UNIVERSITÀ DEGLI STUDI DI TORINO

DEPARTMENT OF ECONOMICS, SOCIAL STUDIES, APPLIED MATHEMATICS AND STATISTICS

MASTER'S DEGREE COURSE IN

ECONOMICS

Comparative Analysis of Statistical Models for Computing Probability of Default: Application of Machine Learning Techniques in Python

Supervisor: Giordano Matteo

Co-Supervisor: Regis Luca

Candidate: Cristescu Adrian Iulian

Accademic Year:2023/2024

Table of Contents

Acknowledgments

First and foremost, I would like to express my deepest gratitude to **Professor Matteo Giordano** for his invaluable guidance and unwavering support throughout the course of my research. His meticulous attention to detail and his constant availability has played a pivotal role in the successful completion of this work. I am especially grateful for his willingness to embark on this project with me, which has been a source of both learning and personal growth.

I would also like to extend my heartfelt thanks to my mother, **Elena**, for being one of the few who consistently believed in me. Her confidence in my abilities, even at times when I doubted myself, has been a driving force behind my progress. Her unwavering emotional support has been instrumental in keeping me focused on my goals.

Finally, I would like to thank my fiancée, **Carlotta**, for her relentless encouragement. Her insistence that I persevere and complete this journey has been a constant source of motivation. Her emotional support and patience have been invaluable in helping me see this project through to the end.

To these three individuals, I owe a great deal of gratitude for their faith in me and for their contributions to this accomplishment. Thank you.

Chapter 1
Introduction

1.1 Objective and Importance

The primary objective of this thesis is to develop and evaluate machine learning models for predicting the probability of default (PD) of individuals on loans. Accurately forecasting credit risk is crucial for financial institutions, as it helps them make informed decisions about loan approvals, pricing, and capital reserves. This study compares six machine learning algorithms, ranging from traditional models like Logistic Regression to more advanced methods such as Random Forests and XGBoost, to determine which provides the best performance and interpretability.

The significance of this research lies in its contribution to both academic and practical fields. As machine learning models continue to evolve, their application in credit risk analytics offers the potential for improved accuracy and efficiency in default prediction, which is vital for minimizing financial losses and maintaining stability in lending markets. Furthermore, the emphasis on model interpretability ensures that these advanced techniques can be adopted in real-world, highly regulated environments.

1.2 Structure

This thesis is organized into eight chapters, each contributing to the progression of the research and findings on the prediction of loan default using machine learning models.

Following this introduction, Chapter 2 provides a comprehensive literature review that traces the evolution of credit risk prediction, focusing on traditional and modern techniques. It also critically examines existing studies, identifying gaps in model comparison, evaluation metrics, and dataset transparency. This chapter establishes the theoretical foundation for the subsequent research and highlights the need for more comparative analyses of models and evaluation metrics.

Chapter 3 outlines the methodology employed in this study, detailing the dataset, preprocessing techniques, and the six machine learning models used to predict default. This chapter explains the data preparation steps, including stratification, synthetic minority oversampling (SMOTE), and normalization, which ensure that the models are trained effectively. Each model—Logistic Regression, Random Forest, Gradient Boosting, XGBoost, Decision Tree, and AdaBoost—is discussed in detail, with particular focus on how the Probability of Default (PD) is computed for each. Additionally, cross-validation is explained as a crucial step in model validation.

In Chapter 4, the results of the analysis are presented, focusing on the performance of each model based on key evaluation metrics such as accuracy, recall, and F1-score. This chapter also delves into the comparison of model pairings, offering insights into which combinations of algorithms perform best.

Chapter 5 presents a discussion of the findings, providing interpretations and reflections on the implications of the results. It analyses the strengths and weaknesses of the models, addressing both their predictive power and interpretability, which are critical for real-world financial applications.

Chapter 6 synthesizes the insights derived from the research, particularly identifying the most effective models for default prediction and the significant predictive features. Section 5.3.2 articulates the contributions of this study to the existing body of knowledge, underscoring the efficacy of machine learning models in improving credit risk evaluation. Recommendations for practical application are discussed in Section 5.3.3, providing guidelines for institutions considering machine learning for default prediction. Section 5.3.4 suggests future research directions, emphasizing the integration of subjective behavioural data into models, which could be quantified and incorporated into risk assessments to improve prediction accuracy further. The chapter concludes with final observations on the broader impact of the findings.

Chapter 7: References and Chapter 8: Appendix provide essential supplementary material. The references section includes all sources cited, ensuring transparency and academic integrity, while the appendix offers additional details, such as dataset descriptions and code snippets that support reproducibility and extend the utility of the research.

Chapter 2
Literature Review

2.1 Introduction

The prediction of credit risk, specifically the probability of default (PD), is a crucial aspect of risk management for financial institutions. Credit risk refers to the possibility that a borrower will fail to meet their debt obligations, and accurately estimating this risk allows lenders to make informed decisions regarding loan approvals, pricing, and capital reserves. Over the years, numerous statistical and machine learning models have been developed to predict the likelihood of default, from traditional approaches like Logistic Regression to more advanced machine learning algorithms such as Random Forests and Gradient Boosting Machines.

The need for accurate prediction models stems from the financial implications of credit risk. According to Scheule, Roesch, and Baesens (Credit Risk Analytics: The R Companion), different borrower characteristics play varying roles in influencing the likelihood of default. This study emphasizes the importance of selecting relevant features and models that can capture the non-linear relationships present in the data. In the context of modern credit risk modelling, machine learning techniques are gaining traction due to their ability to model complex patterns in large datasets while also providing flexibility in handling diverse types of data.

Historically, Logistic Regression has been the predominant model for estimating the probability of default in credit risk analytics (Hosmer & Lemeshow, 2013). The popularity of this model lies in its interpretability and ease of use. Logistic regression provides a direct probability estimate based on a linear combination of features, where each feature's importance is determined by its associated coefficient. The model assumes a linear relationship between the input features and the log-odds of default.

This method has been widely adopted due to its transparency, allowing financial institutions to understand how individual factors such as income, credit history, and employment status affect default risk. However, despite its popularity, logistic regression has limitations when it comes to capturing non-linear relationships or interactions between variables. These limitations have led researchers to explore more sophisticated models capable of handling complex patterns in the data (Menard, 2002; Kleinbaum & Klein, 2010).

In recent years, machine learning models have increasingly been applied to credit risk analytics, offering significant improvements in accuracy and flexibility over traditional statistical methods (Breiman, 2001). Unlike logistic regression, machine learning algorithms such as Random Forests, Gradient Boosting Machines (GBM), and Extreme Gradient Boosting (XGBoost) can model complex, non-linear relationships and interactions between features, making them particularly useful for high-dimensional data.

Random Forests, as introduced by Breiman (2001), is an ensemble learning method that builds multiple decision trees on random subsets of the data and averages their predictions to reduce overfitting and improve accuracy. This model is especially advantageous in credit risk because it can capture subtle interactions between borrower features that may not be obvious with simpler models.

Gradient Boosting and XGBoost (Friedman, 2001; Chen & Guestrin, 2016) take this idea further by sequentially building trees, where each new tree corrects the errors of the previous ones. These methods tend to outperform other models in terms of predictive accuracy, especially when fine-tuned through techniques like grid search. As with Random Forests, Gradient Boosting and XGBoost can model complex relationships in credit risk datasets, making them ideal for predicting the probability of default.

Furthermore, the ability of machine learning models to assign varying importance to different features is particularly valuable in credit risk modeling. In line with findings from Scheule, Roesch, and Baesens, machine learning algorithms automatically learn which features (e.g., debt-to-income ratio, credit score, employment history) carry more weight in determining the likelihood of default. This feature importance is key to refining lending criteria and improving the robustness of credit risk models.

One challenge in adopting machine learning for credit risk analysis is balancing accuracy and interpretability. While traditional models like logistic regression offer clear explanations of feature contributions, more complex models such as Random Forests and Gradient Boosting are often seen as "black boxes." However, recent advancements have made it possible to interpret machine learning models through feature importance metrics, partial dependence plots, and SHAP (SHapley Additive exPlanations) values (James et al., 2013). These methods allow credit risk practitioners to better understand how specific features influence the predicted probability of default, a critical factor when deploying models in highly regulated environments like finance.

In practice, certain features have been found to consistently play a larger role in predicting default risk. For instance, Scheule, Roesch, and Baesens highlight that financial ratios, such as debt-to-income and credit utilization, as well as borrower-specific attributes, like credit score and employment history, tend to have higher predictive power in determining default risk. These features are often given greater importance by machine learning algorithms, a finding that aligns with existing research on the determinants of credit risk (Hosmer & Lemeshow, 2013; Breiman, 2001).

Despite the advances in machine learning models for credit risk prediction, there remain areas that require further investigation. While machine learning algorithms have demonstrated superior performance in predicting defaults, the interpretability of these models, particularly in highly regulated sectors like finance, remains an ongoing challenge. This thesis aims to explore the application of these advanced models in predicting the probability of default, while also examining the trade-offs between model accuracy and interpretability. Furthermore, by analyzing the importance of individual borrower features, this study seeks to contribute to a more nuanced understanding of credit risk analytics.

One critical factor in credit risk modelling is the reliability and transparency of the data used for training and evaluating predictive models. High-quality data, sourced from credible institutions, is essential for building models that can be confidently applied in real-world scenarios. In many cases, however, existing literature either does not fully disclose the origin of the dataset used or relies on proprietary datasets, which makes it difficult to assess the generalizability of the results or to replicate the findings.

In this thesis, I utilize a dataset sourced from Scheule, Roesch, and Baesens (Credit Risk Analytics: The R Companion), which is a well-established and reliable source in the field of credit risk analytics. The transparency of the dataset not only ensures that the results are reproducible but also adds robustness to the analysis by providing clear documentation of the features and variables involved. This transparency contrasts with other studies that fail to disclose their data sources, potentially raising concerns about the validity of their findings.

2.2 Comparison to existing work

In the domain of credit risk prediction, the use of machine learning models has grown rapidly in recent years, offering significant improvements in accuracy and flexibility over traditional statistical methods. However, despite these advancements, several critical issues remain in the existing body of work, particularly regarding model comparison, the use of appropriate evaluation metrics, and dataset transparency.

One of the primary concerns in the literature is the lack of transparency concerning the datasets used. Many papers fail to fully disclose the origin of their data or rely on proprietary datasets, which hampers reproducibility and generalizability. In particular, Noriega et al. (2023) and Guegan & Hassani (2018) highlight that while credit risk modelling has benefited from public datasets, the consistent failure to clearly describe or disclose dataset origins creates significant challenges for replication and comparative studies. This lack of transparency prevents researchers from properly assessing the representativeness of the data and the broader applicability of the models. Invented datasets are frequently used in published papers, a fact that can be identified by examining the results of analyses where, despite minimal preprocessing, algorithms achieve improbably high accuracy in predicting the results.

Another key limitation in existing studies is the lack of comprehensive model comparisons. Historically, most studies have focused on a single model or small sets of models, typically using Logistic Regression or similar statistical methods for credit risk prediction. While models such as Random Forests and Gradient Boosting are becoming more common, there are few papers that compare a wide range of machine learning models in a systematic way.

Recent research has started to address this gap. Hamzic et al. (2023) and Singh et al. (2023) have conducted comparative studies involving multiple models, including XGBoost, Random Forest, KNN, and Decision Trees, and evaluated them on loan default prediction. These studies show that machine learning models generally outperform

traditional approaches like Logistic Regression, particularly on more complex datasets. However, comprehensive comparisons that include a broad range of models remain relatively rare in the literature.

While accuracy has traditionally been the primary metric for evaluating model performance in credit risk prediction, it is increasingly recognized that accuracy alone is insufficient, particularly when dealing with imbalanced datasets where non-defaults vastly outnumber defaults. In such cases, models can achieve high accuracy simply by predicting most cases as non-defaults, but this overlooks the critical task of identifying true defaulters.

Several recent studies, such as Noriega et al. (2023) and Hamzic et al. (2023), have emphasized the importance of using more nuanced metrics such as recall, precision, F1-score, and AUC to provide a more comprehensive evaluation of model performance. These metrics are particularly important in credit risk modelling, where the cost of false negatives (failing to predict a default) is much higher than the cost of false positives (incorrectly predicting a default).

A final critical issue in the existing literature is the trade-off between model performance and interpretability. While advanced machine learning models like Random Forests and Gradient Boosting offer superior predictive performance, they are often criticized for being "black boxes" that do not offer clear insights into which features drive their predictions.

In contrast, Scheule, Roesch, and Baesens have emphasized the importance of interpretable models in financial decision-making. While they acknowledge the potential of machine learning, they also emphasize that understanding the key drivers of default risk is crucial for building trust in model predictions and ensuring compliance with financial regulations. Some recent studies, such as Guegan & Hassani (2018), have also started using techniques like feature importance rankings and SHAP values to improve the interpretability of machine learning models.

2.3 Summary and outlook

In summary, the literature on credit risk prediction has evolved significantly, with both traditional statistical models like Logistic Regression and more advanced machine learning algorithms such as Random Forests, Gradient Boosting Machines, and XGBoost demonstrating their respective strengths and limitations. Logistic Regression remains popular due to its interpretability, yet its limitations in capturing non-linear relationships have prompted researchers to explore more complex models. Machine learning techniques have gained traction because of their ability to model intricate patterns and interactions between features, leading to improved predictive performance.

However, despite these advancements, several challenges persist in the literature. Data transparency remains an issue, as many studies fail to disclose their data sources, which limits the replicability of results and the generalizability of models. Moreover, the focus

on accuracy as the primary evaluation metric is increasingly seen as insufficient, particularly in the context of imbalanced datasets where identifying defaulters (recall) is crucial. Newer studies have emphasized the importance of alternative metrics such as recall, precision, and F1-score, which provide a more comprehensive assessment of model performance.

Furthermore, there is a notable lack of comprehensive model comparisons in the literature. Although recent studies have begun to compare multiple models, including newer machine learning algorithms, such comparisons remain limited. This thesis addresses this gap by comparing six machine learning models and prioritizing both accuracy and interpretability, ensuring that the models not only perform well but also provide insights into the factors driving predictions.

By using a transparent and reliable dataset, emphasizing model interpretability through feature importance, and focusing on multiple evaluation metrics, this thesis contributes to the ongoing effort to refine credit risk prediction models, offering both practical and academic value to the field.

Chapter 3
Methodology

3.1 Dataset Description

The considered dataset is provided by the book "*H. Scheule, D. Roesch, B. Baesens, Credit Risk Analytics: The R Companion, Scheule Roesch Baesens"(2017)*[1].

The dataset contains 5960 observations, 3364 of which are complete. The dependent variable indicating the defaulters/non-defaulters is:

BAD: 1 = applicant defaulted on loan or seriously delinquent; 0 = applicant paid loan

The regressors used for the analysis are the following:

LOAN: Amount of the loan request

MORTDUE: Amount due on existing mortgage

VALUE: Value of current property

REASON: DebtCon = debt consolidation / HomeImp = home improvement

JOB: Occupational categories

YOJ: Years at present job

DEROG: Number of major derogatory reports

DELINQ: Number of delinquent credit lines

CLAGE: Age of oldest credit line in months

NINQ: Number of recent credit inquiries

CLNO: Number of credit lines

DEBTINC: Debt-to-income ratio

The following table presents the first 20 rows of the dataset, offering a quick overview of the data structure and content.

[1]The website containing all the information about the book and about the authors is (creditriskanalytics.net)

BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
1	$1,700.00	$ 30,548.00	$ 40,320.00	HomeImp	Other	9	0	0	101.47	1	8	37.11
1	$1,800.00	$ 28,502.00	$ 43,034.00	HomeImp	Other	11	0	0	88.77	0	8	36.88
0	$2,300.00	$102,370.00	$120,953.00	HomeImp	Office	2	0	0	90.99	0	13	31.59
1	$2,400.00	$ 34,863.00	$ 47,471.00	HomeImp	Mgr	12	0	0	70.49	1	21	38.26
0	$2,400.00	$ 98,449.00	$117,195.00	HomeImp	Office	4	0	0	93.81	0	13	29.68
0	$2,900.00	$103,949.00	$112,505.00	HomeImp	Office	1	0	0	96.10	0	13	30.05
0	$2,900.00	$104,373.00	$120,702.00	HomeImp	Office	2	0	0	101.54	0	13	29.92
1	$2,900.00	$ 7,750.00	$ 67,996.00	HomeImp	Other	16	3	0	122.20	2	8	36.21
1	$2,900.00	$ 61,962.00	$ 70,915.00	DebtCon	Mgr	2	0	0	282.80	3	37	49.21
0	$3,000.00	$104,570.00	$121,729.00	HomeImp	Office	2	0	0	85.88	0	14	32.06
0	$3,200.00	$ 74,864.00	$ 87,266.00	HomeImp	ProfExe	7	0	0	250.63	0	12	42.91
1	$3,300.00	$130,518.00	$164,317.00	DebtCon	Other	9	0	6	192.29	0	33	35.73
0	$3,600.00	$100,693.00	$114,743.00	HomeImp	Office	6	0	0	88.47	0	14	29.39
0	$3,600.00	$ 52,337.00	$ 63,989.00	HomeImp	Office	20	0	0	204.27	0	20	20.47
1	$3,700.00	$ 17,857.00	$ 21,144.00	HomeImp	Other	5	0	0	129.72	1	9	26.63
0	$3,800.00	$ 51,180.00	$ 63,459.00	HomeImp	Office	20	0	0	203.75	0	20	20.07
1	$3,900.00	$ 29,896.00	$ 45,960.00	HomeImp	Other	11	0	0	146.12	0	14	24.48
0	$3,900.00	$102,143.00	$118,742.00	HomeImp	Office	2	0	0	85.28	0	13	29.34
0	$4,000.00	$105,164.00	$112,774.00	HomeImp	Office	1	0	0	94.72	0	13	29.39
0	$4,000.00	$ 54,543.00	$ 61,777.00	HomeImp	Office	21	0	0	205.59	0	19	21.81

Table 1: Sample of the dataset used in this study.

3.2 Data Preprocessing

As mentioned, the dataset contains missing values. After removing incomplete observations, 3364 complete records remain. The dataset is highly imbalanced, with 11% defaults and 89% non-defaults, and features like income, age, and title vary in scale. To address these issues, SMOTE is used to handle class imbalance, improving predictive performance for the minority class, which is more critical. Additionally, normalization is applied to ensure that features with different ranges and units contribute equally to the final prediction.

3.2.1 Stratify

Stratification is a technique used to ensure that the training, test, and validation sets are representative of the entire dataset. This is achieved by maintaining the proportional representation of different classes across these sets, mirroring their distribution in the original dataset.

By following the original proportions rather than splitting the data randomly, stratification prevents scenarios where one set might consist entirely of one class, such as all default or all non-default observations. This approach ensures that the model is trained and tested on data that accurately reflects the overall class distribution.

3.2.2 SMOTE - Synthetic Minority Over-sampling Technique

SMOTE generates synthetic instances for the minority class, but it does not merely duplicate existing observations. Instead, it first randomly selects a minority class instance $\boldsymbol{a}$ and identifies its $\boldsymbol{k}$ nearest neighbors within the minority class. A synthetic instance is then created by randomly choosing one of these $\boldsymbol{k}$ nearest neighbors, $\boldsymbol{b}$, and forming a line segment between $\boldsymbol{a}$ and $\boldsymbol{b}$ in the feature space. The synthetic instance is generated as a convex combination of the two selected instances $\boldsymbol{a}$ and $\boldsymbol{b}$.

The ***k***-nearest neighbors of ***a*** are determined by calculating the Euclidean distance between ***a*** and other minority class instances ***b*** in the dataset. The synthetic point is then computed using the following formula:

$$s = a + u \cdot (b - a).$$

s = synthetic point

a = minority class of which the model searches the k-nearest

b = k-nearest

u = random between 0 and 1

In the analysis, a specialized version of SMOTE is used: Borderline SMOTE. This variant is particularly effective in addressing the misclassification problem for minority class instances located near the decision boundary between classes. These instances are challenging to classify and have a higher likelihood of being misclassified by models. Borderline SMOTE concentrates on generating synthetic instances close to the borderline between minority and majority classes, focusing on the most difficult-to-classify cases. When SMOTE is applied, the model's ability to correctly classify minority class instances, especially those near the decision boundary, improves significantly, by generating synthetic instances of the minority class, particularly focusing on those near the decision boundary between classes.

3.2.3 Normalization

Machine learning algorithms are often trained with the assumption that all features contribute equally to the final prediction. However, this assumption fails when the features differ in range and unit, hence affecting their importance.

Hence, normalization is a vital step in data preprocessing that ensures uniformity of the numerical magnitudes of features. This uniformity avoids the domination of features that have larger values compared to the other features or variables.

3.3 Data Splitting

After the removal of incomplete data, it is required to split the dataset into different datasets. They are needed to train, validate and test the models.

3.3.1 Train Set

This represents the training set used in the machine learning process. It consists of the data from which the model learns underlying patterns, relationships, and features, providing the foundation for the predictive analysis.

3.3.2 Validation Set

The validation set is a subset of the data used to provide an unbiased evaluation of the model during hyperparameter tuning. It serves as a checkpoint for assessing model performance without altering the learning process. As hyperparameters are optimized, the risk of overfitting on this set increases, potentially making the evaluation less generalizable.

Hyperparameters are settings that influence the learning process and control how the model is trained, such as learning rate, optimization algorithms, tree count, and leaf number. They are crucial in determining the model's ability to generalize effectively.

3.3.3 Test Set

The test set is a sample of data used to provide an unbiased evaluation of the final model's performance after training. This set is critical for assessing how well the model generalizes to unseen data, which ensures its reliability in real-world applications. Importantly, no information in the test set should be used during the model training process. The test set is where the model's learning is objectively validated, ensuring that the model's predictions are not influenced by prior exposure to this data.

3.4 Models

This analysis includes the results of different models. This section defines and explain these models.

3.4.1 XGB – Extreme Gradient Boosting

Model and Parameters

XGBoost[2] is used for supervised learning problems, where we use the training data (with multiple features) x_i to predict a target variable y_i.

The model in supervised learning usually refers to the mathematical structure by which the prediction y_i is made from the input x_i . A common example is a *linear model*, where the prediction is given as $\hat{y}_i = \sum_j \theta_j x_{ij}$,a linear combination of weighted input features. The prediction value can have different interpretations, depending on the task, i.e., regression or classification. For example, it can be logistic transformed to get the probability of positive class in logistic regression, and it can also be used as a ranking score when we want to rank the outputs.

The parameters are the undetermined part that we need to learn from data. In linear regression problems, the parameters are the coefficients θ. Usually, we will use θ to denote the parameters (there are many parameters in a model, our definition here is sloppy).

[2] Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785-794.

Objective Function: Training Loss + Regularization

With judicious choices for y_i, we may express a variety of tasks, such as regression, classification, and ranking. The task of training the model amounts to finding the best parameters θ that best fit the training data x_i and labels y_i. To train the model, we need to define the objective function to measure how well the model fit the training data.

A salient characteristic of objective functions is that they consist of two parts: training loss and regularization term:

$$obj(\theta) = L(\theta) + \Omega(\theta).$$

where **L** is the training loss function, and **Ω** is the regularization term, which controls the complexity of the model, helping against overfitting. The training loss measures how predictive our model is with respect to the training data. A common choice of **L** is the mean squared error, which is given by

$$L(\theta) = \sum_i (y_y - \hat{y}_i)^2.$$

Tree Boosting

The learning of the trees starts by defining an objective function and optimize it:

$$obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \omega(f_i).$$

What the model needs to learn are the functions (f_i), each containing the structure of the tree and leaf scores. It is intractable to learn all the trees at once. Instead, it is fixed what's learned and a new tree is added at each step. The prediction value at step t is $\hat{y}_i^{(t)}$, after which we have:

$$\hat{y}_i^{(0)} = 0.$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i).$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i).$$

…

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i).$$

Given the above, the wanted tree to add at each step is the one that optimizes the objective function.

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \omega(f_i).$$

$$obj^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)}) + f_t(x_i) + \omega(f_i) + constant.$$

And considering the MSE (mean squared error) as the loss function, we have:

$$obj^{(t)} = \sum_{i=1}^{n} (y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)))^2 + \sum_{i=1}^{t} \omega(f_i).$$

$$obj^{(t)} = \sum_{i=1}^{n} \left[2\left(\hat{y}_i^{(t-1)} - y_i\right) f_t(x_i) + f_t(x_i)^2\right] + \omega(f_t) + constant.$$

Model Complexity

We have now to define the complexity of the tree $\omega(f_t)$. First, we refine the definition of the tree $f(x)$ as

$$f_t(x) = w_{q(x)}, \qquad w \in R^T, \qquad q: R^d \to \{1,2,\dots,T\}.$$

Here, w is the vector of scores on leaves, $\boldsymbol{q}$ is a function assigning each data point to the corresponding leaf and $\boldsymbol{T}$ is the number of leaves. For XGB, the complexity is:

$$\omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2.$$

After re-formulating the tree model, we can write the objective value with the t-th tree as:

$$obj^{(t)} \approx \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i) w_j + \lambda) w_j^2 \right] + \gamma T.$$

Where $I_j = \{i | q(x_i) = j\}$ is the set of indices of data points assigned to the j-th leaf. Further compressing the expression, we have:

$$obj^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T.$$

Where w_j are independent with respect to each other, and the best w_j for a given structure q(x) and the best objective reduction we can get is:

$$w_j^* = -\frac{G_j}{H_j + \lambda}.$$

$$obj^{(*)} = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T.$$

The obj* equation measures how good a tree structure q(x) is.

Regularization

The used parameters in the coding phase are the following:

- Learning Rate (learning_rate). It controls the step size of each iteration while moving toward a minimum of the loss function
    - Lower rate means the model learns slowly which often results in a more robust model, but it may require more boosting rounds, increasing the computational intensity
- Number of Estimators (n_estimators). It's the number of trees. Each tree is sequentially added to correct the errors of the previous one
    - Lower number of estimators means that the model is faster but may not capture the underlying patterns in the data
    - Higher values allow the model to learn more complex patterns and correct errors better but increase the risk of overfitting.
- Maximum Depth (max_depth). It controls how deep the trees can grow, allowing the model to capture more complex patterns.
    - Lower depth can capture only the most significant patterns but may miss the capture of more complex interactions.
    - Higher depth leads the capture of more complex interactions between features but also increases the risk of overfitting where there is noise or irrelevant features

To conclude, XGBoost effectively enhances prediction accuracy by iteratively refining its models through gradient-boosting, making it a powerful tool for both classification and regression tasks.

3.4.2 RF – Random Forest

Random forests[3] or random decision forests is an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time. For classification tasks, the output of the random forest is the class selected by most trees. For regression tasks, the mean or average prediction of the individual trees is returned. Random decision forests correct for decision trees' habit of overfitting their training set.

Training algorithm

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, \dots, x_n$ with responses $Y = y_1, \dots, y_n$ bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b=1, …, B:

1) Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2) Train a classification or regression tree f_b on X_b, Y_b.

[3] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f}(x') = \frac{1}{B}\sum_{b=1}^{B} f_b(x').$$

Or by taking the majority vote in the case of classification trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

Additionally, an estimate of the uncertainty of the prediction can be made as the standard deviation of the predictions from all the individual regression trees on x′:

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}.$$

From bagging to random forest

Random forests also include another type of bagging scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated.

Typically, for a classification problem with p features, √p (rounded down) features are used in each split. For regression problems the inventors[4] recommend p/3 (rounded down) with a minimum node size of 5 as the default. In practice, the best values for these parameters should be tuned on a case-to-case basis for every problem.

Properties

Random forests can be used to rank the importance of variables in a regression or classification problem in a natural way. The following technique was described in Breiman's original paper and is implemented in the R package randomForest.

To measure variable importance in a dataset $D_n = \{(X_i, Y_i)\}_{i=1}^{n}$, a random forest is fitted, and the out-of-bag error is recorded. The importance of each feature is assessed by permuting the feature's values, recalculating the out-of-bag error, and observing the change. Features causing larger error increases are more important. However, this method can be biased towards categorical variables with more levels, struggle with correlated

[4] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5-32.

features, and may miss important features when collinearity is present. Techniques like partial permutations and grouping correlated features can address these issues.

Mean Decrease in Impurity Feature Importance

This feature importance for random forests is the default implementation in sci-kit learn and R. It is described in the book "Classification and Regression Trees" by Leo Breiman. Variables which decrease the impurity during splits a lot are considered important:

$$unormalized\ average\ importance(x) = \frac{1}{n_T}\sum_{i=1}^{n_T}\sum_{node\ j \in T_i | split\ variables\ (j)=x} p_{Ti}(j)\Delta i_{Ti}(j).$$

x = a feature

n_T = number of trees in the forest

T_i = tree i

$p_{Ti}(j) = \frac{n_j}{n}$ is the fraction of samples reaching node j

$\Delta i_{Ti}(j)$ = change of impurity in tree **t** at node **j**. As impurity measure for samples falling in a node e.g. the following statistics can be used: *entropy*, *Gini coefficient*, *MSE*.

The sci-kit learn default implementation of Mean Decrease in Impurity Importance is susceptible to misleading feature importances:

- The importance measure prefers high cardinality features
- It uses training statistics and therefore does not "reflect the ability of feature to be useful to make predictions that generalize to the test set

Nearest neighbours

A relationship between random forests and the k-nearest neighbour algorithm (k-NN) was pointed out by Lin and Jeon in 2002. It turns out that both can be viewed as so-called weighted neighbourhoods' schemes. These are models built from a training set $\{(x_i, y_i)\}_{i=1}^{n}$ that makes predictions $\hat{y}$ for the new points x' by looking at the "neighbourhood" of the point, formalized by a weight function W:

$$\hat{y} = \sum_{i=1}^{n} W(x_i, x')y_i.$$

Here, $W(x_i, x')$ is the non-negative weight of the i-th training point relative to the new point x' in the same tree. For any particular x', the weights for points x_i, must sum to one. Weight functions are given as follows:

- In the k-NN, the weights are , $W(x_i, x') = \frac{1}{k}$ if x_i is one of the k points closest to x', and zero otherwise.

- In a tree, $W(x_i, x') = \frac{1}{k'}$ if x_i is one of the k' points in the same leaf as x', and zero otherwise.

Since a forest averages the predictions of a set of m trees with individual weight functions W_j, its predictions are:

$$\hat{y} = \frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{n} W_j(x_i, x')y_i = \sum_{i=1}^{n}\left(\frac{1}{m}\sum_{j=1}^{m} W_j(x_i, x')\right)y_i .$$

This shows that the whole forest is again a weighted neighbourhood scheme, with weights that average those of the individual trees. The neighbours of **x'** in this interpretation are the points $\mathbf{x_i}$ sharing the same leaf in any tree **j**. In this way, the neighbourhood of x' depends in a complex way on the structure of the trees, and thus on the structure of the training set. Lin and Jeon show that the shape of the neighbourhood used by a random forest adapts to the local importance of each feature.

Regularization

Hyperparameter tuning is crucial for optimizing the performance, hence the following ones are used in the current analysis:

- Number of trees (n_estimators). Each tree contributes to the final prediction through majority voting in the case of classification.
    - Low number of estimators may lead to underfitting
    - High number of trees improves the stability and accuracy but increases the computational cost
- Maximum Depth of Trees (max_depth). Determines how detailed the trees can be
    - Low values can cause underfitting
    - High values can cause overfitting by capturing noise in the training data
- Minimum Samples to Split a Node (min_samples_split). It influences how deep the trees can grow
    - Low values make the trees deeper which may create overfitting
    - High values lead to lower trees, reducing the variance but may cause underfitting
- Minimum samples per Leaf (min_samples_leaf). It affects the number of samples that must be in a leaf node.
    - Low values can lead to overfitting by creating overly specific rules
    - High values force the model to be more generalized, improving performance on unseen data

To conclude, Random Forest is a powerful ensemble method in classification tasks, leveraging the combined predictions of multiple decision trees to enhance accuracy and robustness. By averaging these predictions, random forests reduce overfitting and improve generalization. Additionally, they provide valuable insights into feature importance, making them particularly effective for complex, high-dimensional datasets.

3.4.3 LR – Logistic regression

In statistics, the logistic model[5] (or logit model) models the log-odds of an event as a linear combination of independent variables. In binary logistic regression, the dependent variable is binary (0 or 1), while independent variables can be either binary or continuous. The probability of the event labelled "1" ranges from 0 to 1, converted from log-odds via the logistic function. The log-odds scale is measured in logits, giving rise to the model's name.

Supervised machine learning

Logistic regression is a supervised machine learning algorithm commonly used for binary classification tasks, such as predicting whether a loan will default or not. It uses the logistic (or sigmoid) function to convert a linear combination of input features into a probability value between 0 and 1, representing the likelihood of default. The logistic function's S-shaped curve effectively maps any real number to this probability range, making logistic regression ideal for accurately modelling binary outcomes and supporting informed decision-making in classification tasks.

Model

The logistic function has the following form:

$$p(x) = \frac{1}{1 + e^{\frac{-(x-\mu)}{s}}}.$$

Where:

- μ is the location parameter (midpoint of the curve -> p(μ) = ½)
- s is a scale parameter

The function can be rewritten as:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}.$$

Where:

- β_0 = -μ/s which is the intercept
- β_1 = 1/s which is the rate parameter

Fit

In logistic regression, the goodness of fit is measured using logistic loss (log loss), which is the negative log-likelihood. For a given x_k and y_k, where p_k=$p(x_k)$, the log loss for the k-th point ℓ_k is:

$$\ell_k = \begin{cases} -\ln p_k & if\ y_k = 1 \\ -\ln(1 - p_k) & if\ y_k = 0. \end{cases}$$

[5] Hosmer, D.W., Lemeshow, S., & Sturdivant, R.X. (2013). Applied Logistic Regression (3rd ed.). Wiley.

Log loss quantifies how "surprising" the actual outcome y_k is relative to the predicted probability p_k. It's always greater than or equal to 0, reaching 0 only for a perfect prediction. As the prediction worsens (i.e., when p_k approaches 0 for y_k =1 or 1 for y_k =0), the log loss increases towards infinity. Unlike linear regression, where zero loss is possible if all points lie on a line, logistic regression's log loss remains strictly positive since probabilities p_k are bounded between 0 and 1.

We can rewrite as a unique function:

$$\ell_k = -y_k \cdot \ln(p_k) - (1 - p_k) \cdot \ln(1 - p_k).$$

This is the cross-entropy of the predicted distribution $(p_k, (1 - p_k))$ from the actual distribution $(y_k, (1 - y_k))$, as probability distributions on the two-element space of (default, non-default).

The sum of these, the total loss, is the overall negative log-likelihood $-\ell$, and the best fit is obtained for those choices of β_0 and β_1 for which $-\ell$ is minimized.

Parameter estimation

Since ℓ is nonlinear in β_0 and β_1, determining their optimum values will require numerical methods. One method of maximizing ℓ is to require its derivatives with respect to β_0 and β_1 to be zero:

$$0 = \frac{\partial \ell}{\partial \beta_0} = \sum_{k=1}^{K} (y_k - p_k).$$

$$0 = \frac{\partial \ell}{\partial \beta_1} = \sum_{k=1}^{K} (y_k - p_k) x_k.$$

And for a n-th β_n:

$$0 = \frac{\partial \ell}{\partial \beta_n} = \sum_{k=1}^{K} (y_k - p_k) x_{kn}.$$

Solving the two equations for β_0 and β_1 will give maximize ℓ.

Interpretation of the Coefficients

The coefficients (β) provide insights into the relationship between the independent variables (predictors) and the dependent variable (outcome). The log-odds of the outcome are modelled by the logistic regression:

$$logit(p) = ln\left(\frac{p}{1 - p}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \cdots + \beta_n x_n.$$

Here, $\left(\frac{p}{1-p}\right)$ is the odds of the outcome occurring, and *logit(p)* is the natural logarithm of the odds (log-odds)

- β_0 is the intercept, when all predictors are zero.
- $\beta_1, \ldots, \beta_2$ are the coefficients for each predictor variable. They indicate how much the log-odds of the outcome change with a one-unit increase in the predictor, holding all other variables constant.

Interpretation of the Odds Ratio

By exponentiating the coefficients, we get a good interpretation of the odds-ratio:

$$Odds\ Ratio = e^{\beta n}.$$

The odds-ratio represents the multiplicative change in the odds of the outcome for a one-unit increase in the predictor x_n. For example, if $e^{\beta 1}$ =2, a one-unit increase in x_1 doubles the odds of the outcome occurring:

- if $e^{\beta n} > 1$ then the predictor increases the odds of the outcome
- if $e^{\beta n} < 1$then the predictor decreases the odds of the outcome
- if $e^{\beta n} = 1$ the predictor has no effect on the odds

Significance of Coefficients

Each coefficient is associated with a standard error and a hypothesis test can determine whether the coefficient is significantly different from zero.

p – value = indicates whether the coefficient is statistically significant. If it is below a certain threshold, normally 0.05, then the coefficient is considered significantly different from zero, meaning that the predictor has a significant effect on the outcome.

Regularization

To prevent overfitting, some regularization is required. By adding a penalty to the cost function, it discourages large coefficients and therefore reduce the model complexity.

In the present analysis two types were considered, but only L2 made it to the end, since L1 was never among the winning combinations.

- L1 regularization (Lasso):
  - The penalty term is : $\lambda \sum_{n=1}^{N} |\beta_n|$
  - It encourages sparsity by shrinking some coefficients to zero, effectively selecting important features
- L2 Regularization (Ridge):
  - The penalty term is : $\lambda \sum_{n=1}^{N} \beta_n^2$
  - It shrinks coefficients towards zero without eliminating them, reducing the impact of less important features.

Choosing λ controls the strength of the regularization:

- High λ -> more regularization and more shrinkage

- Low λ -> less regularization and closer fit to the training data
- λ = 0 means no regularization, equivalent to standard logistic regression

In the current analysis, during the coding phase, the parameter C is used. To be considered as:

$$C = \frac{1}{\lambda}.$$

To conclude, logistic regression is a robust and widely used method for binary classification that effectively balances simplicity and interpretability with predictive power. Its popularity and straightforward interpretation favour it when conducting classification analysis.

3.4.4 DT – Decision Tree

A decision tree[6] is a hierarchical model that uses a tree-like structure to represent decisions and their possible outcomes, including event outcomes, costs, and utilities. Each internal node tests an attribute, each branch shows the result, and each leaf node represents a class label or decision. The paths from root to leaf represent classification rules. Decision trees are commonly used as visual and analytical tools to calculate the expected values or utilities of different alternatives.

There are two approaches for decision tree: Univariate decision tree and multivariate decision tree. The current analysis used the univariate decision tree for classification since they provide clearer insights into the relationship between specific features and the likelihood of default. The algorithm used is CART.

CART – Classification and Regression trees

It was introduced by Breiman in 1984. CART algorithm builds both classification and regression trees. The classification tree is constructed by CART by the binary splitting of the attribute. Gini Index is used as selecting the splitting attribute. The CART is also used for regression analysis with the help of regression tree. The regression feature of CART can be used in forecasting a dependent variable given a set of predictor variable over a given period. CART have an average speed of processing and supports both continuous and nominal attribute data.

The model

The CART (Classification and Regression Trees) algorithm builds decision trees by recursively splitting the data into two branches at each node. The goal is to create the "purest" nodes possible, where the target variable is predominantly one class (for classification) or has minimal variance (for regression).

In classification tasks, CART uses the Gini Index to measure the "impurity" of a node. The Gini Index for a node t is calculated as:

[6] Breiman, L., Friedman, J., Stone, C.J., & Olshen, R.A. (1984). Classification and Regression Trees. Chapman & Hall/CRC.

$$Gini(t) = 1 - \sum_{i=1}^{C} p_i^2.$$

Where:

- C is the number of classes
- P_i is the proportion of observations of class i in node p

A Gini Index of 0 indicates a "pure" node, where all instances belong to one class. CART selects splits that minimize the Gini Index, creating purer nodes as tree grows.

Parameter Estimation

The Splitting Criterion is based on the feature that results in the largest reduction in the Gini Index. This reduction is called Gini Fain, and it is calculated as:

$$\Delta Gini = Gini(parent) - \left(\frac{N_{left}}{N_{total}} Gini(left) + \frac{N_{right}}{N_{total}} Gini(right)\right).$$

Where:

- Gini(parent) is the Gini Index of the parent node
- Gini(left) and Gini(right) are the Gini Indices of the left and right child nodes
- N_{LEFT} and N_{RIGHT} are the number of observations in the left and right nodes
- N_{TOTAL} is the number of observations in the parent node

The feature and threshold that result in the highest Gini Gain are chosen to split the node.

Tree Pruning

To avoid overfitting, CART uses pruning techniques to simplify the tree after it has been fully grown. Pruning removes branches that do not provide significant improvements in the classification accuracy, making the tree more generalizable to unseen data.

CART typically employs cost-complexity pruning, where a penalty is added for the complexity (size) of the tree. The cost-complexity function is defined as:

$$R_\alpha(T) = R(T) + \alpha|T|.$$

Where:

- R(T) is the misclassification rate of the tree T
- |T| is the number of terminal nodes (leaf nodes)
- α is the complexity parameter that controls the trade-off between the size of the tree and its accuracy

Higher values of α result in simpler, smaller trees, while lower values allow more complex trees to be maintained. Pruning helps to create a more balanced model by preventing overfitting and improving generalization on new data.

Regularization

To prevent overfitting and control the complexity of the decision tree, regularization techniques are applied through parameters such as Max Depth(max_depth),Minimum Samples to Split (min_samples_split), and Minimum Samples per Leaf (min_samples_leaf). These parameters ensure that the tree does not grow too complex, which could lead to overfitting by capturing noise in the training data.

- Max Depth. Controls how deep the tree can grow, balancing the model's ability to capture complexity with the risk of overfitting.
    - High Values – tree grows deeper, capturing more complex data but increases the risk of overfitting by becoming too specific to the training data.
    - Low Values – limits the tree's growth, reducing the risk of overfitting but may lead to underfitting, as the model may not capture important patterns in the data.
- Minimum Samples to Split. Determines the minimum number of samples required to split a node, helping to manage the tree's complexity and avoid overfitting.
    - High values – the nodes have more data before splitting, reducing the tree's depth and complexity and it prevents overfitting by ensuring that splits only happen when there's sufficient data.
    - Low values allow the tree to split more frequently, capturing finer details in the data but increases the risk of overfitting as the model becomes sensitive to small fluctuations in the data.
- Minimum Samples per Leaf. Ensures that each leaf node contains enough samples, preventing the model from becoming too specific and improving generalization.
    - High values allow each leaf to have more samples, making the model more robust and reducing overfitting but it can prevent the model from learning specific nuances of the training data, leading to underfitting.
    - Low values allow leaf nodes to be more specific, capturing small patterns in the data but increases the risk of overfitting by learning from noises in the training data.

In conclusion, decision trees for classification are powerful tools for modelling binary or multi-class outcomes by recursively splitting the data based on feature values. With proper tuning of parameters like maximum depth and minimum samples per leaf, they effectively balance complexity and generalization. This makes them highly interpretable while providing reliable predictions for classification tasks.

3.4.5 ADA – AdaBoost

AdaBoost[7], short for Adaptive Boosting, is a statistical classification meta-algorithm formulated by Yoav Freund and Robert Schapire in 1995, who won the 2003 Gödel Prize

[7] Freund, Y., & Schapire, R.E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences, 55(1), 119-139.

for their work. It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. Usually, AdaBoost is presented for binary classification, although it can be generalized to multiple classes or bounded intervals on the real line.

Algorithm

For a dataset with N samples, the process initializes with the weight of each data point with $w_i = \frac{1}{N}$. For m=1 to M:

- Fit a classifier $T^{(m)}(x)$ using all the training data using weights w_i
- Compute $err^{(m)} = \frac{\sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(x_i)\right)}{\sum_{i=1}^{n} w_i}$, where :
    - c_i is the ground truth value of the target variable
    - $w_i^{(m)}$ is the weight of the sample i at iteration m
    - $x_i$is the feature vector for the i-th observation, which is used to predict the label
    - $\mathbb{I}$ is the indicator function: 1 is true, 0 otherwise
- Compute $\alpha^{(m)} = log\frac{1-err^{(m)}}{err^{(m)}}$
- Re-normalize w_i

New predictions computed by $C(x) = arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}\left(T^{(m)}(x) = k\right)$.

Statistical understanding of boosting

Boosting is a form of linear regression in which the features of each sample x_i are the outputs of some weak learner h applied to x_i.

While regression tries to fit F(x) to y(x) as precisely as possible without loss of generalization using the least square error (most of the times) $E(f) = (y(x) - f(x))^2$, whereas the AdaBoost error function $E(f) = e^{-y(x)f(x)}$ takes into account the fact that only the sign of the final result is used, thus $|E(f)|$ can be far larger than 1 without increasing error. However, the exponential increase in the error for sample x_i as $-y(x_i)f(x_i)$ increases, resulting in excessive weights being assigned to outliers.

One feature of the choice of exponential error function is that the error of the final additive model is the product of the error of each stage, that is, $e^{\sum_i -y_i f(x_i)} = \prod_i e^{-y_i f(x_i)}$. Thus, it can be seen that the weight update in the AdaBoost algorithm is equivalent to recalculating the error on $F_t(x)$ after each stage.

There is a lot of flexibility allowed in the choice of loss function. If the loss function is monotonic and continuously differentiable, the classifier is always driven towards purer solutions. Zhang (2004) provides a loss function based on least squares, a modified Huber loss function:

$$\phi\big(y, f(x)\big) = \begin{cases} -4yf(x) & if\ yf(x) < -1, \\ (yf(x) - 1)^2 & if\ -1 \leq yf(x) \leq 1 \\ 0 & if\ yf(x) > 1. \end{cases}$$

Where:

- $\phi\big(y, f(x)\big)$ is the loss function
- y is the true label of the observation
- $f(x)$ is the prediction of the classifier for input x
- $yf(x)$ is the margin. If both have the same sign, then the prediction is correct.
    - $if > 1$ the classifier is confident hence correct
    - $if < 1$ the classifier is not so confident hence incorrect

This function is more well-behaved than LogitBoost for $f(x)$ close to 1 or -1, does not penalise 'overconfident' predictions ($yf(x) > 1$), unlike unmodified least squares, and only penalises samples misclassified with confidence greater than 1 linearly, as opposed to quadratically or exponentially, and is thus less susceptible to the effects of outliers.

Boosting as gradient descent

Boosting can be understood as minimizing a convex loss function over a set of weak learners. Specifically, the loss function minimized by AdaBoost is the exponential loss:

$$\sum_i \phi(i, y_i, f) = \sum_i e^{-y_i f(x_i)}.$$

In this context, y_i is the true label and $f(x_i)$ is the classifier's prediction for the i-th training sample. The exponential loss penalizes misclassified points more heavily, which driver the boosting algorithm to focus on difficult or incorrectly classified examples in subsequent iterations.

In the gradient descent analogy, the output of the classifier for each training point is treated as a point $(F_t(x_1), \ldots, F_t(x_n)$ in n-dimensional space, where each weak learner $h(x)$ corresponds to a vector with fixed orientation and length. The goal is to move towards the target point $(y_1, \ldots, y_n)$ in the fewest steps by reducing the loss at each iteration.

Thus, **AdaBoost** can be interpreted as performing gradient descent to minimize classification error, choosing $h(x)$ with the steepest gradient that minimizes the test error or optimizing for training error at each step.

Regularization

AdaBoost is designed to work with weak learners, which are typically very simple models like decision stumps (i.e., decision trees with a single split). These weak learners are not meant to be highly complex models themselves; instead, AdaBoost works by combining many of these simple models to form a strong ensemble.

Because the default weak learners are decision stumps (trees with a max_depth of 1), max_depth, min_samples_split, and min_samples_leaf are not as relevant or critical when using AdaBoost with decision stumps as the base learner.

Therefore, different parameters are used here:

- Number of estimators (n_estimators). This parameter controls the number of weak learners (e.g., decision stumps) that are added sequentially to the model.
    - High values: allows the model to fit more complex patterns in the data, potentially improving accuracy but lead to overfitting by capturing noises and specific idiosyncrasies in the training data
    - Low values: leads to a simpler model, which may underfit the data and not capture all relevant patterns.
- Learning Rate (learning_rate): controls the contribution of each weak learner to the final ensemble prediction. Essentially, it shrinks the impact of each estimator on the final prediction.
    - High values: each estimator has a larger impact on the final model. While this can speed up convergence, it may also cause the model to overfit quickly, as each weak learner has more influence on the final prediction.
    - reduces the influence of each weak learner, making the model more stable and less prone to overfitting.

In conclusion, AdaBoost is a powerful ensemble learning algorithm that iteratively boosts the performance of weak learners, such as decision stumps, by focusing on misclassified instances.

3.4.6 GB – Gradient Boosting

Gradient boosting[8] is a machine learning technique that boosts in a functional space, targeting pseudo-residuals instead of traditional residuals. It builds an ensemble of weak prediction models, typically simple decision trees, known as gradient-boosted trees. Unlike other boosting methods, it allows optimization of any differentiable loss function and is built in a stage-wise manner.

Algorithm

We introduce the model by explaining the least-squares regression setting, in which the goal is to teach a model F to predict values of the form $\hat{y} = F(x)$ by minimizing the MSE $\frac{1}{n}\sum_i(\hat{y}_i - y_i)^2$, where i indexes over some training set of size n of actual values of the output variable y:

- $\hat{y}$ is the predicted value $F(x_i)$
- y_i is the observed value

[8] Friedman, J.H. (2001). Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29(5), 1189-1232.

- n is the number of samples in *y*.

Let's consider now a gradient boosting algorithm with M stages. At each stage m ($1 \leq m \leq M$) of gradient boosting, suppose some imperfect model F_m, our algorithm should add some new estimator, $h_m(x)$, having:

$$F_{m+1}(x_i) = F_m(x_i) + h_m(x_i) = y_i.$$

Or equivalently:

$$h_m(x_i) = y_i - F_m(x_i).$$

The gradient boosting will then fit h_m to the residual $y_i - F_m(x_i)$. As in the other boosting variants, each F_{m+1} attempts to correct the errors of its predecessor F_m.

A generalization of this concept to loss functions beyond squared error, as well as to classification and ranking problems, stems from the insight that the residuals $h_m(x_i)$ for a given model are proportional to the negative gradients of the MSE loss function with respect to $F(x_i)$:

$$L_{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - F(x_i))^2 - \frac{\partial L_{MSE}}{\partial F(x_i)} = \frac{2}{n}\big(y_i - F(x_i)\big) = \frac{2}{n}h_m(x_i).$$

Gradient boosting could be specialized to a gradient descent algorithm, and generalizing it entails "plugging in" a different loss and its gradient.

In many supervised machine learning problems, there is an output variable *y* and a vector of input variables *x*, related to each other with some probabilistic distribution. The goal is to find some function $\hat{F}(x)$ that best approximates the output variable from the values of input variables. This is accomplished by the loss function $L(y, F(x))$ which must be minimized in its expectation:

$$\hat{F} = \underset{F}{\operatorname{argmin}}\, \mathbb{E}_{x,y}[L(y, F(x))].$$

The gradient boosting method assumes a real-valued *y*. It seeks an approximation $\hat{F}(x)$ in the form of a weighted sum of M functions $h_m(x)$ from some class $\mathcal{H}$, called weak learners:

$$\hat{F}(x) = \sum_{m=1}^{M} \gamma_m h_m(x) + const.$$

Given the training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ of known sample values of x and corresponding values of y, the method seeks to find an approximation $\hat{F}(x)$ that minimizes the average loss function value on the training set, thereby reducing the empirical risk. This is achieved by initially starting with a model that consists of a constant function $F_0(x)$ and then incrementally expanding the model in a greedy manner:

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma).$$

$$F_m(x) = F_{m-1}(x) + \left(\underset{\gamma}{\operatorname{argmin}} \left[\sum_{i=1}^{n} L\big(y_i, F_{m-1}(x_i) + h_m(x_i)\big) \right]_{h_m \in \mathcal{H}} \right)(x).$$

for $m \geq 1$ where $h_m \in \mathcal{H}$ is a base (weak) learner function.

Since choosing the best function h_m at each step for an arbitrary loss function L is a computationally very difficult optimization problem, a more restricted approach is to be applied. The idea is to apply the steepest descend step to find a local minimum of the loss function by iteration on $F_{m-1}(x)$.

Moving then a small amount γ such that the linear approximation remains valid:

$$F_m(x) = F_{m-1}(x) - \gamma \sum_{i=1}^{n} \nabla_{F_{m-1}} L\big(y_i, F_{m-1}(x_i)\big).$$

Where $\gamma > 0$. For small γ, this implies that $L(y_i, F_m(x_i)) \leq L(y_i, F_{m-1}(x_i))$

Furthermore, by finding the γ for which the loss function has a minimum, γ is optimized:

$$\gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F_m(x_i)) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) - \gamma \nabla_{F_{m-1}} L\big(y_i, F_{m-1}(x_i)\big)).$$

Considering the continuous case, we have:

$$F_m(x) = F_{m-1}(x) - \gamma_m \sum_{i=1}^{n} \nabla_{F_{m-1}} L\big(y_i, F_{m-1}(x_i)\big).$$

where γ_m is the step length, defined above.

Gradient tree boosting

Gradient boosting is typically used with decision trees (especially CARTs) of a fixed size as base learners. For this special case, Friedman[9] proposes a modification to gradient boosting method which improves the quality of fit of each base learner.

Generic gradient boosting at the *m*-th step would fit a decision tree $h_m(x)$ to pseudo-residuals. Let J_m be the number of its leaves. The tree partitions the input space into J_m disjoint regions $R_{1m}, \ldots, R_{J_m m}$ and predicts a constant value in each region. Using the indicator notion, the output of $h_m(x)$ for input x can be written as the sum:

[9] Friedman, J. H. (2001). "Greedy Function Approximation: A Gradient Boosting Machine." *Annals of Statistics*, 29(5), 1189-1232.

$$h_m(x) = \sum_{j=1}^{J_m} b_{jm} 1_{R_{jm}}(x).$$

where $b_{jm}$is the value predicted in the region R_{jm}.

The coefficients b_{jm} are multiplied by some value γ_{jm}, chosen using line search so as to minimize the loss function, and the model is updated as follows:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x), \ \ where \ \gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

Friedman proposes to modify this algorithm so that it chooses a separate optimal value γ_{jm} for each of the tree's regions, instead of a single value γ_{jm} for the whole tree. He calls the modified algorithm "TreeBoost". The coefficients b_{jm} from the tree-fitting procedure can be then simply discarded and the model update rule becomes:

$$F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} 1_{R_{jm}}(x), \ \ where \ \gamma_{jm}$$
$$= \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}}^{n} L(y_i, F_{m-1}(x_i) + \gamma).$$

Regularization

The parameters used in the present analysis are the following:

- Number of estimators (n_estimators). Controls the number of weak learners (typically decision trees) that are added to the ensemble.
    - High values: allows the model to capture more complex patterns and relationships improving accuracy but there's a risk of overfitting
    - Low values: Leads to a simpler model but may underfit the data
- Learning Rate (learning_rate). This parameter controls how much each weak learner contributes to the final ensemble prediction.
    - High values: causes each estimator to have a larger impact on the final model but may cause overfitting.
    - Low values: reduces the influence of each weak learner but requires more estimators to achieve a better performance.
- Max Depth (max_depth). This parameter limits the maximum depth of the decision trees used as weak learners.
    - High values: Allows each tree to learn more detailed and complex patterns in the data but can increase the risk of overfitting if the tree becomes too complex.
    - Low values: Limits the complexity of each tree, which can help prevent overfitting but may also lead to underfitting if important patterns are missed.

In conclusion, gradient boosting is a powerful ensemble learning technique that iteratively improves model performance by focusing on misclassified samples. By carefully tuning regularization parameters such as the number of estimators, learning rate, and max depth, the model can balance complexity and generalization, preventing overfitting while capturing meaningful patterns.

3.5 Probability of Default – How it is computed

Every model has its own way of computing the probability of default. Following it will be shown how each of the models computes the predicted probability of default.

3.5.1 Extreme Gradient Boosting

XGBoost is a highly optimized version of Gradient Boosting with advanced regularization and efficiency techniques. It still follows the same fundamental boosting process, where trees are added sequentially to correct previous errors.

The probability of default is also computed by[10]:

$$\hat{y}(x) = \sum_{m=1}^{M} \gamma_m h_m(x).$$

And for the classification case, it's also true that:

$$P(default = 1|X) = \frac{1}{1 + e^{-\hat{y}(x)}}.$$

The difference is that XGBoost introduces regularization terms to control the complexity of the model:

$$L(\theta) = \sum_{i=1}^{N} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(h_k).$$

Where $\Omega(h_k) = \gamma T + \frac{1}{2}\lambda\|w\|^2$ is a regularization term on the number of leaves T and the leaf weights w, helping to prevent overfitting.

3.5.2 Random Forest

Random Forest is an ensemble of decision trees, where each tree is trained on a random subset of the data (bagging) and a random subset of features. Each decision tree independently classifies the data, and the final prediction is based on the majority vote or average of the individual trees.

[10] Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785-794.

The probability of default is then computed by[11]:

$$P(default = 1|X) = \frac{1}{N}\sum_{n=1}^{N} P_{tree_i}(default = 1|X).$$

Where P_{tree_i} is the probability predicted by the i-th tree while the overall probability is the average of the probabilities across all trees. Each tree provides a binary prediction (0 or 1) and the final probability is the fraction of trees prediction default.

3.5.3 Logistic Regression

Logistic Regression uses the logistic function to compute the probability of default based on a linear combination of input features.

The equation is[12]:

$$P(default = 1|X) = \frac{1}{1 + e^{-(\beta_0+\beta_1 X_1+\cdots+\beta_n X_n)}}.$$

The coefficients$\beta_0, \beta_1, .., \beta_n$ are learned during the training through the Maximum Likelihood Estimation, which aims to find the parameter values (coefficients) that maximize the likelihood of observing the given data.

3.5.4 Decision Tree

A Decision Tree splits the data recursively based on feature values that maximize the information gain (or minimize the Gini impurity). The tree continues to split until a stopping criterion is met (e.g., a maximum depth or minimum samples per leaf).

The probability of default is then computed like:

$$P(default = 1|X) = \frac{Number\ of\ default\ samples\ in\ the\ leaf}{Total\ number\ of\ samples\ in\ the\ leaf}.$$

The probability of default is simply computed as the fraction of samples in the leaf node that are in the "default" class.

3.5.5 AdaBoost

AdaBoost is a boosting algorithm that trains weak learners (usually decision tree stumps) in sequence. Each new learner is trained on the misclassified samples from the previous learners, with increased weight given to misclassified samples.

The probability of default is computed in the following way[13]:

[11] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5–32.
[12] Hosmer, D.W., Lemeshow, S., & Sturdivant, R.X. (2013). *Applied Logistic Regression*.
[13] Freund, Y., & Schapire, R. E. (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting"

$$P(default = 1|X) = \frac{1}{1 + e^{-\sum_{m=1}^{M} \alpha_m h_m(x)}}.$$

Where α_m is the weight assigned to the m-th weak learner and $h_m(x)$ is the prediction (1 or 0) from the weak learner. The output is passed through a sigmoid to compute the final probability.

3.5.6 Gradient Boosting

Gradient Boosting is an ensemble method where models are added sequentially to reduce the errors of the previous models. Each model (usually a decision tree) is fit on the residuals (errors) of the previous model, rather than the original target variable.

The probability of default is computed in the following way:

$$\hat{y}(x) = \sum_{m=1}^{M} \gamma_m h_m(x).$$

Where $h_m(x)$ is the prediction of the m-th decision tree (weak learner) and γ_m is the weight assigned to that tree. The final prediction is the sum of these prediction.

In the case of classification tasks, $\hat{y}(x)$ (the prediction) is transformed into a probability using the logistic function[14]:

$$P(default = 1|X) = \frac{1}{1 + e^{-\hat{y}(x)}}.$$

The model then minimizes the following cross-entropy loss:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)].$$

Where p_i is the predicted probability and y_i is the actual label.

To conclude, the computation is done in the following way:

- Logistic Regression: it directly uses the logistic function to transform the linear model's output into a probability
- Gradient Boosting / XGBoost / AdaBoost: the sum of weak learners' predictions is passed through a sigmoid function to produce probabilities. Gradient Boosting and XGBoost add a loss minimization process using gradient descent.
- Random Forest: the probability is the average of the individual trees' predictions (it is the fraction of trees predicting default)
- Decision Tree: the probability is the proportion of default samples in the leaf node where the sample ends up.

[14] Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 29(5), 1189–1232.

3.6 Cross Validation

To evaluate and optimize the performance of the machine learning models used in this thesis, 10-fold-cross validation (CV) was implemented during the training and hyperparameter tuning phases. Cross-validation is a robust technique that helps avoid overfitting by training the model on multiple subsets of the data and testing it on unseen subsets. This ensures that the model's performance is generalizable to new, unseen data, and not just tailored to a specific train-test split.

In 10-fold cross-validation, the dataset is divided into 10 equally sized folds. For each iteration, the model is trained on 9 folds and tested on the remaining fold, with the process repeated 10 times, each time with a different test fold. The final performance metric is averaged over all 10 iterations to provide a reliable estimate of model performance.

I employed grid search with cross-validation to tune hyperparameters for each model. By using cross-validation during grid search, I ensured that the optimal hyperparameters were chosen based on robust performance across multiple data splits, thus reducing the likelihood of overfitting to a particular training set.

The consistent use of cv=10 across all models allowed for a fair comparison of model performance and ensured that the models were evaluated on the same data splits.

While cross-validation provides more reliable estimates of model performance, it also increases computational cost, as the model is trained multiple times. However, this trade-off is essential for ensuring that the model's performance is not biased by a single train-test split.

3.7 Performance Evaluation

A solid analysis must be comprehensive of multiple evaluations and metrics; hence, the following tests will be applied: Recall, Precision, F1-score, AUC and ROC.

3.7.1 Recall – True Positive Rate

Recall also known as sensitivity, measures the ability of a model to identify true positives, i.e., defaulters correctly predicted as defaulters. It is defined as the ratio of correctly predicted positive instances to all actual positive instances[15]:

$$Recall = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Negatives\ (FN)}.$$

High recall indicates that the model has a strong ability to capture actual positives but may include false positives.

[15] Powers, D. M. W. (2011). "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness, and correlation." *Journal of Machine Learning Technologies*, 2(1), 37-63.

3.7.2 Precision

Precision is the ratio of true positive predictions to the total number of positive predictions made by the model. It evaluates the accuracy of the positive class predictions[16]:

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Positives\ (FP)}.$$

High precision indicates that fewer irrelevant or false positives were predicted.

3.7.3 F1-Score

The F1-score is the harmonic mean of precision and recall. It provides a balanced measure when both precision and recall are equally important. The formula is[17]:

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}.$$

A high F1-score indicates a balance between low false positives and low false negatives.

3.7.4 ROC Curve (Receiver Operating Characteristic Curve)

The ROC curve plots the True Positive Rate (Recall) against the False Positive Rate across different thresholds. It provides a visual tool to assess the trade-offs between sensitivity and specificity[18].

$$False\ Positive\ Rate = \frac{False\ Positives\ (FP)}{True\ Negatives\ (TN) + False\ Positives\ (FP)}.$$

A higher ROC curve indicates a better-performing model.

3.7.5 AUC (Area Under the Curve)

AUC measures the area under the ROC curve and evaluates the performance of a classifier by determining its ability to distinguish between classes. AUC ranges from 0.5 (random chance) to 1 (perfect classifier)[19].

To conclude, is it important to mention that the test Accuracy wasn't considered here since the dataset is very unbalanced and it may give misleading results.

---

[16] Sokolova, M., & Lapalme, G. (2009). "A systematic analysis of performance measures for classification tasks." *Information Processing & Management*, 45(4), 427-437.
[17] Sokolova, M., & Lapalme, G. (2009). "A systematic analysis of performance measures for classification tasks." *Information Processing & Management*, 45(4), 427-437.
[18] Fawcett, T. (2006). "An introduction to ROC analysis." *Pattern Recognition Letters*, 27(8), 861-874.
[19] Bradley, A. P. (1997). "The use of the area under the ROC curve in the evaluation of machine learning algorithms." *Pattern Recognition*, 30(7), 1145-1159.

3.7.6. Accuracy

Accuracy is one of the most commonly used metrics for evaluating the performance of classification models. It measures the proportion of correctly predicted instances out of the total instances in the dataset. Mathematically, accuracy is defined as[20]:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}.$$

While accuracy provides a straightforward measure of model performance, it can be misleading when used with imbalanced datasets.

[20] Powers, D. M. W. (2011). "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness, and correlation." *Journal of Machine Learning Technologies*, 2(1), 37-63.

Chapter 4
Results

This chapter presents the outcomes of the machine learning models used to predict the probability of default, comparing their performance based on several evaluation metrics. The six models—Logistic Regression, Random Forest, Gradient Boosting, XGBoost, Decision Tree, and AdaBoost—are evaluated on their ability to predict defaults accurately. The key metrics such as accuracy, recall, precision, F1-score, AUC, and the ROC curve that have been defined in the previous chapter are used to assess the effectiveness of each model. The results highlight the strengths and weaknesses of each algorithm in terms of both predictive power and model interpretability.

4.1 Descriptive information

4.1.1 General Information

The following table presents the key descriptive statistics of the dataset, offering an initial overview of the data that will be analysed:

Statistic	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
count	3364	3364	3364	3364	3364	3364	3364	3364	3364	3364	3364	3364	3364
mean	0.09	19154.4	76249.62	107501.39	0.3	1.45	9.11	0.15	0.28	180.99	1.04	22.11	34.14
std	0.29	10875.42	45095.37	54728.24	0.46	1.41	7.6	0.58	0.81	82.77	1.55	9.39	7.95
min	0	1700	5076	21144	0	0	0	0	0	0.49	0	0	0.84
25%	0	12000	49351.25	71235	0	0	3	0	0	118.69	0	16	29.36
50%	0	17000	67278.5	94453.5	0	1	7	0	0	176.74	1	21	35.13
75%	0	23825	92986.75	122339.25	1	3	13	0	0	230.4	2	27	39.09
max	1	89900	399412	512650	1	5	41	10	10	1168.23	13	64	144.19

Table 2. Summary statistics of the dataset showing key metrics

From Table 2, it can be observed that the average loan request is approximately $19,000, while individuals have around $76,000 in existing mortgages, with properties valued at an average of $107,000. Notably, 29.5% of borrowers sought loans for home improvement, while the remainder requested funds for debt consolidation. The average borrower has been employed at their current job for around 9 years.

Furthermore, borrowers report an average of 0.14 major derogatory reports and 0.27 delinquent credit lines. The oldest credit line is typically 180 months (15 years) old, with 1 recent credit inquiry on average. The mean debt-to-income ratio is 34/1, reflecting the financial leverage of the individuals in this dataset.

Another crucial information is the number of defaulters compared to total number of individuals. In the dataset there are 300 individuals which defaulted and 3064 which

successfully paid what's due:

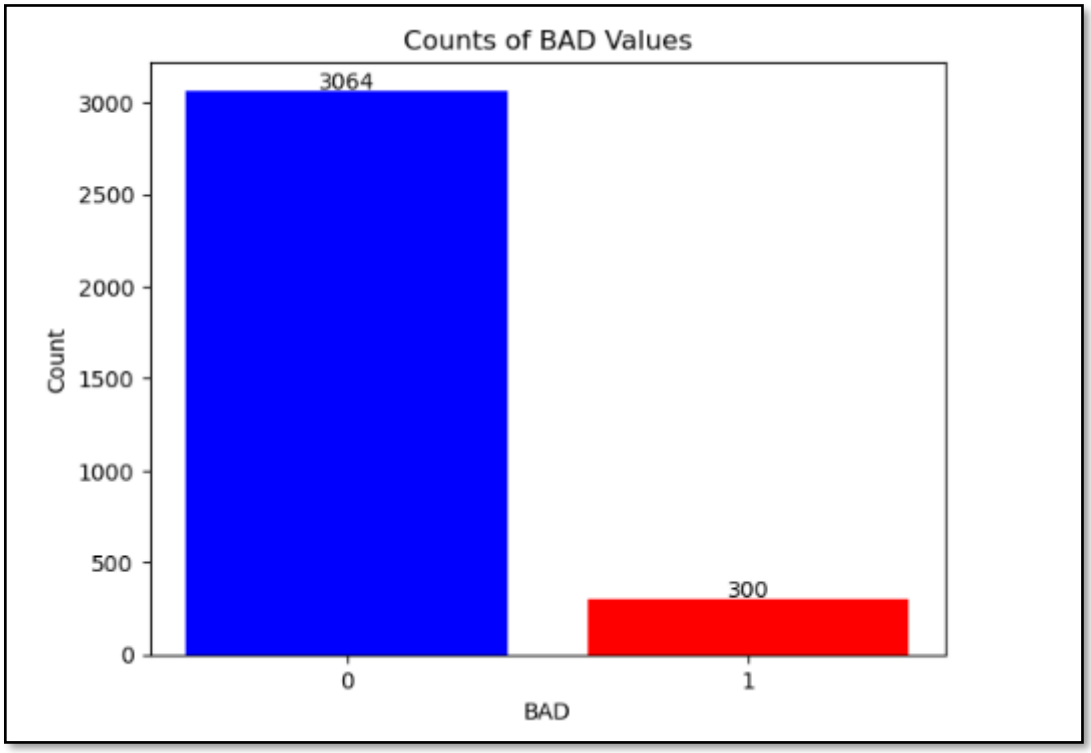

Figure 1: Distribution of Defaulters vs Non-Defaulters

4.1.2 Correlation Matrix

A correlation matrix is a tool that shows the pairwise correlation coefficients between variables. These coefficients measure the strength and direction of a linear relationship between variables, ranging from -1 (perfect negative correlation) to 1 (perfect positive correlation), with 0 indicating no correlation. The matrix is critical for identifying relationships between variables and detecting potential issues like multicollinearity, which can affect the performance and interpretability of statistical models, particularly Logistic Regression.

In machine learning, multicollinearity arises when two or more independent variables are highly correlated, meaning they carry redundant information. This can be problematic in models like logistic regression, where high multicollinearity inflates the variance of coefficient estimates, making it difficult to determine the true effect of each independent variable. Multicollinearity can lead to unstable models, where small changes in the data can lead to large variations in the estimated coefficients.

Also, particularly when using models like Random Forests or XGBoost, multicollinearity is less of an issue because these models can handle correlated features more effectively due to their non-parametric nature. However, understanding the relationships between variables remains important for model interpretability and performance optimization.

	BAD	LOAN	MORTDUE	VALUE	REASON	JOB	YOJ	DEROG	DELINQ	CLAGE	NINQ	CLNO	DEBTINC
BAD	1.00	-0.03	-0.02	-0.03	-0.01	-0.01	-0.06	0.25	0.27	-0.11	0.13	0.01	0.22
LOAN	-0.03	1.00	0.27	0.38	-0.18	0.15	0.08	-0.02	-0.07	0.07	0.05	0.08	0.16
MORTDUE	-0.02	0.27	1.00	0.87	0.01	0.32	-0.10	-0.04	-0.04	0.14	0.04	0.32	0.19
VALUE	-0.03	0.38	0.87	1.00	0.04	0.35	-0.02	-0.05	-0.07	0.20	-0.01	0.22	0.13
REASON	-0.01	-0.18	0.01	0.04	1.00	0.07	0.08	-0.00	0.02	0.06	-0.14	-0.05	-0.02
JOB	-0.01	0.15	0.32	0.35	0.07	1.00	-0.07	-0.02	-0.03	0.16	-0.04	0.19	-0.02
YOJ	-0.06	0.08	-0.10	-0.02	0.08	-0.07	1.00	-0.06	0.02	0.23	-0.05	-0.00	-0.05
DEROG	0.25	-0.02	-0.04	-0.05	-0.00	-0.02	-0.06	1.00	0.15	-0.05	0.18	0.07	0.04
DELINQ	0.27	-0.07	-0.04	-0.07	0.02	-0.03	0.02	0.15	1.00	0.01	0.00	0.12	0.04
CLAGE	-0.11	0.07	0.14	0.20	0.06	0.16	0.23	-0.05	0.01	1.00	-0.09	0.20	-0.05
NINQ	0.13	0.05	0.04	-0.01	-0.14	-0.04	-0.05	0.18	0.00	-0.09	1.00	0.08	0.16
CLNO	0.01	0.08	0.32	0.22	-0.05	0.19	-0.00	0.07	0.12	0.20	0.08	1.00	0.14
DEBTINC	0.22	0.16	0.19	0.13	-0.02	-0.02	-0.05	0.04	0.04	-0.05	0.16	0.14	1.00

Table 3: Correlation Matrix

The correlation matrix provided offers several key insights into the relationships between variables:

1. Positive Correlations with the BAD Variable:
   - **DEROG (0.25):** A moderately positive correlation between the number of derogatory reports and defaulting on loans. This is intuitive, as derogatory reports, such as missed payments and bankruptcies, are indicators of financial instability, increasing the likelihood of default.
   - **DELINQ (0.27):** The number of delinquent credit lines has the strongest positive correlation with BAD. Delinquent accounts represent instances of failure to meet repayment obligations, making individuals more likely to default again.
   - **DEBTINC (0.22):** A positive correlation between the debt-to-income ratio and default risk. As debt increases relative to income, individuals are more likely to struggle with payments, which increases default risk.

2. Variables with Low or No Correlation to BAD:
   - **LOAN (-0.03)**, **MORTDUE (-0.02)**, **VALUE (-0.03)**, and **JOB (-0.01)** have very low correlations with the BAD variable, suggesting that these factors do not have a significant linear relationship with default in this dataset. These findings indicate that variables such as the current mortgage due or loan amount do not directly impact default probability.

3. Multicollinearity Between Independent Variables:

- **MORTDUE** and **VALUE** exhibit a strong positive correlation (**0.87**). This indicates that individuals with higher property values also tend to have higher amounts due on their mortgages, a logical relationship in real estate financing.
- **LOAN** and **VALUE** are also moderately correlated (**0.38**), suggesting that larger loans are generally associated with higher property values.

These high correlations between independent variables can introduce multicollinearity, particularly in linear models like logistic regression, which could distort the interpretation of individual predictors. This puts some limitations on this model, that can be easily surpassed by other models.

4.1.3 Default distribution

This section provides useful information about the distribution of defaulters by professional occupancy and reason of loan request.

Starting with the job title, we have that:

- **Office**: individuals employed in administrative or office-work positions. These roles are usually mid-level in terms of income and stability.
    - From a total of 577 individuals, 38 defaulted, i.e., 6.59%
- **Mgr:** Managers, these jobs typically come with higher salaries and more responsibility, which may indicate greater financial stability and a lower risk of default.
    - From a total of 450 individuals, only 45% defaulted, i.e., 10%
- **ProfExe:** Professional/Executive, referring to high-earning, highly stable jobs such as lawyers, doctors, engineers, and top executives. This group is generally considered to have the lowest risk of default due to higher income and job security.
    - From a total of 899 individuals only 58 defaulted, i.e., 6.45%
- **Sales:** Refers to individuals working in sales, which may vary in terms of stability and income depending on whether the position is commission-based or salaried. This category may carry a moderate risk of default, depending on the nature of the job and the economic environment.
    - From a total of 53 individuals, 14 defaulted, i.e., 26.4%
- **Self:** Refers to individuals who are self-employed. While self-employment can potentially yield high income, it is often associated with higher income variability and financial uncertainty, which could increase the risk of default.
    - From a total of 99 individuals, 13 defaulted, i.e., 13.1%

The results align with established findings regarding the influence of employment status on default probability. Several studies have shown that certain job categories, such as professionals and executives, generally have lower default rates due to their higher income stability, while categories such as self-employed or sales workers are more prone to financial volatility, resulting in higher default risks (Guegan & Hassani, 2018; Singh et al., 2023).

The distribution of defaults across job categories in the dataset reflects these trends, with higher default rates among the self-employed and those in lower-paying or less stable job categories, consistent with findings in the literature that emphasizes employment as a key predictor of credit risk.

Below, the graph of the default distribution by professional occupancy:

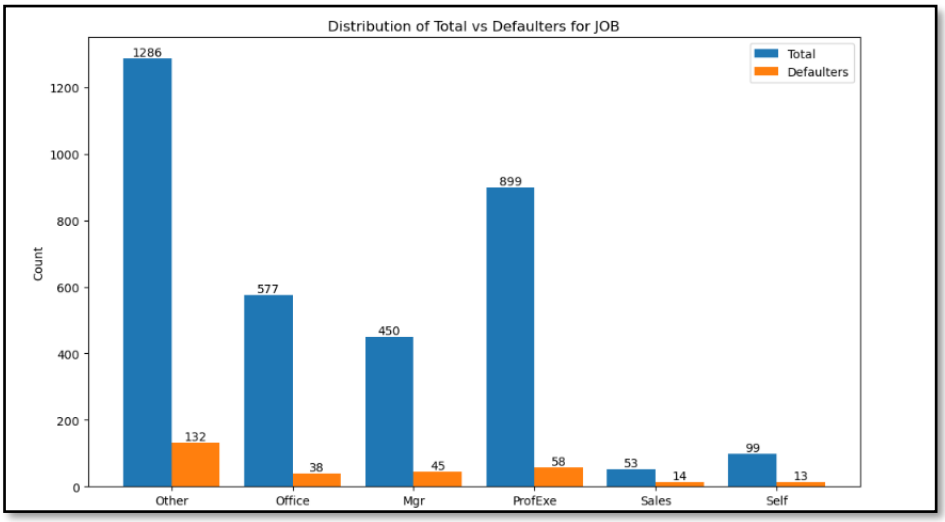

Figure 2: Distribution of defaulters by JOB.

As for the second categorical variable, we have that 995 people asked a loan for home improvements while 86 defaulted (i.e., 8.6%) and 2369 people asked a loan as debt consolidation while 214 defaulted (i.e., 21.5%).

This is consistent with previous studies that suggest that borrowers who take out loans for home improvement tend to have a lower probability of default compared to those who take loans for purposes like home purchases or debt consolidation.

In general, borrowers who seek loans for home improvement are often considered less risky because these loans tend to be smaller, and borrowers may already have some financial stability through existing home equity. Improving or maintaining a home can also be seen as a sign of commitment to property ownership, which aligns with financial stability and a lower likelihood of default (Qi, 2024).

On the other hand, borrowers who take out loans for debt consolidation or home purchases often carry a higher risk of default. For home purchases, particularly with higher loan-to-value (LTV) ratios, the risk of default increases because these loans are larger, and the borrower's financial strain is higher. Higher LTV ratios, which reflect a smaller down payment and larger loan relative to property value, have been shown to be strong predictors of default (Wong et al., 2005; Qi, 2024).

In our case, the data showing higher default rates for debt consolidation aligns with findings in the literature, where consolidating debt typically indicates that the borrower is already financially stressed, further increasing the risk of default (Wong et al., 2005; Qi, 2024).

Below, the graph of the default distribution by reason of loan request:

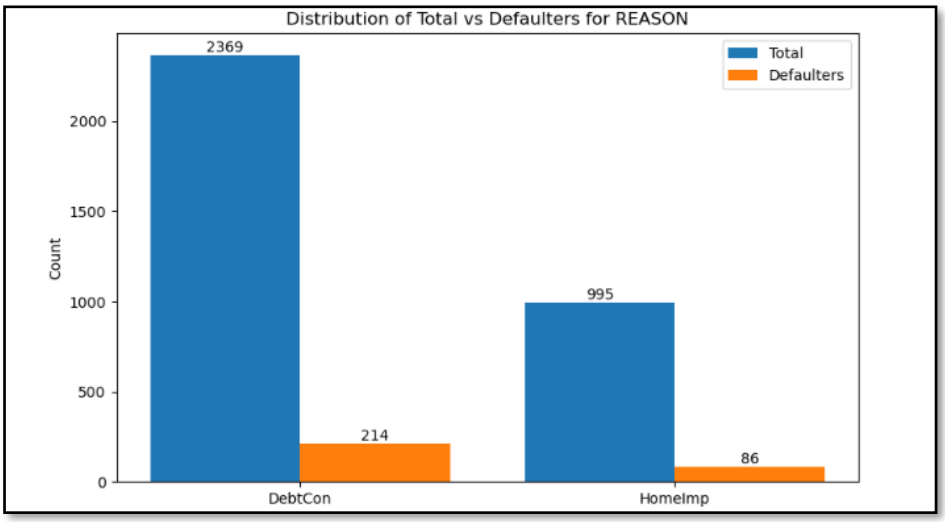

Figure 3: Distribution of Defaulters by reason of loan-request

4.2 Test Results

In this section, the results of various tests will be presented, with a particular emphasis on the outcomes derived from the test set. The results obtained from the validation tests can be reproduced by following the python code provided in the appendix.

4.2.1 XGBoost Results

Starting from Extreme Gradient Boosting we have the following:

```
Test Set Classification Report
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       304
           1       1.00      0.77      0.87        30

    accuracy                           0.98       334
   macro avg       0.99      0.88      0.93       334
weighted avg       0.98      0.98      0.98       334

AUC (Test): 0.9425
CPU times: total: 8.25 s
Wall time: 882 ms
```

Table 4:XGBoost Evaluation results

Table 4 presents the results of a 10-fold cross-validation. The data used in the multi-analysis consists of 304 non-defaulters and 30 defaulters, maintaining the exact ratio as in the original dataset.

The AUC is 0.9425 while the accuracy is 98%. The results by default/non default are:

- Default (1):
  - Precision: 100%

- Recall: 77%
- F1-Score 87%

- Non-Default (0):
  - Precision: 98%
  - Recall: 100%
  - F1-Score: 99%
  - The visual representation of the ROC-AUC is the following:

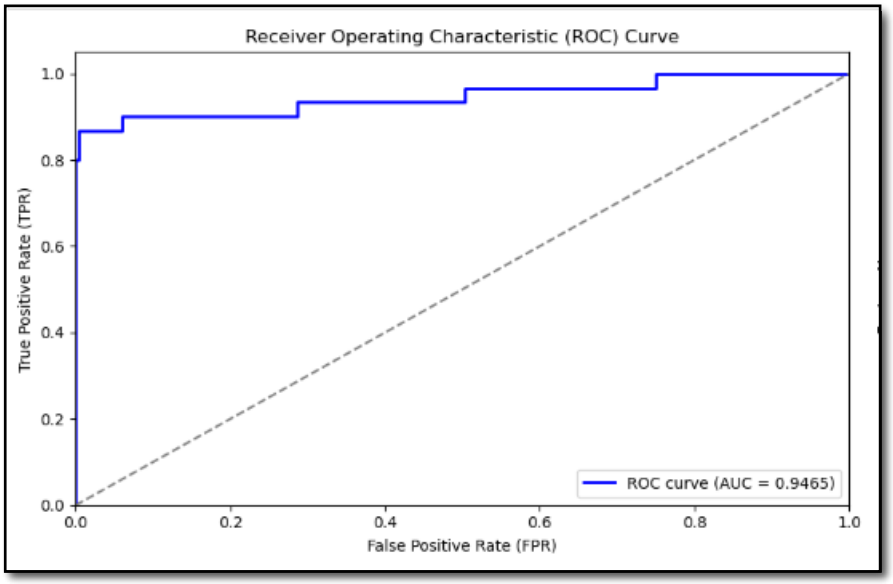

Figure 4:ROC-AUC XGBoost model

While the hierarchy of the features that the model considered best is:

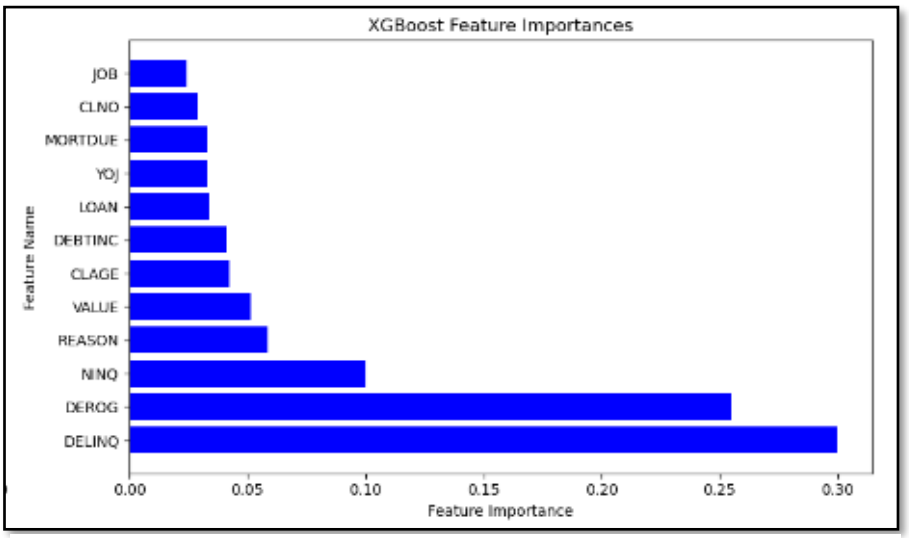

Figure 5: Features XGBoost

The top 3 features are in this case:

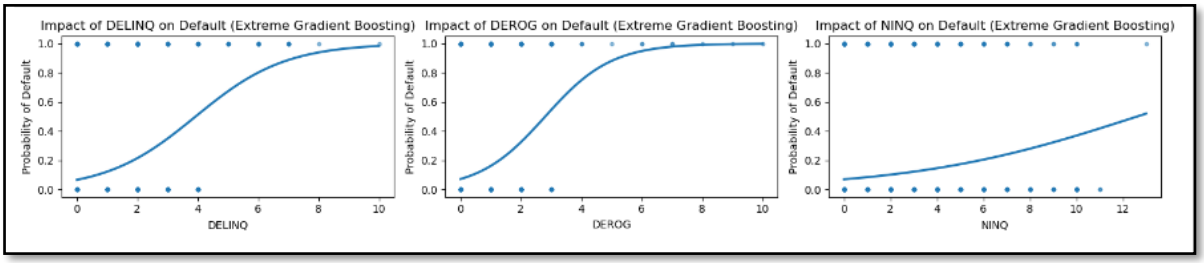

Figure 6: Top 3 features for XGBoost model

The confusion matrix generated by the model is therefore:

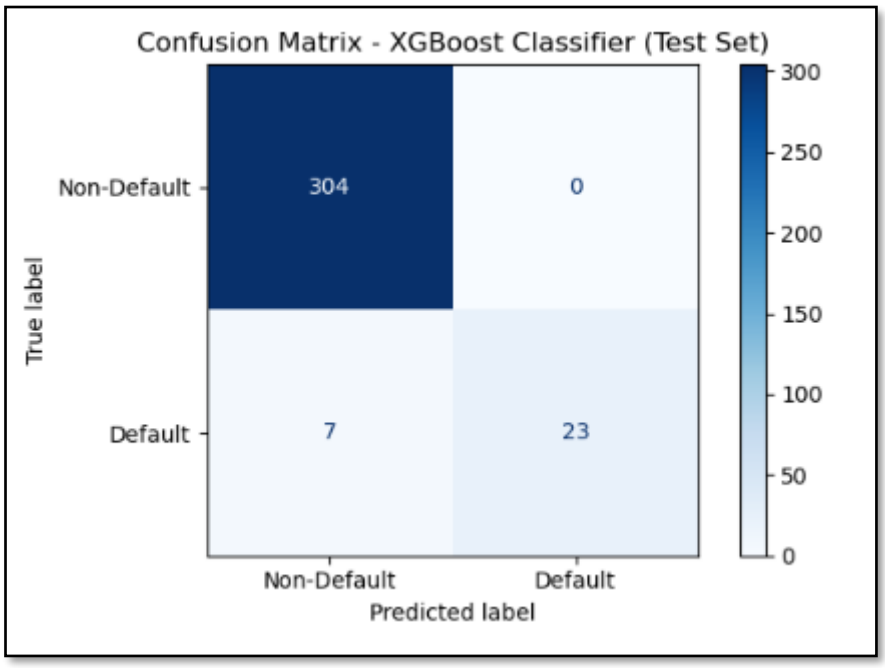

Figure 7: Confusion Matrix of XGBoost model

It can be observed that the model correctly identified 23 out of the 30 defaulters, while no misclassifications were made among the non-defaulters.

4.1.2 Random Forest Results

The results of the test are the following:

```
Test Set Classification Report
              precision    recall  f1-score   support

           0       0.98      1.00      0.99       304
           1       0.96      0.77      0.85        30

    accuracy                           0.98       334
   macro avg       0.97      0.88      0.92       334
weighted avg       0.98      0.98      0.97       334

AUC (Test): 0.9823
CPU times: total: 2.14 s
Wall time: 2.15 s
```

Table 5: Random Forest Evaluation results

Table 5 shows the results of the Random Forest model. The same 334 individuals are tested, and it shows the following:

The AUC is 0.9823 while the accuracy is 98%. The results by default/non default are:

- Default (1):
  - Precision: 96%
  - Recall: 77%
  - F1-Score 85%
- Non-Default (0):
  - Precision: 98%
  - Recall: 100%
  - F1-Score: 99%

The visual representation of the ROC-AUC is the following:

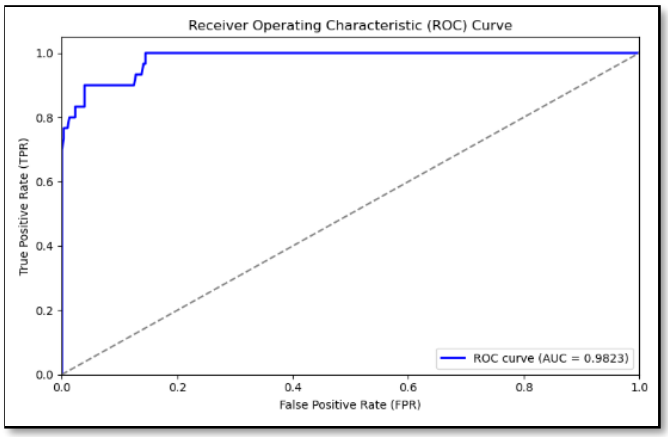

Figure 8: ROC-AUC Random Forest model

While the features list is:

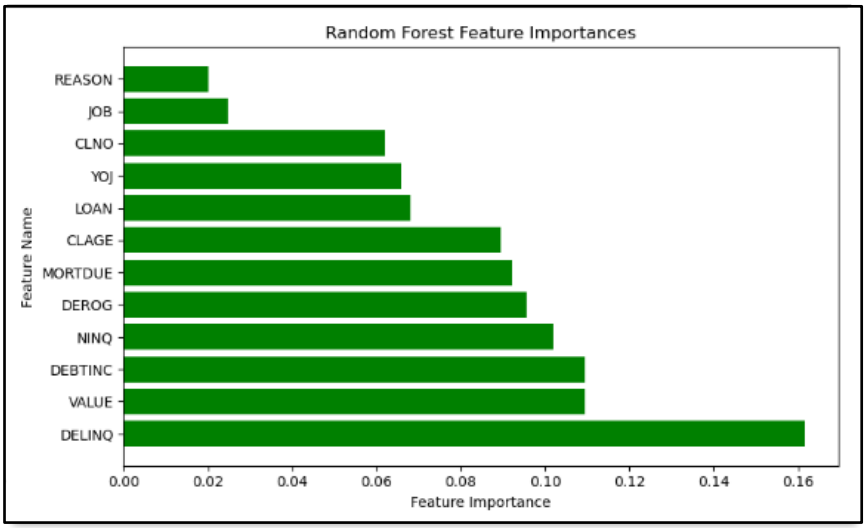

Figure 9: Features Random Forest

Below there can be seen the top 3 features that the model considers important:

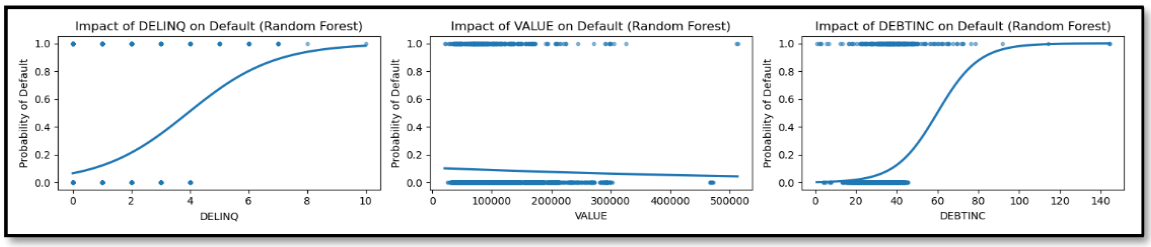

Figure 10: Top 3 features for Random Forest model

The confusion matrix generated by the model is therefore:

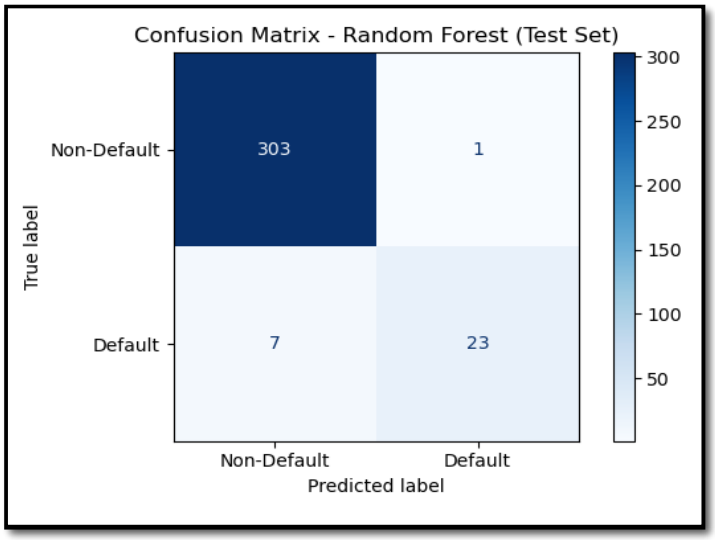

Figure 11: Confusion Matrix of Random Forest model

Figure 12 shows that 23 out of 30 defaulters are correctly individuated.

4.1.3 Logistic Regression Results

The main results are the following:

Logistic Regression Test Set Classification Report

	precision	recall	f1-score	support
0	0.96	0.77	0.85	304
1	0.22	0.63	0.32	30
accuracy			0.76	334
macro avg	0.59	0.70	0.59	334
weighted avg	0.89	0.76	0.81	334

AUC (Test): 0.7679
CPU times: total: 31.2 ms
Wall time: 25.7 ms

Table 6: Logistic Regression Evaluation results

Table 6 shows the results of the Logistic Regression model for the same 334 individuals.

The AUC is 0.7679 while the accuracy is 76%. The results by default/non default are:

- Default (1):
  - Precision: 22%
  - Recall: 63%
  - F1-Score 32%
- Non-Default (0):
  - Precision: 96%
  - Recall: 77%
  - F1-Score: 85%
  - The visual representation of the ROC-AUC is the following:

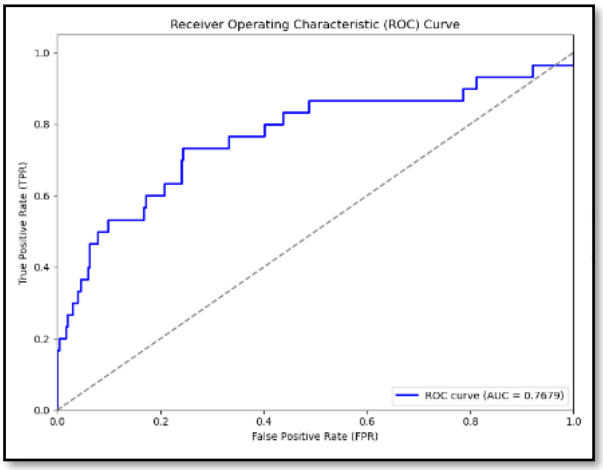

Figure 12: ROC-AUC Logistic Regression model

The coefficients estimated by the model are the following:

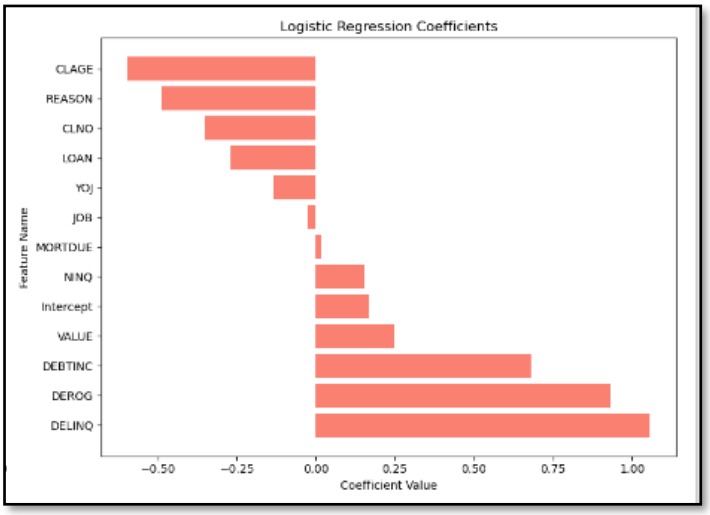

Figure 13: Coefficients of Logistic Regression model

While the 3 most important features are:

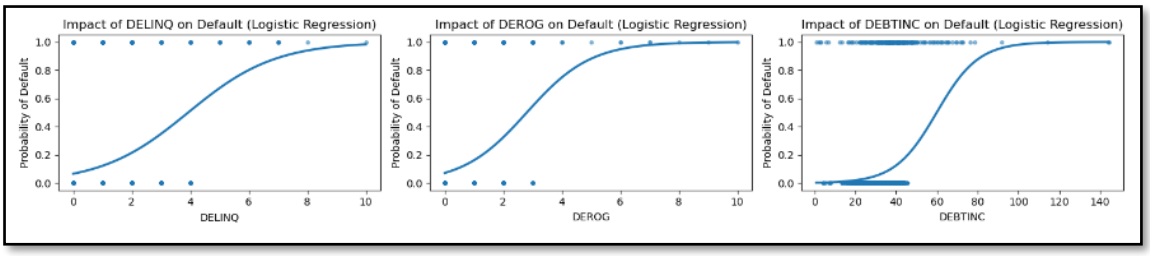

Figure 14: Top 3 features of Logistic Regression model

The confusion matrix generated by the model is:

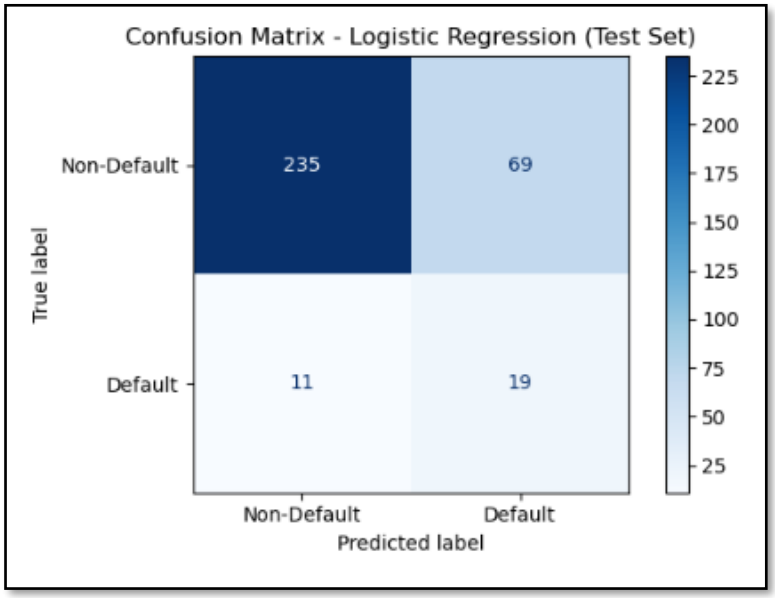

Figure 15: Confusion Matrix of Logistic Regression model

It can be seen that the model correctly predicts 19 out of 30 defaulters, while only 235 non-defaulters are correctly estimated out of 304 total non-defaulters.

4.1.4 Decision Tree Results

The test results are the following:

Decision Tree Validation Set Classification Report

	precision	recall	f1-score	support
0	0.96	0.96	0.96	304
1	0.63	0.63	0.63	30
accuracy			0.93	334
macro avg	0.80	0.80	0.80	334
weighted avg	0.93	0.93	0.93	334

AUC (Validation): 0.7986
CPU times: total: 31.2 ms
Wall time: 34.6 ms

Table 7: Decision Tree Evaluation results

Table 7 shows the results of the Decision Tree model for the same 334 individuals.

The AUC is 0.7986 while the accuracy is 93%. The results by default/non default are:

- Default (1):
  - Precision: 63%
  - Recall: 63%
  - F1-Score 63%
- Non-Default (0):
  - Precision: 96%
  - Recall: 96%
  - F1-Score: 96%

The graph of the ROC-AUC is the following:

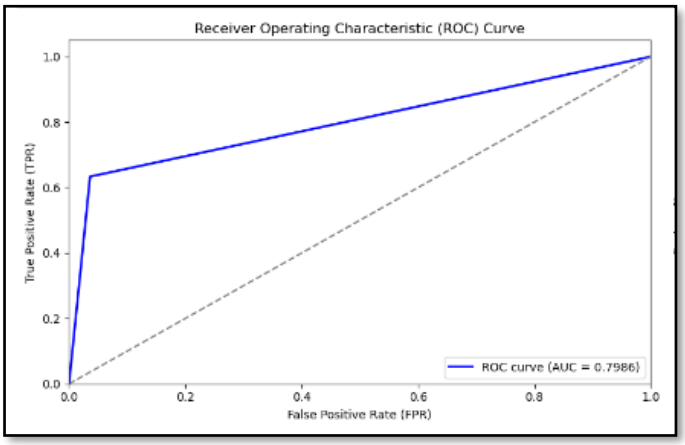

Figure 16: ROC-AUC of Decision Tree model

The features ranked by importance are:

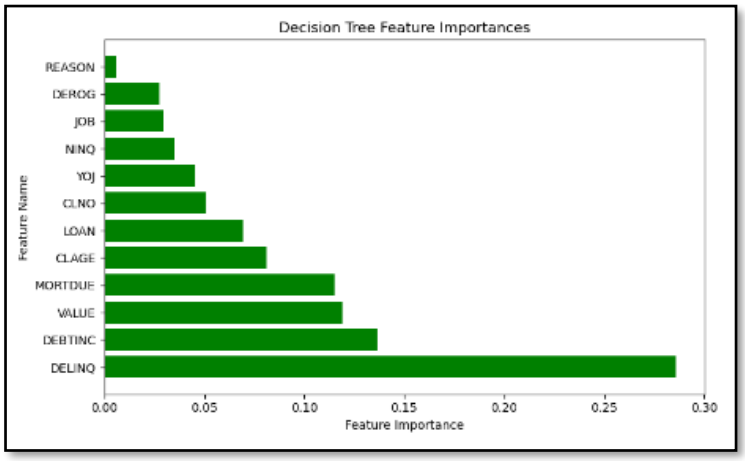

Figure 17: Features of Decision Tree model

And the top 3 features considered by the model are:

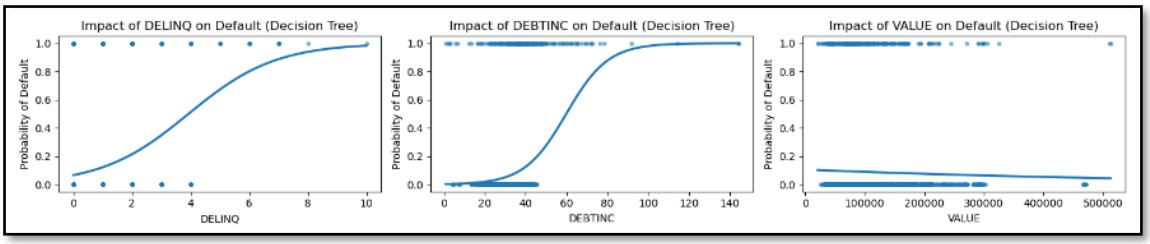

Figure 18: Top 3 features of Decision Tree model

The confusion matrix is:

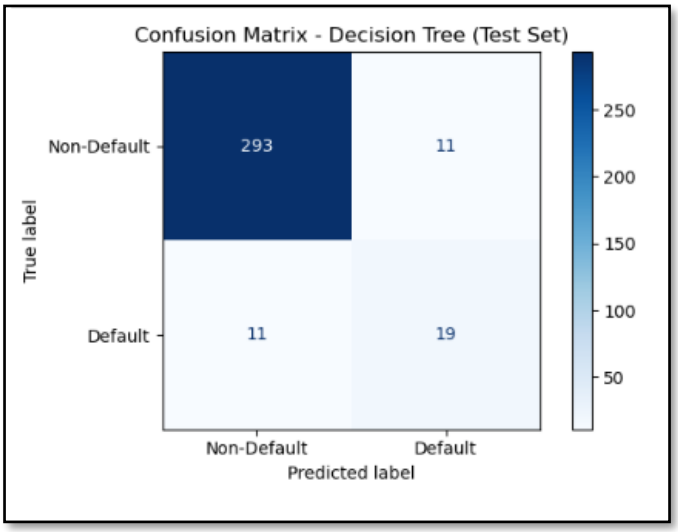

Figure 19: Confusion Matrix of Decision Tree model

Figure 15 shows that 19 out of 30 defaulters are correctly estimated while 293 non-defaulters out of 304 are correctly estimated.

4.1.5 AdaBoost Results

The results are:

Test Set Classification Report

	precision	recall	f1-score	support
0	0.98	0.95	0.96	304
1	0.62	0.80	0.70	30
accuracy			0.94	334
macro avg	0.80	0.88	0.83	334
weighted avg	0.95	0.94	0.94	334

AUC (Test): 0.8910

Table 8: AdaBoost Evaluation results

Table 8 shows the results of the AdaBoost model for the same 334 individuals.

The AUC is 0.8910 while the accuracy is 94%. The results by default/non default are:

- Default (1):
  - Precision: 62%
  - Recall: 80%
  - F1-Score 70%
- Non-Default (0):
  - Precision: 98%
  - Recall: 95%
  - F1-Score: 96%
  - The ROC-AUC graph is the following:

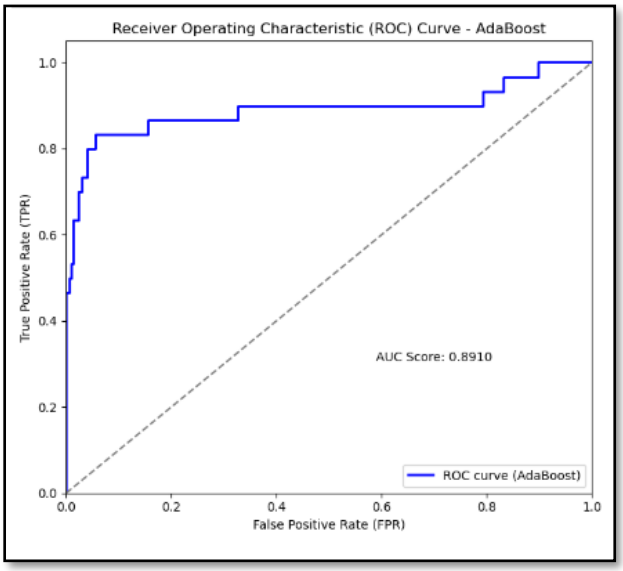

Figure 20: ROC-AUC of AdaBoost Model

Adaboost model's interpretability may be lower compared to other machine learning models, especially in scenarios where many weak learners are involved. The boosting mechanism, where models are trained sequentially, makes it harder to understand which individual feature had the largest impact across all iterations. Hence, for this model, the graph of feature importance is skipped since even if provided, it could be less intuitive and accessible compared to the other models.

Jumping straight to the confusion matrix we have:

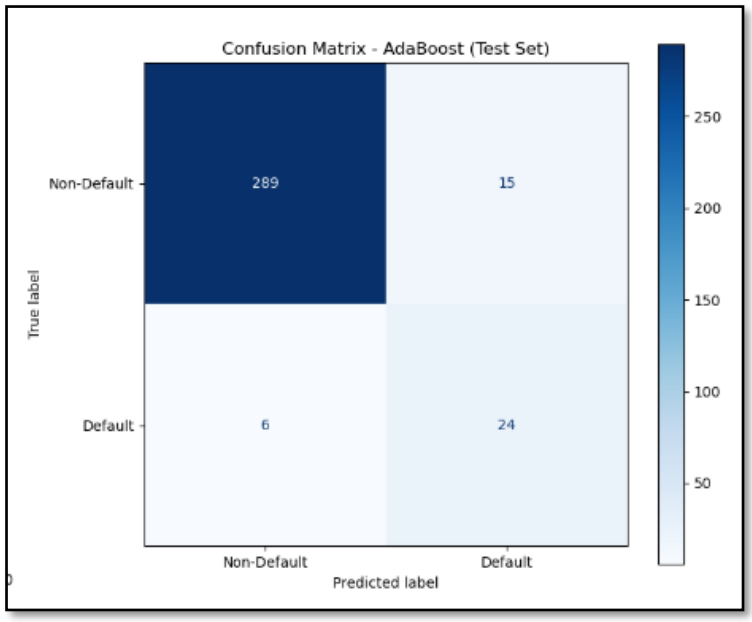

Figure 21: Confusion Matrix of AdaBoost model

The model correctly predicts 24 out of 30 defaulters but only 289 out of 304 non-defaulters.

4.1.6 Gradient Boosting Results

The results are:

Test Set Classification Report

	precision	recall	f1-score	support
0	0.98	0.98	0.98	304
1	0.80	0.80	0.80	30
accuracy			0.96	334
macro avg	0.89	0.89	0.89	334
weighted avg	0.96	0.96	0.96	334

AUC (Test): 0.9203

Table 9: Gradient Boosting Evaluation results

Table 9 shows the results of the Gradient Boosting model for the same 334 individuals.

The AUC is 0.9203 while the accuracy is 96%. The results by default/non default are:

- Default (1):
  - Precision: 80%
  - Recall: 80%
  - F1-Score 80%
- Non-Default (0):
  - Precision: 98%
  - Recall: 98%

- F1-Score: 98%

The ROC-AUC graph is the following:

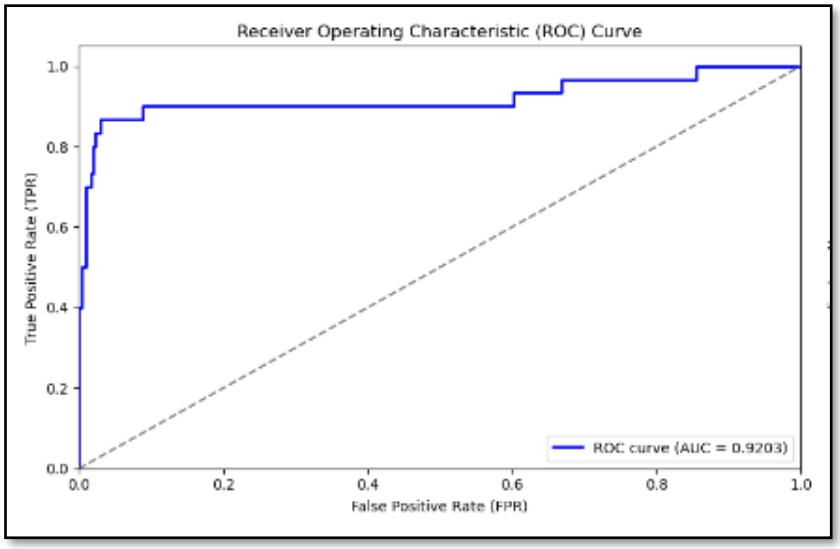

Figure 22: ROC-AUC of Gradient Boosting model

While the ranking of the features is the following:

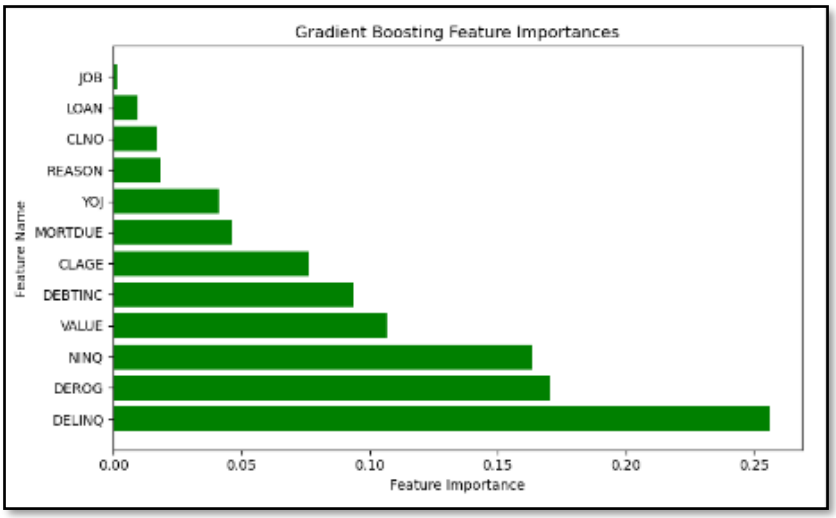

Figure 23: Features of Gradient Boosting model

And the top 3 features are the following:

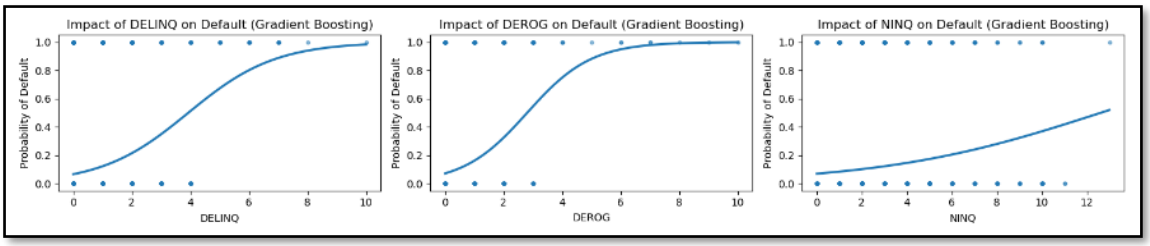

Figure 24: Top 3 features of Gradient Boosting model

The confusion matrix is therefore:

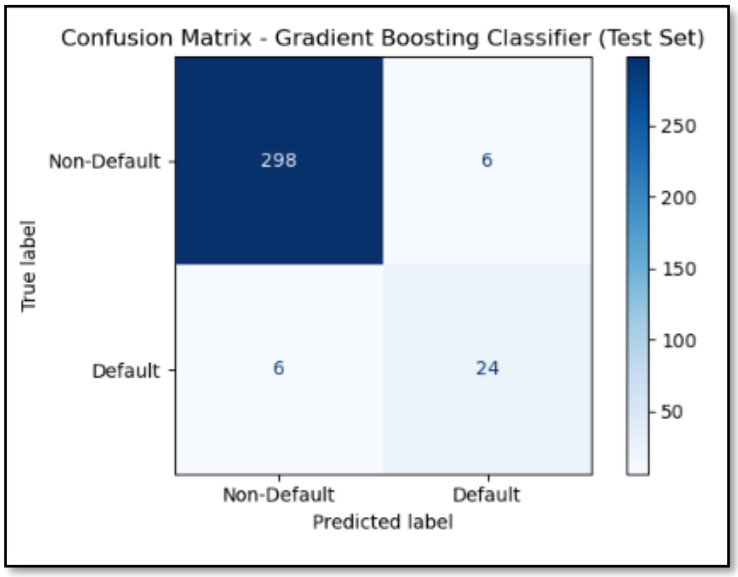

Figure 25: Confusion matrix of Gradient Boosting model

The model correctly predicts 24 out of 30 defaulters while on the non-defaulters side the model predicts correctly 298 individuals out of 304.

4.2 Analysis of Model Pairing

When conducting an analysis to determine whether an individual is likely to default on a loan, it is crucial not to rely solely on a single machine learning model. Different models are designed to capture different aspects of the data, and each may perform differently depending on the structure and complexity of the dataset. By comparing multiple models, a more comprehensive understanding of the patterns and correlations in the data can be achieved. This approach allows us to identify which models are better suited for predicting defaulters in specific contexts and how they complement one another. In this study, I developed a comparison matrix to evaluate the predictions made by several models. The matrix highlights how many defaulters each model predicts and, more importantly, identifies whether the same individuals are consistently predicted as defaulters across different models. This comparative analysis helps to mitigate the risk of

missing key defaulters by leveraging the strengths of each model, ensuring a more reliable and accurate prediction of default risk.

The following analysis examines the 30 individuals who defaulted in the test set. The values in the matrix represent the number of individuals correctly predicted by each pair of models. Specifically, the matrix shows how many defaulters were jointly predicted by each model pair, reflecting agreement in identifying true defaulters. A higher value indicates a greater consistency between models in identifying the same defaulters, suggesting a higher level of agreement and predictive alignment between those models.

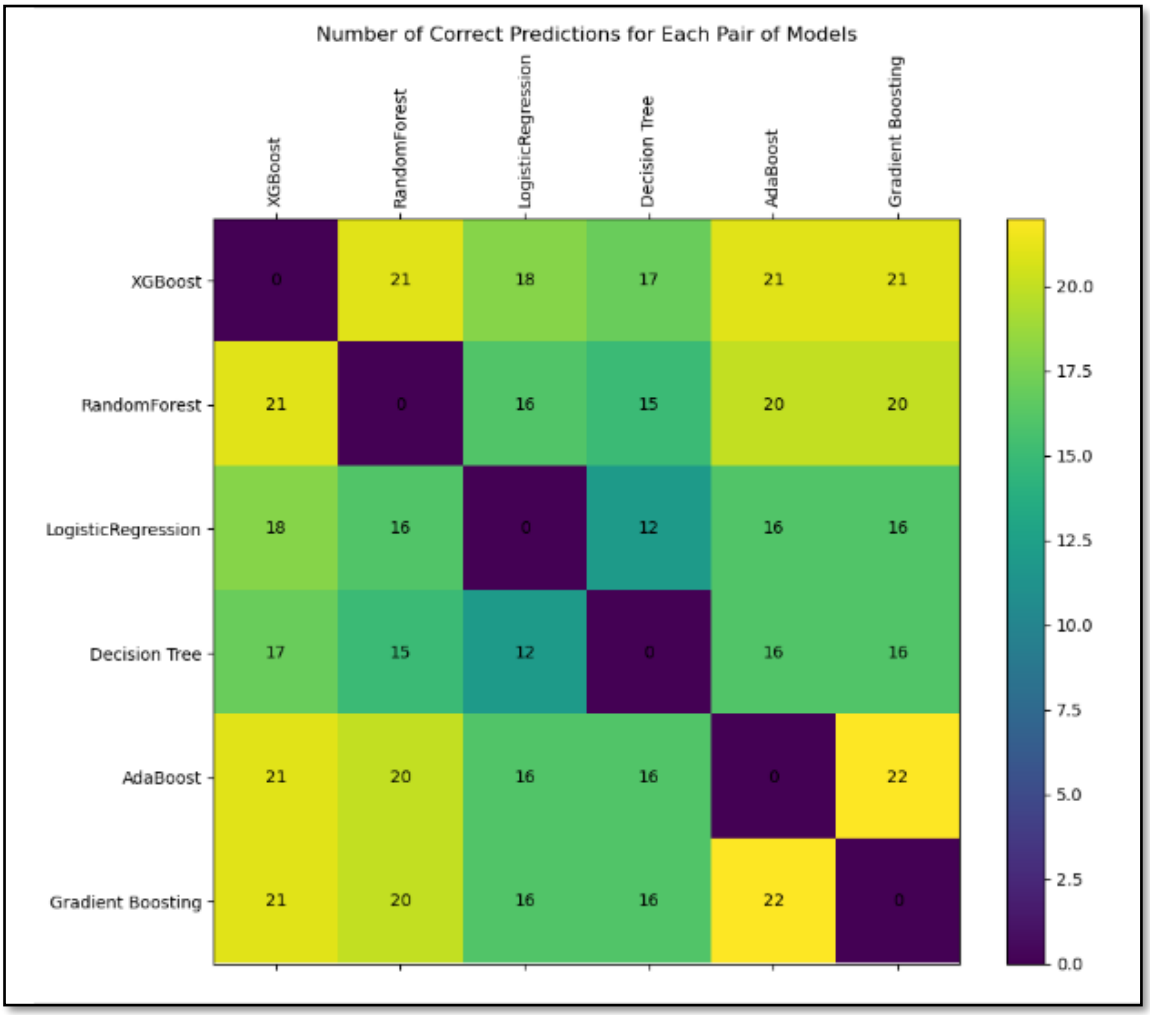

Figure 26: Matrix of common predictions among models

Figure 26 shows that:

- Top tier pair with 22 correct predictions in common:
  - Gradient Boosting-AdaBoost
- Second tier pair with 21 correct predictions in common:
  - XGBoost – AdaBoost
  - XGBoost – Random Forest
  - XGBoost – Gradient Boosting

- Third tier pair with 20 correct predictions in common:
  - Random Forest – AdaBoost
  - Random Forest – Gradient Boosting

In the next chapter there will be provided some solutions and interpretations to these results.

Chapter 5
Discussion

5.1 Objective

The primary objective of this thesis is to demonstrate the varying performance levels of different machine learning models when applied to the prediction of credit default and to underline the significant role that parameter tuning plays in optimizing these models. Furthermore, this analysis aims to stress the importance of focusing on certain evaluation metrics, particularly Recall, in determining the probability of default for individuals. This is because financial institutions are generally more concerned with accurately identifying defaulters (true positives), as missing these cases can lead to considerable financial losses.

This study includes a comparative analysis of Extreme Gradient Boosting (XGBoost), Random Forest, Logistic Regression, Gradient Boosting, Decision Tree, and AdaBoost. Through this analysis, the strengths and weaknesses of each model are highlighted, offering insights into their capabilities for predicting credit defaults. Although there is existing literature comparing machine learning models in the context of credit risk, such as Hamzic et al. (2023), Noriega et al. (2023), Singh et al. (2023), and Guegan & Hassani (2018), comprehensive comparisons that explore model performance across a wide array of metrics, specifically for predicting the probability of default, are relatively recent. This type of broad model comparison is gaining importance as credit risk modelling increasingly incorporates machine learning techniques.

Moreover, the emphasis on Recall as a key metric is underrepresented in much of the recent literature, despite its critical role in credit risk contexts where dataset imbalance is a major concern. While some studies, such as Powers (2011), Sokolova & Lapalme (2009), and James et al. (2013), have underscored the relevance of Recall, there remains a gap in recent publications specifically addressing the significance of this metric in the context of credit risk, where the cost of missing a default is far greater than a false positive. This thesis aims to bridge this gap by providing a comprehensive evaluation of multiple machine-learning models, emphasizing metrics like Recall to improve the accuracy and reliability of predicting true defaulters.

5.2 Discussion

5.2.1 Overview of Key Findings and Comparison with Existing Literature

As discussed in the previous chapter, there is a positive correlation between default risk and key financial indicators such as the number of major derogatory reports, number of delinquent credit lines, and the debt-to-income ratio. These relationships are consistent with findings in recent literature, which emphasize the significance of these variables as predictors of credit default. For instance, the analysis by IME (Dataiku) highlights that

major derogatory reports, delinquent credit lines, and debt-to-income ratio are critical features for predicting credit risk, indicating their strong association with increased default likelihood. Similarly, the study by Rawat (2023) in the context of the CRISP-DM methodology underscores the role of these credit-related indicators in developing machine learning models for credit risk prediction. This convergence of evidence from both empirical data and existing studies reinforces the robustness of these indicators in understanding default risk.

On the other hand, the models considered suggest some interesting insights:

All the models (with exception being AdaBoost for the mentioned reason) classify "DELINQ" as being the most important feature when predicting default. This is consistent with the findings of Noriega et al. (2023) where highlights the significant role of delinquency indicators, such as the number of delinquent credit lines, in predicting default risk. This study suggests that delinquent credit lines are one of the primary factors used to assess an individual's likelihood of default, as they provide critical insights into past credit behaviour and financial stress.

The second sport is occupied by "DEROG" and "NINQ". This is also backed by Shi et al. (2022) and Weber et al. (2023) which states that derogatory reports (e.g., missed payments or delinquencies) have a direct impact on an individual's likelihood to default. These reports are clear indicators of historical financial stress, making them powerful predictors of future credit behaviour and that frequent credit inquiries typically signal a borrower in financial distress, increasing the risk of default. This makes recent credit inquiries a reliable indicator in credit scoring models.

Discussion on Extreme Gradient Boosting algorithm

Class 0 (Non-Defaulters): The precision for class 0 is 0.98 and the recall is 1.00, which is consistent with the findings by Hamzic et al. (2023), who highlighted the strengths of XGBoost in classifying majority classes effectively due to its sequential learning approach. This indicates that the model is highly capable of correctly identifying non-defaulters, with no false negatives.

Class 1 (Defaulters): The precision for defaulters is 1.00, meaning that when the model predicts a defaulter, it is always correct. The recall for defaulters is 0.77, implying that 23% of actual defaulters were missed, similar to findings by Singh et al. (2023), who noted that XGBoost can sometimes fall short in capturing minority classes fully without fine-tuning. Their results indicate a recall of 78% of the minority class while Hamzic et al. (2023) found a 76% Recall.

The F1-score for defaulters is 0.87, indicating a good but not perfect balance between precision and recall. It is in line with Noriega et al. (2023), who argued that ensemble methods strike a balance between precision and recall in imbalanced datasets. Given that F1-score is particularly useful in dealing with imbalanced datasets—like credit risk data where the cost of missing defaulters is high—this result demonstrates that the model effectively balances both metrics to a large extent.

Overall, the accuracy of 0.98 is quite high; however, in this context, accuracy alone can be misleading due to the imbalance between defaulters and non-defaulters. The model performs exceptionally well for the majority class, leading to a high accuracy score, but it is more important to focus on metrics like recall and AUC to evaluate the model's practical effectiveness in predicting defaults. The macro average recall is 0.88, suggesting good overall recall, but there is still room for improvement, especially considering the potential high cost of undetected defaults for lending institutions.

The AUC of 0.9425 indicates that the model has a very good ability to distinguish between defaulters and non-defaulters, highlighting strong performance in separating the positive class from the negative class. This AUC is comparable to Hamzic et al. (2023) and Singh et al. (2023), which indicates strong discrimination between classes using XGBoost for credit risk prediction, reflecting its suitability for high-stakes decision-making contexts. Their findings are AUC of 93% and 94% respectively.

Overall, the high precision for defaulters is an encouraging outcome, as it means that when the model identifies an individual as likely to default, it is always correct. While the recall of 0.77 suggests some defaulters are missed, it also shows that the model successfully identifies a majority of defaulters with complete confidence. This trade-off is a natural challenge in the context of imbalanced data, but the results here show promising performance that could be fine-tuned to further improve recall without sacrificing precision.

Discussion on Random Forest algorithm

Class 0 (Non-Defaulters): The precision is 0.98 and the recall is 1.00, showing that the model is excellent at correctly identifying non-defaulters, with no false negatives for this class. This implies that every individual who did not default was accurately predicted, resulting in a very high F1-score of 0.99. These findings are consistent with Noriega et al. (2023) who demonstrated that Random Forest excels at managing majority classes due to its ensemble nature and found a precision of 98%.

Class 1 (Defaulters): For defaulters, the precision is 0.96, meaning that almost all predicted defaulters were indeed correct, showing high confidence in its predictions. The recall for defaulters is 0.77, which means that 23% of actual defaulters were missed. Even though this implies some missed predictions, the model still correctly identifies 77% of defaulters, which is comparable to other models discussed in recent research. A recall of 0.77 is comparable to the findings by Hamzic et al. (2023), who indicated that Random Forest models often require hyperparameter tuning to sufficiently capture minority classes, even when their precision is high (0.96 in this case). Their findings provided a recall of 72%.

The F1-score of 0.85 for class 1 indicates a solid balance between precision and recall, though the model could benefit from adjustments to reduce the false negatives.

Overall , the accuracy of 0.98 is very high, but as with the XGB model, accuracy can be misleading in imbalanced datasets where the majority class dominates. The macro average recall is 0.88, indicating room for improvement in identifying defaulters more

effectively. Given the significant consequences of missing defaulters in a credit risk scenario, this is an area where potential fine-tuning of the model could be beneficial.

The AUC score is 0.9823, which suggests that the model is highly effective at distinguishing between defaulters and non-defaulters. This high AUC indicates that, overall, the model performs well in separating positive and negative cases, despite the challenges in recall for the minority class. These results are similar to those reported by Hamzic et al. (2023), demonstrating that Random Forest provides high discriminative power (AUC=0.98 in their case).

Interpretation and Practical Implications: The high precision for class 1 (defaulters) is particularly positive, as it indicates that when the model predicts a defaulter, it is correct. However, with a recall of 0.77, the model misses 23% of actual defaulters, which is a drawback that could impact its practical utility. Nevertheless, the 77% recall means that a significant proportion of defaulters are identified with complete confidence, providing reliable predictions when defaults are detected. For practical applications, further tuning could focus on improving recall while preserving the excellent precision, to minimize missed defaulters and optimize risk assessment strategies.

Discussion on Logistic Regression algorithm

Class 0 (Non-Defaulters): The precision for class 0 is 0.96, indicating that most of the predictions for non-defaulters were correct. However, the recall is 0.77, meaning that 23% of non-defaulters were incorrectly classified as defaulters, which is far from ideal compared to the other models discussed. The F1-score of 0.85 shows a reasonable balance between precision and recall, but the recall could be improved to reduce false positives. These results are consistent with Menard (2002), which emphasized poor recall rates when used in highly imbalanced datasets, indicating the model's limitation in accurately classifying non-defaults compared to more sophisticated machine learning models.

Class 1 (Defaulters): The performance for defaulters is where Logistic Regression falls short. The precision is 0.22, meaning that only 22% of predicted defaulters were correct—an extremely poor result. The recall is 0.63, meaning t hat 37% of true defaulters were missed. While the recall here is slightly higher than the precision, this indicates that the model struggles both with catching defaulters and with accurately predicting who is a defaulter. The precision of 0.22 and recall of 0.63 reveal the model's inadequacy, as also noted by Noriega et al. (2023), who emphasized that Logistic Regression struggles with non-linear patterns in credit risk data.

The F1-score of 0.32 is quite low, reflecting this imbalance and poor performance overall.

Overall, the accuracy of 0.76 is substantially lower than the other models discussed, which is expected given the poor performance in identifying defaulters accurately. The macro average recall is 0.70, and the macro average precision is 0.59, indicating that the model does not perform well overall. The imbalance in performance between classes is stark, and for a dataset dealing with credit default, these results are inadequate for practical implementation.

The AUC score is 0.7679, indicating limited discriminative power in separating defaulters from non-defaulters. Compared to the AUC scores from the other models (such as 0.9425 for XGBoost and 0.9823 for Random Forest), this score highlights that Logistic Regression is less effective for this problem, and other models provide far superior separation between the classes. The AUC of 0.7679 aligns with the findings of Guegan & Hassani (2018) where the AUC values were below 0.78.

The low precision for defaulters (0.22) and relatively modest recall (0.63) demonstrate that Logistic Regression has significant limitations, especially when tasked with accurately identifying potential defaulters. Given that the cost of incorrectly classifying a defaulter is very high for lending institutions, relying on this model would pose serious risks. The F1-score of 0.32 for defaulters, combined with the low AUC, reinforces the need to consider more sophisticated methods that can better handle the complexities and imbalances inherent in credit data.

Overall, these results clearly illustrate why Logistic Regression, despite its interpretability, is not an optimal choice for this kind of problem. The much higher precision, recall, and AUC values achieved by more advanced models like XGBoost and Random Forest make a strong case for shifting towards these ensemble learning techniques. They provide significantly better performance in identifying defaulters, which is crucial for effective credit risk management. This substantiates the objective of this thesis—to move away from older, linear models like Logistic Regression and instead leverage more powerful machine learning algorithms that better handle the nuances and imbalances of credit data.

Discussion on Decision Tree algorithm

Class 0 (Non-Defaulters): The precision for non-defaulters is 0.96, and the recall is also 0.96, which shows that the Decision Tree is effective in correctly identifying non-defaulters. The F1-score of 0.96 demonstrates that the balance between precision and recall is strong for this class, indicating that the model handles the majority class well.

Class 1 (Defaulters): For defaulters, the precision is 0.63 and the recall is also 0.63. This implies that the model only correctly identifies 63% of true defaulters while also having a high rate of false positives, as indicated by the relatively modest precision. Guegan & Hassani (2018) also reported a precision for defaulters around 60-65% and a recall of approximately 60%, also in line with the current findings.

The F1-score of 0.63 reveals a considerable gap in performance compared to class 0, showing that the model struggles with effectively capturing minority class cases. This is a common limitation for simple Decision Tree models when dealing with imbalanced datasets.

Overall, the accuracy of 0.93 might look good at first glance; however, it does not reflect the true performance for the minority class, which is critical in this context. The macro average of precision and recall, both at 0.80, indicates that the performance is skewed significantly in favor of non-defaulters. This imbalance is something that needs to be considered since undetected defaults can have significant financial implications.

The AUC score is 0.7986, which is notably lower compared to the other models such as Random Forest or XGBoost. This suggests that the Decision Tree is less capable of effectively distinguishing between defaulters and non-defaulters, confirming its limitations in providing strong discriminative performance. Hamzic et al.(2023) reported an AUC of 0.80, similar with current findings.

The Decision Tree model demonstrates a typical trade-off: while it has high interpretability and simplicity, it struggles to adequately identify defaulters. With a recall of 0.63 for defaulters, it is clear that 37% of actual defaulters are missed, which is problematic in the context of risk assessment where missing defaults can lead to significant losses. Additionally, the moderate AUC shows that its overall discriminative power is lower than more advanced models, making it less suitable for accurate credit risk prediction.

Compared to more sophisticated ensemble models like Random Forest or XGBoost, the Decision Tree lacks the flexibility to learn from complex data patterns and tends to overfit on training data, leading to poorer generalization. Thus, while Decision Trees provide an easy-to-understand model, their application in credit risk analysis is limited by their inability to consistently capture key risk features, making them a suboptimal choice for this problem. This further supports the thesis objective of moving towards ensemble models that provide both better performance and more nuanced handling of credit risk features.

Discussion on AdaBoost algorithm

Class 0 (Non-Defaulters): The precision for non-defaulters is 0.98, and the recall is 0.95, indicating that the model is highly capable of correctly identifying non-defaulters, but it does make a few errors in this class. The F1-score of 0.96 reflects a strong balance between precision and recall for non-defaulters, demonstrating that the model captures the majority class effectively. The high precision and recall are consistent with the findings of Freund & Schapire (1997), who found that AdaBoost's performance for majority classes due to its iterative learning process.

Class 1 (Defaulters): For defaulters, the precision is 0.62, meaning that of the predicted defaulters, 62% were correct, which implies a fair rate of false positives. However, the recall of 0.80 is a notable strength, indicating that the model correctly identifies 80% of true defaulters, which is particularly relevant in the context of predicting credit risk. The recall of 0.80 is higher than that of the simpler models and aligns with Sokolova & Lapalme (2009), who argued that boosting algorithms like AdaBoost are particularly effective at capturing true positives in imbalanced datasets. Shi et al.(2023) also found rates of 81% and a precision of 63%, very similar to the current findings.

The F1-score of 0.70 reflects the model's effectiveness in capturing defaulters, balancing the relatively lower precision with a solid recall, which is critical for this type of problem.

Overall, the accuracy of 0.94 is competitive with other models, though accuracy is less informative in the presence of class imbalance. The macro average recall of 0.88 shows that the model performs well across both classes, but the lower macro average precision

of 0.80 suggests that there is still a need to reduce false positives. The weighted averages of precision, recall, and F1-score show a good balance, especially considering the imbalanced nature of the data.

The AUC score of 0.8910 indicates a decent ability to distinguish between defaulters and non-defaulters. While not as high as Random Forest or XGBoost, this AUC is still competitive and suggests that the model provides a reasonable separation of classes. This AUC is also consistent with Singh et al. (2023), indicating that while AdaBoost offers a good balance, it may require further parameter tuning to match the performance of other ensemble techniques like Random Forest or Gradient Boosting. Their findings reported an AUC in the range of 0.88-0.89, exactly as the current findings.

The recall of 0.80 for defaulters is a key strength of the AdaBoost model, highlighting its ability to correctly identify most individuals likely to default. Given that identifying defaulters is more critical than avoiding false positives in this context, AdaBoost shows strong potential. The relatively lower precision for defaulters implies that while it catches a significant number of true defaulters, it also misclassifies some non-defaulters as defaulters, which may lead to additional caution in credit approval.

Compared to other models, AdaBoost has demonstrated good recall but slightly lower precision than desired, meaning it might be more aggressive in flagging defaults, which is acceptable depending on the risk appetite of the lender. These results suggest that while AdaBoost could be a viable alternative to traditional models like Logistic Regression, it may still benefit from tuning to further balance precision and recall, especially to lower the false positive rate while retaining its strong recall.

Discussion on Gradient Boosting algorithm

Class 0 (Non-Defaulters): The precision and recall for non-defaulters are both 0.98, showing that the Gradient Boosting model is highly effective at correctly predicting non-defaulters. This results in an F1-score of 0.98, indicating a perfect balance between precision and recall for this class. The model successfully minimizes both false positives and false negatives for non-defaulters, making it highly reliable for the majority class. These results are consistent with Friedman (2001), who demonstrated that Gradient Boosting is capable of high precision and recall for the majority class through its boosting framework.

Class 1 (Defaulters): For defaulters, the precision is 0.80, which means that when the model predicts a defaulter, it is correct 80% of the time. The recall for defaulters is also 0.80, indicating that 80% of actual defaulters are correctly identified by the model. This is a good level of recall for the defaulter class, meaning that the model does a commendable job in minimizing false negatives. The results of Recall matches findings by Noriega et al. (2023), who emphasized Gradient Boosting's strength in identifying true defaulters while managing false positives effectively and reported a recall in the range of 79-82%, while Hamzic et al. (2023) reported recall rates around 82% and precision close to 80%.

The F1-score for defaulters is 0.80, which reflects a good balance, especially in the context of imbalanced credit data, where catching defaulters is crucial.

Overall, the accuracy of 0.96 is high, showing that the model is generally effective. However, as emphasized before, accuracy can be misleading in imbalanced datasets like credit risk. The macro average recall and precision of 0.89 each indicate a balanced model performance across both classes. The weighted averages similarly show a good overall balance, which is important when considering both majority and minority classes.

The AUC score of 0.9203 suggests that the Gradient Boosting model has a very good ability to distinguish between defaulters and non-defaulters, providing confidence in the model's discriminative power. This score is comparable to, though slightly below, the XGBoost model's AUC, but still represents a strong result. It is similar to those reported in Noriega et al. (2023), indicating that Gradient Boosting provides a robust performance for separating defaulters and non-defaulters. Their findings provided an AUC of 0.92.

The 80% recall for defaulters is particularly important, showing that the model catches most true defaulters without compromising too much on false positives. The precision of 0.80 implies that the model is confident and correct in its positive predictions a significant portion of the time, although there is some room for improvement. Compared to simpler models like Logistic Regression and Decision Tree, Gradient Boosting performs significantly better in terms of identifying defaulters effectively.

These findings make Gradient Boosting a powerful tool for credit risk modelling, providing a good balance between capturing true defaulters (high recall) and ensuring precision, which is crucial for practical decision-making. It competes well with other ensemble models like Random Forest and XGBoost, making it a solid candidate for credit default prediction, particularly when feature interactions and non-linear relationships need to be captured effectively. This substantiates the thesis objective of leveraging advanced machine learning techniques over traditional methods to improve prediction accuracy in credit risk assessments.

5.2.2 Interpretation of Findings

The performance of the six models—XGBoost, Random Forest, Logistic Regression, Decision Tree, AdaBoost, and Gradient Boosting—highlights distinct strengths and weaknesses in predicting the probability of default of individuals. Each model's ability to handle imbalanced datasets and correctly classify defaulters plays a crucial role in determining its effectiveness for credit risk assessment.

XGBoost emerged as the top-performing model with an AUC score of 0.9425 and perfect precision for defaulters. This means the model is extremely confident in its predictions of defaulters. However, a recall of 0.77 indicates that 23% of true defaulters are missed. This highlights a key trade-off: XGBoost is highly reliable when making positive predictions, but it may still leave some true defaulters undetected.

Random Forest displayed a similarly high AUC score of 0.9823, demonstrating strong overall discriminative ability. However, its recall for defaulters stood at 0.70, lower than

that of XGBoost, meaning more true defaulters are missed. The model remains effective at separating non-defaulters from defaulters, which is supported by the high AUC, but the missed defaulters indicate potential limitations in capturing risky cases.

Gradient Boosting offered a strong balance between precision (0.80) and recall (0.80) for defaulters, resulting in an AUC score of 0.9203. Its ability to both identify a significant number of defaulters and maintain a moderate false positive rate positions it as a balanced option. It is capable of capturing complex relationships in the dataset, which accounts for its overall high performance.

In contrast, Logistic Regression and Decision Tree models were noticeably weaker in performance. Logistic Regression, with an AUC of 0.7679, lacked both precision and recall in identifying defaulters, indicating significant limitations when applied to this problem. Decision Tree, while achieving a slightly better AUC of 0.7986, also demonstrated a recall of 0.63 for defaulters, reflecting poor handling of minority classes in imbalanced datasets.

AdaBoost, on the other hand, showcased its strength with a recall of 0.80 for defaulters, indicating a strong capacity for capturing defaulters effectively. However, the precision of 0.62 reveals a downside, as it tends to produce false positives, suggesting overestimation of risk. This high recall-low precision scenario means AdaBoost is effective at finding defaulters but may also incorrectly flag non-defaulters.

A significant finding from the comparison of these models is the value of recall for defaulters. Models like AdaBoost and Gradient Boosting displayed higher recall, making them particularly useful for settings where identifying defaulters is crucial, even if this comes with increased false positives. In contrast, XGBoost and Random Forest provided higher precision, which makes them suitable for scenarios where false positives are costly and must be minimized.

To add depth to the analysis, a matrix comparing correct predictions for each pair of models was generated (Figure 22). This matrix reveals how frequently each pair of models agrees on predicting defaulters. The strongest overlaps are observed between Gradient Boosting and AdaBoost, which have 22 shared correct predictions, and between XGBoost and both AdaBoost and Gradient Boosting, each having 21 shared correct predictions. These overlaps indicate that these models exhibit similar predictive strengths for identifying defaulters. This consistency suggests a potential benefit in combining these models through ensemble voting approaches to enhance the stability and reliability of the predictions. The notable agreement between XGBoost, Gradient Boosting, and AdaBoost highlights their robustness and reliability in identifying high-risk individuals.

Overall, the interpretation of these findings reveals a distinct trade-off between precision and recall in each model, underlining the need for a tailored approach depending on the context. When aiming to reduce false positives, XGBoost appears ideal. On the other hand, AdaBoost or Gradient Boosting might be more appropriate if capturing as many defaulters as possible is the goal, even if some increase in false positives is acceptable.

5.2.3 Strengths and Limitations

The conducted analysis has highlighted several strengths and limitations in the models used for predicting default risk. XGBoost and Gradient Boosting were the standout models in terms of precision and AUC, indicating their ability to effectively discriminate between defaulters and non-defaulters. Their superior AUC scores and balanced performance between recall and precision demonstrate their robustness in handling complex data relationships. XGBoost particularly benefits from its ability to minimize false positives, making it well-suited for scenarios requiring high precision. Furthermore, Gradient Boosting's balance between recall and precision indicates its suitability in scenarios where capturing true defaulters without an excessive rate of false alarms is critical.

However, limitations were also observed, particularly in models like Logistic Regression and Decision Tree. Logistic Regression's performance was underwhelming due to its linear nature, which hinders its ability to deal with the complexity inherent in credit risk datasets. The Decision Tree model, while interpretable, struggled significantly in distinguishing defaulters from non-defaulters, as evidenced by its lower AUC and recall. The AdaBoost model, although exhibiting strong recall, struggled with precision, indicating a propensity for false positives that could lead to overestimation of risk.

Another major limitation across most models was related to recall—particularly for defaulters, where missing even a small percentage can be costly. While some ensemble models like Random Forest and AdaBoost demonstrated reasonably good recall, they still left a portion of true defaulters undetected. This gap, while less pronounced than in simpler models, suggests the need for continuous improvement and parameter optimization to further minimize false negatives, especially in practical credit risk assessment scenarios.

Additionally, computational complexity presents another limitation for the more sophisticated ensemble methods like XGBoost and Random Forest. While these models demonstrated superior predictive power, they come at a cost of increased computational resources and training time, which could limit their scalability in environments with restricted resources or time constraints.

The strengths of this comparative analysis lie in its holistic approach, taking into consideration different metrics beyond just accuracy, such as recall, precision, and AUC. This is critical for imbalanced datasets where accuracy alone would not provide an adequate assessment of the model's effectiveness in distinguishing defaulters. Nevertheless, future work could benefit from deeper exploration of hyperparameter optimization, leveraging techniques such as Bayesian optimization or genetic algorithms to further improve the balance between performance and interpretability.

5.2.4 Practical Implications

The findings from this study have several practical implications for financial institutions interested in credit risk management. The ensemble models, such as XGBoost, Random Forest, and Gradient Boosting, have proven to be far superior compared to traditional

methods like Logistic Regression. For financial institutions, adopting these advanced models means enhanced accuracy in default prediction, which directly impacts their ability to make informed decisions regarding loan approvals, pricing, and capital reserves. The high precision for defaulters, especially in models like XGBoost and Gradient Boosting, indicates that when these models classify an individual as a defaulter, it can be done with a high degree of confidence. This reduces the risks associated with granting loans to individuals likely to default, thereby mitigating financial losses.

However, there are trade-offs to consider. AdaBoost and Gradient Boosting, while effective in capturing defaulters (high recall), also have a risk of classifying non-defaulters as defaulters, which could result in missed opportunities for lenders who wrongly deny credit. In practice, this means that institutions using these models must carefully balance risk appetite with business opportunities, potentially adjusting thresholds to find the optimal balance between precision and recall depending on market conditions and risk tolerance.

Furthermore, the matrix of model pairing results provides valuable insights into the robustness of predictions. By examining how often different models agree on identifying defaulters, lenders can adopt ensemble voting techniques, where predictions are accepted only if multiple models concur. This could serve as a powerful method to mitigate model risk by avoiding over-reliance on a single algorithm, thus combining the strengths of multiple models to increase overall reliability.

For smaller institutions or those in need of more interpretability, Logistic Regression and Decision Tree might still have some utility, particularly in regulatory contexts where transparency is required. While these models do not perform as well in terms of AUC or recall, their transparency allows for easy interpretation, which could be beneficial for compliance and customer explanation purposes. However, this study's findings make a compelling argument for phasing out traditional models in favor of more advanced ensemble techniques whenever feasible.

Lastly, the importance of feature importance analysis highlighted through this research allows institutions to better understand key factors contributing to default, such as derogatory reports, delinquent credit lines, and recent credit inquiries. By focusing on these features, financial institutions can refine their credit scoring models and more effectively tailor risk mitigation strategies, leading to improved lending policies and more informed decision-making.

Chapter 6
Conclusion

6.1 Summary

The main objective of this thesis was to evaluate the performance of various machine learning models in predicting the probability of default (PD) for individuals seeking loans. Models such as XGBoost, Random Forest, AdaBoost, and Logistic Regression were compared to determine their efficacy. Key findings highlighted the strengths and weaknesses of each model, with ensemble models generally outperforming traditional methods. The importance of specific metrics like recall was emphasized, particularly in capturing true defaulters, which is critical in risk assessment for lending institutions.

6.2 Contribution to Knowledge

This thesis contributes to existing knowledge in multiple ways: first, by offering a comparative analysis of various machine learning models for credit risk prediction, an approach rarely seen in prior literature; second, by highlighting the significance of recall as a key evaluation metric in credit risk contexts, rather than focusing solely on accuracy. This study demonstrates that advanced ensemble models, such as XGBoost and Gradient Boosting, substantially outperform traditional models like Logistic Regression in detecting potential defaulters, thus providing insights that bridge gaps in current research.

6.3 Recommendations for Practice

For financial institutions aiming to improve their risk assessment processes, this research suggests prioritizing ensemble models like XGBoost or Gradient Boosting due to their balance of high recall and precision. Using recall as a primary metric rather than focusing solely on accuracy is critical for minimizing undetected defaults. Additionally, combining multiple models using ensemble approaches could further increase prediction reliability, given the consistency observed across models like XGBoost, Gradient Boosting, and AdaBoost.

6.4 Future Research

Current models for predicting credit default predominantly rely on objective numerical data: whether an individual has delinquencies, how many inquiries they have made, or their salary figures. These models are built on concrete, quantifiable information. Future research could expand these models to also incorporate subjective data, systematically quantified for use in machine learning.

This subjective data could come from specialized questionnaires designed by behavioral experts or psychologists, administered when a loan application is made. The questions should be formulated in such a way that they allow professionals to detect inconsistencies or potential dishonesty based on cross-referencing responses. For example, answers to certain questions must logically align with responses to others, and deviations might signal an unreliable profile.

The responses from these questionnaires must then be translated into numerical values that are suitable for input into predictive models. This way, behavioral insights—such as risk attitudes, honesty, and psychological stability—can enhance the robustness of credit risk models, potentially leading to a more holistic assessment of a borrower's likelihood of default. This approach represents a promising avenue for improving the accuracy and reliability of credit risk predictions beyond purely financial metrics.

Another important consideration is the computational power of the machine running the models. This analysis was performed on a standard PC, which imposed limitations on computational complexity as outlined in "*5.2.3 Strengths and Limitations"*. Expanding computational resources would allow for the simulation of more complex parameters with broader ranges, significantly enhancing the effectiveness of machine learning models. This would result in quicker predictions and improved optimization of results, maximizing the models' potential in handling larger datasets and complex features.

6.5 Final Observations

This study reaffirms the growing relevance of machine learning techniques in financial risk management, especially in predicting loan defaults with greater accuracy and reliability. Ensemble models like XGBoost and Gradient Boosting have shown significant advantages, yet interpretability remains a challenge that should not be ignored in practice. As credit risk management evolves, financial institutions must balance model accuracy with transparency to ensure sound decision-making that complies with regulatory standards and builds trust with stakeholders.

Chapter 7
References

Scheule, H., Roesch, D., & Baesens, B. Credit Risk Analytics: The R Companion. Scheule Roesch Baesens.

Hosmer, D.W., Lemeshow, S., & Sturdivant, R.X. (2013). Applied Logistic Regression (3rd ed.). Wiley.

Qi, X. (2024). Factors Influencing Loan Default–A Credit Risk Analysis.

Wong, J., Fung, L., Fong, T., & Sze, A. (2005). Residential Mortgage Default Risk and the Loan-to-Value Ratio.

Menard, S. (2002). Applied Logistic Regression Analysis. SAGE Publications.

Kleinbaum, D.G., & Klein, M. (2010). Logistic Regression: A Self-Learning Text (3rd ed.). Springer.

Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861-874.

Van Rijsbergen, C. J. (1979). Information Retrieval (2nd ed.). Butterworth-Heinemann.

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition, 30(7), 1145-1159.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer.

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

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing & Management, 45(4), 427-437.

Bhumika, G., Aditya, R., & Akshay, J. (2017). Analysis of Various Decision Tree Algorithms for Classification in Data Mining.

Breiman, L., Friedman, J., Stone, C.J., & Olshen, R.A. (1984). Classification and Regression Trees. Chapman & Hall/CRC.

Powers, D. M. W. (2011). Evaluation: From precision, recall and F-measure to ROC, informedness, markedness, and correlation. Journal of Machine Learning Technologies, 2(1), 37-63.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An Introduction to Statistical Learning with Applications in R. Springer.

Raschka, S. (2018). Python Machine Learning (2nd ed.). Packt Publishing Ltd.

Zhu, J., Rosset, S., Hastie, T., & Tibshirani, R. (2009). Multiclass AdaBoost. Statistics and Its Interface, 2(3), 349–360.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics, 29(5), 1189–1232.

Freund, Y., & Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences, 55(1), 119–139.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785–794).

Quinlan, J. R. (1986). Induction of Decision Trees. Machine Learning, 1(1), 81–106.

Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

Hamzic, D., Hadzajlic, N., Dizdarevic, M., & Selmanovic, E. (2023). Machine Learning for Enhanced Credit Risk Analysis: A Comparative Study of Loan Approval Prediction Models.

Shi et al. (2022). Machine learning-driven credit risk: a systemic review. (Neural Computation and Applications)

Weber et al. (2023). Applications of explainable artificial intelligence in finance.

Noriega, J. P., Rivera, L. A., & Herrera, J. A. (2023). Machine Learning for Credit Risk Prediction: A Systematic Literature Review.

Guegan, D., & Hassani, B. (2018). Credit Risk Analysis Using Machine and Deep Learning Models.

Singh, S., Saini, B., & Rani, R. (2023). A Comparative Study of Machine Learning Models for Credit Risk Prediction.

A Gentle Introduction to Imbalanced Classification - MachineLearningMastery.com

What is Normalization in Machine Learning? A Comprehensive Guide to Data Rescaling | DataCamp

SMOTE for Imbalanced Classification with Python - MachineLearningMastery.com

ML | Gestione dei dati sbilanciati con SMOTE e algoritmo Near Miss in Python - GeeksforGeeks

Parameters, Hyperparameters, Machine Learning | Towards Data Science

Introduction to Boosted Trees — xgboost 2.1.1 documentation

Random forest - Wikipedia

Chapter 8
Appendix

This appendix provides a comprehensive overview of the Python code implemented in this analysis. It not only serves as documentation of the computational methods applied but also acts as a foundation for subsequent research and analysis. The inclusion of the code ensures transparency and reproducibility, which are critical for validating results and allowing future researchers to build upon this work. Moreover, it offers opportunities for refinement and optimization, ensuring the methodology remains scalable and adaptable to evolving analytical challenges.

On the following page, a PDF extract from the Jupyter Notebook utilized in this analysis is attached, offering detailed insight into the specific code structure and execution employed throughout the study.

Cristescu Adrian Iulian

Logistic Regression of a portfolio.

The objective of this exercise is to find, with logistic regression, which on the regressors impact the most the Probability of Default (PD) after 90 days.

Dataset: 3.364 rows of full data (no blank values) The dataset contains the information of around 3300 individuals and the exercise is trying to predict the default or non default of the variable "BAD" which indicates 0 for NON-Default and 1 for Default. _____ The considered variables on the dataset are:

- BAD: 1 = applicant defaulted on loan or seriously delinquent; 0 = applicant paid loan
- LOAN: Amount of the loan request
- MORTDUE: Amount due on existing mortgage
- VALUE: Value of current property
- REASON: DebtCon = debt consolidation; HomeImp = home improvement
- JOB: Occupational categories
- YOJ: Years at present job
- DEROG: Number of major derogatory reports
- DELINQ: Number of delinquent credit lines
- CLAGE: Age of oldest credit line in months
- NINQ: Number of recent credit inquiries
- CLNO: Number of credit lines
- DEBTINC: Debt-to-income ratio

H. Scheule, D. Roesch, B. Baesens, Credit Risk Analytics: The R Companion, Scheule Roesch Baesens, 2017.

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
import numpy as np
from matplotlib.ticker import FuncFormatter
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import mean_squared_error, ConfusionMatrixDisplay
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import LabelEncoder
from statsmodels.stats.outliers_influence import variance_inflation_factor
from scipy import stats
from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score, accuracy_score, roc_auc_score, roc_curv
import warnings
import xgboost as xgb
from sklearn.metrics import ConfusionMatrixDisplay, classification_report
from scipy.stats import zscore
from imblearn.over_sampling import SMOTE , BorderlineSMOTE, ADASYN
from imblearn.combine import SMOTEENN, SMOTETomek
from sklearn.svm import SVC
from itertools import combinations
```

Let's see how many rows we have and also drop the rows with empty values

In [ ]:
```
%%time
data = pd.read_csv('hmeq.csv')
# Separate rows with 0 and 1 values in the 'BAD' column
#zeros = databeforecut[databeforecut['BAD'] == 0]
#ones = databeforecut[databeforecut['BAD'] == 1]

# Randomly sample 10% of the rows with 0 values
#sampled_zeros = zeros.sample(frac=0.32, random_state=0)
```

```
data.dropna(inplace=True)
data.info()
```

Define the categorical values: Job profession and Reason of loan

In [3]:
```
# Define manual mapping dictionaries for 'REASON' and 'JOB'
reason_mapping_manual = {'DebtCon': 0, 'HomeImp': 1}  # Assigning numbers starting from 0
job_mapping_manual = {'Other': 0, 'Office': 1, 'Mgr': 2, 'ProfExe': 3, 'Sales': 4, 'Self': 5}  # Assigning numbers starting fr

# Apply manual mapping for 'REASON' and 'JOB'
data['REASON'] = data['REASON'].map(reason_mapping_manual)
data['JOB'] = data['JOB'].map(job_mapping_manual)
```

In [ ]:
```
pd.options.display.float_format = '{:.2f}'.format  # Format float values to 2 decimals
pd.set_option('display.max_columns', None)  # Show all columns without truncating
pd.set_option('display.width', 1000)  # Control width to avoid line breaks

# Show the descriptive statistics
stats = data.describe()
print(stats)
```

Correlation Matrix

In [ ]:
```
m_data=data.corr()
m_data.style.background_gradient(cmap='viridis', axis=None, low=0, high=1, vmin=-1, vmax=1, subset=None).format("{:.2f}")
```

Let's see the difference of Defaults and Non Defaults

In [6]:
```
#remove the variable we want to regress against, which is default after 90 days
X = data.drop('BAD', axis=1) #we remove the "default>90days" variable and we plot it as dependent variable below
y = data['BAD']
```

In [ ]:
```
filtered_data = data[data['BAD'].isin([0, 1])]
value_counts = filtered_data['BAD'].value_counts()
colors = ['blue' if x == 0 else 'red' for x in value_counts.index]
plt.bar(value_counts.index, value_counts.values, color=colors)
for i, v in enumerate(value_counts.values):
    plt.text(i, v + 10, str(v), color='black', ha='center')
plt.xlabel('BAD')
plt.ylabel('Count')
plt.title('Counts of BAD Values')
plt.xticks(value_counts.index)
plt.show()
```

We start training the model. We have:

- **Training Set**: Used to train the machine learning model by providing labeled data for learning patterns and relationships between features and target outputs.
- **Validation Set**: Used to assess and tune the model during training, ensuring it generalizes well to unseen data by evaluating its performance on data not used in training.
- **Test Set**: Used to provide an unbiased evaluation of the final trained model's performance on completely unseen data, measuring how well it generalizes to new observations.

In [8]:
```
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)  # 30% test set
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.33, random_state=42, stratify=y_temp)  # 33% val,
```

- In the first sentence we say that 70% of the dataset is used for training and the rest of 30% is put aside
- in the second sentence we further split the temporary set into a validation and test set, amounting at 33%.
- The stratification parameter ensures that the class distribution in y is preserved in both splits, keeping representative samples across all sets. This ensures that since Defaults and Non-Defaults aren't uniform (different number of observations). By using stratification we enture that the proportion of observations are mantained during the training. In our case, the ratio is almost 10 to 1, hence, during the training, the sets keep this ratio.

SMOTE (Synthetic Minority Over-sampling Technique)

```
- We generate synthetic examples of the minority class by interpolating between existing minority class
instances. This technique helps balance class distribution in imbalanced datasets (ours) by enhancing the
model's ability to learn from minority class examples without introducing biaqs frim simply duplication
data points.
- The synthetic examples are generated by selecting the minority class (Default) instance at random and
```

computing the k-nearest neighbors for this instance. The examples are then created along the line segments joining these neighbors.

Step 1:

SMOTE - BORDERLINE

- We use a particular type of SMOTE: the Borderline version. This type manages better the wrong classification problem of the minority class which is located near the border between classes. These cohorts are difficult to classify, hence, they have higher probability of being classified wrongly by models. Borderline SMOTE focuses on the generation of synthetic cohorts near the borderline of minority/majority classes. It adresses the istances that are more difficult to classify.

Step 2:

Data Normalization

- StandardScaler is used to ensure all features are on the same scale, aiding machine learning models to their performance and convergence.

Step 3:

Transform back to DataFrame

- We now transform back the normalized data into a DataFrame

In [9]:
```
#Step 1
#Apply SMOTE (Synthetic minority oversampling technique)
smote=BorderlineSMOTE(sampling_strategy='minority', kind='borderline-1')
X_train_sm, y_train_sm = smote.fit_resample(X_train, y_train)

#Step 2
#normalize data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_sm)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

#Step 3
# Convert the scaled data back to DataFrame
X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_val_scaled_df = pd.DataFrame(X_val_scaled, columns=X_val.columns)
X_test_scaled_df = pd.DataFrame(X_test_scaled, columns=X_test.columns)
```

MODELS - Introduction of used terms and metrics

- **ROC Curve** : Receiver Operating Characteristic Curve
  - It is a graphical representation of the performance of a binary classifier system as its discrimination threshold is varied.
  - It plots the True Positive Rate against False Positive Rate ar various threshold settings.
  - A diagonal ROC represents random guessing, while an ideal classifier will have its ROC curve reaching the top-left corner of the plot.
- **AUC** : Area under the ROC Curve
  - It quantifies the overall performance of a binary classification model based on its ability to discriminate between positive and negative examples.
  - AUC ranges from 0 to 1, where an AUC of 0.5 indicates a model that performs no better than random, and an AUC of 1 indicates a perfect classifier.
  - A higher AUC value generally indicates a better-performing model in terms of its ability to distinguish between positive and negative classes.
- **Validation Set** : it is used during model training to tune hyperparameters. Different combinations of parameters are insertet manually until the AUC for the Validation Set is high. After the best parameters are found, they are inserted as unique parameters for the final test set.
- **Test Set** : This set is used only once after the parameters has been chosen.
- **Precision** : The proportion of true positive predictions among all positive predictions made. It indicates how many selected items are relevant:
  - $\frac{TruePositives}{TruePositives+FalsePositives}$
- **Accuracy** : The proportion of true positive and true negative predictions among all predictions made. It indicates the overall correctness of the model:

    - $\frac{TruePositives+TrueNegatives}{TotalPredictions}$
- **Recall** : The proportion of true positive predictions among all actual positive cases. It indicates how many relevant items are selected.
    - $\frac{TruePositives}{TruePositives+FalseNegatives}$
- **F1-Score** : The harmonic mean of precision and recall. It balances the two metrics, providing a single measure of a test's accuracy.
    - $\frac{Precision \times Recall}{Precision+Recall}$
- **CV - Cross Validation** : In every model, there is the "grid_search" code which join the estimators with the chosen model. There is also the metric "cv" which is the cross-validation. For every mode I've chosen 10 cv, which means it divides the set in 10 splits and performs the tests on them, averaging the result in the end

Predicted Probabilities: How are they computed?

Logistic Regression:

After training, when making predictions on new data (X_{test}), the model computes a linear combination of the input features weighted by learned coefficients. This linear combination is transformed using the logistic (sigmoid) function:

- [

$$P(y = 1 \mid X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n)}}$$

]

Explanation:

- $(\beta_0, \beta_1, \ldots, \beta_n)$ are the coefficients learned during training.
- $(X_1, X_2, \ldots, X_n)$ are the features of the new data point.

Output:

- The result of the sigmoid function gives the predicted probability [$P(y = 1 \mid X)$], which represents the model's confidence that the output (y) (default/non-default, in this case) is 1 (default) given the features (X)]

t. point. a point. point.

Decision Tree

A Decision Tree makes predictions by traversing from the root of the tree to a leaf node, making decisions at each node. The predicted probability of a class is the proportion of samples of that class in the leaf node:

[$P(y = c \mid X) = \frac{N_c}{N}$]

where N_c is the number of samples of class c in the leaf node and N is the total number of samples in the leaf node.

Random Forest

Random Forest predicts probabilities by averaging the probabilities predicted by each individual tree in the forest. For a binary classification problem:

$$\[P(y = 1 \mid X) = \frac{1}{T} \sum_{t=1}^{T} P_t(y = 1 \mid X)\]$$

where T is the number of trees in the forest and $P_t(y = 1 \mid X)$ is the probability predicted by the t-th tree.

AdaBoost

AdaBoost computes the predicted probabilities by combining the weighted predictions of weak classifiers. The weight of each classifier depends on its accuracy. The final prediction is given by:

$$\[F(x) = \sum_{m=1}^{M} \alpha_m h_m(x)\]$$

where M is the number of classifiers, α_m is the weight of the m-th classifier, and $h_m(x)$ is the prediction of the m-th classifier. The predicted probabilities are then given by:

$$\[P(y = 1 \mid X) = \frac{1}{1 + e^{-F(x)}}\]$$

Gradient Boosting

Similarly, predicts probabilities indirectly through a score $F(x)$, which is the cumulative sum of predictions from individual trees, weighted by learning rates η:

$$\[F(x) = \sum_{m=1}^{M} \eta \cdot h_m(x)\]$$

where $h_m(x)$ is the prediction from the m-th tree. Predicted probabilities are then given by:

$$\[P(y = 1 \mid X) = \frac{1}{1 + e^{-F(x)}}\]$$

XGBoost

XGBoost, like other gradient boosting methods, predicts probabilities by combining the predictions of multiple trees. The model computes a score $F(x)$, which is the sum of the predictions from individual trees weighted by the learning rate η:

$$\[F(x) = \sum_{m=1}^{M} \eta \cdot h_m(x)\]$$

where M is the number of trees, η is the learning rate, and $h_m(x)$ is the prediction from the m-th tree. The predicted probabilities are then given by:

$$\[P(y = 1 \mid X) = \frac{1}{1 + e^{-F(x)}}\]$$

XGBoost model:

XGB is a boosting machine learning model with a great performance. It's the first model considered since it has some PROs:

- **High Predictive Accuracy**: it has a superior performance in predictive accuracy. It handles complex relationships and dependencies in data through ensemble learning techniques like gradient boosting
- **Missing Values**: it handles very well the missing values. Reduces the preprocessing effots.
- **Learning**: After the first predictions, the residuals are calculated and a new model is trained to predict those errors. The new predictions are added to the old model. This process in as many times as indicated. Below in "param_grid_xgb" we have n_estimators=450, which is the number of boosting rounds.
- **Flexibility**: can handle a variety of data types and can be used for classification and regression.
- **Scalability**: it can handle large datasets and can train them faster
- **Regularization**: includes regularization techniques such as L1 and L2 ti prevent overfitting and improve generalization.
  - **L1**: adds the abs. value of the coefficients to the loss function. It prodices sparse models, meaning it drives some coefficients to exactly zero. It is useful for identifying the most important features in the dataset. If β are the model coefficients, it adds $\lambda \sum_i |\beta_i|$ to the loss function where λ is a regularization parameter that controls the strenght of the penalty
  - **L2**: adds the squared values of the coefficients $\lambda \sum_i \beta_i$ to the loss function. It distributes the penalty more evenly across all coefficients, shrinking the towards zero but not necessarily making theme xactly zero. This helps to keep all features in the model, reducing the risk of overfitting without eliminating any features.
- **Feature Importance**: It helps to identify which features have the most impact on predictions and understand the underlying relationships in data

In [ ]:
```
%%time
#XGBoost model
param_grid_xgb = { #the parameters were chosen manually after repeated attemps with different values. These gave the best AUC
    'learning_rate': [0.04, 0.05, 0.03, 0.09], #values tipically between 0.01 and 0.3: For lower learning rate it provides mor
                           # can produce overfitting. The opposite happens with higher learning rate.
    'n_estimators': [500, 400, 450], #number of boosting round
    'max_depth':[14, 10, 12] #is the maximum depth of a tree. It controls the complexity of the model. Bigger the values the m
                    #but it can lead to overfitting
}

# Initialize the XGBoost model
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)

# Set up the grid search with cross-validation
grid_search = GridSearchCV(estimator=xgb_model, param_grid=param_grid_xgb, scoring='accuracy', cv=10, n_jobs=-1)

# Fit the GridSearchCV to the training data
grid_search.fit(X_train_scaled_df, y_train)

# Get the best parameters from the grid search
best_params = grid_search.best_params_
print(f"Best parameters found: {best_params}")
```

```
# Get the best estimator
best_xgb_model = grid_search.best_estimator_
# Validate the model on the validation set
y_val_pred_xgb = best_xgb_model.predict(X_val_scaled_df)
y_val_pred_proba_xgb = best_xgb_model.predict_proba(X_val_scaled_df)[:, 1]

# Print classification report for validation set
print("Validation Set Classification Report")
print(classification_report(y_val, y_val_pred_xgb))

# Print AUC score for validation set
print(f"AUC (Validation): {roc_auc_score(y_val, y_val_pred_proba_xgb):.4f}")
```

In [ ]:
```
%%time
# Initialize the XGBoost model with the best parameters extracted from GridSearchCV
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='auc', random_state=42, **best_params_xgb)

# Train the model on the training data
xgb_model.fit(X_train_scaled_df, y_train_sm)

# Evaluate the model on the test data
y_test_pred_xgb = xgb_model.predict(X_test_scaled_df)
y_test_pred_proba_xgb = xgb_model.predict_proba(X_test_scaled_df)[:, 1]

# Print classification report for test set
print("Test Set Classification Report")
print(classification_report(y_test, y_test_pred_xgb))

# Print AUC score for test set
print(f"AUC (Test): {roc_auc_score(y_test, y_test_pred_proba_xgb):.4f}")
```

In [ ]:
```
# Calculate ROC curve for test set
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_proba_xgb)

# Calculate AUC score
auc_score = roc_auc_score(y_test, y_test_pred_proba_xgb)

# Feature importances for XGBoost
xgb_importances = best_xgb_model.feature_importances_
xgb_df = pd.DataFrame(data=xgb_importances, index=X_train_scaled_df.columns, columns=['Value']).sort_values('Value', ascending

# Create a 1x2 subplot layout
fig, axs = plt.subplots(1, 2, figsize=(16, 5))

# Plot ROC curve
axs[0].plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {auc_score:.4f})')
axs[0].plot([0, 1], [0, 1], color='gray', linestyle='--')  # Plotting the diagonal line for random model
axs[0].set_xlim([0.0, 1.0])
axs[0].set_ylim([0.0, 1.05])
axs[0].set_xlabel('False Positive Rate (FPR)')
axs[0].set_ylabel('True Positive Rate (TPR)')
axs[0].set_title('Receiver Operating Characteristic (ROC) Curve')
axs[0].legend(loc='lower right')

# Plot feature importances
axs[1].barh(xgb_df.index, xgb_df['Value'], color='blue')
axs[1].set_xlabel('Feature Importance')
axs[1].set_ylabel('Feature Name')
axs[1].set_title('XGBoost Feature Importances')


plt.tight_layout()
plt.show()
```

In [ ]:
```
# Function to plot impact of top features on default probability
def plot_feature_impact(top_features, model_name):
    plt.figure(figsize=(15, 3))  # Adjust figsize as needed
    num_features = len(top_features)
    num_cols = 3  # Number of columns for subplots

    # Calculate number of rows needed
    num_rows = (num_features // num_cols) + (num_features % num_cols > 0)

    for i, feature in enumerate(top_features):
        plt.subplot(num_rows, num_cols, i + 1)
        sns.regplot(x=X[feature], y=y, logistic=True, ci=None, scatter_kws={"s": 10, "alpha": 0.5})
        plt.title(f'Impact of {feature} on Default ({model_name})')
        plt.xlabel(feature)
        plt.ylabel('Probability of Default')
        plt.tight_layout()

    plt.tight_layout()
    plt.show()

# Example usage for XGradient Boosting (GB) model
top_features_xgb = xgb_df.head(3).index  # Select top 3 features for Gradient Boosting
plot_feature_impact(top_features_xgb, 'Extreme Gradient Boosting')
```

In [ ]:
```
# Compute confusion matrix for test set
cm = confusion_matrix(y_test, y_test_pred_xgb)

# Create ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Non-Default', 'Default'])

# Plot confusion matrix with annotations
fig, ax = plt.subplots(figsize=(6, 4))
disp.plot(cmap='Blues', ax=ax, values_format='.4g')

plt.title('Confusion Matrix - XGBoost Classifier (Test Set)')
plt.tight_layout()
plt.show()
```

Random Forest Classifier model:

It is an ensemble learning method that constructs a multitude of decision trees during training.

- **Bootstrap Sampling**: Each tree in the Random Forest is trained on a bootstrap sample of the data, which involves randomly selecting samples with replacement from the training set.
- **Feature Randomness**: At each split in the decision tree, a random subset of features is considered. This randomness helps to decorrelate the trees and reduce overfitting.
- **Voting Mechanism**: For classification tasks, the final prediction of the Random Forest is determined by a majority vote (mode) of the predictions of its constituent trees.
- **Handling of Missing Values**: It handles missing values in the dataset by using surrogate splits in the trees, reducing the need for imputation or preprocessing.
- **Scalability**: It is scalable and can efficiently handle large datasets and high-dimensional feature spaces. The training process can be parallelized across multiple processors.
- **Robust to Overfitting**: By averaging predictions across multiple trees (bagging) and introducing randomness in feature selection and tree construction, Random Forest tends to generalize well and is less prone to overfitting compared to individual decision trees.
- **Easy to Tune**: Although it has several hyperparameters (e.g., number of trees, maximum depth of trees), it is relatively easier to tune compared to some other complex models like neural networks.
- **Handling Imbalanced Data**: It handles imbalanced datasets by balancing class weights or using techniques like SMOTE (Synthetic Minority Over-sampling Technique) during training, which is beneficial for classification tasks with unequal class distributions.

Validation set

In [ ]:
```
%%time
#Random Forest model
param_grid_rf = {
    'n_estimators': [385, 380], #Higher values can improve performance at the cost of increased computational
                                #resources and potential overfitting. Here, we are testing 370 and 380 trees.
                                #Controls the number of decision trees in the forest.

    'max_depth': [None, 5, 1],#A deeper tree can model more complex relationships but increases the risk of overfitting

    'min_samples_split': [2],#Higher values prevent the model from learning overly specific patterns, potentially reducing ove

    'min_samples_leaf': [1 ],# Similar to min_samples_split, higher values can prevent the model from becoming too specific.
    'criterion': ['gini'],  # Criterion for splitting (you can test both)
    'max_features': ['sqrt', None]  # Number of features to consider at each split
}

# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Set up the grid search with cross-validation
grid_search = GridSearchCV(estimator=rf_model,param_grid=param_grid_rf, scoring='roc_auc', cv=10, n_jobs=-1, verbose=2, error_

# Fit the GridSearchCV to the training data
grid_search.fit(X_train_scaled_df, y_train_sm)

# Get the best parameters from the grid search
best_params_rf = grid_search.best_params_
print(f"Best parameters found: {best_params_rf}")

# Get the best estimator
best_rf_model = grid_search.best_estimator_

# Validate the model on the validation set
y_val_pred_rf = best_rf_model.predict(X_val_scaled_df)
y_val_pred_proba_rf = best_rf_model.predict_proba(X_val_scaled_df)[:, 1]

# Print classification report for validation set
print("Validation Set Classification Report")
print(classification_report(y_val, y_val_pred_rf))
```

```
# Print AUC score for validation set
print(f"AUC (Validation): {roc_auc_score(y_val, y_val_pred_proba_rf):.4f}")
```

Test set

In [ ]:
```
%%time
# Initialize the RandomForestClassifier with the best parameters extracted from GridSearchCV
rf_model = RandomForestClassifier(random_state=42, **best_params_rf)

# Train the model on the training data
rf_model.fit(X_train_scaled_df, y_train_sm)

# Evaluate the final model on the test data
y_test_pred_rf = rf_model.predict(X_test_scaled_df)
y_test_pred_proba_rf = rf_model.predict_proba(X_test_scaled_df)[:, 1]

# Print classification report for test set
print("Test Set Classification Report")
print(classification_report(y_test, y_test_pred_rf))

# Print AUC score for test set
print(f"AUC (Test): {roc_auc_score(y_test, y_test_pred_proba_rf):.4f}")
```

In [ ]:
```
# Calculate ROC curve for test set
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_proba_rf)

# Calculate AUC score
auc_score = roc_auc_score(y_test, y_test_pred_proba_rf)

# Feature importances for Random Forest
rf_importances = best_rf_model.feature_importances_
rf_df = pd.DataFrame(data=rf_importances, index=X_train_scaled_df.columns, columns=['Value']).sort_values('Value', ascending=F

# Create a 1x2 subplot layout
fig, axs = plt.subplots(1, 2, figsize=(16, 5))

# Plot ROC curve
axs[0].plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {auc_score:.4f})')
axs[0].plot([0, 1], [0, 1], color='gray', linestyle='--')  # Plotting the diagonal line for random model
axs[0].set_xlim([0.0, 1.0])
axs[0].set_ylim([0.0, 1.05])
axs[0].set_xlabel('False Positive Rate (FPR)')
axs[0].set_ylabel('True Positive Rate (TPR)')
axs[0].set_title('Receiver Operating Characteristic (ROC) Curve')
axs[0].legend(loc='lower right')

# Plot feature importances
axs[1].barh(rf_df.index, rf_df['Value'], color='green')
axs[1].set_xlabel('Feature Importance')
axs[1].set_ylabel('Feature Name')
axs[1].set_title('Random Forest Feature Importances')

plt.tight_layout()
plt.show()
```

In [ ]:
```
# Function to plot impact of top features on default probability
def plot_feature_impact(top_features, model_name):
    plt.figure(figsize=(15, 3))  # Adjust figsize as needed
    num_features = len(top_features)
    num_cols = 3  # Number of columns for subplots

    # Calculate number of rows needed
    num_rows = (num_features // num_cols) + (num_features % num_cols > 0)

    for i, feature in enumerate(top_features):
        plt.subplot(num_rows, num_cols, i + 1)
        sns.regplot(x=X[feature], y=y, logistic=True, ci=None, scatter_kws={"s": 10, "alpha": 0.5})
        plt.title(f'Impact of {feature} on Default ({model_name})')
        plt.xlabel(feature)
        plt.ylabel('Probability of Default')
        plt.tight_layout()

    plt.tight_layout()
    plt.show()

# Example usage for Gradient Boosting (GB) model
top_features_rf = rf_df.head(3).index  # Select top 3 features for Gradient Boosting
plot_feature_impact(top_features_rf, 'Random Forest')
```

In [ ]:
```
# Compute confusion matrix for test set
cm = confusion_matrix(y_test, y_test_pred_rf)

# Create ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Non-Default', 'Default'])

# Plot confusion matrix with annotations
```

```
fig, ax = plt.subplots(figsize=(6,4))
disp.plot(cmap='Blues', ax=ax, values_format='.4g')

plt.title('Confusion Matrix - Random Forest (Test Set)')
plt.tight_layout()
plt.show()
```

Logistic Regression Model

It is a linear model that models the relationship between the dependent binary variable (Default and Non Default) and one or more indipendent variables using a logistic function

- **Probabilistic Model**: It models the probability that an instance belongs to a particular class (usually the positive class) using the logistic (sigmoid) function, which maps any real-valued input to a value between 0 and 1.
- **Binary Classification**: Logistic Regression is primarily used for binary classification tasks where the response variable is categorical with two levels (e.g., yes/no, 0/1).
- **Coefficient Interpretation**: The coefficients (weights) learned by Logistic Regression represent the influence of each feature on the probability of the outcome. Positive coefficients indicate a positive association with the log-odds of the outcome, while negative coefficients indicate a negative association.
- **Log-Likelihood Optimization**: it maximizes the likelihood function (or equivalently minimizes the negative log-likelihood) to estimate the optimal coefficients. This is typically done using optimization algorithms like Gradient Descent.
- **Regularization**: Can include regularization techniques like L1 (Lasso) and L2 (Ridge) regularization to penalize large coefficients and prevent overfitting. This is controlled by the regularization parameter (C or λ).

CONS

- **Feature Scaling**: Logistic Regression is sensitive to the scale of the features. It is common practice to scale features to a similar range (e.g., using StandardScaler) before fitting the model to ensure stable and optimal performance.
- **Assumptions**: Logistic Regression assumes a linear relationship between the independent variables and the log-odds of the dependent variable. It also assumes that there is little or no multicollinearity among the independent variables.

Validation set

In [ ]:
```
%%time

# Define parameter grid for Logistic Regression
param_grid_lr = {
    'C': [0.5, 0.35, 0.6, 0.7, 0.4],  # Regularization parameter, by attempts, 0.7 is the best and 0.1 is the next best one
    'penalty': ['l1', 'l2']# Regularization type, the more efficient is the l1.
}

# Initialize Logistic Regression model
logistic_reg = LogisticRegression(max_iter=5000, solver='liblinear', random_state=42)

# Set up the grid search with cross-validation
grid_search_lr = GridSearchCV(estimator=logistic_reg, param_grid=param_grid_lr, scoring='roc_auc', cv=10, n_jobs=-1)

# Fit the GridSearchCV to the training data
grid_search_lr.fit(X_train_scaled_df, y_train_sm)

# Get the best parameters from the grid search
best_params_lr = grid_search_lr.best_params_
print(f"Best parameters found: {best_params_lr}")

# Get the best estimator
best_lr_model = grid_search_lr.best_estimator_
# Validate the model on the validation set
y_val_pred_lr = best_lr_model.predict(X_val_scaled_df)
y_val_pred_proba_lr = best_lr_model.predict_proba(X_val_scaled_df)[:, 1]

# Print classification report for validation set
print("Logistic Regression Validation Set Classification Report")
print(classification_report(y_val, y_val_pred_lr))

# Print AUC score for validation set
print(f"AUC (Validation): {roc_auc_score(y_val, y_val_pred_proba_lr):.4f}")
```

Test set

In [ ]:
```
%%time
# Initialize the RandomForestClassifier with the best parameters extracted from GridSearchCV
lr_model = LogisticRegression(max_iter=5000, solver='saga', random_state=42, **best_params_lr)

# Train the model on the training data
lr_model.fit(X_train_scaled_df, y_train_sm)

# Evaluate the final model on the test data
y_test_pred_lr = lr_model.predict(X_test_scaled_df)
```

```
y_test_pred_proba_lr = lr_model.predict_proba(X_test_scaled_df)[:, 1]

# Print classification report for test set
print("Logistic Regression Test Set Classification Report")
print(classification_report(y_test, y_test_pred_lr))

# Print AUC score for test set
print(f"AUC (Test): {roc_auc_score(y_test, y_test_pred_proba_lr):.4f}")
```

In [ ]:
```
# Calculate ROC curve for test set
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_proba_lr)

# Calculate AUC score
auc_score = roc_auc_score(y_test, y_test_pred_proba_lr)

# Get the coefficients of the model
coefficients = np.append(best_lr_model.intercept_, best_lr_model.coef_.flatten())
logistic_df = pd.DataFrame(data=coefficients, index=['Intercept'] + [col for col in X_train_scaled_df.columns], columns=['Valu

# Create a 1x2 subplot layout
fig, axs = plt.subplots(1, 2, figsize=(16, 6))

# Plot ROC curve
axs[0].plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {auc_score:.4f})')
axs[0].plot([0, 1], [0, 1], color='gray', linestyle='--')  # Plotting the diagonal line for random model
axs[0].set_xlim([0.0, 1.0])
axs[0].set_ylim([0.0, 1.05])
axs[0].set_xlabel('False Positive Rate (FPR)')
axs[0].set_ylabel('True Positive Rate (TPR)')
axs[0].set_title('Receiver Operating Characteristic (ROC) Curve')
axs[0].legend(loc='lower right')

# Plot logistic regression coefficients
axs[1].barh(logistic_df.index, logistic_df['Value'], color='salmon')
axs[1].set_xlabel('Coefficient Value')
axs[1].set_ylabel('Feature Name')
axs[1].set_title('Logistic Regression Coefficients')

plt.tight_layout()
plt.show()
```

In [ ]:
```
# Function to plot impact of top features on default probability
def plot_feature_impact(top_features, model_name):
    plt.figure(figsize=(15, 3))  # Adjust figsize as needed
    num_features = len(top_features)
    num_cols = 3  # Number of columns for subplots

    # Calculate number of rows needed
    num_rows = (num_features // num_cols) + (num_features % num_cols > 0)

    for i, feature in enumerate(top_features):
        plt.subplot(num_rows, num_cols, i + 1)
        sns.regplot(x=X[feature], y=y, logistic=True, ci=None, scatter_kws={"s": 10, "alpha": 0.5})
        plt.title(f'Impact of {feature} on Default ({model_name})')
        plt.xlabel(feature)
        plt.ylabel('Probability of Default')
        plt.tight_layout()

    plt.tight_layout()
    plt.show()

# Example usage for Gradient Boosting (GB) model
top_features_lr = logistic_df.head(3).index  # Select top 3 features for Gradient Boosting
plot_feature_impact(top_features_lr, 'Logistic Regression')
```

In [ ]:
```
# Compute confusion matrix for test set
cm = confusion_matrix(y_test, y_test_pred_lr)

# Create ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Non-Default', 'Default'])

# Plot confusion matrix with annotations
fig, ax = plt.subplots(figsize=(6,4))
disp.plot(cmap='Blues', ax=ax, values_format='.4g')

plt.title('Confusion Matrix - Logistic Regression (Test Set)')
plt.tight_layout()
plt.show()
```

Decision Tree Model

It is a supervised machine learning algorithm used for both classification and regression tasks. It is a flowchart-like structure where each internal node represents a "decision" on a feature (or

attribute), each branch represents the outcome of that decision, and each leaf node represents the final decision or the prediction label (class or value).

- **Binary Splitting**: Decision Trees use recursive binary splitting to partition the data into subsets based on the values of features that best separate the target variable.
- **Non-linear Model**: They can capture non-linear relationships between features and the target variable, making them suitable for complex datasets.
- **Feature Importance**: They can rank features by their importance in predicting the target variable, based on metrics such as information gain (for classification) or variance reduction (for regression).
- **Handling Mixed Data Types**: Decision Trees can handle both numerical and categorical data without requiring feature scaling.
- **Splitting Criteria**: Common criteria for splitting nodes include Gini impurity and entropy for classification tasks, and variance reduction or mean squared error for regression tasks.

CONS

- **Overfitting**: Without proper constraints (like pruning or limiting tree depth), Decision Trees can overfit the training data, capturing noise and details that do not generalize well to unseen data.

Validation Set

In [ ]:
```
%%time
param_grid_dt = {
    'max_depth': [None, 10, 20, 1, 2], #Maximum depth of the tree. It controls the maximum number of levels in the decision tr
                            #None: The tree is expanded until all leaves are pure or until all leaves contain less than min_sa
                            #20: Limits the maximum depth to 30 levels.

    'min_samples_split': [2, 3, 10], #This param. ensures that a node in the decision tree must have at least 2 samples to be
                                    #considered for further splitting.
                                    #It helps control the complexity of the tree and can prevent the model from learning noise

    'min_samples_leaf': [1,2,5,10] #This parameter sets the minimum number of samples required to be at a leaf node.
                            #It ensures that each leaf node contains a sufficient number of samples to make robust predictions
                            #from being too specific to the training data.
    }
# Initialize the DecisionTreeClassifier
dt_model = DecisionTreeClassifier(random_state=42)

# Initialize the GridSearchCV with the DecisionTreeClassifier and param_grid_dt
grid_search = GridSearchCV(estimator=dt_model, param_grid=param_grid_dt, cv=10, n_jobs=-1, verbose=1)

# Fit the GridSearchCV to the training data
grid_search.fit(X_train_scaled_df, y_train_sm)

# Get the best parameters from the grid search
best_params_dt = grid_search.best_params_
print(f"Best parameters found: {best_params_dt}")

# Get the best model from the grid search
best_dt_model = grid_search.best_estimator_

# Validate the model on the validation set
y_val_pred_dt = best_dt_model.predict(X_val_scaled_df)
y_val_pred_proba_dt = best_dt_model.predict_proba(X_val_scaled_df)[:, 1]

# Print classification report for validation set
print("Decision Tree Validation Set Classification Report")
print(classification_report(y_val, y_val_pred_dt))

# Print AUC score for validation set
print(f"AUC (Validation): {roc_auc_score(y_val, y_val_pred_proba_dt):.4f}")
```

Test Set

In [ ]:
```
%%time
#Initialize the DecisionTreeClassifier
dt_model = DecisionTreeClassifier(random_state=42,**best_params_dt)

# Fit the GridSearchCV to the training data
dt_model.fit(X_train_scaled_df, y_train_sm)

# Validate the model on the validation set
y_test_pred_dt = dt_model.predict(X_test_scaled_df)
y_test_pred_proba_dt = dt_model.predict_proba(X_test_scaled_df)[:, 1]

# Print classification report for validation set
print("Decision Tree Validation Set Classification Report")
print(classification_report(y_test, y_test_pred_dt))

# Print AUC score for validation set
print(f"AUC (Validation): {roc_auc_score(y_test, y_test_pred_proba_dt):.4f}")
```

In [ ]:
```
# Calculate ROC curve for test set
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_proba_dt)

# Calculate AUC score
auc_score = roc_auc_score(y_test, y_test_pred_proba_dt)

# Feature importances for Decision Tree
dt_importances = best_dt_model.feature_importances_
dt_df = pd.DataFrame(data=dt_importances, index=X_train_scaled_df.columns, columns=['Value']).sort_values('Value', ascending=F

# Create a 1x2 subplot layout
fig, axs = plt.subplots(1, 2, figsize=(16, 5))

# Plot ROC curve
axs[0].plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {auc_score:.4f})')
axs[0].plot([0, 1], [0, 1], color='gray', linestyle='--')  # Plotting the diagonal line for random model
axs[0].set_xlim([0.0, 1.0])
axs[0].set_ylim([0.0, 1.05])
axs[0].set_xlabel('False Positive Rate (FPR)')
axs[0].set_ylabel('True Positive Rate (TPR)')
axs[0].set_title('Receiver Operating Characteristic (ROC) Curve')
axs[0].legend(loc='lower right')

# Plot Decision Tree feature importances
axs[1].barh(dt_df.index, dt_df['Value'], color='green')
axs[1].set_xlabel('Feature Importance')
axs[1].set_ylabel('Feature Name')
axs[1].set_title('Decision Tree Feature Importances')

plt.tight_layout()
plt.show()
```

In [ ]:
```
# Function to plot impact of top features on default probability
def plot_feature_impact(top_features, model_name):
    plt.figure(figsize=(15, 3))  # Adjust figsize as needed
    num_features = len(top_features)
    num_cols = 3  # Number of columns for subplots

    # Calculate number of rows needed
    num_rows = (num_features // num_cols) + (num_features % num_cols > 0)

    for i, feature in enumerate(top_features):
        plt.subplot(num_rows, num_cols, i + 1)
        sns.regplot(x=X[feature], y=y, logistic=True, ci=None, scatter_kws={"s": 10, "alpha": 0.5})
        plt.title(f'Impact of {feature} on Default ({model_name})')
        plt.xlabel(feature)
        plt.ylabel('Probability of Default')
        plt.tight_layout()

    plt.tight_layout()
    plt.show()

# Example usage for Gradient Boosting (GB) model
top_features_dt = dt_df.head(3).index  # Select top 3 features for Gradient Boosting
plot_feature_impact(top_features_dt, 'Decision Tree')
```

In [ ]:
```
# Compute confusion matrix for test set
cm = confusion_matrix(y_test, y_test_pred_dt)

# Create ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Non-Default', 'Default'])

# Plot confusion matrix with annotations
fig, ax = plt.subplots(figsize=(6,4))
disp.plot(cmap='Blues', ax=ax, values_format='.4g')

plt.title('Confusion Matrix - Decision Tree (Test Set)')
plt.tight_layout()
plt.show()
```

AdaBoost Model

It is an ensemble learning algorithm that combines multiple weak learners, typically decision stumps, to create a strong predictive model by iteratively focusing on and correcting the errors of previous learners.

- **Boosting Performance**: normally gives a high accuracy by combining the strenghts of multiple weak learners and it reduces variance and bias
- **Robust against overfitting**: By focusing on the hardest-to-classify samples, AdaBoost inherently reduces the risk of overfitting, especially when used with simple weak learners.

- **Handling Imbalanced Data**: It assigns higher weights to misclassified instances, which helps in handling imbalanced datasets where certain classes are underrepresented.

CONS

- **Sensitive to Noisy Data**:can be very sensitive to noisy data and outliers, as it will assign higher weights to these misclassified instances, potentially degrading model performance.
- **Computational Complexity** : The iterative nature of AdaBoost, where each weak learner is trained sequentially, can lead to longer training times compared to other algorithms, especially with large datasets.

Validation Set

In [ ]:
```
%%time
param_grid_ada_val = {'n_estimators': [530, 520, 536, 538],'learning_rate': [1.72, 1.73, 1.69,1.68]}

# Initialize AdaBoost model
ada_model = AdaBoostClassifier(algorithm='SAMME.R',random_state=42)

# Set up the grid search with cross-validation
grid_search_ada_val = GridSearchCV(estimator=ada_model, param_grid=param_grid_ada_val, scoring='roc_auc', cv=10, n_jobs=-1, ve

# Fit the GridSearchCV to the training data
grid_search_ada_val.fit(X_train_scaled_df, y_train_sm)

# Get the best parameters from the grid search
best_params_ada_val = grid_search_ada_val.best_params_
print(f"Best parameters found: {best_params_ada_val}")

# Get the best estimator
best_ada_model_val = grid_search_ada_val.best_estimator_

# Validate the model on the validation set
y_val_pred_ada = best_ada_model_val.predict(X_val_scaled_df)
y_val_pred_proba_ada = best_ada_model_val.predict_proba(X_val_scaled_df)[:, 1]

# Print classification report for validation set
print("Validation Set Classification Report")
print(classification_report(y_val, y_val_pred_ada))

# Print AUC score for validation set
print(f"AUC (Validation): {roc_auc_score(y_val, y_val_pred_proba_ada):.4f}")
```

Test Set

In [ ]:
```
%%time
ada_model = AdaBoostClassifier(algorithm='SAMME.R',random_state=42)
# Fit the GridSearchCV to the training data
ada_model.fit(X_train_scaled_df, y_train_sm)

# Evaluate the final model on the test data
y_test_pred_ada =ada_model.predict(X_test_scaled_df)
y_test_pred_proba_ada =ada_model.predict_proba(X_test_scaled_df)[:, 1]

# Print classification report for test set
print("Test Set Classification Report")
print(classification_report(y_test, y_test_pred_ada))

# Print AUC score for test set
print(f"AUC (Test): {roc_auc_score(y_test, y_test_pred_proba_ada):.4f}")
```

In [ ]:
```
# Calculate ROC curve for AdaBoost
fpr_ada, tpr_ada, thresholds_ada = roc_curve(y_test, y_test_pred_proba_ada)

# Plotting ROC curve and confusion matrix
plt.figure(figsize=(14, 6))

# Plotting ROC curve for AdaBoost
plt.subplot(1, 2, 1)
plt.plot(fpr_ada, tpr_ada, color='blue', lw=2, label='ROC curve (AdaBoost)')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')  # Plotting the diagonal line for random model
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR)')
plt.title('Receiver Operating Characteristic (ROC) Curve - AdaBoost')
plt.legend(loc='lower right')

# Calculate and print AUC score for AdaBoost
auc_score_ada = roc_auc_score(y_test, y_test_pred_proba_ada)
plt.text(0.7, 0.3, f'AUC Score: {auc_score_ada:.4f}', ha='center', va='center', transform=plt.gca().transAxes)

# Compute confusion matrix for AdaBoost
cm_ada = confusion_matrix(y_test, y_test_pred_ada)
```

```
# Create ConfusionMatrixDisplay object for AdaBoost
disp_ada = ConfusionMatrixDisplay(confusion_matrix=cm_ada, display_labels=['Non-Default', 'Default'])

# Plot confusion matrix with annotations for AdaBoost
plt.subplot(1, 2, 2)
disp_ada.plot(cmap='Blues', ax=plt.gca(), values_format='.4g')
plt.title('Confusion Matrix - AdaBoost (Test Set)')
plt.tight_layout()

plt.show()
```

Gradient Boosting Model

It is used for regression and classification tasks. It builds an ensemble of decision trees in a stage-wise fashion, where each subsequent tree attempts to correct the errors of the previous trees.

- **High Predictive Accuracy**: delivers superior predictive performance compared to single decision trees or less complex ensemble methods.
- **Flexibility**: Capable of handling a variety of loss functions (e.g., regression, classification, ranking).
- **Feature Importance**: Provides insights into the importance of features in the dataset, aiding in feature selection and understanding model behavior.
- **Handles Various Data Types**: Works with both numerical and categorical data.

CONS

- **Computationally Intensive**:Training gradient boosting models can be time-consuming and resource-intensive, especially with large datasets.
- **Prone to Overfitting**: Without proper tuning, gradient boosting models can overfit to the training data, leading to poor generalization on unseen data.
- **Sensitivity to Noisy Data**: Gradient Boosting can be sensitive to noisy data and outliers, which might necessitate additional data preprocessing steps.

Validation Set

In [ ]:
```
%%time
# Define parameter grid for Gradient Boosting
param_grid_gb_val = {
    'n_estimators': [200, 210], # Refers to the number of boosting stages (or trees) that will be iteratively built.
                                # Each boosting stage improves the model by correcting errors of the previous stages.
                                # After a certain amount of estimators, there's good chances of overfitting.
                                # higher values makes the model more complex but also we risk overfitting

    'learning_rate': [0.25, 0.3], #controls the contribution of each tree to the model. It scales the contribution
                                # of each tree by multiplying the predictions of each tree by this learning rate.
                                # lower learning rate makes the model more robust by shrinking the contribution of each tr
                                # overfitting.

    'max_depth': [10, 11]       # Determines the maximum depth of each tree in the boosting process. Depth is the number of no
                                # the root to the farthest leaf node.
                                # Helps prevent overfitting. A deeper tree can model more complex relationships in the dat
                                # memorize noise in the training data, reducing generalization to new data.
}

# Initialize Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=42)

# Set up the GridSearchCV with the model and parameter grid
grid_search_val = GridSearchCV(estimator=gb_model, param_grid=param_grid_gb_val, scoring='roc_auc', cv=10, n_jobs=-1)

# Fit the GridSearchCV to the training data
grid_search_val.fit(X_train_scaled_df, y_train_sm)

# Get the best parameters from the grid search
best_params_gb = grid_search_val.best_params_
print(f"Best parameters found: {best_params_gb}")

# Get the best model from the grid search
best_gb_model_val = grid_search_val.best_estimator_

# Make predictions on the test data
y_val_pred_gb_val = best_gb_model_val.predict(X_val_scaled_df)
y_val_pred_proba_gb_val = best_gb_model_val.predict_proba(X_val_scaled_df)[:, 1]

# Print classification report for validation set
print("Validation Set Classification Report")
print(classification_report(y_val, y_val_pred_gb_val))
```

```
# Print AUC score for validation set
print(f"AUC (Validation): {roc_auc_score(y_val, y_val_pred_proba_gb_val):.4f}")
```

Test Set

In [ ]:
```
# Initialize Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=42)

# Fit the GridSearchCV to the training data
gb_model.fit(X_train_scaled_df, y_train_sm)

# Evaluate the final model on the test data
y_test_pred_gb_test = gb_model.predict(X_test_scaled_df)
y_test_pred_proba_gb_test = gb_model.predict_proba(X_test_scaled_df)[:, 1]

# Print classification report for test set
print("Test Set Classification Report")
print(classification_report(y_test, y_test_pred_gb_test))

# Print AUC score for test set
print(f"AUC (Test): {roc_auc_score(y_test, y_test_pred_proba_gb_test):.4f}")
```

In [ ]:
```
# Calculate ROC curve for test set
fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_proba_gb_test)

# Calculate AUC score
auc_score = roc_auc_score(y_test, y_test_pred_proba_gb_test)

# Feature importances for Gradient Boosting
gb_importances = gb_model.feature_importances_
gb_df = pd.DataFrame(data=gb_importances, index=X_train.columns, columns=['Value']).sort_values('Value', ascending=False)

# Create a 1x2 subplot layout
fig, axs = plt.subplots(1, 2, figsize=(16, 5))
# Plot ROC curve
axs[0].plot(fpr, tpr, color='blue', lw=2, label=f'ROC curve (AUC = {auc_score:.4f})')
axs[0].plot([0, 1], [0, 1], color='gray', linestyle='--')  # Plotting the diagonal line for random model
axs[0].set_xlim([0.0, 1.0])
axs[0].set_ylim([0.0, 1.05])
axs[0].set_xlabel('False Positive Rate (FPR)')
axs[0].set_ylabel('True Positive Rate (TPR)')
axs[0].set_title('Receiver Operating Characteristic (ROC) Curve')
axs[0].legend(loc='lower right')

# Plot Gradient Boosting feature importances
axs[1].barh(gb_df.index, gb_df['Value'], color='green')
axs[1].set_xlabel('Feature Importance')
axs[1].set_ylabel('Feature Name')
axs[1].set_title('Gradient Boosting Feature Importances')

plt.tight_layout()
plt.show()
```

In [ ]:
```
# Function to plot impact of top features on default probability
def plot_feature_impact(top_features, model_name):
    plt.figure(figsize=(15, 3))  # Adjust figsize as needed
    num_features = len(top_features)
    num_cols = 3  # Number of columns for subplots

    # Calculate number of rows needed
    num_rows = (num_features // num_cols) + (num_features % num_cols > 0)

    for i, feature in enumerate(top_features):
        plt.subplot(num_rows, num_cols, i + 1)
        sns.regplot(x=X[feature], y=y, logistic=True, ci=None, scatter_kws={"s": 10, "alpha": 0.5})
        plt.title(f'Impact of {feature} on Default ({model_name})')
        plt.xlabel(feature)
        plt.ylabel('Probability of Default')
        plt.tight_layout()

    plt.tight_layout()
    plt.show()

# Example usage for Gradient Boosting (GB) model
top_features_gb = gb_df.head(3).index  # Select top 3 features for Gradient Boosting
plot_feature_impact(top_features_gb, 'Gradient Boosting')
```

In [ ]:
```
# Compute confusion matrix for test set
cm = confusion_matrix(y_test, y_test_pred_gb_test)

# Create ConfusionMatrixDisplay object
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Non-Default', 'Default'])

# Plot confusion matrix with annotations
fig, ax = plt.subplots(figsize=(6, 4))
disp.plot(cmap='Blues', ax=ax, values_format='.4g')
```

```
plt.title('Confusion Matrix - Gradient Boosting Classifier (Test Set)')
plt.tight_layout()
plt.show()
```

Some analysis regarding the output

Following, we can see the dataset TEST with:

- First column is the id of the individual
- Second column is the actual default/non default
- Each sequent column is the predicted default/non default of each model

In [ ]:
```
y_test_actual = y_test  # Actual default/non-default values from the test set
predictions={
    'Actual': y_test_actual,
    'XGBoost': y_test_pred_xgb,
    'RandomForest': y_test_pred_rf,
    'LogisticRegression': y_test_pred_lr,
    'Decision Tree': y_test_pred_dt,
    'AdaBoost': y_test_pred_ada,
    'Gradient Boosting': y_test_pred_gb_test
}

predictions_df = pd.DataFrame(predictions)
from IPython.display import display
display(predictions_df)
```

In [ ]:
```
# Filter the DataFrame for defaulted observations (Actual == 1)
defaulted_df = predictions_df[predictions_df['Actual'] == 1]

# Get model names
model_names = ['XGBoost', 'RandomForest', 'LogisticRegression', 'Decision Tree', 'AdaBoost', 'Gradient Boosting']

# Initialize a DataFrame to hold the number of correct predictions for model pairs
pair_matrix = pd.DataFrame(index=model_names, columns=model_names, data=0)

# Calculate the number of correct predictions for each pair of models
for model1, model2 in combinations(model_names, 2):
    # Get predictions for the pair of models
    pred1 = defaulted_df[model1]
    pred2 = defaulted_df[model2]

    # Calculate the number of correct predictions where both models are correct
    correct_predictions = np.sum((pred1 == defaulted_df['Actual']) & (pred2 == defaulted_df['Actual']))

    # Store the number of correct predictions in the matrix
    pair_matrix.loc[model1, model2] = correct_predictions
    pair_matrix.loc[model2, model1] = correct_predictions

# Display the matrix
print(pair_matrix)
```

In [ ]:
```
# Filter the DataFrame for defaulted observations (Actual == 1)
defaulted_df = predictions_df[predictions_df['Actual'] == 1]

# Get model names
model_names = ['XGBoost', 'RandomForest', 'LogisticRegression', 'Decision Tree', 'AdaBoost', 'Gradient Boosting']

# Initialize a DataFrame to hold the number of correct predictions for model pairs
pair_matrix = pd.DataFrame(index=model_names, columns=model_names, data=0)

# Calculate the number of correct predictions for each pair of models
for model1, model2 in combinations(model_names, 2):
    # Get predictions for the pair of models
    pred1 = defaulted_df[model1]
    pred2 = defaulted_df[model2]

    # Calculate the number of correct predictions where both models are correct
    correct_predictions = np.sum((pred1 == defaulted_df['Actual']) & (pred2 == defaulted_df['Actual']))

    # Store the number of correct predictions in the matrix
    pair_matrix.loc[model1, model2] = correct_predictions
    pair_matrix.loc[model2, model1] = correct_predictions

fig, ax = plt.subplots(figsize=(10, 8))
cax = ax.matshow(pair_matrix, cmap='viridis')

# Add color bar
fig.colorbar(cax)

# Set axis labels
ax.set_xticks(np.arange(len(model_names)))
```

```
ax.set_yticks(np.arange(len(model_names)))
ax.set_xticklabels(model_names)
ax.set_yticklabels(model_names)

# Rotate the x labels
plt.xticks(rotation=90)

# Annotate each cell with the numeric value
for i in range(len(model_names)):
    for j in range(len(model_names)):
        ax.text(j, i, pair_matrix.iloc[i, j], va='center', ha='center')

plt.title('Number of Correct Predictions for Each Pair of Models')
plt.show()
```

In [ ]:

```
# Function to add counts on bars
def add_bar_labels(ax, bars):
    for bar in bars:
        yval = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2, yval, int(yval), va='bottom', ha='center')

# Plot for REASON distribution
total_reason = data['REASON'].value_counts().sort_index()
default_reason = data[data['BAD'] == 1]['REASON'].value_counts().sort_index()

fig, ax = plt.subplots(figsize=(8, 5))
total_bars = ax.bar(total_reason.index - 0.2, total_reason, width=0.4, label='Total', align='center')
default_bars = ax.bar(default_reason.index + 0.2, default_reason, width=0.4, label='Defaulters', align='center')

# Adding labels and title for REASON
ax.set_xticks([0, 1])
ax.set_xticklabels(['DebtCon', 'HomeImp'])  # Assuming your REASON labels are DebtCon = 0, HomeImp = 1
ax.set_ylabel('Count')
ax.set_title('Distribution of Total vs Defaulters for REASON')
ax.legend()

# Add counts on the bars for REASON
add_bar_labels(ax, total_bars)
add_bar_labels(ax, default_bars)

plt.tight_layout()
plt.show()

# Plot for JOB distribution
total_job = data['JOB'].value_counts().sort_index()
default_job = data[data['BAD'] == 1]['JOB'].value_counts().sort_index()

fig, ax = plt.subplots(figsize=(10, 6))
total_bars = ax.bar(total_job.index - 0.2, total_job, width=0.4, label='Total', align='center')
default_bars = ax.bar(default_job.index + 0.2, default_job, width=0.4, label='Defaulters', align='center')

# Adding labels and title for JOB
job_labels = ['Other', 'Office', 'Mgr', 'ProfExe', 'Sales', 'Self']  # JOB labels as per your mapping
ax.set_xticks(range(6))
ax.set_xticklabels(job_labels)
ax.set_ylabel('Count')
ax.set_title('Distribution of Total vs Defaulters for JOB')
ax.legend()

# Add counts on the bars for JOB
add_bar_labels(ax, total_bars)
add_bar_labels(ax, default_bars)

plt.tight_layout()
plt.show()
```

In [ ]:

In [ ]: