Improving Stability of Tree-Based Models

Morten Blørstad

June, 2023

Supervisors: *Nello Blaser & Berent Lunde*

UNIVERSITETET I BERGEN

Det matematisk-naturvitenskapelige fakultet

Department of Informatics

Abstract

The insurance industry investigates tree-based models to improve their pricing models compared to traditional generalized linear models (GLMs) but has experienced tree-based methods being unstable. This may lead to large changes in predictions when training data is modified slightly and increases the risk of the premium portfolio being too small to cover the losses. This thesis focuses on stabilizing the predictions when updating a model and addresses the instability challenge through three objectives: 1) provide stable update methods for regression trees, 2) extend them to random forest and gradient tree boosting (GTB), and 3) show that methods can also be used for claims frequency estimation. Five stable update methods for regression trees are proposed. The methods are also extended to random forest and GTB. The methods are evaluated in terms of performance and stability and assessed on a broad range of data characteristics before being evaluated on claims data. The general results show that all methods applied to regression trees are more stable than the baseline, with some methods achieving similar or better performance. The methods extended to random forests and GTB show similar results. The general results are transferable to claims frequency estimation. The three-based update methods provide a more stable model than their baselines and better performance than GLM. However, GLM is still more stable than stable tree updates. Whether the stability improvement is sufficient requires further analysis of the impact on the premium portfolio.

Key words: stability, tree-based methods, Bayesian inference, information theory, self-training, claim frequency estimation

Acknowledgements

I would like to thank my supervisors, Nello Blaser and Berent Lunde, for their valuable guidance throughout this thesis. To Nello, I want to express my gratitude for your helpfulness and availability to discuss minor and major topics of my thesis and for your valuable input. I would also like to extend my gratitude to Berent for all the intriguing discussions on the statistical properties of tree-based methods at BKB; Those discussions have expanded my knowledge and understanding significantly, and I am eager to learn more.

To my fellow students, I want to thank you for the time we shared together over the past two years, filled with joy and memorable experiences. A special mention goes to Audun Ljone Henriksen for his valuable feedback on the thesis and the countless hours we spent together during our studies.

Finally, I would like to thank my fiancé, Tilde, for her incredible support throughout a year of hard work. Your understanding and encouragement have allowed me to immerse myself in deep thinking and coding for extended periods, all while helping me maintain a balanced life. Your presence in my life is truly invaluable, and I am eternally grateful.

Morten Blørstad
01 June, 2023

Contents

List of Figures

List of Tables

1 Introduction

The fundamental role of insurance is to provide protection from financial loss by transferring the risk from the insured to the insurer in exchange for a fee known as an insurance premium. The insurer determines the premium before the actual value of the potential loss is known using a pricing model based on statistical/machine learning techniques. Therefore, having a well-performing pricing model is important for achieving the aim of collecting sufficient premiums to cover the claims.

In addition to model predictability, the competitive market is an important aspect of insurance. Every insured should be charged a fair premium based on the insureds risk profile in order to minimize the risk of adverse selection (Dionne et al. [1998]). If the price is too cheap, the insurer will not be able to cover the loss. Overpricing is also problematic as the model assumes that these risks will also contribute to the overall premium when, in reality, they will be lost to competition with more fair premiums. If the pricing model's learning algorithm is unstable, meaning the predictions of the model are very sensitive to changes in the training examples, it increases the probability that the predicted prices will be under- or overpriced, highlighting the importance of stability when applying machine learning algorithms to the insurance industry.

Pricing models are often built in two stages, where the frequency and severity of the claims are considered separately. Traditionally, generalized linear models (GLMs), introduced by Nelder and Wedderburn [1972], are used for non-life insurance pricing models. These models are based on assumptions such as linearity and specific probability distributions (e.g., Gaussian, Gamma, Poisson, etc). GLMs are stable and their results are easy to interpret. If the model assumptions are correct, GLMs provide high predictive capability. However, the data often contains non-linear and interaction components. These non-linear and interaction components need to be included manually through feature engineering when fitting a GLM and are often a tedious analytical task.

Technological advancements have increased the interest in machine learning and provided alternatives to GLMs. Especially, tree-based machine learning methods have shown promising results for predictive modeling in non-life insurance (Guelman [2012];Liu et al. [2014];Henckaerts et al. [2020]) and have been utilized in numerous winning solutions for Kaggle competitions for structured data. The use of neural networks has also been proposed (Ferrario et al. [2018];Schelldorfer and Wüthrich [2019]). Unlike GLMs, tree-based methods and neural networks can model complex non-linear relationships between variables and can automatically capture interaction effects. Due to these properties, they have higher variance and can adapt more to the data and achieve very high performance. However, having a higher variance increases the risk of adapting to noise in the data, potentially making them less stable.

The use of predictive models in insurance or the finance industry, in general, is heavily regulated. In 2018, the European Unions General Data Protection Regulation (GDPR) established the requirement of "algorithmic accountability" of decision-making algorithms. A data subject has a right to contest an individual algorithm decision (Article 22) and to request access to meaningful information about the logic involved (Article 15) (Kaminski and Malgieri [2021]). The requirement of "algorithmic accountability" emphasizes the importance of being able to "look under the hood" of the pricing models, making tree-based models more appealing compared to neural networks.

A pricing model is often updated annually as new claim information from policyholders becomes available. The model is then retrained on previous data plus the newly available data. If the machine learning algorithm used is very sensitive to changes in the training examples, the model update will be unstable, increasing the risk of the premium portfolio being too small to cover the loss. This thesis aims to contribute to solving this issue through three objectives: 1) provide stable update methods for regression trees, 2) show how these methods can be extended to provide stable update methods for random forest and gradient tree boosting, and 3) show that the stable update methods can provide more stable tree-based models for claims frequency estimation. The proposed update strategies are compared to the baseline strategy of retraining the models and are evaluated in terms of performance and stability.

To the best of my knowledge, no other scientific work specifically investigates methods for improving the stability of updating tree-based methods, neither for general regression tasks nor for claim frequency estimation. However, Last et al. [2002] conducted a relevant study for classification tasks, which showed that Info-Fuzzy Networks could be used to learn more compact and, what they call, semantically stable decision trees while preserving a reasonable level of predictive accuracy. Although their work focused on decision trees and classification, they showcase the potential to improve the stability of tree-based methods.

The rest of the thesis proceeds as follows. In Section 2, I will introduce the fundamentals of non-life insurance, key concepts of supervised learning, and tree-based methods. Section 3 defines the notion of update stability and describes the proposed update methods for improving update stability. Section 4 reviews the data used to conduct the experiment, while Section 5 presents the experimental setup and provides the results. Finally, Section 6 discusses the results and concludes the thesis.

2 Background

In this section, I will present the core concepts for the thesis. First, I will introduce the basic concepts of non-life insurance and the stability problem. I then generalize the problem to any supervised learning task and present essential concepts and theories on which the methodology is based on. Examples of how the theory fits the special case of claim frequency estimation will be provided throughout the section.

2.1 Non-Life Insurance Pricing

The fundamental role of insurance is to provide protection from financial loss for the customer (insured) when accidents occur. The insurance company (insurer) takes on the risk of the insured in exchange for a fee known as an *insurance premium*. The risk transfer is formulated as a contract called a non-life insurance policy in which the insurer and insured agree upon what risks are covered and the insurance premium. If an accident occurs that is covered by the policy, the insured can file a request for payment to the insurer according to the terms of the policy. Such a request is called a *claim*. Since the insurer determines the premium before knowing the actual value of the loss, it is essential that the insurer properly assesses the risks in its portfolio. The insurance portfolio consists of many policies, meaning the loss of the entire portfolio is the sum of the loss of the individual policies. Using the law of large numbers, the insurer can estimate the expected loss of individual policies and, thus also the expected loss of the portfolio. Different policies have different exposure to risk and, therefore, different expected loss. Due to fairness and the competitive market, the premium of a policy should reflect the policy's expected loss. To this end, insurance companies use pricing models based on statistical/machine learning methods. It is important that the heterogeneity of risks is properly reflected in the pricing model in order to minimize the risk of adverse selection (Dionne et al. [1998]). If heterogeneity of risks is not properly reflected in the pricing model, the insurer risks that the overall premium is too small to cover the losses. The customers will select the company with the lowest price, and the insurer will therefore attract customers that it under-prices, leading to eroding margins. The over-priced risks are equally problematic as the model is calibrated on the assumption that those risks will also be included in the overall premium but in reality, will be lost to competition with more fair premiums.

In order to price non-life insurance policies, the insurer develops a pricing model that maps the observable characteristic of the insured to the loss cost (also known as risk premium). The insurer then uses the model to predict the loss cost for each insured based on observable characteristics. Commonly, the pricing model is built using two separate models for the frequency and severity of claims,

$$\textit{Risk premium} = \textit{Expected claim frequency} \times \textit{Expected claim severity.}$$

How much the risk premium changes when the insurance data is slightly modified is known as the model's *stability* and will be covered in more detail in Section 2.3.4. If a model is unstable, it indicates that the model fails to clearly distinguish persistence patterns from random patterns (Last et al. [2002]). Meaning the instability will most likely negatively affect the model's ability to reflect the heterogeneity of risks, increasing the risk of adverse selection. In this thesis, I only consider the task of estimating the claim frequency. Claim frequency estimation is a regression problem and falls under supervised learning.

2.2 Supervised Learning

Machine learning consists of understanding and building methods that "learn" from experience. "Learning" is closely related to generalization, meaning the ability to apply knowledge from experiences to new situations. Machine learning is traditionally divided into three broad learning categories, *supervised learning*, *unsupervised learning*, and *reinforcement learning*. This thesis only addresses supervised learning. In supervised learning, the experiences are input-response pairs and the goal is to learn a function that maps inputs to responses. In this section, I will present some core concept in supervised learning which is essential for the thesis.

2.2.1 The Supervised Learning Task

Supervised learning is the task of learning a function f that maps the input vector $\mathbf{x} \in \mathbb{R}^m$ to the corresponding response variable y. When the response variable is categorical it is a classification task, $y \in \mathbb{C}$ where $\mathbb{C}$ is the set of classes. Otherwise, it is a regression task, $y \in \mathbb{R}$. Learning is a search through space of candidate functions $\mathcal{F}$ with the aim of finding the optimal function f^* which minimizes the expected loss given a *loss function* $\mathcal{L}(\cdot, \cdot)$,

$$f^* = \operatorname*{argmin}_{f \in \mathcal{F}} \mathbb{E}[\mathcal{L}(y, f(\mathbf{x}))].$$

The loss function measures the discrepancy between the predicted response $\hat{y} = f(\mathbf{x})$ and true response y. The ultimate goal is to generalize as well as possible, i.e. lowest possible expected loss. Since the expected loss is unknown, an empirical average is used as an estimation. Minimizing the empirical loss is known as empirical risk minimization (ERM) (Vapnik [1991]) and finds an empirical approximation $\hat{f}$ of the optimal function f^*. The expected loss is estimated by using an empirical average over a training dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$,

$$f^* \approx \hat{f} = \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} [\mathcal{L}(y_i, f(\mathbf{x}_i))].$$

In the case of claim frequency estimation, $\mathcal{D}$ is historical claims data where $\mathbf{x}$ is the characteristics of the insured and y is the corresponding claim frequency. The process of minimizing loss to find $\hat{f}$ using a training dataset is called the training phase.

2.2.2 Maximum Likelihood Estimation and Link functions

Before going into more detail about loss functions, I will briefly introduce *maximum likelihood estimation*, *generalized linear models*, and *link functions*. Maximum likelihood estimate (MLE) is a technique used for estimating the most likely parameters of an assumed probability distribution based on observed data. Consider an observed response variable y with the assumed parametric probability distribution $P(y, \theta)$, where the θ is in the parameter space Θ. The MLE of θ is the value that maximizes the likelihood or, equivalently, the value that minimizes the negative log-likelihood,

$$\hat{\theta} = \operatorname*{argmin}_{\theta} - \log P(y, \theta). \tag{1}$$

Note that the objective of supervised learning is closely connected to MLE when the function search is restricted to a parametric family of functions and the loss function is defined as the negative log-likelihood, $\mathcal{L} = -\log P$.

The generalized linear model (GLM) (Nelder and Wedderburn [1972]) is an example of a parametric family of functions. It is a generalization of the linear regression model, which allows the model to handle non-normal response variables, e.g., count data, by specifying a probability distribution that describes the behavior of the response variable. GLMs also allow for the linear model to be related to the response variable via a link function.

The purpose of a link function in GLMs is to "link" the linear relations between the input variables $\mathbf{x}$ to the mean, μ, of a distribution. The link function also ensures that the predictions are within the correct range and have the appropriate distributional properties. The link function $\Phi(\cdot)$ must be monotonic and differentiable such that $\mu = \Phi^{-1}(f(\mathbf{x}))$. In short, link functions are used to connect the input to the expected mean of the response variable in a linear way, and its inverse is used to make predictions in the original form. Link functions are also used in generalized additive models (GAMs) (Hastie and Tibshirani [1986]) and gradient tree boosting (Friedman [2001]). In gradient tree boosting, the link function maps the predicted value from the trees to a scale that is appropriate for the specific target variable and loss function.

2.2.3 The Loss Function

Section 2.2.1 introduced the idea of finding the best predictive function $\hat{f}$ that minimizes a loss function in the training phase. The choice of loss function for a particular problem should reflect some discrepancy between the true response variable and the predictions that one wants to minimize. Loss functions are often likelihood-based and reflect the assumptions of the data-generating process. They can be interpreted as negative log-likelihoods. The most common loss function for regression is the *squared error* loss,

$$\mathcal{L}(y_i, \hat{f}(\mathbf{x}_i)) = \sum_{i=1}^{n} (y_i - \hat{f}(\mathbf{x}_i))^2, \tag{2}$$

The squared error assumes that the response variable conditioned on the input has a normal distribution, $y|\mathbf{x} \sim \mathcal{N}(\mu, \sigma)$. The likelihood-based loss function is the negative log-likelihood of the normal distribution

$$\mathcal{L}(y_i, \mu) = \sum_{i=1}^{n} \frac{1}{2} \log 2\pi\sigma^2 + \frac{1}{2\sigma^2} (y_i - \mu)^2 . \tag{3}$$

Here the link function for normal distribution is used, which is the identity link. Meaning, the function is estimated on the original parameter scale $f(\mathbf{x}) = \Phi(\mu) = \mu$. Assuming a constant σ^2 that is of no interest of estimating, the expression in Equation 3 can be simplified to be equivalent to the squared error loss. As the squared error loss assumes a conditional normal distribution, it is suitable for continuous distributions which are symmetrical around their means and have a constant variance, i.e., elliptical distributions (Henckaerts et al. [2020]). However, claim frequency data does not have an elliptical distribution, thus another loss function is required. Claim frequency is typically assumed to be Poisson distributed, $y|\mathbf{x} \sim \text{Pois}(\lambda)$. Using this assumption, a likelihood-based loss can be constructed using the negative log-likelihood of the Poisson distribution

$$\mathcal{L}(y_i, \lambda) = \sum_{i=1}^{n} \lambda - y_i \log(\lambda) + \log(y_i!). \tag{4}$$

Here λ is the mean of the Poisson distribution. Using the link function for Poisson distribution, which is the log link, the function can be estimated on the log scale, $\hat{f}(\mathbf{x}) = \Phi(\lambda) = \log(\lambda)$. The terms not containing λ can be removed and λ can be rewritten in terms of $\hat{f}(\mathbf{x})$, resulting in the following expression

$$\mathcal{L}(y_i, \hat{f}(\mathbf{x}_i)) = \sum_{i=1}^{n} e^{\hat{f}(\mathbf{x}_i)} - y_i \hat{f}(\mathbf{x}_i). \tag{5}$$

Hereafter the loss in Equation 5 is referred to as the Poisson loss.

2.2.4 Claim Frequency Estimation and Offset Variable

Claims frequency estimation is based on count data and assumes the number of claims is Poisson distributed. Let N_i be the annualized claim frequency for insured i where $N_i \sim Pois(\lambda_i)$ and λ_i is a function of the insured's characteristics $(\mathbf{x}_i)$. Let E_i (exposure) be the time period the insured i was observed and Y_i be the number of claims during the time period. Insurance companies are interested in the annualized claim frequency N_i. Unfortunately, N_i cannot be observable, but the number of claims Y_i over the period E_i can. $N_i \sim Pois(\lambda_i)$ can equivalently be written as $Y_i \sim Pois(\lambda_i \cdot E_i)$. By using a log link function, $\lambda_i = e^{f(\mathbf{x}_i)}$ one gets,

$$Y_i \sim Pois(e^{f(\mathbf{x}_i)+\log E_i}),$$

where $\log E_i$ is an offset variable (Charpentier [2014]). This ensures that the expected number of claims is directly proportional to the exposure. With less mathematical notation, the goal of claim frequency estimation is to provide an accurate prediction of the number of claims that a policyholder is likely to report during a 1-year period based on their exposure and relevant risk factors. The inclusion of the offset variable ensures that the model accounts for the impact of exposure on the frequency of claims.

2.2.5 The Model and Learning Algorithm

The function space $\mathcal{F}$ is also known as the *hypothesis space* and refers to the set of possible functions/models that a learning algorithm can learn. It represents the set of functions the algorithm can choose from to model the data. For instance, the hypothesis space for linear regression is all the possible linear functions that can be used to model the data. Recall from 2.2.1 that the aim is to find $\hat{f}$ by minimizing the empirical average over a training dataset which is essentially an optimization problem. The learning algorithm determines how the optimization problem is solved. The learning algorithm uses the training data $\mathcal{D}$ as input and outputs a fitted model $\hat{f}$. Different choices of loss functions and learning algorithms will lead to different solutions. As implied by the No Free Lunch theorem (Wolpert and Macready [1997]), no single learning algorithm is universally superior for all types of tasks or dataset. The theorem highlights the importance of understanding the characteristics of the data, and how they may affect the performance of different machine learning algorithms when building a predictive model.

2.2.6 Numerical Optimization in Function Space

With the exception of the simplest cases, the maximum likelihood objective and supervised learning objective are not straightforward and require numerical methods to be solved. The numerical methods are often iterative and involve updating the parameters in each iteration until a convergence criterion is met. Local quadratic approximations are a common iterative optimization method that approximates a function locally around the current point of interest using a quadratic function, which is much simpler to optimize than the original function. The term "local quadratic approximation" is also known as the second-order Taylor approximation and is used to approximate a function $f(x)$ around a specific point x_0. For example, the second-order Taylor approximation of the loss function $\mathcal{L}$ around a value of θ_k can be expressed as

$$\begin{aligned}\mathcal{L}(y_i, f(\mathbf{x}_i;\theta)) \approx \mathcal{L}(y_i, f(\mathbf{x}_i;\theta_k)) &+ \nabla_\theta \mathcal{L}(y_i, f(\mathbf{x}_i;\theta_k))(\theta-\theta_k) \\ &+ \frac{1}{2}(\theta-\theta_k)^T \nabla_\theta^2 \mathcal{L}(y_i, f(\mathbf{x}_i;\theta_k))(\theta-\theta_k).\end{aligned}$$

The right-hand side of the expression above can be used to construct the update rule for the parameters in the Newton-Raphson method,

$$\theta_{k+1} = \theta_k - \frac{\nabla_\theta \mathcal{L}(\theta_k)}{\nabla_\theta^2 \mathcal{L}(\theta_k)}.$$

Example - Generalized linear models Consider a generalized linear model parameterized by θ

$$f(\mathbf{x}_i;\theta) = \theta_0 + \sum_{j=1}^{m} \theta_j \mathbf{x}_{i,j}.$$

One want to estimate a parameter vector $\theta = (\theta_0, \dots, \theta_m)$ that minimizes a negative log-likelihood loss from an exponential family

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \ell(y_i, f(\mathbf{x}_i;\theta)),$$

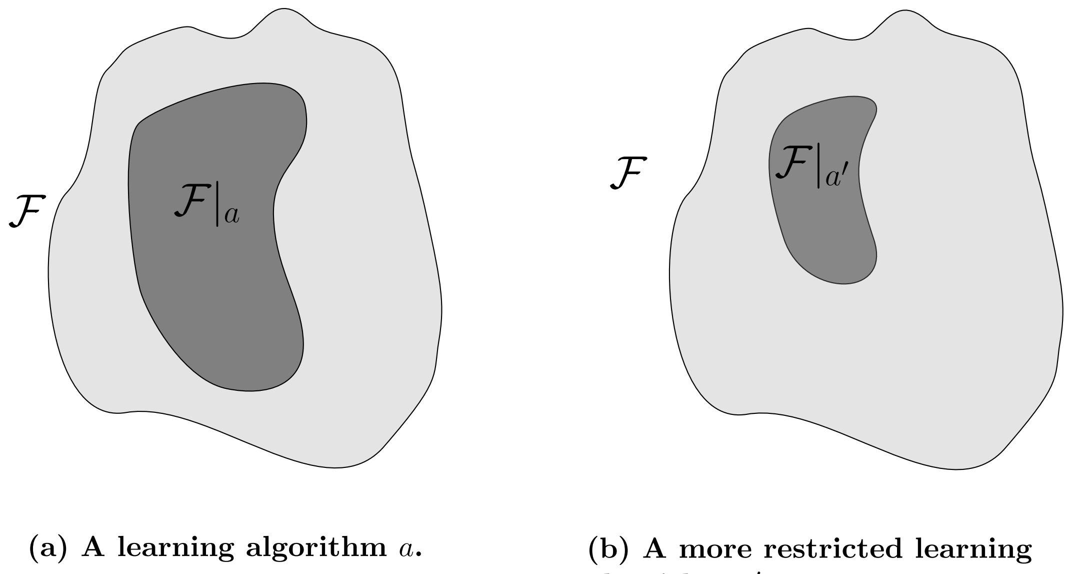
where ℓ is the negative log-probability of $y|\mathbf{x}$. The MLE $\hat{\theta}$ in Equation 1 often requires numerical methods to be found, since the score equation, $\nabla_\theta \mathcal{L}(\theta) = 0$, cannot be solved analytically. The iteratively reweighted least squares method is typically used to optimize the parameters of GLMs and is essentially a Newton-Raphson method.

2.3 Model Selection

Since there is no single best learning algorithm and different learning algorithms lead to different solutions, it is common to train several models using different learning algorithms. These models are called *candidate models*. The process of choosing the best model from a set of candidate models for a given dataset is known as model selection. In this subsection, I discuss the different aspects of model selection.

2.3.1 The Model Capacity

The goal is to learn a model that generalizes well, i.e., performs well on new unseen data and not just the data the model is trained on. How well a learning algorithm performs is determined by two factors, its ability to make the training loss small and its ability to make the difference between training loss and expected loss small (Goodfellow et al. [2016]). These factors correspond to the two central challenges in machine learning, namely *underfitting* and *overfitting*. Underfitting refers to the model's inability to obtain a sufficiently low training loss, whereas overfitting refers to the model's inability to make the difference between training loss and expected loss sufficiently small. Overfitting and underfitting are highly impacted by a learning algorithm's *capacity*. The capacity refers to the learning algorithm's ability to represent complex functions (i.e. the size of the function space $\mathcal{F}$) and is often also referred to as model complexity. Learning algorithms with low capacity may not be able to fit the training data (i.e. underfit). Learning algorithms with high capacity may overfit the training data by learning the noise in the training data, resulting in poor generalization. Different learning algorithms have different capacities and their capacity can also be adjusted through *hyperparameters*. Hyperparameters are parameters that control the learning process of learning algorithms. Different learning algorithms and hyperparameters can be seen as different restrictions or preferences in the function space as illustrated by Figure 1.



(a) A learning algorithm a.

(b) A more restricted learning algorithm a'.

Figure 1: The functions space covered by a learning algorithm. The figure shows the function space $\mathcal{F}$ and the function space covered by a learning algorithm a, $\mathcal{F}|_a$. $\mathcal{F}|_a$ is the model capacity of a. a' represents a more restricted learning algorithm either due to regularization, choice of hyperparameters, or use of different learning algorithms with less variance. The learning algorithm a, **(a)**, has a larger search space (model capacity) compared to the restricted learning algorithm a', **(b)**.

2.3.2 The Bias-Variance Trade-off

The capacity of a model and its ability to generalize can be described with *the bias-variance trade-off*. The bias-variance trade-off describes the trade-off between the model's ability to fit the training data and its generalization performance. In order to make a prediction on new unseen data based on training data, the learning algorithm needs to use some assumptions (or prior knowledge) of the underlying data structure. These assumptions or prior knowledge are referred to as *inductive* *bias*. For example, linear regression assumes that there is a linear relationship between the input and the response variable. Bias is the error due to false model assumptions, whereas variance is the error due to sensitivity to small fluctuations in the training dataset. Bias and variance are connected to under- and overfitting. Underfitting is associated with high bias and low variance, whilst overfitting is associated with low bias and high variance. One can reduce a models variance by increasing its bias and vice versa, hence the name bias-variance trade-off. The aim is to strike a balance between bias and variance in order to minimize the total error as illustrated by Figure 2.

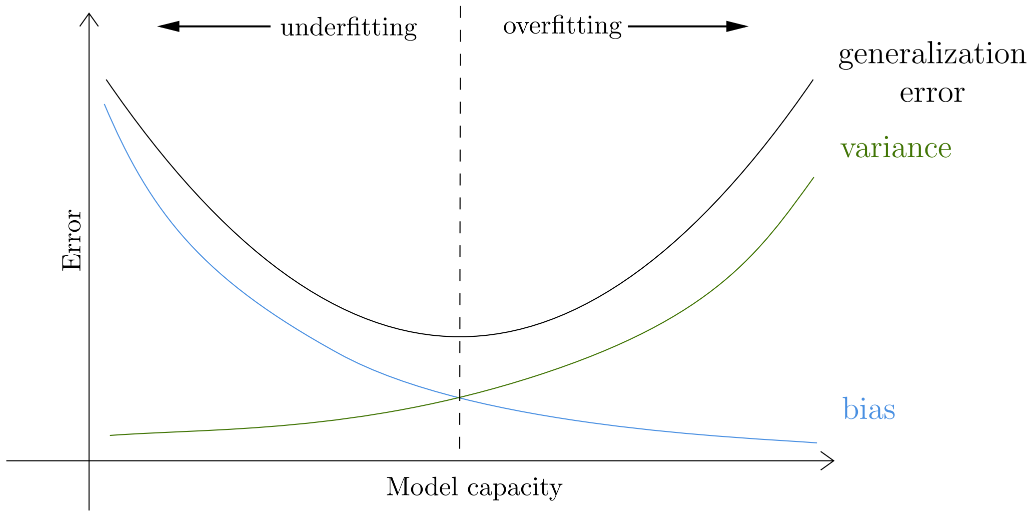


Figure 2: The bias-variance trade-off
The figure illustrates the bias-variance trade-off and how bias and variance vary in opposite ways with respect to model capacity. As model capacity increases, bias decreases while variance increases and vice versa. The generalization error is the sum of bias squared and variance. The optimal learning algorithm minimizes the generalization error, highlighted by the dotted line, avoiding underfitting and overfitting.

The trade-off can also be thought of in the sense of function space restriction as Figure 3 shows. Bias can be represented as the distance from the optimal function f^* to the optimal function $f^*|a$ in restricted function space $\mathcal{F}|_a$. Variance is the distance from $f^*|a$ to the function found by minimizing training loss $\hat{f}|a$.

2.3.3 Regularization

Regularization is a technique to deal with the high-variance problem (i.e. overfitting) by explicitly or implicitly limiting the learning process. Both explicit and implicit regularization techniques are used to control the flexibility of the learning process in order to improve the generalization performance of a model.

Explicit regularization refers to the use of a penalty term added to the loss function to encourage simpler solutions. The penalty term is a measure of the model's complexity. As there exists no universal complexity measure, the choice of complexity measure has to come from a priori knowledge about the task at hand (Bousquet et al. [2003]). For example, in linear regression, the $L1$ or $L2$ norm of the coefficients are common complexity measures, whereas in gradient tree boosting the $L1$ or $L2$ of the leaf values are used. When the $L1$ or $L2$ norm of the parameters are used as the penalty term, they are known as $L1$ and $L2$ regularization, respectively. A penalized loss can be generalized as

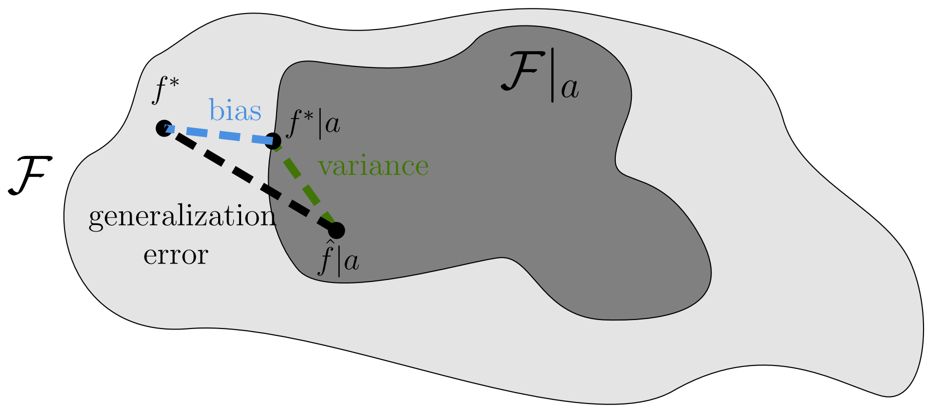


Figure 3: The bias-variance trade-off as a function space restriction. The figure shows how the bias-variance trade-off can be conceptualized as a restriction of function space. Here, bias is the distance from the optimal function f^* to the optimal function $f^*|a$ in the restricted function space $\mathcal{F}|_a$. If f^* is in $\mathcal{F}|_a$ then the bias is zero. Variance is the distance from $f^*|a$ to the function found by minimizing training loss $\hat{f}|a$. Bias and variance make up the generalization error. By increasing the restriction of the function space, bias increases while variance decreases and vice versa.

$$\mathcal{L}(\cdot,\cdot) + \gamma\Omega, \tag{6}$$

where Ω is the complexity measure and γ is a hyperparameter determining the strength of complexity penalty. By penalizing complexity one insert a bias toward the simpler models, i.e. trading variance for bias.

Implicitly regularization refers to regularization techniques that do not explicitly use a regularization penalty term. Certain learning algorithms inherently handle high variance by design such as bagging algorithms (Hastie et al. [2001]) whilst other algorithms, such as stochastic gradient descent, inherently promote simpler models that generalize well (Smith et al. [2021]). Other examples of implicit regularization are early stopping, data augmentation, dropout, etc.

2.3.4 Stability

Stability is a concept in computational learning theory and refers to the ability of a machine learning algorithm to produce consistent and reliable predictions on new, unseen data. It is connected to generalization and the bias-variance trade-off and has an inverse relation to variance (Bousquet and Elisseeff [2002]). Stability aims to determine how the variance of a learning algorithm affects its generalization error with respect to changes in the training set (Bousquet and Elisseeff [2002]). Different notions of stability have been used to set a bound on the generalization error of certain learning algorithms such as Hypothesis Stability, Error Stability, and Uniform Stability (Devroye and Wagner [1979]; Kearns and Ron [1999]; Bousquet and Elisseeff [2002]). Uniform stability is the strongest notion since it implies both Hypothesis Stability and Error Stability (Bousquet and Elisseeff [2002]).

Figure 4 aims to illustrate how stability is related to the bias-variance trade-off. The aim is to find the supremum of the training loss fluctuation, β, which is the maximum difference between the original training loss and the loss after removing any one data point. Since β is a bound on variance, it is also a bound on the generalization error since the generalization error is the sum of bias squared and variance.

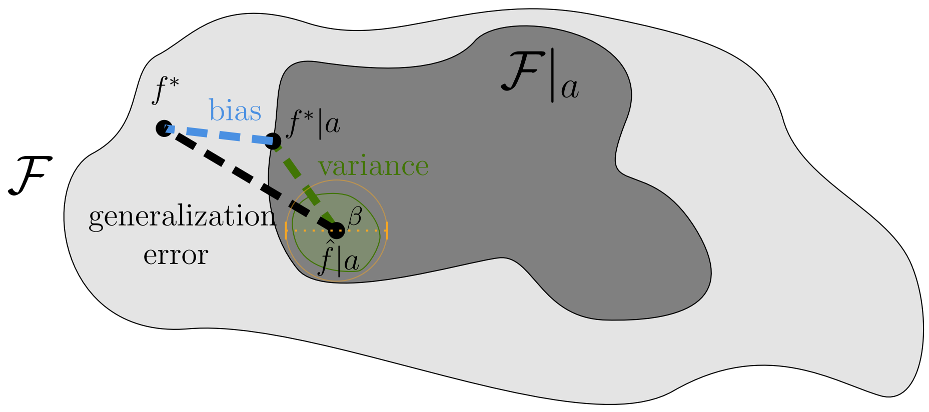


Figure 4: The relation between stability and the bias-variance trade-off The figure shows the relationship between stability and the bias-variance trade-off and how it can be used as a bound on generalization error. The green area demonstrates how the function $f^*|a$, obtained by minimizing the training loss, can vary when a data point is removed from the training data. The orange circle represents the bound on this variation. The circle's diameter, β, is the maximum change of $f^*|a$ caused by the removal of a single data point i.e., the impact of one data point on the loss is less than or equal to β. Note the circular bound is used to simplify the illustration.

Stability in the traditional sense refers to how much predictions change due to the removal or replacement of a data point and aims to put a bound on the generalization error. There is another notion of stability different from the traditional notions called *semantic stability* (Turney [1995]). Semantic stability is specific to classification algorithms and refers to how often two models fitted on random samples from the same data-generating process assign examples to the same class. It is a performance measure to determine how stable a classification algorithm is.

2.3.5 Model Selection and Evaluation

In model selection, several candidate models are trained. The aim is to select one of the candidate models that performs/generalizes best. Traditionally, model selection and evaluation use the *holdout method* where the data is divided into three parts, training, validation, and test sets. The training set is used to train the models, whereas validation and test set are used for model selection and evaluation. A single model can be evaluated by comparing performance on the training and validation dataset. High model performance on the training but not on the validation dataset indicates high variance (i.e., overfitting). Low performance on both the training and validation dataset indicates high bias (i.e., underfitting). Multiple models can be evaluated by comparing the models' performance on the validation dataset. The model with the best performance on the validation dataset is considered the best. The performance estimate from the validation set is biased in the sense that is used for model selection. The test set is data the selected model has not used nor seen and represents new unseen data. Thus, the test data can be used to get an unbiased estimate of the selected model's ability to generalize to unseen data.

A disadvantage of the holdout method is it may result in a performance estimate with high variance. *Cross-Validation* is an alternative method that provides a performance estimate with less variance. It involves dividing the data into training and test set rather than training, validation, and test set. The training data is divided into k-folds. The models are trained on the $k-1$ folds and evaluated on the remaining fold. This process is repeated k times, with each fold being used as the validation set ones. The performance estimates from each of the folds are averaged to get the final validation score which is used for model selection. The test set is used to obtain the unbiased performance estimate of the selected model.

2.4 Model Updating

Over time as more data is obtained, the learned model may need to be updated. Either due to concept drift (i.e., changes in the data-generating process over time) or simply the fact that more training data (more information) is expected to reduce the gap between training loss and the true expected loss, leading to a model that generalizes better. *Model updating* is the process of adjusting a model to new information. Traditionally models are learned using *batch learning*, which means every time new data becomes available, the entire model has to be retrained from scratch using all data. New information cannot be added to the model without retraining the model. Batch training can be memory expensive for large data amounts as it uses all available data when training. Batch learning is suitable for tasks where the model updating is infrequent and memory and computation resources are not an issue. *Incremental learning*, in contrast to batch training, trains a model and updates the model incrementally as new data becomes available instead of retraining from scratch. Incremental learning can adjust knowledge learned from previous data and add knowledge obtained from the new data. It can also adapt to concept drift. Incremental learning is less memory expensive since it only works with smaller chunks of data. The choice between using batch or incremental learning is determined by the task at hand, i.e., update frequency, memory, and computation expense (Schlimmer and Fisher [1986];Utgoff [1989]).

2.4.1 Bayesian Updating

Updating a model can be viewed from the Bayesian perspective. Using Bayesian inference for model updating allows the model to learn from new data and update its belief in its parameters. Here the model's parameters are treated as random variables and updated using Bayes' theorem (Bayes [1763]),

$$P(\theta|\mathcal{D}) = \frac{P(\mathcal{D}|\theta)P(\theta)}{P(\mathcal{D})}.$$

The prior distribution $P(\theta)$ represents the model's prior beliefs about the parameters before observing any new data. The likelihood $P(\mathcal{D}|\theta)$ represents how well the model's parameters explain the data. The product of the prior and the likelihood normalized by the probability of observing the data $P(D)$ results in a posterior distribution $P(\theta|\mathcal{D})$. The posterior represents the updated belief or likelihood in the parameters given the observed data. In model updating, only the relative likelihood of the parameters is needed and $P(D)$ can be excluded resulting in the simpler expression,

$$\underbrace{P(\theta|\mathcal{D})}_{posterior} \propto \underbrace{P(\mathcal{D}|\theta)}_{likelihood} \times \underbrace{P(\theta)}_{prior}. \tag{7}$$

The posterior becomes the new prior that again can be used to update the belief in the parameters when even more data becomes available. In a Bayesian setting, the *maximum a posteriori* (MAP) is used to estimate the most probable parameter values by maximizing the combined likelihood of data and the prior distribution. MAP is similar to MLE but unlike MLE, MAP also incorporates prior knowledge into the estimation process.

Regularization can also be viewed from the Bayesian perspective. If the loss function is likelihood-based, then $L1$ regularization imposes a Laplacian prior to the model's parameter, whereas $L2$ regularization imposes a Gaussian prior. Thus, regularization has a Bayesian interpretation, where the Ω in Equation 6 is the prior belief that the model should be simple, and γ is the strength of the belief.

Example - Model update of GLMs Consider a GLM model, $\hat{f}$, already trained on dataset $\mathcal{D}^{(t)}$ with the parameter vector $\hat{\theta}^{(t)}$. Let $\mathcal{D}^{(t+1)}$ be a new dataset one wants to include in $\hat{f}$. Using Bayes theorem with negative log-likelihoods, the model's updated parameters, $\hat{\theta}^{(t+1)}$, can be found by minimizing negative log posterior,

$$\underbrace{-\log P\left(\hat{\theta}^{(t+1)}|\mathcal{D}^{(t)}, \mathcal{D}^{(t+1)}\right)}_{posterior} \propto \left[\underbrace{-\log P\left(\mathcal{D}^{(t+1)}|\hat{\theta}^{(t+1)}, \mathcal{D}^{(t)}\right)}_{likelihood}\right] + \left[\underbrace{-\log P\left(\hat{\theta}^{(t)}|\mathcal{D}^{(t)}\right)}_{prior}\right].$$

Assuming a Gaussian prior results in the following update loss,

$$\begin{aligned}\mathcal{L}^{t+1}\left(y, \hat{f}\left(\mathbf{x};\theta^{(t)}\right), \hat{f}\left(\mathbf{x};\theta^{(t+1)}\right)\right) \propto\, & \mathcal{L}\left(y, \hat{f}\left(\mathbf{x};\theta^{(t+1)}\right)\right) \\ & + \gamma\left(\hat{f}\left(\mathbf{x};\theta^{(t)}\right) - \hat{f}\left(\mathbf{x};\theta^{(t+1)}\right)\right)^2,\end{aligned}$$

which imposes a belief (bias) that the updated model should be similar to the current model.

2.4.2 Bootstrap

Bootstrap is a statistical resampling method first introduced by Efron [1979]. The basic idea of bootstrapping is to estimate the characteristics of a population using only the information from a sample. Rather than making assumptions about the underlying population distribution, the sample is treated as the population. This allows for new samples to be generated from the empirical distribution by resampling with replacement from the original sample. These resamples can then be used to make inferences about the population. This makes bootstrap particularly useful in situations where the population distribution is unknown or difficult to model.

Bootstrap is widely used in machine learning as one can treat the training set as the data-generating process and resample with replacement from the original training set $\mathcal{D}$ to generate B new bootstrap training sets, $\{\mathcal{D}_b\}_{b=1,\ldots,B}$. This is for example used in model averaging, which I will come back to in section 2.5.2 when introducing bagging and random forest. Bootstrapping plays an important part in this thesis as two of the proposed methods rely on it.

2.4.3 Semi-Supervised Learning and Self-Training

Semi-supervised learning is a learning process that combines supervised learning and unsupervised learning (van Engelen and Hoos [2020]). Supervised learning is used to train a predictive model using labeled data, whereas unsupervised learning is used to uncover patterns and structures in unlabeled data. Semi-supervised learning algorithms use a combination of labeled and unlabeled data to improve model performance. *Self-training* and *co-training* are examples of semi-supervised learning algorithms (Amini et al. [2022]). With an initial model trained on labeled training data, self-training iteratively improves the model by generating pseudo-labels to unlabeled training samples, which are used to enrich the labeled training data. The model is then retrained/updated using the enriched training data. Co-training uses multiple models trained on different subsets of the data to improve performance. The models predict the labels of the unlabeled data as the majority (classification) or average (regression), and the predictions are used as pseudo-labels. The pseudo-labeled data is added to the training data and the models are retrained/updated. The predictive model at a current iteration is learned by minimizing the empirical loss,

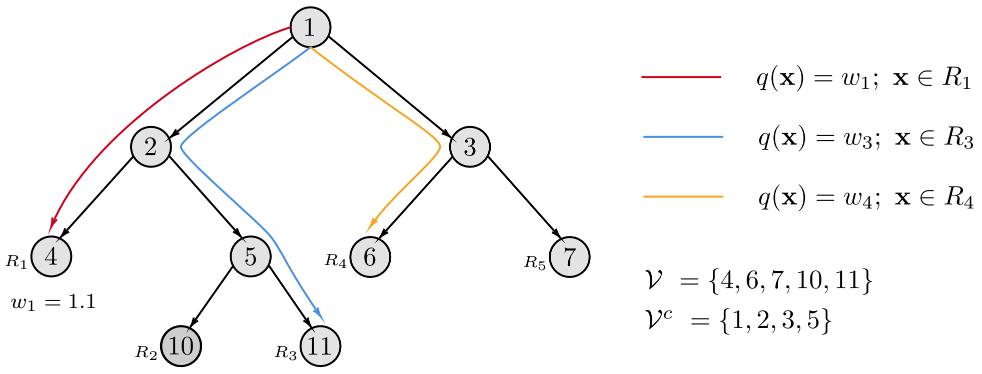
$$\frac{1}{n}\sum_{(\mathbf{x},y)\in\mathcal{D}} \mathcal{L}(f(\mathbf{x}), y) + \frac{\gamma}{\tilde{n}}\sum_{(\mathbf{x},\tilde{y})\in\tilde{\mathcal{D}}} \mathcal{L}(f(\mathbf{x}), \tilde{y}),$$

where $\tilde{\mathcal{D}}_{i=1}^{\tilde{n}} = \{\mathbf{x}, \tilde{y}\}$ is the unlabeled data that have been pseudo-labeled, and γ is hyperparameter controlling the impact of the pseudo-labeled data on the learning.

It is worth noting that using unlabeled data does not always improve model performance, and there is no guarantee that introducing unlabeled data will not reduce performance (van Engelen and Hoos [2020]). For example, Chapelle et al. [2006] compared eleven semi-supervised learning algorithms using supervised support vector machines and k-nearest neighbors as baselines. They found that no algorithm consistently outperformed the others, with varying performance across datasets. Some algorithms substantially improved performance compared to the baselines for some datasets while reducing performance on others. Therefore, semi-supervised learning should be viewed as one of the possible learning algorithms to consider when searching for the best model for a specific task.

2.5 Tree-Based Methods

In this section, I introduce the necessary details for understanding the tree-based models used in this thesis, which is mainly based on the book by Hastie et al. [2001]. I start by describing the Regression Tree (Breiman et al. [1984]), followed by its extensions, Random Forest (Breiman [2001]) and Gradient Tree Boosting (Friedman [2001]). Section 3 describes the proposed updating methods for each of the tree-based models.



1
2
3
R_1 4
5
R_4 6
R_5 7
$w_1 = 1.1$
R_2 10
R_3 11
$q(\mathbf{x}) = w_1;\ \mathbf{x} \in R_1$
$q(\mathbf{x}) = w_3;\ \mathbf{x} \in R_3$
$q(\mathbf{x}) = w_4;\ \mathbf{x} \in R_4$
$\mathcal{V} = \{4, 6, 7, 10, 11\}$
$\mathcal{V}^c = \{1, 2, 3, 5\}$

Figure 5: The Classification and Regression Tree
An example of a Classification and Regression Tree (CART) with five leaf nodes ($\mathcal{V}$) and four internal nodes ($\mathcal{V}^c$). The vector $\mathbf{w} = (w_1, w_2, w_3, w_4, w_5)$ is the possible predictions the tree can make. The feature mapping $q(\mathbf{x})$ is a function that maps inputs to their corresponding region R.

2.5.1 Regression Tree

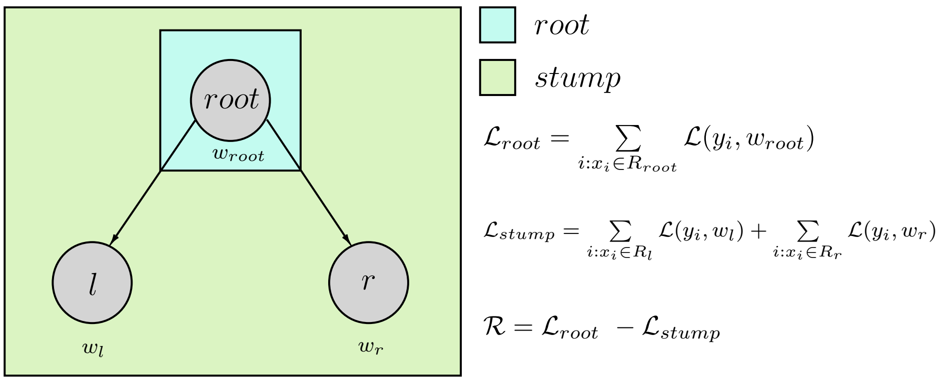
Decision or regression trees partition the feature space into a set of rectangles based on yes-no questions and fit a simple model (like a constant or linear model) in each one (Hastie et al. [2001]). A widely used approach to building tree models is the Classification And Regression Tree (CART) algorithm by Breiman et al. [1984] (see Figure 5). Let $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$ be a dataset of size n that consists of a response, y_i, and m features, $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,m})$, for each observation i. The CART algorithm partitions the feature space $R = \mathbb{R}^m$ into T non-overlapping regions, $R_1, \ldots, R_t, \ldots, R_T$ (see Figure 7a). In each region, the fitted response, w_t, is the average of the response observations within that region. The model maps input $\mathbf{x}$ to y as w_t of the region R_t the input belongs to

$$\hat{f}(\mathbf{x}) = \sum_{t=1}^{T} w_t I(\mathbf{x} \in R_t).$$

The CART algorithm uses a top-down greedy approach as it is computationally infeasible to consider every possible partition of the feature space R. Starting with a single *root* node containing all the data, the algorithm considers a splitting variable $j \in \{1, \ldots, m\}$ and split value s that creates the half-planes $R_l(j,s) = \{x|x_j \leq s\}$ and $R_r(j,s) = \{x|x_j > s\}$. The algorithm seeks the splitting variable j and split value s that maximizes the loss reduction $\mathcal{R}$,

$$\mathcal{R}(j,s) = \underbrace{\sum_{i:x_i \in R_{root}} \mathcal{L}(y_i, \hat{w}_{root})}_{\mathcal{L}_{root}} - \underbrace{\left[\sum_{i:x_i \in R_l(j,s)} \mathcal{L}(y_i, \hat{w}_l) + \sum_{i:x_i \in R_r(j,s)} \mathcal{L}(y_i, \hat{w}_r) \right]}_{\mathcal{L}_{stump}}. \quad (8)$$

Here *stump* refers to the *root* node and its two resulting regions, R_l and R_r as illustrated by Figure 6.



$$\mathcal{L}_{root} = \sum_{i:x_i \in R_{root}} \mathcal{L}(y_i, w_{root})$$

$$\mathcal{L}_{stump} = \sum_{i:x_i \in R_l} \mathcal{L}(y_i, w_l) + \sum_{i:x_i \in R_r} \mathcal{L}(y_i, w_r)$$

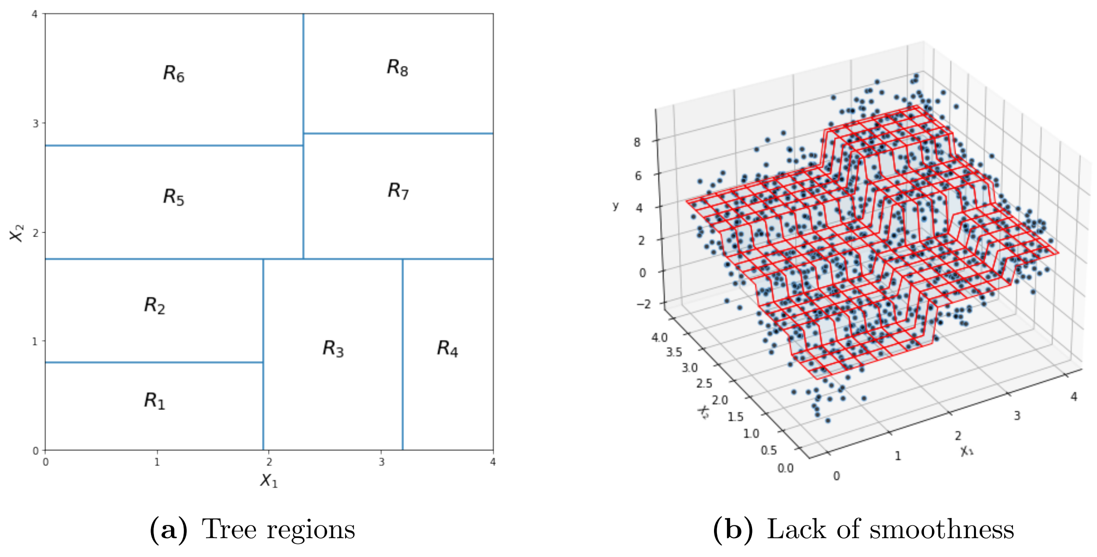
$$\mathcal{R} = \mathcal{L}_{root} - \mathcal{L}_{stump}$$

Figure 6: The root and stump
The figure shows how the loss reduction $\mathcal{R}$ of a split is calculated. $\mathcal{L}_{root}$ is the loss before the split and $\mathcal{L}_{stump}$ is the loss after splitting the data by a splitting variable $j \in \{1, \dots, m\}$ and a split value s. The loss reduction is the difference in the loss before and after splitting, $\mathcal{R} = \mathcal{L}_{root} - \mathcal{L}_{stump}$. After finding the split, the splitting procedure repeats on the resulting nodes l and r, where l and r become "*roots*". The splitting procedure stops when a stopping criterion is reached.

After finding the partition that results in the largest loss reduction, the CART algorithm recursively repeats the splitting process on the resulting regions until a stopping criterion (or multiple) is satisfied (e.g., a maximum tree depth (*max depth*) or a minimum number of observations in a leaf node (*min sample leaf*)).

2.5.1.1 Stability of Trees

Trees are known to have high variance. Due to the high variance, small changes in data can often lead to very different splits which again can lead to inconsistent predictions on new unseen data, i.e., instability. Trees are unstable because changes in the top split affect all splits below it due to the hierarchical nature of the learning process, i.e., greedy splitting. The use of a more stable split criterion can alleviate this instability to some extent, though it cannot entirely eliminate it (Hastie et al. [2001]). *Evolutionary trees* can also be used to address the hierarchical instability as it finds the tree structure based on a global criterion rather than a local criterion (Jankowski and Jackowski [2014]). Another limitation of regression trees is the lack of smoothness of the predictions, which limits their performance where the underlying function is expected to be smooth, as illustrated by Figure 7b (Hastie et al. [2001]). One way of dealing with the high variance problem is to use regularization and/or ensemble methods such as *random forest* and *gradient tree boosting*. The ensemble methods also address the lack of smoothness problem.



(a) Tree regions **(b)** Lack of smoothness

Figure 7: Tree regions and prediction surface

The figure shows **(a)** how a tree splits the feature space into non-overlapping regions, and **(b)** the prediction surface of the same tree. The step-like function illustrates the lack of smoothness of regression trees. The tree is trained on simulated data with two features, $\mathbf{x}_1, \mathbf{x}_2 \sim \mathcal{U}(0, 4)$ and $y \sim \mathcal{N}(\mathbf{x}_1 + \mathbf{x}_2, 1)$.

2.5.2 Random Forest

Bagging or *bootstrap aggregation* (Breiman [1996]) is an ensemble technique that reduces the variance of an estimated model by training multiple models on bootstrapped samples of the training data to create an ensemble of models and then aggregating their predictions. The variance reduction of this model averaging stabilizes the predictions and improves the predictive performance compared to using a single model. Bagging works well for high-variance, low-bias models like trees. More formalized, bagging involves training B trees independently by taking bootstrap samples $\{\mathcal{D}_b\}_{b=1,\ldots,B}$ from a training dataset $\mathcal{D}$. After B trees are trained, the bagging predictions are given by

$$f_{\text{bagg}}(\mathbf{x}) = \frac{1}{B} \sum_{b=1}^{B} f^b(\mathbf{x}|\mathcal{D}_b),$$

where the conditional $(\cdot|\mathcal{D}_b)$ means that the tree was trained on the bootstrap sample $\mathcal{D}_b$. In bagging, the trees generated are identically distributed, meaning that the expected value of the average of the trees is equal to the expectation of any of the individual trees. Any performance improvement can only be achieved through variance reduction (Hastie et al. [2001]). The variance of bagging is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2,$$

where ρ is the positive pairwise correlation between the bagged trees. As B increases, the second term disappears, but not the first. Therefore, the benefit of averaging is limited by the correlation between bagged trees. The *random forest* algorithm, developed by Breiman [2001], improves upon bagging by introducing additional randomization, which reduces the correlation between trees and further enhances variance reduction. This involves randomly selecting a subset of features for each split in the tree. Putting it all together results in Algorithm 1.

Algorithm 1: Random Forest (Hastie et al. [2001])

```
Input  : Training data $\mathcal{D} = \{\mathbf{X}_i, y_i\}_{i=1}^n$
         Number of trees $B$
Output: The model, $f_{\text{rf}}^B$
1 for $b = 1$ to $B$ do
      (a) Draw a bootstrap sample $\mathcal{D}_b$ of size $n$ from training data.
      (b) Fit a random forest tree $T_b$ to the bootstrapped data, by recursively repeating steps
      for each terminal node of the tree until the stop criterion is reached.
          i. Select $m'$ variables at random from $m$ variables.
          ii. Pick the best variables/split-point among the $m'$.
          iii. Split the node into two child nodes.
2 Output the ensemble model return $f_{\text{rf}}^B$
```

The number of variables to randomly sample as candidates at each split, m', is a hyperparameter and is in many implementations called *mtry*. Hastie et al. [2001] recommends *mtry* $= \lfloor m/3 \rfloor$ and *min sample leaf* $= 5$ as default hyperparameters for random forests. The number of trees in the forest is also a hyperparameter and is often set to 100 as the default value (Pedregosa et al. [2012]). Segal [2003] demonstrates that controlling the depths of individual trees in random forests can result in a small gain in performance. However, Hastie et al. [2001] recommends the used the use of fully-grown trees as they experience the performance gain of controlling the tree depths to be too modest to be worth the additional hyperparameter.

2.5.3 Bumping

Breiman [1996] showed how bootstrap and averaging could improve the predictive performance compared to a single tree. Tibshirani and Knight [1999] proposed a different use of bootstrapping, called *bumping*, that does not involve averaging or combining trees but rather finding a better single tree. It involves using bootstrap samples of the training data to provide candidate models. Due to the greedy splitting procedure, CART tends to find local minima. Bumping can prevent the learning algorithm from getting stuck in suboptimal solutions (Hastie et al. [2001]). Similar to bagging, bootstrap samples are drawn, and a tree is fitted to each one. However, instead of averaging the predictions, one selects the tree fitted on a bootstrap sample that minimizes the loss on the original training data. That is, B bootstrap samples, $\mathcal{D}_b$ where $b = 1, \ldots, B$, are drawn from the original training set $\mathcal{D}$. A tree is fitted to each bootstrap sample, and the tree with the lowest loss on the original training set is selected. Bumping aims to improve function search by perturbing the data in order to move the search around to good areas of function space. For instance, if a few data points lead to a suboptimal solution, removing those points in a bootstrap sample can result in a better solution (Hastie et al. [2001]).

2.5.4 Gradient Tree Boosting

Gradient tree boosting (GTB), introduced by Friedman [2001], is another ensemble learning algorithm that uses trees. Unlike random forests that build multiple trees in parallel and aggregate their predictions, GTB builds trees sequentially, where each tree tries to improve the errors made by the previous ones. The derivation is based on previous literature on gradient boosting, specifically on the second-order Taylor approximation method first introduced by Friedman et al. [2000] and later implemented for CARTs (Chen and Guestrin [2016]; Lunde et al. [2020]). The concept is similar to the Newton-Raphson algorithm described in section 2.2.6. Let $f^{(k-1)} = f_0 + \ldots + f_{k-1}$ be a GTB model with $k-1$ already fitted trees. To sequentially improve the predictions, one wants to add a new tree f_k that minimizes

$$\hat{f}_k = \underset{f_k}{\operatorname{argmin}} \sum \mathcal{L}(y_i, f^{(k-1)}(\mathbf{x}_i) + f_k(\mathbf{x}_i)),$$

approximately to the second order. The second-order approximation of the loss can be computed by using the current model, $f^{(k-1)}$, to get the current predictions $\hat{y}_i^{(k-1)} = f_0(\mathbf{x}_i) + \ldots + f_{k-1}(\mathbf{x}_i)$, which again is used to get the first and second derivatives,

$$g_{i,k} = \frac{\partial}{\partial \hat{y}_i} \mathcal{L}(y_i, \hat{y}_i^{k-1}), \; h_{i,k} = \frac{\partial^2}{\partial \hat{y}_i^2} \mathcal{L}(y_i, \hat{y}_i^{k-1}). \tag{9}$$

The derivatives are used to compute a second-order Taylor approximation of the original loss,

$$\tilde{\mathcal{L}}\left(y, \hat{y}_i^{(k-1)} + f_k(\mathbf{x}_i)\right) = \sum_{i=1}^{n} \left[\mathcal{L}(y_i, \hat{y}_i^{(k-1)}) + g_{i,k} f_k(\mathbf{x}_i) + \frac{1}{2} h_{i,k} f_k(\mathbf{x}_i)^2\right].$$

For a given feature mapping q_k, the prediction $\hat{w}_t$ for a leaf node t is given by

$$\hat{w}_{tk} = -\frac{G_{tk}}{H_{tk}}, \; G_{tk} = \sum_{i:\mathbf{x}_i \in R_{tk}} g_{ik}, \; H_{tk} = \sum_{i:\mathbf{x}_i \in R_{tk}} h_{ik}, \tag{10}$$

and results in the following improvement of the current model's training loss

$$\tilde{\mathcal{L}}^{(t)}(q_k) = -\frac{1}{2} \sum_{t=1}^{T} \frac{G^2}{H}. \tag{11}$$

Using Equation 11, a loss reduction similar to Equation 8 can be established,

$$\tilde{\mathcal{R}}(s, j) = \frac{1}{2} \left[\frac{(\sum_{i:x_i \in R_l} g_i)^2}{\sum_{i:x_i \in R_l} h_i} + \frac{(\sum_{i:x_i \in R_r} g_i)^2}{\sum_{i:x_i \in R_r} h_i} - \frac{(\sum_{i:x_i \in R_{root}} g_i)^2}{\sum_{i:x_i \in R_{root}} h_i}\right]. \tag{12}$$

Using the CART algorithm together with this formula for evaluating the split candidates, trees can be built sequentially, where each new tree aims to improve the errors made by the previous trees. Algorithm 2 sums up the full second-order GTB procedure.

Algorithm 2: Second order gradient tree boosting. (Hastie et al. [2001]; Chen and Guestrin [2016]; Lunde et al. [2020])

```
Input  : Training data $\mathcal{D} = \{\mathbf{X}_i, y_i\}_{i=1}^n$
         Differentiable loss function $\mathcal{L}(\cdot,\cdot)$
         Learning rate $\eta \in (0,1]$
         Number of boosting iterations $K$
Output: The model, $f^{(K)}$
1 $f^{(0)} = \hat{w} = \text{argmin}_w \sum_{i=1}^n \mathcal{L}(y_i, w)$
2 for $k = 1$ to $K$ do
      i. Compute derivatives (9)
      ii. Determine the tree structure $q_k$ by the recursively selecting the binary split $j$ and $s$
          that maximizes (12) until stopping criterion is reached
      iii. Compute the leaf predictions (10), given $q_k$
      iv. Scale the tree with the learning rate, $f_k(\mathbf{x}) = \eta \sum_{t=1}^T w_{tk} I(\mathbf{x} \in R_{tk})$
      v. Update the model $f^{(k)}(\mathbf{x}) = f^{(k-1)}(\mathbf{x}) + f_k(\mathbf{x})$
3 Output the model: return $f^{(K)}$
```

The hyperparameter "learning rate" $\eta \in (0,1]$ in Algorithm 2 is also known as shrinkage (Friedman [2002]), and is used to shrink the effect of each new tree on the ensemble by a constant factor. The shrinking of trees leaves space for future trees to improve the model. Note for squared error loss the second-order approximation is exact. Furthermore, the special case of $\eta = 1$ and $K = 1$ results in a single tree. By letting the $f^{(0)}$ be a constant model predicting the mean of the response variable, $f^{(0)}(\mathbf{x}) = \bar{y}$, then $f^{(0)}(\mathbf{x}) + f_1(\mathbf{x})$ will be equivalent to a standard CART model. However, for Poisson loss, the second-order approximation is just that, an approximation. As a result, the $f^{(0)}(\mathbf{x}) + f_1(\mathbf{x})$ will only be approximately a standard CART model.

2.5.5 Statistical Implication of Greedy Splitting

Statistical inference using greedy splitting differs from fixed splitting, e.g., splitting at a position independent of the training loss. The latter can rely on the central limit theorem (CLT), while the former cannot, as it is a multiple comparison procedure with different statistical properties that require adjustment (Jensen and Cohen [2000]). For simplicity, consider the fitted response w of a node with squared error loss, which is the average of the response variable in that node. Consider a greedy splitting of a (local) root node. Let $w_l(\pi)$, $\pi \in [0,1]$, be a function of the fitted response in the left node given the proportion of the data going from the root node to the left node, where $\pi = 0$ and $\pi = 1$ correspond to none and all of the data, respectively. With fixed splitting (i.e., π being fixed), CLT asserts that the w_l converges to a normal distribution, but not for greedy splitting. The asymptotic behavior of $w_l(\pi)$ when moving more and more data from the root node to the left node by moving the split point from left ($\pi = 0$) to the right ($\pi = 1$) can be described by Donsker's invariance principle (Donsker [1951]). Donsker's invariance principle states that function $w_l(\pi)$ converges to a Brownian bridge. The same applies to the right node $w_r(\pi)$. Since the squared error loss of w_l and w_r is quadratic, then the squared error loss of $w_l(\pi)$ and $w_r(\pi)$ is quadratic over a Brownian motion (a bivariate Gaussian process). The statistical implication of greedy splitting is important for the next subsection, where an information theoretic approach for determining the tree's complexity is introduced.

2.5.6 Adaptive Tree Complexity

Typically the tree complexity is determined through a hyperparameter search of the stopping criterion (*max depth* or *min sample leaf*) or by first building a very large tree and then pruning it using *cost-complexity pruning* (Hastie et al. [2001], p. 308). Lunde et al. [2020] introduce an information theoretic approach for determining the tree complexity without the need for hyperparameter tuning. The theory behind the approach involves a significant amount of statistical and mathematical theory. I will provide a simplified description of the approach, but I encourage readers to read the paper of Lunde et al. [2020] for a more detailed explanation. Lunde et al. [2020] show (under assumptions typical for a H_0 hypothesis and a feature x) how the GTB's optimism C (test loss minus training loss) of the greedy splitting process converges to a specific Cox-Ingersoll-Ross (CIR) process (Cox et al. [1985]). This convergence is established by using the fact that the second-order approximation of the loss is quadratic in $w_l(\pi)$ and $w_r(\pi)$ and using the bivariate Gaussian process obtained via Donsker's invariance principle (Section 2.5.5).

The approximation of the generalization loss reduction $\tilde{\mathcal{R}}^0$ for a node is the training loss reduction $\mathcal{R}$ plus the difference in the approximation of optimism in (local) root and stump,

$$\tilde{\mathcal{R}}^0 = \mathcal{R} + \tilde{\mathcal{C}}_{root} - \tilde{\mathcal{C}}_{stump}.$$

The splitting process of a branch is stopped when $\tilde{\mathcal{R}}^0 < 0$. The great advantage of the adaptive tree complexity is that it alleviates the need for hyperparameter tuning and pruning. The optimism in the root node can be approximated by

$$\tilde{\mathcal{C}}_{root} \approx \frac{\sum_{i \in I}(g_i + h_i \hat{w}_t)^2}{n \sum_{i \in I} h_i}, \tag{13}$$

where $g = \partial \mathcal{L}(y_i, f(x_i))$ and $h = \partial^2 \mathcal{L}(y_i, f(x_i))$ are the first and second order gradient of the loss function, which corresponds to the Takeuchi Information Criterion (TIC) (Takeuchi [1976]).

The tree optimism cannot be handled directly by TIC when a variant of greedy splitting is involved in learning $q(x)$. Therefore, Lunde et al. [2020] employ the Donsker invariance principle from Section 2.5.5 and show that locally for the model selection choice between a split (resulting in a stump model) and no-split (only a single root node), the loss reduction optimism $\tilde{\mathcal{C}}_R = \tilde{\mathcal{C}}_{root} - \tilde{\mathcal{C}}_{stump}$ can be estimated as

$$\tilde{\mathcal{C}}_R = -\tilde{\mathcal{C}}_{root} \pi \mathbb{E}\left[\max_j B_j\right]. \tag{14}$$

Here $B_j = \max_\tau S_j(\tau)$, where S_j is a CIR process, and π is the fraction of training data passed to the node. $\tilde{\mathcal{C}}_{root}$ is calculated as Equation 13. Estimating $\mathbb{E}[\max_j B_j]$ is complicated. Following Lunde et al. [2020], S_j is found to be a specification of the CIR process with parameters $\alpha = 2$, $\beta = 1$, and $\sigma = 2\sqrt{2}$. Using the specification of the CIR process, extreme value theory can be used to approximate

$\max_\tau S_j(\tau)$. Since the CIR process has a gamma stationary distribution, the maximum of a large number of CIR-generated variables converges to the Gumbel distribution (i.e., the CIR process is in the maximum domain of attraction of the Gumbel distribution), $B_j \sim \mathit{Gumbel}$. An asymptotic approximation of B_j can be obtained by fitting the CIR to the Gumbel distribution. For more than one feature, the features are assumed to be independent, and the expectation can be approximated as

$$\mathbb{E}\left[\max_j B_j\right] = \int_0^\infty 1 - \prod_{j=1}^m P(B_j \leq z)dz. \tag{15}$$

With the approximate expectation from Equation 15, the loss-reduction optimism $\tilde{\mathcal{C}}_R$ can be computed and used to calculate the approximation of the generalization loss reduction $\tilde{\mathcal{R}}^0$. Figure 8 compares the complexity of a tree using the adaptive method to a tree using hyperparameter search through grid search and cross-validation.

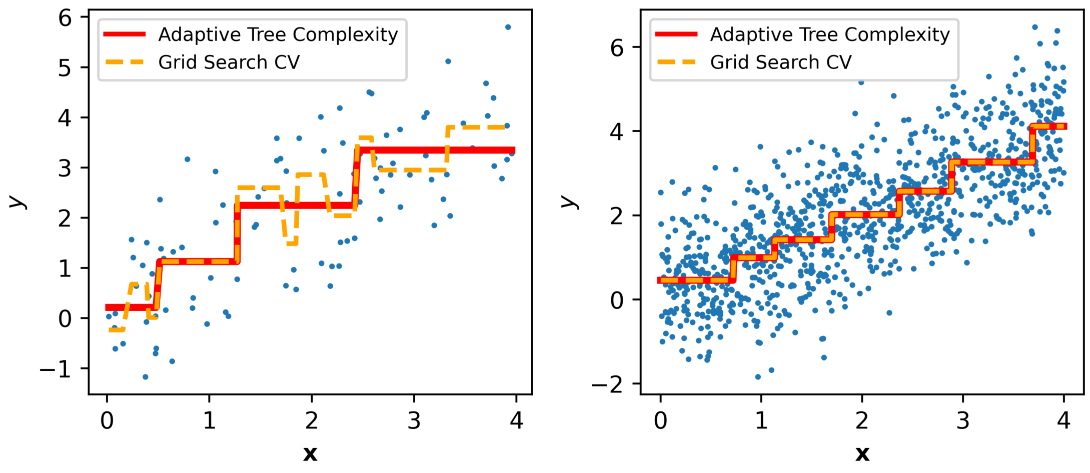


Figure 8: Adaptive tree complexity vs. grid search
The figure compares the predictions of a regression tree using adaptive tree complexity to a regression tree using a grid search with 5-fold cross-validation to find tree complexity. Each step on the function corresponds to a region (i.e., more regions means a more complex tree structure). The figure shows the adaptive tree complexity method adaptive increase the complexity as the data size increases from $n = 100$ (left plot) to $n = 1000$ (right plot). The grid search used hyperparameters and values $\mathit{max\ depth} = [5, 10, \infty]$, $\mathit{min\ sample\ leaf} = [1, 5]$, and $\mathit{cost\ complexity\ alpha} = [0, 0.01]$.

In addition to adaptively determining the complexity of trees, the information criterion of Lunde et al. [2020] can also be used to automatically determine when to stop the iterative boosting procedure of the GTB algorithm by stopping when

$$\eta(2-\eta)\mathcal{R} + \eta\tilde{\mathcal{C}}_R < 0.$$

Based on the theory above, Lunde et al. [2020] developed a fast gradient tree boosting algorithm called `aGTBoost` in which the complexity of trees and the number of boosting iterations are adaptively determined.

3 Methodology

In this section, I define the notion of stability used in the thesis. I also introduce the two conflicting objectives of optimizing both performance and stability and present various update strategies aimed at optimizing both objectives. The update strategies require some modifications to the original CART algorithm which also will be presented in this section.

3.1 Performance Measure

To evaluate and compare models' performances, a performance measure is needed. This thesis uses the mean squared error (mse)

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(\mathbf{x}_i))^2, \tag{16}$$

and mean Poisson deviance,

$$\frac{2}{n}\sum_{i=1}^{n}\left[y_i \log\left(\frac{y_i}{e^{\hat{f}(\mathbf{x}_i)}}\right) - \{y_i - e^{\hat{f}(\mathbf{x}_i)}\}\right], \tag{17}$$

as performance measures. The mean Poisson deviance is based on the Poisson likelihood and loss. It is a common performance measure for measuring the overall discrepancy between observed and predicted counts in a Poisson regression model. The mse is used when the learning algorithm aims to minimize the squared error loss (2), whereas mean Poisson deviance is used when minimizing the Poisson loss (5).

3.2 Update Stability

To recap from Section 2.3.4, stability is traditionally defined as how much a model's predictions change due to changes in the training data, such as the replacement or removal of a data point. It reflects the impact of each data point on the learning process and is often used to establish a bound on the generalization error. However, this thesis is concerned with how much a model's predictions change due to updating the model when more data becomes available, which I named *update stability.* Update stability is a performance measurement to determine how stable a model update is, rather than a bound on the generalization error.

A model f_1 is trained at time t_1 with the training data $\mathcal{D}_1$. At t_2 more data becomes available. The model is updated using all available data $\mathcal{D}_2$ to get the updated model, f_2 (see Figure 9 for notation). Update stability is the discrepancy between the predictions of f_1 and f_2. Similar to the choice of loss, how to measure the discrepancy depends on the data. When $y|\mathbf{x} \sim \mathcal{N}(\mu, \sigma)$ is assumed, then mean squared error is a natural measurement,

$$S_{mse} = \frac{1}{n}\sum_{i=1}^{n}(f_1(\mathbf{x}) - f_2(\mathbf{x}))^2. \tag{18}$$

The S_{mse} measures the average squared distance between the predictions. For claims frequency estimation, which assumes $y|\mathbf{x} \sim \text{Pois}(\lambda)$, squared distance is

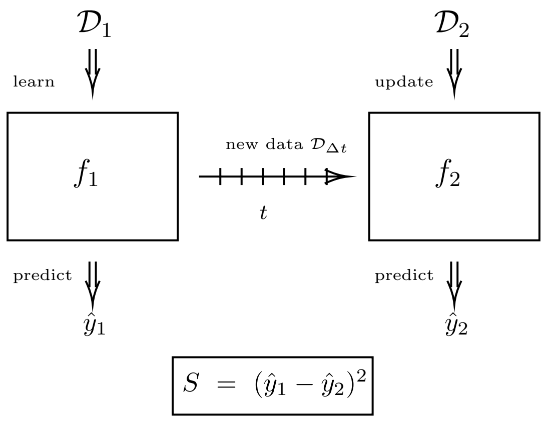


Figure 9: Update stability

The figure explains the notion of update stability. Let $\mathcal{D}_1$ be the training data at time $t = 1$. With $\mathcal{D}_1$, a model f_1 is learned and outputs the predictions $\hat{y}_1 = f_1(\mathbf{x}_{\text{val}})$ on a validation set. At the time t_2 more data is available $D_{\Delta t}$, where $\Delta t = t_2 - t_1$. The model is updated using data $\mathcal{D}_2 = \mathcal{D}_1 \cup \mathcal{D}_{\Delta t}$ to get the updated model, f_2. The updated model outputs the predictions $\hat{y}_2 = f_2(\mathbf{x}_{\text{val}})$ on the same validation set. The update stability is the discrepancy between $\hat{y}_1$ and $\hat{y}_2$. Here represented as the squared error.

not appropriate as it will have a larger emphasis on high-frequency predictions than low-frequency predictions. Hence, a different stability measure is needed. After discussing with a domain expert, a new stability measure was defined for claims frequency estimation,

$$S_{sdlr} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\bar{\hat{\epsilon}} - \hat{\epsilon}_i)^2} \text{ , where } \hat{\epsilon}_i = \log\left(\frac{f_2(\mathbf{x}_i)}{f_1(\mathbf{x}_i)}\right), \tag{19}$$

where $\bar{\hat{\epsilon}}$ is the average log ratio of the predictions. S_{sdlr} is the standard deviation of the log ratio of the predictions from f_1 and f_2. The ratios measure the relative change in the predictions and are in the range $[0, \infty]$. The log transformation makes the ratios symmetrical around zero in the range $[-\infty, \infty]$. S_{sdlr} can be used to produce a standard deviation-based confidence interval for the change from f_1 to f_2:

$$-S_{sdlr} < \log\left(\frac{f_2(\mathbf{x}_i)}{f_1(\mathbf{x}_i)}\right) < S_{sdlr}$$
$$e^{-S_{sdlr}} \times f_1(\mathbf{x}_i) < f_2(\mathbf{x}_i) < f_1(\mathbf{x}_i) \times e^{S_{sdlr}}$$

For example if $S_{sdlr} = 0.3$, then 68% of data satisfy $74\% \times f_1(\mathbf{x}_i) < f_2(\mathbf{x}_i) < 135\% \times f_1(\mathbf{x}_i)$. If the log-deviation is normally distributed, approximately 32% of the predictions will have a deviation outside this interval. For small values, e^x is a close approximation of $1 + x$ and S_{sdlr} can therefore be interpreted directly as a rough estimate of a kind of coefficient of variation of the given prediction. For both S_{mse} and S_{sdlv}, low values indicate stability and large values indicate instability.

Note that stability in the traditional sense refers to how much predictions change due to the removal or replacement of a data point and the order of the removal or replacement of a data point does not matter. For update stability, the order does matter. For instance, the discrepancy between f_1 and f_2 may differ if f_1 is trained on $\mathcal{D}_{\Delta_t}$ rather than $\mathcal{D}_1$. Throughout the rest of the thesis, the term stability will refer to update stability.

3.3 Performance-Stability Trade-off

The changes in the predictions from t_1 to t_2 can be attributed to two main factors: noise in the datasets or the additional information gained from more data. Ideally, a learning algorithm would be able to adapt to new information, ignore the noise in data, and remain close to f_1. However, achieving such a system is challenging since allowing the algorithm to adapt to new information often comes at the expense of increased risk of adapting to noise, reducing stability. Similarly, prioritizing stability reduces the risk of adapting to noise but can limit the algorithm's ability to adapt to new information, reducing performance. The goal is to have a learning algorithm that is Pareto-efficient, meaning there is no change in the current performance-stability trade-off that will lead to improvement of performance or stability without compromising the other. There may exist multiple Pareto-efficient situations. The set of Pareto-efficient solutions is called the Pareto frontier.

3.4 Modifications to CART

This thesis modifies the CART algorithm by using the second-order approximation of loss reduction $\tilde{\mathcal{R}}$ rather than the exact loss reduction $\mathcal{R}$, i.e., the modified CART uses the GTB procedure with $K = 1$ and $\eta = 1$. Remember from Section 2.5.4 that the GTB procedure with $K = 1$ and $\eta = 1$ results in a CART tree and, for squared error loss, this tree will be identical to a CART tree built using exact loss reduction. The benefit of using the second-order approximation of loss reduction is that it allows for the use of the adaptive tree complexity method from Section 2.5.6. This alleviates the need for multiple hyperparameters, which are commonly utilized to optimize tree complexities. Another benefit is that it is easy to add custom loss functions as long as the loss function is twice differentiable. This comes in handy as several of the proposed update methods involve modification of the original loss function used to train the model.

It is worth mentioning a challenge with using the second-order approximation for Poisson loss. For high-frequency observations that are substantially larger than the initial constant prediction, the Newton-Raphson algorithm might take too large steps leading to larger predictions than expected due to approximation errors from the Equation 10. The same issue occurs for low-frequency observations, in this case, Newton-Raphson takes too small steps, also leading to larger predictions than expected. In gradient boosting, the inaccuracy in step size, w, is handled by shrinkage and sequentially adding trees to correct for the errors of the previous tree. For the modified CART, the inaccuracy in w needs to be addressed. One possible but complex solution might be the use of *Adaptive Damping*, which can adaptively adjust the inaccuracy of the step size (Grosse [2021]). However, since experiments show (Appendix A) that regions based on the second-order approximation of the loss reduction used in this thesis result in similar regions as the exact loss reduction, a simpler solution would be to use the exact MLE of w instead approximate MLE (Equation 10).

To summarize, the modified CART algorithm involves using the second-order approximation of loss reduction instead of the exact loss reduction to find the tree splits. But due to inaccuracy in the second-order approximation of the leaf prediction w for Poisson loss, the algorithm uses the exact MLE of w,

$$w_{\text{MLE}} = \Phi\left(\frac{1}{n}\sum_{i}^{n} y_i\right),$$

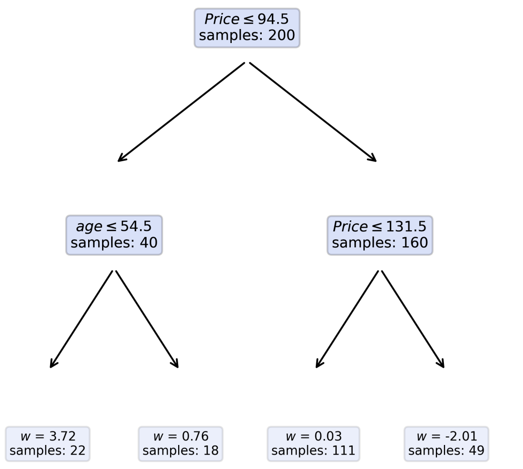
where $\Phi(\cdot)$ is the link function.

3.5 Update methods for Regression Tree

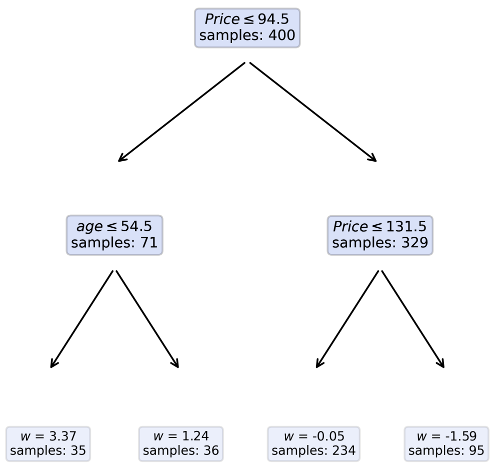
With the modified CART algorithm established, the proposed method of updating it can be introduced. This thesis proposes five methods for updating regression trees using the modified CART algorithm. The proposed methods are based on the use of a node data structure containing the node's split information (j and s), prediction w, and pointers to its children (l and r).

3.5.1 Naive Update

Clearly, not updating the tree will be the most stable solution, but then the new information is not utilized. Expanding on that thought, the most stable update method will be to leave the current tree structure, q, as is and only update the tree's predictions w_t for $t \in [1, T]$ to fit with all the available data. Since the naive update method (`NU`) only updates the leaf node predictions, it makes the assumption that the tree structure of f_1 is sufficiently good and does not need updating. Thus, the `NU` method heavily focuses on stability, and any improvement in performance is due to variance reduction of the leaf predictions. The other update methods, presented in the upcoming subsections, consider finding a better balance between performance and stability. The `NU` method is presented in Algorithm 3. Figure 10 shows how the model change from t_1 to t_2 using `NU`.



(a) Tree at time t_1.



(b) Tree at time t_2.

Figure 10: Model updating using `NU`
The figure shows how the tree changes from t_1 (a) to t_2 (b) using `NU` using the `Carseats` dataset. The tree structure and splits of (b) are the same as (a), but the predictions in the leaves are updated. Details about the `Carseats` dataset are provided in Appendix C.

Algorithm 3: Naive update

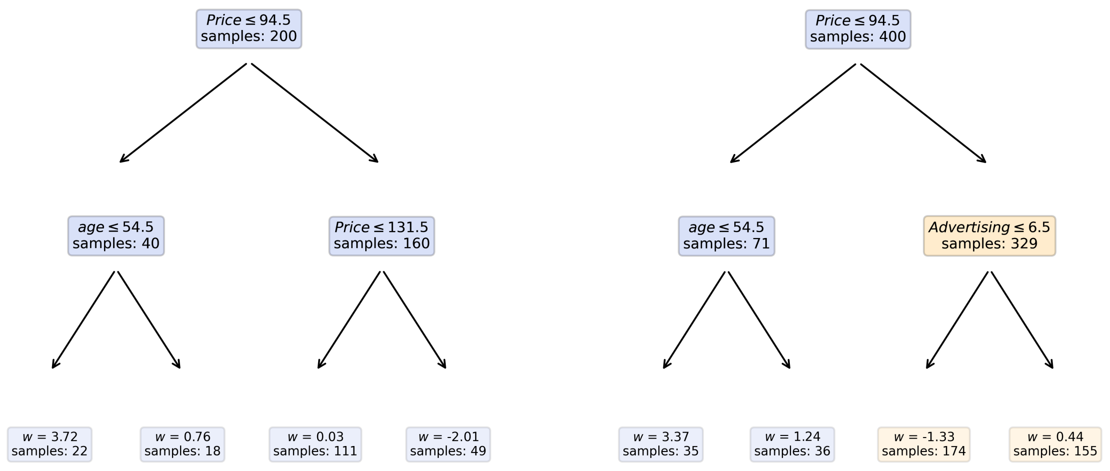
```
input  : Training data $\mathcal{D} = \{\mathbf{X}_i, y_i\}_{i=1}^n$ at time $t_2$
         node, the root node of a tree.
output: updated root node node
1 Function update($\mathcal{D}$, node):
2     if node is a leaf then
3         Set the node's prediction to $\frac{1}{n}\sum_{i=1}^n y_i$
4         return node
5     Let $l$ and $r$ be the node's left and right child
6     Let $\mathcal{D}_l$ and $\mathcal{D}_r$ be the left and right partition of $\mathcal{D}$ given the node's split information $j$
       and $s$
7     $l$ = update($\mathcal{D}_l$, $l$)
8     $r$ = update($\mathcal{D}_r$, $r$)
9     return node
```

3.5.2 Tree Reevaluation

The *Tree Reevaluation* (`TR`) is an extension of the `NU`. Similar to `NU`, the prediction in the tree is updated, but `TR` also allows for parts of the tree structure to change if one is confident that it will lead to a better tree structure. `TR` uses a top-down reevaluation of the tree's nodes. The idea is that as more data is available the current tree structure may no longer be the best and some parts of the tree may need to be updated. The method is based on Hoeffding Trees and Hoeffding bound (Manapragada et al. [2018]). Hoeffding bound states that given n independent random variables $v_1, \dots, v_n$ with range R and mean $\bar{v}$, the true mean is at least $\bar{v} - \epsilon$ with probability $1 - \delta$ where ϵ is

$$\epsilon = \sqrt{\frac{R^2 \ln 1/\delta}{2n}}. \tag{20}$$

Using the idea of Hoeffding bound, `TR` can check for each node whether the ratio of loss reduction of the current split $\{j,s\}_a$ and best split, $\{j,s\}_b$, $\text{?}\mathcal{R} = \frac{\mathcal{R}_{\{j,s\}_b}}{\mathcal{R}_{\{j,s\}_a}}$, is likely to be greater than 1. If so, the best split can be selected with a degree of confidence. For a given tolerance δ, if $\text{?}\mathcal{R} > 1 + \epsilon$ then one can assert with confidence $1 - \delta$ that the split $\{j,s\}_b$ is better. By adding a small value, say α, to $1 + \epsilon$ then one can require $\{j,s\}_b$ to be at least $\alpha \times 100$ percent better than $\{j,s\}_a$. Here δ and α are hyperparameters, where δ determines the confidence in the new split is better and α determines how much better the new split is required to be. The tree is updated by using a top-down reevaluation of the nodes in the tree. For each node the reduction ratio $\text{?}\mathcal{R}$ is computed, if the ratio is above the threshold $(1 + \alpha) + \epsilon$, the current node and all of its descendants are replaced with a new subtree, where the new subtree is built using the CART process with the part of the data that ends up in the node. Otherwise, the current node is kept. For leaf nodes, they are replaced with internal nodes if it is possible to expand them using the CART process. The `TR` method is presented in Algorithm 4. Figure 11 shows how the model change from t_1 to t_2 using the `TR` method. The updated tree is identical to the updated tree using the `NU` except for the subtree highlighted in orange. Here `TR` found, with a confidence 95%, that the split `Advertising` ≤ 6.5 is 5% better than the previous split `Price` < 131.5 and replaces the subtree with a new subtree.



(a) Tree at time t_1. **(b) Tree at time t_2.**

Figure 11: Model updating using `TR`

The figure shows how the tree changes from t_1 to t_2 using `TR` with a $\delta = 0.05$ and $\alpha = 0.05$ using the `Carseats` dataset. The orange nodes highlight the subtree that has changed. Details about the `Carseats` dataset are provided in Appendix C.

Algorithm 4: Tree Reevaluation

```
input  : Training data at $t_2$, $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$
         $\delta$, the acceptance probability of choosing the wrong split at a given node
         node, the root node of a tree.
output: updated root node node
Function update(node ,$\mathcal{D}$, $\delta$):
    if node has no children then
        return attemptSplit(node, $\mathcal{D}$)
    else
        node, changed = reevaluateSplit(node, $\mathcal{D}$, $\delta$)
        if !changed then
            Let $l$ and $r$ be the nodes left and right child
            Let $\mathcal{D}_l$ and $\mathcal{D}_r$ be the left and right partition of $\mathcal{D}$ given the nodes split
              information
            $l$ = update($l$, $\mathcal{D}_{left}$, $\delta$)
            $r$ = update($r$, $\mathcal{D}_r$, $\delta$)
        return node
Function reevaluateSplit(node ,$\mathcal{D}$, $\delta$):
    Compute the loss reduction ratio $\mathcal{LR}$
    Compute $\epsilon$ using Equation 20
    changed = False
    if $\mathcal{LR} > (1 + \alpha) + \epsilon$ then
        Build new subtree using CART
        Replace node and descendants with new subtree
        changed = True
    return node, changed
Function attemptSplit(node ,$\mathcal{D}$):
    if cannot split then
        return node
    Replace node with a new internal node using CART
    return node
```

3.5.3 Stable Loss

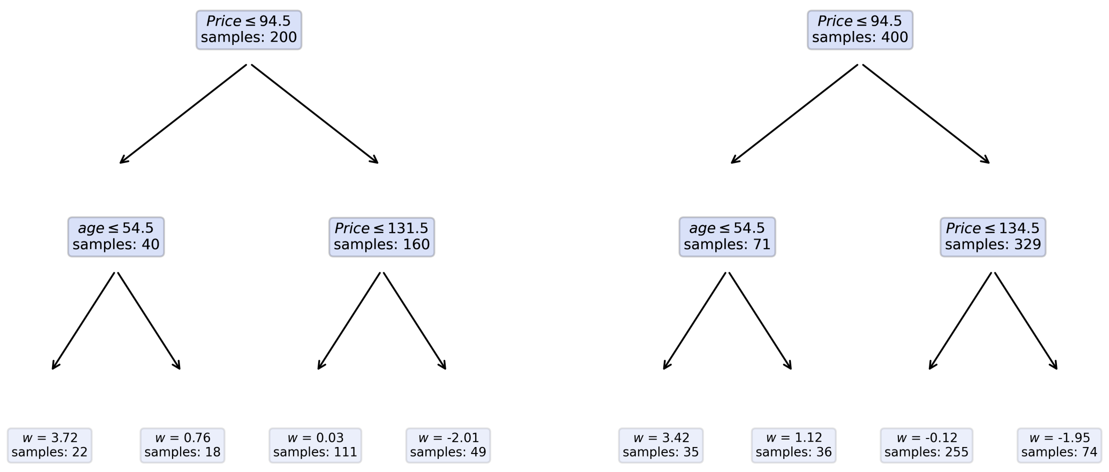
Both `NU` and `TR` use implicit stability regularization. The following methods use explicit stability regularization, starting with what this thesis refers to as *Stable Loss* (`SL`) update. The `SL` method adds a stability regularization term to the loss. The updated tree, f_2 is built using the modified CART algorithm with the stable loss. The method consists of using the predictions of the current tree f_1 on data $\mathcal{D}_2 = \{\mathbf{x}^{(2)}, y^{(2)}\}$ to regularize the loss function and prevent the predictions of f_2 from deviating too much from the predictions of f_1. Therefore the stable loss requires three inputs, the response variable $y^{(2)}$, the predictions from f_1, and the predictions from f_2,

$$\mathcal{L}_{\mathrm{SL}}\left(y^{(2)}, f_1(\mathbf{x}^{(2)}), f_2(\mathbf{x}^{(2)})\right) = \sum_{i=1}^{n}\left[\underbrace{\mathcal{L}(y_i^{(2)}, f_2(\mathbf{x}_i^{(2)}))}_{loss} + \underbrace{\gamma\mathcal{L}(f_1(\mathbf{x}_i^{(2)}), f_2(\mathbf{x}_i^{(2)}))}_{regularization}\right].$$

where γ is a hyperparameter that determines the strength of the regularization (belief in f_1). Here the regularization term uses the same loss function as the loss term, i.e., if squared/Poisson loss then the regularization term is the squared/Poisson loss between the predictions of f_1 and f_2. By using the same loss function for the regularization term, the stable loss has an explicit solution for MLE of a leaf prediction,

$$w_{\mathrm{MLE}} = \Phi\left(\frac{1}{(1+\gamma)n}\sum_{i}^{n} y_i^{(2)} + \gamma f_1(\mathbf{x}_i^{(2)})\right),$$

where $\Phi(\cdot)$ is the link function. The predictions $f_1(\mathbf{x}^{(2)})$ together with $\mathcal{D}_2$ are recursively split and passed to the nodes as the new tree is built and are used to compute the leaf prediction. Figure 12 shows how the model change from t_1 to t_2 using the `SL` method.



(a) Tree at time t_1. **(b) Tree at time t_2.**

Figure 12: Model updating using `SL`

The figure shows how the tree changes from t_1 (a) to t_2 (b) using `SL` with $\gamma = 0.25$ using the `Carseats` dataset. From (a) to (b) the split variables remain the same, but the split value is slightly adjusted for one of the nodes. It is worth noting that although the split variables remain the same in this particular illustration, they may also change. Details about the `Carseats` dataset are provided in Appendix C.

3.5.4 Approximate Bayesian Update

The fourth method is named Approximate Bayesian Update (`ABU`) and expands on the idea of regularizing the splitting process, but instead of having the strength of regularization γ as a hyperparameter it is found analytically using a Bayesian approach. In `SL` γ is a scaler, whereas for `ABU` γ is an array of weights.[1] Section 2.4.1 introduced the Bayesian perspective of model updating and that L1 and L2 regularization has a Bayesian interpretation. Using Bayes theorem from Equation 7, a Bayesian belief system can be established. The idea of `ABU` is rooted in an incremental learning perspective, which is why $\mathcal{D}_1$ and $\mathcal{D}_{\Delta_t}$ are used to illustrate the concept. The concept of ABU can also be applied when using $\mathcal{D}_2$ instead of $\mathcal{D}_{\Delta_t}$. Therefore, once the concept has been explained, `ABU` uses $\mathcal{D}_2$ rather than $\mathcal{D}_{\Delta_t}$, to be consistent with the rest of the thesis. Let $\mathcal{D}_1 = \{\mathbf{x}_i^{(1)}, y_i^{(1)}\}_i^{n_1}$ be the datasets at time t_1 and $\mathcal{D}_{\Delta_t} = \{\mathbf{x}_i^{(\Delta_t)}, y_i^{(\Delta_t)}\}_i^{n_{\Delta_t}}$ be new available data at time t_2, respectively. The aim is to minimize the regularized loss

$$\min \sum_{i=1}^{n_{\Delta_t}} \mathcal{L}\left(y_i^{(\Delta_t)}, f_2(\mathbf{x}_i^{(\Delta_t)})\right) + \gamma \sum_{i=1}^{n_1} \mathcal{L}\left(f_1(\mathbf{x}_i^{(1)}), f_2(\mathbf{x}_i^{(1)})\right).$$

Let $f^*(\mathbf{x}^{(2)}, \hat{w}^*)$ be the belief in the model predictions if it had been trained on $\mathcal{D}_1$ and $\mathcal{D}_{\Delta_t}$ combined, $\mathcal{D}_2 = \mathcal{D}_1 \cup \mathcal{D}_{\Delta_t} = \{\mathbf{x}_i^{(2)}, y_i^{(2)}\}_i^{n_2}$. The belief update is then,

$$\underbrace{P(w|\mathcal{D}_2)}_{posterior} \propto \underbrace{P(\mathcal{D}_{\Delta_t}|w)}_{likelihood} \times \underbrace{P(w^{(1)}|\mathcal{D}_1)}_{prior}.$$

[1]Another difference between `ABU` and `SL` is that `SL` regularizes based on similar prediction on $\mathbf{x}^{(2)}$, whereas `ABU` regularizes based on similar prediction on $\mathbf{x}^{(1)}$.

For squared error loss, the posterior, prior, and likelihood are all normal-based, which results in the following negative log normal posterior,

$$\begin{aligned}
-\log P(w|\mathcal{D}_2) &\propto \underbrace{\frac{1}{\sigma^2_{y|\mathbf{x}}}}_{\substack{\textit{likelihood}\\ \textit{precision}}} \left(y^{(\Delta_t)} - f_2(\mathbf{x}^{(\Delta_t)})\right)^2 + \underbrace{\frac{1}{\sigma^2_{w^{(1)}}}}_{\substack{\textit{prior}\\ \textit{precision}}} \left(f_1(\mathbf{x}^{(1)}) - f_2(\mathbf{x}^{(1)})\right)^2 \\
&= \underbrace{\frac{1}{\sigma^2_{y|\mathbf{x}}}}_{\substack{\textit{likelihood}\\ \textit{precision}}} \mathcal{L}\left(y^{(\Delta_t)}, f_2(\mathbf{x}^{(\Delta_t)})\right) + \underbrace{\frac{1}{\sigma^2_{w^{(1)}}}}_{\substack{\textit{prior}\\ \textit{precision}}} \mathcal{L}\left(f_1(\mathbf{x}^{(1)}), f_2(\mathbf{x}^{(1)})\right) \\
&\propto \mathcal{L}\left(y^{(\Delta_t)}, f_2(\mathbf{x}^{(\Delta_t)})\right) + \frac{\sigma^2_{y|\mathbf{x}}}{\sigma^2_{w^{(1)}}} \mathcal{L}\left(f_1(\mathbf{x}^{(1)}, w^1), f_2(\mathbf{x}^{(1)})\right),
\end{aligned}$$

where $\sigma^2_{y|\mathbf{x}}$ is the variance of the response variable and $\sigma^2_{w^{(1)}}$ is the variance of the predictions. The response and prediction variance ratio $\gamma = \frac{\sigma^2_{y|\mathbf{x}}}{\sigma^2_{w^{(1)}}}$ reflects the belief in f_1. A larger ratio suggests a stronger belief in f_1, whereas a smaller ratio suggests a weaker belief. Both $\sigma^2_{y|\mathbf{x}}$ and $\sigma^2_{w^{(1)}}$ can be estimated by drawing bootstrap samples $\{\mathbf{x}^b_i\}_{i=1}^{n_1}$ from the empirical distribution function (EDF) of $\mathbf{x}^{(\Delta_t)}$ with the assumption that $\mathcal{D}_1$ and $\mathcal{D}_{\Delta_t}$ are from the same distribution. The bootstrap samples are passed to f_1 and the estimated $\hat{\sigma}^2_{y|\mathbf{x}}$ and $\hat{\sigma}^2_{w^{(1)}}$ are the response and prediction variance of the leaf nodes the bootstrap samples belong to. The belief system uses bootstrap estimates from $\mathbf{x}^{(\Delta_t)}$ to approximate statistics of $\mathbf{x}^{(1)}$, hence the name Approximate Bayesian Update. The response variance is straightforward to calculate. It is the variance of the response variable in each node. If the split process had been random or fixed, the prediction variance could have been calculated with the standard formula from the central limit theorem. However, the splitting process is greedy, making the prediction variance more difficult to calculate. In Section 2.5.6, I showed, based on the work of Lunde et al. [2020], how the root optimism $\tilde{C}_{root}$ and stump optimism $\tilde{C}_{stump}$ can be used to find the tree complexity. Lunde et al. [2020] shows that a node's optimism can be written as

$$\tilde{C}_{root} = \mathbb{E}_{x,y}\left[\frac{1}{n}\sum_{i=1}^{n} h_i\right]\sigma^2_w. \tag{21}$$

By solving the equation with respect to σ^2_w the prediction variance can be calculated as

$$\sigma^2_w = \frac{\tilde{C}_{root}}{\frac{1}{n}\sum_{i=1}^{n} h_i} = \frac{n\tilde{C}_{root}}{\sum_{i=1}^{n} h_i},$$

where $\tilde{C}_{root}$ is calculated as per Equation 13. The maximum a posteriori (MAP) for a leaf node using the posterior squared error loss is

$$w_{\text{MAP}} = \Phi\left(\frac{1}{\sum_i^{n_1}\gamma_i + n_{\Delta_t}}\left[\sum_i^{n_{\Delta_t}} y_i^{(\Delta_t)} + \sum_i^{n_1}\gamma_i w_i^{(1)}\right]\right). \tag{22}$$

A similar posterior loss can be established for Poisson loss by swapping the normal negative log-likelihood of data to a negative log-Poisson likelihood (Equation 5). This is equivalent to Poisson loss with L2 regularization,

$$\begin{aligned} -\log P(w|\mathcal{D}_2) &\propto e^{f_2(\mathbf{x}^{(\Delta_t)})} - y^{(\Delta_t)} f_2(\mathbf{x}^{(\Delta_t)}) + \frac{1}{\sigma^2_{w^{(1)}}}(f_1(\mathbf{x}^{(1)}) - f_2(\mathbf{x}^{(1)}))^2 \\ &= \mathcal{L}(y^{(\Delta_t)}, f_2(\mathbf{x}^{(\Delta_t)})) + \frac{1}{\sigma^2_{w^{(1)}}}(f_1(\mathbf{x}^{(1)}) - f_2(\mathbf{x}^{(1)}))^2. \end{aligned}$$

The Poisson posterior loss does not contain the response variance $\sigma^2_{y|\mathbf{x}}$ and γ becomes simply the prior precision, $\frac{1}{\sigma^2_{w^{(1)}}}$. However, experimental testing shows that better results are obtained using $\gamma = \frac{\sigma^2_{y|\mathbf{x}}}{\sigma^2_{w^{(1)}}}$ and are therefore used for both squared error and Poisson loss. The posterior Poisson loss does not have a closed-form solution for the MAP of w and is approximated by using Equation 22. The concept of `ABU` can be applied when using $\mathcal{D}_2$ instead of $\mathcal{D}_{\Delta_t}$. In this case, `ABU` approximates $P(w|\mathcal{D}_1 \cup \mathcal{D}_2)$ rather than $P(w|\mathcal{D}_2)$. To be consistent with the other methods, `ABU` uses $\mathcal{D}_2$ except for in Section 5.2.1.5 where the properties of `ABU` are demonstrated. The `ABU` method is summarized in Algorithm 5.

Algorithm 5: Approximate Bayesian Update.

```
input : Previous tree $f_1$
        Training set at time $t_1$ $\mathcal{D}_1 = \{\mathbf{x}_i^{(1)}, y_i^{(1)}\}_{i=1}^{n_1}$
        Training set at time $t_2$ $\mathcal{D}_2 = \{\mathbf{x}_i^{(2)}, y_i^{(2)}\}_{i=1}^{n_2}$
        Loss function $\mathcal{L} : \mathbb{R}^m \to \mathbb{R}$
output: updated tree $f_2$
1 Function update($\mathcal{D}_2$,$f_1$):
2     sample $\{\mathbf{x}_i^b\}_{i=1}^{n_1}$ from $EDF(\mathbf{x}^{(2)})$
3     get $\sigma^2_{w^{(1)}}{}_i^b$ and $\sigma^2_{y|\mathbf{x}}{}_i^b$ from $f_1$ using $\{\mathbf{x}_i^b\}_{i=1}^{n_1}$
4     train $f_2$ using loss $\mathcal{L}\left(y^{(2)}, f_2(\mathbf{x}^{(2)})\right) + \sum_{i=1}^{n_1} \frac{\sigma^2_{y|\mathbf{x}}{}_i^b}{\sigma^2_{w^{(1)}}{}_i^b}(f_1(\mathbf{x}_i^b) - f_2(\mathbf{x}_i^b))^2$
5     return $f_2$
```

Note, an important detail is that the bootstrap procedure of $\mathcal{D}_1$ from `ABU` is partially non-parametric and partially parametric. The bootstrap of $\mathbf{x}$ is non-parametric whereas the bootstrap of $y|\mathbf{x}$ comes from the model (i.e., parametric), meaning $\mathcal{D}_1^b = \{\mathbf{x}_i^b, f(\mathbf{x}_i^b)\}$.

3.5.5 Bumping Approximate Bayesian Update

Bumping Approximate Bayesian Update (`BABU`) is based on the principle of bumping and ideas from semi-supervised learning to improve the function search. Bumping was introduced in Section 2.5.3 and is a stochastic search that aims to improve the function search by perturbing the data in order to move the search to more promising areas of model space. With an initial model trained on the $\mathcal{D}_1$ (prior), `BABU` uses ideas from bumping and self-training to induce an updated model (posterior) that guides the stochastic search iteratively toward a more promising CART model. In more detail, the `ABU` method approximates the posterior $P(w|\mathcal{D}_1 \cup \mathcal{D}_2)$ with $P(w|\mathcal{D}_1^b \cup \mathcal{D}_2)$ where $\mathcal{D}_1^b$ is sampled from $\mathcal{D}_2$. `BABU` utilizes this property by first learning an initial tree using $\mathcal{D}_1$ to obtain the prior $P(w^{(1)}|\mathcal{D}_1)$. By treating $\mathcal{D}_1$ as a "new" dataset, and applying the update method of `ABU` one gets a posterior $P(w|\mathcal{D}_1^b \cup \mathcal{D}_1)$ where $\mathcal{D}_1^b = \{\mathbf{x}_i^b, f(\mathbf{x}_i^b)\}$ is sampled from $\mathcal{D}_1$. From a self-training perspective, $\mathbf{x}_i^b$ can be viewed as an unlabeled example and $f(\mathbf{x}_i^b)$ as a pseudo-label. The posterior $P(w|\mathcal{D}_1^b \cup \mathcal{D}_1)$ is an approximation of the posterior of a model if $\mathcal{D}_1$ was of twice the size. By letting the posterior be the new prior one can iterate the learning process B times to approximate a posterior of a model if $\mathcal{D}_1$ was of B times the size. Here B is a hyperparameter that determines the number of self-learning iterations. Larger values of B should, in theory, lead to more stable models due to the prediction variance decreasing as the size of the training data increases, which makes the regularization term more dominant. Additionally, for large values of B, the dataset will mainly consist of pseudo-labeled data, also contributing smaller prediction variance. In terms of bias-variance trade-off, larger training data leads to less model variance. Since model variance has an inverse relationship with stability, larger training data increases stability. `BABU` uses bumping and self-training to find a better initial model and `ABU` to update the model. Whereas the other methods only stabilize the updating of a model (f_1 to f_2), `BABU` also stabilizes the learning of a model (searching for a stable initial model f_1).

3.6 Update Methods for Random Forest

All the update methods presented for a regression tree can be applied to random forests using Algorithm 1, as the individual trees can be learned independently of each other. However, some additional implementation is needed in order for the methods to work. Since the random forest randomly selects a subset of features for each split in a tree, each node needs to keep track of its feature subset to ensure that the updated tree uses the same feature subsets. Additionally, any seeds or random states are reset before updating, ensuring the same randomness in both the learning and updating. This also includes the baseline strategy.

Furthermore, there is an option of keeping track of the individual tree's bootstrap sample using a $n \times B$ matrix of bootstrapped indices, where the b'th column vector represents indices of the bootstrap samples of b'th tree in the forest. The number of rows expands as more data becomes available. The purpose of this is to ensure that the trees in the forest are receiving the same bootstrap sample of $\mathcal{D}_1$ when updated, potentially increasing stability. Since one uses the indices to keep track of the bootstrapped samples, the order of the training data must not change. Specifically, this option requires that data points from $\mathcal{D}_1$ are before new data points and in the same order. Keeping track of this indices matrix quickly becomes memory expensive with large forests and large datasets.

This thesis uses random forest implementation without the option of keeping track of the bootstrap indices, but some testing was done where bootstrap indices are kept track of.

3.7 Updating Gradient Tree Boosting

In GTB, the trees are learned sequentially and the current tree depends on the error of the previous trees. Due to the sequential dependencies between trees, one needs to stabilize the predictions of the ensemble and not the predictions of the individual trees. Since each new tree added to GTB aims to improve the training loss of the previous trees, one can build upon the ideas of `SL` and `ABU`, i.e., the use of a stability regularization term to the loss. `SL` is the simplest to implement and is almost identical to `SL` with a single regression tree. Given a GTB model f_1 trained on $\mathcal{D}_1$ an updated GTB model f_2 can be learned with the GTB learning algorithm using the stable loss and the ensemble predictions $f_1(\mathbf{x}^{(2)})$.

Making `ABU` work for GTB is significantly more complex than `SL` due to the task of computing the weights of the prior, $\gamma = \frac{\sigma^2_{y|x}}{\sigma^2_w}$. Since the trees are sequentially dependent in GTB, statistical challenges arise due to cross-correlation between trees. Solving this issue would be a step towards making a Bayesian approach to GTB (or BGTB if you like). However, due to time-constrained, this is left as future work.

This thesis uses the `aGTBoost` algorithm of Lunde et al. [2020] as the baseline GTB and is also extended to include the option of using the `SL` update method.

3.8 Implementation

Based on the theory presented in this section, I have developed a Python library for tree-based methods called `StableTrees` that includes the proposed update methods for regression trees, random forests, and GTB. To optimize performance, the update methods have been implemented in C++ and exposed to Python through C++ bindings. The Python package is available at `https://github.com/MortenBlorstad/StableTrees` or `https://pypi.org/project/stabletrees/`.

4 Data

In this section, the datasets used for the experiment are presented, providing an overview of their source, size, and included variables. Moreover, this section also provides information on any data preprocessing or cleaning to prepare the data for the experiment.

4.1 Simulated Dataset

Simulated data were used when developing the proposed update methods. This subsection presents how the datasets were simulated. Four features are simulated, two continuous, one nominal, and one ordinal. Table 1 shows how $\mathbf{x}$ is simulated.

Feature	Unit	simulation
$x_{\cdot,1}$	interval	$x \sim \mathcal{U}(0, 4)$
$x_{\cdot,2}$	interval	$x \sim \mathcal{U}(0, 4)$
$x_{\cdot,3}$	nominal	$x \sim \mathcal{U}(0, 1)$
$x_{\cdot,4}$	ordinal	$x \sim \mathcal{U}(0, 4)$

Table 1: Summary of the feature variables in the simulated data. The continuous features ($x_{\cdot,1}$ and $x_{\cdot,2}$) were drawn from a continuous uniform distribution, whereas the nominal ($x_{\cdot,3}$) and ordinal ($x_{\cdot,4}$) were drawn from a discrete uniform distribution.

The simulated features were used to create 3 different datasets with different relationships to y. Let $\phi(\mathbf{x})$ be a polynomial function defining the relationship of

$\mathbf{x}$ to y. The data-generating process is $y_i \sim \mathcal{N}(\phi(\mathbf{x}_i), \sigma)$, where $i = 1, \ldots, n$ and $n = 1000$. Tree different polynomials are used to create 3 different cases (datasets). The first polynomial is a linear polynomial,

$$\phi_1(\mathbf{x}_i) = x_{i,1},$$

and defines the first and simplest case, with only feature $x_{\cdot,1}$, ($\mathbf{x} \in \mathbb{R}^1$). The second polynomial is a quadratic polynomial,

$$\phi_2(\mathbf{x}_i) = x_{i,1}^2 + 0.75x_{i,2},$$

and defines the second case, increasing the function complexity by using both feature $x_{\cdot,1}$ and feature $x_{\cdot,2}$, ($\mathbf{x} \in \mathbb{R}^2$). The third polynomial is also a quadratic polynomial,

$$\phi_3(\mathbf{x}_i) = x_{i,1}^2 + 0.75x_{i,2} - 0.25x_{i,4} + 0.1x_{i,3} \times x_{i,1} - 0.05x_{i,4} \times x_{i,2},$$

and defines the third and final case. Here all four features ($\mathbf{x} \in \mathbb{R}^4$) are used to define the relationship to y and are therefore the most complex case of the three. Note also the interaction effects. The simulation of the three case datasets is summarized in Table 2.

Dataset	$n \times m$	Data-generating process
Case 1	1000×1	$y \sim \mathcal{N}(\phi_1(\mathbf{x}), 1)$
Case 2	1000×2	$y \sim \mathcal{N}(\phi_2(\mathbf{x}), 3)$
Case 3	1000×4	$y \sim \mathcal{N}(\phi_3(\mathbf{x}), 3)$

Table 2: Summary of the simulated datasets for the different cases.

4.2 ISLR Datasets

The proposed methods are evaluated using various benchmark datasets for regression from the book "Introduction to Statistical Learning"(ISLR) by Gareth James et al. [2013]. The datasets are diverse and cover areas such as finance, marketing, and sports, as summarized in Table 3. This diversity allows for a comprehensive assessment of the performance and stability of the proposed methods across a broad range of data characteristics.

The datasets contain a mix of continuous and categorical features. Categorical features that are nominal are one-hot encoded, whereas ordinal categorical features are converted to ordinal integers. For the datasets `College`, `Hitters`, and `Wage` the response variable is log-transformed as they are skewed due to a small number of very large values. The squared error loss is sensitive to outliers and the log-transformation reduces the influence of these small numbers of large numbers on the loss, which may lead to increased performance. The `Hitters` dataset contains 59 data points where the response variable has missing values. These data points are removed. The datasets are available at `https://www.statlearning.com/`.

Dataset	n × m	Description
Boston	506 × 13	A dataset containing housing values in 506 suburbs of Boston.
Carseats	400 × 11	A simulated dataset containing sales of child car seats at 400 different stores.
College	777 × 18	Statistics for a large number of US Colleges from the 1995 issue of US News and World Report.
Hitters	263 × 20	Major League Baseball Data from the 1986 and 1987 seasons.
Wage	3000 × 16	Wage and other data for a group of 3000 male workers in the Mid-Atlantic region.

Table 3: Overview of the ISLR datasets.
This table gives an overview of the ISLR datasets. The dimensions are after data preprocessing, i.e., one-hot encoding and removal of missing values. (James et al. [2022])

An overview of the available variables for each of the ISLR datasets is provided in Appendix B to F.

4.3 Claim Frequency Data

To evaluate how the proposed update methods perform for claim frequency estimation, a motor third-party liability (MTPL) insurance portfolio from the French insurance industry is used. The dataset, named freMTPLfreq, contains 413 169 unique policyholders with 10 variables. An overview of the available variables is listed in Table 4.

Variable	Description
PolicyID	The policy ID.
ClaimNb	The number of claims during the exposure period.
Exposure	The period of exposure for a policy, in years.
Power	The power of the car (ordered categorical).
CarAge	The vehicle age, in years.
DriverAge	The driver age, in years (in France, people can drive a car at 18).
Brand	The car brand divided in the following groups: A - Renaut, Nissan and Citroen B - Volkswagen, Audi, Skoda, and Seat, C - Opel, General Motors, and Ford, D - Fiat, E - Mercedes, Chrysler, and BMW, F - Japanese (except Nissan) and Korean, G - Other.
Gas	The car gas, diesel or regular.
Region	The policy region in France (based on the 1970-2015 classification).
Density	The density of inhabitants (number of inhabitants per km^2) in the city the driver of the car lives in.

Table 4: The available variables in the freMTPLfreq dataset (Dutang and Charpentier [2018].

The `freMTPLfreq` dataset is available through the R library CASdatasets (Dutang and Charpentier [2018]).

4.3.1 Data Preprocessing

As mentioned in the introduction, GLMs assume a linear relationship between features and response variables which might be too restrictive as it is common that the data contains non-linear and interaction components. By binning continuous features (i.e., constructing categorical features from continuous features) it is possible to capture potential non-linearities in the relationship between a feature and the response variable which Figure 13 illustrates.

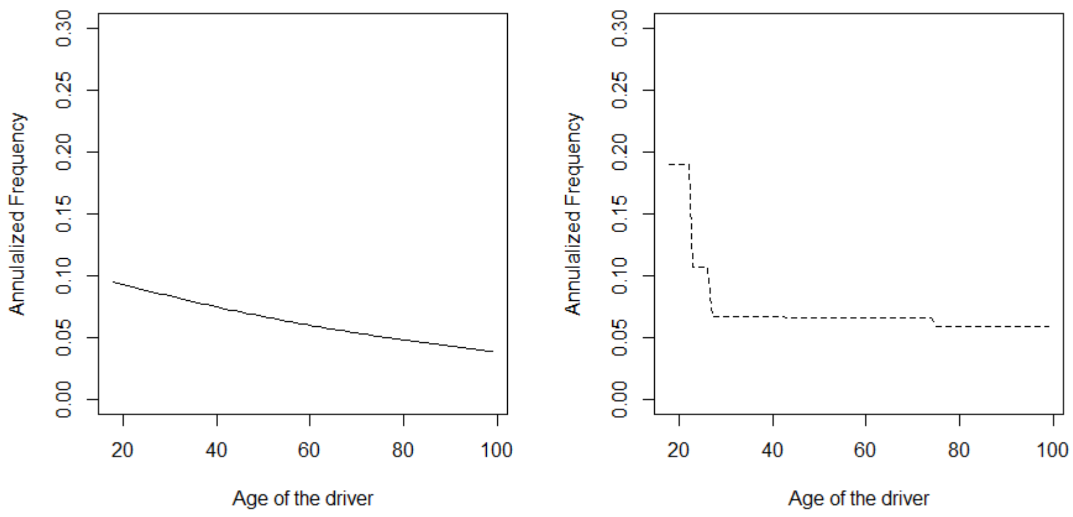


Figure 13: Poisson regression of the annualized frequency on `DriverAge`. Poisson regression of the annualized frequency on the age of the driver, with a *linear model*, and when the age of the driver is a categorical variable (adapted from Charpentier [2014] p. 488).

The book "Computational Actuarial Science with R" by Charpentier [2014] provides a detailed description of how to build predictive models for claim frequency estimation with GLMs using the `freMTPLfreq` dataset, including feature engineering such as binning of continuous features. Table 5 shows the feature engineering of the variables. These variables, plus `Gas`, are one-hot encoded and used to build a predictive model with GLM. Binning is not necessary for tree-based models as they can capture non-linear relationships without the need for binning. The tree-based models use the variables from Table 4 (except `PolicyID`) and nominal categorical features are one-hot encoded and ordinal categorical features are converted to ordinal integers.[2]

[2]GLM uses the features engineered variables plus `Gas` to predict the annualized frequency. The tree-based methods use `Power`, `CarAge`, `DriverAge`, `Brand`, `Gas`, and `Density`. All models use log `Exposure` as an offset.

Variable	Bins
`Power`	$["DEF","GH","other"]$
`CarAge`	$[0, 15, \infty]$
`DriverAge`	$[17, 22, 26, 42, 74, \infty]$
`Brand`	$["F","other"]$
`Density`	$[0, 40, 200, 500, 4500, \infty]$

Table 5: The feature engineered variables for GLM.

4.3.2 Exploratory Data Analysis

Figure 14 shows how `ClaimNb` and `Exposure` are distributed. The portfolio's claim frequency (ratio of the total number of claims to the total exposure in years) is 6.93%. The majority of policyholders (95.3%) are claim-free during their insured period. Only 4.5% of the policyholders file a single claim, whereas the remaining 0.2% of the policyholders file two, three, or four claims during their insurance period. Almost a third of policyholders (29.4%) have an exposure equal to one, meaning they are covered by the insurance for a year. Most policyholders (70.5%) have an exposure below a year and only a few (0.1%) have an exposure greater than a year.

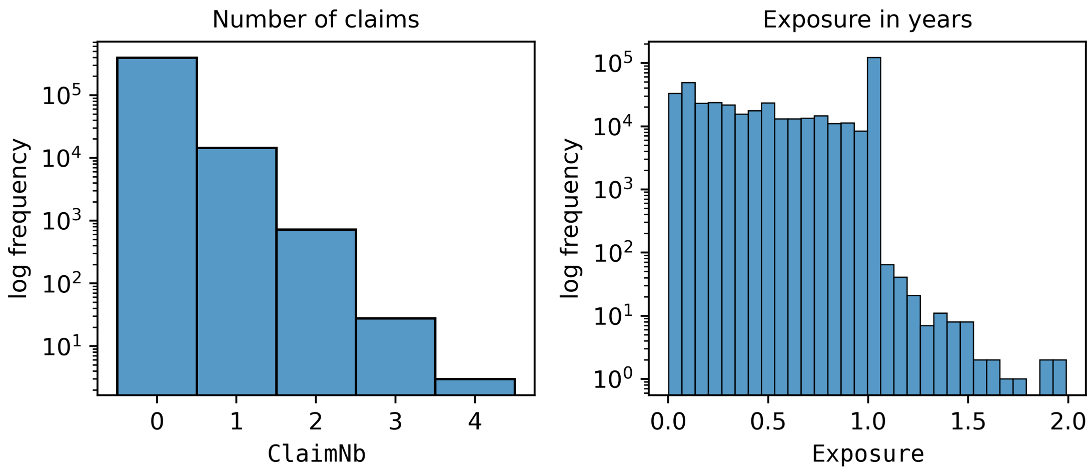


Figure 14: The distribution of the number of claims, exposure in years, and claim frequency per year

Figure 15 shows how the features from Table 4 are distributed. The `freMTPLfreq` dataset contains three categorical features, `Power`, `Brand`, and `Gas`. The `Power` is ordered categorical. Most policyholders (58.3%) have low-power cars (d-f), whereas 28.5% have medium-power cars (g-h). Only 13.2% have high-power (i-o). The `Brand` is divided into 7 groups (A-G, see Table 4). Most policyholders have cars belonging to either group A (52.8%) or F (19.1%). Only 28.1% of the policyholders have cars belonging to the other groups. The `Gas` informs about what type of fuel the cars use. There are two types of fuel, diesel (49.8%) and regular/gasoline (50.2%).

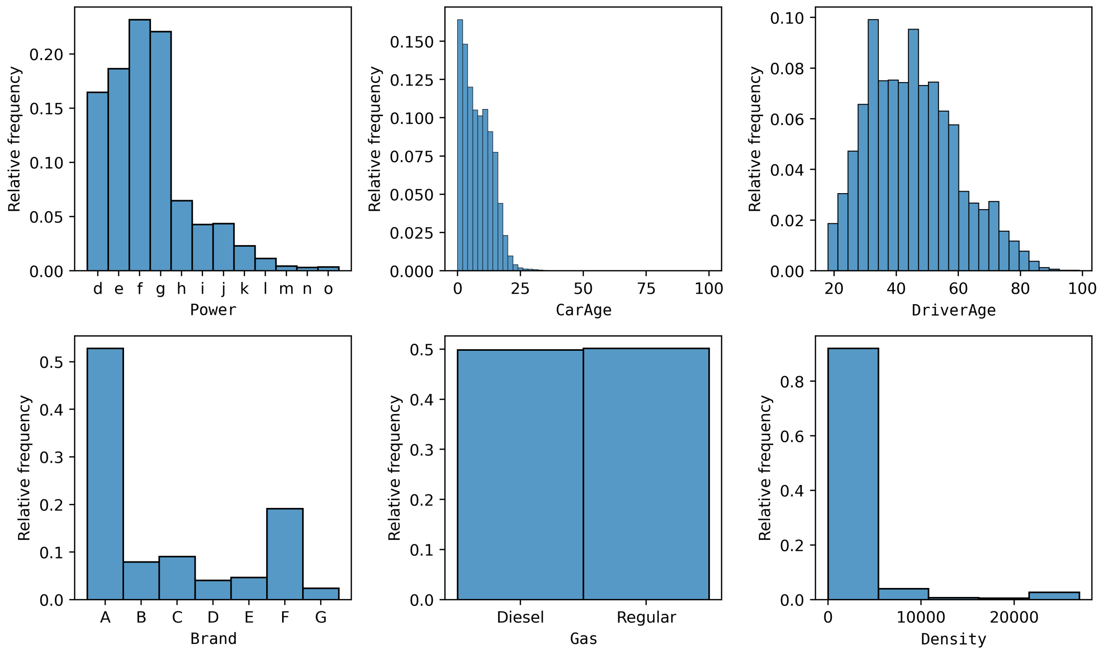


Figure 15: The distribution of the features in the `freMTPLfreq` dataset.

The `freMTPLfreq` dataset contains three continuous features, `CarAge`, `DriverAge`, and `Density`. Most cars are less than 20 years old (97.8%) and 90.2% of the policyholders are aged between 25 and 75. Most policyholders (91.6%) live in cities with less than 5000 inhabitants per km^2.

5 Experiments

In this section, the proposed update methods are evaluated on different regression tasks. The section begins by describing the experimental setup, followed by a presentation of the results. The experiments are structured into three parts. The first part provides an in-depth presentation of the individual methods for regression trees on simulated data used for developing them. Next, the `ISLR` datasets are used to evaluate the proposed methods for regression trees, random forests, and GTB across a broad range of data characteristics. Finally, the `freMTPLfreq` dataset is used to evaluate the methods for claim frequency estimation.

5.1 Experimental Setup

This subsection provides a detailed description of the experimental setup for the three parts of the experiments, starting with the setup for the simulated experiments, followed by the setup for experiments on ISLR datasets and claim frequency estimation.

5.1.1 The Base Setup

All three parts of the experiment use to some degree the same setup and this setup will be referred to as the *base setup*. In this thesis, I am concerned with how the output of a model changes when more data becomes available and how to reduce this change without reducing performance. The base setup aims to reflect how the insurance industry needs to adapt its pricing models to new information and consists of a start time t_1 and an update time t_2. At t_1 a model is initialized and trained on the currently available data $\mathcal{D}_1$. At time t_2 new information is available that is added to $\mathcal{D}_1$ to obtain $\mathcal{D}_2$. The model is then updated using one of the proposed update methods. The performance and stability measures are computed using hold-out test data. To reduce variance in the performance and stability measures, a repeated cross-validation method is used. For each dataset, the dataset is divided into k-folds, where $k-1$ of the folds represents $\mathcal{D}_2$ and half of $\mathcal{D}_2$ represents $\mathcal{D}_1$. The last fold is used as test data. This process is repeated r times and the average of the performance and stability estimates from test data is used to obtain the final performance and stability measures. The values of r and k are specified for each experiment in the following subsections. Algorithm 6 summarizes the base setup.

Algorithm 6: The base setup

```
input  : Dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^n$
         Model $f$
output: Averaged performance and stability measures
Function setup($\mathcal{D}$):
    for $i \in 1, \dots, r$ do
        Shuffle $\mathcal{D}$
        Create $k$-folds
        for $j \in 1, \dots, k$ do
            Let $\mathcal{D}_{test}$ be the $j$th fold
            Let $\mathcal{D}_2$ be the other folds
            Let $\mathcal{D}_1$ be half of $\mathcal{D}_2$
            Train $f$ on $\mathcal{D}_1$ to obtain $f_1$
            Use update method on $\mathcal{D}_2$ to obtain $f_2$
            Compute and store performance score of $f_2$ on $\mathcal{D}_{test}$ using mse (16) or mean
              Poisson deviance (17)
            Compute and store stability score of $f_2$ on $\mathcal{D}_{test}$ using $S_{mse}$ (18) or $S_{sdlr}$ (19)
    return Averaged performance and stability measures
```

5.1.2 Simulated Data

The simulated datasets from Section 4 are used to give an in-depth presentation of the individual methods, highlighting method assumptions and showing how different hyperparameters affect the performance-stability trade-off. The baseline and the update methods use the adaptive tree complexity method, and *min sample leaf* = 5, unless otherwise specified in the individual setup descriptions.[3]

5.1.2.1 Baseline

The baseline update strategy retrains the model from scratch using the modified CART. As a sanity check, I compare the baseline strategy to a regression tree from the popular open-source machine learning library, Scikit-learn (also known as Sklearn) (Pedregosa et al. [2012]). Here the baseline is compared to Sklearn's implementation of the regression tree using the base setup with $r = 1$ and $k = 10$ (i.e., 10-fold cross-validation). Since the baseline uses the adaptive tree complexity method, which does not exist for Sklearn's implementation, the baseline is compared to Sklearn when both use the same fixed set of hyperparameters, *max depth* = 5 and *min sample leaf* = 5. Additionally, I compare the baseline to Sklearn when the baseline uses adaptive tree complexity and *min sample leaf* = 5, while Sklearn uses a grid search with a 5-fold cross-validation to highlight any differences in performance. The hyperparameters and values for the grid search is *max depth* $= [5, 10, \infty]$, *min sample leaf* $= [1, 5]$, and *cost complexity alpha* $= [0, 0.01, 0.1, 0.3]$.

5.1.2.2 Naive Update

The experiment aims to illustrate how the performance of `NU` depends on the size of $\mathcal{D}_1$ as it assumes the initial tree structure learned using $\mathcal{D}_1$ is sufficiently good. The

[3]A minimum requirement of five observations in a node is set for regression trees to ensure enough observation to compute response variance and leaf prediction variance in each leaf node. The response variances and leaf prediction variances node are essential for the proper functioning of both `ABU` and `BABU`.

experiment involves executing the base setup ($r = 10$ and $k = 5$) multiple times on increasing data size using the data-generating processes described in Table 2. Each iteration involves increasing the data size n. The data size starts at $n = 1000$ (i.e, $n_1 = 400$, $n_2 = 800$ and $n_{\text{test}} = 200$) and increases by 1000 until $n = 10000$. As the tree structure for NU only is determined by $\mathcal{D}_1$, one should expect that as the size of $\mathcal{D}_1$ increases, so should the performance as the tree structure should approach the optimal structure.

5.1.2.3 Tree Reevaluation

TR has two hyperparameters δ and α. The experiment for TR entails using the base setup with $r = 10$ and $k = 5$ to compare how different hyperparameter settings of TR affect the performance-stability trade-off. The hyperparameter settings consist of the 9 different combination of $\alpha = [0, 0.05, 0.1]$ and $\delta = [0.05, 0.1, 0.25]$. $\texttt{TR}_{\alpha,\delta}$ refers to TR using the hyperparameters α and δ, expressed as percentages. For example $\texttt{TR}_{0,5}$ means a TR update using the hyperparameters $\alpha = 0$ and $\delta = 0.05$. TR extends NU by allowing (parts of) the tree structure to change if one can with confidence $1 - \delta$ say that the change results in a greater loss reduction. The hyperparameter δ determines the confidence level, while α determines the minimum improvement of the change (Section 3.5.2).

5.1.2.4 Stable Loss

The SL method has one hyperparameter γ, which determines the strength of the explicit stability regularization. The base setup with $r = 10$ and $k = 5$ is used to compare how different values of γ affect the performance-stability trade-off. The different values are $\gamma = [0.1, 0.25, 0.5, 0.75, 0.9]$. $\texttt{SL}_\gamma$ refers to SL using that particular value of γ. For example $\texttt{SL}_{0.1}$ means a SL update using $\gamma = 0.1$.

5.1.2.5 Approximately Bayesian Update

ABU is based on the idea of approximating the posterior of a model trained on $\mathcal{D}_1$ and $\mathcal{D}_{\Delta_t}$ combined using only $\mathcal{D}_{\Delta_t}$. The ABU experiment aims to assess the accuracy of this approximation and showcase the properties of ABU. To this end, a different setup than the base setup is used. The experiment consists of two parts. Both parts use 1000 Monte Carlo simulations from the data-generating process for Case 2. For each Monte Carlo simulation, the size of the entire dataset, $\mathcal{D}_2 = \mathcal{D}_1 \cup \mathcal{D}_{\Delta_t}$, is $n_2 = 1000$. $\mathcal{D}_1$ and $\mathcal{D}_{\Delta_t}$ are non-overlapping and of equal size, i.e., $n_1 = n_{\Delta_t} = 500$. Also, a separate test set of size $n_{test} = 1000$ is simulated from the same data-generating process. The first part assesses the accuracy of the approximation. Here, for each Monte Carlo simulation, the mean squared error on the test set is calculated and used to compute the kernel density estimation (KDE) for a tree trained on $\mathcal{D}_1$ (prior), a tree trained on $\mathcal{D}_{\Delta_t}$ (second tree), a tree trained on $\mathcal{D}_2$ (posterior), and a tree using ABU. ABU involves training an initial tree on $\mathcal{D}_1$ and using ABU on $\mathcal{D}_{\Delta_t}$ to approximate the tree trained on $\mathcal{D}_2$. Additionally, for each Monte Carlo simulation, the change between the prior's predictions and the predictions of the other trees on the test set are calculated and used to compute the stability KDEs for the second tree, ABU, and the posterior tree.

The second part aims to showcase ABU's adaptability. Here the same Monte Carlo simulations as the first part is conducted, but the performance and stability KDEs are computed for a tree trained on $\mathcal{D}_1$ (prior), a tree trained on $\mathcal{D}_2$ (baseline tree), a tree trained on $\mathcal{D}_1 \cup \mathcal{D}_2$ (posterior), and a tree using ABU. ABU involves training an initial tree on $\mathcal{D}_1$ and using ABU on $\mathcal{D}_2$. The posterior tree, $\mathcal{D}_1 \cup \mathcal{D}_2$, represents the posterior tree ABU would approximate without adaptively adjusting the individual γ values.

5.1.2.6 Bumping Approximately Bayesian Update

BABU has a hyperparameter B which determines the number of bumping (self-learning) iterations. The base setup with $r = 10$ and $k = 5$ is used to compare how the value of B affects the performance-stability trade-off. The different values of B in the experiment are $B = [1, 3, 5, 7, 10, 20]$. $\texttt{BABU}_B$ refers to BABU using that particular value of B. For example $\texttt{BABU}_1$ refers to BABU using $B = 1$.

5.1.2.7 Tracking Model Updates Over Time

In the base setup, different update methods are evaluated for a single update. However, in this experiment, the different update methods are evaluated over multiple update periods to illustrate their long-term behavior. The experiment involves simulating a training and a test set, $\mathcal{D}$ and $\mathcal{D}_{\text{test}}$, both of size 1000. Each update method is trained on $\mathcal{D}$, and for each update period, 1000 new data points are added to $\mathcal{D}$, and the models are updated. For each updated model, the performance and stability scores are calculated using the $\mathcal{D}_{\text{test}}$. The experiment evaluates the methods over ten update periods and is repeated 50 times, and the average is used to reduce variance in the estimates. The datasets use the same data-generating process as the Case 1 dataset. The hyperparameters for TR are $\alpha = 0$ and $\delta = 0.05$. SL uses $\gamma = 0.75$, while BABU uses $B = 5$.

5.1.3 ISLR Datasets

The ISLR datasets from Section 4 are used to evaluate the update methods on a broad range of data characteristics using the squared error loss. The experiment is divided into three parts, one for each tree-based method: regression trees, random forests, and GTB. For each part, the base setup is used to obtain performance and stability estimates for the methods. The result is presented as a scatter plot with stability and performance as the axes, where the Pareto frontier is highlighted in bold. The Pareto frontier represents the boundary of the best possible models achievable when optimizing performance and stability simultaneously. The part for regression trees uses the base setup with $r = 5$ and $k = 10$ (i.e., 5 repeated 10-fold cross-validation), whereas random forest and gradient tree boosting use the base setup with $r = 1$ and $k = 10$ (i.e., 10-fold cross-validation).[4]

For regression trees, each tree uses the adaptive tree complexity method, and *min sample leaf* $= 5$. The random forests use the default hyperparameter settings from Section 2.5.2, i.e., 100 trees with *min sample leaf* $= 5$ and *mtry* $= \lfloor m/3 \rfloor$. A hyperparameter search is performed to determine whether controlling for the individual trees' complexity using the adaptive tree complexity method improves performance. For each iteration in the base setup with $r = 1$ and $k = 10$, $\mathcal{D}_1$ is

[4]Due to the performance and stability estimates for regression trees being based on repeated cross-validation, it is necessary to adjust their standard error estimates. The adjustment is required because repeated cross-validation does not introduce any new independent test cases, meaning all data points have already been included once in the initial k-fold validation. The adjustment is based on a heuristic where $\sqrt{k}$ is used in the denominator of the standard error calculation rather than $\sqrt{rk}$ (`https://stats.stackexchange.com/q/448784`).

divided into a training (70%) and a validation set (30%). The baseline forest is trained on the training set and evaluated on the validation set with and without adaptive tree complexity. The setting with the lowest average loss is selected and used for the baseline and the update methods. The GTBs use the `aGTBoost` algorithm of Lunde et al. [2020], which adaptively determines the number of trees in the ensemble and the tree complexity of each individual tree. The learning rate is set to $\eta = 0.1$ for the GTBs. Table 6 summarizes the settings used for the different tree-based methods.

	number of trees	min sample leaf	mtry	adaptive tree complexity	η
regression tree	1	5	NA	yes	NA
random forest	100	5	$\lfloor m/3 \rfloor$	hyperparameter search	NA
GTB	adaptive	1	NA	yes	0.1

Table 6: Setting for the tree-based methods for the `ISLR` datasets

5.1.4 Claim Frequency Data

The `freMTPLfreq` dataset from Section 4 is used to evaluate the update method for the task of claim frequency estimation using the Poisson loss. Here the update methods for regression tree, random forest, and GTB are evaluated and compared to the traditional method of estimating claim frequency, namely the GLM. The base setup with $r = 1$ and $k = 6$ (i.e., 6-fold cross-validation) is used to obtain performance and stability estimates. The result is presented by plotting the performance against stability estimates and highlighting the Pareto frontier. Here 6-fold cross-validation is used to be consistent with other papers that investigate the use of tree-based methods in the insurance industry (Henckaerts et al. [2020]).

Regression tree, random forest, and GTB use the same settings as described in Section 5.1.3. and is summarized in Table 7.

	number of trees	min sample leaf	mtry	adaptive tree complexity	η
regression tree	1	5	NA	yes	NA
random forest	100	5	$\lfloor m/3 \rfloor$	hyperparameter search	NA
GTB	adaptive	1	NA	yes	0.1

Table 7: Setting for the tree-based methods for the `freMTPLfreq` dataset

5.2 Experimental Results

The experiment results are presented in the same order as the setup of the experiments was introduced.

5.2.1 Simulation Experiments

The preceding subsections present the results of the simulated experiments for the individual methods, highlighting method assumptions and showing how different hyperparameters affect the performance-stability trade-off.

5.2.1.1 Baseline

Here, a comparison of the baseline tree to Sklearn's implementation of a regression tree is presented. Table 8 presents the performance and stability of the models using the base setup r = 1 and k = 10. Panel A shows when the baseline and Sklearn use the same hyperparameters, whereas panel B shows when the models aim to optimize the tree structure. With fixed hyperparameters, the baseline and Sklearn give the same performance and stability but differ when optimizing the tree structure.

For optimized trees, the baseline uses the adaptive tree complexity, whereas Sklearn uses a grid search. The performance is very similar across all cases, with Sklearn having a slightly better performance for Case 1. For stability, the difference between the optimized trees is larger, with the baseline tree being more stable across all cases. The difference in stability between the models is particularly large for Case 3. The big advantage of the baseline over Sklearn is that the adaptive tree complexity removes the need for hyperparameter tuning to determine the tree complexity.

	Case 1		Case 2		Case 3	
	performance	stability	performance	stability	performance	stability
Panel A: Fixed						
Baseline	1.00 ± 0.01	1.00 ± 0.03	1.00 ± 0.01	1.00 ± 0.03	1.00 ± 0.01	1.00 ± 0.03
Sklearn	1.00 ± 0.01	1.00 ± 0.03	1.00 ± 0.01	1.00 ± 0.03	1.00 ± 0.01	1.00 ± 0.03
Panel B: Optimized						
Baseline (adaptive tree complexity)	1.00 ± 0.05	1.00 ± 0.14	1.00 ± 0.04	1.00 ± 0.12	1.00 ± 0.05	1.00 ± 0.15
Sklearn (grid search)	0.99 ± 0.05	1.20 ± 0.13	1.00 ± 0.04	1.06 ± 0.12	1.00 ± 0.05	2.20 ± 0.38

Table 8: Baseline on Simulated Data

The table compares the performance (mse) and stability (S_{mse}) of the baseline model to Sklearn's regression tree using the base setup with $r = 1$ and $k = 10$. Panel A compares the models when the tree structure is fixed to *max depth* $= 5$ and *min sample leaf* $= 5$. Panel B compares the trees when searching for the optimal tree structure. The baseline uses the adaptive tree complexity, whereas Sklearn uses a grid search (5-fold cross-validation) over the hyperparameters *max depth* $= [5, 10, \infty]$, *min sample leaf* $= [1, 5]$, and *cost complexity alpha* $= [0, 0.01, 0.1, 0.3]$.

5.2.1.2 Naive Update

Figure 16 shows the performance and stability of `NU` as a function of the data size, n, for the different simulated datasets. `NU` is more stable but performs worse than the baseline across all cases and data sizes. `NU`'s performance and stability improve as n increases. As the figure shows, the improvement of increasing n is largest for smaller values of n, and diminishes as n becomes larger.

The rate of improvement also depends on the complexity of the data, where Case 1 is the simplest and Case 3 is the most complex dataset. The difference in performance between `NU` and the baseline is the smallest for Case 1 and has the smallest stability improvement. For Case 3, it is the opposite: a bigger difference in performance but a larger stability improvement. For all datasets and data sizes, the stability values are in the range $[0.05 - 0.13]$, meaning that `NU` increases stability by 87% to 95%. The figure highlights that the `NU` method drastically improves stability, and the performance reduction of `NU` decreases with increased n. However, as the complexity of the data increases, the performance difference between `NU` and the baseline also tends to increase.

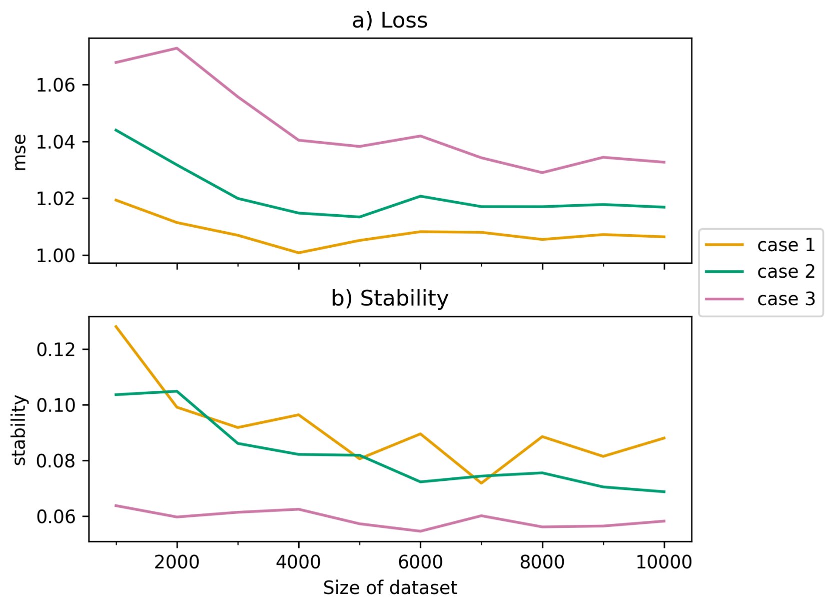


Figure 16: The assumption of the naive update
This figure plots a) the performance (mse) and b) the stability (S_{mse}) as a function of the size of the dataset, n. Each dataset of size n use the base setup. The performance and stability values are normalized relative to the baseline values.

5.2.1.3 Tree Reevaluation

Figure 17 shows how the hyperparameters impact both performance and stability. Larger values of α lead to decreased performance and increased stability and vice versa. The figure shows that the value of α has a larger impact on performance and stability than the value δ as it seems like α sets the location and δ adds variation around the location.

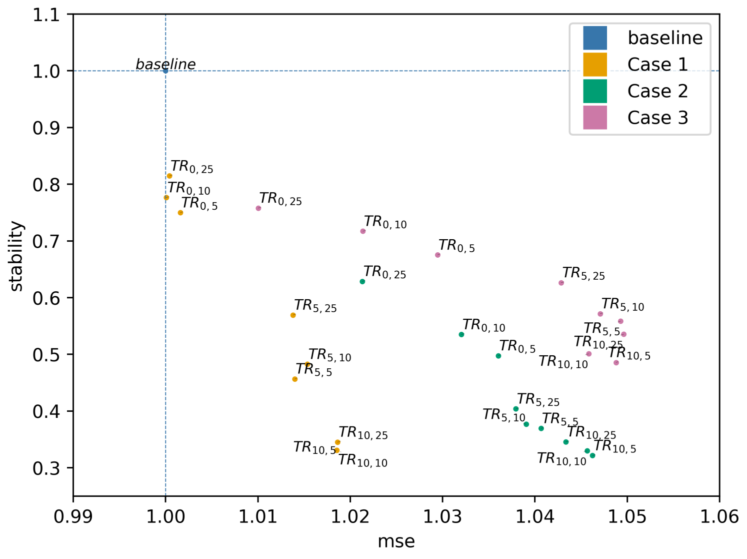


Figure 17: The impact of SL's hyperparameter
The figure shows TR's hyperparameters' impact on performance (mse) and stability (S_{mse}) on the three simulated datasets. Both axes are normalized such that the baseline values are 1. $\texttt{TR}_{\alpha,\delta}$ refers to TR using the hyperparameters α and δ, expressed as percentages. For each dataset, the base setup with $r = 10$ and $k = 5$ is used.

The value of δ determines the confidence level and thus also the trade-off between Type I (larger values of δ) and Type II (lower values of δ) error when deciding whether or not to update parts of the tree. `TR` leads to a better performance compared to `NU` (Figure 16 when $n = 1000$), but `NU` leads to better stability. By allowing for changes in the tree structure, one trades stability for performance.

5.2.1.4 Stable Loss

Figure 18 shows the performance and stability estimates using the `SL` method with different regularization strengths determined by the hyperparameter γ. For Case 1, increasing the value of γ leads to increased stability at the cost of decreased performance. Interestingly, for Case 2 and 3, increasing γ improves both performance and stability up to a certain threshold value ($\gamma < 0.25$ for Case 2 and $\gamma < 0.5$ for Case 3).

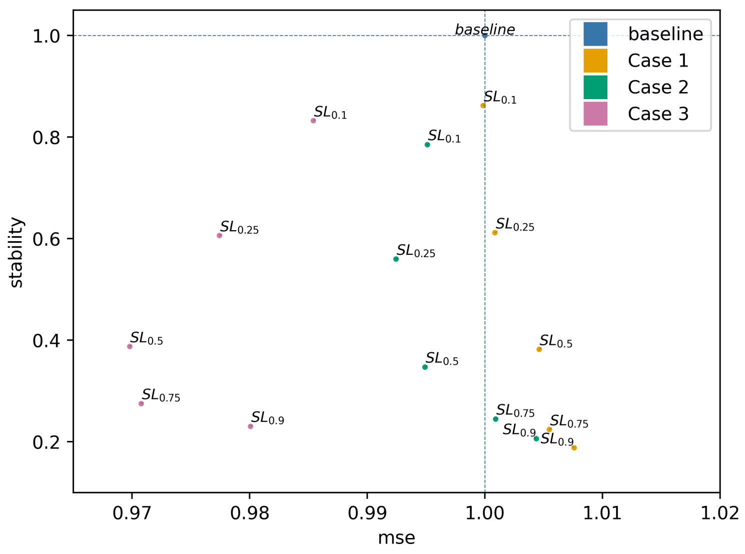


Figure 18: The impact of `SL`'s hyperparameter

The figure shows how `SL`'s hyperparameter γ impacts the performance (mse) and stability (S_{mse}) on the three simulated datasets. Both axes are normalized such that the baseline values are 1. $\texttt{SL}_\gamma$ refers to `SL` using that particular value of γ. For each dataset, the base setup with $r = 10$ and $k = 5$ is used.

5.2.1.5 Approximately Bayesian Update

Figure 19 compares the kernel density estimation (KDE) of the performance (left) and stability (right) obtained by `ABU` to the prior (initial) tree, a (second) tree trained only on $\mathcal{D}_{\Delta_t}$, and the posterior tree it approximates. The posterior tree has the highest performance, followed by `ABU`. The prior tree trained and the second tree have similar performances and are the lowest. The posterior tree has the highest performance as it is trained on $\mathcal{D}_2 = \mathcal{D}_1 \cup \mathcal{D}_{\Delta_t}$. The stability KDEs are the change in predictions of the different trees from the prior tree. The posterior tree is the most stable, followed by `ABU` and the second tree. The posterior tree is more stable than the other as it is trained on $\mathcal{D}_2$, which contains $\mathcal{D}_1$. Both `ABU` and the second tree are given $\mathcal{D}_{\Delta_t}$ as input, but the `ABU` is more stable than the second tree. Note that `ABU` uses an approximation of $\mathcal{D}_1$ to regularize the updating of the tree. The figure illustrates the idea behind `ABU` by showing how `ABU` using only

$\mathcal{D}_{\Delta_t}$ can give a performance and stability closer to a tree trained on $\mathcal{D}_2$, showcasing that ABU approximates the posterior tree.

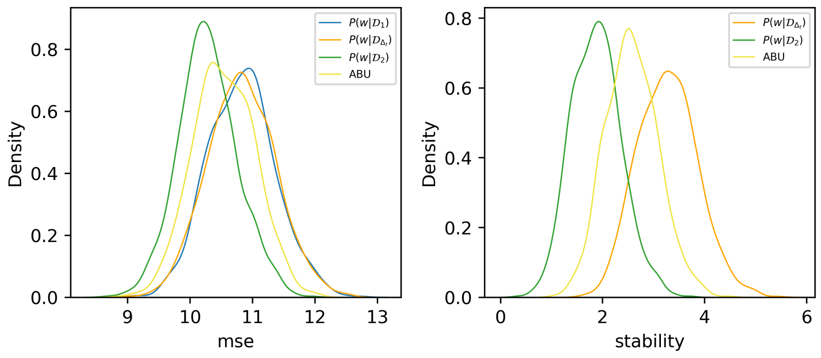


Figure 19: Approximate posterior with ABU

The figure illustrates the property of ABU by comparing the kernel density estimation (KDE) of the mean squared error (left) and stability (right) of ABU to the posterior tree it approximates. The blue line is the prior tree trained on $\mathcal{D}_1$, the orange line is a tree trained only on $\mathcal{D}_{\Delta_t}$, whereas the green line is the posterior tree trained on $\mathcal{D}_2 = \mathcal{D}_1 \cup \mathcal{D}_{\Delta_t}$. ABU (yellow line) approximate $P(w|\mathcal{D}_2)$ with $P(w|\mathcal{D}_1^b \cup \mathcal{D}_{\Delta_t})$ where $\mathcal{D}_1^b$ is sampled from $\mathcal{D}_{\Delta_t}$. The stability KDEs are the change in predictions of the different trees using the prior tree as the initial tree. The Kernel density estimates were computed based on 1000 Monte Carlo simulations using the same data-generating process used for Case 2. The size of entire dataset, $\mathcal{D}_2$, is $n = 1000$ and $\mathcal{D}_1$ and $\mathcal{D}_{\Delta_t}$ are of equal size, i.e., $n_1 = n_{\Delta_t} = 500$.

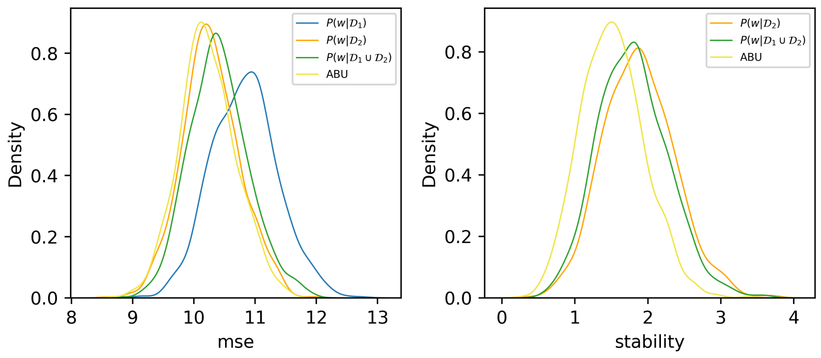


Figure 20: The adaptability of ABU

The figure illustrates the adaptability of ABU by comparing the kernel density estimation (KDE) of the mean squared error (left) and stability (right) of ABU to baseline tree and the posterior tree. The blue line is the prior (initial) tree trained on $\mathcal{D}_1$, the orange line is the initial tree retrained on $\mathcal{D}_2$ (baseline tree), whereas the green line is the posterior tree and is the initial retrained on both $\mathcal{D}_1$ and $\mathcal{D}_2$. ABU (yellow line) approximate $P(w|\mathcal{D}_1 \cup \mathcal{D}_2)$ with $P(w|\mathcal{D}_1^b \cup \mathcal{D}_2)$ where $\mathcal{D}_1^b$ is sampled from $\mathcal{D}_2$. The stability KDEs are the change in predictions of the different trees from the prior tree. The Kernel density estimates were computed based on 1000 Monte Carlo simulations using the same data-generating process used for Case 2. The size of $\mathcal{D}_1$ and $\mathcal{D}_2$ are $n_1 = 500$ and $n_2 = 1000$ where $\mathcal{D}_2$ contains $\mathcal{D}_1$.

Figure 20 compares the KDE of the performance (left) and stability (right) obtained by ABU to the prior tree, the baseline tree, and the posterior tree. Here, the baseline tree and ABU are trained on $\mathcal{D}_2$. The posterior tree, $P(w|\mathcal{D}_1 \cup \mathcal{D}_2)$, represents the posterior tree ABU would approximate without adaptively adjusting the individual γ values. The figure shows that ABU has the highest performance, followed by the baseline tree, the posterior tree, and the prior tree. Also, ABU is the most stable model, followed by the posterior tree and the baseline tree. Due to the posterior tree being trained on $\mathcal{D}_1 \cup \mathcal{D}_2$, it is biased towards the prior tree as $\mathcal{D}_1$ is included twice (i.e., $\mathcal{D}_2$ contains $\mathcal{D}_1$). `ABU` can adaptively adjust the regularization strength, γ, for each of the data points in $\mathcal{D}_1^b$. This adaptability allows `ABU` to have less restrictions on the predictions it is less certain about while having greater restrictions on predictions it is more confident in. `ABU`'s adaptability is highlighted by `ABU` being the most stable while also having the highest performance.

5.2.1.6 BABU

The `BABU` method has a hyperparameter B that determines the number of bumping (or self-learning) iterations. Figure 21 shows how the value of B impacts the performance and stability. It shows similar patterns as Figure 18 showed for `SL`. For Case 1, increasing the value of B leads to greater improvement in stability at the cost of decreased performance. For Case 2 and 3, increasing B improves both performance and stability up to a certain value of B ($B < 7$ for Case 2 and $\gamma < 5$ for Case 3).

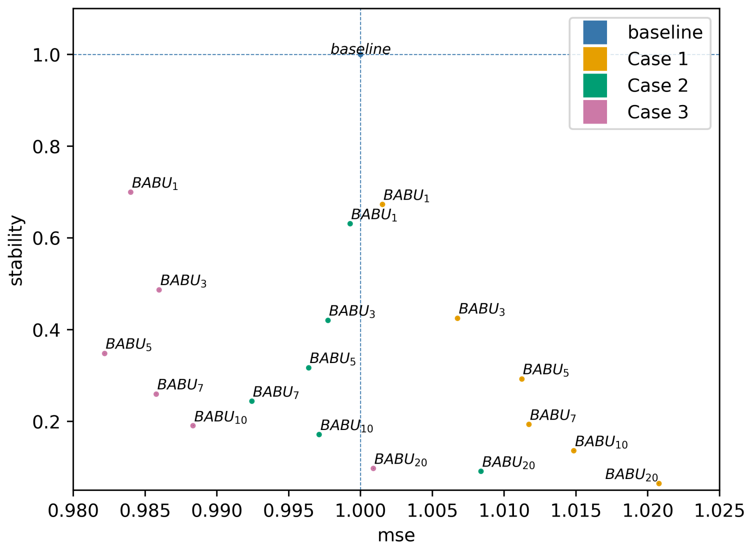


Figure 21: The impact of BABU's hyperparameter

The figure shows how `BABU`'s hyperparameter B impacts the performance (mse) and stability (S_{mse}) using the three simulated datasets. Both axes are normalized such that the baseline values are 1. `BABU`$_B$ refers to `BABU` using that particular value of B, e.g., `BABU`$_1$ means `BABU` with $B = 1$. For each dataset, the base setup with $r = 10$ and $k = 5$ is used.

5.2.1.7 Tracking Model Updates Over Time

Figure 22 shows the different update methods evaluated over multiple update periods on the Case 1 dataset. Each update method provides a more stable update than the baseline across all update periods. The `NU` is consistent across all updates being very stable but with poor performance. `TR` seems to be the poorest method as the other methods have better performance and/or are more stable. For instance, for the first update $t = 1$, swapping `TR` with `NU` will result in a more stable update, and swapping for `ABU` will lead to better performance. Swapping for `SL` or `BABU` will result in both improved performance and a more stable model than `TR`. `BABU` is dominated by `SL` in the first four updates before providing a slightly more stable update but with a slightly worse performance. ABU is closest to the baseline but always provides a more stable update than the baseline and often with a better performance. As t increases, the number of data points also increases, and the difference between SL, ABU, and BABU becomes smaller and smaller. The difference between the baseline and SL, ABU, and BABU also becomes smaller as t increases, which might indicate that they are converging towards the same (or equivalent) predictive function. NU and TR also trend in the same direction, but at a much slower rate, with NU being the slowest.[5]

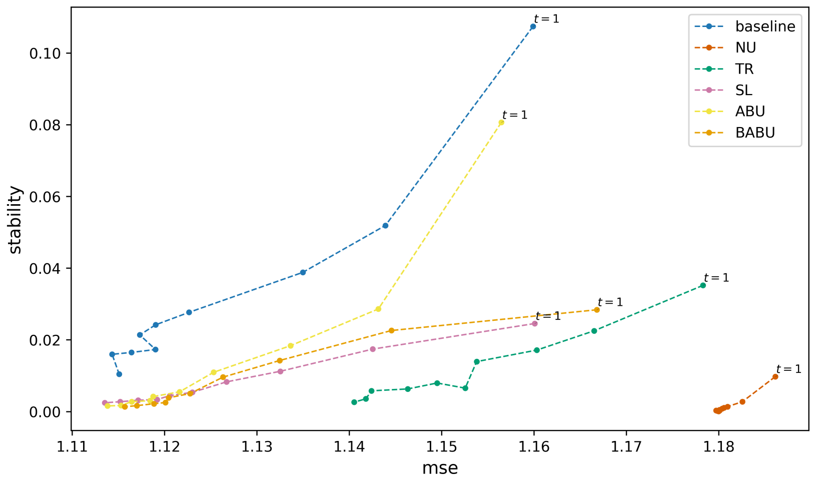


Figure 22: Tracking model updates over time
The figure tracks the performance-stability trade-off of the different model update strategies over ten update periods. Starting with a dataset of 1000 points, 1000 new data points are added to the dataset each update period using the data-generating process of Case 1. The x-axis shows the mean squared error (mse), whereas the y-axis shows the stability measure associated with mse (Equation (18)). Each dot on the lines represents a model update, starting from the first update marked by the dots labeled $t = 1$. Absolute values of performance and stability are reported.

5.2.2 ISLR Datasets

This subsection presents the results of the update methods on the ISLR datasets. The presentation of the results is divided into three parts, one for regression tree, random forest, and GTB, respectively. Each part compares the proposed methods relative to the performance and stability scores of their baseline. Table 9 reports the absolute values of the performance and stability scores, along with their corresponding standard error estimates for the baselines. The random forests do not use adaptive tree complexity for the individual trees as the hyperparameter search showed that the best average performance (i.e., loss) across 10-fold validation is achieved *without* the use of adaptive tree complexity for all the ISLR datasets. The table shows that the baseline for the regression tree has the worst performance and is less stable than the baseline for the random forest and GTB across all datasets. Notably, the baseline tree's stability score for the Boston dataset is significantly higher (15.843) compared to the random forest (1.206) and GTB (1.809). Among the baseline models, the random forest performs the best for three of the five datasets. The random forest tends also to be the most stable as it achieves the

[5]The same experiment was also conducted using simulated Poisson data, and the result can be found in Appendix G.

lowest stability score for four of the five datasets. The baseline GTB model performs the best for the `College` and `Wage` datasets. For the `Wage` dataset, the GTB is also the most stable model of the baselines.

	Regression tree		Random forest		GTB	
dataset	performance	stability	performance	stability	performance	stability
`Boston`	20.786 ± 3.563	15.843 ± 3.138	$\mathbf{13.838 \pm 2.899}$	$\mathbf{1.144 \pm 0.156}$	14.515 ± 2.199	1.809 ± 0.228
`Carseats`	6.115 ± 0.365	2.450 ± 0.313	$\mathbf{5.203 \pm 0.485}$	$\mathbf{0.295 \pm 0.034}$	5.361 ± 0.456	0.397 ± 0.036
`College`	0.055 ± 0.004	0.041 ± 0.004	0.049 ± 0.004	$\mathbf{0.007 \pm 0.000}$	$\mathbf{0.036 \pm 0.003}$	0.008 ± 0.001
`Hitters`	0.283 ± 0.041	0.132 ± 0.02	$\mathbf{0.209 \pm 0.040}$	$\mathbf{0.017 \pm 0.003}$	0.219 ± 0.038	0.041 ± 0.021
`Wage`	0.088 ± 0.004	0.007 ± 0.001	0.084 ± 0.004	0.002 ± 0.000	$\mathbf{0.084 \pm 0.004}$	$\mathbf{0.001 \pm 0.000}$

Table 9: Baseline comparison of the tree-based methods.
The table reports performance (mse) and stability (S_{mse}) scores, along with their standard errors, for the baseline model for each of the tree-based methods on the `ISLR` datasets. A 10-fold cross-validation is used to compute the scores (i.e., the base setup with $r = 1$ and $k = 10$). The values are reported in absolute values. The methods with the best performance and/or stability are highlighted **bold**.

The proceeding subsection reports the performance and stability scores relative to the scores in Table 9, providing insights into the performance and stability of the update methods compared to their respective baselines.

5.2.2.1 Regression Tree

The proposed update methods applied to regression trees are evaluated and compared relative to the baseline tree by plotting their performance and stability estimates and highlighting the Pareto frontier. Figure 23 presents the results for each of the `ISLR` datasets. On all datasets, the proposed methods are more stable than the baseline. Out of the five datasets, the baseline tree is only included in the Pareto frontier once, specifically in the case of the `College` dataset, where it achieves the best performance. The best models tend to be `NU`, `SL`, and `BABU`, as these methods are included on the Pareto frontier for all the datasets. NU is the most stable on four of the five datasets but with the worst performance. Different Pareto efficient solutions can be achieved with `SL` and `BABU` by tuning their respective hyperparameters γ and B. For instance, `SL` are heavily represented on the Pareto frontier for the `Carseats`, `College`, and `Wage` dataset, whereas the `BABU` models work particularly well on the `Boston` and `Hitters` dataset. `ABU` is included in the Pareto frontier once (`Carseats` dataset), where it achieves the best performance. `ABU` achieve similar performance and stability scores as $\texttt{SL}_{0.1}$ and $\texttt{BABU}_1$ but is often dominated by one of them. The `TR` method is always inferior to the other methods.

The results show that both `NU` and `TR` improve stability compared to the baseline but that the stability improvement comes at the cost of worse performance. In contrast, `SL`, `ABU`, and `BABU` can achieve similar or better performance compared to the baseline while improving stability. Any performance improvement of `SL` and `ABU` is typically small (0.5% to 2%), whereas the performance improvement of `BABU` can be rather large. For instance, on `Boston` dataset, $\texttt{BABU}_5$ improved performance by approximately 20% while being 40% more stable than the baseline. Similarly, for `Hitters` dataset where $\texttt{BABU}_3$ improved performance by approximately 10% while also improving stability by roughly 35%. Furthermore, one observes that typically increasing γ and B leads to increased stability and decreased performance, but in many cases, increasing γ and B also improves performance up to a certain value.

This is similar to patterns observed for `SL` (Figure 18) and `BABU` (Figure 21) on the simulated data.

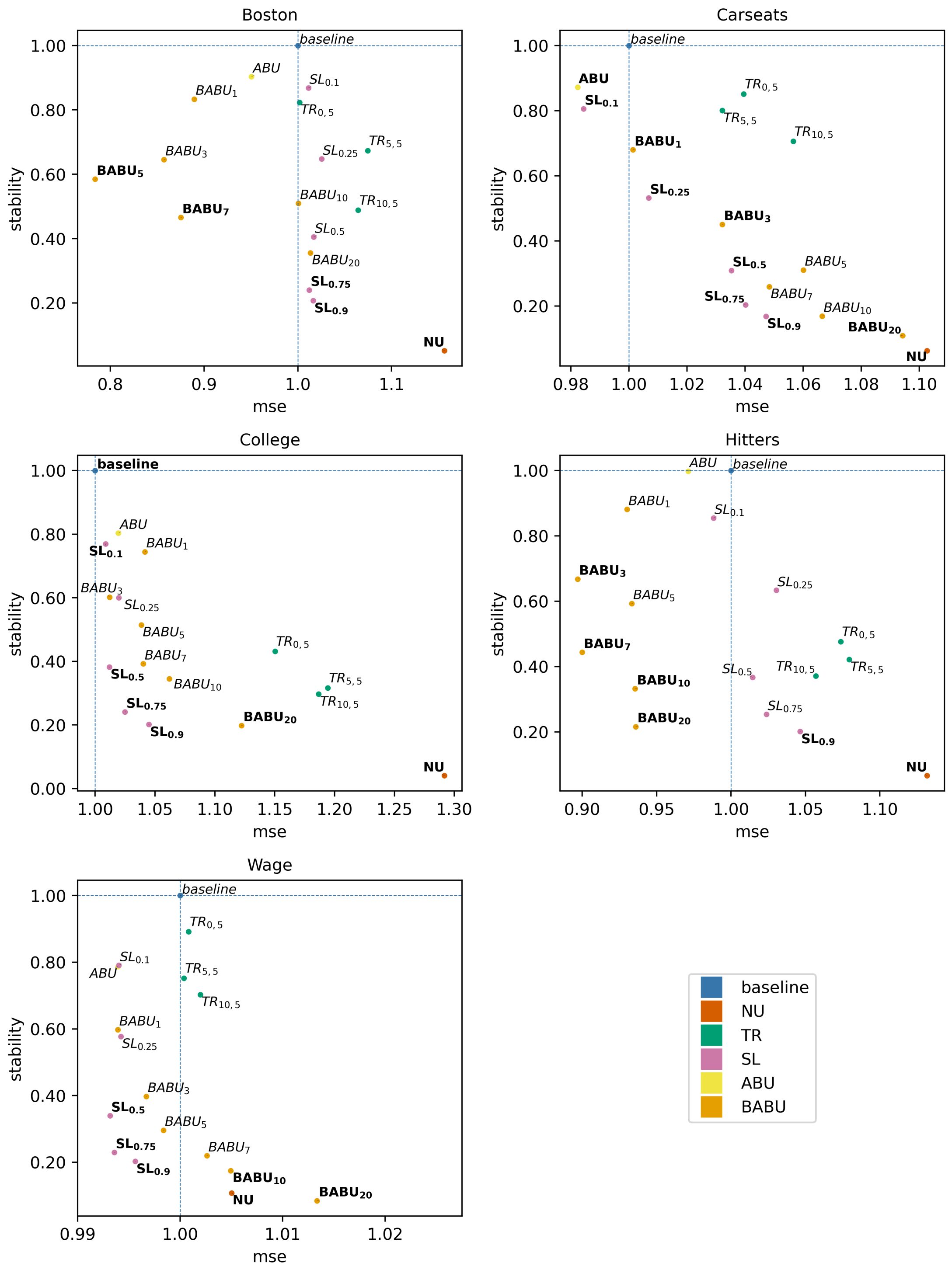


Figure 23: Pareto frontier for regression tree on the ISLR datasets
The figure plots the performance (mse) and stability (S_{mse}) of the proposed models relative to the baseline and highlights the models that form the Pareto frontier in **Bold**.

5.2.2.2 Random Forest

Figure 24 shows the performance and stability scores of the proposed update methods applied to random forests relative to the baseline random forest using the `ISLR` datasets. Across the datasets, all the proposed update methods are more stable compared to the baseline, except for $\texttt{TR}_{0,5}$ on the `Boston` dataset. The baseline is included in the Pareto frontier twice, for the `Carseats` and `College` where it achieves the best performance. One or two of the five `SL` models are on the Pareto frontier for three of the five datasets, which makes `SL` less representative compared to the method applied to regression trees. The best models are mainly `BABU` with different values of B. One or more of the three `BABU` models are on the Pareto frontier for all five datasets, with $\texttt{BABU}_5$ always being represented. Interestingly, `NU` is on the Pareto frontier only once (`College` dataset), as other methods tend to be more stable, which deviates from the observations for regression trees. `ABU` is also included on the Pareto frontier once (`Hitters` dataset). `ABU` is essentially `BABU` with $B = 0$ and is dominated by the other `BABU` models with $B > 0$ for three of the five datasets. The `TR` methods applied to regression trees are consistently inferior to the other methods. This is not the case for random forests as $\texttt{TR}_{10,5}$ is on the Pareto frontier for the `Carseats` and `College` dataset.

Like regression trees, `SL`, `ABU`, and `BABU` applied to random forests can achieve similar or better performance compared to the baseline while improving stability. However, achieving any performance improvement tends to be more difficult and is rather modest (0-1%) compared to regression trees. Additionally, increasing γ and B can improve stability and performance up to a certain threshold value as observed for `BABU` and `SL` on the `Boston` and `Wage` datasets, respectively. For the `Wage` dataset, this threshold value is notably large for `SL` where performance and stability improve as γ increases up $\gamma = 0.75$. Beyond this threshold, increasing γ further increases stability but decreases performance, as shown by $\texttt{SL}_{0.9}$ being more stable but with a slightly lower performance. Another notable observation for `Wage` is the behavior of `NU`. Typically, `NU` tends to be very stable but with lower performance than other methods. However, for `Wage` dataset, `NU` has higher performance compared to other models while still maintaining a relatively high level of stability.

5.2.2.3 Gradient Tree Boosting

Figure 25 displays the performance and stability scores of the `SL` update method applied to GTB relative to the baseline GTB for the `ISLR` datasets. All the `SL` models provide a more stable model than the baseline. The baseline is included in the Pareto frontier for four of the five datasets, where it achieves the best performance. Different Pareto efficient solutions can be achieved with `SL` by tuning γ. `SL` applied to GTB show the same general trend as `SL` for regression trees and random forests. With increasing γ, stability increases and is generally at the cost of performance. For the `Hitters` dataset increasing γ from 0 (baseline) to 0.1 increases both stability and performance, which is consistent with $\texttt{SL}_{0.1}$ applied to regression tree and random forest for the same dataset. An anomaly is observed for $\texttt{SL}_{0.5}$ on the `Boston` dataset where increases γ from 0.25 to 0.5 leads to decreased performance and stability. In general, $\texttt{SL}_{0.1}$ improves stability by 5-15%, whereas $\texttt{SL}_{0.75}$ and $\texttt{SL}_{0.9}$ improves stability by 30-60%. The rate of stability improvement tends to diminish as γ increases. Like the observation for random forest, achieving better performance with `SL` than the baseline GTB is difficult and is only achieved once.

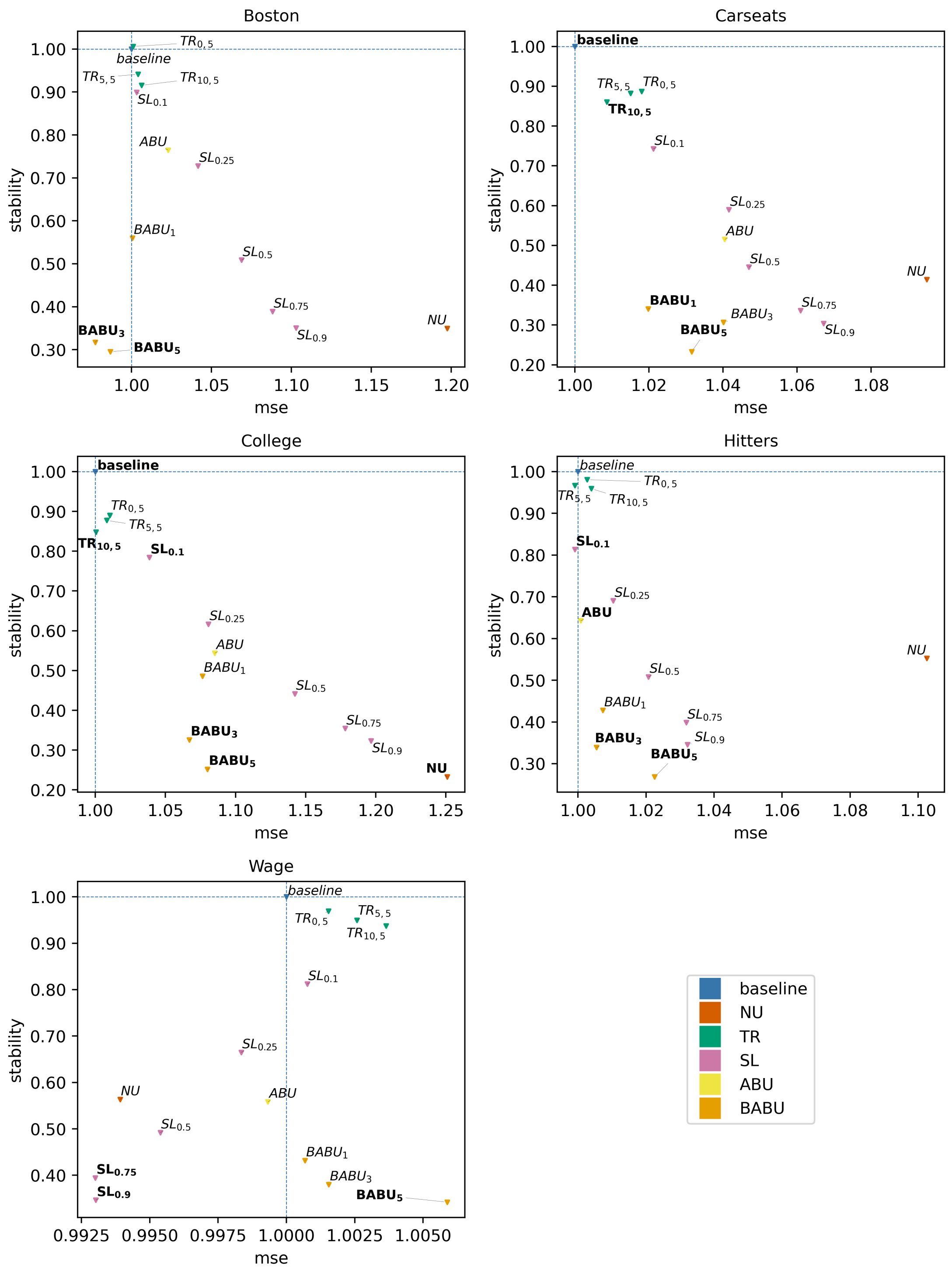


Figure 24: Pareto frontier for random forest on the ISLR datasets
The figure plots the performance (mse) and stability (S_{mse}) of the proposed models relative to the baseline and highlights the models that form the Pareto frontier in **Bold**.

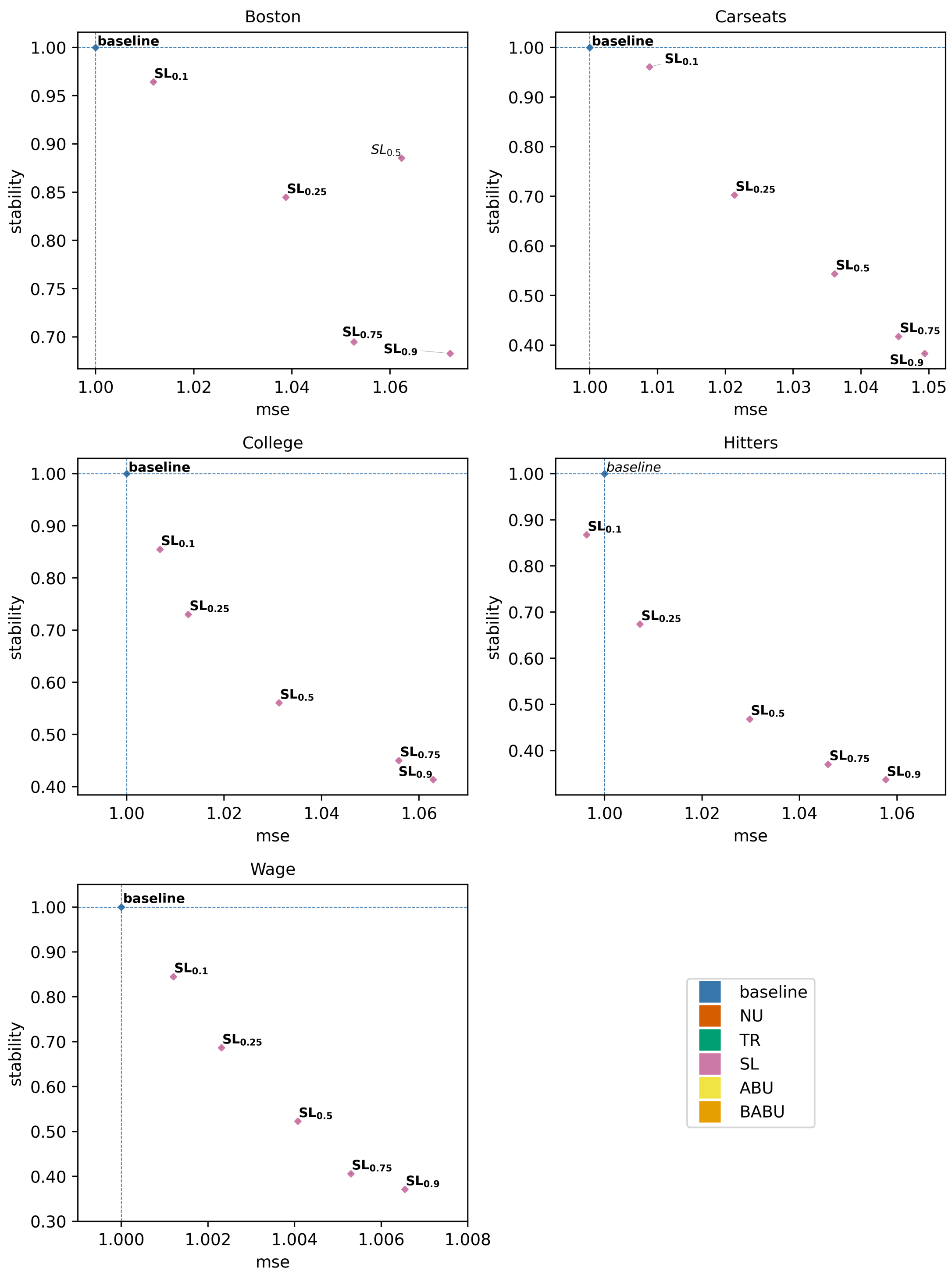


Figure 25: Pareto frontier for GTB on the ISLR datasets
The figure plots the performance (mse) and stability (S_{mse}) of the proposed models relative to the baseline and highlights the models that form the Pareto frontier in **Bold**.

5.2.3 Claim Frequency Estimation

Figure 26 compares the performance and stability of the proposed updates methods applied regression trees, random forest, and GTB for claim frequency estimation. Random forests use adaptive tree complexity as the hyperparameter search showed that the best average performance across 6-fold validation is achieved *with* the use of adaptive tree complexity. The Pareto frontier consists of the GLM model, the baseline GTB, and SL applied to GTB. The GLM is the most stable of the model by a large margin, but it also has the poorest performance. The GTB models are the top-performing models, with the baseline GTB having the best performance. SL method applied to GTB trades stability at the cost of decreased performance. The trade-off is larger for increasing γ. For instance, $\texttt{SL}_{0.9}$ increases stability by 41.66% while reducing performance by 0.06% compared to the baseline GTB. An anomaly is detected for $\gamma = 0.75$ and is the only SL applied to GTB that is not on the Pareto frontier. This is similar to what was observed for $\texttt{SL}_{0.5}$ applied to GTB for the Boston dataset. A closer examination of the stability score at the different folds reveals an extremely larger stability score for a single fold, almost five times larger than the other fold scores. If this fold is excluded, the average stability score becomes 0.0018, moving $\texttt{SL}_{0.75}$ applied to GTB down to the rest of GTB models.

Furthermore, Figure 26 shows increasing γ for SL leads to increased stability and decreased performance for GTB. For regression trees and random forests, increasing γ also improves performance up to $\gamma = 0.25$. This trend can also be observed for BABU applied to regression trees. Here, increasing B improves stability and performance up to $B = 3$. Surprisingly, BABU applied random forest deviates the trend, as increasing B results in worse stability. This contradicts the previous results for BABU where increased B results in improved stability and sometimes also improved performance. Additionally, all the update method applied to trees improves stability compared to the baseline tree, except for $\texttt{TR}_{0,5}$, which performs better but are less stable. The update methods applied to random forest improve stability compared to the baseline forest except for BABU with $B \geq 3$. Notably, TR is inferior to the other methods for regression trees and when applied to random forest. Additionally, one can observe that ABU and BABU applied to trees improve stability more than SL compared to the results from the ISLR datasets. One also observes that $\texttt{BABU}_{20}$ and NU for trees are more stable than any of the random forest methods, though with worse performance.

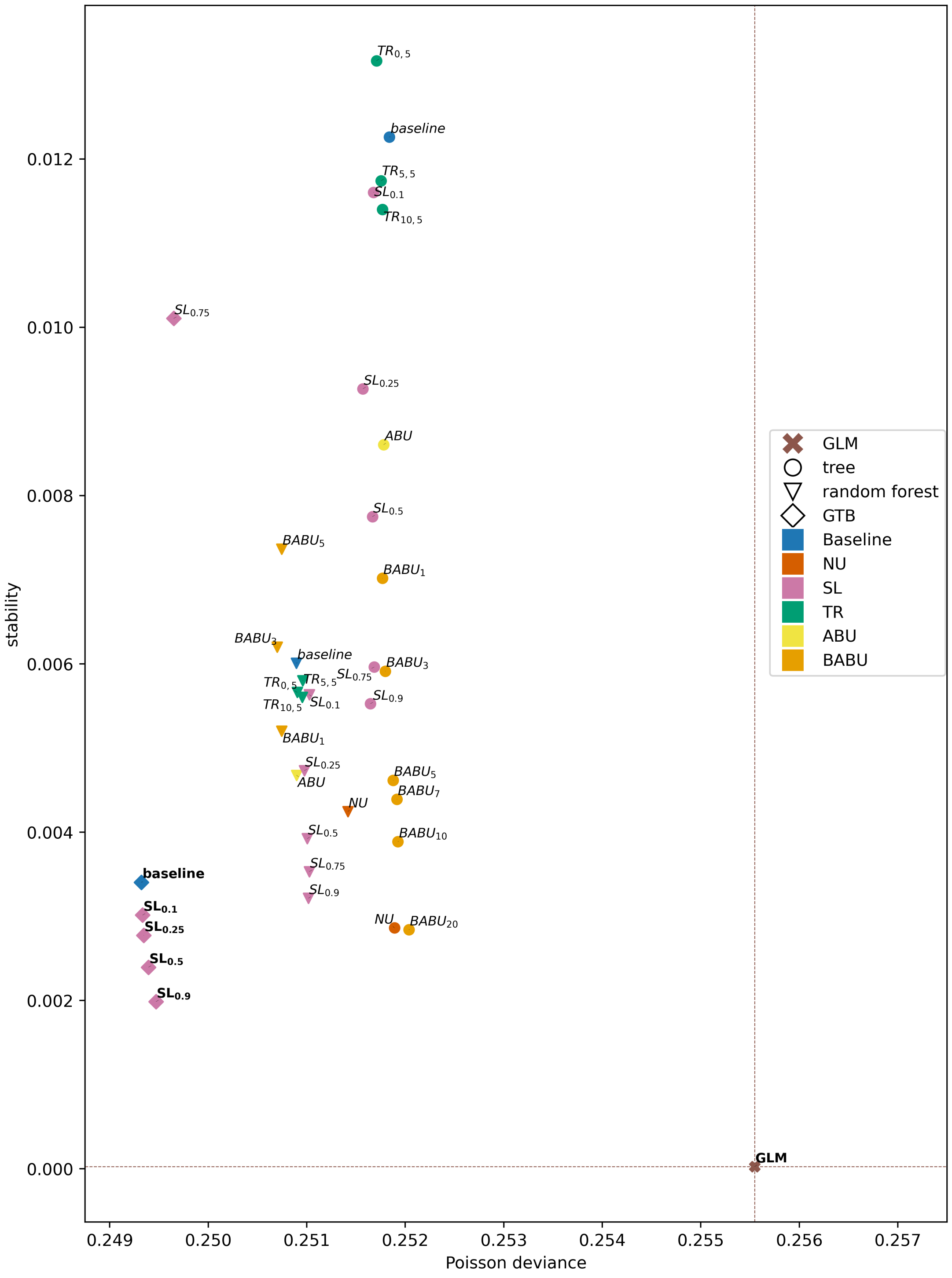


Figure 26: Pareto frontier for the `freMTPLfreq` dataset
The figure plots performance (mean Poisson deviance) and stability (S_{sdlr}) (in absolute values) of the proposed models for regression tree, random forest, and gradient tree boosting (GTB). The different proposed update methods are color-coded, with the method applied to the regression tree marked with a circle. The methods applied to the random forest are marked with a triangle, while the methods applied to the GTB are marked with a diamond. A GLM model, marked with a cross, is also included as a reference model. The models highlighted in **bold** represent the Pareto frontier.

6 Discussion

This section interprets and analyzes the results presented in the preceding section, offering insights into the significance, implications, and limitations of the proposed update methods. The remainder of this section is organized as follows: A discussion of the main findings, followed by a reflection on the strengths and limitations of the thesis. Next, potential future research directions are discussed, and finally, the thesis is concluded.

6.1 Main Findings

6.1.1 Regression Tree

The first objective of this thesis was to provide stable update methods for regression trees. With that in mind, I developed five update methods for regression trees, namely `NU`, `TR`, `SL`, `ABU`, and `BABU`. The methods for regression trees are thoroughly tested using simulated data. The result of the simulated experiments shows that all methods improve stability. `NU` drastically improves stability up to between 87% and 95% but at the cost of decreased performance. The result of the `NU` experiment shows that as the data size n increases, the performance reduction decreases while stability increases or stays the same. The decreased performance reduction suggests that the initial tree structure approaches the optimal tree structure, aligning with the expectation from the law of large numbers, $\lim_{n \to \infty} \hat{f} \to f^*$. The same effect is expected to apply to the other methods, meaning that the baseline and the update methods should converge to the same function, most likely with different rates. The `TR` extends `NU`, allowing for parts of the tree to change if one is confident that the change will improve performance, mitigating some of the performance reduction of `NU` at the cost of a smaller stability improvement. The result for `TR` shows that the performance-stability trade-off is mainly impacted by its α and should be prioritized when tuning hyperparameters. Increasing α decreases performance and increases stability and vice versa. While both `NU` and `TR` improve stability at the cost of decreased performance compared to the baseline, `SL`, `ABU`, and `BABU` can achieve similar or better performance than the baseline while also improving stability. `SL` and `BABU` each have a hyperparameter, γ and B, respectively. The simulated data show that increasing γ and B increases stability and generally decreases performance. However, there are cases where increasing γ and B also improves performance up to a certain value. The result of the `ABU` experiment highlights `ABU`'s properties as `ABU` can improve both performance and stability compared to the baseline given the same dataset as input. This is achieved by approximating $\mathcal{D}_1$ with $\mathcal{D}_1^b$ and adaptively adjusting the regularization strength for each data point based on its confidence. An evaluation of the long-term behavior of the methods shows that the difference between `SL`, `ABU`, and `BABU` becomes smaller and smaller, which suggests that these methods converge towards the same function as the number of data point increases. Furthermore, the difference between these methods and the baseline also becomes smaller, indicating that the baseline converges towards the same function but at a slower rate. Both `NU` and `TR` also trend in the same direction, but at a much slower rate, with `NU` being the slowest. The observations are aligned with the expectation that the baseline and the update methods should converge to the same function as $n \to \infty$ but with different rates.

Similar results were observed when the update methods were evaluated on the ISLR datasets. All update methods improve stability compared to the baseline, with NU, SL, and BABU being the best models. NU is on four of the five datasets the most stable but has the poorest performance, whereas SL and BABU can achieve similar or better performance as the baseline while being more stable by tuning their respective hyperparameters γ and B. ABU also achieves similar or better performance as the baseline while being more stable. These results are consistent with the observation from the simulated data. However, for the ISLR datasets, either SL or BABU often provides a better solution than ABU. The results of the proposed methods on the ISLR datasets validate the observations from simulated data, demonstrating their practical relevance across a broad range of data characteristics. These results strongly support the achievement of the first objective of providing stable update methods for regression trees.

Even though the methods can provide a more stable update, it is not obvious which one to choose. The choice depends on the preference of the performance-stability trade-off of the user or stakeholder. A conservative user might want a more stable model and therefore opt for NU, or SL and BABU with high γ and B values. On the other hand, a user primarily concerned with performance would not opt for NU, but rather for SL and BABU with lower γ and B values, respectively. In some cases, the baseline tree and ABU could be preferable. The hyperparameters of SL and BABU allow the users to include their performance-stability preference by decreasing or increasing γ and B. The choice of method (and hyperparameters) may vary depending on the dataset, user preference, and application domain. This means that the model selection and choice of performance measure that balances performance and stability are crucial for the application of the stable tree updates.

Since SL, ABU, and BABU can often achieve better performance compared to the baseline, it suggests that these methods often are able to strike a better balance between bias and variance than the baseline. The stability regularization of these methods imposes an assumption that the model should not change much, which increases bias and decreases variance. Since regression trees are known to have high variance, variance reduction can often improve performance. NU and TR also provide variance reduction. However, the decreased performance suggests that the assumptions of these methods are too strict, reducing variance too much. Another contributing factor is that SL, ABU, and BABU incorporate stability regularization in the loss function (i.e., explicit regularization), whereas NU and TR do not (i.e., implicit regularization). The advantage of explicit stability regularization over implicit regularization is that ensures that also stability is taken into account when adaptively determining the tree's complexity. From the ISLR datasets, one observed that any performance improvement of SL and ABU is typically small, whereas the performance improvement of BABU can be rather large, as shown for the Boston and the Hitters dataset. This suggests for these datasets, the greedy splitting procedure of CART finds a local minimum and that BABU is able to improve the function search using the ideas from bumping and semi-supervised learning. BABU were not able to improve performance for Carseats and College datasets which is consistent with the observation of Chapelle et al. [2006] that there is no guarantee that semi-supervised learning will improve performance.

6.1.2 Random Forest and GTB

Moving on to the second objective to show that these methods can be extended to provide stable update methods for random forest and GTB. All the update methods were extended to random forests, whereas only `SL` were extended to GTB. The methods were evaluated on the `ISLR` datasets. The results of the update methods applied to random forests are similar to the result for regression trees. All update methods improve stability compared to the baseline (except for $\texttt{TR}_{0,5}$ on the `Boston` dataset). The best models can be achieved with `SL` and `BABU` by tuning their respective hyperparameters γ and B. Notably, `BABU` tends to provide better solutions compared to `SL`. Similar to regression trees, `SL` and `BABU` applied to random forests also result in more stable models compared to the baseline. Although `SL` and `BABU` for random forest also can achieve similar or better performance as the baseline while being more stable, it tends to be more difficult compared to regression trees. The difficulty of the update method to improve performance in random forests compared to single regression trees is most likely due to random forests having less variance, which reduces the impact of any variance reduction provided by the stability regularization on performance.

Although the result of the methods applied to random forests show similar results as regression trees, there are some notable differences. The first notable difference is `NU` only appearing on the Pareto frontier once. Unlike regression trees, `NU` applied to random forests often do not result in the most stable model.[6] `BABU` or `SL` with $\gamma > 0.5$ are often more stable than `NU`. `BABU` and `SL` incorporated stability into the calculation of w which is not the case for `NU`. Also, `NU` keeps the structure fixed, therefore the trees in the `NU` forest will be less complex compared to `BABU` and `SL`. The result suggests that incorporating stability into the calculation of w and aggregating more complex trees result in a more stable forest than keeping the individual tree structures fixed and only updating their estimates. Moreover, `NU` improves performance compared to the baseline for the `Wage` dataset. This may be explained by `NU` keeping the structure fixed which is essentially a way of controlling the tree depth when updating the forest. Controlling the tree depth has been shown to result in a small gain in performance (Segal [2003]). For the `Wage` dataset, one also observed that for `SL`, increasing γ improves both performance and stability up to very large values of γ. The result of `NU` and `SL` suggest that the regularization effect of the update methods that heavily prioritize stability in this case also leads to the highest performance. However, this explanation is contradicted by the fact that increasing B for `BABU` leads to increased stability but decreased performance. It is worth noting that the spread of the relative performance between the methods is very small, ranging from 0.9925 to 1.0025.

Regarding `SL` applied to GTB, it shows the same general trend as `SL` applied to regression trees and random forests. Increasing the hyperparameter γ generally increases stability at the expense of performance, but in some cases, it also improves performance up to a certain value as illustrated by $\texttt{SL}_{0.1}$ on the `Hitters` dataset. This is consistent with $\texttt{SL}_{0.1}$ applied to the regression tree and random forest for the same dataset. Like random forest, achieving similar or better performance with `SL` as the baseline GTB is harder as GTB can reduce both bias and

[6]This was initially hypothesized to be due to not tracking bootstrap indices, causing more variation in the leaf predictions. However, tests showed similar results even when tracking bootstrap indices, suggesting it has little impact on the behavior of `NU`.

variance, reducing the impact of any variance reduction provided by the stability regularization on performance.

The methods applied to random forests and GTB show similar results as for regression trees, confirming that these methods can be extended to provide stable update methods for random forests and GTB. However, the observations for `NU` for random forests indicate that the aggregation of trees may affect the properties of the methods and should be investigated further. Additionally, an anomaly for GTB was detected on the `Boston` dataset where increasing γ from 0.25 to 0.5 leads to increased loss and less stability, which calls for a thorough examination of how any modification of the loss function could affect the `aGTBoost` algorithm.

6.1.3 Claim Frequency Estimation

Proceeding to the third and last objective, the goal is to show that the stable update methods can provide more stable tree-based models for claims frequency estimation compared to the baseline strategy. The update methods applied to regression trees, random forests, and GTB were evaluated for the task of claim frequency estimations using the `freMTPLfreq` dataset. The results show that the tree-based models achieve better performance compared to the GLM, with GTB models being the best performing, followed by random forests and regression trees. This is partially consistent with Henckaerts et al. [2020], which found that GTB outperforms GLM but that GLM performs better than random forests and regression trees using a MTPL insurance portfolio from the Belgium insurance industry. The results of the tree-based models outperforming GLM suggest that the tree-based models are able to find interaction effects and non-linearities that are not included in the GLM manually. However, the results show that the GLM is still more stable than the tree-based models, confirming the stability issue the insurance industry has experienced. The Pareto frontier consists of the GLM model, the baseline GTB, and `SL` applied to GTB (except $\texttt{SL}_{0.75}$). Baseline GTB performs best but is the least stable model of the frontier models. Oppositely, GLM is the most stable model but performs the worst. A more stable GTB can be achieved at the cost of some performance by applying `SL` but not nearly the stability of GLM, regardless of the value of γ. Notably, $\texttt{SL}_{0.9}$ improves the stability of GTB by 41.66% while only reducing the performance by 0.06%. This demonstrates a substantial improvement in stability at a negligible cost in performance. Despite this stability improvement, GLM is drastically more stable than $\texttt{SL}_{0.9}$, with $S_{sdlv} = 2.636 \times 10^{-5}$ compared to $S_{sdlv} = 1.986 \times 10^{-3}$. Whether the performance boost justifies the trade-off in stability when considering switching from GLM to one of the GTB models is an open question that requires further extensive analysis beyond the scope of this thesis and is up to the domain experts to determine.

The result of claim frequency estimation shows that the results using squared error loss are transferable to Poisson loss. This demonstrates that the stable update methods can be used to provide more stable tree-based models for claims frequency estimation, achieving the third and last objectives. For Poisson loss, one observes that `ABU` and `BABU` tend to be more stable than `SL` for regression trees. This is most likely due to the use of different regularization terms in the Poisson loss for `SL` and `ABU`/`BABU`. For squared error, all three methods use the same regularization term, the squared error (i.e., L2 regularization). However, for Poisson loss, `SL` uses the Poisson loss as the regularization term, whereas `ABU` and BABU use L2 regularization. The methods applied to random forests and GTB showed some irregularities for the ISLR datasets. These irregularities are also found for the freMTPLfreq dataset. Furthermore, the behavior of BABU for random forests on the freMTPLfreq dataset stands out, as increasing B decreases stability, highlighting an additional irregularity. These irregularities further stress the need for a thorough investigation of the properties of the methods applied to random forests and GTB.

6.2 Strengths and Limitations

The thesis provides innovative and novel approaches for updating regression trees in a stable manner by exploring the statistical properties of trees. To the best of my knowledge, there is no other scientific work that specifically investigates methods for improving the stability of updating tree-based methods for regression tasks, neither in general nor for claim frequency estimation. This highlights the originality and novelty of this thesis. The thesis introduces a new notion of stability, update stability, that differs from the traditional notions of stability as defined by previous studies (Devroye and Wagner [1979]; Kearns and Ron [1999]; Bousquet and Elisseeff [2002]). Traditional stability measures the sensitivity of algorithm predictions to the addition of a new data point and aims to determine how the variance of a learning algorithm affects its generalization error. In contrast, updated stability is a performance measure that determines how much predictions change when updating the model using additional data. Update stability also differs from semantic stability by Turney [1995], as semantic stability is specific to classification algorithms, measuring a classifier's ability to assign examples to the same class when trained on different subsets of the data. Regarding the use of tree-based methods for claim frequency estimation, Guelman [2012], Liu et al. [2014], and Henckaerts et al. [2020] all found GTB to outperform GLM, but none of them considered the stability aspect. Last et al. [2002] improved the semantic stability of decision trees while preserving a reasonable level of predictive accuracy using an Info-Fuzzy Network approach. This is similar to the results of this thesis which finds the update methods for regression trees to be more stable than the baseline, with some methods achieving similar or better performance. However, the results are not directly comparable as they use a different stability notion (semantic) and focus on initial model learning rather than updating with additional data.

A major strength of the thesis is the extensive validation of the update methods for regression trees. The methods were thoroughly tested on simulated data and further validated across a broad range of data characteristics, including claim data. The extensive validation strengthens the credibility of the results for the proposed update methods applied to regression trees.

The methods are developed for regression trees but are also extended to random forests and GTB under the assumption that they will work similarly. The properties of the proposed update methods applied to random forests and GTB are not tested to the same degree as regression trees. The lack of extensive testing on simulated data for random forests and GTB leaves a greater uncertainty about the results of the proposed method extended to random forests and GTB. Irregularities found in the methods applied for random forests suggest that further investigation of how the aggregation of trees may affect the properties of NU and BABU is needed.

Another limitation of the thesis is the limited hyperparameter tuning for the random forest. Random forest in this thesis uses only 100 trees and other default hyperparameters. The only hyperparameter search conducted is to determine whether or not the use of adaptive tree complexity for the individual trees in the forest. The limited hyperparameter tuning for the random forest makes it less competitive with GTB, which finds the tree complexity and the number of trees adaptively. The GLM model used for claim frequency estimation is based on the book of Charpentier [2014]. Perhaps further feature engineering and data exploration could improve the performance of GLM.

6.3 Future Work

In light of the findings and limitations discussed, there are several directions in which the thesis could be improved in feature research. As some irregularities are found in the methods applied to random forests, further studies should investigate these irregularities to improve the methods for random forests. Additionally, various tailor-made methods for random forests, such as stacking, tree replacement, and co-training, were considered but not included due to time constraints and would be interesting to explore in future work. Stacking involves assigning weights to individual trees in the forest using gradient descent and explicit stability regularization, whereas tree replacement involves replacing a subset of trees in the forest with new trees when new data becomes available. Co-training might improve `BABU` for random forests. For the random forests, it might make more sense to use the average predictions of all the trees in the forest as pseudo-labels (co-training) rather than each tree creating its own pseudo-labels (self-training).

`ABU` for GTB was not implemented due to the statistical challenges caused by the cross-correlation between trees. Solving this issue would be a step toward making a Bayesian approach to GTB. An interesting extension of the update methods for regression trees would be the use evolutionary algorithm to create evolutionary trees. Evolutionary trees are an alternative to the greedy approach of CART and enable the use of a global loss which can help avoid getting stuck in local optimum.[7]

This thesis assumes no concept drift, a future extension could be to investigate how the methods would handle concept drift. This exploration would provide valuable insights into the applicability in real-world scenarios where the underlying data-generating process can change over time.

Another direction could be to perform an extensive analysis of the impact of the methods on the premium portfolio. Such an analysis would help determine whether the stability improvement provided by the update methods can justify the choice of replacing GLM with tree-based methods. A natural extension of the thesis is to include zero-inflated Poisson loss, which is better suited to handle claims data mainly consisting of non-claims observation (which is the case for the `freMTLPfreq` dataset).

[7] An implementation of an evolutionary update approach was almost completely implemented but put on hold as other methods were prioritized. A description of the work is provided in Appendix H.

6.4 Conclusion

Motivated by the instability of tree-based methods experienced by the insurance industry, this thesis provides innovative and novel methods for updating regression trees in a stable manner. All methods are shown to increase stability with some methods increasing both stability and performance compared to the baseline. These methods can be extended to random forests and GTB and show similar results as for regression trees. However, some of the methods extended to random forests show some irregularities, which should be investigated further. Furthermore, this thesis demonstrates the potential of the stable update methods in improving claims frequency estimation in the insurance industry but further analysis is required to determine their impact on the premium portfolio.

References

Amini, M.-R., V. Feofanov, L. Pauletto, E. Devijver, and Y. Maximov (2022, 2). Self-Training: A Survey.

Bayes, T. (1763, 1). LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S. *Philosophical Transactions of the Royal Society of London 53*, 370–418.

Bousquet, O., S. Boucheron, G. Lugosi, U. Luxburg, and G. Rätsch (2003, 1). Introduction to Statistical Learning Theory. *Advanced Lectures on Machine Learning, 169-207 (2004).*

Bousquet, O. and A. Elisseeff (2002, 3). Stability and Generalization. *J. Mach. Learn. Res. 2*, 499–526.

Breiman, L. (1996). Bagging predictors. *Machine Learning 24*(2), 123–140.

Breiman, L. (2001). Random Forests. *Machine Learning 45*(1), 5–32.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone (1984). *Classification and Regression Trees* (1st Edition ed.). New York.

Chapelle, O., B. Scholkopf, and E. A. Zien (2006). *Semi-Supervised Learning* (1st ed. ed.), Volume b. Cambridge: The MIT Press.

Charpentier, A. (2014). *Computational Actuarial Science with R* (1st Edition ed.). New York: Chapman and Hall.

Chen, T. and C. Guestrin (2016, 3). XGBoost: A Scalable Tree Boosting System. *arXiv*.

Cox, J. C., J. E. Ingersoll, and S. A. Ross (1985). A Theory of the Term Structure of Interest Rates. *Econometrica 53*(2), 385–407.

Devroye, L. and T. Wagner (1979). Distribution-free performance bounds for potential function rules. *IEEE Transactions on Information Theory 25*(5), 601–604.

Dionne, G., C. Gourieroux, and C. Vanasse (1998, 2). Evidence of Adverse Selection in Automobile Insurance Markets. In *Automobile Insurance: Road Safety, New Drivers, Risks, Insurance Fraud and Regulation*, pp. 13–46.

Donsker, M. D. (1951). An invariance principle for certain probability limit theorems. *Memoirs of the American Mathematical Society*, 12–12.

Dutang, C. and A. Charpentier (2018). CASdatasets R package vignette. Reference manual. Version 1.0-8. Technical report, packaged 2018-05-20.

Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics 7*(1), 1 26.

Ferrario, A., A. Noll, and M. V. Wüthrich (2018). Insights from Inside Neural Networks. Technical report.

Friedman, J. (2002, 2). Stochastic Gradient Boosting. *Computational Statistics & Data Analysis 38*, 367–378.

Friedman, J., T. Hastie, and R. Tibshirani (2000, 4). Additive Logistic Regression: A Statistical View of Boosting. *The Annals of Statistics 28*, 337–407.

Friedman, J. H. (2001, 10). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics 29*(5), 1189–1232.

Gareth James, Trevor Hastie Robert, Tibshirani, and Daniela Witten (2013). *An introduction to statistical learning with applications in R.* NewYork: New York : Springer, 2013.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning.* MIT Press.

Grosse, R. (2021). Chapter 4 Second-Order Optimization. Technical report, University of Toronto, Department of Computer Science, Toronto.

Guelman, L. (2012). Gradient boosting trees for auto insurance loss cost modeling and prediction. *Expert Systems with Applications 39*(3), 3659–3667.

Hastie, T. and R. Tibshirani (1986). Generalized Additive Models. *Statistical Science 1*(3), 297–310.

Hastie, T., R. Tibshirani, and J. Friedman (2001). *The Elements of Statistical Learning.* New York, NY: Springer New York.

Henckaerts, R., M.-P. Côté, K. Antonio, and R. Verbelen (2020). Boosting insights in insurance tariff plans with tree-based machine learning methods. *North American Actuarial Journal 25*(2), 255–285.

James, G., D. Witten, T. Hastie, R. Tibshirani, and B. Narasimhan (2022). ISLR2. Technical report, packaged 2022-11-20.

Jankowski, D. and K. Jackowski (2014). Evolutionary Algorithm for Decision Tree Induction. In *Computer Information Systems and Industrial Management*, Berlin, Heidelberg, pp. 23–32. Springer Berlin Heidelberg.

Jensen, D. D. and P. R. Cohen (2000). Multiple Comparisons in Induction Algorithms. *Machine Learning 38*(3), 309–338.

Kaminski, M. E. and G. Malgieri (2021, 4). Algorithmic impact assessments under the GDPR: producing multi-layered explanations. *International Data Privacy Law 11*(2), 125–144.

Kearns, M. and D. Ron (1999, 7). Algorithmic Stability and Sanity-Check Bounds for Leave-One-Out Cross-Validation. *Neural Computation 11*(6), 1427–1453.

Last, M., O. Maimon, and E. Minkov (2002). Improving Stability of Decision Trees. *International Journal of Pattern Recognition and Artificial Intelligence 16*(02), 145–159.

Liu, Y., B.-J. Wang, and S.-G. Lv (2014). Using Multi-class AdaBoost Tree for Prediction Frequency of Auto Insurance. *Journal of Applied Finance &amp; Banking 4*(5).

Lunde, B. . S., T. S. Kleppe, and H. J. Skaug (2020, 8). An information criterion for automatic gradient tree boosting. *arXiv*.

Manapragada, C., G. I. Webb, and M. Salehi (2018). Extremely Fast Decision Tree. In *Proceedings of the 24th ACM SIGKDD International Conference on*

Knowledge Discovery & Data Mining, KDD '18, New York, NY, USA, pp. 1953–1962. Association for Computing Machinery.

Nelder, J. A. and R. W. M. Wedderburn (1972). Generalized Linear Models. *Journal of the Royal Statistical Society. Series A (General) 135*(3), 370–384.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and . Duchesnay (2012, 1). Scikit-learn: Machine Learning in Python. *arXiv*.

Schelldorfer, J. and M. V. Wüthrich (2019). Nesting Classical Actuarial Models into Neural Networks. Technical report.

Schlimmer, J. and D. Fisher (1986, 3). A Case Study of Incremental Concept Induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 496–501.

Segal, M. (2003, 5). Machine Learning Benchmarks and Random Forest Regression. *Technical Report, Center for Bioinformatics & Molecular Biostatistics, University of California, San Francisco*.

Smith, S. L., B. Dherin, D. G. T. Barrett, and S. De (2021, 1). On the Origin of Implicit Regularization in Stochastic Gradient Descent. *arXiv*.

Takeuchi, K. (1976, 5). Distribution of information statistics and validity criteria of models. *Mathematical Science 153*, 12–18.

Tibshirani, R. and K. Knight (1999). Model Search by Bootstrap "Bumping". *Journal of Computational and Graphical Statistics 8*(4), 671–686.

Turney, P. (1995, 7). Technical Note: Bias and the Quantification of Stability. *Machine Learning 20*, 23–33.

Utgoff, P. E. (1989). Incremental Induction of Decision Trees. *Machine Learning 4*(2), 161–186.

van Engelen, J. E. and H. H. Hoos (2020). A survey on semi-supervised learning. *Machine Learning 109*(2), 373–440.

Vapnik, V. (1991). Principles of Risk Minimization for Learning Theory. In J. Moody, S. Hanson, and R. P. Lippmann (Eds.), *Advances in Neural Information Processing Systems*, Volume 4. Morgan-Kaufmann.

Wolpert, D. H. and W. G. Macready (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*(1), 67–82.

A Second-order Approximation vs. Exact for Poisson Loss

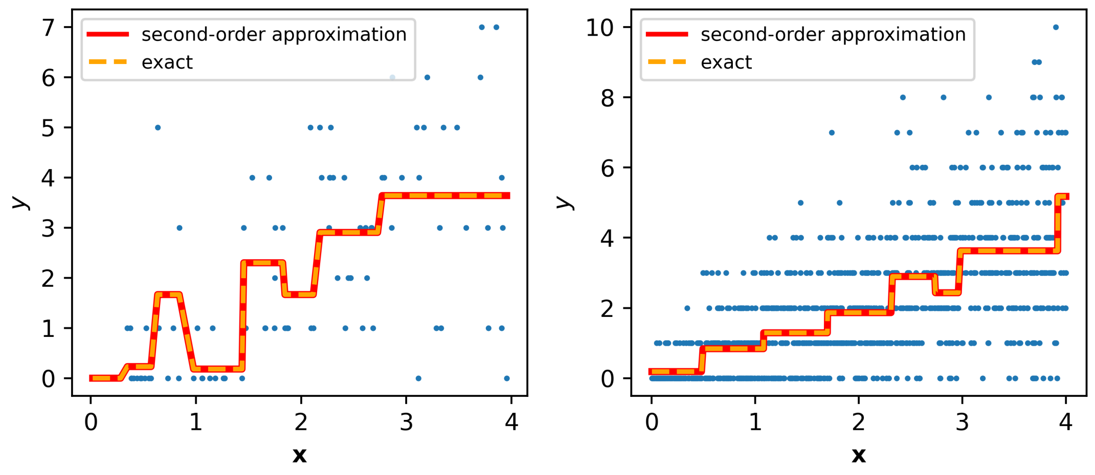


Figure 27: Second-order approximation vs. Exact for Poisson loss
The figure compares the predictions of a regression tree using the second-order approximation of loss reduction (and exact MLE of w) to a regression tree using exact loss reduction for Poisson loss. The trees are trained on simulated data with one feature, $\mathbf{x}_1 \sim \mathcal{U}(0, 4)$ and $y \sim Pois(\mathbf{x}_1)$ Both trees use *max depth* $= 3$ and *min sample leaf* $= 5$. Each step on the function corresponds to a region. Left plots for data size $n = 100$ whereas right plots for $n = 1000$.

B The Boston Dataset

Variable	Description
`crim`	per capita crime rate by town.
`zn`	proportion of residential land zoned for lots over 25,000 sq.ft.
`indus`	proportion of non-retail business acres per town.
`chas`	Charles River dummy variable (=1 if tract bounds river; 0 otherwise).
`nox`	nitrogen oxides concentration (parts per 10 million).
`rm`	average number of rooms per dwelling.
`age`	proportion of owner-occupied units built prior to 1940.
`dis`	weighted mean of distances to five Boston employment centres.
`rad`	index of accessibility to radial highways.
`tax`	full-value property-tax rate per $10,000.
`ptratio`	pupil-teacher ratio by town.
`lstat`	lower status of the population (percent).
`medv`	median value of owner-occupied homes in $1000s.

Table 10: The available variables in the `Boston` data set. (James et al. [2022])

C The Carseats Dataset

Variable	Description
`Sales`	Unit sales (in thousands) at each location.
`CompPrice`	Price charged by competitors at each location.
`Income`	Community income level (in thousands of dollars).
`Advertising`	Local advertising budget for companies at each location (in thousands of dollars).
`Population`	Population size in region (in thousands).
`Price`	Price company charges for car seats at each site.
`ShelveLoc`	A factor with levels Bad, Good, and Medium indicating the quality of the shelving location for the car seats at each site.
`Age`	Average age of the local population.
`Education`	Education level at each location.
`Urban`	A factor with levels No and Yes to indicate whether the store is in an urban or rural location.
`US`	A factor with levels No and Yes to indicate whether the store is in the US or not.

Table 11: The available variables in the `Carseats` data set. (James et al. [2022])

D The College Dataset

Variable	Description
`Private`	A factor with levels No and Yes indicating private or public university.
`Apps`	Number of applications received.
`Accept`	Number of applications accepted.
`Enroll`	Number of new students enrolled.
`Top10perc`	Percentage new students from top 10% of H.S. class.
`Top25perc`	Percentage new students from top 25% of H.S. class.
`F.Undergrad`	Number of full-time undergraduates
`P.Undergrad`	Number of part-time undergraduates.
`Outstate`	Out-of-state tuition.
`Room.Board`	Room and board costs.
`Books`	Estimated book costs.
`Personal`	Estimated personal spending.
`PhD`	Percentage of faculty with Ph.D.s.
`Terminal`	Percentage of faculty with a terminal degree.
`S.F.Ratio`	Student/faculty ratio
`perc.alumni`	Percentage of alumni who donate.
`Expend`	Instructional expenditure per student.
`Grad.Rate`	Graduation rate.

Table 12: The available variables in the `College` data set. (James et al. [2022])

E Appendix E: The Hitters Dataset

Variable	Description
`AtBat`	Number of times at bat in 1986.
`Hits`	Number of hits in 1986.
`HmRun`	Number of home runs in 1986.
`Runs`	Number of runs in 1986.
`RBI`	Number of runs batted in 1986.
`Walks`	Number of walks in 1986.
`Years`	Number of years in the major leagues.
`CAtBat`	Number of times at bat during his career.
`CHits`	Number of hits during his career.
`CHmRun`	Number of home runs during his career.
`CRuns`	Number of home runs during his career.
`CRBI`	Number of runs batted in during his career.
`CWalks`	Number of walks during his career.
`League`	A factor with levels A and N indicating players' league at the end of 1986.
`Division`	A factor with levels E and W indicating players' division at the end of 1986.
`PutOuts`	Number of put outs in 1986.
`Assists`	Number of assists in 1986.
`Errors`	Number of errors in 1986.
`Salary`	1987 annual salary on opening day in thousands of dollars.
`NewLeague`	A factor with levels A and N indicating players' league at the beginning of 1987.

Table 13: The available variables in the `Hitters` data set. (James et al. [2022])

F The Wage Dataset

Variable	**Description**
`year`	Year that wage information was recorded.
`age`	Age of worker.
`maritl`	A factor with levels 1. Never Married 2. Married 3. Widowed 4. Divorced, and 5. Separated indicating marital status.
`race`	A factor with levels 1. White 2. Black 3. Asian, and 4. Other indicating race.
`education`	A factor with levels 1. ¡HSGrad 2. HSGrad 3. Some College 4. College Grad, and 5. Advanced Degree indicating the education level of a worker.
`region`	Region of the country (mid-Atlantic only).
`jobclass`	A factor with levels 1. Industrial and 2. Information indicating type of job.
`health`	A factor with levels 1. ¡=Good and 2. ¿=Very Good indicating the health level of a worker.
`health_ins`	A factor with levels 1. Yes and 2. No indicating whether a worker has health insurance.
`wage`	Workers raw wage.

Table 14: The available variables in the `Wage` data set. (James et al. [2022])

G Tracking Model Updates Over Time for Simulated Poisson Data

Figure 28 shows how the different update methods perform over multiple update periods using simulated Poisson data. Here the models are learned using the Poisson loss and result in similar results as the experiment using squared error loss. Each update method provides a more stable update than the baseline across all update periods. The `NU` is consistent across all updates being very stable but with a larger loss. `TR` seems to perform the poorest as the other methods have lower loss and/or are more stable. For instance, for the first update $t = 1$, swapping `TR` with `NU` will result in a more stable update, and swapping for `ABU` will lead to lower loss. Swapping for `SL` or `BABU` will result in both a lower Poisson deviance and a more stable model than `TR`. `BABU` is dominated by `SL` in the first two updates before providing a more stable update but with a higher Poisson deviance. `ABU` is closest to the baseline in the first updates but provides a more stable update than the baseline and often has a lower Poisson deviance.

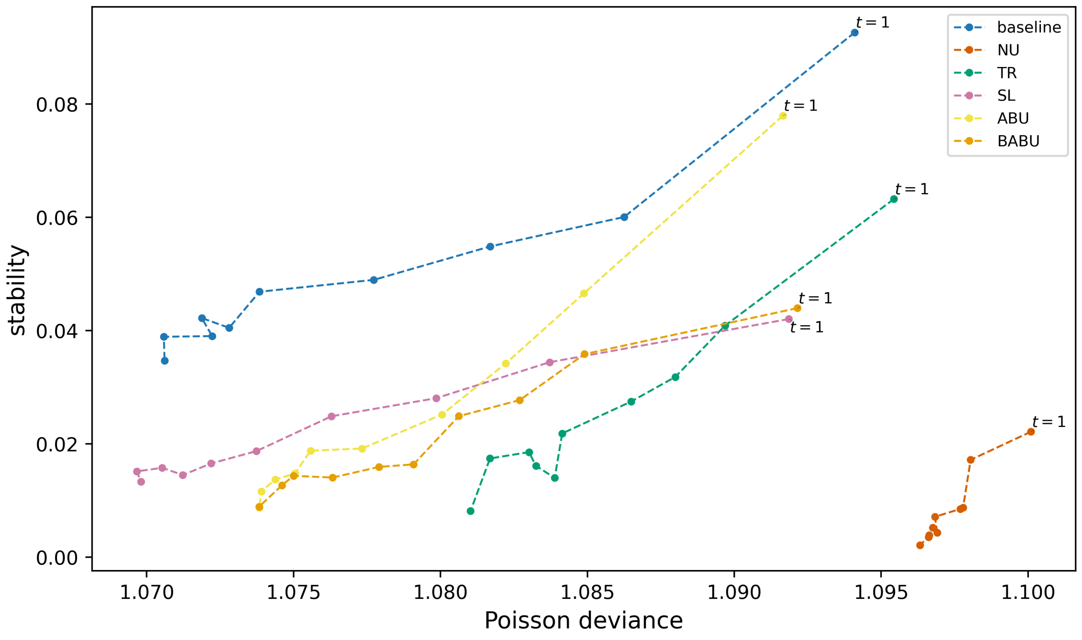


Figure 28: Tracking Model Updates Over Time for Simulated Poisson Data
The figure tracks the loss-stability objective of the different model update strategies over 10 update periods. Starting with a dataset of 1000 points, 1000 new data points are added to the dataset each update period using the data-generating process $Pois(\phi_1(\mathbf{x}))$ where ϕ_1 is the polynomial from Section 4.1 and $\mathbf{x}$ is simulated as showed in Table 1. The x-axis shows the mean Poisson deviance, whereas the y-axis shows the stability measure associated with Poisson data (Equation (19)). Each dot on the lines represents a model update, starting from the first update marked by the dots labeled $t = 1$. Absolute values of loss and stability are reported.

As t increases the number of data points also increases and the difference between `ABU` and `BABU` becomes smaller and smaller, i.e. converge towards the same function. This differs from the experiment using squared error loss, where `SL`, `ABU`, and `BABU` converged towards the same function. The difference is most likely due to the use of different regularization terms in Poisson loss for `SL` and `ABU`/`BABU`. For squared error, all three methods use the same regularization term, the squared error (i.e., L2 regularization). However, for Poisson loss, `SL` uses the Poisson loss as the regularization term, whereas `ABU` and `BABU` uses L2 regularization.

H Evolutionary Update

The CART's top-down greedy approach focuses on "how" to induce a tree and has a tendency to converge towards local optima. The evolutionary update method (`EVU`) aims to find a global optimum by shifting the focus from "how" to "what criteria a tree must satisfy". Instead of building a tree top-down greedy, the algorithm starts with a population of different trees and uses the concepts of crossover and mutation to create new generations of trees. Which trees to be selected for crossover and mutation is based on a global criterion known as a fitness function. After a predefined number of generations, the final generation is evaluated using the fitness function, and the best tree from the population is selected as the final tree. Algorithm 7 shows the pseudocode of the Evolutionary update method and is inspired by the work of Jankowski and Jackowski [2014].

Algorithm 7: Evolutionary update

```
input  : A training set $\mathcal{D}_2 = \{\mathbf{x}_i, y_i\}_{i=1}^n$
         Population size $N_p$
         Number of generations $N_g$
output: updated tree $f_2$
1 Function update($\mathcal{D}$,):
2     create a population of trees $p$ of size $N_p$
3     for $i \in 1, \dots, N_g$ do
4         selected individuals based on fitness score
5         apply crossover and mutation to selected individuals
6         update population $p$
7     select best tree, $f$, from population based on fitness score
8     return $f_2$
```

The initial population is created using the split process of CART, but instead of selecting split greedily, the splits are sampled using the loss to assign a probability to the splits. e.g., given a dataset with m features, the best split and the corresponding loss for each of the features are computed. Then, each split is assigned a probability based on its loss squared,

$$p_i = \frac{\frac{1}{\mathcal{L}_i^2}}{\sum_{i=1}^{m} \frac{1}{\mathcal{L}_i^2}}. \tag{23}$$

In addition to split sampling, the trees' max depth and minimum number of observations to split, a node are uniformly sampled between $(2, 11]$ and $(5, 11]$, respectively. The sampling of splits and hyperparameters ensures that the population is diverse.

Each generation is given a fitness score using a fitness function. The fitness score is used to rank the population. The fitness function used in the thesis is a weighted sum of three components, performance, stability, and tree complexity,

$$\text{fitness score} = w_1 \times \text{performance} + w_2 \times \text{stability} + w_3 \times \text{tree complexity}.$$

The performance is the MSE or mean Poisson deviance between y and $f_2(\mathbf{x})$ whereas stability is the MSE or mean Poisson deviance between $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$.

The complexity is the tree's depth. The lower the fitness score, the better the tree is.

The algorithm uses the current population to create the next generation's children based on the fitness function. Individuals can be selected multiple times, spreading their genes to more children. There exist multiple selection methods, but here stochastic universal sampling is the used selection method.

Crossover involves swapping subtrees between two trees in the current population. It starts by selecting two individuals using a selection algorithm. Next, a random node is chosen from each parent. The identified subtrees in both parents are then swapped, resulting in the creation of a new individual (offspring).

Mutations in the genetic algorithm introduce random changes to new individuals, providing diversity and expanding the search space. The mutation proposed for this method is simply randomly changing the splitting variable and split value of randomly selected nodes.