Tree-based machine learning methods with non-life insurance applications

Henning Zakrisson
Tree-based machine learning methods with non-life insurance applications

Henning Zakrisson

Abstract
Non-life insurance is a field which has been data-driven for a long time, with the statistical framework behind modern-day actuarial sciences laid out at the beginning of the 20th century. Problems regarding the estimation and prediction of risk are relevant to the insurance industry specifically, but also for society as a whole. The rise of machine learning methods has created a new set of tools that can be used to solve these problems. This thesis contains five individual papers, all of which are related to developing machine learning- or data-driven methods and algorithms that can be applied to, but are not limited to, non-life insurance applications.

Paper I takes an existing probabilistic model for claims reserving, the Collective Reserving Model (CRM), and replaces the linear modeling approach of the original paper with non-linear machine learning methods. The paper addresses issues in these applications and provides a framework for how to implement and evaluate machine learning models in a reserving setting. It also discusses how to implement early stopping methods given different levels of data granularity. The models are evaluated on a series of simulated data sets with promising results.

Paper II does not use a machine learning method per se but instead develops the CRM used in Paper I by adding the openness status of the claims to the dynamics and presents the CRM with Openness (CRMO), as a means to model the non-linear effects implied in Paper I. The paper presents how the model can be estimated using regression methods, and provides recursive formulas for the moments of the predicted reserve. The algorithm is evaluated in terms of accuracy on the same data set as in Paper I and shows results that are comparable to the machine learning implementations of the CRM model.

Paper III presents a new boosting algorithm called the Cyclic Gradient Boosting Machine (CGBM). The algorithm extends the classical gradient boosting machine to provide multi-dimensional function approximation. The paper shows how the CGBM can be used to estimate entire probability distributions rather than just the mean of the distribution. The paper also discusses potential problems with hyperparameter tuning in this higher-dimensional hyperparameter space and provides a dimension-wise early stopping method, which is proven useful to avoid overfitting. Numerical illustrations show accurate results on simulated and real data sets.

Paper IV is a paper that is not directly related to non-life insurance but rather to so-called decision trees used for classification and regression. The paper presents the trinary tree algorithm, which is a new way to handle missing input data for tree-based models, meant to provide a more regularized model than other suggested methods. The algorithm is benchmarked against standard methods for missing data-handling and shows promising results even for high rates of missing data.

Paper V presents a generalized linear model with non-linear effects induced by varying coefficients, with the varying coefficients estimated using the CGBM from Paper III. This is a special case of a varying coefficient model (VCM). The model can handle highly non-linear effects while maintaining local interpretability. The paper also shows how tuning, feature selection, and evaluation of interaction effects can be simplified as compared to other VCMs. The model is evaluated on the same data set as in Paper III and shows promising results in terms of accuracy and interpretability.
TREE-BASED MACHINE LEARNING METHODS WITH NON-LIFE INSURANCE APPLICATIONS

Henning Zakrisson
Tree-based machine learning methods with non-life insurance applications

Henning Zakrisson
Till mamma och pappa
List of Papers

This thesis is based on five papers, numbered I–V, which are included in the following order.


Authors’ contributions: In Paper I, Henning Zakrisson contributed to out-lining the experiment as well as writing the theory part on the machine learning models used.

In Paper II, Henning Zakrisson contributed to the creation of the statistical model, based on an idea by M. Lindholm, and to the writing of the paper. All numerical implementations were performed by Henning Zakrisson.

In Paper III, Henning Zakrisson created the algorithm based on an idea by M. Lindholm, and contributed to the writing of the paper. The numerical illustration and evaluation was designed mainly by Henning Zakrisson, and all code implementations were performed by Henning Zakrisson, as was the python package *cyc-gbm* which was released as open-source software in connection with the paper.

To Paper V, Henning Zakrisson contributed to the writing of the paper. The idea behind the usage of the algorithm from Paper III in this setting was
suggested by M. Lindholm. The theoretical development was conducted equally by both authors. All code implementations, including the open-source python package local-glm-boost, were performed by Henning Zakrisson.
Acknowledgements

First and foremost, I would like to thank my supervisor and friend, Mathias Millberg Lindholm, for your never-ending support, your creative and brilliant mind, and your empathy. You have been and will always be a true inspiration in everything from science and work to style and life choices. I will forever cherish the memories of all the hours we have spent arguing in front of a blackboard.

I want to thank my co-supervisor, Filip Lindskog, who helped in increasing the quality of this thesis through his insightful comments. Knowing that your door was always open has been a great comfort over the years. I would also like to thank my other teaching colleagues, Tom Britton and Daniel Ahlberg. They say the best way to learn something is to teach it, and teaching alongside great teachers is obviously even better.

I would like to thank my fellow PhD students, and some in particular. Vilhelm Niklasson—a steady support since our first day at Kräftriket. Lina Palmborg for being a sound voice to discuss with in the office. Felix Wahl, for helping me find my footing as a new and confused PhD student by explaining your thesis during an out-of-breath run around Brunnsviken. Måns Karlsson for being supportive when research was a slog and for being fun to be around when life was good. Fanny Bergström for providing me with other perspectives. Marina Herrera Sarrias for teaching me very little Spanish. Taariq Nazar for all the thought-provoking discussions and brainstorming sessions by our whiteboard.

I would like to thank all my friends, and especially some who have contributed to making this journey easier. Martin Johansson, Niclas Hell and Viktor Flodin—my academic support group, which has given me confidence, perspective and great memes. Johan Bejmar and Johan Hedin for supporting my life decisions, making me laugh and listening to me whine.

I would like to thank my family. Linnea Lundberg for your eternal patience and for making me the happiest man I can be. Agnes Jensen for inspiring me to become an engineer like my big sister. Gustav and Viva Jensen for making our family larger and happier. My mother, Susanne Nordlund, who holds a PhLic in chemistry from Stockholm University, forcing me to go this extra mile to be the family member with the highest academic degree. And my father, Jörgen Zakrisson, who early on taught me the greatest piece of knowledge there ever was to know about the field of mathematics—that it is fun.
Contents

Abstract i
List of papers ix
Acknowledgements xiii

I Introduction 1

1 Motivation 3

2 Theoretical introduction 7
  2.1 Exponential families 7
  2.1.1 Exponential dispersion families 10
  2.2 Convex optimization 11

3 Machine learning regression 15
  3.1 The regression problem 15
  3.2 Model classes 17
  3.3 Training and overfitting 19
  3.4 Model interpretation 21
  3.5 Model class examples 22
    3.5.1 Generalized linear models 22
    3.5.2 Classification and regression trees 23
    3.5.3 Gradient boosting machines 32

4 Non-life insurance applications 43
  4.1 Reserving 44
  4.2 Premium calculation 49

5 Overview of papers 51
  5.1 Paper I 51
  5.2 Paper II 52
  5.3 Paper III 53
  5.4 Paper IV 56
  5.5 Paper V 59
Sammanfattning på svenska
Bibliography

II Papers
   Paper I .......................................................... 69
   Paper II ........................................................... 119
   Paper III ......................................................... 149
   Paper IV ......................................................... 169
   Paper V .......................................................... 177
Part I

Introduction
Chapter 1

Motivation

The concept of regression is central to this thesis. Regression can, widely speaking, be defined as the process of estimating the relationship between a dependent variable, or response, and one or more independent variables, or features. This is done in order to perform either inferential or predictive tasks. Inference is the process of understanding the relationship between the response and the features, while prediction is concerned with estimating unobserved responses given observed features. This thesis focuses mainly on the predictive aspect of regression, and the methods developed are meant to be used for predictive purposes.

A predictive regression model can be described as a function that, given some set of features \( x \), outputs a prediction \( \hat{y} \) of some, yet unobserved, response \( y \). The response can, for example, be the number of insurance claims a policyholder will make, the amount a customer will spend in a store, or the value of a house. The set of features can be anything from the age of a policyholder, the number of previous purchases, to the number of rooms, geospatial location, and the color of a house. Traditionally, analytically tractable statistical models such as linear regression, or generalized linear models, have been used for these tasks. These have the benefit of being easy to implement and interpret, and given the right assumptions, they can be used to make accurate predictions.

In times when data is abundant and complex, constructing these simple models in a way that captures the underlying structure of the data can be difficult and time-consuming. This has led to the development of more data-driven models, known as machine learning models. These models essentially learn the relationship between the features and the dependent variable from the data with limited manual intervention from the statistician using the model. Machine learning models are, however, commonly described as black-box models, with the underlying structure of the model being difficult to interpret.

While training and applying machine learning models is often straightforward, understanding the underlying theory is crucial for truly seeing the strengths
and weaknesses of the different models. Therefore, this thesis will start off by giving a theoretical background on relevant important concepts from statistics and optimization, in order to better understand the methods described later.

The most important model class for this thesis is the so-called decision tree—a deceivingly simple model that can be used for both regression and classification problems. Classification here refers to the process of predicting a categorical response, such as whether a customer will buy a product or not or what type of animal is in a picture. A decision tree is essentially a flowchart of questions, where every response leads to a new question, until a conclusion is reached. This logical process mimics the decision-making process of a human, and is therefore relatively easy to understand and interpret. A simple example of a decision tree is shown in Figure 1.1. Most of this thesis focuses on decision trees and more complex models that use decision trees as building blocks.

![Decision Tree Example](Image)

Figure 1.1: A decision tree that can be used to decide whether to have Runebergstårtta or not.

The main goal of the thesis has been to examine the toolbox of tree-based machine learning regression models, and bring a small set of new tools to the table. The work started out as being purely meant for non-life insurance applications, which refers to any type of insurance that is not life insurance. This includes property insurance and motor insurance, and the applications generally involve predicting the number of claims or the cost of claims, as well as their uncertainty. Such applications are made or mentioned in four of the five papers. During the writing of the thesis, the focus shifted towards developing and adjusting present machine learning algorithms to allow for more flexible, interpretable, and accurate models. Non-life insurance remains the main application area, but the methods developed are general and can be applied to a wide range of problems.

The introductory part of the thesis is structured as follows: Chapter 2 gives a theoretical background on important statistical concepts and presents the idea of convex optimization, which is used in the training of many machine learning
models. Chapter 3 defines what a machine learning regression problem is, and how such problems can be solved or approximated using different model classes. Chapter 4 presents the two most important non-life insurance applications for this thesis, and how the methods described in the previous chapter can be applied to these problems. Chapter 5 provides brief overviews of the five papers that make up the thesis. After the introductory part, the five papers are presented in Part II.
Chapter 2

Theoretical introduction

To better grasp the machine learning methods presented in the next chapter, it is useful to have some background knowledge about the underlying statistical frameworks. Therefore, the following chapter will first give a brief introduction to a certain family of probability distributions in Section 2.1. Thereafter, the concept of convex optimization will be introduced in Section 2.2.

2.1 Exponential families

Exponential families are a class of probability distributions which have several useful properties. The concept dates back to the 1930s (see e.g. Pitman (1936)), and there are many different sources and representations of the concept. The following presentation is mainly based on Sundberg (2019).

Consider a random variable \( Y \in \mathcal{Y} \) where \( \mathcal{Y} \) is a measurable space, which will here be assumed to be either the real line \( \mathbb{R} \) or the set of integers \( \mathbb{Z} \), or a subset thereof. Let \( F \) be the distribution of \( Y \), i.e.

\[
Y \sim F(\theta)
\]

where \( \theta \in \Theta \subseteq \mathbb{R}^d \) is a vector of parameters. The distribution \( F \) belongs to the exponential family if the joint probability density function or probability mass function \( f : \mathcal{Y}^n \rightarrow \mathbb{R}^+ \) of \( n \) independent observations \( y_i \in \mathcal{Y}, i = 1, \ldots, n \) of \( Y \), given parameters \( \theta \), can be written on the form

\[
f(y; \theta) = \frac{b(y)}{C(\theta)} \exp(\theta^T t(y)) \tag{2.1}
\]

where \( y = (y_1, \ldots, y_n)^T \) is the vector of observations. The column vector \( \theta = (\theta_1, \ldots, \theta_d)^T \) is then referred to as the canonical parameter vector and \( t(y) \) is the vector of sufficient statistics. Sufficiency here refers to the fact that
the sufficient statistics contain all information about the parameters $\theta$ that is contained in the data $y$. The function $C : \Theta \to \mathbb{R}^+$ is sometimes referred to as the normalizing constant and needs to fulfill

$$C(\theta) = \int b(y) \exp(\theta^T t(y)) dy$$  

(2.2)

for the density function to integrate to one.

An example of an exponential family is the gamma distribution. Using the $\alpha$-$\beta$ parametrization, its probability density function is

$$f(y; (\alpha, \beta)) = \prod_{i=1}^{n} \frac{\beta^\alpha}{\Gamma(\alpha)} y_i^{\alpha-1} \exp(-\beta y_i)$$

which can be rewritten on the form of (2.1) by setting

$$C(\theta) = \Gamma(\theta_1)^n (-\theta_2)^{-n \theta_1}$$

and

$$b(y) = \prod_{i=1}^{n} y_i^{-1},$$

using canonical parameters $\theta = (\alpha, -\beta)^T$ and sufficient statistics

$$t(y) = \left( \sum_{i=1}^{n} \log y_i, \sum_{i=1}^{n} y_i \right)^T.$$

The probability density function for a single sample from a gamma distribution is shown in Figure 2.1a.

A discrete example of an exponential family is the Poisson distribution, with probability mass function

$$f(y; \lambda) = \prod_{i=1}^{n} \frac{\lambda^{y_i}}{y_i!} \exp(-\lambda)$$
which can be rewritten on the form of (2.1) by setting

\[ C(\theta) = \exp(n \exp(\theta)) \quad \text{and} \quad b(y) = \prod_{i=1}^{n} \frac{1}{y_i!} \]

and using canonical parameters and sufficient statistic

\[ \theta = \log \lambda \quad \text{and} \quad t(y) = \sum_{i=1}^{n} y_i. \]

The probability mass function for a single sample from a Poisson distribution is shown in Figure 2.1b.

Note that the sufficient statistics are not unique, nor is the choice of canonical parameters. An exponential family representation (i.e., a choice of \(C, b, t\) and \(\theta\)) is called \textit{minimal} if the sufficient statistic has the lowest possible dimension, which can be confirmed by making sure there is no linear dependence between the sufficient statistics or between the canonical parameter dimensions. A minimal representation is called \textit{full} if the parameter space equals the canonical space \(\Theta\), which means that it comprises all \(\theta\) for which the normalizing constant in (2.2) is finite. The canonical parameter space \(\Theta\) is a convex set (for the definition of a convex set, see Section 2.2). An exponential family with parameter dimension \(d\) is called regular if the canonical parameter space \(\Theta\) is an open subset of \(\mathbb{R}^d\).

For a regular exponential family, it holds that the vector of sufficient statistics \(t\) has the moments

\[ \mathbb{E}[t] = \nabla_{\theta} \log C(\theta) \quad (2.3) \]
\[ \text{Var}(t) = \nabla^2_{\theta} \log C(\theta) \quad (2.4) \]

where \(\nabla_{\theta}\) and \(\nabla^2_{\theta}\) are the gradient and Hessian operators with respect to \(\theta\) respectively. Note that \(\text{Var}(t)\) here refers to a covariance matrix in the case of multivariate sufficient statistics.

Estimating the parameters \(\theta\) for an observed data set \(y\) is a common problem in statistics. It is often done by maximum likelihood estimation (MLE), i.e. by finding the parameters \(\theta\) which maximize the (log) likelihood function

\[ \theta^* = \underset{\theta \in \Theta}{\text{argmax}} \, \mathcal{L}(y; \theta) \quad (2.5) \]

where \(\mathcal{L}\) here refers to the logarithm of the probability density function or probability mass function in (2.1). Exponential families are convenient for MLE estimation since the log-likelihood function given an observed data set \(y\) can be written on the form

\[ \mathcal{L}(y; \theta) = \theta^T t(y) - \log C(\theta) + \log b(y) \]

which is a smooth and strictly concave function of \(\theta\) and where one can note that the last term does not depend on \(\theta\). Finding the MLE is then a so-called...
**convex optimization problem**, which always has a unique solution. Convex optimization will be discussed further in Section 2.2.

**Remark 2.1.** Often, the canonical parameters are not the most intuitive parameters to work with, and more statistically meaningful parameters are desired, such as the mean and dispersion parameters. Other times, one wants to work with a parameter in a certain space, such as $\mathbb{R}^d$. One can consider an alternative parameterization $\psi \in \mathbb{R}^d$ by using some link function $h : \Theta \rightarrow \mathbb{R}^d$ and instead working with $\psi = h(\theta)$. Note, however, that this requires some caution regarding the concavity of the composition $L(y; h^{-1}(\psi))$ with respect to $\psi$, which is not always concave. It might be useful to first find the MLE in the canonical parameter space and then transform the result to the desired parameter space afterwards in order to make sure that the optimization problem in question is convex. See Section 2.2 for more on convex optimization.

Some distributions are exponential families in their one-parameter form but not in their two-parameter form (i.e., when two parameters are considered unknown). A notable example is the negative binomial distribution, which has the probability mass function

$$f(y; \theta) = \prod_{i=1}^{n} \frac{\Gamma(y_i + k)}{\Gamma(k)y_i!} \pi^k(1 - \pi)^{y_i}$$

In its one-parameter form, i.e. when $k$ is known and the parameter of interest is $\pi$, this can be rewritten on the form of (2.1) by setting

$$C(\theta) = (1 - e^\theta)^{-k} \text{ and } b(y) = \prod_{i=1}^{n} \frac{\Gamma(y_i + k)}{\Gamma(k)y_i!}$$

and using canonical parameter and sufficient statistic

$$\theta = \log(1 - \pi) \text{ and } t(y) = \sum_{i=1}^{n} y_i.$$  

However, if one considers the two-parameter form, i.e. when both $\pi$ and $k$ are considered unknown, then the probability mass function cannot be written in the form of (2.1). In fact, in that case, the log-likelihood function is not even concave, which makes finding the MLE a non-convex optimization problem (Barndorff-Nielsen 2014, p. 185).

### 2.1.1 Exponential dispersion families

Another relevant family of distributions are the exponential dispersion models. The term exponential dispersion model was introduced by Jørgensen (1997), but the idea is much older (Sundberg 2019, p. 181). In this text, the term exponential dispersion family will be used instead, not to confuse the term with
the regression models discussed in Chapter 3. The presentation follows Sundberg (2019, Chapter 9). A distribution with two parameters, \( \theta \) and \( \phi \), is an exponential dispersion family if the probability density function or probability mass function of one observation \( y \in \mathcal{Y} \subseteq \mathbb{R} \) of the random variable \( Y \) can be written as

\[
f(y; (\theta, \phi)) = m(y; \phi) \exp \left\{ \frac{\theta y - \log C(\theta)}{\phi} \right\}.
\]

One can note that if \( \phi = 1 \), this is a one-parameter exponential family with canonical parameter \( \theta \). Also, the transformed variable \( Y/\phi \), for a given \( \phi \), follows an exponential family with canonical parameter \( \theta \), and log-normalizing constant \( \log C(\theta)/\phi \). Using this together with (2.3)–(2.4) one can show that

\[
E[Y|\theta, \phi] = E[Y|\theta, \phi = 1] \text{ and } \text{Var}(Y|\theta, \phi) = \phi \text{Var}(Y|\theta, \phi = 1)
\]

where \( E[Y|\theta, \phi] \) and \( \text{Var}(Y|\theta, \phi) \) are the mean and variance of \( Y \) given some parameter values \( \theta \) and \( \phi \), whereas \( E[Y|\theta, \phi = 1] \) and \( \text{Var}(Y|\theta, \phi = 1) \) are the mean and variance of \( Y \) given parameter value \( \theta \) and a fixed value \( \phi = 1 \). This makes it clear that the dispersion parameter \( \phi \) scales the variance and does not affect the mean. Also, note that the log-likelihood of (2.6) is

\[
\mathcal{L}(y; \theta, \phi) = \phi^{-1} \theta y_i - \phi^{-1} \log C(\theta) + \log m(y; \phi)
\]

which means that the log-likelihood with respect to \( \theta \) satisfies

\[
\mathcal{L}(y; \theta) \propto \theta y - \log C(\theta)
\]

which is free of \( \phi \). This property is handy since it means that one can estimate \( \theta \) without knowledge of \( \phi \), and then estimate \( \phi \) post-hoc. This is specifically used when fitting Generalized Linear Models (see Section 3.5.1) for data that follows some exponential dispersion family.

### 2.2 Convex optimization

Many statistical problems are concerned with finding parameters that minimize a function, i.e., solving

\[
\theta^* = \arg \min_{\theta \in \Theta} \ell(\theta),
\]

where \( \Theta \subseteq \mathbb{R}^d \) is the parameter space, and \( \ell : \Theta \rightarrow \mathbb{R} \) is a function. The function \( \ell \) is often referred to as a loss function, signifying that the loss will be higher for parameter values that are further away from the optimal parameter value \( \theta^* \). An example of this problem is the maximum likelihood estimation (MLE) problem (see Section 2.1, or specifically (2.5)), where \( \ell \) is chosen as proportional to the negative log-likelihood function, i.e. \( \ell(\theta; y) \propto -\log \mathcal{L}(y; \theta) \).

The following presentation is based on Boyd and Vandenberghe (2004). If the parameter space \( \Theta \) is a convex set and the function \( \ell \) is convex in \( \theta \), this is
referred to as a convex optimization problem. A convex set is any set $\Theta$ for which all line segments between any two points in $\Theta$ are contained in $\Theta$, i.e. it holds that
\[
\forall \theta_1, \theta_2 \in \Theta, \forall \lambda \in [0, 1] : \lambda \theta_1 + (1 - \lambda) \theta_2 \in \Theta.
\]
A convex function is a function $\ell : \Theta \to \mathbb{R}$ for which the line segment between any two points in $\Theta$ lies above the graph of $\ell$, i.e. it holds that
\[
\forall \theta_1, \theta_2 \in \Theta, \forall \lambda \in [0, 1] : \ell(\lambda \theta_1 + (1 - \lambda) \theta_2) \leq \lambda \ell(\theta_1) + (1 - \lambda) \ell(\theta_2).
\]
For twice differentiable functions with convex domains, convexity is equivalent to the Hessian matrix being positive semi-definite for all $\theta \in \Theta$, i.e.
\[
\nabla^2_\theta \ell(\theta) \succeq 0 \ \forall \theta \in \Theta:
\]
where $\nabla^2_\theta \ell(\theta)$ is the Hessian matrix of $\ell$ evaluated at $\theta$, i.e.
\[
(\nabla^2_\theta \ell(\theta))_{ij} = \frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j}
\]
and $\succeq$ denotes positive semi-definiteness, i.e.
\[
A \succeq 0 \iff v^T Av \geq 0 \ \forall v \in \mathbb{R}^d.
\]
Minimizing $\ell(\theta)$ with respect to $\theta$ is equivalent to finding the root of the gradient of $\ell$ with respect to $\theta$, i.e. solving
\[
\nabla_\theta \ell(\theta) = 0
\]
where $\mathbf{0}$ is a zero vector. For some functions, this can be done analytically, providing a closed-form solution for the optimal parameter estimate. For others, this is instead done using numerical methods. A standard numerical method for solving convex optimization problems is the gradient descent algorithm (Boyd and Vandenberghe 2004, Chapter 9). Gradient descent is initiated by choosing an initial value $\hat{\theta}^{(0)} \in \Theta$ and then iteratively updating the parameter vector $\theta$ by taking steps in the direction of the negative gradient of $\ell$. If $\Theta = \mathbb{R}^d$, this can be done by making updates
\[
\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} - \epsilon_k \nabla_\theta \ell(\hat{\theta}^{(k)}), \tag{2.8}
\]
where $\epsilon_k > 0$ is a step size parameter. Figure 2.2 shows a visualization of the gradient descent algorithm. Gradient descent guarantees that $\ell(\hat{\theta}^{(k+1)}) \leq \ell(\hat{\theta}^{(k)})$ with the inequality being strict unless $\hat{\theta}^{(k)}$ is the solution to (2.7). The gradient descent steps shown in (2.8) are made until some stopping criterion is met. An example of a stopping criterion is to stop when the loss reduction of the step is under some threshold $\xi$, i.e. to stop when
\[
\ell(\hat{\theta}^{(k-1)}) - \ell(\hat{\theta}^{(k)}) < \xi \tag{2.9}
\]
is fulfilled.
Figure 2.2: Visualization of the gradient descent algorithm for parameter $\theta \in \mathbb{R}$ and function $\ell(\theta) = \theta^2$. Note that the step sizes $\epsilon_k$ are here chosen to be arbitrary values in order to improve the visualization. The algorithm converges in three steps to $\hat{\theta}^{(3)} = 0$. 
Chapter 3

Machine learning regression

3.1 The regression problem

Machine learning regression models can be described as a family of model classes that use more-or-less complex algorithms to learn some underlying structure from data. Regression will here be defined as estimating some unknown function $h : \mathcal{X} \rightarrow \Theta$ using a set of observations $(x_i, y_i)_{i=1}^n$ where $x_i \in \mathcal{X}$ are feature vectors of explanatory variables and $y_i \in \mathcal{Y}$ are the corresponding responses.

Remark 3.1. Note that the bold-face $y_i$ here refers to one observation, contrary to in Section 2.1 where $y$ refers to a vector of observations. The reason for this is that $y_i$ here refers to a random variable that can take on vector values or categorical values. A sample of observations is here instead denoted as $(y_i)_{i=1}^n$.

The responses $y_i$ are assumed to be generated from probability distribution $F$ with parameter $\theta = h(x_i)$, i.e.

$$Y | x \sim F(\theta), \quad \theta = h(x)$$ (3.1)

where $Y$ is a random variable and $\theta \in \Theta$ is a parameter whose value depends on $x$ through the unknown function $h$. If the distribution $F$ is known, the problem of estimating $h$ can be formulated as a maximum likelihood estimation problem. Friedman (2001) describes the goal of regression as finding a function

$$\hat{u} = \arg\min_{u \in \mathcal{M}} E_x [E_y [\ell(u(X); Y) | X]].$$ (3.2)

where the inner expectation is taken over the distribution of $Y$ given $X$ and the outer is taken over the distribution of $X$. $\mathcal{M}$ is here a model class, i.e. a set of functions $u : \mathcal{X} \rightarrow \Theta$ that share a common structure. See Section 3.2 for
further definitions and Section 3.5 for examples of model classes used in this thesis. The loss function $\ell : \Theta \times \mathcal{Y} \rightarrow \mathbb{R}$ fulfills

$$
\ell(\theta; y) \propto -\mathcal{L}(y; \theta),
$$

(3.3)

where $\mathcal{L}$ is the log-likelihood of $F$. The proportionality in (3.3) refers to proportionality with respect to $\theta$.

Note that (3.2) can not be solved directly, but can be approximated by minimizing the empirical loss, i.e. by solving

$$
\hat{u} = \text{argmin}_{u \in \mathcal{M}} \sum_{i=1}^{n} \ell(u(x_i); y_i).
$$

(3.4)

Note that if $F$ is a member of the exponential family, the negative log-likelihood is a convex function of $\theta$, meaning that the loss function in (3.3) is convex. This does not necessarily make (3.4) a convex optimization problem since the loss function is not always convex with respect to the function $u$.

If the distribution belongs to the exponential dispersion family and one assumes that the dispersion parameter $\phi$ is constant (i.e. does not depend on $x$), one can use a loss function that satisfies (3.3) for non-dispersion parameters $\theta$, since the log-likelihood with respect to $\theta$ can be optimized regardless of the value of $\phi$ (see Section 2.1.1).

The most common setting is that $\theta = \mathbb{E}[Y|x]$, i.e. that one tries to estimate a function $h$ that determines the expected value given a feature vector $x$. Also, if one wants to avoid making distributional assumptions about $F$, it is common to use a loss function that instead measures the difference between the observations and their expected value given the function estimate $\hat{u}$, i.e.

$$
\ell(\theta; y) = ||\theta - y||^2
$$

(3.5)

where $|| \cdot ||$ is a norm. The by far most common choice is the Euclidean norm, i.e., $|| \cdot ||_2$ (see e.g. Friedman (2001), Hastie et al. (2009), Breiman et al. (1984)). For $d = 1$, this is referred to as least squares regression. The setting where $\theta$ is a distributional parameter determining the entire distribution $F$ is sometimes referred to as probabilistic regression (Gneiting and Raftery 2007, Duan et al. 2020).

**Remark 3.2.** Gneiting and Raftery (2007) discusses which choice of loss functions can be considered proper scoring rules, i.e. loss functions that are minimized by the true distribution of $Y$ given $x$. Gneiting and Raftery (2007) show that the loss function in (3.5) is a proper scoring rule.

Finding an approximate solution to (3.4) is generally possible using some training algorithm (see Section 3.3). However, confirming that this solution is a good approximation of the solution to (3.2) is generally harder. For a discussion on training algorithms and the specific model classes’ ability to approximate the solution to (3.2), see Section 3.3.
Remark 3.3. In many cases, the distribution of $Y$ also depends on some weights $w \in \mathbb{R}$. Then (3.1) can be rewritten as $Y|x,w \sim F(\theta,w)$, and the loss function should satisfy $\ell(\theta; y, w) \propto -L(y; \theta, w)$. The weights will not affect the theory presented in this section and will be omitted in the following to simplify notation. However, this is a standard setting in, e.g., non-life insurance applications. For example, the duration of a policy often enters the distribution of the number of claims as a weight. See Chapter 4 for more on non-life insurance applications, and note that the loss functions defined there follow the same structure as those in this section but with a known weight factor included per observation.

Remark 3.4. Other loss functions than least squares and log-likelihood can be used in regression – many of them in order to impose some regularization on the model (see Section 3.3). This thesis focuses mainly on log-likelihood and least squares, and the reader is referred to, e.g., Hastie et al. (2009) or Gneiting and Raftery (2007) for more thorough discussions of other loss functions. Regularization is discussed in Section 3.3, and this thesis mainly uses hyperparameter tuning in the form of early stopping to impose regularization.

3.2 Model classes

A model class will, in this thesis, be defined as a set of functions whose output is determined by input $x$ and model parameters $\alpha \in A$. A class $M$ refers to the set of all functions $u_\alpha$ with a common functional form given $\alpha$, i.e.

$$ M = \{ u_\alpha : X \to \Theta | \alpha \in A \} $$

where $\alpha \in A$ is the model parameter space and where the functional form of $u_\alpha$ is fully determined by $\alpha$. Note that the feature space $X$, the output space $\Theta$, and the model parameter space $A$ are all part of the model class definition and will vary between different model classes. A specific model instance $u_\alpha$ is a function that maps from $X$ to $\Theta$ and is fully determined by the model parameters $\alpha$.

A trivial example of a model class would be the class of null or intercept-only models,

$$ M_0 = \{ u_\theta : X \to \Theta | \theta \in \Theta \} $$

where $u_\theta(x) = \theta$ for all $x \in X$, whereas a simple linear model class can be written as

$$ M_1 = \{ u_\beta : \mathbb{R} \to \mathbb{R} | \beta = (\beta_0, \beta_1) \in \mathbb{R}^2 \} $$

where

$$ u_\beta(x) = \beta_0 + \beta_1 x. \quad (3.6) $$

A more flexible linear model is the Generalized Linear Model introduced in Section 3.5.1.
Given a model class $\mathcal{M}$, solving a regression problem, i.e. solving (3.4) for a given loss function $\ell$, is equivalent to finding a model parameter $\hat{\alpha}$ such that
\[
\hat{\alpha} = \arg\min_{\alpha \in A} \sum_{i=1}^{n} \ell(u_\alpha(x_i); y_i).
\] (3.7)
The $u$ that solves (3.4) is then $u_\alpha$. Note that the solution to (3.7) is not necessarily the best possible estimate of $h$, but rather the best estimate within the model class $\mathcal{M}$ and given the loss function $\ell$. The following section will discuss iterative algorithms for solving (3.7).

**Remark 3.5.** In this thesis, the word parameter will be used to refer to three different concepts:

1. The parameters, or parameter functions, $\psi$ or $\theta$ related to distribution $F$ in (3.1).
2. The model parameters $\alpha$ defining the output function of a model instance $u_\alpha$
3. The hyperparameters used when training a model. These are defined in Section 3.3 and influence the model parameter space $A$ used in (3.7), by reducing the search space to some subset $A' \subseteq A$.

Hyperparameters will always be referred to using the word hyperparameter. The word parameter will be used to refer to either model parameters or parameters related to the distribution $F$ – the context should make it clear which one is meant.

There are many models which map to $\mathbb{R}^d$, which might not be identical to the parameter space of interest, $\Theta$. However, one can use a link function $h : \Theta \rightarrow \mathbb{R}^d$ in order to map the output of the model to $\Theta$. This link function can be viewed as a part of the model class, e.g., by using a log-link and replacing the functional form of the linear model in (3.6) with
\[
u_\beta(x) = \exp\{\beta_0 + \beta_1 x\}
\]
to change the mapping from $\mathbb{R}$ to $\mathbb{R}^+$. Another option is to include the link function in the loss function $\ell$, e.g. by using a $\ell$ such that
\[
\ell(u(x); y) \propto -\log \mathcal{L}(y; h^{-1}(u(x)))).
\] (3.8)
Note that convexity of $-\log \mathcal{L}(y; \theta)$ does not necessarily imply convexity of $\ell$, meaning one has to be careful when choosing link functions if the training algorithm relies on the convexity of the loss function. The model class examples in Section 3.5 are all defined such that the output of the model is in $\mathbb{R}^d$. In order to extend them to map to $\Theta$, one can use a link function $h$ and define the loss using (3.8).
3.3 Training and overfitting

A training algorithm will in this thesis refer to an algorithm that produces a model parameter estimate meant to approximate the solution of (3.7). In this thesis, the emphasis will lie on iterative training algorithms. This refers to algorithms that, given a training data set \((x_i, y_i)_{i=1}^n\), produce a sequence of model parameters \((\hat{\alpha}^{(k)})_{k=0}\) such that

\[
\sum_{i=1}^n \ell(\hat{u}_{\hat{\alpha}^{(k)}}(x_i); y_i) \leq \sum_{i=1}^n \ell(\hat{u}_{\hat{\alpha}^{(k-1)}}(x_i); y_i)
\]

for all training iterations \(k = 1, \ldots, K\). The idea of the iterative training algorithm is to take a step, adjusting the current model parameter estimate \(\hat{\alpha}^{(k)}\) to a new estimate \(\hat{\alpha}^{(k+1)}\), such that the empirical loss on the training data is reduced. The notation of a sequence of function estimates is especially common when describing boosting algorithms, see e.g. Friedman (2001) or Hastie et al. (2009, Chapter 10).

A training algorithm is repeated until it reaches a pre-defined stopping criterion. One such criterion is a threshold on the change in the loss function, which means running the algorithm until reaching a \(k\) such that

\[
\sum_{i=1}^n \ell(\hat{u}_{\hat{\alpha}^{(k-1)}}(x_i); y_i) - \sum_{i=1}^n \ell(\hat{u}_{\hat{\alpha}^{(k)}}(x_i); y_i) < \xi
\]

for some small \(\xi > 0\). Note that this is the regression equivalent of the stopping criterion in (2.9). The percentage loss change can also be used as a stopping criterion, and in cases where there is a well-defined measure of distance between model parameters \(\hat{\alpha}^{(k)}\) and \(\hat{\alpha}^{(k-1)}\), one can also use a stopping criterion based on this distance being sufficiently small (this is applicable for e.g. generalized linear models, see Section 3.5.1).

Many machine learning model classes have a very large number of model parameters, which in parameter regression can lead to overfitting on the training data. Overfitting refers to when an overparameterized model class is able to fit the training data very well but has assumed a functional structure that does not translate well to out-of-sample data. Some model classes can, given sufficient amount of training iterations, reach a parameter setting where the model perfectly fits the training data, i.e.

\[
\lim_{k \to \infty} u_{\hat{\alpha}^{(k)}}(x_i) \in \left\{ \theta^*_i : \theta^*_i = \arg\min_{\theta \in \mathbb{R}^d} \ell(\theta; y_i) \right\}
\]

for all \(i = 1, \ldots, n\). Models that have reached this step will henceforth be referred to as saturated models. Note that if \(\theta\) represents the expected value of a one-dimensional \(Y\) given \(X\), then (3.10) is equal to

\[
\lim_{k \to \infty} u_{\hat{\alpha}^{(k)}}(x_i) = y_i
\]
for all $i = 1, \ldots , n$, meaning the model is able to perfectly predict the training data. The fact that the training algorithm converges to the saturated model after sufficiently many training steps is generally desirable since such a model class likely can fit any functional form arbitrarily well and is, therefore, very flexible. However, a model that fulfills (3.10) is generally not suitable for out-of-sample prediction, since the overparameterization can lead to the model being very adapted to noise in the training data. Such a model is therefore unlikely to be a good approximation of the solution of (3.2).

To mitigate this, different methods of regularization can be used. This thesis will focus on early stopping, which means that one stops the training algorithm before reaching (3.10). In early stopping, a subset of the training data is set aside as a validation set. Let $(x_i, y_i)_{i=1}^m$ be the validation set, and run the training algorithm on the remaining training data set $(x_i, y_i)_{i=m+1}^n$, creating a sequence of model parameters $(\hat{\alpha}^{(k)})_k$, until satisfying the stopping criterion

$$\sum_{i=1}^m \ell(u_{\hat{\alpha}^{(k)}}(x_i); y_i) > \sum_{i=1}^m \ell(u_{\hat{\alpha}^{(k-1)}}(x_i); y_i).$$  (3.11)

The fact that (3.11) is satisfied indicates that the model has found the general structure of the function and is thereafter modeling noise in the training data – which should lead to the out-of-sample loss increasing rather than decreasing with further training iterations. Note that the validation loss

$$\sum_{i=1}^m \ell(u_{\hat{\alpha}}(x_i); y_i)$$

in (3.11) can be seen as an estimate of the out-of-sample loss

$$\mathbb{E}_X [\mathbb{E}_Y [\ell(u_{\hat{\alpha}}(X); Y) | X]].$$

seen in (3.2). The point of early stopping is to try to find the model parameter which minimizes the out-of-sample loss, for which the validation loss is used as a proxy. The algorithm assumes that this model parameter lies somewhere in the sequence of model parameters produced by the training algorithm. The $k$ that fulfills (3.11) is then chosen as the number of training iterations $\kappa$ and the model is re-trained on the full training data $(x_i, y_i)_{i=1}^n$. Often, one uses a stopping criterion based on a number of boosting steps, i.e., finding a $k$ such that (3.11) is fulfilled for the last $m$ steps. Figure 3.1 shows the loss on a training and validation set in such a setting, illustrating the point of early stopping.

**Remark 3.6.** The early stopping scheme described above can be extended to cross-validation by splitting the training data into $K$ subsets and using $K - 1$ of these subsets as a training set and the remaining subset as a validation set. This is used in, e.g., Papers I and II.

Many model classes have a set of hyperparameters that are used in order to regularize the training procedure and avoid overfitting. Often, $\kappa$ is considered
a hyperparameter (e.g. in regression trees, $\kappa$ represents the depth of the tree, see Section 3.5.2). Setting hyperparameters to specific values is equivalent to limiting the model parameter space to some subset $A' \subseteq A$. For more on what hyperparameters are relevant for different classes, see the class descriptions in Section 3.5.

### 3.4 Model interpretation

One of the most common criticisms of machine learning models is that they are essentially black-box models where one can not easily follow the path from input $x$ to output $u_{\alpha}(x)$. There are, however, tools that can be applied to interpret the results even for more complex machine learning models. The tools needed to interpret the different model classes considered in this thesis will be presented in their respective sections. In this introductory section, model interpretation will be split into two different categories - global and local interpretation, which is a distinction also made in e.g. Molnar 2020, Chapter 3.

Global interpretation focuses on trying to answer the question “How does the input $x$ influence the output?” in general terms. Answering this question is similar to the concept of statistical inference, meaning one is looking for an explanation of how the distribution of the observations changes with the features. Global interpretation is essential to understand the data structure in broader terms.

Local interpretation is trying to answer the question “Why did feature vector
Some global interpretation methods are model class agnostic, e.g. partial dependence plots (Hastie et al. 2009, Chapter 10.13) used for global interpretation. A local interpretation method with this property is the SHapley Additive exPlanations (SHAP), as presented in Lundberg and Lee (2017). Since none of these methods are used in the papers of this thesis, they will not be discussed further here, but the interested reader is referred to Molnar (2020, Chapter 6).

### 3.5 Model class examples

The following sections describe a few model classes $\mathcal{M}$ that are frequently used in this thesis. Further, potential hyperparameters, training algorithms, overfitting issues, and how to interpret the models and their predictions are discussed.

#### 3.5.1 Generalized linear models

Before defining the model class for Generalized Linear Models (GLMs), note that this section presents all models as if they map to the real line $\mathbb{R}$. Thus, for consistency, the GLMs are here presented as linear models, and the link function is not explicitly mentioned in the definition of the model class. Instead, it is assumed that the link function is incorporated in the definition of the loss function, and that one uses a link function when predicting using the model.

The model class for Generalized Linear Models (GLMs) can be defined as

$$\mathcal{M}_{\text{GLM}} = \left\{ u_\beta : \mathbb{R}^p \to \mathbb{R} \mid \beta \in \mathbb{R}^{p+1} \right\}$$

using the linear function

$$u_\beta(x) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$

(3.12)

where $x_j$ refers to the $j$:th feature in vector $x$.

**Remark 3.7.** Note, again, that the GLM defined by (3.12) is a linear mapping to the real line. GLMs are generally defined such that the linear predictor is mapped to the outcome space using a specific link function. The distribution is assumed to be an exponential dispersion family. This means that one can fit the model using maximum likelihood theory without assuming any value of the dispersion parameter (see Section 2.1.1). Since the presentation in this
section assumes that the link function is incorporated in the definition of the loss function, the link function is not explicitly mentioned in the definition of the model class. However, the two representations are equivalent. For a more standard and thorough presentation of GLMs, see e.g. McCullagh and Nelder (1989) or Sundberg (2019, Chapter 9).

To use a GLM for regression, one needs to solve (3.4), which is equivalent to solving

$$
\hat{\beta} = \arg\min_{\beta \in \mathbb{R}^{p+1}} \sum_{i=1}^{n} \ell \left( \beta_0 + \sum_{j=1}^{p} \beta_j x_j; y_i \right).
$$

(3.13)

If $\ell(\theta; y)$ is convex in $\theta$, it will also be convex in $\beta$, meaning that solving (3.13) is a convex optimization problem that can be solved using e.g. gradient descent as described in Section 2.2. Applying (2.8) to the loss function in (3.13) yields a sequence of parameters that comply with the definition of a training algorithm in Section 3.3. This sequence is guaranteed to converge to a global minimum of the loss function on the training data set if the loss function is convex.

If the training data size $n$ is larger than the parameter vector size $p + 1$, GLMs are not guaranteed to reach a saturated model, meaning that (3.10) will not be satisfied. However, if $p$ is large in relation to $n$, it might be useful to regularize the model using, e.g., Lasso or Ridge regression. Lasso or Ridge regression introduces hyperparameters that alter the loss function used in training and penalizes large values of the coefficients of $\beta$ (see Hastie et al. (2009, Chapter 3.4), where these methods are referred to as shrinkage methods).

One of the main advantages of using GLMs, especially in the insurance applications discussed in Chapter 4, is that the models and their output are relatively easy to interpret. A parameter $\beta_j$ can always be described as the output change given a one-unit increase in feature $x_j$. Note that this will have different meanings if one uses a link function. When the link function is the natural logarithm, which is common when working with e.g. Poisson distributions, the estimate of parameter $\theta = \exp\{u_\beta(x)\}$ changes by a factor $\exp(\beta_j)$ for a one-unit increase in feature $x_j$. Note that these interpretations apply both locally and globally.

GLMs are used in the framework of the Collective Reserving Model with Openness presented in Paper II. The model class is also used as a benchmark in papers I and III, and the varying coefficient model presented in Paper V is a combination of a GLM and a Gradient Boosting Machine (see Section 3.5.3 for an introduction to Gradient Boosting Machines, as defined by Friedman (2001)).

### 3.5.2 Classification and regression trees

The Classification And Regression Tree (CART) was introduced in Breiman et al. (1984). The CART is based on the idea that observations that are similar in $X$ should be grouped together into subsets. The class of CARTs for regression
can be described as
\[ \mathcal{M}_{\text{CART}} = \{ u_\nu : \mathcal{X} \to \mathbb{R}^d | \nu \in \mathcal{N} \} \]
where \( \mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_p \) is the feature space for a \( p \)-dimensional feature vector. An individual feature space \( \mathcal{X}_j \) can be either ordinal, meaning that \( \mathcal{X}_j \subseteq \mathbb{R} \), or categorical, i.e. \( \mathcal{X}_j \) is a finite set of non-ordered values. The output function is given by
\[ u_\nu(x) = \sum_{l=1}^{m} \mathbf{1}_{\{x \in A_l\}} \delta_l \]
where \( \mathbf{1}_{\{\cdot\}} \) is the indicator function, \( A = \{ A_l \}_{l=1}^{m} \) is a disjoint partition of \( \mathcal{X} \) into \( m = |A| \) subsets, and \( \delta_l \in \mathbb{R}^d \) is the output parameter for subset \( A_l \). The parameters \( \nu = \{ A, \delta \} \), where \( \delta = \{ \delta_l \}_{l=1}^{m} \), fully define the tree structure, and the parameter space \( \mathcal{N} \) is the set of all possible tree structures. The individual subsets \( A_l \) are called regions. A CART is a binary tree, which means that the regions are defined as the intersections of a sequence of binary splits of the feature space \( \mathcal{X} \).

**Remark 3.8.** In most presentations on regression trees, and specifically in Breiman et al. (1984), the observation space \( \mathcal{Y} \) is assumed to be \( \mathbb{R} \), and the loss function is assumed to be the sum of squared errors. However, the CART model class is not limited to this loss function or observation space. The rest of this section will present the CART model class in a more general setting but will assume that the loss function is convex.

One can note that, given that \( \ell \) is convex, the optimal parameter \( \hat{\delta}_l \) for a given region \( A_l \) can be found by solving
\[ \hat{\delta}(A_l) = \arg\min_{\delta \in \mathbb{R}^d} \sum_{i : x_i \in A_l} \ell(\delta; y_i) \quad (3.14) \]
where it is assumed that the sum only covers observations in the training data set, meaning that fitting a tree is a matter of finding a tree structure
\[ \hat{A} = \arg\min_{A \in \mathcal{A}} \sum_{l=1}^{m} \sum_{i : x_i \in A_l} \ell(\hat{\delta}(A_l); y_i) \quad (3.15) \]
where \( \mathcal{A} \) is the set of all possible partitions of \( \mathcal{X} \) and \( \hat{\delta}_l \) is defined as in (3.14). The parameters \( \hat{\delta} \) will then be direct consequences of the tree partition.

**Remark 3.9.** In Breiman et al. (1984) and several other presentations on CARTs, the loss is not defined on a per-observation basis but rather on a per-region basis. Such loss functions are often referred to as cost functions or impurity functions. For many losses, there are equivalent losses for both the per-observation and per-region formulation. See Remark 3.12 for an example of this.
Solving (3.15) for a CART is not a convex optimization problem, and thus, one has to test all possible parameter settings $A \in \mathcal{A}$. Testing the entire search space of partitions $\mathcal{A}$ is generally not feasible. Instead, CARTs, as presented in Breiman et al. (1984), are fitted using a greedy approach, with certain constraints on the binary splits defining the partition $A$.

The training algorithm is initialized with $\hat{\nu}^{(0)} = \{\hat{A}^{(0)}, \hat{\delta}^{(0)}\}$, where $\hat{A}^{(0)} = \{X\}$ and $\hat{\delta}^{(0)} = \hat{\delta}(X)$ is the optimal parameter for the full training data set. Then, the algorithm iterates over the regions in the current partition $\hat{A}^{(k)}$, and for each region, finds an optimal binary split $\hat{B}^{(k)}_l$ of the region by solving

$$\hat{B}^{(k)}_l = \text{argmin}_{\hat{B} \in \mathcal{B}} \left\{ \sum_{i : x_i \in \hat{A}^{(k)}_l} \ell(\hat{\delta}(\hat{B}); y_i) + \sum_{i : x_i \in \hat{B}} \ell(\delta(\hat{B}); y_i) \right\}$$

(3.16)

where $\hat{\delta}(B)$ is short-hand notation for $\hat{\delta}(\hat{A}^{(k)}_l \cap B)$ using (3.14), $\hat{B}$ is the complement of $B$ with respect to $\hat{A}^{(k)}_l$, and $\mathcal{B}$ is the set of all axis-aligned subsets of $X$. An axis-aligned subset is a subset that only splits the feature space along one dimension, which means that it is on the form

$$B = X_1 \times \cdots \times X_{j-1} \times B_j \times X_{j+1} \times \cdots \times X_p$$

(3.17)

for feature $j$, where $B_j \subseteq X_j$ is a subset of the $j$:th feature space. This essentially means that a single split can not split $X$ using more than one feature.

After solving (3.16), the partition is updated to the new partition $\hat{A}^{(k+1)}$ using the splits $\hat{B}^{(k)}_l$ as

$$\hat{A}^{(k+1)}_{2l} = \hat{A}^{(k)}_l \cap \hat{B}^{(k)}_l$$

(3.18)

$$\hat{A}^{(k+1)}_{2l+1} = \hat{A}^{(k)}_l \cap \bar{\hat{B}}^{(k)}_l$$

(3.19)

for $l = 1, \ldots, 2^k$, whereafter the optimal output parameters $\hat{\delta}_{2l}$ and $\hat{\delta}_{2l+1}$ for the new regions are found using (3.14). This way, an updated parameter $\hat{\nu}^{(k+1)}$ is found, whereafter the algorithm (3.16)–(3.19) is repeated for the new partition $\hat{A}^{(k+1)}$. The process is repeated until $k = d_{\text{max}}$, meaning that the tree has reached its maximum depth. The value $d_{\text{max}}$ is a hyperparameter of the model. The training algorithm is outlined in Algorithm 1.
Algorithm 1: CART training algorithm

Let

- \((y_i, x_i)_{i=1}^n\) be the training data, \(y_i \in Y, x_i \in X\)
- \(\ell : \mathbb{R}^d \times Y \rightarrow \mathbb{R}\) be the loss function
- \(B\) be the set of all axis-aligned subsets of \(X\)
- \(d_{\text{max}}\) be the maximum tree depth

Initialize \(\nu^{(0)} = (\hat{A}^{(0)}, \hat{\delta}^{(0)})\) where

\[
\hat{A}^{(0)} = \{X\}, \quad \hat{\delta}^{(0)} = \arg\min_{\delta \in \mathbb{R}^d} \sum_{i=1}^n \ell(\delta; y_i)
\]

For \(k = 0, \ldots, d_{\text{max}} - 1:\)

For \(\hat{A}_l \in \hat{A}^{(k)}:\)

1. Split

\[
\hat{B}_l^{(k)} = \arg\min_{B \in B} \sum_{i : x_i \in \hat{A}_l^{(k)}} \left\{ \sum_{i : x_i \in B} \ell(\delta(B); y_i) + \sum_{i : x_i \in \bar{B}} \ell(\delta(\bar{B}); y_i) \right\}
\]

where

\[
\hat{\delta}(B) = \arg\min_{\delta \in \mathbb{R}^d} \sum_{i : x_i \in \hat{A}_l^{(k)} \cap B} \ell(\delta; y_i)
\]

2. Grow

\[
\hat{A}_l^{(k+1)} = \hat{A}_l^{(k)} \cap \hat{B}_l^{(k)}
\]

\[
\hat{A}_{2l+1}^{(k+1)} = \hat{A}_l^{(k)} \cap \bar{\hat{B}}_l^{(k)}
\]

3. Assign

\[
\hat{\delta}_{2l}^{(k+1)} = \arg\min_{\delta \in \mathbb{R}^d} \sum_{i : x_i \in \hat{A}_{2l+1}^{(k+1)}} \ell(\delta; y_i)
\]

\[
\hat{\delta}_{2l+1}^{(k+1)} = \arg\min_{\delta \in \mathbb{R}^d} \sum_{i : x_i \in \hat{A}_{2l+1}^{(k+1)}} \ell(\delta; y_i)
\]

Return \(\nu^{(k+1)} = (\hat{A}^{(k+1)}, \hat{\delta}^{(k+1)})\)

See Figure 3.2 for an example of the output in different stages of this training algorithm. A visualization of a CART is given in Figure 3.3.
Figure 3.2: Estimates of the function $y = \sin x$ from regression trees at different stages of training (i.e. trees with different depths), using a squared error loss function. As can be seen, the estimation error is reduced for deeper trees. Note that the signal here contains no noise and the example means only to illustrate the increasing complexity when training trees further.

The regions of the last partition are referred to as **terminal nodes** of the tree, while the preceding regions are just called **nodes**. Note that a node can be referred to using the index $k, l$, where $k$ is the depth of the node and $l$ is the horizontal position of the node in the tree.

**Remark 3.10.** Note that the solution to (3.16) can be $\hat{B}_1^{(k)} = \emptyset$. Such a split choice essentially turns the node into a terminal node before reaching depth $d_{\text{max}}$ since no further loss reduction is possible from splitting the data. This can generally only happen if all $\mathbf{x}_i$ in the node have the same value for all features $j = 1, \ldots, p$, meaning there is no way to produce further non-empty subsets, or if all observations in the node have the same response value $y_i$, meaning it is not possible to reduce the loss further.

Figure 3.3: A CART of depth 2 with 4 terminal nodes estimating $y \in \mathbb{R}$.

To solve (3.16) for axis-aligned splits, one has to loop through all features $j = 1, \ldots, p$, and for each feature finding a subset $B_j$ of $\mathcal{X}_j$ that minimizes the loss function in (3.16). when splitting using (3.17). The search space of subsets $B_j$ will depend on whether the feature is ordinal or categorical.
For ordinal features, i.e. $\mathcal{X}_j \subseteq \mathbb{R}$, the subsets $B_j$ are restricted to continuous intervals of the form

$$B_j = \{ x \in \mathcal{X}_j : x \leq t \}$$

for some $t \in \mathcal{X}_j$. When training, the search space is reduced by only threshold values $t$ that correspond to a unique split of the training data set. Usually, the search space for $t$ is set to the midpoints between the values of the feature in the training data set, i.e.

$$t \in \left\{ \frac{x_{(i)} + x_{(i+1)}}{2} \right\}_{i=1}^{n_l}$$

where $x_{(i)}$ is the $i$:th smallest value of the feature in the training data set and $n_l$ is the number of data points from the training data that fulfill $x_i \in \hat{A}_l^{(k)}$. This restriction means that the search space for $t$ is (at most) of size $n_l - 1$. For large datasets, this can still be a very large number. To reduce the search space, one can instead consider a smaller set of threshold values by binning the data into a set number of $M$ ordered bins and then only considering splits between bins, i.e.,

$$t \in \left\{ \frac{x_{\lfloor \frac{i}{M} \rfloor} + x_{\lfloor \frac{i}{M} \rfloor + 1}}{2} \right\}_{i=1}^{M-1}$$

where $\lfloor \cdot \rfloor$ is the floor operator. The binning method reduces the search space to $M - 1$ elements. In Ke et al. (2017), this method is used in a boosting algorithm called LightGBM (see Section 3.5.3 for more on boosting). Binning the data before defining the split candidates is sometimes referred to as histogram-based splitting (see, e.g., the sklearn package (Pedregosa et al. 2011)). Figure 3.4 illustrates the difference between the two methods.

![Figure 3.4: Splitting candidates for an ordinal feature consisting of 9 numbers. There are $9 - 1 = 8$ relevant splits, as seen in (a). Binning the numbers into 3 bins, there are $3 - 1 = 2$ relevant splits, as seen in (b).](image)

For categorical features, i.e. $\mathcal{X}_j = \{a_1, \ldots, a_K\}$, where $K$ is the number of categories, the subsets $B_j$ can be any set in the power set of $\mathcal{X}_j$. This means that there are $2^{K-1} - 1$ relevant subsets, as seen in Figure 3.5a. For high-cardinality categorical features, i.e., for high $K$, this can be a very large set.
If the response space is $\mathbb{R}$, one can calculate the mean response value for each category $a \in X_j$ according to

$$\bar{y}_a = \frac{\sum_{i: x_i \in \hat{A}_l^{(k)}} 1\{x_{ij} = a\} y_i}{\sum_{i: x_i \in \hat{A}_l^{(k)}} 1\{x_{ij} = a\}}$$

and then order the categories by their mean response value as $(a_{(1)}, \ldots, a_{(|X_j|)})$, whereafter one can consider only splits

$$B_j = \{x \in X_j : x \in \{a_{(1)}, \ldots, a_{(t)}\}\}$$

for some $t \in \{1, \ldots, K - 1\}$. Figure 3.5 illustrates the difference between the two methods. This search space is guaranteed to contain the optimal splitting

![Diagram](image-url)

(a) Non-ordered split candidates  
(b) Ordered split candidates

Figure 3.5: Splitting candidates for a categorical feature consisting of 4 colors. If the categories can not be ordered, there are $2^4 - 1 = 7$ relevant splits, as seen in (a). However, if one can order the categories, e.g., by the average response value for each category, there are only $4 - 1 = 3$ relevant splits, as seen in (b). Here, $d = 1$, meaning there is only one ordering of the categorical features.

A popular hyperparameter in CARTs, apart from the max depth, is the minimum samples per node $n_{\min}$. $n_{\min}$ is used to avoid overfitting by ensuring that the $\delta$-estimates are based on a sufficient number of observations. This hyperparameter only allows splits $B$ that fulfill

$$n_{\min} \leq \left| \{i : x_i \in \hat{A}_l^{(k)} \cap B\} \right| \leq \left| \{i : x_i \in \hat{A}_l^{(k)}\} \right| - n_{\min}$$

rule (Hastie et al. 2009, p. 310). For response spaces $\mathbb{R}^d$, $d > 1$, making such orderings for each output dimension can be used to reduce the search space, but the optimal solution is no longer guaranteed to be in the search space in that case (Hastie et al. 2009, p. 310).
Figure 3.6: Example of a depth 2 CART ($u_\nu$) and a linear model ($u_\beta$) fitted using a squared error loss to observations of the deterministic process $y = f(x) = x + 1.5 \cdot \sin x$. The left-hand plot shows that CART captures the function decently on the training data, with an average loss of 0.14 as compared to the linear model result of 1.13. However, the range of $x$ in the training set is not representative of the global range of $x$, and the right-hand plot shows that the CART model performs poorly on out-of-sample data with a loss of 13.44 while the linear model model performs better with a loss of 1.52. Note that the example is highly fictitious, with no noise in the observations, and specifically tailored to be problematic for CARTs.

as solutions to (3.16), where $|\{i : x_i \in B\}|$ means the number of observations in the training data where the feature vector is in subset $B$. For more hyper-parameters in CARTs, see e.g. Hastie et al. (2009, Chapter 9.2), Breiman et al. (1984) or the sklearn package (Pedregosa et al. 2011).

Remark 3.11. Note that a tree with $n_{\text{min}} = 1$ will, given a sufficiently large $d_{\text{max}}$, become a saturated model, i.e. it will satisfy (3.10) Since trees are generally used as building blocks in so-called ensemble methods, it is more common to set a larger $n_{\text{min}}$ and a small $d_{\text{max}}$. See Section 3.5.3 for a presentation of Gradient Boosting Machines – a popular ensemble method frequently used in this thesis.

A caveat when using CARTs and other tree-based models for out-of-sample prediction is that the models are very reliant on the assumption that the distribution of feature values in the training data is representative of the distribution of feature values in out-of-sample data. This issue has to do with the piecewise constant nature of the output of a CART model, which makes extrapolation beyond known feature values difficult. Figure 3.6 shows an example of the issue. While the example is fictitious and specifically tailored to be problematic for CARTs, the extrapolation issues should be taken into account when using tree-based models for prediction tasks.

Remark 3.12. To extend this tree representation to classification trees, consider a categorical response variable $y \in \{a_1, \ldots, a_d\}$. Most loss functions used
for classification use the impurity representation of the loss (see Remark 3.9). A common loss function for classification is the Gini index of a set \((y_i)_{i \in I}\), see Breiman et al. (1984, p. 38), defined as

\[
J(A_l) = \sum_{c=1}^{d} \hat{p}_{lc}(1 - \hat{p}_{lc})
\]

where \(J(A_l)\) is the Gini index of the set \((y_i)_{i : x_i \in A_l}\), and

\[
\hat{p}_{lc} = \frac{1}{|\{i : x_i \in A_l\}|} \sum_{i : x_i \in A_l} 1\{y_i = c\}
\]

Note that if one uses one-hot encoding for the response variable, i.e., the response is transformed to a vector that takes on the value 1 in dimension \(j\) if \(y_i = c_j\) and 0 otherwise, i.e.,

\[
y_i = \begin{pmatrix}
1\{y_i = c_1\} \\
\vdots \\
1\{y_i = c_d\}
\end{pmatrix}
\]

then the Gini index is equivalent to using a sum of squared error loss and the optimal parameter \(\delta\) is the empirical probability vector \(\hat{p}_l = (\hat{p}_{lc})_{c=1}^{d}\) which is equal to the empirical average of the one-hot-encoded observations.

As for interpretability, CARTs are reasonably graspable on a local level in the sense that one can follow the splitting rules from the root node to the terminal node to see which features affect the result and how. The structure also mimics the decision process of a human expert in, e.g., a medical setting (Hastie et al. 2009, p. 305). For global interpretability, CARTs are less beneficial. Large trees can become visually complex to interpret, and the general impact of a feature on the response is not easily grasped. The feature importance is a widely used measure to quantify the effect a feature has on the output in a CART. The importance of a feature \(x_j\) is defined as the total loss reduction stemming from splits in the tree that used the feature during training. Let

\[
L_{k,l} = \sum_{i : x_i \in A_i^{(k)}} \ell(\hat{\delta}_i^k; y_i)
\]

denote the total loss of observations in node \(A_i^{(k)}\), and let

\[
LR_{k,l} = L_{k,l} - L_{k+1,2l} - L_{k+1,2l+1}
\]

denote the loss reduction from splitting the node \(A_i^{(k)}\) into the nodes \(A_{2l}^{(k+1)}\) and \(A_{2l+1}^{(k+1)}\). Also, remember that for a node that uses feature \(j\) in its split, the split \(\hat{B}_i^{(k)}\) will satisfy \((\hat{B}_i^{(k)})_j \neq X_j\), i.e., the \(j\):th dimension of the subspace is
not the entire feature space, since it has been split using feature \( j \). The feature importance of feature \( j \) is defined as

\[
FI_j = \sum_{k=0}^{d_{\text{max}}-1} \sum_{l=1}^{2^k} \sum_{(k,l):(B_{k,l})_j \neq X_j} \text{LR}_{k,l}.
\]  

(3.20)

The feature importance score thus adds all loss reductions stemming from splits made on the respective feature. If a lot of loss reduction is caused by splits using a certain feature, the feature will be deemed more important. The feature importance defined above is a slightly modified and generalized version of the measure presented by Breiman et al. (1984, p. 147), to fit the regression representation of this text. The feature importance scores are generally normalized over all features \( j = 1, \ldots, p \) to sum to 1. While this can provide meaningful insights into which features matter and which do not in a regression problem, it does not say anything about how the features affect the response. The feature importance is used for the boosting machine presented in Paper III and as a feature selection method in Paper V.

CARTs are essential building blocks in several models, which can be called ensemble methods or, more specifically, tree-based models. One tree-based model family is Gradient Boosting Machines, discussed in Section 3.5.3. Paper IV introduces a way to handle missing data in CART models; see Section 5.4.

### 3.5.3 Gradient boosting machines

Friedman (2001) introduced Gradient Boosting Machines (GBMs). GBMs are ensemble methods, meaning they consist of several other machine learning or regression models. GBMs are the only ensemble method presented in this thesis – the reader is referred to e.g. Hastie et al. (2009, Chapter 16) for a more thorough introduction to ensemble methods in general. The model class of GBMs for regression can be defined as

\[
M_{\text{GBM}} = \{u_\gamma : \mathcal{X} \rightarrow \mathbb{R} | \gamma \in \Gamma \}
\]

where the functional form can be stated as

\[
u(x) = \theta^{(0)} + \sum_{k=1}^\kappa \epsilon_k v_{\nu(k)}(x).
\]  

(3.21)

Here, \( \theta^{(0)} \in \mathbb{R} \) is a constant initial estimate, \( \epsilon_k \leq 1 \) are shrinkage factors, \( \kappa \) is the number of boosting steps and \( v_{\nu(k)} \in M_{\text{CART}} \) are regression trees. The model parameter \( \gamma = \{\theta^{(0)}, \nu^{(1)}, \ldots, \nu^{(\kappa)}\} \) fully defines the gradient boosting machine. The GBM model is inspired by gradient descent (see Section 2.2) but instead of looking for a parameter \( \theta \in \mathbb{R}^d \), GBMs try to estimate some parameter function \( \bar{\theta} : \mathcal{X} \rightarrow \mathbb{R} \).

**Remark 3.13.** Note that in Friedman (2001), the so-called weak learners \( v_{\nu(k)} \) are allowed to be any function, not just regression trees. However, Friedman
(2001) focuses on regression trees and that is the standard in most implementations of GBMs (see e.g. Hastie et al. 2009, Chapter 10 or Pedregosa et al. 2011). In this thesis, only CARTs will be considered for \( v_{\nu(k)} \). For the use of other weak learners in boosting, see e.g. Bühlmann and Yu (2003).

The most commonly used hyperparameters for GBMs are the number of boosting steps \( \kappa \), a sequence of shrinkage factors \( (\epsilon_k)_{k=1}^{\infty} \), as well as CART-specific hyperparameters \( d_{\text{max}} \) and \( n_{\text{min}} \). Note that the shrinkage terms \( \epsilon_k \) and the number of boosting steps will interact with each other since smaller values of \( \epsilon_k \) will require more boosting steps to reach the same level of complexity. It is, therefore, not advisable to tune both hyperparameters simultaneously. Both Friedman (2001) and Hastie et al. (2009, Chapter 10) argue that choosing a constant but small \( \epsilon_k = \epsilon \) and then using early stopping to find a suitable value of \( \kappa \) is a good strategy.

It is apparent that the space of model parameters \( \Gamma \), even under hyperparameter constraints is too extensive for an exact solution. Instead, GBMs are trained using a greedy approach by boosting one step at a time. Note that after \( k \) boosting steps, the model output is

\[
 u_{\gamma(k)}(x) = \theta(0) + \sum_{l=1}^{k} \epsilon_l v_{\nu(l)}(x)
\]

The model is initiated using a minimum loss estimator

\[
 \hat{\theta}(0)(x_i) := \hat{\theta}(0) = \arg\min_{\theta \in \mathbb{R}} \sum_{i=1}^{n} \ell(\theta; y_i)
\]

which can be found using convex optimization techniques (see Section 2.2). The point-wise derivatives (or gradients) of the loss function with respect to the unknown parameter function \( \theta \) are calculated, and evaluated in the current parameter function estimate as

\[
 g_i^{(k)} = \left. \frac{\partial}{\partial \theta} \ell(\theta; y_i) \right|_{\theta = u_{\gamma(k-1)}(x_i)}
\]

for \( i = 1, \ldots, n \). These gradients are used as a measure of how steep the loss function is at each point of the training data, similar to how the gradient is used in gradient descent (seen in Section 2.2). Friedman (2001) argues that without knowledge of the function value at every point, one can instead use a weak learner to create a function that approximates the negative gradient given the training data. A regression tree partition is fitted to the negative gradients using a sum-of-squares loss function as

\[
 \hat{A}^{(k)} = \arg\min_{A \in A} \sum_{l=1}^{m} \sum_{i \in A_l} \left( g_i^{(k)} + \bar{g}_l^{(k)} \right)^2
\]
where $\bar{g}_l^{(k)}$ is the average gradient in partition $A_l$,

$$\bar{g}_l^{(k)} = \frac{\sum_{i=1}^{n} g_i^{(k)} 1_{\{x_i \in A_l\}}}{\sum_{i=1}^{n} 1_{\{x_i \in A_l\}}}.$$ 

Equation 3.24 can be solved using the techniques described in Section 3.5.2. After finding the structure of the tree, the $\delta$’s of the terminal nodes are optimized to minimize the loss function $\ell$ for every region $\hat{A}_l^{(k)}$ of the partition $\hat{A}^{(k)}$ as

$$\hat{\delta}_l^{(k)} = \arg\min_{\delta \in \mathbb{R}^d} \sum_{i : x_i \in \hat{A}_l^{(k)}} \ell(u_{\gamma^{(k-1)}}(x_i) + \delta ; y_i). \quad (3.25)$$

The parameter is then updated by adding $\nu^{(k)} = (\hat{A}^{(k)}, \hat{\delta}^{(k)})$, which means that the new parameter function estimate is

$$u_{\gamma^{(k)}}(x) = u_{\gamma^{(k-1)}}(x) + \epsilon_k \nu^{(k)}(x). \quad (3.26)$$

The calculations of (3.23)–(3.26) are then repeated $\kappa$ times for increasing $k$. This training algorithm can be seen as a parameter regression equivalent to the gradient descent algorithm described in Section 2.2. Algorithm 2 summarizes the complete training algorithm. Note that Algorithm 2 produces a sequence of model parameters $(\gamma^{(k)})_{k=1}^{\kappa}$, meaning that early stopping is possible. Early stopping is used as the main hyperparameter tuning method in all GBM examples in this thesis.

An illustration of the output of a GBM trained using different values of $\kappa$ can be seen in Figure 3.7.

Local interpretation of a GBM, i.e. explaining why feature vector $x$ leads to a certain output, is limited since it lacks both the multiplicative structure of a GLM and the clear input-to-output path of a single CART. A method to mitigate this issue is presented further in Paper V, where a boosted varying coefficient model is used to provide more meaningful local interpretations while maintaining the flexibility of the functional form of a GBM. For global interpretation, the feature importance scores used for CARTs can be computed for each individual tree in a GBM using (3.20). The feature importance score can be calculated for the entire GBM by summing and normalizing. Note that the feature importance still retains its original interpretation as the total loss reduction of the model when splitting on a certain feature.

GBMs are used in Papers I, III, and V. Paper I uses a GBM to solve reserving problems, which are presented in 4.1. Paper III introduces the Cyclic Gradient Boosting Machine (CGBM), which extends the model class to multivariate regression, i.e. allowing $u_{\gamma} : \mathcal{X} \rightarrow \mathbb{R}^d$ for $d > 1$. Paper V introduces a boosted varying coefficient model, which is a special case of a CGBM with a GLM structure as its core, meant to improve the local interpretability of the model.
Algorithm 2: GBM training algorithm

Let

- \((y_i, x_i)_{i=1}^n\), be the training data, \(y_i \in \mathcal{Y}, x_i \in \mathcal{X}\)
- \(\ell : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}\) be the loss function
- \(v_\nu : \mathcal{X} \to \mathbb{R}\) be a CART
- \(\kappa, (\epsilon_k)_{k=1}^\kappa\) be boosting hyperparameters
- \(u_\gamma(k)(x) = \theta(0) + \sum_{j=1}^k \epsilon_j v_\nu(j)(x)\) be the output of the model after \(k\) boosting steps

Initialize \(\hat{\gamma}^{(0)} = (\hat{\theta}^{(0)})\) where

\[
\hat{\theta}^{(0)} = \arg\min_{\theta \in \mathbb{R}} \sum_{i=1}^n \ell(\theta; y_i).
\]

For \(k = 1, \ldots, \kappa\):

1. Compute

\[
g^{(k)}_i = \frac{\partial}{\partial \theta} \ell(\theta; y_i) \bigg|_{\theta = u_\gamma(k-1)(x_i)}
\]

2. Approximate

Fit

\[
\hat{A}^{(k)} = \arg\min_{A \in \mathcal{A}} \sum_{l=1}^m \sum_{i : x_i \in A_l} \left( g^{(k)}_i + \bar{g}^{(k)}_l \right)^2
\]

where \(\bar{g}^{(k)}_l\) is the average value of \(g^{(k)}_i\) for \(i\) where \(x_i \in A_l\)

Adjust for \(l = 1, \ldots, m\)

\[
\hat{\delta}^{(k)}_l = \arg\min_{\delta \in \mathbb{R}} \sum_{i : x_i \in \hat{A}^{(k)}_l} \ell(\theta^{(k-1)}(x) + \delta; y_i)
\]

Return

\[
\hat{\gamma}^{(k)} = (\hat{\theta}^{(0)}, \hat{\nu}^{(1)}, \ldots, \hat{\nu}^{(k)})
\]

where \(\hat{\nu}^{(k)} = (\hat{\gamma}^{(k)}, \hat{\delta}^{(k)})\)

3.5.3.1 XGBoost

XGBoost (Chen and Guestrin 2016) is a popular version of a GBM that can be defined using

\[
\mathcal{M}_{\text{XGBoost}} = \{ u_\gamma : \mathbb{R}^p \to \mathbb{R} \mid \gamma \in \Gamma \} \]
Figure 3.7: Out-of-sample prediction of parameter function $\mu(x) = \sin(x)x^4$, $x \in \mathbb{R}$, for GBMs fitted using data $y_i \sim \mathcal{N}(\mu(x_i), 1)$ and $\ell$ proportional to the negative log-likelihood of $\mathcal{N}$. The GBMs have been fitted using different numbers of boosting steps $\kappa$. Note that the signal-to-noise ratio here is chosen as very high in order to illustrate the effect of choosing different values of $\kappa$. 
where the functional form can be stated as
\[ u_\gamma(x) = \theta(0) + \sum_{k=1}^{\kappa} \epsilon_k v_{\nu(k)}(x). \]

One can notice that the structure of the model is the same as for GBMs, and the parameter \( \gamma \) and its components are defined in the same way. The difference lies in the training algorithm, with the most significant difference being how the trees are trained. This section will not go into detail on other differences, and the reader is referred to Chen and Guestrin (2016) for a more thorough introduction.

The XGBoost algorithm is initialized identically to a GBM by (3.22). The consecutive updates, however, utilize a second-order Taylor approximation around the current parameter estimate. Let \( \hat{\theta}^{(k)} \) be the current parameter function estimate. The parameter function estimate in a point \( i \) given an update \( v \) can be written as
\[ u_{\hat{\gamma}(k)}(x_i) = u_{\hat{\gamma}(k-1)}(x_i) + v \]
which means that the loss function for observation \( i \) can be approximated as
\[ \ell(u_{\hat{\gamma}(k)}(x_i); y_i) = \ell(u_{\hat{\gamma}(k-1)}(x_i) + v; y_i) \approx \ell(u_{\hat{\gamma}(k-1)}(x_i); y_i) + g_i^{(k)} \cdot v + \frac{1}{2} h_i^{(k)} \cdot v^2 \] (3.27)
where \( g_i^{(k)} \) are the derivatives from (3.23) and
\[ h_i^{(k)} = \frac{\partial^2}{\partial^2 \theta} \ell(\hat{\theta}; y_i) \bigg|_{\theta = u_{\hat{\gamma}(k-1)}(x_i)} \]
are second derivatives of the loss function evaluated at the current parameter estimate. Note that the first term in (3.27) is constant with respect to \( v \). Using the approximation in (3.27), the tree for the boosting step is then fitted as
\[ \hat{\nu}^{(k)} = \arg\min_{\nu \in \mathcal{N}} \sum_{i=1}^{n} \left\{ g_i^{(k)} \cdot v_{\nu}(x_i) + \frac{1}{2} h_i^{(k)} \cdot v^2(x_i) \right\} \] (3.29)

Note that the approximate loss function in (3.29) is convex in \( v \), meaning that this tree can be trained using regular methods of regression trees (see Section 3.5.2), with optimal parameter \( \hat{\delta}_l^{(k)} \) for a region \( \hat{A}_l \) given by (3.14) as
\[ \hat{\delta}_l^{(k)} = \frac{-\sum_{i: x_i \in \hat{A}_l} g_i^{(k)} \sum_{i: x_i \in \hat{A}_l} h_i^{(k)}}{\sum_{i: x_i \in \hat{A}_l} h_i^{(k)}}. \] (3.30)

The XGBoost algorithm thus skips the adjustment step (3.25) and instead uses the \( \delta \) from (3.30) directly. This multitasking decreases the computational complexity of the algorithm, making training faster. Note that the step taken by the
XGBoost algorithm has similarities with a so-called Newton step used in many optimization algorithms, see e.g. Boyd and Vandenberghe (2004, Chapter 9.5). Thereafter, the XGBoost parameter function estimate is updated using (3.26), whereafter the process is repeated until \( k = \kappa \). The algorithm also introduces regularization terms to the loss function as well as faster tree training by different approximations that are not discussed here. For details on this, see Chen and Guestrin (2016). According to Chen and Guestrin (2016), this does not sacrifice performance, and the algorithm is widely used.

The Taylor expansion idea is also used in, e.g., the LightGBM algorithm (Ke et al. 2017), which also utilizes so-called histogram-based splits (see Section 3.5.2) to further increase computational efficiency.

### 3.5.3.2 DeltaBoost

DeltaBoost (Lee and Lin 2018) is another boosting model, which can be defined as

\[
\mathcal{M}_{\text{DeltaBoost}} = \{ u_\gamma : \mathbb{R}^p \to \mathbb{R} | \gamma \in \Gamma \}
\]

where the functional form can be stated as

\[
u_{\gamma}(x) = \theta^{(0)} + \sum_{k=1}^{\kappa} \epsilon_k v_{\nu(k)}(x).
\]

As apparent, the functional form and parameter space are identical to those of a GBM or XGBoost.

In DeltaBoost, however, the trees are not fitted to the gradient of the loss function given the current model. Instead, they directly minimize the loss function. The training algorithm is initialized like a GBM, i.e., by solving (3.22). Given model parameter \( \hat{\gamma}^{(k-1)} \), the tree of this boosting step is trained by directly solving

\[
t^{(k)} = \arg \min_{\nu \in \mathcal{N}} \sum_{i=1}^{n} \ell(u_{\hat{\gamma}^{(k-1)}}(x_i) + v_{\nu}(x_i); y_i)
\]  

(3.31)

where \( v_{\nu} \) is a regression tree (see Section 3.5.2). As long as \( \ell \) is convex, an approximate solution to (3.31) can be achieved by using the techniques described in Section 3.5.2. This way, steps (3.23)–(3.24) in the GBM training are replaced by solving (3.31) directly, which according to Lee and Lin (2018) improves computational efficiency. The name delta boost stems from the terminal node parameters in the tree obtained by solving (3.31) can be seen as approximations of the individual loss minimizers

\[
\delta^{(k)} = \arg \min_{\delta \in \mathbb{R}} \ell(\theta^{(k)}(x_i) + \delta; y_i)
\]

which Lee and Lin (2018) refer to as \( \delta \)'s. Lee and Lin (2018) show that delta boosting is optimal for a number of different distributions and loss functions. Optimality is, in Lee and Lin (2018), defined as the state where the individual
splits in the tree structure resulting from (3.31) are such that one can not decrease the loss by moving any observation to the other node. They also note that the delta boosting algorithm is not optimal for all distributions and loss functions, and that for distributions where optimality does not hold, one can instead fit trees by fitting a tree structure as

$$\hat{A}^{(k)} = \arg\min_A \sum_{l=1}^{m} \sum_{i:x_i \in A_l} \ell(u_{\gamma(k-1)}(x_i) + \Delta_l; y_i)$$

(3.32)

where $A$ is a tree partition (see Section 3.5.2) and $\Delta_l$ is defined as

$$\Delta_l = \sum_{i:x_i \in A_l} \frac{g_i^{(k)}}{h_i^{(k)}}$$

where $g_i^{(k)}$ and $h_i^{(k)}$ are the first and second derivatives of the loss function defined in (3.23) and (3.28) respectively. The tree parameters $\delta^{(k)}$ are then set to $\Delta_1, \ldots, \Delta_m$. The procedure is similar to the XGBoost algorithm, see Section 3.5.3.1, and will yield the same output parameters $\delta$ for a given node as XGBoost but not necessarily the same tree structure since the loss functions are not identical.

In Lee (2020) and Lee (2021), DeltaBoost is extended to allow for $u_{\gamma}: \mathbb{R}^p \rightarrow \mathbb{R}^2$ using the asymptotic $\Delta$’s from above but adjusted for the multivariate case as

$$\Delta_{jl} = \sum_{i:x_i \in A_l} \frac{g_{ij}^{(k)}}{h_{ij}^{(k)}}$$

where $g_{ij}^{(k)}$ and $h_{ij}^{(k)}$ are the first and second partial derivatives of the loss function with respect to the $j$:th parameter of the $i$:th observation, $j = 1, 2$. Trees are fitted using (3.32) individually for both parameters, leading to a cyclic training algorithm. The procedure is similar to the Cyclic Gradient Boosting algorithm presented in Paper III, see Section 5.3. The authors use the extension to fit a model to insurance data using the Negative Binomial distribution (Lee 2020) and a zero-inflated Poisson distribution (Lee 2021) respectively.

### 3.5.3.3 NgBoost

In Duan et al. (2020), the NgBoost algorithm is introduced. This algorithm differs from the previous GBM algorithms in that it allows for multivariate output. The model class can be stated as

$$\mathcal{M}_{\text{GBM}} = \{ u_\gamma : \mathbb{R}^p \rightarrow \mathbb{R}^d | \gamma \in \Gamma \}$$
where the functional form can be stated as

$$u_\gamma(x) = \theta^{(0)} + \sum_{k=1}^{\kappa} \epsilon_k v_{\nu(k)}(x).$$

where $v_{\nu(k)}(x)$ is the output of $d$ regression trees, i.e.

$$v_{\nu(k)}(x) = \left[v_{\nu_1(k)}(x), \ldots, v_{\nu_d(k)}(x)\right]^T.$$  \hspace{1cm} (3.33)

and $\theta^{(0)} \in \mathbb{R}^d$. Note that the structure of the model is the same as for the previously presented GBMs but extended to map to $\mathbb{R}^d$ instead of $\mathbb{R}$ by using a multivariate intercept $\theta^{(0)}$ and several trees $v_{\nu(j)}$ per boosting step. For simplicity, assume that the loss function $\ell$ is here the negative log-likelihood of a multivariate exponential family distribution.

The model is initiated just like a GBM, i.e., by solving

$$\hat{\theta}^{(0)} = \arg\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{n} \ell(\theta; y_i).$$

One way to extend the GBM training algorithm to the multivariate case is to fit a multivariate tree to the gradients of the loss function with respect to the parameter function estimate, i.e., to given model parameter $\hat{\gamma}^{(k-1)}$, the partial derivatives are calculated as

$$g_i^{(k)} = \left[\frac{\partial \ell(\theta; y_i)}{\partial \theta_1}, \ldots, \frac{\partial \ell(\theta; y_i)}{\partial \theta_d}\right]^T \bigg|_{\theta = u_\gamma^{(k-1)}(x_i)}$$  \hspace{1cm} (3.34)

Duan et al. (2020) argue that this approach is not optimal, as the partial derivatives in (3.34) do not necessarily live on the same scale for different parameter dimensions. Instead, they propose to consider the so-called generalized natural gradients, which are defined as the direction of the steepest descent in the space of model parameters, which are invariant to the parameterization of the model.

In the case where $\ell$ is the convex and twice differentiable negative log-likelihood of some probability distribution it will hold that

$$g_i^{(k)} \propto \left[\frac{\partial^2 \ell(\theta; y_i)}{\partial \theta_1^2}, \ldots, \frac{\partial^2 \ell(\theta; y_i)}{\partial \theta_1 \partial \theta_d}, \ldots, \frac{\partial^2 \ell(\theta; y_i)}{\partial \theta_d^2}\right]^T \times I(\theta; y_i)^{-1} \bigg|_{\theta = u_\gamma^{(k-1)}(x_i)}$$  \hspace{1cm} (3.35)

where $I(\theta; y_i)$ is the observed Fisher information matrix of observation $i$, i.e.

$$I(\theta; y_i) = -\begin{bmatrix}
\frac{\partial^2 \ell(\theta; y_i)}{\partial \theta_1^2} & \cdots & \frac{\partial^2 \ell(\theta; y_i)}{\partial \theta_1 \partial \theta_d} \\
\vdots & \ddots & \vdots \\
\frac{\partial^2 \ell(\theta; y_i)}{\partial \theta_d \partial \theta_1} & \cdots & \frac{\partial^2 \ell(\theta; y_i)}{\partial \theta_d^2}
\end{bmatrix}. $$
After this, \( d \) univariate regression trees are fitted to the negative generalized natural gradients in parallel, i.e. for \( j = 1, \ldots, d \),

\[
\hat{\nu}_j^{(k)} = \arg\min_{\nu \in \mathbb{N}} \sum_{i=1}^{n} (v_\nu(x_i) + g_{ij}^{(k)})^2 \tag{3.36}
\]

where \( g_{ij}^{(k)} \) is the \( j \)-th element of the natural gradient vector in (3.35). The trees can be fitted using the techniques described in Section 3.5.2.

After fitting the trees, a line search is performed to find the optimal step size \( \rho \) in the direction of the trees as

\[
\rho^{(k)} = \arg\min_{\rho \in \mathbb{R}} \sum_{i=1}^{n} \ell(u_\theta^{(k-1)} + \rho v_\nu^{(k)}(x_i); y_i) \tag{3.37}
\]

where \( v_\nu^{(k)}(x) \) is defined in (3.33). Finally, the parameter function estimate is updated as

\[
u_\theta^{(k)}(x) = u_\theta^{(k-1)}(x) + \epsilon_k \rho^{(k)} v_\nu^{(k)}.
\tag{3.38}
\]

The boosting procedure of (3.35)–(3.38) is updated \( \kappa \) times or until reaching some stopping criterion.

Note that in the case where \( d = 1 \), the natural gradient is

\[
g_i^{(k)} = -\frac{\partial \ell(\theta; y_i)}{\partial \theta} \cdot \left( \frac{\partial^2 \ell(\theta; y_i)}{\partial \theta^2} \right)^{-1} \bigg|_{\theta = \theta^{(k-1)}(x_i)} = -\frac{g_i^{(k)}}{h_i^{(k)}} \tag{3.39}
\]

where \( g_i^{(k)} \) is the regular partial derivative defined in (3.23) and \( h_i^{(k)} \) is the secondary derivative defined in (3.28). The natural gradient form of (3.39) makes the algorithm similar to the XGBoost algorithm, with the difference being that the trees in NGBoost are fitted to (3.39) using a sum-of-squares loss function and thereafter adjusted using (3.25), while the trees in XGBoost are fitted directly using (3.29) without adjustment.
Chapter 4

Non-life insurance applications

Non-life insurance is insurance that covers property, business, and other things that are not explicitly connected to the life and death of an individual. The most common types of non-life insurance are motor, property, and liability insurance. An insurance policy is a contract between an insurer and an insured, where the insured pays a premium to the insurer in exchange for a promise to pay for certain losses or damages suffered by the insured. This contract is a risk transfer from the insured to the insurer. For the insurer, i.e., the insurance company, to carry the risk of many individuals, the insurer needs to predict the risk it assumes from its policyholders. The field of non-life insurance is vast, and the applications suggested in this thesis mainly cover two topics: reserving and pricing. The following two sections define these problems and present a method of solving them using the regression methods described in the previous chapter.

When handling real data, especially insurance data, it is important to note that solving these problems using machine learning methods often becomes a balancing act between accuracy and interpretability. For example, policyholders can make demands on the insurer to explain why they are paying a certain premium. More interpretable models, such as GLMs, can be superior to black-box-type models, such as GBMs, in those settings. The interpretability problem of boosting machines is one of the critical problems that led to the creation of the boosted varying coefficient model in Paper V. Paper III also proposes that the CGBM will yield more interpretable results than other multivariate GBMs. Paper IV discusses interpretability in the presence of missing values.

It should be mentioned that apart from papers I and II, the papers in this thesis describe methods that apply to a wide range of regression problems, and not only to non-life insurance applications. However, further application areas are
4.1 Reserving

An insurance company needs to be able to estimate the number of outstanding payments it will have to make for the portfolio of insurance policies it currently holds. This requirement is both for its liquidity and regulatory purposes. The prediction of these payments is called **reserving**. The following description of the reserving problem is based on Wüthrich (2023, Section 9). The most basic data structure used in reserving is the **claims triangle** – a matrix of claims payments for a portfolio of insurance policies. Table 4.1 shows an example of a claims triangle. The triangle in Table 4.1 shows observed total claims payments $C_{ij}$ for accident year $i$ and development year $j$ are shown from the perspective of the insurance company at the end of development year 3. The empty cells are unknown payments.

<table>
<thead>
<tr>
<th>Accident year</th>
<th>Development year</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4</td>
</tr>
<tr>
<td>1</td>
<td>$C_{11}$ $C_{12}$ $C_{13}$ $C_{14}$</td>
</tr>
<tr>
<td>2</td>
<td>$C_{21}$ $C_{22}$ $C_{23}$</td>
</tr>
<tr>
<td>3</td>
<td>$C_{31}$ $C_{32}$</td>
</tr>
<tr>
<td>4</td>
<td>$C_{41}$</td>
</tr>
</tbody>
</table>

Table 4.1: Example of a claims triangle. The observed cumulative payments $C_{ij}$ for accident year $i$ and development year $j$ are shown from the perspective of the insurance company at the end of development year 3. The empty cells are unknown payments.

payments $C_{ij}$ for accident year $i$ and development year $j$. That means the sum of all payments for accidents that occurred in accident year $i$ and were paid out within $j$ calendar years after the accident. A claims triangle is shown from the perspective of the insurance company at the end of development year $m$. Thus, all payments of claims that occurred in the first accident year and were made within $m$ years after the accident are known. Payments for accidents that occurred in the $m$:th accident year are only known if they were made in the same year as the accident occurred. Usually, a cut-off point for payment delays, $J$, is chosen, and the triangle does not include payments made later than $J$ years after the accident. It will be assumed that $J = m$ in the remainder of this text (as in Table 4.1). This assumption is straightforward to relax if needed.

To formalize the reserving problem further, introduce $\mathcal{F}_m$ as the information available to the insurance company at the end of development year $m$. The payments made up to and including development year $m$ are observed, and hence

$$\{C_{ij} : i + j \leq m + 1\} \subset \mathcal{F}_m.$$
The reserve in this setting is the total amount of yet unknown payments for claims from accidents that have occurred in these $m$ accident years. This can be expressed as

$$R = \sum_{i=1}^{m} (C_{im} - C_{i,m-i+1})$$

Note that this, given the information in $\mathcal{F}_m$, is a random variable if $m < J$. The main goal of reserving is to predict $\hat{R} = \mathbb{E}[R|\mathcal{F}_m]$, which means estimating the unknown payments $\hat{C}_{im} = \mathbb{E}[C_{im}|\mathcal{F}_m]$ for each $i$.

The most common baseline model for reserving is the chain-ladder method. The procedure is described in Mack (1993) and predicts the cumulative payments $C_{ij}$ using the assumption

$$\mathbb{E}[C_{i,j+1}|C_{i,1},\ldots,C_{i,j}] = C_{ij} \cdot f_j$$

with development factors $f_j$, which are estimated as

$$\hat{f}_j = \frac{\sum_{i=1}^{m-j} C_{i,j+1}}{\sum_{i=1}^{m-j} C_{ij}}$$

for $j = 1,\ldots,m-2$. Equation (4.1) means that for the $m$:th development year, the predicted cumulative payments can be estimated as

$$\hat{C}_{im} = C_{i,m-i+1} \prod_{j=m-i+1}^{m-1} \hat{f}_j.$$

Mack (1993) also includes a method for estimating the variance of the reserve by estimating the variance of a cumulative payment given the previous cumulative payment as

$$\text{Var}(C_{i,j+1}|C_{i,1},\ldots,C_{i,j}) = \hat{\sigma}_j^2 \cdot C_{ij}.$$  

which makes the variance $\text{Var}(C_{im}|\mathcal{F}_m)$ estimable using a recursive formula. The variance parameters $\hat{\sigma}_j^2$ are estimated using

$$\hat{\sigma}_j^2 = \frac{1}{m-j-1} \sum_{i=1}^{m-j} \left( \frac{C_{i,j+1}}{C_{ij}} - \hat{f}_j \right)^2$$

for $j = 1, 2, \ldots, m - 2$.

The chain-ladder model is a distribution-free model, which means that it does not make any assumptions about what distribution the payments follow. While the structure in (4.1) and (4.2) makes it possible to estimate the first two moments, it does not provide a complete distributional description of the reserve. This can be problematic for regulatory purposes, where the insurer is often to provide further measures of uncertainty. The European Union’s Solvency II
directive, for example, requires insurance companies to calculate the Value at Risk (VaR) of the reserve. Under certain distributional assumptions accepted by the Solvency II guidelines, this can be done by estimating the expected value and variance of the reserve (Wahl 2019). However, the chain-ladder model does not provide a complete distributional description of the reserve, and hence, it is not possible to calculate the VaR using the chain-ladder model without further assumptions (see Merz and Wüthrich (2008) for an example of how this can be done).

A more modern approach to reserving is to assume that the payments follow some probability distribution, and then model the parameters of this distribution using regression models. A more detailed and specific data structure is useful to accomplish this goal. First, assume that the number of claims \( N_{ij} \) from accidents that occur in accident year \( i \) and are reported during the \( j \):th calendar year after the accident. From the perspective of an insurer at the end of the \( m \):th year, the observed number of claims are

\[
N_m = \{N_{ij} : i + j \leq m + 1\}
\]

where one can note that, using the same notation for the currently known information as above, \( N_m \subset F_m \). Then, let \( Y_{ijk} \) denote the total amount of payments from claims where the accident occurred in year \( i \), the claim was reported during the \( j \):th calendar year after the accident and the payment was made during the \( k \):th calendar year after the claim was reported. The observed payments are then

\[
Y_m = \{Y_{ijk} : i + j + k \leq m + 1\}.
\]

In this setting, \( Y_m \subset F_m \), i.e., the past payments are viewed as observed information.

**Remark 4.1.** The cumulative payments \( C_{ij} \) in the claims triangle can be expressed as

\[
C_{ij} = \sum_{k=1}^{j} Y_{i,j-k+1,k}.
\]

A useful approach is to model the number of claims \( N_{ij} \) and the payments \( Y_{ijk} \) as random variables, and then use a suitable model class to perform parameter regression (see Chapter 3). To start with the number of claims, assume they are stochastic processes following some known distribution, i.e.

\[
N_{ij} \sim F(\nu_{ij}), \tag{4.3}
\]

where \( F \) is a discrete distribution described by some parameter \( \nu_{ij} \) that depend on the accident year \( i \) and reporting delay \( j \). Similarly, the payments \( Y_{ijk} \) can be modeled as

\[
Y_{ijk} | N_{ij} \sim G(\psi_{ijk}; N_{ij}), \tag{4.4}
\]

where \( G \) is a continuous distribution with explicit dependence on \( N_{ij} \) and parameter \( \psi_{ijk} \). If one is then able to estimate the parameters \( \nu_{ij} \) and \( \psi_{ijk} \),
the reserve can be seen as a sum of random variables with known distributions and estimated parameters, meaning that both the expected value and variance of the reserve can be estimated.

An example of a relatively recent stochastic framework for reserving is the *Collective Reserving Model* (CRM) introduced by Wahl et al. (2019). While the CRM is based on assumptions about claims on a micro-level, it can be fitted to aggregated data as above. The model assumes that the number of claims $N_{ij}$ follow an *overdispersed* Poisson distribution, i.e.

$$N_{ij} \sim \text{ODP}(\nu_{ij}, \phi),$$

where the overdispersed Poisson distribution is defined such that

$$\mathbb{E}[N_{ij}] = \nu_{ij}$$
$$\text{Var}(N_{ij}) = \phi \nu_{ij}.$$  

The number of payments from these claims are assumed to follow a Poisson distribution, i.e.,

$$N_{ijk}^{\text{paid}} | N_{ij} \sim \text{Pois}(N_{ij} \lambda_{ijk}),$$

whereafter the payments $Y_{ijk}$ are assumed to be sums of $N_{ijk}^{\text{paid}}$ individual payments, i.e.

$$Y_{ijk} = \sum_{l=1}^{N_{ijk}^{\text{paid}}} P_{ijkl},$$

where $N_{ijk}^{\text{paid}}$ is the number of payments stemming from accidents that occurred in year $i$, were reported in year $j$, and were paid within $k$ years after being reported.

The individual payments $P_{ijkl}$ are assumed to be independent of each other for each $i$, $j$, and $k$, and follow some distribution such that

$$\mathbb{E}[P_{ijkl}] = \mu_{ijk}$$
$$\text{Var}(P_{ijkl}) = \sigma_{ijk}^2.$$  

which means that the first two moments of the payments $Y_{ijk}$ stemming from reported claims are

$$\mathbb{E}[Y_{ijk}|N_{ij}] = N_{ij}\lambda_k \mu_{ijk}$$
$$\text{Var}(Y_{ijk}|N_{ij}) = \frac{\mu_{ijk}^2 + \sigma_{ijk}^2}{\mu_{ijk}} \mathbb{E}[Y_{ijk}|N_{ij}], \quad (4.5)$$

Further, if the assumption that the overdispersion term in (4.5) is constant, i.e.,

$$\frac{\mu_{ijk}^2 + \sigma_{ijk}^2}{\mu_{ijk}} = \phi,$$

47
and if one introduces a parameter \( \psi_{ijk} = \mu_{ijk} \lambda_k \), then Wahl et al. (2019) claim that the payments \( Y_{ijk} \) have a quasi-likelihood proportional to an overdispersed Poisson distribution. The concept of quasi-likelihood will not be discussed further here, but the interested reader is referred to McCullagh and Nelder (1989, Chapter 8).

Whether one uses the CRM or some other structure (e.g. some other distributions \( F \) and \( G \) in (4.3) and (4.4)), one can use the negative log-likelihoods (or, in the case of the CRM, the negative quasi-log-likelihoods) of the corresponding distributions as loss functions for the model classes presented in Section 3.2, and use the \( i, j, \) and \( k \) values as features. The parameters \( \nu_{ij} \) and \( \psi_{ijk} \) can then be modeled using regression models, whereafter the reserve can be predicted.

To construct the feature vector \( \mathbf{x} \) used for the regression models of the parameters in the distributions in a reserving model, the most straightforward approach would be to view it as a three-dimensional vector of ordered integers,

\[
\mathbf{x} = \begin{pmatrix} i \\ j \\ k \end{pmatrix}. \tag{4.6}
\]

This would however mean that, if modeling the parameters \( \nu_{ij} \) and \( \psi_{ijk} \) using GLMs, one would assume linearity in the accident and development years. Since that assumption is not necessarily reasonable, one could instead view the accident and development years as categorical variables and use one-hot encoding to represent them. The features for \( h \) can be stated as a stacked vector

\[
\mathbf{x} = \begin{pmatrix} \mathbf{x}_i \\ \mathbf{x}_j \\ \mathbf{x}_k \end{pmatrix}
\]

where

\[
\mathbf{x}_i = \begin{pmatrix} 1 \{i=1\} \\ \vdots \\ 1 \{i=m\} \end{pmatrix}, \quad \mathbf{x}_j = \begin{pmatrix} 1 \{j=1\} \\ \vdots \\ 1 \{j=m\} \end{pmatrix}, \quad \mathbf{x}_k = \begin{pmatrix} 1 \{k=1\} \\ \vdots \\ 1 \{k=m\} \end{pmatrix}. \tag{4.7}
\]

That is, the feature vector contains indicators (0 or 1) signifying which accident year, reporting delay, and payment delay the observation lies in.

An example of an approach to model the parameter \( \psi_{ijk} \) in the CRM model is to use a GLM with a log-link function (see Section 3.5.1). Using the the one-hot encoded feature vector from (4.7), the model can be written as

\[
\psi_{ijk} = \exp(\beta_i + \beta_j + \beta_k). \tag{4.8}
\]

where \( \beta_i \) is the coefficient for the indicator for accident year \( i \) in (4.7), and so on. This model has \( 3m \) parameters.

**Remark 4.2.** Note that one needs to use a structure such that the parameters are estimable from the data in \( F_m \). This means that the models in question need
to be able to extrapolate to combinations of \(i, j,\) and \(k\) such that \(i+j+k > m+1.\) The model in (4.8) is estimable from the data in \(\mathcal{F}_m.\) However, if one instead uses a model such as

\[
\psi_{ijk} = \exp(\beta_{ijk}),
\]

where \(\beta_{ijk}\) is a parameter for each combination of \(i, j,\) and \(k,\) then the model is not estimable from the data in \(\mathcal{F}_m\) since all combinations of \(i, j,\) and \(k\) are not observed.

Papers I and II are concerned with reserving and use the CRM as a basis. In Paper I, the CRM is fitted using a series of different classes of machine learning regression models. In Paper II, the CRM is extended to include the openness status of claims as a factor. The model is then fitted using a GLM. For more on the CRM, see Paper I, Paper II, or the original sources Wahl et al. (2019) and the doctoral thesis of Wahl (2019).

4.2 Premium calculation

Premium calculation and pricing in non-life insurance is a vast field. While it is not explicitly used as an application in any of the papers in this thesis, papers III and V showcase how their respective regression model classes can be used to estimate distributions of claim counts and claim amounts. The practical usefulness of this application is not explicitly discussed in the papers, why a short intro on how these regression models can be utilized in premium calculation will be covered in this section.

The point of calculating a risk premium is to price an insurance policy in a way that is fair to both the insurer and the insured. This means that the calculation has to be based on reasonable assumptions and that it has to incorporate not only the expected payments to be made from the insurer to the insured, but also the uncertainty of these payments. Assuming that the insurer has some way to estimate the expected value and variance of payments \(Y_i\) to policyholder \(i,\) the so-called standard deviation principle is a pricing method saying that the premium of such a contract should be

\[
\pi_i = \mathbb{E}[Y_i] + \alpha \cdot \sqrt{\text{Var}(Y_i)}.
\]  

(4.9)

for some positive constant \(\alpha\) (Mikosch 2009, p. 85). The constant \(\alpha\) is called the security loading and varies with the insurer’s risk aversion. In general, an insurer with a large number of policies can afford to have a lower security loading than an insurer with a small number of policies without risking bankruptcy. For other pricing principles, see e.g. Mikosch (2009, Chapter 3.1.3) or Wüthrich (2023, Section 6).

Setting a premium is then a problem of modeling claim amounts, which can be done by introducing some distributional assumptions. Assume that the number
of claim amounts $N_i$ for policyholders $i = 1, \ldots, n$ are independent and follows some discrete distribution $F$, i.e.

$$N_i \sim F(\theta_i),$$

where $\theta_i$ is a vector of parameters for the distribution. Also, assume the individual claim amounts $Y_{ij}$ for policyholder $i$ are independent and identically distributed, i.e.

$$Y_{ij} \sim G(\phi_i),$$

where the total claim cost is $Y_i = \sum_{j=1}^{N_i} Y_{ij}$. Also assume that the insurer has access to a set of features specific to each policyholder, in the form of a feature vector $x_i$. By assuming that the parameters $\theta_i$ and $\phi_i$ are functions of the feature vector $x_i$, one can view this as a regression problem (see Chapter 3).

For a thorough discussion on creating pricing models using GLMs, see Ohlsson and Johansson (2010).

Using the model classes described in Chapter 3, or the ones introduced in Papers III and V, one can estimate the functions and thus predict the number of claims and the claim amounts for a new policyholder given his or her feature vector $x_i$, as well as their full distributions. These parameter function estimates make it possible to estimate the moments used in (4.9).
Chapter 5

Overview of papers

5.1 Paper I

Paper I, ‘Machine learning, regression models, and prediction of claims reserves’, concerns the reserving problem presented in Section 4.1. The paper uses the Collective Reserving Model (CRM) framework described in Section 4.1 and in Wahl et al. (2019). This means that, in order to produce reserve estimates, one needs to estimate the expected number of claims and the expected amount of payments per claim, which under the CRM assumptions and using a log-link is equivalent to estimating the unknown parameter functions $\nu_{ij}$ and $\psi_{ijk}$ (see Section 4.1). In Wahl et al. (2019), the corresponding functions are estimated using GLMs. In Paper I, the parameter functions are instead estimated using GBMs (described in Section 3.5.3) and neural networks in order to allow for more flexible models with non-linear effects and interactions between the factors $i$, $j$, and $k$.

A neural network is a model class that can approximate any continuous function arbitrarily well, given a sufficient number of parameters (this means it can reach a saturated model, see Section 3.3). Since the model class is only used in Paper I, it will not be described in detail here, and the reader is referred to the introduction provided in the paper or e.g. Hastie et al. (2009) for a more thorough description.

In order to be able to use early stopping for these methods, training- and validation split for this specific problem has to be considered, which is non-trivial due to the data structure (see Section 4.1). A method of doing this based on sub-portfolios of insurance policies is presented in the paper.

The main contribution of the paper is to create a framework for using machine learning regression models in the reserving setting using CRM-type models. The paper also shows promising results for the models, which outperform the GLM formulations on a series of data sets simulated using Gabrielli and Wü-
A potential, yet unexplored, extension of the paper would be to design a validation method that does not rely on sub-portfolios, in order to be able to use the models on data that is already aggregated into run-off triangles, which is the standard format for reserving data. This could, potentially, be done by splitting the aggregated data points into training and validation sets.

5.2 Paper II

The idea behind Paper II, ‘A collective reserving model with claim openness’, was to create a more flexible model than the CRM model, in order to model the non-linearities implied by the results of Paper I. Instead of using machine learning models, the goal was to create a model that would not lose the interpretability of the CRM model, and that was estimable using GLMs. The data used in Paper I contained more granular information than the one used, namely in the form of the opening and closing history of all individual claims. The CRM model does not explicitly use this information. The Collective Reserving Model with Openness (CRMO) was created to make use of this data to produce more accurate reserve estimates.

Considering the structure and definition of $N_{ij}$ as defined in Section 4.1, the CRMO uses a (overdispersed) Poisson distribution as $F$, just like the CRM. Then, another layer of data is introduced between the reported claims $N_{ij}$ and the payments $Y_{ijk}$, namely the number of open claims $N_{ij}^{open}$. The number of open claims can be stated as the sum of claims that have stayed open and claims that have been re-opened, i.e.,

$$N_{ij}^{open} = N_{ij}^{stay-open} + N_{ij}^{re-open}.$$

These individual numbers are then assumed to follow binomial distributions, i.e.

$$N_{ij}^{stay-open} | N_{i,j,k-1}^{open} \sim \text{Bin}(N_{i,j,k-1}^{open}, p_{ijk})$$

$$N_{ij}^{re-open} | N_{i,j,k-1}^{closed} \sim \text{Bin}(N_{i,j,k-1}^{closed}, q_{ijk}).$$

Using a sigmoid link function

$$\text{sigm}(x) = \frac{1}{1 + e^{-x}}$$

to map from $\mathbb{R}$ to $[0, 1]$, the probabilities $p_{ijk}$ and $q_{ijk}$ can then be estimated using some regression model. The paper uses a GLM to do this.

A caveat of the re-open and stay-open split in (5.1) and (5.2) is that it requires the data to be aggregated in a particular way. While this is not unrealistic in real-life settings, a simpler version of the CRMO can also be defined in order to
apply it to aggregated data sets, where the number of open claims are assumed to follow a binomial distribution, i.e.,
\[ N_{ij}^{\text{open}} | N_{ij}^{\text{open}} \sim \text{Bin}(N_{ij}^{\text{open}}, \tilde{p}_{ijk}). \]
which requires less data but does not utilize the information from the openness status to the same extent.

The total claim amounts are modeled as
\[ Y_{ijk} | N_{i,j,k}^{\text{open}} - 1 \sim G(\psi_{ijk}; N_{i,j,k}^{\text{open}} - 1), \]
instead of the equivalent distribution for \( Y_{ijk} \) in the CRM model, which is expressed as conditional on \( N_{ij} \).

The CRMO, fitted using GLMs, is evaluated on the same data sets as in Paper I. The model shows promising results and is even able to outperform the GBM and neural network formulations of the CRM model in some data sets. Note that the GLM is also more interpretable than a GBM or a neural network, highlighting another benefit of the CRMO approach.

In order to further improve accuracy, the machine learning implementation of the CRM used in Paper I could be extended to fit the model structure of the CRMO. This would mean that the probability of staying open and re-opening could also be estimated using a neural network or a GBM, or that a combination of GLMs and machine learning models could be used for the different layers of the CRMO. This is not explored in the paper, but is a potential extension of the work that could provide an interesting balance between flexibility and interpretability.

5.3 Paper III

Paper III, *On cyclic gradient boosting machines*, presents a novel machine learning model known as the *Cyclic Gradient Boosting Machine* (CGBM). The idea behind the model was born when considering the overdispersed distributions considered in Papers I and II. In both of those papers, the mean parameter of those distributions was modeled using regression models, while the dispersion parameter was assumed to be constant. The data indicated that the assumption of constant overdispersion was not satisfied, implying that there could be a benefit to modeling the dispersion parameter using some regression model as well. This modeling could be done using, e.g., the neural network framework presented in Paper I, but the gradient boosting machine used in the same paper does not inherently support multidimensional output. By combining the idea of cyclic coordinate descent (see Wright (2015) for details on cyclic coordinate descent) with the gradient boosting machine, the CGBM model class was created. It can be defined as
\[ \mathcal{M}_{CGBM} = \{ u_\phi : \mathcal{X} \to \mathbb{R}^d | \phi \in \Phi \} \]
where $\phi = \{\gamma_1, \ldots, \gamma_d\}$ is the parameters that define the model, and $\gamma_j \in \Gamma$ are the parameters of a univariate GBM defined in Section 3.5.3, for $j = 1, \ldots, d$. The output of a CGBM can be defined as

$$u_{\phi}(x) = \begin{pmatrix} v_{\gamma_1}(x) \\ \vdots \\ v_{\gamma_d}(x) \end{pmatrix}$$

where $v_{\gamma_j} \in \mathcal{M}_{\text{GBM}}$ for $j = 1, \ldots, d$ are univariate, but dependent, GBMs.

The training of a CGBM is similar to the training of a GBM but with an inner loop that iterates over the dimensions of the parameter function. This produces a sequence of model parameters that are easiest to describe using the notation $\hat{\phi}^{(j,k)}$ where $k = 1, \ldots, \kappa$ is the boosting step and $j = 1, \ldots, d$ is the parameter dimension last updated in that step. The gradients of (3.23) are replaced with partial derivatives, i.e.

$$g_{ij}^{(k)} = \frac{\partial}{\partial \theta_j} \ell(\theta; y_i) \Big|_{\theta = u_{\hat{\phi}^{(j,k-1)}}(x_i)}$$

where $u_{\hat{\phi}^{(j-1,k)}}(x_i)$ is the current prediction of the model and $j = 1, \ldots, d$ is the parameter dimension to be updated. Note that for $j = 1$, the current prediction has to be replaced by $u_{\hat{\phi}^{(d,k-1)}}(x_i)$.

The hyperparameters are the same as for the GBM but can be varied over the different dimensions of the output. It is, for example, possible to choose different learning rates $\epsilon_j$ for different dimensions $j = 1, \ldots, d$. Also, an early stopping scheme similar to the one presented in Section 3.3 is presented, that can be used to stop the individual GBMs at different values of $\kappa_j$ for different parameter dimensions $j = 1, \ldots, d$.

While other algorithms deal with multidimensional prediction for gradient boosting machines, such as XGBoost (Chen and Guestrin 2016), Delta Boosting (Lee 2020), and NGBoost (Duan et al. 2020), outlined in Sections 3.5.3.1–3.5.3.3, there are advantages to modeling the different dimensions of the target parameter cyclically. NGBoost, for example, forces the different dimensions to be modeled using the same amount of boosting steps. This could lead to parameter dimension-wise over- or underfitting, which one can avoid by using the CGBM and different $\kappa_j$ values. Also, the structure of the CGBM means that it can provide parameter dimension-wise feature importances (see Section 3.5.3) which increases the interpretability of the model.

The paper shows the strength of the CGBM versus a GBM only modeling one parameter dimension in a simulation study using a variety of distributions from the exponential family. It is also used on a real data set of French car insurance claims, where the model seems to find some structure in the overdispersion parameters that the GBM was forced to model as constant.

Proving convergence for the CGBM training algorithm is not straightforward and is only done under very specific conditions in the paper. The main issue
is that when $d > 1$, there is no guarantee that there exists a solution to
\[ \theta_i^* = \arg\min_{\theta \in \mathbb{R}^d} \ell(\theta; y_i) \]  
(5.3)
that produces a finite loss $\ell(\theta_i^*; y_i)$, meaning that there is no meaningful definition of in-sample convergence. To see that this is indeed a problem, consider the case where the response follows a two-dimensional normal distribution, i.e.,
\[ Y \mid x \sim N(\mu(x), \sigma(x)). \]
Using an identity and log-link function for the mean and dispersion parameters respectively, the parameters of interest are
\[ \theta(x) = \left( \frac{\theta_1(x)}{\theta_2(x)} \right) = \left( \frac{\mu(x)}{\log \sigma(x)} \right). \]
Note that $\ell$ will then be convex with respect to $\theta_1$ and $\theta_2$ individually, but not jointly. A loss function $\ell$ that is proportional to the negative log-likelihood of a normal distribution takes the form
\[ \ell(\theta; y_i) = \theta_2 + \frac{(y_i - \theta_1)^2}{2e^{2\theta_2}} \]
meaning that there exists no solution to (5.3). This can be seen by first solving with respect to $\theta_1$, which gives $\theta_1^* = y_i$. Then, the loss with respect to $\theta_2$ given $\theta_1^* = y_i$ is simply $\theta_2$, meaning there is no finite parameter that minimizes the loss, nor a finite minimum loss. Since the proof used for proving in-sample convergence of a standard GBM is based on convergence to a finite value of the loss function on the training data, it can not be applied in the general sense.

A proposition discussed briefly in the paper is to construct a slightly modified training data set $\left( x_i, y_i, x_{i-n}, y_{i-n} + \varepsilon \right)$, for $i = n+1, \ldots, 2n$, where $\varepsilon > 0$ is a small value. Since the prediction function $u_{\phi}$ of a CGBM can only take on a single output value for a given input $x$, the model will no longer be able to reach in-sample convergence in the sense defined in (3.10) due to the duplicate $x$ values. However, there is now a well-defined minimum training loss
\[ \sum_{i=1}^{2n} \ell(\theta_i^*; y_i) = 2n \log \frac{\varepsilon}{2} + n \]
where the MLE parameter $\theta_i^*$ is defined as the loss minimizer for the two observations with the same $x$ value, i.e.
\[ \theta_i^* = \arg\min_{\theta \in \mathbb{R}^d} \ell(\theta; (y_i, y_i + \varepsilon)) \]
\[ = \arg\min_{\theta \in \mathbb{R}^d} \left\{ 2\theta_2 + \frac{1}{2e^{2\theta_2}} \left( (\theta_1 - y_i)^2 + (\theta_1 - y_i - \varepsilon)^2 \right) \right\}. \]
This is solved by
\[ \theta^*_i = \left( y_i + \frac{\varepsilon}{\log \varepsilon} \right). \]

Since there now exists a minimum possible training loss for the model class, the arguments of the convergence proof of a univariate GBM can be applied to show convergence in this, fictitious setting.

The CGBM algorithm is also available as a Python package on PyPI under the name cyc-gbm. The complete source code can be accessed in the GitHub repository at https://github.com/henningzakrisson/cyc-gbm.

5.4 Paper IV

Paper IV, *Trinary decision trees for handling missing data*, does not explicitly deal with insurance applications. It introduces a new class of tree-based models called *trinary trees*. A trinary tree is a CART but with the binary splits replaced by a trinary split, i.e., a split into three groups instead of two. The third group is created for data points with missing values in the features. The model class can, using the notation of Chapter 3, be described as
\[ \mathcal{M}_{TT} = \left\{ u_\nu : \bar{\mathcal{X}} \to \mathbb{R}^d \mid \nu \in \mathbb{N} \right\}, \]
where \( \bar{\mathcal{X}} = \bar{\mathcal{X}}_1 \times \cdots \times \bar{\mathcal{X}}_p \) is the feature space for a \( p \)-dimensional feature vector, defined analogously to \( \mathcal{X} \) in Section 3.5.2, with the difference that \( \bar{\mathcal{X}}_j = \mathcal{X}_j \cup \{ \text{nan} \} \) where \text{nan} represents a missing value. This means that the model accepts inputs with missing values for all feature dimensions.

The output function can still be described as
\[ u_\nu(x) = \sum_{l=1}^m 1_{\{x \in A_l\}} \delta_l \]
with the difference being that the partition \( A \) is now a partition of the space \( \bar{\mathcal{X}} \) instead of \( \mathcal{X} \).

The parameter \( \nu \) defines the structure of the tree and is defined similarly to the parameter \( \theta \) in Section 3.5.2. The main difference is that the partitions now always include a third node, for observations with missing values. The missing value handling is performed by replacing the binary split of the feature space performed by (3.18)–(3.19) is replaced with
\[
\begin{align*}
\hat{A}_{3l}^{(k+1)} &= \hat{A}_l^{(k)} \cap \hat{B}_{(k,l)} \\
\hat{A}_{3l+1}^{(k+1)} &= \hat{A}_l^{(k)} \cap \bar{\hat{B}}_{(k,l)} \\
\hat{A}_{3l+2}^{(k+1)} &= \hat{A}_l^{(k)} \cap \hat{B}_m^{(k,l)},
\end{align*}
\]
where 
\[ \hat{B}^{m}_{(k,l)} = X_1 \times \cdots \times X_{j-1} \times \{ \text{nan} \} \times X_{j+1} \times \cdots \times X_p, \]
for the same \( j \) as in (3.18)–(3.19). This means that data points where the chosen feature is missing are assigned to a third node. Then, the parameters of the first two nodes are fitted using (3.14), whereas the third node is fitted using
\[
\hat{\delta}(\hat{A}^{(k+1)}_{2l+2}) = \arg\min_{\delta \in \mathbb{R}^d} \sum_{i: x_i \in \hat{A}^{(k)}_i} \ell(\delta; y_i)
\]
where one can note that the data set used to fit the third node is the same as the data set used to fit the parent node. The third node leads to a regularizing effect since it is fitted without making assumptions about the missing values and their relationship to the target variable. The training algorithm is then continued as usual, with the exception that the third node is assigned the complete data set of the parent node instead of the subset created by the split. This is done in order to avoid making any assumptions about the missing values, and instead find another feature to split the data on in case the best feature is missing for a data point.

For an example to illustrate the difference between the trinary tree and other popular methods for handling missing data in trees, consider a binary regression tree with depth 1, trained with a using the data set presented in Table 5.1. Note that observations 4, 5, and 6 are missing their feature values.

<table>
<thead>
<tr>
<th>( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i )</td>
<td>0</td>
<td>1</td>
<td>nan</td>
<td>nan</td>
<td>nan</td>
<td></td>
</tr>
<tr>
<td>( y_i )</td>
<td>0</td>
<td>0</td>
<td>12</td>
<td>0</td>
<td>12</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 5.1: Example data set for the trinary tree example.

Assume that a sum of squared loss \( \ell(\delta; y) = (\delta - y)^2 \) is used. There are a number of different ways to handle the missing feature value.

The majority rule algorithm assigns missing values to the node with the largest number of observations, which in this example means the node that is assigned observations 1, and 2. This results in a tree structure
\[
\hat{A}_1 = \{ x : x \leq 1.5 \text{ or } x \text{ missing} \}, \quad \hat{A}_2 = \{ x : x > 1.5 \}
\]
with parameters
\[
\hat{\delta}_1 = 4.8, \quad \hat{\delta}_2 = 12.
\]

The missing in attributes (MIA) algorithm assigns missing values to the node which results in the lowest total loss. Here, that would mean assigning the missing values to the node with the most similar average response, which is the node with observations 3, having the response 12. This results in a tree structure
\[
\hat{A}_1 = \{ x : x \leq 1.5 \}, \quad \hat{A}_2 = \{ x : x > 1.5 \text{ or } x \text{ missing} \}
\]
with parameters

\[ \hat{\delta}_1 = 0, \quad \hat{\delta}_2 = 9. \]

The trinary tree, on the other hand, would not make assumptions on where the missing data points belong, and would instead create a third node that represents the situation where one simply does not know which of the two ordinary splits to assign the data point to, i.e.

\[ \hat{A}_1 = \{ x : x \leq 1.5 \}, \quad \hat{A}_2 = \{ x : x > 1.5 \}, \quad \hat{A}_3 = \{ x : x \text{ missing} \} \]

and then fit parameters to the ordinary nodes using their assigned data and the third node using the entire data set, as

\[ \hat{\delta}_1 = 0, \quad \hat{\delta}_2 = 12, \quad \hat{\delta}_3 = 6, \]

with \( \hat{\delta}_3 = 6 \) being the average of the responses in the full data set.

The trinary tree will not always be the optimal choice for handling missing data, but in the case where data is Missing Completely at Random (MCAR), the idea behind the trinary tree is to not make any assumptions about the missing data, and instead only split data when being sure as to where it belongs in the tree structure.

The details of training a trinary tree is given in the paper, followed by a proof of an example where other missing value handling methods yield a biased estimate when compared to the trinary tree. The paper also presents a larger numerical illustration, where the trinary tree is shown to yield good results on data sets with MCAR data. For data with so-called Informative Missingness (IM), i.e., when the probability of a missing value depends on the value of the feature, the trinary tree is not as powerful, but an amalgamation between the trinary tree and the Missing in Attributes (MIA) method presented in Twala et al. (2008) is shown to yield good results for both MCAR and IM settings.

Compared to other missing feature handling methods, the trinary tree has an advantage in terms of local interpretability since the path of a single prediction never splits the data using missing features. For example, this would be done in the MIA method, which could cause issues when trying to justify, say, an insurance contract premium.

The trinary trees could have further potential if used as a base learner in a boosting algorithm, which could provide further flexibility and interpretability. This would however require further considerations, since the update step of a boosting algorithm when faced with missing values is not straightforward. Such a boosting algorithm is left as a potential extension of the work and has not been explored yet.

A proof-of-concept python package for the trinary tree is available on PyPI under the name trinary-tree. The complete source code can be accessed in the GitHub repository https://github.com/henningzakrisson/trinary-tree.
5.5 Paper V

Paper V, *A tree-based varying coefficient model*, introduces a tree-boosted Varying Coefficient Model (VCM) class, which can be defined as

$$
\mathcal{M}_{TVCM} = \{u_\lambda : \mathcal{Z} \times \mathbb{R}^p \rightarrow \mathbb{R} | \lambda \in \Lambda \}.
$$

The parameter vector can be written as $\lambda = \{\beta_0, \gamma_1, \ldots, \gamma_p\}$, where $\beta_0 \in \mathbb{R}$ is an intercept term and $\gamma_j \in \Gamma$, $j = 1, \ldots, p$, are parameters for $p$ individual gradient boosting machines. $\mathcal{Z}$ is a feature space for a $q$-dimensional feature vector $\mathbf{z}$, which can contain both categorical and numerical features. $u_\lambda$ has the form

$$
u_\lambda(\mathbf{z}, \mathbf{x}) = \beta_0 + \sum_{j=1}^{p} \beta_j(\mathbf{z}) \cdot x_j,
$$

where $\beta_j = u_{\gamma_j}$ is a gradient boosting machine, defined in (3.21). This is a special case of a varying coefficient model (VCM) (Hastie and Tibshirani 1993), where the coefficients of a linear model are allowed to vary with the input, but not necessarily using GBMs. Note also the similarity between Equations (5.4) and (3.12) - the model presented in the paper is essentially a GLM with every individual parameter $\beta \in \mathbb{R}$ replaced by a function $\beta : \mathbb{R}^p \rightarrow \mathbb{R}$.

The idea comes from the LocalGLMnet, introduced by Richman and Wüthrich (2023). In the LocalGLMnet, the underlying model for the coefficient functions $\beta_j(\mathbf{z})$ is a neural network instead of a CGBM. The idea is to maintain some of the local interpretability of the GLM while allowing for the flexibility of a more complex machine learning model by stating that for feature $\mathbf{x}$, the effect of a one-unit increase in feature $x_j$ is $\beta_j(\mathbf{z})$. Note that this is only approximately true if the feature space $\mathcal{X}_j \subset \mathcal{Z}$, since it can then not be guaranteed that this one-unit increase of $x_j$ does not alter the value of any coefficient function. Note also that in the LocalGLMnet, the feature space $\mathcal{Z}$ is the same as $\mathcal{X}$, whereas in this model, the two spaces are allowed to differ, as in the general VCM.

The paper utilizes the CGBM introduced in Paper III, since the tree-based VCM requires a boosting model with multidimensional output. Contrary to other tree-based boosted VCMs such as Wang and Hastie (2014) or Zhou and Hooker (2022), the VCM in Paper V allows for dimension-wise early stopping, which is a benefit of using the CGBM. This allows for a model that can handle different levels of complexity between the coefficient functions, which should help against overfitting.

The paper introduces the cyclically boosted varying coefficient model, as well as some useful methods for tuning its hyperparameters. The fact that a CGBM can provide dimension-wise feature importance scores also helps in analyzing the feature interaction effects. This capability makes feature selection simpler than in, e.g., a LocalGLMnet model.

A problem that could arise when considering $\mathcal{Z} \subset \mathcal{X}$ is that the model in (5.4) is no longer unique and there are thus several representations of the functional
form that would yield the exact same output. This is discussed and considered in the paper but could be looked into further in future research, particularly since this likely affects the convergence of the model.

The model is run against GBMs, GLMs, and a LocalGLMnet using the same simulated and real data sets as in Richman and Wüthrich (2023). The model shows satisfactory results, being on par with the LocalGLMnet. The source code for the implementation of the model used in the numerical illustrations can be found at https://github.com/henningzakrisson/local-glm-boost under the name LocalGLMBoost.


I Artikel II använder inte en maskininlärningsmetod, utan utvecklar istället CRM som används i Artikel I genom att lägga till försäkringsärendens öppenhet till dynamiken, och presenterar CRM-modellen med öppenhet (CRMO). Tanken bakom modellen är att tillhandahålla ett enkelt, tolkningsbart och intuitivt sätt att modellera vissa icke-linjära effekter som antyds maskininlärningsmodellernas framgång i Artikel I. I artikeln presenteras hur modellen kan skattas med hjälp av regressionsmetoder, och presenterar rekursiva formler för momenten i den estimerade reserven. Algoritmen utvärderas med avseende på träffsäkerhet på samma dataset som i Artikel I och visar resultat som är jämförbara med maskininlärningsimplementeringarna av CRM-modellen.

I Artikel III presenteras en ny boosting-algoritm som kallas Cyclic Gradient Boosting (CGBM). Målet med algoritmen är att utöka den klassiska GBM-
algoritmen till approximation av flerdimensionella funktioner. Artikeln visar
hur detta kan användas för att estimera hela sannolikhetsfördelningar istället
för bara väntevärdet av fördelningen. I artikeln diskuteras också potentiella
problem med hyperparameterinställning för multivariata boosting-algoritmer
och ger en enkel early stopping-modell för att göra detta. Numeriska illustra-
tioner visar mycket träffsäkra resultat på simulerat datamaterial. När algorit-
men tillämpas på ett verklig datamaterial inom motorförsäkring visar den en
bättre förmåga att fånga variansen i data än den univariata gradient boosting-
maskinen.

Artikel IV är en artikel som inte är direkt relaterad till skadeförsäkring och
behandlar istället den generella maskininlärningsmodellfamiljen beslutsträd. I
artikeln presenteras trinary tree-algoritmen, som är ett nytt sätt att hantera
saknade datapunkter för trädbaserade modeller, avsedd att ge en mer regular-
eriserad modell än andra föreslagna metoder. Algoritmen jämförs med nuvarande
metoder för hantering av saknade datapunkter och visar lovande resultat även
för höga andelar av saknade data. Även om algoritmerna kan vara användbara
i skadeförsäkringsmiljöer, diskuteras inte sådana tillämpningar i artikeln.

I Artikel V presenteras en boostad modell med varierande koefficienter, en
så kallad VCM (Varying Coefficient Model). Detta är ett specialfall av en
modell med varierande koefficienter, alltså en modell där koefficienterna i en
linjär modell modelleras med hjälp av någon regressionsmodell. I artikeln
används CGBM-modellen från Artikel III. Detta resulterar i en modell som kan
hantera icke-linjära effekter samtidigt som en viss nivå av lokal tolkningsbarhet
bibehålls. I artikeln visas också hur modellanpassning, val av kovariater och
utvärdering av interaktionseffekter kan förenklas jämfört med tidigare metoder
som inducerar icke-linjära effekter i generaliserade linjära modeller. Modellen
utvärderas på samma datamaterial som i Artikel III och visar lovande resultat
när det gäller noggrannhet och tolkningsbarhet.

Sammanfattningen är baserad på en översättning av DeepL Translator
Bibliography


Part II

Papers