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

Grayson Steven Felt

# Predicting Loan Default with XGBoost: An Examination of Strength and Application

Actuarial and Financial Engineering

Master's Thesis (30 ECTS)

Supervisor: Assoc. Prof. Raul Kangro

Tartu 2024

# PREDICTING LOAN DEFAULT WITH XGBOOST: AN EXAMINATION OF STRENGTH AND APPLICATION

Master thesis

Grayson Steven Felt

**Abstract**

This thesis explores the application of the XGBoost algorithm for predicting loan defaults, a vital aspect of credit risk management. By leveraging advanced machine learning techniques, the study aims to improve the accuracy and reliability of default predictions over traditional methods. We begin with an overview of fundamental machine learning concepts, including loss functions and tree-based models, which sets the stage for a detailed examination of gradient boosting and its implementations. The focus then shifts to XGBoost, where we delve into its objective function, optimization process, and hyperparameters. Using a publicly available dataset from Bondora, we conduct thorough data preprocessing, followed by careful hyperparameter tuning using grid search and cross-validation. Our results highlight XGBoost's ability to handle complex, real-world data effectively, resulting in significant improvements in prediction performance. This study illustrates the importance of sophisticated algorithms in advancing the field of financial predictive analytics.

**CERCS research specialisation:** P160 Statistics, operations research, programming, financial and actuarial mathematics.

**Key Words:** XGBoost, loan default prediction, credit risk, machine learning, hyperparameter tuning.

**LAENU MAKSEJÕUETUSE ENNUSTAMINE XGBOOSTIGA: UURIMUS MEETODI TUGEVUSTEST JA RAKENDUSVÕIMALUSTEST**

Magistritöö

Grayson Steven Felt

**Lühikokkuvõte**

See lõputöö uurib XGBoost algoritmi rakendamist laenu maksejõuetuse ennustamiseks, mis on krediidiriski juhtimise oluline aspekt. Masinõppetehnikaid rakendades on uuringu eesmärk parandada maksejõuetuse ennustuste täpsust ja usaldusväärsust võrreldes traditsiooniliste meetoditega. Alustame ülevaatega põhilistest masinõppe kontseptsioonidest, sealhulgas kaofunktsioonidest ja otsustuspuudel põhinevatest mudelitest, mis loob aluse gradientboostingu ja selle rakenduste üksikasjalikuks uurimiseks. Seejärel nihkub fookus XGBoostile, kus süveneme selle sihtfunktsiooni, optimeerimisprotsessi ja hüperparameetritesse. Kasutades Bondora avalikult kättesaadavat andmekogumit, viime läbi põhjaliku andmete eeltöötluse, millele järgneb hoolikas hüperparameetrite häälestamine, kasutades ruudustikuotsingut ja ristvalideerimist. Meie tulemused tõstavad esile XGBoosti võimet tõhusalt käsitleda keerulisi reaalmaailma andmeid, mille tulemuseks on ennustuste täpsuse oluline paranemine. See uuring illustreerib keeruliste algoritmide tähtsust finantsanalüütika valdkonna edendamisel.

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

**Märksõnad:** XGBoost, laenu maksejõuetuse prognoosimine, krediidirisk, masinõpe, hüperparameetrite häälestamine.

# Contents

# Introduction

Machine learning and credit risk have become a perfect pairing in a world of expansive and complex datasets, growing in parallel to the rise of sophisticated and technical statistical models. More specifically, elaborate models such as random forest, support vector machines, and even neural networks have benefited default risk prediction. This paper examines where XGBoost stands alongside other popular models in credit risk modeling.

First, we establish core concepts of machine learning, such as loss functions and tree models. Then, we move into a brief overview of gradient boosting and popular boosting models. This sets the foundation for a detailed summary of XGBoost which is the main model of focus for this paper. Finally, we attach these theoretical models to a real dataset and examine the results.

The dataset processing and application of models are done in Python, while the content of this thesis is presented in LaTeX. The datasets provided are all public and can be accessed through the links provided.

# 1 Essentials of Machine Learning

The mathematical concepts and theory behind machine learning and make it clear how these complex models benefit in analyzing and predicting sophisticated datasets. While outputs of modern machine learning methods may be hard to interpret, a deeper look into their training steps and foundational functions uncovers the logic that builds robust results.

First, loss functions are introduced and compared. The concept of tree models and the CART algorithm are then explored to establish the basic components of other models referenced in this paper. Logistic regression is also examined due to its simplicity and omnipresence in credit risk modelling. A look at random forest finishes this chapter in order to understand tree model aggregation.

## 1.1 Introduction to Loss Functions

Loss functions are a foundation of any machine learning model, and choosing said function is a critical first step in analysis. We can be more confident in building an effective model by comparing available loss functions for binary classification, such as hinge loss, logistic loss, and exponential loss.

In the simplest sense, loss functions measure the difference between target values and modeled prediction output. Loss functions are minimized to set a mathematical objective through which we optimize our model. The actual formulation of each loss function, say $L(\hat{y}, y)$, is described for each. Note that $\hat{y}$ represents model prediction outputs of $y$. Binary classification is the primary objective for the models applied in this paper, which filters a few criterion that we examine in each loss function.

We refer to Hastie, Tibshirani, and Friedman, 2009 for the results of this subsection. As well as Cortes and Vapnik, 1995 specifically for reading on hinge loss.

### 1.1.1 Hinge Loss

Outputs are typically formulated as the set 0, 1 for binary classification. For hinge loss, however, the output must be transformed into the set {-1 ,1}. The reason becomes clear once you examine the formula below, which outputs the maximum between 0 and the product between $1 - y_i \cdot \hat{y}_i$.

$$L(\hat{y}, y) = max(0, 1 - y \cdot \hat{y}) \tag{1}$$

Hinge loss is most commonly applied to *support vector machine* models since it determines a "minimum margin" distance in which predictions are determined misclassified. Here $\hat{y}_i$ represents a linear hyperplane, formulated as $\hat{y} = \mathbf{w} \cdot \mathbf{x} + b$. This hyperplane divides the space between our two classes. If $\hat{y}$ shares the same sign with $y$, signifying a correct classification, and $\hat{y} > 1$, $L(\hat{y}, y) = 0$. If $\hat{y}$ shares the same sign with $y$ but $\hat{y} < 1$, this means that the prediction lies inside the margin boundary and provides a positive value for the loss function. In all other cases, with $y$ not sharing the same sign with $\hat{y}$, our loss function increases linearly.

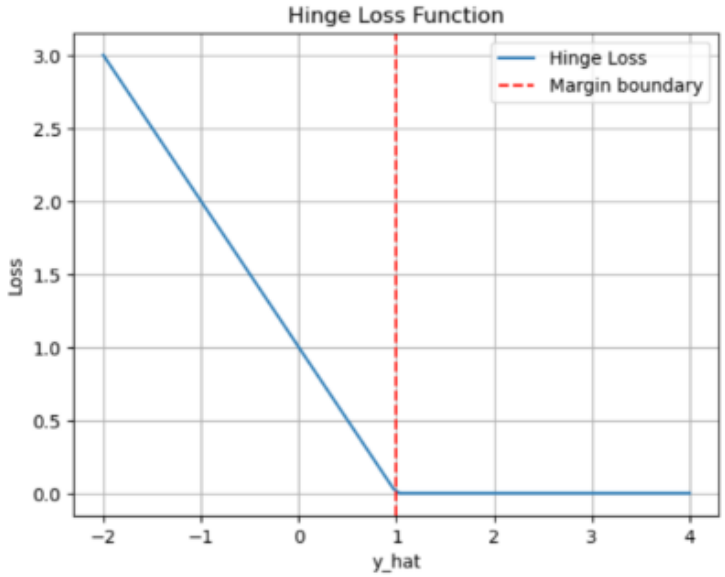


Figure 1: Graph of the Hinge Loss Function, illustrating how the loss varies with different values of $\hat{y}$ (predicted values). In this case, our loss function is set to $L(\hat{y}, 1)$, meaning the true outcome is controlled to be 1. The blue line represents the hinge loss, and the red dashed line indicates the margin boundary at $\hat{y} = 1$. The graph was created using Python's Matplotlib library.

The above plot describes the shape of the hinge loss function. The decision of where to place the margin boundary is dependent on how strict of a separation you desire between the two classes.

### 1.1.2 Exponential Loss

Exponential loss is a critical component in boosting algorithms, especially AdaBoost, which utilizes this loss function for updating the weights training instances. The exponential loss function increases the influence of misclassified points, thereby focusing the learning algorithm more on the difficult cases in subsequent iterations.

$$L(y, \hat{y}) = e^{-y \cdot \hat{y}} \tag{2}$$

Here, the loss function is presented in its multiplicative form, meant for binary classification. If values $y$ and $\hat{y}$ share the same sign, $e^{-y \cdot \hat{y}} < 1$, indicating a lower loss value. Opposite signs increase loss, with steepening magnitude as $\hat{y}$ moves further from the true $y$ value. This steepening penalty can best be described in the visualization below:

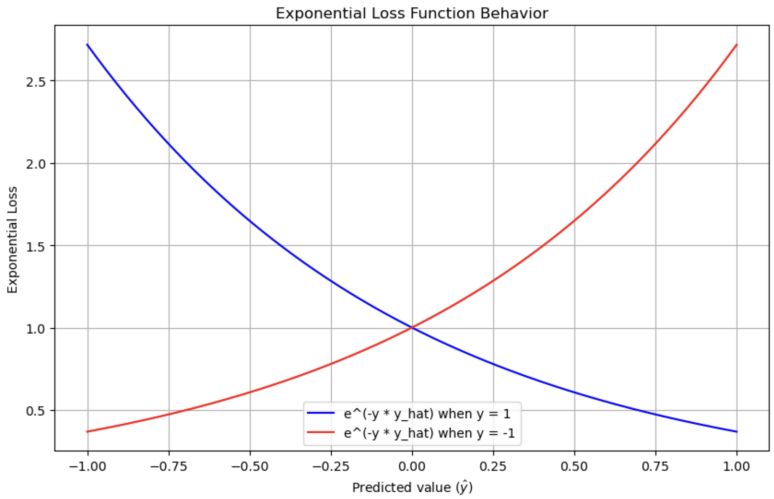


Figure 2: Plot illustrating the exponential loss function where the x-axis represents theoretical predicted value $\hat{y}$ and the y-axis represents the loss. As the product decreases, indicating increased disagreement between the predicted and actual labels, the loss sharply increases, emphasizing the cost of misclassifications.

The behavior of the exponential loss function well suits boosting models that correct errors from previous learning iterations. Sharp penalties of misclassification ensure that misclassified data points receive significantly higher weights in subsequent iterations. This pairing becomes more clear as gradient boosting, as well as specific boosting algorithms, are introduced in future sections.

### 1.1.3 Cross-Entropy/Log Loss

Cross-entropy loss, more commonly known as "log loss," is a much more dynamic loss function when compared to hinge loss. A critical flaw of other loss functions

that becomes prominent in this study is its inability to consider the confidence of predictions. Log loss instead includes the probability of each class, adding a measure of confidence into the calculation of loss. This is shown in the log loss formula below:

$$L(y, p) = -\left(y \log(p) + (1 - y) \log(1 - p)\right) \tag{3}$$

Here $y$ is the true binary 0,1 outcome, and $p_i$ is the predicted probability. Since probabilities are directly included in the formula, very confident but incorrect predictions are more heavily penalized. Not only does it consider accuracy, but confidence of prediction is measured, which is an essential consideration in credit risk.

When classifying risk in finance, the underlying probability distribution contains valuable information. It is clear that the default probabilities of 0.49 and 0.10 can convey a separation of financial risk or uncertainty. This separation should then be applied to a default model's loss function. The log loss function is useful because it heavily penalizes confident misclassifications of default cases. A misclassification with a high probability produces greater loss under this function.

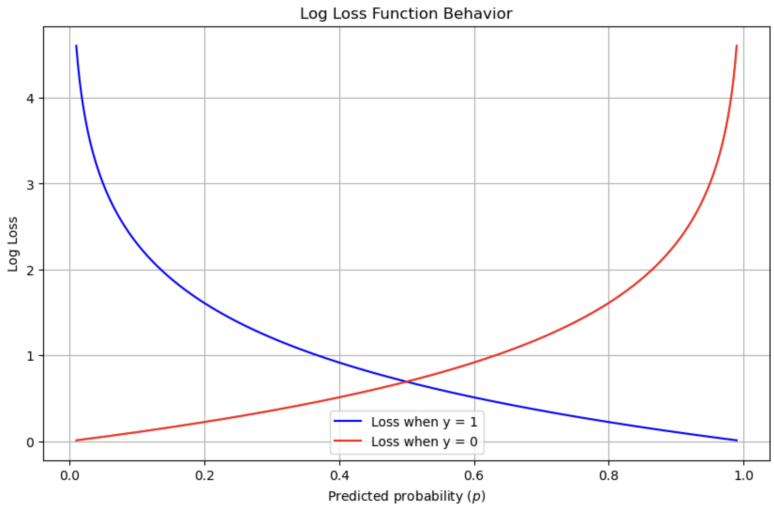


Figure 3: Visualization of Log-Loss Functions. This graph shows the log-loss functions for binary classification, where -log(p(x)) represents the loss for the positive class and -log(1-p(x)) for the negative class. The curves were generated using Python's Matplotlib library.

## 1.2 Logistic Regression

We refer to Cox, 1958 for the results regarding logistic regression in this subsection.

Logistic regression is one of the most widely used models, not only in general classification machine learning but also in credit risk modelling. It is mathematically and conceptually simple, providing easy to interpret results. For this reason, logistic regression is considered the "baseline model", or the model which this paper aims to significantly improve upon.

### 1.2.1 Model Basics

The logistic function, sometimes referred to as the sigmoid function, is the core of logistic regression modelling.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{4}$$

Where $z$ is a linear combination of the input features included in the model, and $\sigma(z)$ represents the output sigmoid function.

The underlying function in logistic regression can be easily expressed as a probabilistic equality. In default prediction, the probability that an input vector $X$ belongs to default class $l$ is expressed below:

$$P(Y = 1|X) = \sigma(X\beta + \beta_0) \tag{5}$$

- $X$ being input feature data.
- $\beta$ is the vector of weights.
- $\beta_0$ is the intercept.

Beta coefficients are determined using the maximum likelihood estimate (MLE) method, minimizing the average log loss given the model parameters. The MLE function is closely related to the logistic loss function introduced in the previous section, apart from a few key differences.

$$\ell(\beta) = \sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \tag{6}$$

Where $p_i = \frac{1}{1+e^{-\beta^T x_i}}$ is the probability of $y_i = 1$ given $x_i$ and $\beta$ parameters.

Log loss is solely cost function which penalizes incorrect predictions whereas MLE directly estimates parameters which will maximise the probability of given observed data.

MLE aims to set model parameters to values which maximize the log-likelihood of observing the given sampled data. Since the probability of observing the given $y$ outcomes given $X$ predictors is modeled using the logistic function in logistic regression, the likelihood function determines how probable the observed values are given set $\beta$ values.

The linear combination of input features and their corresponding coefficients is then transformed by the sigmoid function in order to restrict output as probabilities between 0 and 1. To then make classifications, a *decision boundary* needs to be established.

A decision boundary determines the threshold where probabilities are divided into the two classes of interest. By default, 0.5 is the probability threshold above which observations are classified as the positive class. Threshold choice is dependent on the cost of misclassifications between each class.

For instance, in medical fields it is much more critical that unhealthy and at risk patients are determined healthy (a false negative). This scenario would then suggest a lower decision boundary probability below 0.5 to ensure a higher accuracy of predicting the negative class. The finance industry is often at the other end of the spectrum, only approving borrowers who undoubtedly present low probability of default, but it ultimately depends on risk appetite of each financial institution.

### 1.2.2 Feature Reduction

A limitation of logistic regression prevalent in this research is its inability to handle complex datasets that include a high amount of features. This motivated the introduction of feature reduction common among linear models.

A combination of $L_1$ and $L_2$ regularization was chosen, known as ElasticNet. For each function below let $n$ be the number of features, $\lambda$ be a predetermined hyperparameter which controls the weight of regularization, and again $\beta_j$ be the coefficients of each model.

L1 can then be defined as:

$$L1(\beta) = \lambda \sum_{j=1}^{n} |\beta_j| \tag{7}$$

L2 as:

$$L2(\beta) = \lambda \sum_{j=1}^{n} \beta_j^2 \tag{8}$$

ElasticNet then being the combination:

$$ElasticNet(\beta) = \lambda \left( \alpha \sum_{j=1}^{n} |\beta_j| + \frac{1-\alpha}{2} \sum_{j=1}^{n} \beta_j^2 \right) \tag{9}$$

With $\alpha$ being a hyperparameter which controls the ratio between $L_1$ and $L_2$ regularization used. $\alpha = 0$ corresponds to pure $L_1$ regularisation (more commonly known as Ridge), and $alpha = 1$ corresponding to pure $L_2$ regularisation.

### 1.2.3 Advantages and Limitations

A clear advantage of logistic regression is its simplicity to implement and interpret. No other models applied in this research will be able to plainly provide coefficient weights. A clear limitation of logistic regression is that it requires well-chosen interaction terms in order to capture non-linear relationships in the data. This is quite often a difficult task, and complicates this advantage of simple interpretation.

## 1.3 Decision Tree Models

We refer to Breiman et al., 1984 for results on decision trees and the CART algorithm in this subsection.

The simple but powerful logic within decision trees have inspired a diverse multitude of machine learning models. This method of decision-making allows for clear and straightforward predictions, making decision trees a popular choice for various applications. Despite their simplicity and interpretability, single decision trees often struggle with overfitting and may not generalize well to new data. To overcome these limitations, advanced models like Random Forest and XGBoost have been developed.

### 1.3.1 CART Algorithm

Simple tree models are built using the CART (Classification And Regression Tree) algorithm, which continuously divides the dataset into binary splits with “leaves”. A leaf in this context is simply the final decision region at the end of a logic path, which is also known as a "branch". At a leaf, the final condition is presented which outputs the model prediction. The CART algorithm creates binary decision trees by splitting the source data into subsets based on feature values that result in the largest information gain for classification tasks or the greatest reduction in variance for regression tasks. Each decision node in the tree represents a single input feature and a split point on that feature, aimed at differentiating the observations in a way that organizes the data into increasingly homogeneous subsets.

The algorithm’s steps for classification are defined in Breiman et al. (1984). First, the root node is constructed including the entire dataset. The Gini impurity or entropy is calculated for each possible initial feature split in the dataset. The aim is

to maximize the decrease in impurity from the parent node to the weighted average impurity of the two child nodes.

Gini impurity is mathematically defined below:

$$G(S) = 1 - \sum_{i=1}^{k} p_i^2 \tag{10}$$

Where $S$ is the subset of the dataset for which the Gini impurity is being calculated, $k$ is the number of different classes in the subset $S$, $p_i$ is the proportion (or probability) of class i within subset $S$.

The above process is then applied recursively to each derived subset until one of the pre-defined stopping criteria is met:

1. Maximum tree depth is reached.
2. Maximum node size is achieved.
3. No further decrease in impurity or variance is possible.

A visualization of how CART recursively splits a dataset into a tree is shown below:

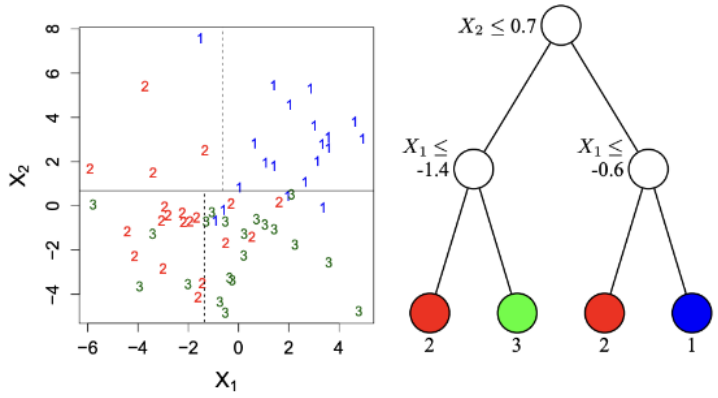


Figure 4: Decision tree model illustrating splits based on features X1 and X2, adapted from "Classification and Regression Trees" by Wei-Yin Loh.

CART trees are visually satisfying and relay their simple but powerful logic. Their ability to separate data into non-linear sections almost always results in impressive performance. This is why most machine learning competitions, including Kaggle, see decision tree based models performing close behind if not at the top of the podium. Still, there are methods of improving decision tree models.

## 1.4 Random Forest

We refer to Breiman, 2001 for results in this subsection.

At a high level, random forest is an ensemble learning technique that enhances the performance of decision trees by constructing multiple trees and combining their outputs. This method addresses the overfitting problem commonly associated with individual decision trees and improves generalization by leveraging the diversity of multiple trees.

The random forest algorithm works by creating a "forest" of randomly created decision trees. Each tree is trained on a different bootstrap sample from the original

dataset, meaning that each tree sees a slightly different view of the data. Additionally, when constructing each tree, the algorithm selects a random subset of features at each split, further introducing variability.

### 1.4.1 Algorithm Steps

The algorithm follows these steps:

- For each tree, generate a bootstrap sample (a random sample with replacement) from the training data.
- At each node of the tree, a random subset of features is selected as candidates for splitting. This selection is typically controlled by a hyperparameter, such as $\sqrt{n}$ or $\log n$, where $n$ is the number of available features.
- A decision tree is grown on the bootstrap sample by recursively splitting the nodes based on the feature that provides the best split according to a chosen impurity criterion (e.g., Gini impurity or entropy).
- For regression tasks, the final prediction is obtained by averaging the predictions from all trees. For classification tasks, the final prediction is determined by majority voting.

The optimization process in Random Forest focuses on finding the best splits at each node to minimize the impurity. Given a node $t$ containing a subset of training instances, the algorithm evaluates all possible splits and selects the one that minimizes the impurity measure.

For each candidate split, the dataset $S$ is divided into two subsets $S_L$ and $S_R$. The weighted impurity $I_{split}$ of the split can then be calculated using the following equation:

$$I_{split} = \frac{|S_L|}{|S|} I(S_L) + \frac{|S_R|}{|S|} I(S_R) \tag{11}$$

Where $I$ is the chosen impurity measure (e.g., Gini or entropy), and $|S|$ is the number of instances in subset $S$. The split which results in the lowest impurity is then chosen.

### 1.4.2 Advantages and Limitations

Random forest can successfully reduce the risk of overfitting and provide robust predictions in compared to lone decision trees by incorporating multiple tree predictions. This method is useful for both regression and classification tasks.

However, random forest models are harder to interpret in comparison to a decision tree or linear models. They are also much more computationally expensive and time consuming over large datasets. In this sense, the flexibility of random forest is a sacrifice.

# 2 Gradient Boosting

This chapter introduces the concept of gradient boosting, and compares this method with other models already introduced. A comprehensive outline of the optimization steps are also established in order to further understand the method's distinctiveness.

## 2.1 Background on Gradient Boosting

We refer to Friedman, 2001 for results on general gradient boosting in this subsection.

Gradient boosting, in the basic sense, is a method of minimizing a loss function through adding predictions of multiple weak models together. Weak meaning simple and performing marginally better than just random guessing. These weak learners are usually decision trees that perform just slightly better than random guessing. The philosophy of gradient boosting is to improve the model step-by-step by addressing the mistakes made in previous iterations.

One of the main distinctions between gradient boosting and the CART algorithm defined in the previous chapter, particularly in classification tasks, is the criterion used for making splits in the trees. While CART typically uses measures like Gini impurity or information gain (entropy) to decide how to split the data, gradient boosting takes a different path. Instead of these impurity measures, gradient boosting aims to find splits that directly reduce a specific loss function, such as log loss for classification or mean squared error for regression. This approach allows gradient boosting to align more closely with the goal of minimizing prediction error.

In machine learning, the harmony between bias-variance trade-off is difficult to

achieve through hyperparameter selection. A complex tree model can minimize training loss by including a large amount of leaves or extreme depth. Still, a model like this would exhibit undesirable results when applied to separated test data. To optimize the trade-off between bias and variance, random forest models fit many complex trees and average the result. Gradient boosting instead prefers a sequential approach. A simple or "weak" tree is created at the first step which provides an array of residuals. A second tree is then created to predict these residuals, repeating this dependent process until a stopping criteria is met, and a final prediction is made. In this sense, we are not making giant leaps between variance and bias, as each iteration is a small controlled step towards minimizing the specified loss function. The stopping criterion allows greater control over complexity and overfitting, but also provides a complex choice which to optimize.

To be more specific, in the context of tree-based models, gradient boosting trees are built based on the residuals of previous trees in a sequence until the residual size is not significantly reduced or the maximum amount of trees is reached. Residuals from previous trees are predicted in a new tree, with their predictions being multiplied by a learning rate. Final outcomes are measured by the sum of the initial prediction and scaled residual predictions from sequential trees.

Gradient boosting, when applied to classification problems, can also be easily compared to logistic regression. Gradient boosting aims to find a prediction function that can take any real numbers as values. In the context of classification, this prediction function could represent log odds, which a logistic function can then convert to probabilities. This specifically enables the use of log loss, which is the preferred loss function in this case. Alternatively, we can aim to make a prediction function positive for one class and negative for the other. The chosen loss function measures the goodness of the prediction function for a given outcome and must be differentiable with respect to its second argument. This differentiability is crucial as it

allows the gradient boosting algorithm to leverage the additive information gain effectively. By iteratively adding the differentiated and scaled information gain to previous predictions, gradient boosting minimizes bias through many small steps, ensuring that variance is not sacrificed.

### 2.1.1 Optimization Steps

The basic steps, as outlined by Friedman, 2001 introduction to this method, are as follows:

1. Initialize the model with a constant value $\rho$, which minimizes the loss function that is applied to the entire dataset.

$$F_0(x) = \arg\min_{\rho} \frac{1}{N} \sum_{i=1}^{N} L(y_i, \rho) \tag{12}$$

   This calculates the initial leaf condition from which further trees are built.

2. Iteratively calculate interim residuals $\tilde{y}_i$ computed as negative gradients determined from the loss function $L$. Repeat this calculation over $M$ iterations.

$$\tilde{y}_i = - \left[ \frac{\partial L}{\partial \hat{y}} (y_i, F(x_i)) \right], \quad i = 1, N \tag{13}$$

3. Fit a learner to the resulting $\tilde{y}$ residuals. The parameters of this learner $a_m$ are determined by minimizing the squared summation of the differences between interim-residuals $\tilde{y}_i$ and correction term $h(x_i; a)$.

$$a_m = \arg\min_{a} \sum_{i=1}^{N} [\tilde{y}_i - h(\mathbf{x}_i; a)]^2 \tag{14}$$

4. The residual corrections of this new learner are then added to the previous model's predictions. The loss function is again applied to this aggregated prediction and is minimized by changing step size $p$.

$$p_m = \eta \cdot \arg\min_p \frac{1}{N} \sum_{i=1}^{N} L(y_i, F_{m-1}(\mathbf{x}_i) + ph(\mathbf{x}_i; a_m)) \tag{15}$$

After minimizing the loss over all training instances, the optimal step size $p_m$ is calculated.

5. The new predictions for $F_m(x)$ are then determined by adding the scaled base learner's predictions ${}_m h(x; a_m)$ to the previous iteration's $m-1$ predictions $F_{m-1}(x)$.

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + p_m h(\mathbf{x}; a_m) \tag{16}$$

## 2.2 Comparison with Other Boosting Algorithms

Gradient boosting has birthed a handful of popular machine learning models. LightGBM and AdaBoost are introduced in short to present the achievements made from this approach.

### 2.2.1 AdaBoost

We refer to Freund and Schapire, 1997 for results in this subsection.

Adaptive Boosting, known as AdaBoost, is one of the earliest boosting algorithms. Unlike Gradient Boosting, which focuses on minimizing a loss function through gradient descent, AdaBoost increases the weights of misclassified instances, giving them more focus in subsequent iterations. This iterative process combines multiple

weak learners into a single, more accurate model.

Another key difference between AdaBoost and gradient boosting can be seen within the sequential training process. Instead of fitting new models to correct the residuals of previous ones, AdaBoost changes the weights of misclassified samples to improve the next model.

AdaBoost can also be interpreted within the framework of gradient descent. Specifically, it corresponds to gradient descent with an exponential loss function. This connection can be seen in the fact that AdaBoost iteratively adjusts weights in a manner that aligns with minimizing the exponential loss $L(y, \hat{y}) = e^{-y\hat{y}}$.

### 2.2.2 LightGBM

We refer to Ke et al., 2017 for results in this subsection.

Light Gradient Boosting Machine, or LightGBM, is a newer algorithm developed by Microsoft, designed to enhance the efficiency and scalability of Gradient Boosting. It includes innovations like histogram-based algorithms and leaf-wise tree growth, which make it well-suited for large datasets.

The most important quality of LightGBM, as emphasized in its name, is its efficiency. LightGBM employs a histogram-based method to speed up training by reducing the number of potential split points. This suggests that LightGBM is better suited for large datasets and high-dimensional data compared to traditional Gradient Boosting methods.

### 2.2.3 Use Cases and Performance

Choosing the right boosting algorithm clearly depends on the specific needs of the problem. AdaBoost is simple and effective for smaller, balanced datasets. Gradient Boosting offers flexibility and can be fine-tuned for various applications. LightGBM is advantageous for handling large-scale data due to its efficiency and speed. XGBoost is the remaining gradient boosting model to define, which is ubiquitous in the industry.

# 3 Introduction to XGBoost

We refer to Chen and Guestrin, 2016 for results in this section.

While the core concepts of gradient boosting provide a strong foundation for understanding machine learning models, XGBoost takes this framework a step further. XGBoost, short for Extreme Gradient Boosting, is an extension of the gradient boosting algorithm that has become widely popular in both academia and industry due to its scalability and robustness.

One of the fundamental distinctions of XGBoost lies in its departure from the independent tree-building process characteristic of random forests. Instead of training trees in isolation, XGBoost adopts the gradient boosting philosophy of sequential learning, where each subsequent tree is trained to correct the errors made by its predecessors. This approach gives a tighter control over model complexity and generalization, improving predictive accuracy.

## 3.1 Objective Function

The objective function that XGBoost aims to optimize is defined below:

$$F_{obj} = \sum_{i=1}^{n} L(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) \tag{17}$$

It consists of two parts: a loss function summed over all training samples $n$, and a regularization function.

$y_i$ is the true label of the $i$-th sample (either 0 or 1 for binary classification)

$\hat{y}_i^{(t-1)}$ is the predicted output of the $(t-1)$-th iteration

$f_t(x_i)$ is the prediction of the $t$-th tree for the $i$-th sample

$L(y, \hat{y})$ is the loss function, which is the log loss for classification:

$$L(y, \hat{y}) = (-y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}) \tag{18}$$

$\Omega(f_t)$ is the regularization term, which penalizes the complexity of the $t$-th tree. It can be defined as:

$$\Omega(f_t) = \alpha T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{19}$$

$T$ is the number of leaves weighted by the regularization hyperparameter $\alpha$. The other term is equivalent to the $L_2$ regularization term in linear regression, penalizing the squared weights $w$ of each leaf in the tree and weighted by the hyperparameter $\lambda$. This regularization term added to the objective function is one of the main distinguishing features of XGBoost.

## 3.2 Taylor Series Approximation and Optimization

The XGBoost algorithm optimizes its objective function by utilizing a second-order Taylor series approximation to the loss function. This method is important for efficiently finding an optimal solution with fewer iterations and more stability compared to methods that use only the first derivative. The original XGBoost paper by Chen and Guestrin details this approach as follows:

Using a second-order Taylor series approximation, $\mathcal{L}^{(t)}$ can be approximated as:

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{N} \left[ L(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

where $g_i = \frac{\partial L}{\partial \hat{y}}(y_i, \hat{y}_i^{t-1})$ and $h_i = \frac{\partial^2 L}{\partial \hat{y}^2}(y_i, \hat{y}_i^{(t-1)})$ are the first and second derivatives

of the loss function with respect to the predictions at the $(t-1)$-th iteration.

Omitting the constant term as it does not affect optimization, the simplified objective function becomes:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{N} \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

For a given tree structure, let $I_m = \{i | x_i \in R_m\}$ denote the set of instances in leaf $m$. The equation then simplifies further:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{m=1}^{M} \left[ \left( \sum_{i \in I_m} g_i \right) \gamma_m + \frac{1}{2} \left( \sum_{i \in I_m} h_i + \lambda \right) \gamma_m^2 \right] + \alpha M$$

where $\gamma_m$ is the prediction of leaf $m$, and $M$ is the total number of leaves.

The optimal value for $\gamma_m$ that minimizes $\tilde{\mathcal{L}}^{(t)}$ is given by:

$$\gamma_m^* = -\frac{\sum_{i \in I_m} g_i}{\sum_{i \in I_m} h_i + \lambda}$$

The corresponding optimal value of the loss function is:

$$\tilde{\mathcal{L}}^{(t)}(\gamma_m^*) = -\frac{1}{2} \sum_{m=1}^{M} \frac{(\sum_{i \in I_m} g_i)^2}{\sum_{i \in I_m} h_i + \lambda} + \alpha M$$

## 3.3 Optimization Algorithm

The optimization process, as introduced in the previous section, aims to minimize the objective function. Following an additive approach, predictions at any $t$ iteration results from the sum of previous $t-1$ iteration's prediction and new information gain from the $f_t(x_i)$ tree.

The optimization procedure, in case of using log loss, can be summarized as follows:

1. Initialize the model: The initial prediction $F_0(x)$ is set to be the log odds of the positive class, which can be calculated as:

$$F_0(y) = \log\left(\frac{\bar{y}}{1-\bar{y}}\right) \tag{20}$$

where $\bar{y}$ is the mean of the target variable $y$.

2. Iterating over all trees m $m = 1, 2, ..., M$, compute the gradient (negative gradient) of the loss function with respect to the current prediction:

$$g_i^{(m)} = \frac{\partial L}{\partial \hat{y}}(y_i, \hat{y}_i^{(t-1)}) \tag{21}$$

3. Fit a new tree $h_m(y)$ directly to the gradients $g_i^{(m)}$ using the splitting criterion defined in the previous section.

4. Calculate the update for the model using the pre-defined learning rate hyperparameter $\eta$:

$$F^{(m)}(y) = F^{(m-1)}(y) + \eta h_m(y) \tag{22}$$

5. Update the model prediction:

$$F^{(m)}(y) = F^{(m-1)}(y) + \eta_m h_m(y) \tag{23}$$

Final prediction: The final prediction, the probability that $\hat{y} = 1$ is obtained by applying a sigmoid transformation to the cumulative sum of the predictions:

$$\hat{y} = \frac{1}{1 + e^{-F^{(M)}(y)}} \tag{24}$$

## 3.4 Hyperparameters

There are quite a few important hyperparameters to determine before training an XGBoost model. $\eta$, already introduced above in equation 22, acts as a learning rate in gradient boosting models. Values closer to zero "shrink" the amount of information gained from sequential models. $\gamma$ also determines the complexity of XGBoost trees by setting the minimum amount of information gain needed to further split a leaf. There are also hyperparameters designed to reduce model variance, such as 'colsample', which determines the subsample size of columns used for each tree, level, or leaf. Other hyperparameters are more intuitive; maximum tree depth, maximum leaves, or $L_1$ and $L_2$ regularization weight.

## 3.5 Feature Importance

Interpretation is one of the greatest strengths for simple classification models. Still, there are opportunities to combat this disability in gradient boosting models. Feature importance is the most commonly employed solution since it mimics coefficient magnitude from linear and generalized linear models. It is important however, to understand that feature importance *cannot* be interpreted as coefficient magnitude. These two descriptors are mathematically very different, necessitating a explanation of the details behind feature importance.

Feature importance in XGBoost is not calculated using the Gini impurity, as in traditional decision tree models, but rather by measuring the reduction in the specified loss function that results from splits using each feature. This method leverages the actual loss function used to train the model, which can vary depending on the task (e.g., logistic loss for classification or squared error for regression).

When a feature is used to split a decision node, XGBoost evaluates the improvement in the objective function due to this split. The importance of a feature is then determined by the total improvement in the loss function across all the trees in the model. The more a feature contributes to reducing the loss function, the higher its importance score.

$$I_j = \sum_{t=1}^{T} \sum_{s \in S_t(j)} \Delta L_{s,t} \tag{25}$$

Here, $T$ is the total number of trees in the model, $S_t(j)$ is the set of splits on feature $j$ in tree $t$, and $\Delta L_{s,t}$ is the reduction in the loss function due to split $s$ in tree $t$.

This paper relies on feature importance specifically to compare and examine significant features within gradient boosting models. A comparison is also made between the features deemed most important in gradient boosting models as a result of feature importance, compared to coefficient strengths in logistic regression. These comparisons are only made based on the ordinal ranking of important features, and are not compared as a difference of real value.

## 3.6 Advantages and Limitations

A series of advantages and limitations are presented in order to understand where XGBoost stands in comparison to the models already defined. Greater detail is also presented on how XGBoost can benefit or complicate credit risk analysis and default prediction.

### 3.6.1 Advantages

One of the most significant advantages of XGBoost is its computation speed. XGBoost is known for its fast execution speed due to optimizations such as parallel

processing, tree pruning, and cache awareness. This efficiency not only gives XGBoost an advantage over other machine learning models, but it is also efficient in comparison to gradient boosting models as well. Financial institutions often deal with large volumes of data, as evidenced with the dataset introduced in this paper. A computationally efficient model not only saves time for financial institutions, but can also reduce costs.

Another advantage was introduced as the defining feature of XGBoost: a combination of $L_1$ and $L_2$ feature regularization. Feature regularization assists in both reducing overfitting and improving model generalization.

With respect to model tailoring, XGBoost can assign weights to classes, which is beneficial in credit risk scenarios where default cases are typically less frequent than non-default cases. The diverse set of hyperparameters which are available to XGBoost models enable detailed but complex training procedures.

### 3.6.2 Limitations

Without careful tuning and proper regularization, XGBoost models can overfit, particularly on smaller datasets. This danger adds another layer of complication to the training process. XGBoost has many hyperparameters, which can make the model complex and require extensive tuning to optimize performance.

The most important downside of XGBoost in the context of credit risk is the difficulty in interpretation. Compared to simpler classification models like KNN or Logistic Regression, where one predictor is trained, and decisions can be easily explained, XGBoost is what's considered a "black box". Hundreds of predictors help make a final prediction in XGBoost and make the process inexplicable to those unfamiliar with machine learning methods. Still, model output can be scrutinized

through confusion matrices, loss measures, and feature importance.

The complexity and lack of transparency can be a disadvantage in regulated industries like finance, where explainability is crucial for compliance. Models must be auditable, and the complexity of XGBoost can make it challenging to document and explain decisions comprehensively. Despite providing feature importance, the overall model can be seen as a "black box," making it harder for stakeholders to understand the full decision-making process.

# 4 Methodology

This chapter introduces the dataset used for model application and analysis, the data processing techniques, and how each model was applied. As with most data research projects, careful data preparation will become the most time-intensive and difficult step. Credit risk datasets are no different and require detailed cleaning. All steps conducted in data preprocessing will be summarized to provide a strong foundation for the introduction of models.

The dataset used in this research comes from Bondora. The dataset was chosen as it is publicly available, includes detailed documentation, and is relevant to credit risk analysis. One significant quality of Bondora as a company is that it is a peer-to-peer (P2P) lender. P2P lending is a modern method of debt financing that allows individuals to borrow and lend money without using an official financial institution as an intermediary. This model connects borrowers directly to lenders through an online platform. Investors looking to lend money can review the available loan requests on the platform and choose to fund those that meet their investment criteria. They can also diversify their investment by lending small amounts across many different loans.

P2P lending provides easier access to loans, especially for those struggling to secure traditional bank financing. This, however attracts a unique customer set to be considered when analyzing credit risk. We can expect a significant increase in risk when compared to traditional lending since P2P platforms often provide loans to individuals who may not qualify for traditional bank loans due to lower credit scores or unconventional income sources. Many of these P2P loans are also unsecured, meaning that they do not require collateral from the borrower.

P2P loans often carry higher interest rates than traditional bank loans to compen-

sate for higher perceived risk. While this can attract investors looking for higher returns, it also increases the financial burden on borrowers, potentially leading to higher default rates. The platform often determines interest rates based on an assessment of the borrower's credit risk or through a bidding process where lenders offer funds at varying interest rates. These characteristics of the dataset are explored further after data preparation.

To the dataset, five main steps are applied: dimensionality reduction, imputation, encoding, standardization, and splitting. Data requirements between machine learning models can be diverse, necessitating a separate pipeline for each. After each pipeline is outlined, the application of each model will be defined.

## 4.1 Data Preprocessing

Statistical models are undoubtedly sensitive to data preparation techniques. The objective of this process is to rectify inconsistencies, construct a sensible structure, and mathematically transform our datasets. One always needs to be continuously cautious and meticulous when constructing preprocessing pipelines. When tools are used incorrectly or applied in baseless scenarios, results become stained.

The first step is the removal of unnecessary or unusable features from the dataset. Many features included, such as the index, have no relation to default. Other unusable features are removed due to having a confounding relationship with default or providing look-ahead bias. These two biases can be easy to overlook when preparing a dataset, but including a variable such as late fee amounts can lead to misleading results. The Bondora dataset used is also available on Kaggle, and as one browses the top results, it becomes easy to spot this mistake. Many users report an accuracy score close to 99.9% yet include features such as "DefaultDate" in the final model.

Another filtering condition used to reduce the amount of columns was the amount of missing data. Some features in the dataset contained nearly 100% missing data, usually indicating private information that could identify an individual borrower. Features that included a smaller proportion of missing data were instead imputed, the details of which will be explained in greater detail.

These preliminary preprocessing steps reduced the features from around 112 to just 41. The next step in this preprocessing pipeline is imputation. The most common solution to missing data is simple arithmetic mean imputation. A more careful and informative approach is random forest imputation. This method trains a random forest model to the entire dataset, treating the column with missing data as an independent variable and predicting the values which are missing. Compared to the simpler approaches, random forest imputation ensures that variance is preserved within the dependent variables in the dataset.

Unfortunately, the same data preparation techniques cannot be applied to every model used. For example, logistic regression requires encoding and standardization, while XGBoost / Random Forest handles these transformations intrinsically. Separate data preparation pipelines were constructed for this purpose.

## 4.2 Hyperparameter Selection

This section examines the fitting and choice of hyperparameters for each model. A few important methods like grid search and cross-validation are used to ensure that optimal hyperparameters are chosen. Not all models share the same hyperparameters, and not all hyperparameters are chosen for each model. These choices are explained alongside their interpretation. After the selection of hyperparameters,

the models are then applied to the dataset, and results are shown.

For each model, a table is created to describe the pool of hyperparameters to which each model is allotted. Computational efficiency is also examined and compared between each model, an important consideration for future application to larger datasets.

### 4.2.1 Grid Search

Hyperparameter choice is automated through grid search. Grid search is a widely used hyperparameter optimization technique that provides and applies a specified range of values for each hyperparameter of a model to determine the combination that produces the best results, typically measured by a predefined performance metric.

The performance metric used to determine the optimal combination is log loss. The combination which *minimizes* the *negative* log loss is chosen as the most optimal pairing. Almost all machine learning optimization algorithms are designed to specifically minimize a positive loss function. This is why the negative log loss is used since the original formulation of the log loss function yields a negative value. In this sense, when we minimize the negative log loss, we are minimizing the positive log loss, pushing model predictions $p$ to align closely with our true labels $y$.

### 4.2.2 Cross-Validation

In every model training process, it is paramount that train and test sets are kept separate in order for the model to be able to generalize effectively to new data it has not seen. Should the training data be included as a test set, statistical tests

would be overconfident. This separation necessitates a foundational machine learning concept – cross-validation. During model training, as unique model instances are conditioned with distinct hyperparameters, the "best" model should be the model which will outperform on most testing sets.

We can estimate testing performance by partitioning the training set into $k$ equally sized subsets. Across $k$ iterations, $k-1$ subsets are used to train the model, and the remaining subset is used to validate and estimate testing performance. Once each subset, or more commonly known as "fold", has been used as a testing set, we calculate an average performance score.

Log loss is calculated using 5-fold cross-validation. The hyperparameters that minimize cross-validated log loss indicate the best-performing model.

## 4.3 Model Application

In this section, we outline the application of our defined models to the dataset, detailing the parameter grids used and the process of fitting each model to the data. Remember that the ultimate goal is to optimize each model's performance using grid search and cross-validation techniques, ensuring robust predictions.

### 4.3.1 Logistic Regression

For the logistic regression model, a pipeline was created that included both preprocessing and model fitting steps. The preprocessing involved transforming the data to ensure it was suitable for the logistic regression model. Logistic regression requires encoding categorical variables into numerical values because it operates on mathematical equations that cannot directly process categorical data. Label

encoding was used, where numerical categories are created without dramatically increasing the dimensionality as done in one-hot encoding. This step is crucial for logistic regression to interpret the categorical variables correctly.

Standardization involves scaling numerical features to have a mean of zero and a standard deviation of one. Resulting standardized values $z$ can be calculated using the equation below:

$$z = \frac{x - \mu}{\sigma} \tag{26}$$

Here $x$ is simply the original value, $\mu$ is the mean of the chosen feature, and $\sigma$ is the standard deviation of the feature. This step is important for logistic regression because it helps ensure that all features contribute equally to the model, preventing features with larger scales from dominating the model coefficients.

A solver was used which was compatible with elastic net regularization, allowing for a mix of $L_1$ and $L_2$ regularization.

Table 1 defines the range of parameters established for logistic regression, from which a matrix of models are trained. A short description of each hyperparameter is also included. The interpretations are needed to understand before ranges of values are chosen for each parameter, as any value outside of the range {0,1} for *l1_ratio* would throw error. These interpretations also assist in forming appropriate ranges that ensure the chosen values make sense in the context of the model and the data, thereby improving the model's performance and interpretability.

One additional step added to the logistic regression preprocessing pipeline: spline transformation. Spline transformation is a valuable technique for enhancing the flexibility and accuracy of regression models. By transforming input features into

piece-wise polynomial functions, it captures non-linear relationships between predictors and the target variable. This method divides the data range into intervals, fitting a polynomial function to each, which ensures smooth transitions at the knots. In logistic regression, which typically assumes a linear relationship between predictors and log-odds of the outcome, spline transformation offers significant benefits. It allows for modeling complex, non-linear patterns, thereby improving predictive performance while maintaining smooth transitions in the data.

A custom scorer was defined to assist in selecting the best model for logistic regression. The custom scorer combines log loss, which measures the accuracy of the predictions, and a complexity term, which penalizes the number of non-zero coefficients. This approach helps favor models with fewer features, promoting a balance between ridge and lasso regression effects. The scorer equation is simply defined below:

$$Score = LogLoss + \lambda * Complexity \tag{27}$$

Where $LogLoss$ is the negative log-likelihood of the predictions, $Complexity$ is the number of non-zero coefficients in the model after regularization, and $\lambda$ is a weight hyperparameter that controls model complexity. For simplicity, $\lambda = 0.1$ was predefined, but it can also be optimized using a separate grid search or cross-validation to find the value that best balances prediction accuracy and model simplicity.

### 4.3.2 Random Forest

The data preprocessing steps applied to logistic regression are not needed for further models. Random forest and XGBoost can handle categorical variables natively by splitting data based on the categories during the tree construction process. These algorithms do not require explicit numerical encoding as they can directly process

categorical splits, simplifying the preprocessing pipeline. Random forest and XGBoost also do not require standardization because they are based on decision trees that are invariant to the scale of the features. These algorithms split data based on feature values, making them inherently robust to varying scales, thus eliminating the need for standardization.

A more diverse set of hyperparameters exist for random forest models as seen in Table 3. Each of these hyperparameters control the complexity of each tree in order to supervise the bias variance trade-off. Of the hyperparameters available, as seen from the *scikitlearn* documentation, 6 of the most significant were chosen. It is important to select log-loss as the chosen *criterion* in order to optimize the model based on the loss function earlier specified. The remaining set of hyperparameters either target similar complexity challenges, or are not relevant to this analysis. Scikit-learn, 2024.

### 4.3.3 XGBoost

The hyperparameters available to XGBoost are very similar to those available to random forest. The main difference can be seen in the regularization terms $\lambda$ and $\alpha$, as defined in equation 19. Short descriptions of each hyperparameter used are also available in Table 5.

# 5 Results

A careful view of the results is provided in this section, which builds a foundation for the conclusions and interpretations made in the conclusions.

## 5.1 Hyperparameter Results

For each set of unique parameters, log-loss performance in the test set is estimated using 5-fold cross validation. The model which maximizes the negative log loss is chosen as the "optimal" model.

### 5.1.1 Logistic Regression

Table 2 defines the parameters of such optimal model resulting from the model application process outlined in the previous section. Note that the $L_1$ ratio result states that absolute $L_1$ penalty is optimal here, which is expected as $L_1$ penalty includes every feature but with many features having near zero coefficients. This occurred despite the use of complexity regularization, which would benefit from gradient descent or grid search optimization on the complexity regularization term.

The only other parameter of note is the constant $C$, regularization strength.

### 5.1.2 Random Forest

The resulting parameters of this model limit the depth of the trees with $max_depth$ : 10, which helps to control overfitting by preventing the trees from growing too complex.

Setting $max_features$ to $sqrt$ means that at each split, the model considers a subset of features that is the square root of the total number of features. This approach helps reduce overfitting while ensuring that useful features are not overlooked. The parameters $min_samples_leaf : 5$ and $min_samples_split : 2$ control the tree's growth by setting minimum thresholds for the number of samples in each leaf and the number of samples required to split a node, respectively. This configuration helps create balanced trees that generalize better to new data. Finally, using 200 estimators $n_estimators : 200$ means the model combines the results of 200 individual trees, enhancing overall performance and robustness. Additionally, the criterion used for splitting nodes is $log_loss$, which focuses on minimizing the log loss during the training process.

### 5.1.3 XGBoost

The final parameters for the XGBoost model provide a well-tuned balance between complexity and performance. Setting *alpha* to 0.4 indicates that $L_1$ regularization is applied, which penalizes feature coefficients and helps prevent overfitting by encouraging sparsity in the model.

Using *gamma* set to 0.3 means that there is a minimum loss reduction required to make a further partition on a leaf node. This ensures that only significant splits are made, which

helps in reducing overfitting and improving model generalization. The *lambda* value of 1 introduces $L_2$ regularization, which penalizes large coefficients, further aiding in preventing overfitting.

A $learning_rate$ of 0.2 allows the model to update moderately fast, providing a good balance between the speed of learning and the stability of convergence, helping to avoid overshooting the optimal solution. The $max_depth$ of 4 ensures that

the trees are not too deep, which helps in maintaining generalization and avoiding overfitting by not capturing overly complex patterns. With $min_child_weight$ set to 8, the model requires a minimum sum of weights of 8 in each child, which helps in forming robust splits by ensuring that nodes are created only when they contain a sufficient number of samples.

Using 200 estimators $n_estimators$ 200 means that the model will build 200 trees, each contributing to the final prediction, enhancing the model's robustness and accuracy. Additionally, the parameter $enable_categorical$ set to True indicates that the model supports categorical variables, making it more flexible in handling different types of data. Finally, the *objective* set to $binary : logistic$ specifies that the model is optimized for binary classification tasks, focusing on predicting binary outcomes accurately.

## 5.2 Performance Metrics

To evaluate the performance of the models, various metrics were calculated. These metrics provide insights into the effectiveness and reliability of the models in predicting outcomes.

First, the top 25 feature importances are displayed in figures 5, 6, and 7. Note that in XGBoost, only 20 features are included as the regularization parameters reduced the amount of features down to 20.

AppliedAmount is the most significant parameter in both random forest and logistic regression. It also places as the fourth most significant parameter in XGBoost. This variable is defined as the amount supplied to the lender after loan approval. Note the diversity of the rest of the variables determined *important* between each model. XGBoost places more emphasis on usage, geographic, and language-related

variables, whereas Random Forest focuses more on direct financial metrics and historical loan data.

XGBoost achieved the lowest log loss value of 0.609, indicating it had the highest predictive accuracy among the models tested. This demonstrates XGBoost's robustness in handling the dataset and making accurate predictions. The Random Forest model also performed well, with a log loss of 0.611, showcasing its effectiveness, though slightly less so than XGBoost.

In contrast, the Logistic Regression model had a higher log loss of 0.663, suggesting it was less effective in predicting outcomes compared to the tree-based methods. This difference highlights the limitations of Logistic Regression in capturing the complexities of the data as effectively as the more sophisticated models.

Furthermore, it is worth noting that including more data from the dataset without random downsampling resulted in marginally better results for XGBoost. This suggests that XGBoost benefits from larger datasets, leveraging more information to improve its predictive performance. This finding emphasizes the importance of utilizing as much relevant data as possible to enhance model accuracy, particularly for advanced machine learning algorithms like XGBoost.

### 5.2.1 Confusion Matrices

Confusion matrices are presented for each model to show the true positives, false positives, true negatives, and false negatives. These matrices are instrumental in understanding how well the models perform in distinguishing between default and non-default cases.

Table 7 displays the confusion matrix for XGBoost. For XGBoost, the precision for the negative class (False) is 0.68, with a recall of 0.76 and an F1-score of 0.72, based on 2950 instances. In contrast, the positive class (True) has a lower precision of 0.60, recall of 0.50, and an F1-score of 0.55 from 2090 instances. The overall accuracy of the XGBoost model is 65%. The macro average precision is 0.64, recall is 0.63, and F1-score is 0.63, indicating a balanced performance across both classes. The weighted averages are consistent with the overall accuracy. This represents that the model's performance is stable across different metrics.

Logistic Regression shows a similar pattern, as seen in Table 8. The precision for class 0 is 0.66, with a recall of 0.74 and an F1-score of 0.70. For class 1, precision drops to 0.56, recall to 0.47, and the F1-score to 0.51. The accuracy of Logistic Regression is slightly lower at 63%. The macro averages for precision, recall, and F1-score are 0.61, 0.60, and 0.61 respectively, while the weighted averages are 0.62, 0.63, and 0.62. These metrics communicate that Logistic Regression, like XGBoost, performs better in identifying non-default cases but struggles with default cases.

Random Forest's performance, as seen in 9, mirrors that of Logistic Regression. The precision, recall, and F1-score for class 0 are 0.66, 0.74, and 0.70, respectively, while for class 1, these values are 0.56, 0.47, and 0.51. The overall accuracy of the Random Forest model is 63%, with macro averages for precision, recall, and F1-score at 0.61, 0.60, and 0.61, and weighted averages at 0.62, 0.63, and 0.62. This similarity in performance metrics between Random Forest and Logistic Regression suggests that both models capture similar patterns within the data and face the same challenges in distinguishing default cases.

# Conclusions

This thesis explored the effectiveness of three machine learning models—XGBoost, Logistic Regression, and Random Forest—in predicting loan defaults. Our findings shed light on the capabilities and limitations of these models in a practical credit risk assessment context.

A consistent trend across all models was their difficulty in accurately predicting defaults. While they were reliable at identifying borrowers unlikely to default, they were less effective at pinpointing those who would. This limitation is significant in the financial industry, where the ability to accurately predict defaults is crucial for risk management.

The evaluation of model performance through log loss reveals that XGBoost outperformed the other models, achieving the lowest log loss. This indicates its superior accuracy in predicting loan defaults. The Random Forest model also performed well, demonstrating its effectiveness but slightly lagging behind XGBoost. In contrast, the Logistic Regression model had a higher log loss suggesting it was less effective in handling the complexities of the dataset.

Several limitations in this study should be acknowledged. The imbalance in the dataset likely influenced the models, favoring predictions of non-defaults. Additionally, the models' struggle to accurately identify defaults suggests there might be room for improvement in feature selection or the application of more advanced modeling techniques.

Future research could benefit from addressing these issues by experimenting with techniques like oversampling the minority class, enhancing feature engineering

methods, or testing more advanced algorithms such as neural networks. Further analysis of different preprocessing steps and more thorough hyperparameter tuning could also lead to better model performance.

In conclusion, although XGBoost showed a slight edge in performance, all models faced challenges in accurately predicting loan defaults. Overcoming the identified limitations and exploring new methodologies will be key steps in advancing predictive accuracy in credit risk modeling.

# References


Breiman, Leo (2001). "Random forests". In: *Machine Learning* 45.1, pp. 5–32.

Breiman, Leo, Jerome Friedman, Charles J Stone, and Richard A Olshen (1984). *Classification and Regression Trees*. CRC press.

Chen, Tianqi and Carlos Guestrin (2016). "XGBoost: A scalable tree boosting system". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, pp. 785–794.

Cortes, Corinna and Vladimir Vapnik (1995). "Support-vector networks". In: *Machine Learning* 20.3, pp. 273–297.

Cox, David Roxbee (1958). "The regression analysis of binary sequences". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 20.2, pp. 215–232.

Freund, Yoav and Robert E Schapire (1997). "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of Computer and System Sciences* 55.1, pp. 119–139.

Friedman, Jerome H (2001). "Greedy function approximation: a gradient boosting machine". In: *Annals of Statistics* 29.5, pp. 1189–1232.

Hastie, Trevor, Robert Tibshirani, and Jerome Friedman (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics.

Ke, Guolin, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu (2017). "LightGBM: A highly efficient gradient boosting decision tree". In: *Advances in Neural Information Processing Systems*. Vol. 30, pp. 3146–3154.

Scikit-learn (2024). *sklearn.ensemble.RandomForestClassifier*. Accessed: 2024-05-22. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

# Appendix

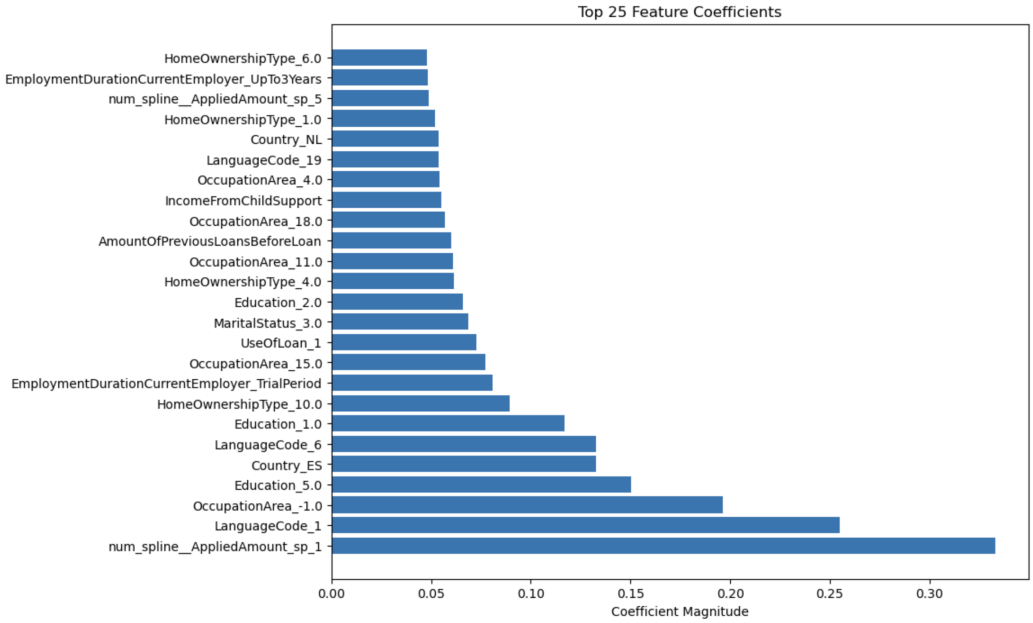


Figure 5: Feature importance in Logistic Regression, generated in Python using Matplotlib. This plot highlights the top 25 most significant features influencing the model's predictions.

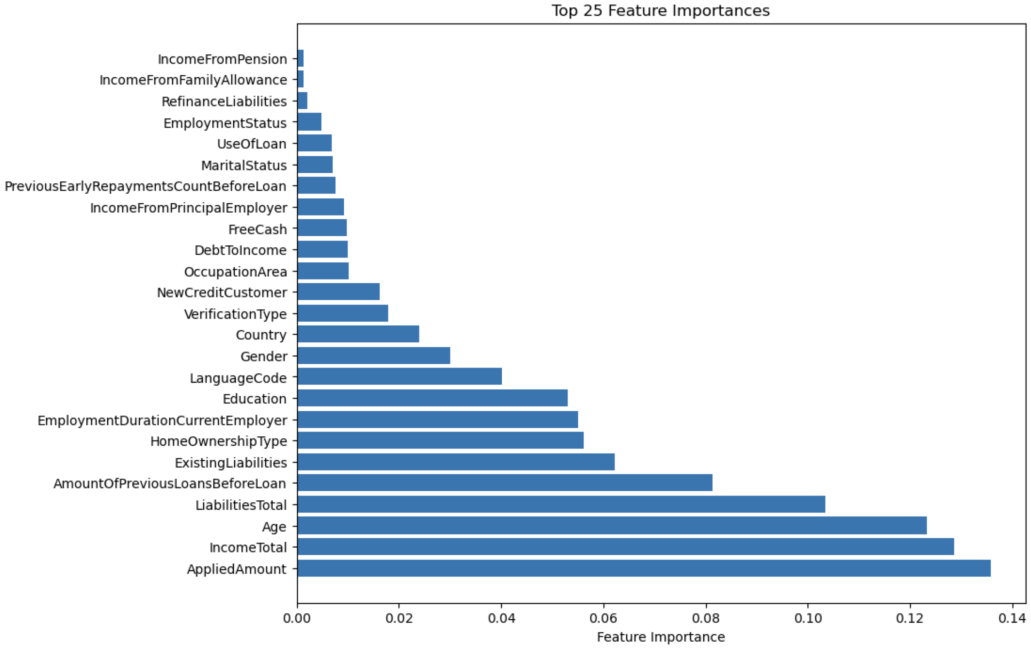


Figure 6: Feature importance in Random Forest, generated in Python using Matplotlib. The plot shows the importance of the top 25 features in the model, with higher values indicating greater importance.

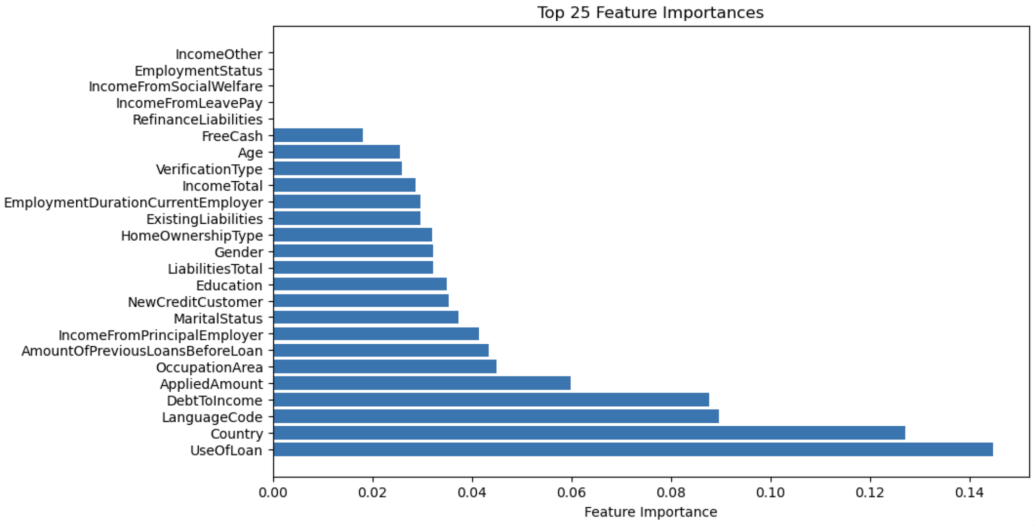


Figure 7: Feature importance in XGBoost, generated in Python using Matplotlib. This chart displays the top 25 features which contribute most significantly to the model's predictions.

## Tables

Table 1
Parameter Grid for Logistic Regression

| Parameter | Values | Description |
|---|---|---|
| `classifier__penalty` | `['elasticnet']` | Type of penalty |
| `classifier__l1_ratio` | `[0, 0.25, 0.5, 0.75, 1]` | Balance between L1 and L2 norms |
| `classifier__C` | `[0.1, 0.5, 1, 5, 10]` | Regularization strength |

Table 2
Best Parameters for Logistic Regression

| Parameter | Value |
|---|---|
| `classifier__C` | 10 |
| `classifier__l1_ratio` | 1 |
| `classifier__penalty` | elasticnet |

Table 3
Parameter Grid for Random Forest

| Parameter | Values | Description |
|---|---|---|
| `n_estimators` | `[100, 200, 300]` | Number of trees in the forest |
| `max_depth` | `[None, 5, 10]` | Maximum depth of each tree |
| `min_samples_split` | `[2, 5, 10]` | Min number of obs. placed in a node before split |
| `min_samples_leaf` | `[1, 2, 5]` | Min number of obs. allowed in a leaf node |
| `max_features` | `['sqrt', 'log2']` | The number of features considered for split |
| `criterion` | `['log_loss']` | Criterion used to determine best split |

Table 4
Best Parameters for Random Forest

| Parameter | Value |
|---|---|
| max_depth | 10 |
| max_features | sqrt |
| min_samples_leaf | 5 |
| min_samples_split | 2 |
| n_estimators | 200 |
| criterion | log_loss |

Table 5
Parameter Grid for XGBoost

| Parameter | Values | Description |
|---|---|---|
| n_estimators | [100, 200, 300] | Number of gradient boosted trees |
| learning_rate | [0.01, 0.1, 0.2] | Step size shrinkage to prevent overfitting |
| max_depth | [3, 7, 10] | Max depth of each tree |
| min_child_weight | [1, 3, 6] | Min sum of instance weight needed in a child |
| gamma | [0, 0.125, 0.2] | Min loss reduction to make a further partition |
| lambda | [1, 1.4, 2] | L2 regularization term on weights |
| alpha | [0, 0.6, 1] | L1 regularization term on weights |
| enable_categorical | [True] | Whether to enable categorical variable support |
| objective | ['binary:logistic'] | Loss function specifier |

Table 6
Best Parameters for XGBoost

| Parameter | Value |
|---|---|
| enable_categorical | True |
| gamma | 0.3 |
| learning_rate | 0.2 |
| max_depth | 4 |
| min_child_weight | 8 |
| n_estimators | 200 |
| alpha | 0.4 |
| lambda | 1 |
| objective | 'binary:logistic' |

Table 7
Confusion matrix for XGBoost, generated in Python.

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| False | 0.68 | 0.76 | 0.72 | 2950 |
| True | 0.60 | 0.50 | 0.55 | 2090 |
| Accuracy | | | 0.65 | 5040 |
| Macro Avg | 0.64 | 0.63 | 0.63 | 5040 |
| Weighted Avg | 0.65 | 0.65 | 0.65 | 5040 |

Table 8
Confusion matrix for Logistic Regression, generated in Python.

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.66 | 0.74 | 0.70 | 2950 |
| 1 | 0.56 | 0.47 | 0.51 | 2090 |
| Accuracy | | | 0.63 | 5040 |
| Macro Avg | 0.61 | 0.60 | 0.61 | 5040 |
| Weighted Avg | 0.62 | 0.63 | 0.62 | 5040 |

Table 9
Confusion matrix for Random Forest, generated in Python.

| | Precision | Recall | F1-Score | Support |
|---|---|---|---|---|
| 0 | 0.66 | 0.74 | 0.70 | 2950 |
| 1 | 0.56 | 0.47 | 0.51 | 2090 |
| Accuracy | | | 0.63 | 5040 |
| Macro Avg | 0.61 | 0.60 | 0.61 | 5040 |
| Weighted Avg | 0.62 | 0.63 | 0.62 | 5040 |

**Non-exclusive licence to reproduce thesis and make thesis public**

I, Grayson Felt,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, Predicting Loan Default with XGBoost: An Examination of Strength and Application, supervised by Raul Kangro.

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

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

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

Grayson Felt
22/05/2024