentalscore	8	50.22	<.0001	47015.8
Last_Agcy_Active	1	22.92	<.0001	46997.9
ANZ_VERTR_V_END_DAT_HH_3J	1	19.15	<.0001	46981.5
Churn_3Mon_No_CI	1	20.29	<.0001	46963
ANZ_LETZ_END_CS_SP	12	42.79	<.0001	46941.9
Dup4	1	19.67	<.0001	46925.3
Resp_OE_Pnr	1	14.72	0.0001	46911.5
Last_Adm_Sys_Active	14	46.34	<.0001	46895.8
LV_TAGE_V_A1	1	15.69	<.0001	46882
MyERGO	1	15.11	0.0001	46869.4

Table A.4: CHAID Stepwise Logistic Regression - Variables' Statistical Significance

Variable Entered	DF	Wald χ^2	Pr $>\chi^2$	AIC
Churn_3Mon	1	212.29	<.0001	50414.2
AG_Chg_Resp_OE_Pnr_1Y	1	295.82	<.0001	49848
Churn_Last_CS_Cat	10	182.28	<.0001	49270.9
No_Mon_First_Contr_Beg	1	72.12	<.0001	48755.7
Delta_No_Cat_3Mon	1	34.26	<.0001	48452.3
AG_Contr_Prop_A_End_Dat_1Y	1	147.63	<.0001	48270.7
No_Cars_HH	1	193.42	<.0001	48039.5
Churn_Last_No_Mon	1	157.19	<.0001	47854.7
Age	1	138.99	<.0001	47712.6
Kgs12_Prop_Hs	1	9.90	0.0017	47581.5
CL_Doc_Reas_A1	7	93.20	<.0001	47488.7
KKM_A_P_Quelle	2	113.90	<.0001	47400.6
Loc_Chg_3Mon	1	71.61	<.0001	47329
Job_Pos	13	72.98	<.0001	47241.1
AG_CONTR	1	63.63	<.0001	47184.3
Active_Passive_24Mon_HH	1	40.13	<.0001	47144
HH_Ins_Cond_Year_A1	21	132.41	<.0001	47094.7
No_Contr_End_Dat_HH_3Y	1	29.07	<.0001	47060.5
HH_Ph_State_Key_A1	17	83.25	<.0001	47011.2
Kgs16_Rentalscore	8	57.43	<.0001	46968.9
Churn_3Mon_No_CI	1	24.05	<.0001	46939.9
CL_Ph_Adm_Sys_Key_A1	1	29.84	<.0001	46908.8
HH_EndDate_Cat	3	35.69	<.0001	46877.7
AG_Contr_Prop_A_Beg_Dat_1Y	1	24.98	<.0001	46846.2
VP_in_LV	1	26.19	<.0001	46821.4
Delta_Kompo_Contr_3Mon	1	25.91	<.0001	46796.4
Id_HH	2	56.83	<.0001	46789
AG_Contr_Prop_A_End_Dat_1Y_DY	1	18.92	<.0001	46770.7
Last_Agcy_Active	1	22.88	<.0001	46753.1
HH_Contr_Chg_Reas_A1	9	40.80	<.0001	46731

Table A.5: Stepwise Logistic Regression - Variables' Statistical Significance

Variable	Effect	Parameter Estimate	Standard Error
Last_Adm_Sys_Active	TR	-5.13	35.17
Last_Adm_Sys_Active	IT	-4.07	49.43
Kgs12_Prop_Hs		-1.15	0.37
CL_Doc_Reas_A1	14	-1.01	0.91
HH_Ins_Cond_Year_A1	2016	-0.93	0.98
Intercept		-0.88	4.83
Int_Marketing	Bank	-0.59	0.20
Delta_Kompo_Contr_3Mon		-0.54	0.04
Job_Pos	?	-0.41	0.07
Loc_Chg_3Mon	0	-0.36	0.04
Churn_Last_CS_Cat	LV	-0.33	0.04
Churn_3Mon_No_CI		0.62	0.14
Churn_3Mon	Yes	0.65	0.04
Last_Adm_Sys_Active	SV	0.86	4.32
HH_Ph_State_Key_A1	13	0.91	2.14
CL_Doc_Reas_A1	60	1.07	0.89
CL_Doc_Reas_A1	62	1.48	0.26

Table A.6: CHAID Stepwise Logistic Regression - Parameter Estimates

Variable	Effect	Parameter Estimate	Standard Error
Id_HH	1	-4.49	0.60
AG_Contr_Prop_A_Beg_Dat_1Y		-2.06	0.41
CL_Doc_Reas_A1	14	-1.39	0.97
Kgs12_Prop_Hs		-1.17	0.37
HH_Ins_Cond_Year_A1	1981	-0.99	0.71
Job_Pos	?	-0.39	0.07
Loc_Chg_3Mon	0	-0.35	0.04
Churn_Last_CS_Cat	LV	-0.31	0.04
Delta_No_Cat_3Mon		-0.31	0.05
Kgs16_Rentalscore	1	0.24	0.06
Churn_Last_CS_Cat	HA	0.24	0.07
AG_Contr_Prop_A_End_Dat_1Y		0.60	0.05
Churn_3Mon_No_CI		0.65	0.13
Churn_3Mon	Yes	0.65	0.04
HH_Ph_State_Key_A1	13	0.95	2.21
HH_Contr_Chg_Reas_A1	60	1.01	0.56
HH_Ins_Cond_Year_A1	2016	1.32	1.05
CL_Doc_Reas_A1	62	1.66	0.27
Intercept		2.86	2.23

Table A.7: Stepwise Logistic Regression - Parameter Estimates