PRAGUE UNIVERSITY OF ECONOMICS AND BUSINESS

FACULTY OF FINANCE AND ACCOUNTING

Department of Banking and Insurance

Application of Machine Learning Models within Credit Risk Modelling

Master's thesis

Author: Bc. Petr Nguyen

Study program: Banking and Insurance

Minor track: Data Engineering

Supervisor: prof. PhDr. Petr Teplý, Ph.D.

Year of defense: 2023

Declaration of Authorship

I, as an author, hereby declare that I wrote and compiled the Master's thesis *"Application of Machine Learning Models within Credit Risk Modelling"* independently, using only the resources and literature listed in bibliography.

25th May 2023, Prague

Petr Nguyen

Abstract

This Master's thesis deals with the custom machine learning (ML) implementation framework that was developed in Python and applied to the application scoring data of US home equity loans (HMEQ). The ML framework involves eight classification models, namely Logistic Regression, Decision Tree, Gaussian Naive Bayes, K-Nearest Neighbors, Random Forest, Gradient Boosting, Support Vector Machine, and Neural Network. It further consists of data exploration, data preprocessing using ADASYN oversampling and Optimal Binning with Weight-of-Evidence, a custom feature selection algorithm that utilizes both Bayesian Optimization and Forward Sequential Feature Selection, and a custom model selection algorithm employed based on Bayesian Optimization and weighted ranking of individual metric ranks. In this thesis, metrics such as F1 score, MCC, AUC, Kolmogorov-Smirnov Distance, Somers' D, and others, are evaluated. Instead of using the standard classification threshold of 0.5, an optimal threshold is calculated using Youden index. The final model is Gradient Boosting trained on the features selected by Neural Network. Such model is further recalibrated and evaluated using both model performance assessment and black-box model explainability inspection. The final model is deployed as a web application using Flask and HTML, which requires filling in the loan application form and outputs the loan approval result, probability of default, and LIME plot, i.e., local explainability of the black-box model around the given prediction.

Keywords: Machine Learning, Credit Risk, Loans, Probability of Default, Python, Web Application, Bayesian Optimization

Abstrakt

Tato diplomová práce se zabývá implementací vlastního rámce strojového učení (ML), který byl vyvinut v Pythonu a aplikován na aplikační scoringová data amerických hypotečních úvěrů (HMEQ). Tento ML rámec zahrnuje 8 klasifikačních modelů, jmenovitě logistickou regresi, rozhodovací strom, Gaussovský naivní Bayes, k-nejbližších sousedů, náhodný les, gradientní boosting, model podpůrných vektorů a neuronovou síť. Dále je zde zahrnutá explorace dat, zpracování dat pomocí ADASYN a optimálního binningu s Weight-of-Evidence, vlastní algoritmus pro výběr prediktorů využívající Bayesovskou optimalizaci a Forwardovou sekvenční selekci prediktorů a vlastní algoritmus pro výběr finálního modelu na základě Bayesovské optimalizace a váženého rankingu podle jednotlivých metrik. V této práci jsou evaluovány metriky jako F1 skóre, MCC, AUC, Kolmogorovova-Smirnovova vzdálenost, Somersovo D a další. Namísto použití standardního klasifikačního prahu 0,5 se optimální prahová hodnota vypočítává pomocí Youdenova indexu. Finálním vybraným modelem je gradientní boosting trénovaný na prediktorech vybraných neuronovou sítí. Tento model je dále rekalibrován a evaluován na základě vyhodnocení výkonnosti modelu a inspekce vysvětlitelnosti black-box modelu. Finální model je nasazen jako webová aplikace využívající Flask a HTML, do které se vyplní formulář žádosti o úvěr a která pak vrátí výsledek o schválení úvěru, pravděpodobnost defaultu a LIME - lokální vysvětlitelnost black-box modelu okolo dané predikce.

Klíčová slova: Strojové učení, Machine Learning, Kreditní riziko, Úvěry, Pravděpodobnost defaultu, Python, Webová aplikace, Bayesovská optimalizace

Acknowledgments

I, as an author, would like to express my deepest gratitude and appreciation to my supervisor, prof. PhDr. Petr Teplý, Ph.D., for his help and significant advices throughout my thesis. Furthermore, I would like to also thank to my family for an enormous support during my studies.

Contents

List of Tables

List of Figures

Acronyms

Acc. Accuracy

ADASYN Adaptive Synthetic Sampling

AUC Area Under the Curve

BCBS The Basel Committee on Banking Supervision

BS Brier Score Loss

CCP Cost–Complexity Pruning

CDF Cumulative Distribution Function

CPU Central Processing Unit

CSS Cascading Style Sheets

DT Decision Tree

EAD Exposure At Default

ECL Expected Credit Loss

EI Expected Improvement

FASB Financial Accounting Standard Board

FN False Negative

FP False Positive

FPR False Positive Rate

GB Gradient Boosting

GNB Gaussian Naive Bayes

GP Gaussian Process

HMEQ Home Equity Data Set

HTML Hypertext Markup Language

IASB The International Accounting Standard Board

IAS 39 International Accounting Standard 39

IFRS 9 International Financial Reporting Standard 9

IRB Internal Ratings Based Approach

KNN K–Nearest Neighbors

KS Kolmogorov–Smirnov Distance

LGD Loss Given Default

LIME Local Interpretable Model–Agnostic Explanations

LR Logistic Regression

MCC Matthews Correlation Coefficient

ML Machine Learning

MLP Multi-Layer Perceptron

MSE Mean Squared Error

NN Neural Network

PD Probability of Default

Prec. Precision

Rec. Recall

ReLU Rectified Linear Unit

RF Random Forest

RFE Recursive Feature Elimination

ROC Receiver Operating Characteristic

SD Somers' D

SFS Sequential Forward Selection

SHAP SHapley Additive exPlanations

SICR Significant Increase In Credit Risk

SMOTE Synthetic Minority Oversampling Technique

SVM Support Vector Machine

Tanh Hyperbolic Tangent

Thres. Threshold

TN True Negative

TP True Positive

TPR True Positive Rate

WoE Weight–of–Evidence

Chapter 1

Introduction

As the backbone of most financial institutions, credit risk modelling plays an integral role as it revolves around assessing the likelihood that a borrower will default on a debt. Conventional methodologies for credit risk assessment have relied on statistical techniques and rule-based systems, however, with the exponential growth of data in recent years, where data-driven decision making is becoming the norm, machine learning is becoming increasingly important in the financial industry as it can bring a competitive edge (PwC 2023).

Therefore, the main goal of this thesis is to implement a custom machine learning framework developed in Python, that involves data exploration, data preprocessing, hyperparameter tuning, feature selection, model selection, recalibration, evaluation, and further development of a web application that deploys the machine learning model, using Flask and HTML. Such comprehensive machine learning framework is applied to an exemplary application scoring data set of US home equity loans (HMEQ) acquired from Credit Risk Analytics (Baesens *et al.* 2016). Regardless of the potential discrepancy between such exemplary data set and the current state of the loan market, the objective of this thesis is rather to showcase a versatile machine learning implementation framework that can be employed for diverse data sets, as a proof of concept.

Particularly, we employ eight different machine learning algorithms, namely Logistic Regression, Decision Tree, Gaussian Naive Bayes, K-Nearest Neighbors, Random Forest, Support Vector Machine, Gradient Boosting, and Neural Network. As evaluation metrics in order to assess model's performance, we use F1 score, Precision, Recall, Accuracy, Matthews Correlation Coefficient, AUC, Kolmogorov–Smirnov Distance, Somers' D, Brier Score Loss and Log Loss.

Regarding the thesis outline, Chapter 2 introduces theoretical aspects of both credit risk and machine learning. Particularly, credit risk definition, regulation and credit scoring approaches are described. Moreover, machine learning terminology, machine learning algorithms, and evaluation metrics are defined. Furthermore, more advanced machine learning techniques are outlined, such as ADASYN oversampling, Optimal Binning, Bayesian hyperparameter optimization, or Forward Sequential Feature Selection.

In Chapter 3, we conduct literature review from both credit risk modelling and machine learning fields. Moreover, we propose 5 hypotheses either based on HMEQ–based studies (Aras 2021; Zurada *et al.* 2014), or studies related to machine learning application in different sectors (de Hond *et al.* 2023; Pintelas *et al.* 2020; Wu *et al.* 2018).

Within Chapter 4, an empirical analysis is performed, i.e., we implement our machine learning framework on the HMEQ data set. Such framework applies exploration analysis, data preprocessing transformations, including data split, ADASYN oversampling, Optimal Binning and Weight of Evidence, hyperparameter tuning with Bayesian Optimization, and feature selection, which is utilized using Forward Sequential Feature Selection. Subsequently, model selection is performed based on the ranking of the proposed range of evaluation metrics. Afterwards, the final model is recalibrated, evaluated, and deployed into a production environment as a web application.

Lastly, Chapter 5 concludes the summary of results by assessing the hypotheses proposed in Chapter 3, discussing the thesis' results with the results from HMEQ–based studies, outlining the key findings and main contributions in this thesis, and proposing the author's recommendations for future research.

Chapter 2

Theoretical Background

This chapter provides an in-depth theoretical exploration of both credit risk and several areas within the field of machine learning. The former describes the basic credit risk definition, its key components, underlying regulation, and credit scoring approaches. The latter includes clarification of machine learning terminology for enhanced understanding in later chapters, an introduction to various machine learning algorithms used in the empirical analysis of this thesis, and a definition of the metrics used to evaluate machine learning algorithms.

This chapter also delves into more sophisticated machine learning topics, including the ADASYN oversampling methodology for handling imbalanced data sets, Bayesian Optimization for tuning the models' hyperparameters, Optimal Binning as a method for preprocessing data, and Forward Sequential Feature Selection as a technique for selecting the most optimal features.

2.1 Credit Risk

According to Gregory (Gregory 2012), credit risk can be defined as the risk that the client may be unable or unwilling to make a payment or fulfill loan contractual obligations, which is often known generically as default. However, default can have various forms and definitions. The most common definition of default is based on days past due (DPD), particularly, the default event occurs when the client is more than 90 days overdue on his loan (Brezigar-Masten *et al.* 2021).

Bessis (Bessis 2015) also defines a default event, which can result from the following events:

- Temporary or indefinite delay in payments;
- Restructuring of debt obligations due to the deterioration of the borrower's creditworthiness;
- Bankruptcy.

The main objective in credit risk modelling is to estimate (expected) credit loss, which has three components and is defined as follows:

$$ECL = PD \times EAD \times LGD \tag{2.1}$$

where PD is the probability of default, indicating the likelihood that the client will not meet scheduled loan payments, EAD is the exposure at default, which is the amount that is due to be repaid at the time of default, and LGD is loss given default, indicating the loss in case of default occurrence (Doumpos *et al.* 2019).

2.1.1 Regulation

Basel II/III

The Basel Committee on Banking Supervision (BCBS) introduced Capital Accords Basel II and Basel III in 2004 and 2010, respectively (Shakdwipee & Mehta 2017), with a goal to set higher risk management and internal control standards for banks (Witzany 2017). Regarding credit risk modelling, the former Accord introduced two approaches to modelling credit risk:

- **Standardized Approach** - The simplest approach, which imposes risk weights on assets based on the external credit ratings from credit rating agencies (Konno & Itoh 2016).
- **Internal Ratings Based (IRB) Approach** - A more sophisticated approach that allows the use of internal rating systems to assess parameters such as PD, LGD and EAD (Baesens *et al.* 2016). Such approach is further divided into two sub-approaches:

  - **Foundation IRB approach** - Only PD is internally estimated by a bank, while LGD and EAD are determined by the Basel Accord or provided by the local regulator (Baesens *et al.* 2016).
  - **Advanced IRB approach** - All the parameters, i.e., PD, LGD and EAD are estimated internally by the bank itself and are also subject to regulatory satisfaction (Joseph 2013).

IFRS 9

The International Accounting Standard Board (IASB), in cooperation with the Financial Accounting Standard Board (FASB), published International Financial Reporting Standard (IFRS) 9 in 2014 in order to replace IAS (International Accounting Standard) 39 (Temim 2016), which incorporates forward–looking information, i.e., the impact of macroeconomic projections on expected credit losses (Porretta *et al.* 2020). IFRS 9 also outlines a three-stage approach considering changes in credit quality since initial recognition of the financial asset (Beerbaum 2015), also known as SICR (Significant Increase In Credit Risk). According to Bellini (Bellini 2019), the stages are defined as follows:

- **Stage 1** - The financial instrument is considered to be in Stage 1 if it has not had a SICR since initial recognition or if it has low credit risk at the reporting date.
- **Stage 2** - The financial instrument is considered to be in Stage 2 if it has experienced a SICR since initial recognition but does not have objective evidence of impairment at the reporting date.
- **Stage 3** - The financial instrument is considered to be in Stage 3 if it has objective evidence of impairment at the reporting date.

In case of Stage 1, 12–month ECL is calculated, while in Stage 2 and Stage 3, lifetime ECL estimation is conducted (Gornjak 2017). While the aim of IFRS 9 regards financial reporting, which should follow a general purpose to provide information to those outside the firm to support decision-making usefulness, Basel II/III seek to decrease the frequency and cost of bank failures and protect the financial system as a whole by limiting the frequency and cost of systemic crises. (Beerbaum 2015).

For more information about the differences between IFRS 9 and Basel II/III, please, refer to the study of Beerbaum and Ahmad (Beerbaum 2015).

2.1.2 Credit Scoring

Credit scoring is an assessment of the credit quality of the client that uses techniques for finding the criteria that best discriminate between defaulters and non–defaulters (Bessis 2015). From such perspective, we distinguish two basic types of credit scoring:

Application Scoring

As the name indicates, application scoring is used to assess the creditworthiness of the client at the moment of loan application, which supports the decision whether a bank should approve or reject the loan application (Baesens *et al.* 2016). The most common variables which are used in application scoring and can be acquired from the loan application are, for instance: age, gender, marital status, income, employment status, and others (Baesens *et al.* 2016).

Behavioral Scoring

Behavioral scoring takes into account the observed client's recent or past behavior and starts once the loan has been approved (Van Gestel & Baesens 2009). In contrast to application scoring, which is static and takes into account only information from the loan application, behavioral scoring accumulates the information retrieved from the monitoring of the repayments and financial behavior of the client (Van Gestel & Baesens 2009).

2.2 Machine Learning Terminology

In order to avoid confusion and ensure clear understanding in this thesis, it is necessary to provide precise definitions for machine learning terminology that may be easily confused with statistical terminology due to their similar meanings but different naming conventions.

- **Target variable** - Dependent variable, output variable or response variable, which we want to predict or classify (Y).
- **Feature** - Predictor, attribute, independent variable or explanatory variable (X), which we want to use to predict the target variable Y. In tabular data, one feature is represented by one column.
- **Instance** - Observation, example, record, data point, case or a row in tabular data.
- **Training** - Fitting process, learning process or estimation of the model's parameters using the training data.
- **Supervised Learning** - Process reflecting the ability of an algorithm to generalize knowledge from the data having target or label cases, so that the algorithm can be used to predict new (unlabelled) cases (Berry *et al.* 2020). In other words, the learning process is supervised by the target variable, hence the model can be used to predict the target variable for new data, either using classification (when having categorical target variable) or regression (when having continuous target variable).
- **Cross–validation** - The data is split into k folds (subsets) and the model is trained k times, each time on $k-1$ folds and evaluated on the remaining fold, in order to obtain reliable estimates of the model generalization error (Tatsat *et al.* 2020), or so called cross-validation score.
- **Overfitting** - Situation, when the model has a small error during the training, but also a high test error (Forsyth 2019). The model is too complex and just purely memorizes the data on which is trained, but cannot generalize well on unseen data.
- **Black–box model** - Models which are very accurate but also very complex and hard to interpret (Pintelas *et al.* 2020).
- **White–box model** - Models which are comprehensible and transparent but are usually less accurate (Pintelas *et al.* 2020).

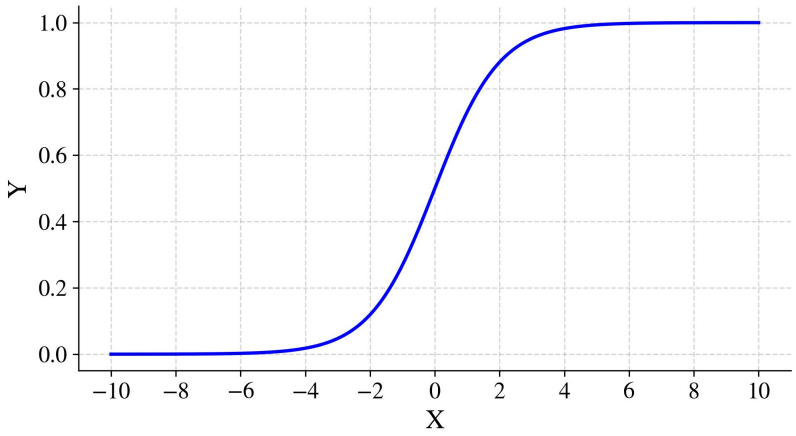
2.3 Algorithms

In this section, several algorithms, that are used in the machine learning implementation, are described. Since the goal is to predict whether or not a given client will default, henceforth only binary classification algorithms are described as part of the supervised learning. In other words, regression models and unsupervised learning algorithms are out of the scope of this thesis.

2.3.1 Logistic Regression

Despite the algorithm's name, logistic regression is not a regression algorithm, but rather a classification algorithm, as linear regression's target variable is continuous, whereas regarding logistic regression, the target variable is categorical, or rather dichotomous in the case of binary classification (Wendler & Gröttrup 2021). For a probability score prediction, it uses a logistic (sigmoid) function, which maps any real value into a probability and takes a S-shaped curve (Zaidi 2022), as can be seen in Figure 2.1.

Figure 2.1: Logistic Function



Source: Author's simulation in Python

The linear form of the logistic regression with n features can be written as:

$$\ln\left(\frac{P}{1-P}\right) = \beta_0 + \sum_{i=1}^{n} \beta_i X_i \tag{2.2}$$

where P is the probability of the event occurring, conditional on the set of given features.

Let us denote $Y = 1$ as an observed target instance where the event occurred (e.g., default), tnus:

$$P = \Pr(Y = 1 \mid X) \tag{2.3}$$

Therefore, the term within the natural logarithm is the odds, or more particularly, the ratio of the probability of the event with respect to the probability of a non-event, both conditional on the same set of given features.

$$\frac{P}{1-P} = \frac{\Pr(Y = 1 \mid X_1, X_2, \ldots, X_n)}{\Pr(Y = 0 \mid X_1, X_2, \ldots, X_n)} \tag{2.4}$$

Referring to the previous equations, solving for P, we get a final equation for computing the probability of an event occurring using logistic regression as follows:

$$P = \frac{1}{1 + e^{-\left(\beta_0 + \sum_{i=1}^{n} \beta_i X_i\right)}} \tag{2.5}$$

Regarding the loss function minimized when estimating the model's β parameters, logistic regression uses the cross-entropy loss function, also known as log loss. This loss function is also used as an evaluation metric for the model's performance, which is further described in Subsection 2.4.6. Let us denote the loss function as $L(y, p)$, where y is the actual target values and p is the predicted probability scores. To this loss function, we can also additionally employ the regularization term C in order to prevent overfitting (Pramoditha 2021) by penalizing the coefficients' magnitudes. We distinguish 3 types of regularization penalties:

- **L1** - Lasso regularization, which takes the absolute value of the coefficients' magnitudes.

$$L(y, p)_{L1} = L(y, p) + C \sum_{j=1}^{k} \mid w_j \mid \tag{2.6}$$

- **L2** - Ridge regularization, which squares the coefficients' magnitudes.

$$L(y, p)_{L2} = L(y, p) + C \sum_{j=1}^{k} w_j^2 \tag{2.7}$$

- **Elastic Net** - Combination of L1 and L2 regularizations, where α indicates the weight of $L1$ penalty.

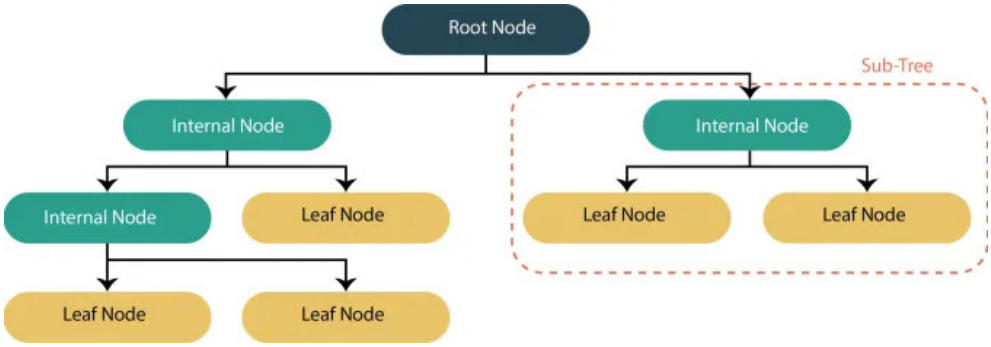
$$L(y, p)_{ElasticNet} = L(y, p) + C \sum_{j=1}^{k} \left[\alpha \mid w_j \mid + (1 - \alpha) w_j^2\right] \tag{2.8}$$

2.3.2 Decision Tree

Decision Tree is a rule–based algorithm that aims to partition the data into smaller and more homogeneous subsets. Such tree contains nodes, which are also visualized in Figure 2.2, namely:

- **Root node** - the topmost node of the tree where the splitting begins.
- **Internal node** - Non–terminal nodes that remain after the split and can be further divided into additional subsets (nodes).
- **Leaf node** - Terminal nodes that remain after the split and cannot be divided into additional subsets.

Figure 2.2: Decision Tree's Nodes



Source: (Gauhar 2020)

The splitting process is based on the homogeneity, or so called purity, of the node with respect to the target variable (Provost & Fawcett 2013). In other words, we want to have the node as pure as possible, which means that the node should contain as high a proportion of one class as possible. In this case, the node should either contain a high proportion of defaulters or non–defaulters, respectively. Such splitting process starts at the root node and continues until the leaf nodes are pure enough or until the stopping criteria are met. The stopping criteria can be either a maximum depth of the tree, a minimum number of observations within the node, or minimum number of observations within the leaf node. One way to measure impurity is using Entropy which ranges from 0 to 1 and is defined as:

$$E = -\sum_{i=1}^{n} p_i \log_2 (p_i) \tag{2.9}$$

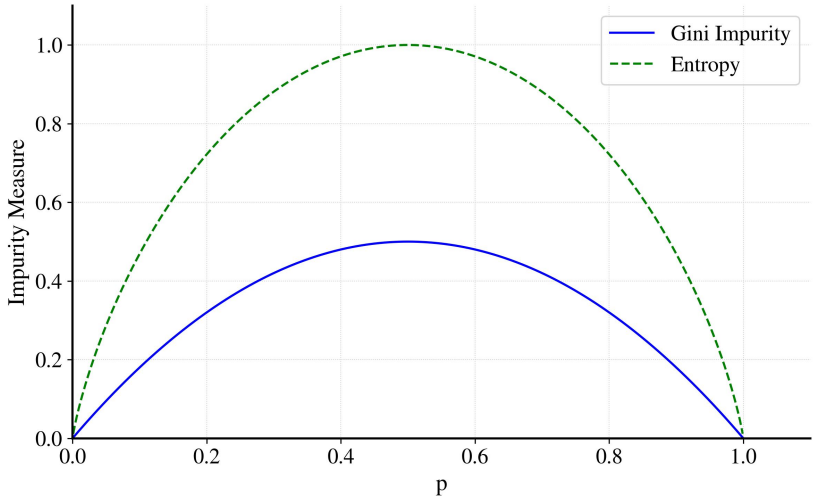
where p_i is the probability of the occurrence of the event i, or in other words, it is a fraction of the observations belonging to the class i within the given

node. In credit risk modelling terms, it is a proportion of defaulters within a given node, and $1 - p_i$ is a proportion of non–defaulters within a given sample. The lower Entropy value, the purer the node is, i.e., the more homogeneous the subset is, where the frequency of one class is dominant over the other class. Therefore, the goal is to minimize the Entropy value since we want to have the purest nodes as possible, i.e., the nodes where there is either a high proportion of defaulters or non–defaulters. However, the Entropy is not the only way to measure the impurity of the node. Another way is to use Gini Impurity which ranges from 0 to 0.5 and is defined as:

$$G = 1 - \sum_{i=1}^{n} p_i^2 \tag{2.10}$$

Both impurity measures are depicted in Figure 2.3 which we want to minimize. Thus, we choose such feature and such rule that result in the lowest impurity measure. Such process is repeated until the stopping criteria are met or until the leaf nodes are pure enough.

Figure 2.3: Gini Impurity vs. Entropy



Source: Author's simulation in Python

After the training, Decision Tree predicts the target variable based on the rules stored in the tree. The prediction process starts at the root node and continues until the leaf node is reached. Within the leaf node, the prediction can be either the most frequent class within the node or the probability of the occurrence of the event, i.e., the proportion of defaulters within the given node.

2.3.3 Naive Bayes

Naive Bayes is a classification and probabilistic machine learning algorithm that is based on the Bayes theorem and defined as follows:

$$\Pr(C = c \mid E) = \frac{\Pr(C = c) \times \Pr(E \mid C = c)}{\Pr(E)} \tag{2.11}$$

where $\Pr(C = c \mid E)$ is the posterior probability, which is the probability that the target variable C takes on the class of interest c after taking the evidence E, $\Pr(C = c)$ is the prior probability of the class c, i.e., the probability we would assign to the class c before seeing any evidence E, $\Pr(E \mid C = c)$ is the probability of seeing the evidence E conditional on the given class c, and $\Pr(E)$ is the probability of the evidence E.

With regards to the binary classification, we can substitute Y as a target variable instead of C, and a set of features X which will refer to the set of evidence E, hence:

$$\Pr(Y \mid X) = \frac{\Pr(Y) \times \Pr(X \mid Y)}{\Pr(X)} \tag{2.12}$$

One of the assumptions of this algorithm is the conditional probabilistic independence among the features. Since all the X features' values combinations do not have to appear at all, we assume their independence (Cichosz 2015). Therefore, instead of computing the probability of all features together, conditional on the class event, for each feature X we compute the conditional joint probability of X given the class event. Hence:

$$\Pr(X \mid Y) = \prod_{i=1}^{n} \Pr(X_i \mid Y) \tag{2.13}$$

Further, the second adjustment is applied to the denominator of the Bayes theorem, i.e., $\Pr(X)$ - since such probability is constant over all the values of the class event, we can omit it from the equation. Therefore, the posterior probability of the class event Y conditional on the subset of features X can be expressed as:

$$\Pr(Y \mid X) = \prod_{i=1}^{n} \Pr(X_i \mid Y) \times \Pr(Y) \tag{2.14}$$

During the training process, the Naive Bayes algorithm computes $\Pr(Y)$

and $\Pr(X \mid Y)$ for each class event Y and each feature X. In terms of default status, it calculates the proportion of defaulters $\Pr(Y = 1)$ and non–defaulters $\Pr(Y = 0)$ within given training sample, as well as the proportion of defaulters and non–defaulters within given features X.

When it comes to the predictions or classification of new instances, we use the trained Naive Bayes model, i.e., the computed probabilities $\Pr(Y)$ and $\Pr(X \mid Y)$ for both default and non–default class. Specifically, based on the new instance's features X, we determine the computed $\Pr(Y)$ and $\Pr(X \mid Y)$ for both classes, and afterwards, as a predicted class, we choose such class with the highest posterior probability. In general, the prediction while maximizing posterior probability is given as:

$$\Pr(Y \mid X) = \text{argmax}_{y \in Y} \Pr(X \mid Y = y) \times \Pr(Y = y) \tag{2.15}$$

Specifically, in terms of binary classification for prediction of a given class, whether it is default or non–default, we can compute the probability of the default event (1) and the non–default event (0), and choose the class with the higher probability.

$$\Pr(Y \mid X) = \max\left(\Pr(Y = 1 \mid X), \Pr(Y = 0 \mid X)\right) \tag{2.16}$$

where:

$$\Pr(Y = 0 \mid X) = \Pr(Y = 0) \times \prod_{i=1}^{n} \Pr(X_i \mid Y = 0) \tag{2.17}$$

respectively:

$$\Pr(Y = 1 \mid X) = \Pr(Y = 1) \times \prod_{i=1}^{n} \Pr(X_i \mid Y = 1) \tag{2.18}$$

This algorithm works well only if the features are categorical. However, most of the data sets contain continuous features as well. Therefore, we introduce **Gaussian Naive Bayes** algorithm, which is suitable for continuous features. Such algorithm assumes that the features follow a normal (Gaussian) distribution, given the class label (Jahromi & Taheri 2017). Assuming the class event $Y = 1$, we can derive the probability density function of the particular value x

of the continuous feature X as:

$$\Pr(X = x \mid Y = 1) = \frac{1}{\sqrt{2\pi\sigma^2_{X|Y=1}}} \exp\left(-\frac{\left(x - \mu_{X|Y=1}\right)^2}{2\sigma^2_{X|Y=1}}\right) \tag{2.19}$$

where $\mu_{X|Y=1}$ and $\sigma_{X|Y=1}$ is the mean and variance of X, respectively, given the class event $Y = 1$. Afterwards, $\Pr(X = x \mid Y = 1)$ derived from the Gaussian distribution can be substituted into Equation 2.18.

2.3.4 K-Nearest Neighbors

The goal of the K-nearest Neighbors algorithm (henceforth KNN) is to find k instances that are most similar to particular instances y in the n-dimensional space, where n is the number of features. The principle of this algorithm lies in the similarity between the instances, as it assumes that the similar instances are close to each other. Based on the predetermined k neighbors, it will predict the class based on the k nearest instances.

There are several ways to measure the distance. The most common one is the Euclidean distance. Geometrically, it is a straight line between the two points, and within two-dimensional space, it can be derived from the Pythagorean theorem, where the hypotenuse is the straight line measuring the distance. In n-dimensional space, we take the sum of the squared differences between the data points x and y, underneath the square root in order to compute the total Euclidean distance.

$$d_{Euclidean}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.20}$$

Another distance measure is the Manhattan distance, also known as a city block distance. In contrast to the Euclidean distance, instead of squared differences, it sums the absolute differences between the data points.

$$d_{Manhattan}(x, y) = \sqrt{\sum_{i=1}^{n}|x_i - y_i|} \tag{2.21}$$

The last measure is the Minkowski distance, which is the generalized form of the Euclidean or Manhattan distance, respectively. It depends on p which

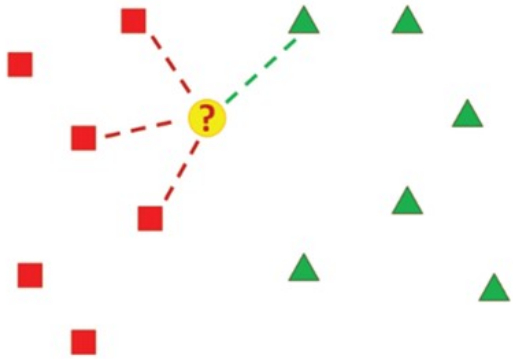
represents the order of the norm. Hence, the Euclidean distance has the second order of the norm, whereas the Manhattan distance has the first order of the norm.

$$d_{Minkowski}(x, y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{2.22}$$

Within the training process, KNN memorizes training instances, and afterwards, when it encounters a new instance, it tries to search for such training instances that most strongly resemble the new instance (Witten *et al.* 2016). Therefore, after the training process, when it comes to the prediction, the KNN compares the new instance to the training instances, calculates the distances between the new input and the training instances, and predicts the class based on the majority voting within the k nearest neighbors, or predicts the probability score as the fraction of positive instances within the k nearest neighbors.

In the following Figure 2.4, let us consider 2-dimensional space and that k is equal to 4, hence we are looking at four nearest neighbors for such new instance. Further, let us consider two classes - red squares and green triangles. By looking at the four nearest neighbors, we can observe that three out of the four nearest neighbors are the red triangles. Therefore, when applying majority voting, KNN would predict such instance as a red triangle, or would predict a probability score of 0.75 for the red triangle.

Figure 2.4: K-Nearest Neighbors with $k = 4$



Source: (Mucherino *et al.* 2009)

2.3.5 Random Forest

Random forest is an ensemble algorithm that is a collection of decision trees where each tree is independently trained on a bootstrap sample of the training data (Han *et al.* 2011), i.e., on a set that is randomly sampled from the training data with replacement and that has the same size as the training data. In such way, each bootstrap sample is unique, can contain duplicates of the original data, or does not have to contain all the original data. Another aspect of randomness in such algorithm, particularly in the variability within trees, is the number of features considered for the split at each node. Instead of considering all the features of the length M, it randomly selects a subset of features of the length m (where $m < M$) and chooses the best split from the subset. The training process for individual decision trees is the same as described in Subsection 2.3.2.

Within classifying new instances, the random forest algorithm predicts the class based on the majority voting of the individual decision trees or predicts a probability score as an average of the probability scores of the individual decision trees (Malley *et al.* 2011), thus:

$$\Pr\left(Y = 1 \mid X\right) = \frac{1}{B}\sum_{b=1}^{B}\Pr\left(Y = 1 \mid X, T_b\right) \tag{2.23}$$

where B is the number of trees and T_b is the b-th tree.

2.3.6 Gradient Boosting

In contrast to Random Forest, whose trees are trained independently, i.e., in a parallel way, Gradient Boosting's trees are trained in a sequential way, where each tree is trained on the residuals of the previous tree (Ayyadevara 2018). Such algorithm wraps a sequential building series of dependent weak learners, i.e., decision trees, in order to create a strong learner.

According to Dias (Dias *et al.* 2018), before adding any weak learners, first we need to initialize a model $F_0(x)$ which is a constant value computed by minimizing the sum of the loss function L over all training samples, i.e., we look for such γ constant that minimizes the loss function L, hence:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, \gamma\right) \tag{2.24}$$

Then, for $m \in M$, where M is the total number of iterations or the number of tree estimators:

1. Compute pseudo–reisduals for each training sample pair (x_i, y_i) as the negative gradient of the loss function L with respect to the prediction of the previous model $F_{m-1}(x)$, hence:

$$r_{i,m} = -\left[\frac{\partial L\left(y_i, F\left(x_i\right)\right)}{\partial F\left(x_i\right)}\right]_{F(x)=F_{m-1}(x)} \tag{2.25}$$

2. Train a weak regression learner $h_m(x)$ with the training data as features and the pseudo–residuals as the target values $(x_i, r_{i,m})$, and then compute the learning rate γ_m which minimizes the loss function L when adding the weak learner $h_m(x)$ to the previous model $F_{m-1}(x)$, hence:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}\left(x_i\right) + \gamma h\left(x_i\right)\right) \tag{2.26}$$

3. Update the model $F_m(x)$ by adding the weak learner $h_m(x)$ with the learning rate γ_m, hence:

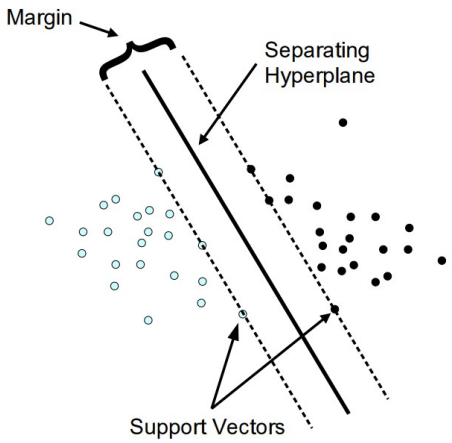
$$F_m(x) = F_{m-1}(x) + \gamma_m h(x) \tag{2.27}$$

Afterwards, such algorithm outputs the final model $F_M(x)$ as a combination of the initial model and the M weak learners.

2.3.7 Support Vector Machine

Support Vector Machine (henceforth SVM) can handle both linear and non–linear classification problems and tries to transform original training data into a higher dimension, where the class points are linearly separable by a (linear) decision boundary, also known as a hyperplane (Han *et al.* 2011). In particular, SVM tries to maximize the margin, which is the distance between the separating hyperplane and the support vectors, i.e., the training data points that are equally closest and parallel to such hyperplane (Tatsat *et al.* 2020), as shown in Figure 2.5.

Figure 2.5: Support Vector Machine 2D Hyperplane



Source: (Meyer 2009)

The hyperplane can be expressed as follows, where X is the feature data, W is the vector of the weights, and b is the intercept:

$$W \cdot X + b = 0 \tag{2.28}$$

Thus, the data points that lie above the hyperplane are classified as 1 and otherwise (Hsu & Lin 2002). Moreover, the distance between the hyperplane and any training data point is expressed as $\frac{1}{\|W\|}$, where $\|W\|$ is the Euclidean norm of the weight vector W. In order to maximize the margin, we minimize the following term as we want to maximize both sides of the hyperplane (i.e., above and below the hyperplane), therefore:

$$\min \frac{2}{\|W\|} \tag{2.29}$$

When the data are not linearly separable, SVM transforms the original data points into a higher dimensional space using a non–linear mapping function, also known as the kernel (Han *et al.* 2011). In general, the kernel function for the training tuples (X_i, X_j) can be expressed as:

$$K(X_i, X_j) = \phi(X_i) \cdot \phi(X_j) \tag{2.30}$$

where ϕ is mapping function for the high–dimensional projection of the feature space. According to Patle (Patle & Chouhan 2013), the most common kernel functions. where γ and r and d represent the kernel parameters, are:

- d**–degree polynomial kernel:**

$$K\left(X_i, X_j\right) = (1 + X_i \cdot X_k)^d \tag{2.31}$$

- **Gaussian radial basis kernel:**

$$K(X_i, X_j) = \exp\left(-\gamma \|X_i - X_j\|^2\right) \tag{2.32}$$

- **Sigmoid kernel:**

$$K(X_i, X_j) = \tanh(\gamma X_i \cdot X_j + r) \tag{2.33}$$

In some cases, the data cannot be linearly separable in higher dimensional space, i.e., we need to account for some potential errors, so we introduce the regularization factor C for penalizing the errors ξ. Therefore, in order to maximize the margin as well as allow for errors, we minimize the following term:

$$\min\left[\frac{2}{\|W\|} + C\sum_{i=1}^{N}\xi\right] \tag{2.34}$$

Hence, by minimizing $\frac{1}{2}\|W\|$ we minimize the weights (i.e., maximize the margin), by minimizing ξ we reduce the number of training errors, and by tuning C we balance between these two properties (Hsu & Lin 2002).

Note, that SVM does not predict probability scores by default, but rather predicts classes based on the location of the given data points, whether they are above or below the hyperplane. In order to obtain SVM's predicted probabilities, as stated by Lin (Lin *et al.* 2007), the Platt scaling (Platt *et al.* 1999) can be applied as a surrogate model to estimate probabilities. Particularly, it

fits logistic regression with true labels Y as target variable and predictions produced by SVM ($f(x)$) as features. Therefore, the predicted probability score can be expressed as:

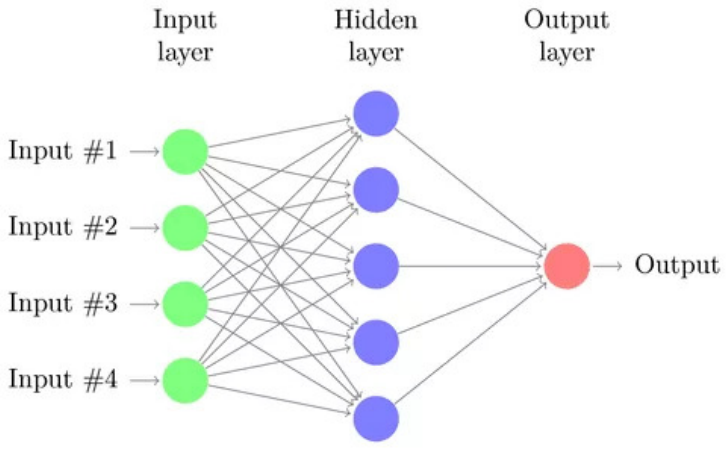
$$\Pr(Y|f(x)) = \frac{1}{1 + \exp(A \cdot f(x) + B)} \tag{2.35}$$

2.3.8 Neural Network

Neural Network is a black–box machine learning algorithm that can handle very complex and non–linear relationships. It contains several layers, where each layer has a certain number of units (neurons). In general, we distinguish 3 types of layers, as also visualized in Figure 2.6:

- **Input layer** - The initial layer that contains n units, where n is the number of features within the data set.
- **Output layer** - The final layer that outputs predictions and contains m units, where m is the number of classes within the target variable. Since we assume only binary classification algorithms, therefore the output layer contains only one unit.
- **Hidden layer** - The intermediate layer that is always between the two layers - either between the input and output layers or even between other hidden layers.

Figure 2.6: Neural Network Architecture



Source: (Dilmegani 2022)

The most common type of hidden layer is the fully connected layer, where each unit is connected to each unit of the previous layer and to each unit of the next layer. Thus, each unit receives the inputs from the previous layer, performs the computation, and feeds the output to the next layer - such a process is called *feed–forward*, hence feed–forward neural networks (Charu 2018). Within the receiving the inputs, let us assume an integration function g which performs a linear combination of the inputs x and the weights w, and adds a bias term b. Therefore, the output value z of the single unit in the hidden layer is calculated as:

$$z = g\left(X, w\right) = b + \sum_{m=1}^{M} w_m x_m \tag{2.36}$$

where M indicates the number of units in the previous layer. Before the output value z is passed to the next layer, it is transformed by an activation function f which is usually a non–linear function. The most common non–linear activation functions are:

- **ReLU** - Rectified Linear Unit (henceforth ReLU) is half–linear function that is rectified from the negative values to zero. The ReLU function is defined as:

$$f(z) = \max(0, z) \tag{2.37}$$

- **Tanh** - Hyperbolic tangent (henceforth Tanh) is another widely used non–linear activation function that reminds of the sigmoid function. The Tanh function is limited to the range between -1 and 1, and is defined as:

$$f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.38}$$

- **Sigmoid** - The sigmoid function was already defined and described in Subsection 2.3.1. Such function is usually used in the output layer as it maps the input values to the range between 0 and 1, i.e., the predicted probability for binary classification. In terms of z, the sigmoid function is defined as:

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.39}$$

The training of a neural network is an iterative process, where within each iteration (also called an epoch), the neural network is fed with the input, passes it through the hidden layers to the output. The output is then compared with

the actual value, and the error is calculated. Here comes another property of a neural network, which is back–propagation, which updates all the weights and biases starting from the last layer by a small amount until the error is minimized as much as possible (Ayyadevara 2018). Thus, the error is back–propagated through the network, and the weights are updated. The process is repeated until the error is minimized or is interrupted once the stop criterion is met.

2.4 Evaluation Metrics

This section focuses on particular measures through which it is possible to determine the predictive power of a model in terms of its performance. There are many ways to evaluate the model's performance, therefore, only the most common and relevant ones are further described. Note that since default prediction regards classification tasks, regression's evaluation metrics are omitted. If not stated otherwise, the higher the metric, the better the model's performance.

2.4.1 Confusion Matrix and Derived Metrics

The confusion matrix is a table that summarizes the classification model's performance with respect to the actual classes and predicted classes. It is a square $n \times n$ matrix, where n determines the number of classes within the target variable. Let us denote the confusion matrix as $C(f)$ for classification algorithm f. Its elements can be denoted as $c_{i,j}$ where i and j refer to the row and column indices, respectively, or more particularly, i refers to the actual class and j to the class predicted by the classifier f. Each element of the confusion matrix refers to the number of instances corresponding to actual class i and predicted class j (Japkowicz & Shah 2014). For instance, the element $c_{2,1}$ would refer to the number of instances that have the actual class 2 but have been classified as class 1. Mathematically, the confusion matrix can be written as follows:

$$C = c_{i,j} = \sum_{l=1}^{m} [(y_l = i) \wedge (f(x_l) = j)] \tag{2.40}$$

Or either in matrix form as:

$$C_{i\times j} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,j} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,j} \\ \vdots & \vdots & \ddots & \vdots \\ c_{i,1} & c_{i,2} & \cdots & c_{i,j} \end{bmatrix} \tag{2.41}$$

In the given matrix, the diagonal elements represent the numbers of correctly classified instances, whereas the non-diagonal elements represent the numbers of misclassified instances. Further, let us consider a binary classification - hence, the confusion matrix will have a form of 2×2:

$$C_{2\times 2} = \begin{bmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{bmatrix} \tag{2.42}$$

We can rewrite the confusion matrix, assuming 2 target variable classes *Positive* and *Negative*, as follows:

$$C_{2\times 2} = \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix} \tag{2.43}$$

where:

- TP is the True Positive which refers to the number of instances that correspond to the actual (*True*) class *Positive* and indeed have been correctly classified as class *Positive*.
- FP is the False Positive which refers to the number of instances that correspond to the actual (*True*) class *Negative*, but have been incorrectly classified as class *Positive*. In statistics and hypothesis–testing terms, it can also be called a Type 1 Error.
- FN is the False Negative which refers to the number of instances that correspond to the actual class *Positive*, but have been incorrectly classified as class *Negative*. In statistics and hypothesis–testing terms, it can also be called a Type 2 Error.
- TN is the True Negative which refers to the number of instances that correspond to the actual class *Negative* and indeed have been correctly classified as class *Negative*.

From such confusion matrix, we can derive several metrics, which are further described on the following pages.

Accuracy

Such metric ranges from 0 to 1 and describes, in relative terms, how many instances the model has correctly predicted. Thus, the goal is to minimize the number of False Positives and False Negatives, or, in credit risk modelling terms, number of defaulters that the model has classified as non–defaulters and the number of non–defaulters that the model has classified as defaulters. However, Accuracy is inappropriate metric for evaluation when there is an imbalanced class, i.e., where the distribution of the target variable is skewed. In such case, the model can achieve a relatively high Accuracy even though it is not able to predict the minority class correctly (Brownlee 2021), thus leading to misleading results. This is the case with credit risk modelling, when the loan portfolio most of the time has a lot of non–defaults and a few defaults. Therefore, it is deemed appropriate to consider other metrics when having an imbalanced class.

$$Accuracy = \frac{TP + FN}{TP + TN + FP + FN} \tag{2.44}$$

Recall

Such metric is also known as True Positive Rate (TPR) or Sensitivity, which also ranges from 0 to 1, and it describes, in relative, terms, how many actual *Positive* instances the model has correctly predicted out of all the actual *Positive* instances. Thus, the goal is to minimize the number of False Negatives, i.e., the number of defaulters that the model has classified as non–defaulters. A lower value of Recall could indicate that either the model is not able to correctly predict the *Positive* classes, resulting in a lower number of True Positives and/or a higher number of False Negatives. Recall metric is useful when having an imbalanced class and should be used alongside the Accuracy metric as it measures the model's ability to correctly identify instances of the minority class (i.e., defaults). Mathematically, based on the confusion matrix elements, it can be computed as:

$$Recall = \frac{TP}{TP + FN} \tag{2.45}$$

Precision

This metric describes, in relative terms, how many predicted *Positive* instances are actually *Positive* out of all the predicted *Positive* instances. Thus, the goal is to minimize the number of False Positives, i.e., the number of non–defaulters that the model has classified as defaulters. A lower value of Precision could therefore indicate that either the model is not able to correctly predict the *Positive* classes (thus a lower number of True Positives) or its prediction of *Positive* classes is too noisy (thus a high number of False Positives). Precision is another metric that should be used alongside Accuracy when having an imbalanced class, as it measures the model's ability to correctly identify instances of the minority class while minimizing false positives (i.e., non–default instances that the model has classified as default). Similarly, Precision also ranges from 0 to 1 and can be derived from the confusion matrix as follows:

$$Precision = \frac{TP}{TP + FP} \tag{2.46}$$

F1 Score

The F1 score incorporates both Recall and Precision into a single value and takes on values between 0 and 1 as well. It is defined as a weighted harmonic mean of these two metrics (Brabec *et al.* 2020), where both Recall and Precision have uniform weights, and the goal is to minimize False Positives and False Negatives at the same time as within Accuracy. Nevertheless, F1 score is deemed a more appropriate metric when dealing with imbalanced class as it provides a more unbiased evaluation of the model's performance in imbalanced data sets compared to Accuracy.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{2.47}$$

Matthews Correlation Coefficient

The drawback of Recall, Precision, and F1 score is that they are asymmetric measures, i.e., they do not take into account the True Negatives. Therefore, such metrics' values will differ if we swap the positive and negative classes (Chicco & Jurman 2020), e.g., class 1 would indicate non–default and vice versa. In order to overcome such drawback, Matthews Correlation Coefficient can be used, as it is symmetric and takes into account all four elements of the confusion matrix as well as the imbalanced class issue. Methodologically, Matthews Correlation Coefficient is defined as a discretization of Pearson correlation for the case of binary variables (Boughorbel *et al.* 2017). Pearson correlation coefficient is defined as:

$$r(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.48}$$

Thus, assuming that x is the vector of True labels and y is the vector of predictions, the Matthews Correlation Coefficient can be defined as:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.49}$$

Matthews correlation coefficient ranges from -1 to 1, where 1 indicates a perfect model's predictions, -1 indicates that the model misclassifies all the instances, and 0 indicates that the model's predictions are not better than random guessing.

2.4.2 ROC Curve and AUC

ROC Curve

In order to derive Area Under the Curve (henceforth AUC), first we need to define Receiver Operating Characteristics (henceforth ROC) curve. ROC curve is a two-dimensional visualization of the model's performance and shows the trade–off between True Positive Rate (TPR) and False Positive Rate (FPR) based on varying the given threshold (Han *et al.* 2011). The former metric was already described in Section 2.4.1, the latter refers to the ratio of the number of $Negative$ instances that are classified as $Positive$ to the total number of the actual $Negative$ instances, hence:

$$FPR = \frac{FP}{TN + FP} \tag{2.50}$$

In order to construct the ROC curve, we need to sort the instances by the predicted probability scores, and based on the given probability score, we set a threshold - what is above the threshold will be classified as $True$ instance, and what is below the threshold will be classified as $False$ instance. Thus, each probability score threshold produces a different confusion matrix and, hence, different TPR and FPR values (Fawcett 2006). Thus, if the probability is 1, the threshold will be 1 as well, and hence:

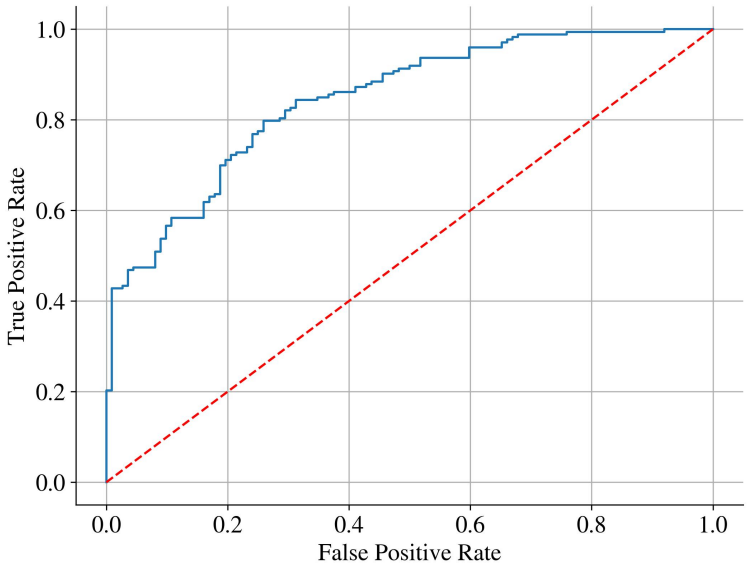
- TPR will be 0, because there is no probability higher than 1, and hence, everything will be classified as $Negative$ which will result in a TP of 0, and subsequently a TPR of 0 as well.
- FPR will be 0, too - since everything will be classified as $Negative$, therefore FP will be 0, which implies FPR to be 0, too.

On the other hand, if the probability is 0, the threshold will be 0 as well, and hence:

- TPR will be 1, because there is no probability lower than 0, and hence, everything will be classified as $Positive$ which will result in a FN of 0, and subsequently a TPR of 1.
- FPR will be 1, too - since everything will be classified as $Positive$, therefore TN will be 0 which implies FPR to be 1.

Thus, based on each threshold, the TPR and FPR will be the coordinates for a single point within the graph, and based on such points, we can construct the ROC curve, which can be depicted in Figure 2.7. Note that the diagonal line represents a random model that randomly guesses the *Positive* class half the time in such a way, that FPR and TPR are the same, i.e., 0.5 (Fawcett 2006).

Figure 2.7: ROC Curve



Source: Author's simulation in Python.

Logically, a decent model should perform better than the random model, thus it the ROC curve should be above the diagonal line. Intuitively, the best possible theoretical model would have TPR of 1 and FPR of 0, meaning that all the *Positive* actual classes should be predicted as *Positive* and all the *Negative* actual classes should not be classified as *Negative*.

AUC

AUC is basically the representation of the ROC curve as a single number, as it aggregates the performance on all possible thresholds ranging from 0 to 1. *AUC* can be interpreted as the probability that the randomly chosen actual *Positive* instance is ranked higher than the randomly chosen actual *Negative* instance (Janitza *et al.* 2013).

As the *AUC* is an area that lies underneath the ROC curve, mathematically, it can be defined with the definite integral, where x is the given threshold:

$$AUC = \int_0^1 TPR\left(FPR^{-1}(x)\right) dx \tag{2.51}$$

Since the ROC curve is a probability curve, it is considering the distribution curve of TP and the distribution curve of TN, separated by particular threshold - hence, TP should have probability scores above the given thresholds, whereas TN should have probability scores below the threshold. If these curves do not overlap, meaning the model can perfectly distinguish between the *Positive* and *Negative* values, the *AUC* would be 1 and the ROC curve would reach the left top corner.

However, this idealistic situation does not occur in practice at all, but rather the two distributions are overlapping since the misclassification of the classes takes place. The bigger the overlap, the lower *AUC* is.

If the distributions are completely overlapping, it implies an *AUC* of 0.5, meaning that the model cannot distinguish between the *Positive* and *Negative* classes, which is the worst scenario. On the other hand, if the distributions are totally opposite (meaning that the TP instances would have probability scores below the given threshold, whereas the TN instances would have probability scores above the given threshold), the *AUC* would be 0 since the model is predicting the *Positive* actual classes instead of *Negative* and vice versa (Narkhede 2018).

2.4.3 Kolmogorov-Smirnov Distance

The Kolmogorov-Smirnov (KS) Distance is a non–parametric metric for assessing discriminant power of a model as it measures distance between the cumulative distribution functions (henceforth CDF) between two classes, and is quantified as a maximum vertical absolute difference between such two CDF's (Adeodato & Melo 2016). In credit risk modelling terms, we can express KS as follows:

$$\text{Kolmogorov Smirnov} = \max_{0 \leq t \leq 1} |F_D(t) - F_{ND}(t)| \tag{2.52}$$

where F_D and F_{ND} are the cumulative distribution functions of default and non-default cases, respectively, and t is the probability score threshold, which ranges between 0 and 1. F_D and F_{ND} are defined as follows:

$$F_D(t) = \frac{1}{M_D} \sum_{i=1}^{M_D} I[f(x_i) \leq t] \tag{2.53}$$

$$F_{ND}(t) = \frac{1}{M_{ND}} \sum_{i=1}^{M_D} I[f(x_i) \leq t] \tag{2.54}$$

where M_D and M_{ND} refer to the number of default and non-default cases, respectively, and I is the indicator function, which is equal to 1 if the condition is true and 0 otherwise (Doumpos *et al.* 2019).

2.4.4 Somer's D

The Somers' D is a metric that is part of the Kendall family of ranking measures. Particularly, assuming X-Y pairs, a Kendall's τ is defined as:

$$\tau(X, Y) = \mathrm{E}[\text{sign}(X_i - X_j)\text{sign}(Y_i - Y_j)] \tag{2.55}$$

Equivalently, Kendall's τ can be defined as the difference between the probability that the two X-Y pairs are *concordant* and the probability that they are *discordant*. X-Y pair is concordant if the larger of the X values is paired with the larger of the Y values, i.e., $X_i < X_j$ and $Y_i < Y_j$. In contrast, X-Y pair is discordant if the larger of X values is associated with the smaller of Y values or vice versa, i.e., $X_i < X_j$ and $Y_i > Y_j$, or $X_i > X_j$ and $Y_i < Y_j$ (Newson 2002).

Therefore, Somers' D can be defined as the difference between the two conditional probabilities of concordance and discordance, given that the two X values are unequal (Newson 2014) as follows:

$$D\left(Y \mid X\right) = \frac{\tau\left(X, Y\right)}{\tau\left(X, X\right)} \tag{2.56}$$

In case of a binary classification, X values would represent *True* labels and Y values would represent predicted probability scores as rank vectors. Such metric ranges from -1 to +1 (likewise as Matthews correlation coefficient is +1). Thus, the higher the value of Somers' D, the better the model's ability to distinguish between borrowers who are likely to default and those who are not.

2.4.5 Brier Score Loss

Methodologically, Brier Score Loss is calculated in the same way as Mean Squared Error (henceforth MSE). However, Brier Score Loss is applied to the predicted probabilities (i.e., assumes that the target variable is dichotomous) (Comotto 2022), whereas MSE is rather used in regression tasks where there is no assumption regarding the continuous target variable. Henceforth, Brier Score Loss is defined as a mean squared error between the $True$ labels (y) and the predicted probabilities ($\hat{y}$) as follows:

$$\text{Brier Score Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.57}$$

Brier Score Loss ranges from 0 to 1, where the ideal scenario would be a Brier Score Loss of 0 - in such case, the model would have perfect predictive power.

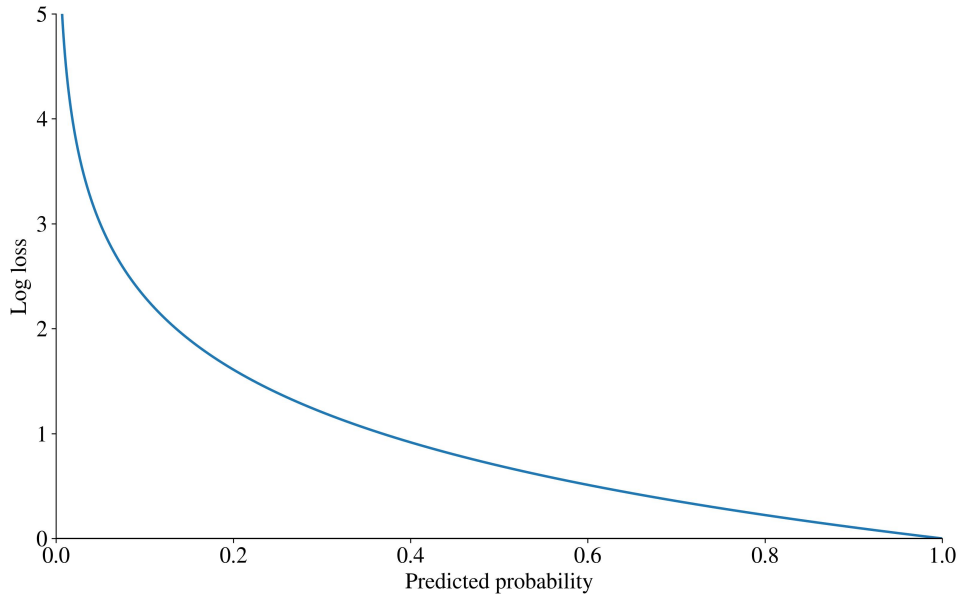
2.4.6 Log Loss

Log loss, also known as logistic loss or cross–entropy loss, is a loss function metric that takes *True* labels and predicted probabilities as input, and then minimizes the differences between these two in a logarithmic function's form. Particularly, it indicates how close the predicted probabilities are to the corresponding *True* labels. The more predicted probabilities diverge from the actual values, the more the Log Loss function penalizes the model's performance (Dembla 2020).

$$textLogLoss = -\frac{1}{N}\sum_{i=1}^{N} y_i \ln(p_i) + (1 - y_i)\ln(1 - p_i) \tag{2.58}$$

Logically, the lower Log loss value, the better the performance of the model is. As depicted in Figure 2.8, we can observe that the closer the predicted probability is to 1 with respect to the actual target class being equal to 1, the closer the Log Loss function is to 0, which is desired in terms of the model's performance.

Figure 2.8: Log Loss Function for $Y = 1$



Source: Author's simulation in Python

2.5 ADASYN Oversampling

Since this thesis pertains to the data set of performing (no–defaulted) and non–performing (defaulted) loans, whose target variable distribution is heavily skewed, it is needed to emphasize the imbalanced class issue, i.e., the frequency distribution of the majority class is significantly higher than the frequency distribution of the minority class. In the case of a loan portfolio, the majority class would be non–default status and the minority class would be default status. This can cause serious degradation in the model's performance as most of the models assume well–balanced class distributions, hence, such models would be biased towards the majority class and would not be able to predict the minority class accurately (Prati *et al.* 2009).

To overcome this issue, one would use Random Undersampling or Random Oversampling to deal with the class imbalance issue. The former randomly eliminates majority class instances, whereas the latter randomly duplicates minority class instances. However, both have particular drawbacks, as random under-sampling may lead to a significant loss of information, and random oversampling may lead to a high degree of repetition of minority instances. Both then could lead to the model's overfitting and deterioration in model's performance (He & Ma 2013). This might be a significant issue when the target variable distribution is heavily imbalanced - assume that we have a data set of 1,000 instances where 990 instances belong to the majority class and 10 instances to the minority class. If we perform random undersampling in order to balance the target variable distribution, we would have to remove 980 instances and end up with an undersampled data set of 20 instances, which is not acceptable for model training. On the other hand, if we perform random oversampling, we would end up with 1,980 instances, where 980 instances would be the same as in the original data set, which would lead to the model's overfitting.

Therefore, the Adaptive Synthetic Sampling (henceforth ADASYN) technique is used for oversampling the minority class in this thesis. ADASYN generates synthetic instances of the minority class based on the nearest neighbors of the minority class instances. Such approach is more effective than the Synthetic Minority Oversampling Technique (henceforth SMOTE) which also uses nearest neighbors to generate synthetic instances of minority class, however, ADASYN generates more synthetic instances, which are hard–to–learn

by K-Nearest Neighbors given the density distributions, whereas SMOTE generates synthetic instances uniformly for each minority class instance (He *et al.* 2008).

In other words, ADASYN generates more synthetic instances in regions where the density of the majority class within K nearest neighbors of minority instance is higher and fewer synthetic instances in regions where the density is lower, thereby ADASYN focuses on generating more hard–to–learn minority instances. Thus, it makes it easier for the machine learning model to learn the decision boundary between the minority and majority classes and boosts the model's performance by focusing on hard–to–learn instances (He *et al.* 2008).

Before the oversampling algorithm's execution, we first need to calculate the number of instances of minority classes to be synthetically generated in order to balance the target variable distribution. The number of instances to be generated G is calculated as follows:

$$G = (m_l - m_s) \times \beta \tag{2.59}$$

where m_l is the number of majority class instances, m_s is the number of minority class instances, and β indicates the desired ratio between the numbers of majority and minority class instances after oversampling.

Then, for each minority class instance x_i, using K-Nearest Neighbors with Euclidean distance, calculate the ratio r_i as:

$$r_i = \frac{\delta_i}{K} \tag{2.60}$$

where δ_i is the number of majority class instances within the K nearest neighbors of x_i. In such case, higher r_i indicates dominance of the majority class in given specific neighborhood of x_i (Nian 2018). Subsequently, all the r_i ratios are normalized as follows:

$$\hat{r}_i = \frac{ri}{\sum_{i=1}^{m_s} r_i} \tag{2.61}$$

In such a way that the sum of all the normalized $\hat{r}_i$ ratios is equal to 1. Hence, we can denote $\hat{r}_i$ as the density distribution.

$$\sum_{i=1}^{m_s} \hat{r}_i = 1 \tag{2.62}$$

Finally, the oversampling process is initialized. For each minority class instance x_i, calculate the number of instances to be synthetically generated based on the respective density distribution $\hat{r}_i$ and the total number of instances to be synthetically generated G. Thus, for the minority classes with higher density, it generates more synthetic instances, since those instances are hard–to–classify by the K-Nearest Neighbors.

$$G_i = G \times \hat{r}_i \tag{2.63}$$

Further, for each minority class instance x, generate G_i synthetic instances s_i as follows:

$$s_i = x_i + (x_{zi} - x_i) \times \lambda \tag{2.64}$$

where x_{zi} represents the randomly chosen minority class instance within the K nearest nearest neighbors for x_i, and $\lambda \in [0, 1]$ is a random number.

2.6 Optimal Binning

In the context of data preprocessing, it is crucial to consider the most appropriate feature transformation method that optimizes the performance of machine learning models. Although common approaches such as dummy encoding, standardization, logarithmic transformation, and normalization are widely used, they may not always be suitable for a given data set due to the presence of certain characteristics. For instance, dummy encoding (also known as one–hot encoding) may not be suitable for categorical features with a large number of categories as it could lead to the curse of dimensionality, therefore, it may result in a decrease in the model's performance (Bera *et al.* 2021).

Therefore, alternative approaches such as discretization, also known as binning, are increasingly being used. Binning is a categorization process to transform a continuous variable into a small set of groups or bins (Zeng 2014). This approach enables the identification of outliers and reduces their impact, as well as capturing missing values without requiring their removal or imputation.

In this thesis, we employ the `BinningProcess` from the `optbinning` module in Python for an optimal binning of both numeric and categorical features with respect to the target variable, developed by Guillermo Navas–Palencia (Navas-Palencia 2020). This approach involves grouping the values of a continuous variable into discrete intervals, or "bins", based on their relationship with the target variable. Similarly, for categorical features, the approach involves grouping the categories based on their relationship with the target variable.

In general, the optimal binning is solved by iteratively merging an initial granular discretization until imposed constraints are satisfied (Navas-Palencia 2020). Particularly, the optimal binning process consists of two phases - pre–binning process (for generating initial granular discretization) and subsequent optimization (for satisfying imposed constraints). The former phase usually uses a decision tree to generate initial split points for creating pre–bins, which are further merged while maximizing Information Value (also known as Jeffreys' divergence) while accounting for the constraints. As Navas–Palencia states (Navas-Palencia 2020), such constraints are:

- **Creation of separate bins for missing values:** Since missing values can have a significant impact on the target variable, hence is important to create a separate bin for them.

- **Minimum bin size constraint:** At least 5 % of the total number of instances should be observed within given bin. Such constraint ensures a bin's representativeness.
- **Each bin should have at least one instance of each target class:** Application of such constraint allows to compute divergence measure.

Thus, using the optimal binning process, the goal is to maximize discriminatory power among bins, i.e., the divergence measure. For more information about optimal binning, please refer to (Navas-Palencia 2020).

After the binning process, the bins as categorical values need to be encoded into numeric values. The most common approach in credit risk modelling is Weight–of–Evidence (henceforth WoE) encoding. The WoE is a commonly used measure of the strength of association between a binary target variable and an independent variable. According to Witzany (Witzany 2017), the WoE of categorical value c can be expressed as the change between the overall log odds ratio and the log odds ratio given the value c, hence:

$$WoE\left(c\right) = \ln\left(\Pr\left(c \mid Y = 0\right)\right) - \ln\left(\Pr\left(c \mid Y = 1\right)\right) \tag{2.65}$$

Assuming feature X and its respective bin b_i, thus $b_i \in X$, we can express WoE in terms of b as follows:

$$WoE_{X,b} = \ln\left(\frac{\Pr\left(X = b \mid Y = 0\right)}{\Pr\left(X = b \mid Y = 1\right)}\right) \tag{2.66}$$

According to Navas–Palencia (Navas-Palencia 2020), the WoE for bin i can also be computed as:

$$WoE_i = \ln\left(\frac{r_i^{ND}/r_T^{ND}}{r_i^{D}/r_T^{D}}\right) \tag{2.67}$$

where r_i^{ND} is the number of non–default instances in the given bin i, r_i^{D} is the number of default instances in the given bin i, r_T^{ND} is the total number of non–default instances in the whole sample and r_T^{D} is the total number of default instances in the whole sample. Thus, a positive WoE value indicates a larger distribution of non–defaults, i.e., a higher likelihood of non–default, while a negative WoE value indicates a larger distribution of defaults, i.e., a higher likelihood of default.

2.7 Bayesian Hyperparameter Optimization

Most machine learning models are configured by a set of hyperparameters that are not learned during the training process, but need to be set beforehand. Such hyperparameters' values must be carefully chosen as they may considerably impact the model's performance (Bischl *et al.* 2023). It is needed to point out the difference between parameter and hyperparameter - while parameter value is learned during the training process from the data, hyperparameter value is set before the training process and cannot be learned from the data (Owen 2022). In case of logistic regression, the parameters are the estimated coefficients and intercept, while the hyperparameters are, for instance, the regularization factor, the optimization solver, etc.

Thus, it is recommended to select optimal hyperparameter values instead of relying on default hyperparameters. Nonetheless, selecting optimal hyperparameter values can be very challenging, especially when there are many hyperparameters to tune. Fortunately, in machine learning practice, there already exist several methods for choosing the best hyperparameters' values. The most common ones are Grid Search and Random Search. The former approach tries all the possible combinations of hyperparameters' values from a predetermined hyperparameter space, which can be very computationally expensive, when the number of hyperparameters is large (Marinov & Karapetyan 2019), when the space of hyperparameters' values is too wide, or when the data set is too large. Regarding the latter approach, instead of trying all the possible combinations from the grid, it randomly selects samples of hyperparameters' combinations. Although such an approach can be less computationally expensive, it does not guarantee finding the global optimum. Another issue pertaining to both approaches is that they do not consider any information from previous iterations and rather check all possible hyperparameter combinations or randomly select hyperparameter combinations, respectively.

The compromise for such drawbacks is hyperparameter tuning with Bayesian Optimization, which is able to find the best hyperparameters' values in less time and with fewer computational resources than Grid Search and also leads to better model's performance than Random Search despite the longer optimization time (Drahokoupil 2022). Bayesian Optimization is a hyperparameter tuning method that uses information from previous iterations to make more informed decisions about which hyperparameters' values to try next. The main property

of hyperparameter tuning is the objective function that we want to optimize (i.e., maximize or minimize in the case of a score or loss, respectively), whose input is the hyperparameters' values and the output is the score or loss. However, the true shape of an objective function is not known beforehand and is very computationally expensive to evaluate.

Therefore, Bayesian Optimization uses a surrogate function, which is the approximation and probability representation of the objective function. The most common surrogate function is the Gaussian Process (henceforth GP), which is able to produce accurate approximations as well as the uncertainty around the approximations (Wang 2020). According to Owen (Owen 2022), GP is utilized as the prior for the objective function along the posterior, which we want to update. In other words, the prior captures the initial belief about the objective function, while the posterior is the updated distribution combining the prior belief and the observed evaluations of the objective function, i.e., it reflects the current belief about the objective function, hence the relation between the hyperparameters' values and the score or loss that we want to maximize or minimize, respectively. GP is the generalization of the normal distribution and describes the distribution over functions (not the distribution of a random variable), hence the objective function f follows a normal distribution with noise as follows:

$$y = f(x) + \epsilon \tag{2.68}$$

Let us denote $f_{1:n} = \{f(x_1), f(x_2), \ldots, f(x_n)\}$ as a set of n observations of the objective function, where $m(x_{1:n})$ is the mean function and K is the covariance kernel. Hence, such set follows normal distribution as well:

$$f_{1:n} \sim N(m(x_{i:n}), K) \tag{2.69}$$

Henceforth, the distribution of predictions generated by GP also follows the normal distribution as follows:

$$p(y \mid x, D) \sim N\left(y \mid \mu^*, \sigma^{*2}\right) \tag{2.70}$$

where μ^* and σ^{*2} can be analytically derived from the K kernel, and D represents observed data regarding the previously evaluated iterations of hyperparameters' values.

Furthermore, Bayesian Optimization also employs an acquisition function,

which accounts for which set of hyperparameters' values should be used in the next iteration (Owen 2022). Particularly, the next hyperparameter set is chosen based on the location where the acquisition function is maximized. The most common acquisition function is the Expected Improvement (henceforth EI), which is according to Owen (Owen 2022) defined as follows assuming that $\sigma(x) \neq 0$:

$$EI(x) = \left(\mu(x) - f\left(x^{+}\right)\right) \times \Phi(Z) + \sigma(x) \times \phi(Z) \tag{2.71}$$

where $\sigma(x)$ is the uncertainty an $\mu(x)$ is the expected performance, both are captured by the surrogate model. $f(x^{+})$ indicates the current best value of the objective function, $\Phi(Z)$ and $\phi(Z)$ are the cumulative distribution function and probability density function of the standard normal distribution, respectively. Moreover, EI allows for a balance between exploration and exploitation, where the former refers to the searching for unknown regions where the objective function may have higher values, i.e., the hyperparameters' values that have not been evaluated yet, while the latter refers to the searching near the best observed values of the objective function, i.e., the hyperparameters' values that have been evaluated already. Referring to Owen (Owen 2022), higher expected performance $\mu(x)$ compared to the current best value $f(x^{+})$ will favor the exploitation process, whereas a very high uncertainty $\sigma(x)$ will support the exploration process.

2.8 Forward Sequential Feature Selection

Instead of using all the features in the data set, we use only a subset of the most relevant ones, which can further reduce the dimensionality of the data set, boost the model's performance, save computational resources, and reduce overfitting. Such process is called feature selection and can be defined as the process of detecting the most relevant features and discarding the noisy and redundant ones. (Bolón-Canedo *et al.* 2015).

With respect to the target variable, we distinguish 2 most common types of feature selection approaches: (1) filter methods and (2) wrapper methods. Using the former approach, the feature selection is independent of the machine learning model itself, but rather is performed using univariate statistical tests, such as correlation measures, Chi–square tests, Fisher scores, and others (Kaushik 2023), and chooses those with the highest or lowest values. This approach is used when we prefer lower computational costs and a faster feature selection process, but since it does not take the model itself into account, it does not have to be able to select the features that would be optimal for a particular model. Regarding the latter approach, the feature selection is based on a specific machine learning model and follows a greedy search approach by evaluating all the possible combinations of features against the evaluation metric criteria (Verma 2022). Although, such approach is very computationally expensive, it is able to select the optimal features for a particular model.

The most common wrapper method for feature selection is Recursive Feature Elimination (henceforth RFE). Such method takes a machine learning model as a base estimator, fits the model to a whole set of features, computes and ranks their coefficients or feature importances, eliminates the least important features, and repeats the process until the desired number of features is reached (Brownlee 2020) or until the stop criterion is met. However, the drawback of RFE is that it always requires feature importances or coefficients to be computed, which is not suitable for models that do not produce any coefficients or feature importances, such as KNN or Naive Bayes. Therefore, this approach is not implemented in this thesis.

Instead, the feature selection is performed with Sequential Feature Selection (henceforth SFS). Such method iteratively adds (or removes) features to the set sequentially. Hence, there are two approaches, Forward SFS and Backward SFS. The former approach keeps adding features to the set until the desired

number of features is reached, while the latter approach starts with the whole set of features and removes them until the desired number of features is reached. In this thesis, Forward SFS is used since it was way more computationally efficient than Backward SFS within the author's analysis. In particular, Forward SFS starts with an empty set of features, and afterwards, variant features are sequentially added until the desired number of features is reached or until the addition of extra features does not reduce the loss or increase the score (Verma 2021). According to Scikit-learn's documentation (scikit-learn 2023g), at each stage, SFS chooses the best feature to add based on the cross–validation score.

2.9 Theoretical Summary

In this chapter, we have introduced the theoretical background of the thesis. In particular, in Section 2.1, we have described the traditional credit risk parameters, i.e., PD, LGD, and EAD, as well as the credit risk regulation regarding Basel II/III and IFRS 9, and also scoring methods, namely application and behavioral scoring. Pertaining to the machine learning part, its terminology has been introduced in order to avoid any misunderstanding in the following chapters, as described in Section 2.2.

Furthermore, as classification algorithms, the following 8 algorithms have been presented in Section 2.3 and are also employed in the empirical analysis of this thesis: Logistic Regression, Decision Tree, Gaussian Naive Bayes, KNN, Random Forest, Gradient Boosting, SVM, and Neural Network.

For assessment of the model's performance, we have defined the following metrics in Section 2.4, which are used in the empirical analysis of this thesis: Accuracy, Precision, Recall, F1 score, Matthews Correlation Coefficient, AUC, Kolmogorov–Smirnov Distance, Somers' D, Brier Score Loss, and Log Loss.

Moreover, other advanced machine learning techniques have been presented, particularly, ADASYN oversampling for dealing with imbalanced, i.e., skewed distribution of the target variable (Section 2.5), Optimal Binning for data preprocessing using discretization with respect to the target variable (Section 2.6), Bayesian Optimization for hyperparameter tuning of models (Section 2.7), and lastly, Forward Sequential Feature Selection for selecting subsets of optimal features (Section 2.8). All these techniques are employed in the empirical analysis of this thesis as well.

Chapter 3

Literature Review

Within the literature review, our objective is to explore the application and implementation of machine learning in credit scoring. We will examine relevant studies in two areas: machine learning applications specifically related to credit scoring and machine learning implementations in various other domains. The aim is to identify machine learning techniques from non-credit scoring contexts that could potentially be useful in credit scoring. We will evaluate the impact of these techniques through hypothesis testing.

Regarding the studies related to machine learning in credit scoring, we will analyze not only the studies focused on the HMEQ data set used in this thesis, but also studies that involve different data sets. This comprehensive approach is intended to develop a broader understanding and facilitate the discovery of valuable insights and methodologies that can be applied across various data sets in the credit scoring field.

It is a known fact that the distribution of default status is imbalanced in the credit scoring domain, since the non–default cases are overrepresented compared to the default cases. Several studies have already addressed this issue. For instance, Owusu (Owusu *et al.* 2023) employed ADASYN oversampling on the peer-to-peer loans data from American Lending Club, which exhibited an improvement in the model's performance in comparison to other benchmark studies. Another way to address such an imbalance issue is by changing the classification threshold cut–off point. By default, the threshold is set to 0.5, which means that if the predicted probability score exceeds the threshold, it will be classified as default, and vice versa. Kazemi (Kazemi *et al.* 2022) demonstrated that utilization of the customized cut–off points leads to a more

accurate classification compared to the default threshold value of 0.5, based on German and Australian credit data sets available to the public in the UCI machine learning data repository.

Whether to choose a transparent, white–box model such as logistic regression or a complex, black–box model such as ensemble models or neural network depends on various factors, such as the type of data set, sample size, data preprocessing techniques, hyperparameter tuning, or even the software used to implement the model. While Teplý and Polena (Teplý & Polena 2020) suggest that Logistic Regression is the best performing model on peer-to-peer loans data from Lending Club, Aniceto (Aniceto *et al.* 2020) demonstrates that the ensemble models, namely Random Forest and AdaBoost, outperformed the Logistic Regression on a large Brazilian bank's loan data set.

In this thesis, an application scoring data set of US home equity loans (HMEQ) is analyzed, which is further described in Subsection 4.2.1. Regarding the studies pertaining to the HMEQ data set, the study of Aras (Aras 2021) and the study of Zurada (Zurada *et al.* 2014) are deemed to be the most relevant to this thesis.

With respect to the former study, the author performed imputation of missing values with mode and mean, respectively, and for an evaluation, the author used a test size of 596 instances. The author also performed an oversampling on the training set using Random Oversampling in order to balance the default status distribution. Regarding the fitted models that are also used in this thesis, the author used KNN, Random Forest (RF), SVM, Decision Tree (DT), Gaussian Naive Bayes (GNB) and Logistic Regression (LR). Such models were also tuned using Grid Search method with 10–fold cross validation.

Within the evaluation, the author assessed Accuracy (Acc.), Recall (Rec.), Precision (Prec.), F1 score, and Matthews Correlation Coefficient (MCC) metrics. The particular results are summarized in Table 3.1. Since the default distribution is imbalanced on the test set, the most relevant metrics are F1 and MCC. As can be seen, only three models exceeded the 0.8 and 0.75 threshold for F1 and MCC, respectively, namely KNN, RF and SVM. Whereas Gaussian Naive Bayes and Logistic Regression performed poorly as they did not even exceed the 0.5 threshold for F1 and MCC, respectively. However, it is not appropriate to compare the computed metrics with the results of this thesis, since the test set size is not the same, and the data preprocessing and modelling are different as well.

Table 3.1: Evaluation Results (Aras 2021)

Model	Acc.	Rec.	Prec.	F1	MCC
KNN	0.953	0.789	0.980	0.874	0.853
RF	0.930	0.789	0.858	0.822	0.779
SVM	0.926	0.756	0.869	0.809	0.766
DT	0.898	0.683	0.792	0.734	0.674
GNB	0.795	0.480	0.504	0.492	0.364
LR	0.705	0.691	0.381	0.491	0.334

Source: (Aras 2021)

If we rank the models by each computed metric in descending order, we can explicitly derive a rank score as an average of the ranks. Based on the rank scores, we can then rank the models, as shown in Table 3.2. As can be seen, KNN dominantly outperformed all the models across all the metrics, while the black–box models such as Random Forest and SVM also performed well but not as well as KNN. On the other hand, Gaussian Naive Bayes's and Logistic Regression's performances exhibited very unsatisfactory performances. Therefore, according to Aras, one would expect that KNN and Random Forest would perform well on the HMEQ data set.

Table 3.2: Evaluation Results - Ranked (Aras 2021)

Model	Acc.	Rec.	Prec.	F1	MCC	Score	Rank
KNN	1	1	1	1	1	1	1
RF	2	1	3	2	2	2	2
SVM	3	2	2	3	3	2.6	3
DT	4	4	4	4	4	4	4
GNB	5	5	5	5	5	5	5
LR	6	3	6	6	6	5.4	6

Source: Ranking of (Aras 2021)

The latter study pertains to the research conducted by Zurada (Zurada *et al.* 2014). Likewise Aras, the author also analyzed various classification models, but only the relevant ones to this thesis are further discussed. Namely, Zurada trained Neural Network (NN), Decision Tree (DT), Support Vector Machine (SVM), K–Nearest Neighbors (KNN) and Logistic Regression (LR), using Weka software. The author did not mention which imputation technique they used

or the data split ratio (i.e., the test size used for an evaluation), but likewise Aras, Zurada conducted a hyperparameter tuning using Grid Search with 10–fold cross validation.

The evaluation results are depicted in Table 3.3, namely the computed metrics such as Accuracy (Acc.), Recall (Rec.) and AUC are assessed. As can be seen, the KNN model, which was the best one in case of Aras' study, yielded a very deteriorated performance, while Logistic Regression still performed badly. On the other hand, Neural Network was the best performing model as a black–box model. Surprisingly, Decision Tree showcased relatively high performance in contrast to Aras' study, where Decision Tree's performance was quite weak.

Table 3.3: Evaluation Results (Zurada *et al.* 2014)

Model	Acc.	Rec.	AUC
NN	0.869	0.590	0.863
DT	0.889	0.548	0.844
SVM	0.848	0.346	0.810
KNN	0.791	0.334	0.826
LR	0.836	0.304	0.794

Source: (Zurada *et al.* 2014)

The same insights can be derived from ranking the results in the same way as in Aras' study, as shown in Table 3.4. Hence, one would expect that Neural Network would perform well on the HMEQ data set, while Logistic Regression's performance would be unsatisfactory.

Table 3.4: Evaluation Results - Ranked (Zurada *et al.* 2014)

Model	Acc.	Rec.	AUC	Score	Rank
NN	2	1	1	1.33	1
DT	1	2	2	1.67	2
SVM	3	3	4	3.33	3
KNN	5	4	3	4.00	4
LR	4	5	5	4.67	5

Source: Ranking of (Zurada *et al.* 2014)

Although, both Zurada and Aras ranked the Logistic Regression as the worst performing model and SVM was ranked ,according to them, as the third best

model, they did not agree on the ranking of Decision Tree and KNN models. Such a disagreement can be attributed to the fact they both used different data preprocessing techniques, different hyperparameters' spaces for fine tuning, and probably also different softwares, random seed, and data split. We also need to account for a relatively small sample size of the HMEQ data set, which can lead to a high variance in the results. Therefore, several steps and procedures are applied in this thesis to ensure reliable and robust results and estimates, which are further described in Section 4.3 and Section 4.4.

3.1 Hypotheses

In this section, we formulate the hypotheses, which are partially derived from the literature review discussed above and are further tested in Section 5.1, i.e., rejected or not rejected, based on the empirical analysis regarding the machine learning implementation in Chapter 4.

Hypothesis #1: *The recalibration of the model enhances model's performance on HMEQ data set.*

The study of de Hond (de Hond *et al.* 2023) focuses on the model re-training (recalibration) in order to improve the model's performance, based on the health record data from Leiden University Medical Center and Amsterdam University Medical Center for a prediction of readmissions or deaths after ICU discharge. Such effect is being achieved by re-training the model on the newest additional data, hence, by increasing the sample size within the model training, we expect the model's predictive power to be enhanced. Such approach is further discussed in Subsection 4.4.4 and assessed in Subsection 4.5.1.

Hypothesis #2: *Either Neural Network or KNN model outperforms all the models on HMEQ data set.*

Based on the studies of Aras (Aras 2021) and Zurada (Zurada *et al.* 2014), the highest ranked models in terms of performance were KNN and Neural Network, respectively. These two models outperformed the traditional Logistic Regression, and in case of Aras, KNN also outperformed Decision Tree, Gaussian Naive Bayes, SVM and even Random Forest. Therefore, we expect that at least one of those two models will outperform all the other models on the

HMEQ data set.

Hypothesis #3: *Black–box models perform better than white–box models on HMEQ data set.*

Following the Section 2.2, there is a trade-off between the model's performance and the interpretability, where the black–box models are more complex and less interpretable, but also more accurate, while the white–box models are less complex and more interpretable, but also less accurate (Pintelas *et al.* 2020). With respect to the algorithms described in Section 2.3, we deem Logistic Regression as a white–box model, since it is interpretable thanks to the estimated coefficients, which can indicate the strength and direction impact of each feature on the target variable, adjusting for all other features (Park 2013). Another white–box model is a Gaussian Naive Bayes as it is based on the Bayes theorem and Gaussian distribution, thus only prior probabilities and conditional means and variances need to be computed per each feature according to Equation 2.18 and Equation 2.19. Decision Tree can be considered a white–box model, assuming that the tree is small (Goethals *et al.* 2022), i.e., not too deep and wide, which enhances the comprehensible interpretation of the model.

One would say that KNN is a white–box model, since it just looks for k closest neighbors around a given data point. However, the interpretation of the model might not be straightforward if the k is large and/or has a higher number of features, which increases the dimensionality, as the model might be too complex. According to Loyala–Gonzalez (Loyola-Gonzalez 2019), KNN is considered a black–box model since it is highly dependent on the distance function, which might bias the classification results, and a change in the distance function can lead to different outcomes. Therefore, we consider KNN as a black–box model. Regarding the rest of the models, they are deemed black–box models due to the following reasonings:

- **SVM** - Very complex mathematical operations involved in such algorithm make it difficult to interpret the model (Loyola-Gonzalez 2019).
- **NN** - The more layers and units within each layer the neural network has, the less transparent and comprehensible the model is (Bathaee 2018).
- **RF, GBM** - Such ensemble models consist of multiple decision trees,

which are combined together to form a more complex model. Therefore, the interpretation of such models is not straightforward.

Therefore, we expect that the black–box models will outperform the white–box models on the HMEQ data set, in overall.

Hypothesis #4: *The longer the execution time of a model, the better its performance on HMEQ data set.*

Although, the research paper's objective of Wu (Wu *et al.* 2018) does not directly regard the relationship between the training (execution) time and the model's performance, it rather focuses on the accuracy of indoor localization techniques in two different environments. However, such study also inspects the evaluation of Bayes Net, SVM and Random Forest models, including their execution times. As shown in Table 3.5, we can observe that there is a trade-off between the model's performance and the execution time, as the SVM has the highest accuracy and a moderate execution time, while the Bayes Net has the lowest accuracy, but also the shortest execution time. Therefore, we expect a positive relationship between execution time and the model's performance.

Table 3.5: Accuracy vs. Execution Time (Wu *et al.* 2018)

Model	Acc.	Time (seconds)
Bayes Net	0.8521 / 0.8474	0.56 / 1.15
SVM	0.9328 / 0.9590	1.79 / 3.21
RF	0.9070 / 0.9574	4.92 / 8.84

Source: (Wu *et al.* 2018)

Hypothesis #5: *The main default drivers are the debt and/or delinquency features on HMEQ data set.*

In the study of Zurada (Zurada *et al.* 2014), the author also inspected the significance of features in order to reduce the dimensionality. In particular, the author employed both correlation analysis with the target variable as well as ranking according to information gain. In this case, the most significant features are the debt–to–income ratio and the number of delinquent credit lines. Therefore, these features are expected to be the most important indicators when predicting a default.

Chapter 4

Empirical Analysis - Machine Learning Implementation

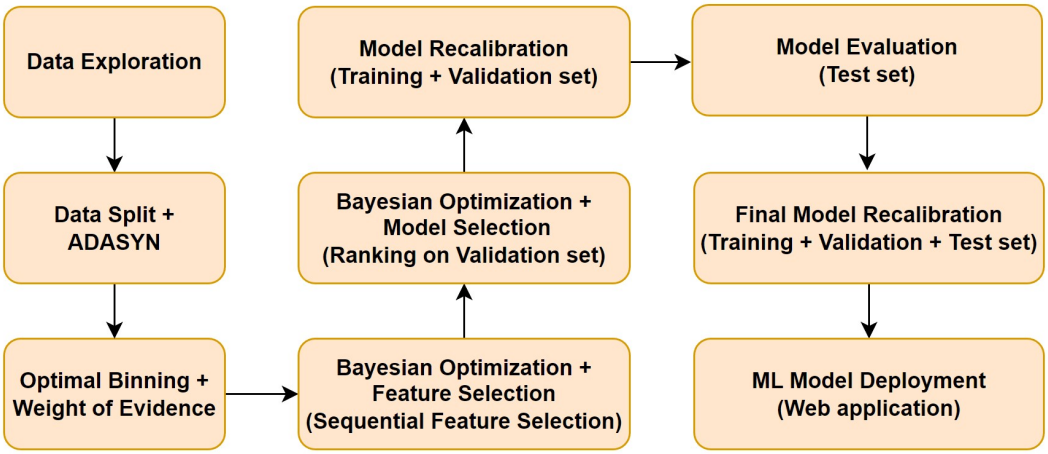
This chapter focuses on the main part of this thesis, particularly on employing the custom machine learning implementation framework. The machine learning framework deployed in this thesis is shown in Figure 4.1 and is described from the high–level–point view as follows:

- **Data Exploration** - Exploration of the data in order to infer some insights about the data quality, distribution of the variables, statistical testing, and association analysis.
- **Data Split and ADASYN** - Splitting of the data that are used separately in different tasks such as model training, hyperparameter tuning, feature selection, model selection, and model evaluation. Further, ADASYN oversampling is performed in order to handle the imbalanced data.
- **Optimal Binning and WoE Encoding** - Optimal binning and the WoE encoding of the features as the main part of the feature preprocessing.
- **Feature Selection** - Feature selection with Forward Sequential Feature Selection in order to reduce the dimensionality of the data and improve the performance of the machine learning models - each input model is tuned with Bayesian Optimization.
- **Model Selection** - Model selection based on weighted ranking where the weights are defined explicitly by the thesis' author, in order to find

the best model. Each input model is tuned with Bayesian Optimization on each subset of selected features.

- **Model Recalibration (Evaluation)** - Recalibration (Re-training) of the final model by re-training it on the joined training and validation sets, which will be further evaluated.
- **Model Evaluation** - Evaluation of the final model on the unseen data from the test set in order to assess the model's predictive power as well as model's explainability.
- **Model Recalibration (Deployment)** - Final recalibration of the final model by re-training it on the joined training, validation, and test sets, which will be further deployed into a production.
- **Model Deployment** - Deployment of the final model into production as a web application.

Figure 4.1: Machine Learning Framework



Source: Author's Framework

4.1 Repository and Environment Structure

The whole machine learning implementation as the scope of this thesis, is developed mainly using Python Programming Language and further with the collaboration of Git and Hypertext Markup Language (HTML). The whole repository can be found in the separate appendix or is available on the GitHub repository which is accessible via the following link: `https://github.com/petr-ngn/FFU_VSE_Masters_Thesis_ML_Credit_Risk_Modelling`. The repository structure is shown in Figure 4.2.

Figure 4.2: Repository Structure

```
|--- data
|  |--- interim_dat.csv
|  |--- preprocessed_data.csv
|  |--- raw_data.csv
|
|--- flask_app
|  |--- inputs
|  |  |--- inputs_flask_app_dict.pkl
|  |
|  |--- templates
|  |  |--- index.html
|  |  |--- results.html
|  |
|  |--- static
|  |
|  |--- app.py
|
|--- models
|  |--- feature_preprocessing
|  |--- feature_selection
|  |--- model_selection
|  |--- objects_FINAL
|
|--- plots
|
|--- Masters_Thesis.ipynb
|--- requirements.yml
```

Source: Author's repository at GitHub

- `data` - Directory containing the raw data, partially preprocessed data (`interim`) and the final preprocessed data.
- `flask_app` - Directory containing the Flask application, which is used for the deployment of the model. Particularly, it contains the `app.py` file, which is the main back-end file of the application, the `templates`

and `static` subdirectories, which contain the front-end HTML files for the application, and the `inputs` subdirectory, which contains the input dictionary for the application (such as the trained model, threshold, final features, etc.).

- `models` - Directory containing the sub–directories of the trained and fitted objects for feature preprocessing, feature selection, and model selection, including the final objects used in evaluation and deployment.
- `plots` - Directory containing the plots generated within the main Python notebook.
- `Masters_Thesis.ipynb` - Main Python notebook containing the main part of the machine learning implementation, such as exploratory analysis, data preprocessing, training, and evaluation of the models. Web application deployment is not included in this notebook, but in `app.py` Python script.
- `requirements.yml` - File containing the list of the required packages and their specific versions used in this project.

This particular solution is developed in Python version 3.10.9 and these are the main packages and modules used in this project:

- `NumPy`, `Pandas` - for data manipulation and analysis;
- `Matplotlib`, `Seaborn` - for data visualization;
- `Scipy` - for statistical analysis;
- `OptBinning` - for optimal binning of features with respect to the target;
- `ImbLearn` - for handling imbalanced data using oversampling;
- `Scikit-learn` - for ML modelling, feature selection, model selection, and model evaluation;
- `Scikit-optimize` - for hyperparameter tuning;
- `Flask` - for deployment of the model as a web application.

To replicate this solution, one may download this repository as a zip file or clone this repository using Git to the local repository. Before running any files or scripts, it is important to set the environment for such project using the file `requirements.yml`. This can be done by running the following command in the Anaconda terminal which will create the new environment with the name `FFU_VSE_Masters_Thesis` and install all the required packages, as shown in Figure 4.3:

Figure 4.3: Environment Setup Command

```
>> conda env create -n FFU_VSE_Masters_Thesis -f requirements.yml
```

Source: Author's command at Anaconda

An user should be aware of his current path directory in the terminal. In order to install the file `requirements.yml`, one needs to define a path to the directory, where such file is located. To achieve this, the user has to either change the path in the terminal using `cd` command, insert the path directory before the `requirements.yml` in terminal, or copy the file `requirements.yml` to the current path directory. The following code in Figure 4.4 shows the former approach:

Figure 4.4: Environment Setup Command with Directory Change

```
>> cd C:\Users\ngnpe\FFU_VSE_Masters_Thesis_ML_Credit_Risk_Modelling
>> conda env create -n FFU_VSE_Masters_Thesis -f requirements.yml
```

Source: Author's command at Anaconda

To preserve the reproducibility of this solution and consistency of the results, the random seed is instantiated to `42`, so for instance, data split, model optimization, or training would be deterministic and not totally random every time when replicating the solution. Some `Scikit-learn` or `Scikit-optimize` objects have the optional argument `n_jobs` which utilizes the number of central processing units (CPU) cores used during the parallelizing computation. Such argument is set to `-1`, hence, all the processors are used in order to speed up the training or optimization process.

4.2 Data Exploration

This section is focused on the exploration of the analyzed loan data set, particularly on data set description, distribution analysis, and association analysis, in order to infer potential valuable insights that can be used in the preprocessing, modelling or evaluation part.

4.2.1 Data Set Description

The analyzed data set pertains to the HMEQ data set, which contains loan application information and default status of 5,960 US home equity loans. Such data set was acquired from Credit Risk Analytics website (Baesens *et al.* 2016). Since this data set regards loan application scoring data, using macroeconomic or other external data is omitted due to the data set's characteristics as well as modelling with behavioral scoring. Thus, our goal is to predict whether the loan applicant would not default based on the information provided in the loan application. Thus, we aim to predict the probability of default only, therefore, EAD, LGD, and ECL are not considered in this thesis.

As can be seen in Table 4.1, the data set contains 13 columns, 12 features, and 1 target variable, `BAD` indicating whether the loan was in default (`1`) or not (`0`). Amongst the 12 features, there are 10 numeric features and 2 categorical features, namely `REASON` which contains 2 categories - Debt consolidation (`DebtCon`) and Home improvement (`HomeImp`), and `JOB` which contains following categories - Administration (`Office`), Sales, Manager (`Mgr`), Professional Executive (`ProfExe`), Self-employed (`Self`), and Other.

Although all the variables and their names seem to be self-explanatory, one may find the `DEROG` feature's meaning unclear. In such case, derogatory report refers to the derogatory mark on a credit report, which is the negative item indicating significant credit risk as an indicator of delinquency issues or late payments (White 2022). Such credit report is provided by the credit bureau based on the bank's request when assessing the creditworthiness of the loan applicant.

Regarding the indirect data quality issues, the data set's source does not specify the time period of its observations or the snapshot date, which may also be relevant point given that some loans might default after the snapshot date even though they had not defaulted before. Nevertheless, as already mentioned

in introduction, the main goal of this thesis is the implementation of a custom machine learning implementation framework, while the data set is used only as a proof of concept. Therefore, we deem it appropriate to omit the raised data quality issues in this thesis.

Table 4.1: Data Set Variables

Variable	**Description**	**Data type**
BAD	Default status	Boolean
LOAN	Requested loan amount	numeric
MORTDUE	Loan amount due on existing mortgage	numeric
VALUE	Value of current underlying collateral property	numeric
REASON	Reason of loan application	categorical
JOB	Job occupancy category	categorical
YOJ	Years of employment at present job	numeric
DEROG	Number of derogatory reports	numeric
DELINQ	Number of delinquent credit lines	numeric
CLAGE	Age of the oldest credit line in months	numeric
NINQ	Number of recent credit inquiries	numeric
CLNO	Number of credit lines	numeric
DEBTINC	Debt-to-income ratio	numeric

Source: (Baesens *et al.* 2016)

After the initial data inspection, the data set does not contain any duplicates but does contain missing values, which are summarized in Table 4.2. Most of the missing values are observed within the feature `DEBTINC` with 1,267 missing observations, whereas variables indicating default status (`BAD`) or requested loan amount (`LOAN`) do not contain any missing values, which is expected as the bank should have the available information about their loans whether they have defaulted or not, and since this data set pertains to the application scoring, when applying for a loan, an applicant should always fill out the requested loan amount.

Table 4.2: Missing Values Summary

Variable	# NA's	% NA's
BAD	0	0.00 %
LOAN	0	0.00 %
MORTDUE	518	8.69 %
VALUE	112	1.88 %
REASON	252	4.23 %
JOB	279	4.68 %
YOJ	515	8.64 %
DEROG	708	11.88 %
DELINQ	580	9.73 %
CLAGE	308	5.17 %
NINQ	510	8.56 %
CLNO	222	3.72 %
DEBTINC	1267	21.26 %

Source: Author's results in Python

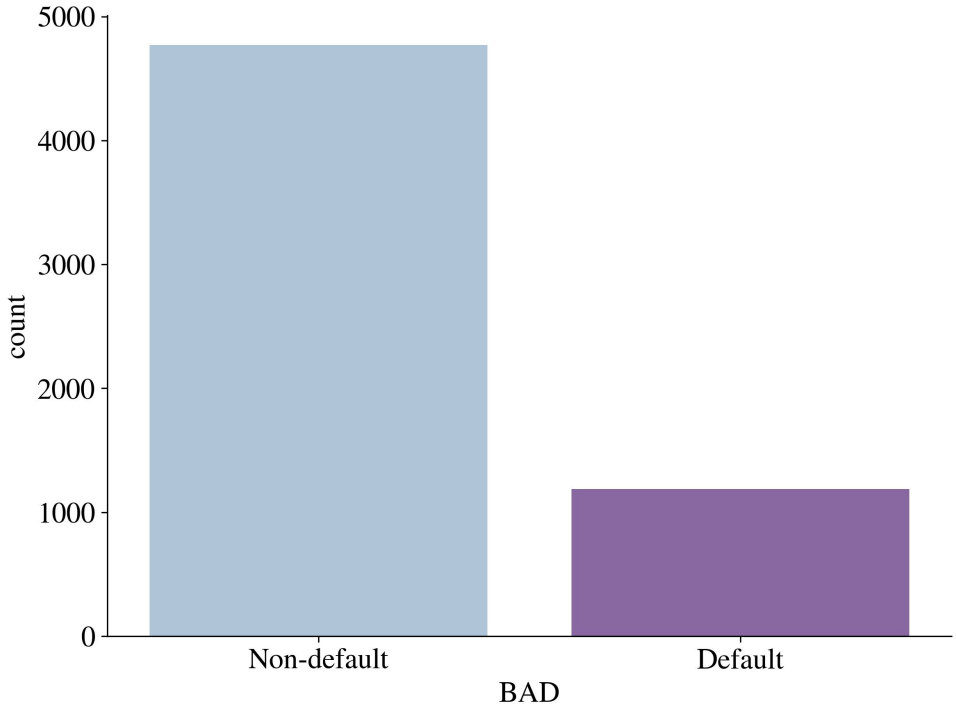
4.2.2 Distribution Analysis

In this subsection, we inspect the distribution of our variables, including the target variable and the features. Such distribution inspection may help us identify outliers, and other potential issues with the data set.

Default Distribution

Regarding the target variable distribution, from the Figure 4.5 we can observe that the default status distribution is heavily imbalanced, as most of the loans have not defaulted yet. Particularly, 80.05% of the observations have been labeled as non-default (4,771 observations) and 19.95% observations labeled as default (1,189 observations). This may cause problems in the modelling part, as the model might be biased towards the majority class, i.e., the non-default class. Such imbalanced class issue will be further treated in Subsection 4.3.1.

Figure 4.5: Default Distribution



Source: Author's results in Python

Numeric Features' Distribution

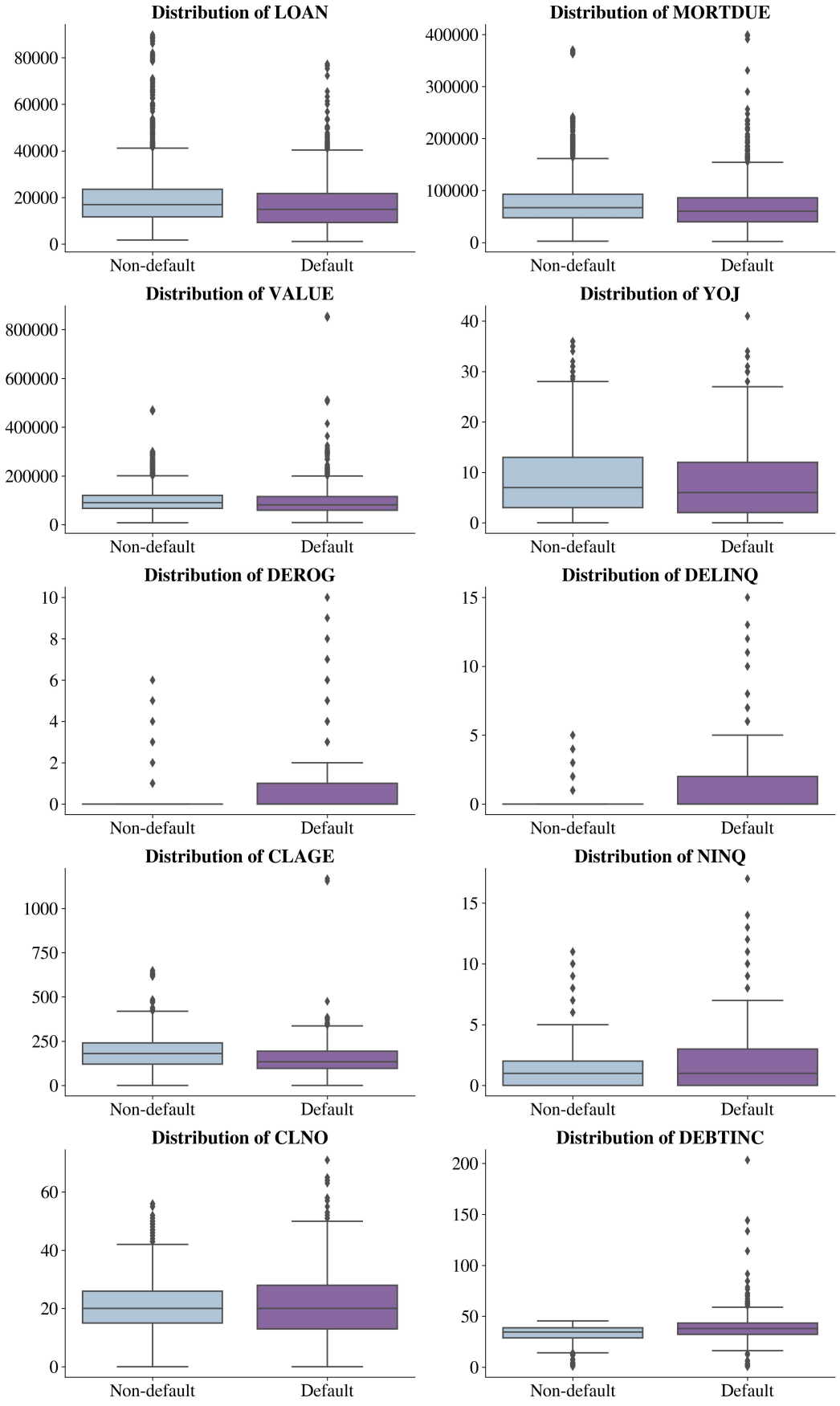
Regarding the numeric features, it can be observed that most of them exhibit a positive skewness and contain outliers, as illustrated in Figure 4.6, which depicts the conditional distribution of the numeric features with respect to the default status via boxplots. All the outliers appear to be valid, indicating that they have not arisen due to data entry errors. This can be attributed to the non-negative nature of all the numeric features, which makes it impossible to have negative values for features such as the number of years at the present job or the number of delinquent credit lines, among others. Additionally, the maximum values of the given features are not unrealistically high, further corroborating the validity of the outliers.

However, it is necessary to treat these outliers as such, as they can bias and jeopardize the model's weights, coefficients, or distance calculation, which may significantly affect the position and orientation of the decision boundary. Such factors can lead to overfitting, inaccurate, and biased predictions. A detailed explanation of the outlier treatment is provided in Subsection 4.3.2.

Concerning the target variable, it can be observed that there are some differences in the distribution shapes of `DEROG` and `DELINQ`, which exhibit less skewness and lower dispersion for non-default cases as compared to default cases. Since both features indicate negative information about delinquency, it is expected that a higher value for these features would increase the likelihood of loan default. Referring to the feature `DEBTINC`, it does not exhibit any extreme values for non-default cases, but some extreme values are present for default cases. From this, it can be inferred that if the debt-to-income ratio is too high, indicating that the applicant's income is not sufficient to cover their debt, the loan is more likely to end in default. The association between the default status and the numeric features is further investigated in Section 4.2.3.

Since the features are substantially positively skewed, they do not follow a Gaussian (normal) distribution. Additionally, Shapiro–Wilk test is conducted in order to assess whether the features' distributions are significantly different from the normal distribution. According to such test, all the numeric features' distributions do not follow the normal distribution on 1% statistical significance level. For a reference, see Table A.1 in Appendix A.

Figure 4.6: Conditional Distribution of Numeric Features



Source: Author's results in Python

Due to the fact that the boxplots do not capture the missing values that occurred in given features, it is also important to inspect the numbers and proportions of missing values in each feature, conditional on the default status. As can be seen in Table 4.3, n_0 refers to the number of missing values in the given feature for non-default cases, and n_1 refers to the number of missing values in the given feature for default cases. N_0 and N_1 refer to the total number of non–default cases and default cases, respectively, therefore, n_0/N_0 refers to the proportion of missing values in the given feature for non-default cases, and n_1/N_1 refers to the proportion of missing values in the given feature for default cases.

Pertaining to the feature `DEBTINC`, we can observe a significant difference in the number of missing values between the default and non-default cases. Out of all defaulted loans, 66.11 % had a missing debt–to–income ratio, whereas only 10.08 % out of all non-defaulted loans had a missing debt–to–income ratio. Therefore, there could be a strong association between the missing debt–to–income ratio and the default.

Similarly, the table depicts a significant difference with respect to `VALUE` as 0.15 % had missing collateral property value out of all non–defaulted loans and 8.92 % defaulted loans had missing collateral property value. It can be inferred that loan applicants who withhold information on their collateral property value or debt-to-income ratio are more likely to default on their loans. This may be due to negative information that they are trying to conceal, such as excessively high debt, low income, or a low collateral property value. Such associations are further investigated in Section 4.2.3.

Table 4.3: Numeric Features NA's Summary

Feature	n_0	n_1	n_0/N_0	n_1/N_1
LOAN	0	0	0 %	0 %
MORTDUE	412	106	8.64 %	8.92 %
VALUE	7	105	0.15 %	8.83 %
YOJ	450	65	9.43 %	5.47 %
DEROG	621	87	13.02 %	7.32 %
DELINQ	508	72	10.65 %	6.06 %
CLAGE	230	78	4.82 %	6.56 %
NINQ	435	75	9.12 %	6.31 %
CLNO	169	53	3.54 %	4.46 %
DEBTINC	481	786	10.08 %	66.11 %

Source: Author's results in Python

Categorical Features' Distribution

Regarding the distribution of categorical features, the data set includes 2 nominal features, namely `REASON` and `JOB`. The conditional distribution of categorical features on the default status is visualized using barplots in Figure 4.7. The plot indicates that most loan applicants applied for debt consolidation, while most job occupancies were labeled as `Other`.

With respect to the default status, there appears to be no strong difference between the default and non-default cases in terms of the relative distribution of the `REASON` feature. However, a slight difference is observed between the default and non-default cases in terms of the relative distribution of the `JOB` feature. Specifically, the categories `Office`, `ProfExe`, and `N/A` exhibit a relatively higher proportion of non-default cases than default cases. Hence, a moderate association between the `JOB` feature and the default status is possible, as further investigated in Section 4.2.3.

Figure 4.7: Conditional Distribution of Categorical Features

Source: Author's results in Python

4.2.3 Association Analysis

In this subsection, we aim to examine potential relationships between the variables by analyzing their associations. Firstly, we investigate the association between the default status and the features. Subsequently, we explore the associations among the features themselves.

Association between Default Status and Numeric Features

To measure the association between the target variable and the numeric features, we use the Point-Biserial correlation coefficient, which is the Pearson's product moment correlation coefficient between a continuous variable and a dichotomous variable (Kornbrot 2015). This coefficient ranges from -1 to +1

and can be used to assess the strength and direction of the relationship between a continuous variable and a binary variable. The formula for computing this coefficient is as follows:

$$r_{pb,X} = \frac{\mu\left(X|Y=1\right) - \mu\left(X|Y=0\right)}{\sigma_X} \sqrt{\frac{N\left(Y=1\right) \times N\left(Y=0\right)}{N\left(N-1\right)}} \tag{4.1}$$

where $\mu\left(X|Y=1\right)$ and $\mu\left(X|Y=0\right)$ represent the means of the given numeric feature X conditional on the default status and non-default status, respectively, while σ_X denotes the standard deviation of X. The values of $N\left(Y=1\right)$ and $N\left(Y=0\right)$ indicate the number of observations with default status and non-default status, respectively, and N represents the total number of observations within the feature X.

The following Table 4.4 displays the computed Point-Biserial coefficient for each numeric feature with respect to the default status, along with its statistical significance. The results show that features such as `DEROG`, `DELINQ`, and `DEBTINC` are moderately and positively associated with the default status at the 1% statistical significance level. These findings support the observations made in Section 4.2.2 regarding the positive associations of these features with the default status. It can be inferred that these features may serve as important predictors in the model.

Table 4.4: Point–Biserial Correlation - Numeric Features vs. Default

Feature	**Coefficient**	**Significance**
LOAN	-0.075	***
MORTDUE	-0.048	***
VALUE	-0.030	**
YOJ	-0.060	***
DEROG	0.276	***
DELINQ	0.354	***
CLAGE	-0.170	***
NINQ	0.175	***
CLNO	-0.004	
DEBTINC	0.200	***

*: $p < 0.10$, **: $p < 0.05$, ***: $p < 0.01$

Source: Author's results in Python

Association between Default Status and Categorical Features

In order to measure the strength of the relationship between the dichotomous default status and categorical variables, we employ Cramer's V, which ranges from 0 to 1 and is defined as:

$$CV_X = \sqrt{\frac{\chi^2}{N\,(k-1)}} \tag{4.2}$$

As noted in Section 4.2.2, the association between the default status and `REASON` is weak, as evidenced by the Cramer's V value being close to zero. Conversely, the association between the default status and `JOB` is slightly stronger, as the categories `Office`, `ProfExe`, and `N/A` exhibit a higher proportion of non-default cases than default cases. Both `REASON`'s and `JOB`'s associations with default status are statistically significant at the 1% significance level.

While statistical significance is important, it does not necessarily indicate that a feature is a strong predictor of the target variable. Ultimately, the usefulness of a feature in predicting the target variable is determined by the performance metrics of the model.

Table 4.5: Cramer's V Association - Categorical Features vs. Default

Feature	**Coefficient**	**Significance**
REASON	0.038	***
JOB	0.135	***

*: $p < 0.10$, **: $p < 0.05$, ***: $p < 0.01$

Source: Author's results in Python

Association between Default Status and Missing Values

Given that the loan data set contains missing values, it is necessary to examine whether the missingness is associated with the default status. One possible approach is to encode the feature with missing values as a binary variable, where 1 indicates the presence of a missing value and 0 otherwise.

To quantify the strength of association between the two binary variables, the Phi coefficient is used, which is defined as:

$$\phi_X = \sqrt{\frac{\chi^2}{n}} \tag{4.3}$$

In line with the finding regarding the `DEBTINC` and `VALUE` in Section 4.2.2, there is a strong and statistically significant association between the missing debt-to-income ratio and default status, and a moderate and statistically significant association between the missing collateral property value and default status, as shown in Table 4.6. Therefore, we can anticipate that these features will be crucial indicators in default prediction. Further details on feature selection are presented in Subsection 4.4.2.

Table 4.6: Phi Correlation Coefficient - NA's vs. Default

Feature	**Coefficient**	**Significance**
LOAN	0.000	
MORTDUE	0.003	
VALUE	0.254	***
REASON	0.004	
JOB	0.064	***
YOJ	0.056	***
DEROG	0.070	***
DELINQ	0.061	***
CLAGE	0.030	**
NINQ	0.039	***
CLNO	0.018	
DEBTINC	0.547	***

*: $p < 0.10$, **: $p < 0.05$, ***: $p < 0.01$

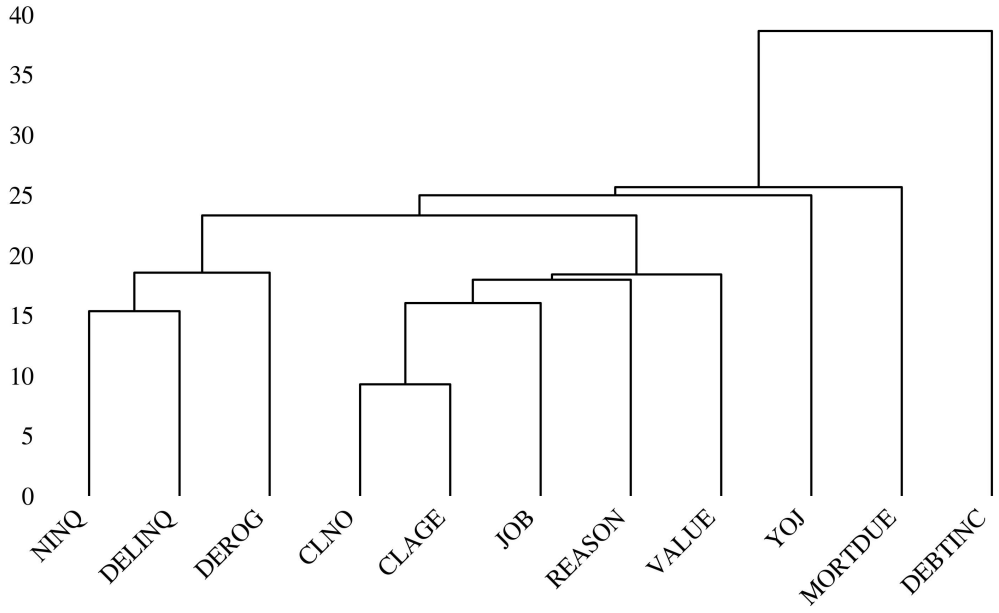
Source: Author's results in Python

Missing Values Association

Additionally, it is imperative to investigate the relationship between missing values and default status, as well as the interrelationship between the missing values themselves. A common approach to identifying patterns of missing data in a data set is through the use of a dendrogram, which clusters variables hierarchically based on the occurrence of missing values. This method groups variables into clusters based on the similarity of their missing value patterns, such that variables with comparable patterns of missingness are clustered together. The dendrogram is constructed by merging the two closest clusters iteratively until all variables are in the same cluster. The distance between the clusters at each step of the merging is shown on the y-axis of the dendrogram, and the order in which the variables are merged is displayed on the x-axis.

In Figure 4.8, the hierarchical clustering of the data set's variables is illustrated, excluding the default status and requested loan amount feature `LOAN`, as these variables do not contain any missing values. As depicted in the dendrogram, the `CLNO` and `CLAGE` features have the most similar patterns of missing value occurrences. Therefore, it can be inferred that a significant number of loan applicants tend to omit information regarding their number of credit lines (`CLNO`) and the age of their most recent credit line (`CLAGE`) when submitting their loan applications.

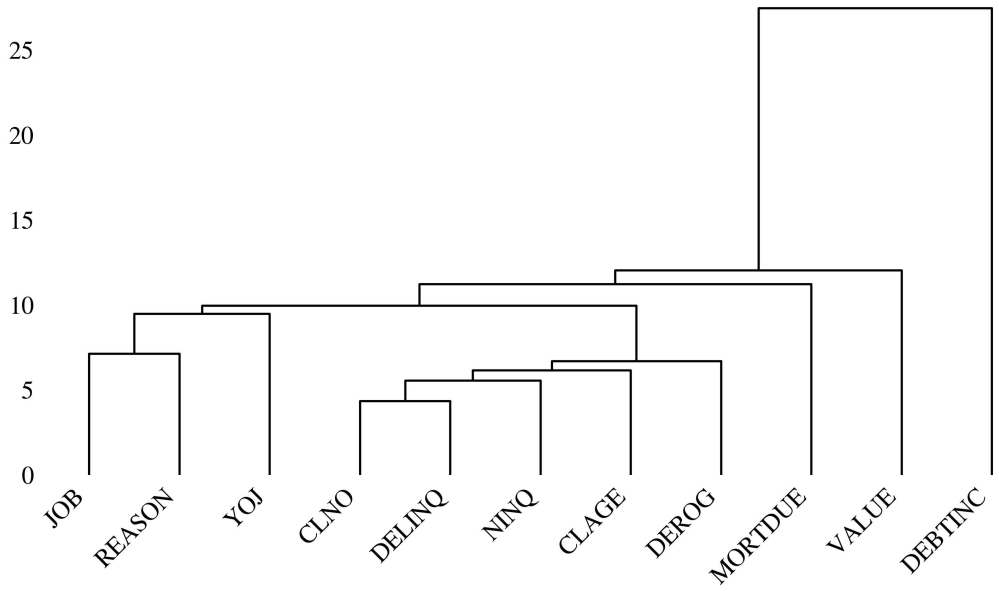
Figure 4.8: Nullity Dendrogram



Source: Author's results in Python

Note that the target variable is imbalanced, therefore, the dendrogram depicted in Figure 4.8 is not representative for the default cases. Therefore, we can also inspect the dendrogram for the default cases only, as shown in Figure 4.9. As can be seen, the hierarchy has changed compared the hierarchy in Figure 4.8. It can be observed that a significant number of loan applicants tend to omit information regarding their number of credit lines (`CLNO`) and the number of delinquent credit lines (`DELINQ`) when submitting their loan applications. Especially regarding the second feature, this might indicate a common delinquency problem among the applicants and, thus, a higher probability of default.

Figure 4.9: Nullity Dendrogram (default cases)



Source: Author's results in Python

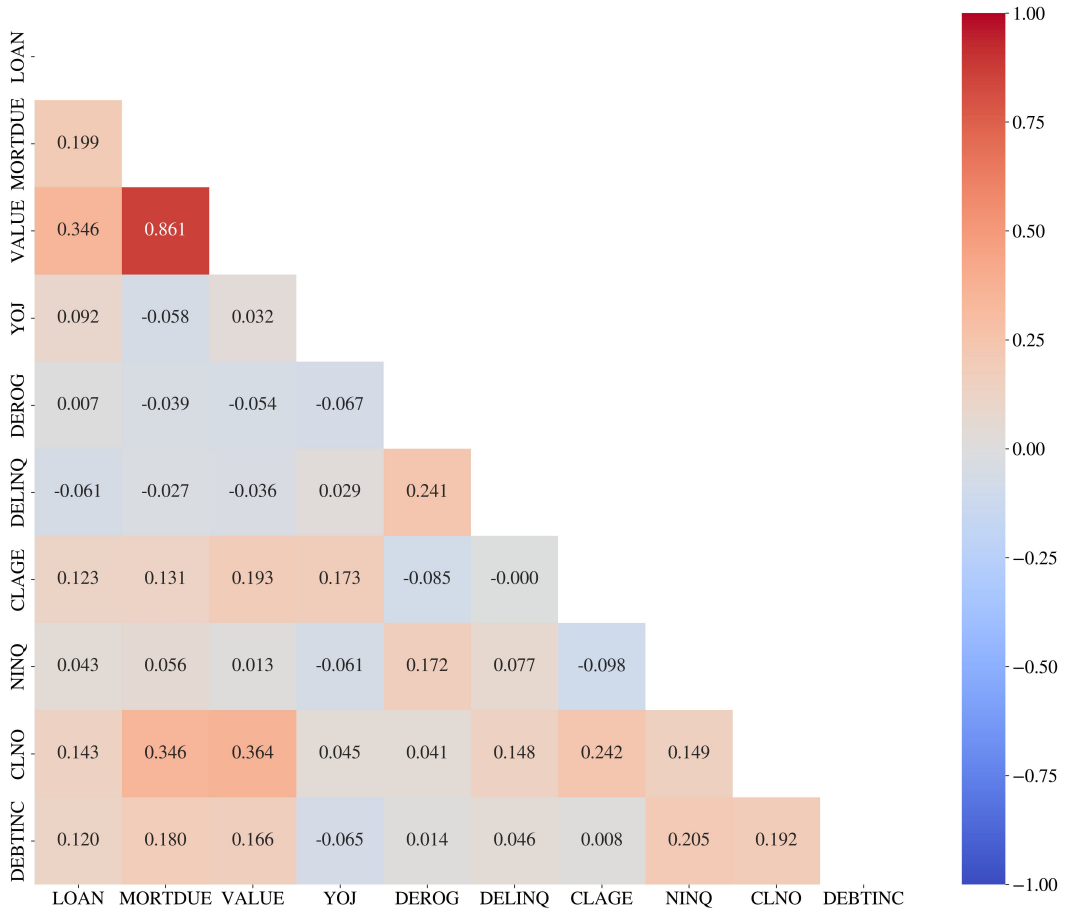
Multicolinearity Analysis

To quantify the association between the numerical features, the Spearman correlation coefficient is utilized as a non-parametric measure that does not make any assumptions regarding the distribution of variables or the linearity of their relationship. The Spearman correlation coefficient is defined as follows:

$$\rho_{spearman} = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n\left(n^2 - 1\right)} \tag{4.4}$$

In the Figure 4.10, we can observe a very strong correlation between the `MORTDUE` and `VALUE` features. Such multicolinearity can cause problems in predictions and model's overfitting. Therefore, a feature selection is recommended - such selection is further described in Subsection 4.4.2.

Figure 4.10: Spearman Correlation Matrix



Source: Author's results in Python

Regarding the association between categorical features, namely `REASON` and `JOB`, the Cramer's V coefficient is used. The Cramer's V coefficient results in a value of 0.285 while being statistically significant at 1% the significance level, thus, we observe a moderate association between the two features, which will be further captured within feature selection. If the association amongst the features themselves is not plausible in terms of model's performance, the feature selection ensures that these features will be excluded.

4.3 Data Preprocessing

In this section, the process of preprocessing data is described as a crucial step prior to machine learning modelling. Particularly, the process of dividing data for various tasks, ADASYN oversampling due to the imbalanced class issue, discretization of features and Weight–of–Eveidence transformation are further discussed and employed.

4.3.1 Data Split and ADASYN Oversampling

To ensure appropriate model training and unbiased evaluation, it is necessary to split the data into separate sets for various purposes. Specifically, the data set is split into three sets: (1) training set for training the model, feature selection or hyperparameter tuning, (2) validation set for model selection, and (3) test set to assess the model's performance on the unseen data (Subasi 2020). In this thesis, the data is split into training, validation, and test sets with a ratio of 70:15:15, which ensures a sufficient amount of data for training, hence the model would be able to generalize well, while keeping the validation and test sets large enough to provide reliable evaluation of the model's performance.

The data is split using stratified splitting to preserve the default status distribution, which is highly imbalanced. Stratification ensures that the distribution of defaults and non-defaults remains the same across all sets, thereby avoiding overfitting and data leakage. Using stratification, each set had 80 % non-defaults and 20 % defaults. This method enables accurate prediction since the model is trained and evaluated on the same population (Igareta 2021).

However, stratification alone may not be sufficient for dealing with imbalanced classes. Therefore, ADASYN oversampling is performed as described in Section 2.5. Note that ADASYN oversampling is performed on the training set only after the split to avoid data leakage and biased evaluation.

In Python, the data are divided and oversampled using a custom function `data_split()`. This function first employs the `train_test_split()` function from the `scikit-learn` module to split the data into training, validation, and test sets with a stratification technique. Next, the `ADASYN()` class from the `imblearn` module is used to oversample the training set. This is achieved by generating synthetic instances of the minority class based on the five nearest neighbors and Euclidean distance.

However, the `ADASYN()` class from `imblearn` is not designed to handle missing values or categorical features encoded as character. To overcome this limitation, the following approach is taken:

1. Separate the categorical and numeric features;
2. Impute the missing values with arbitrary values:
   - Categorical features: string `'N/A'`;
   - Numeric features: number `99999999999999` - such value is chosen since it is highly unlikely to be present in the data set.
3. Convert the categorical features into dummy variables;
4. Join the numeric features with the dummy variables;
5. Perform the oversampling on the joined data set;
6. Convert the dummy variables back into categorical features;
7. Retrieve back the missing values:
   - Categorical features: replace string `'N/A'` with `np.nan`;
   - Numeric features: for each feature X if its value exceeds its original maximum value, then replace it with `np.nan`;

The following Table 4.7 shows the default distribution of the individual sets before and after oversampling. The default distribution in the training set after ADASYN oversampling is balanced, while the default distribution remains the same across the validation and test sets, which is desirable due to stratification.

Table 4.7: Data Split Summary

Set	**# instances**	**% defaults**	**% non–defaults**
Training	4,171	19.95 %	80.05 %
Training (oversampled)	6,437	48.13 %	51.87 %
Validation	895	20.00 %	80.00 %
Test	894	20.00 %	80.00 %

Source: Author's results in Python

Since ADASYN increases the sample size by generating, i.e., adding new instances to the sample, it may have an impact on the distribution of the features. After the inspection in Python, it has no significant impact on the numeric features as the distributions before and after ADASYN oversampling do not differ that much, as can be seen Figure A.1 in Appendix A. However, pertaining to the categorical features, a significant impact of ADASYN oversampling can be observed, as can be seen in Table 4.8 and Figure 4.11, respectively, which depict the distribution of categorical features on the subsample of default cases within the training set, particularly distribution before and after the oversampling. As such, n_B and n_A refer to the number of default cases within a given category before and after oversampling, respectively, and N_B and N_A represent the total number of default cases before and after oversampling, respectively.

Within the `REASON` feature, we can observe a significant increase in the proportion of defaulted clients who applied for a loan due to the debt consolidation (`DebtCon`) among all the defaulters after an oversampling - particularly, the ratio has increased by 12.20 percentage points. Regarding the `JOB` feature, we can observe a substantial increase in the proportion of defaulted managers (`Mgr`) among all the defaulters after oversampling by 38.75 percentage points.

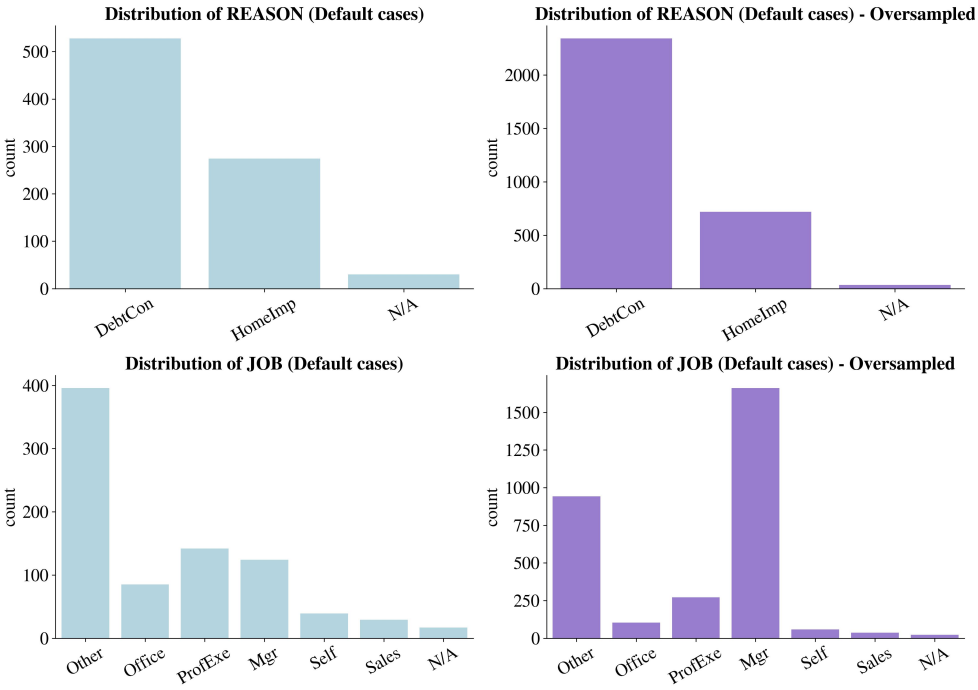
Table 4.8: ADASYN Impact on Categorical Features' Distribution

Features	**Category**	n_B	n_A	n_B/N_B	n_A/N_A	**Diff.**
REASON	DebtCon	528	2,344	63.46 %	75.66 %	12.20 p.p.
REASON	HomeImp	274	720	32.93 %	23.24 %	-9.69 p.p.
REASON	N/A	30	34	3.61 %	1.10 %	-2.51 p.p.
JOB	Mgr	124	1,662	14.90 %	53.65 %	38.75 p.p.
JOB	Other	396	943	47.60 %	30.44 %	-17.16 p.p.
JOB	ProfExe	142	272	17.07 %	8.78 %	-8.29 p.p.
JOB	Office	85	104	10.22 %	3.36 %	-6.86 p.p.
JOB	Self	39	58	4.69 %	1.87 %	-2.82 p.p.
JOB	Sales	29	36	3.49 %	1.16 %	-2.33 p.p.
JOB	N/A	17	23	2.04 %	0.74 %	-1.30 p.p.

Source: Author's results in Python

Such findings regarding the significant increases in the proportion of default cases regarding the `DebtCon` and `Mgr` categories are given to the nature of ADASYN oversampling. As already explained, ADASYN generates more synthetic default class instances in the neighborhood of such default instances, which are hard–to–learn for ADASYN. In other words, ADASYN creates more default instances for such default instances where the density of the non–default class is relatively high in given default class instance's neighborhood. Therefore, default instances which either are managers or applied for a loan due to the debt consolidation are difficult–to–learn for ADASYN, thus ADASYN replicates more instances with such characteristics, hence this results in a higher proportion of default cases in the oversampled set for such category.

Figure 4.11: ADASYN Impact of Categorical Features' Distribution



Source: Author's results in Python

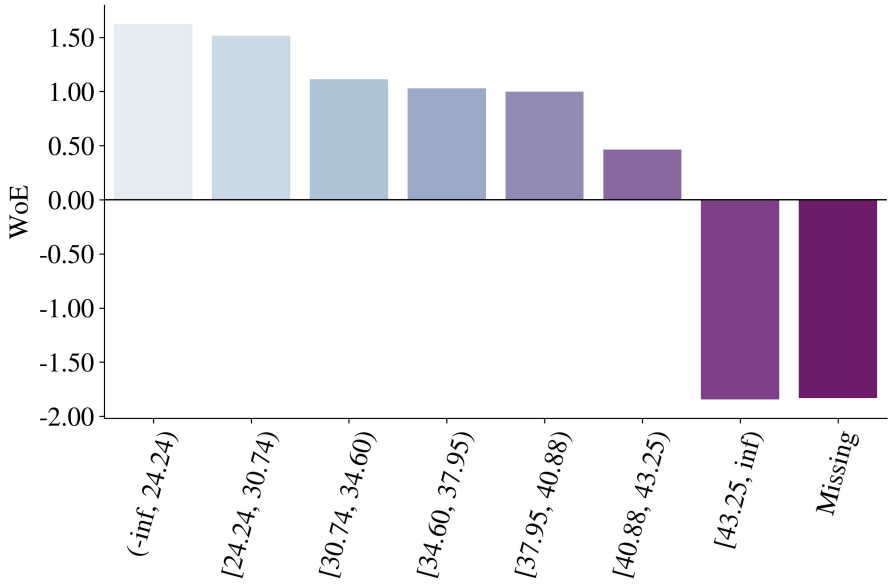
4.3.2 Optimal Binning and Weight–of–Evidence

As already described in Section 2.6, the Optimal Binning is employed as a data preprocessing step in this thesis. Particularly, we employ `BinningProcess` class from the `optbinning` module in Python - such class optimally discretizes the continuous features into interval bins and optimally groups categorical features' categories into sub-group categories with respect to the target variable while maximizing Information Value, and afterwards such bins are encoded using WoE (Navas-Palencia 2020). Besides discretizing and grouping, it also creates special bins for capturing missing values.

In this thesis, this data preprocessing step is wrapped into a custom function `woe_binning()` in Python. First, the `BinningProcess` class is fitted on the training set only in order to avoid data leakage, while it learns the split points for binning, event rates, WoE coefficients, etc., and then is used to transform the training, validation, and test set.

The following Figure 4.12 depicts the WoE bins distribution for the `DEBTINC` feature. It is evident that the bin capturing the missing values has the most negative WoE coefficient, which indicates a larger distribution of defaulters compared to non–defautlters in such bin. We can further observe a monotonic relationship between the WoE coefficients and the bins, as the debt–to–income ratio increases, the WoE coefficient decreases, thus higher likelihood of default. Therefore, we expect that if the debt–to–income ratio is either missing or extremely high, it will have a significant impact on the likelihood of default.

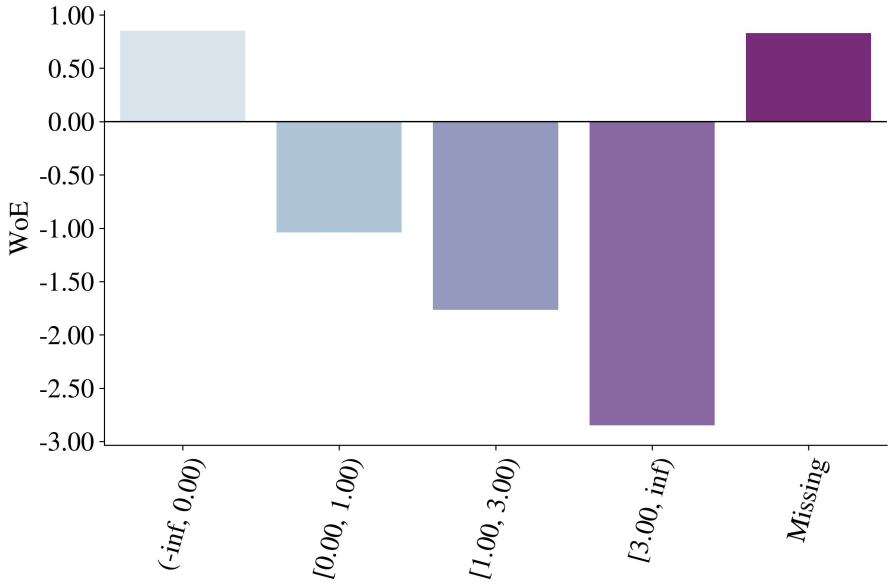
Figure 4.12: WoE Bins Distribution - Debt–To–Income Ratio



Source: Author's results in Python

Another monotonic relationship can be observed in case of `DELINQ` feature as can be seen in Figure 4.13. With increasing number of delinquent credit lines, the WoE coefficient decreases, indicating a larger distribution of defaulters compared to non–defaulters. Therefore, we expect that the higher the number of delinquent credit lines will lead to the higher the likelihood of default.

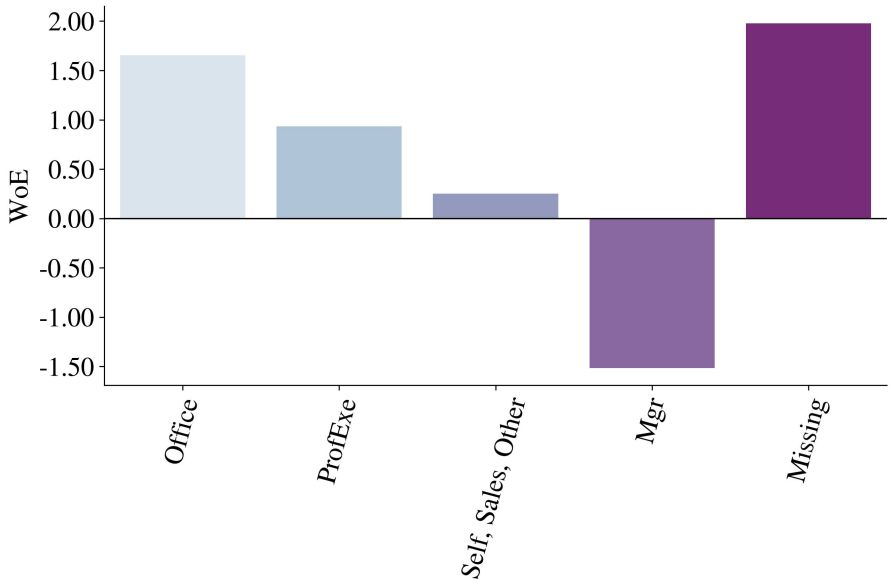
Figure 4.13: WoE Bins Distribution - Number of Delinquent Credit Lines



Source: Author's results in Python

Within the `JOB` feature, specifically regarding the `Mgr` category, we can observe in Figure 4.14 the direct impact of ADASYN oversampling as already described by Table 4.8. Since ADASYN generated more default class instances for original default instances who are managers, this results in a higher default rate within such job category, leading to a larger distribution of defaulters compared to non–defaulters, which is quantified in the negative WoE value. We can also see, that Optimal Binning has grouped categories `Self`, `Sales` and `Others` into a single category bin.

Figure 4.14: WoE Bins Distribution - Job Occupancy



Source: Author's results in Python

The WoE bins distributions of other features are depicted Figure A.2 in the Appendix Appendix A.

4.4 Modelling

Once the data are finally preprocessed, the next step regards the modelling part which includes hyperparameter tuning, feature selection, model selection, and model recalibration. In Python, eight different machine learning classification models from `Scikit-learn` module are used for the default status prediction, which were already described in Section 2.3, namely:

- **Logistic Regression** - `LogisticRegression()`
- **Decision Tree** - `DecisionTreeClassifier()`
- **Gaussian Naive Bayes** - `GaussianNB()`
- **K-Nearest Neighbors** - `KNeighborsClassifier()`
- **Random Forest** - `RandomForestClassifier()`
- **Gradient Boosting** - `GradientBoostingClassifier()`
- **Support Vector Machine** - `SVC()`
- **Multi–Layer Perceptron** (*Neural Network*) - `MLPClassifier()`
  - Henceforth, terms such as Neural Network and Multi–Layer Perceptron are used interchangeably.

4.4.1 Bayesian Hyperparameter Optimization

In this thesis, hyperparameter tuning of models is performed using Bayesian Optimization as described in Section 2.7. In Python, a custom function, `bayesian_optimization()`, is implemented to perform hyperparameter tuning using Bayesian Optimization. This function utilizes `BayesSearchCV` class from the `Scikit-optimize` module, with a 10-fold stratified cross-validation scheme and 50 iterations, while maximizing the F1 score. As a surrogate function, the Gaussian Process is used (scikit-optimize 2023a). The use of `BayesSearchCV` with stratified cross-validation in the hyperparameter tuning process provides a robust and reliable approach to selecting the optimal hyperparameters for the model, while the incorporation of Bayesian Optimization enables the efficient exploration of the hyperparameter space. By maximizing the F1 score, the hyperparameters selected through this process will result in a model with improved performance. Note, that the hyperparameter tuning is performed

solely on training set in order to preserve the independence of the validation and test set, and avoid information and data leakage as well.

For each model, the Bayesian Optimization algorithm performs 50 iterations while searching for the best hyperparameters values that maximize the F1 score. Within each iteration, a 10-fold stratified cross-validation is conducted to evaluate the model's cross–validation F1 score. Moreover, for each model, we specify the hyperparameter space, i.e., the particular hyperparameters to be tuned and their possible ranges that the hyperparameter can take. The ranges are specified using `Integer` class to define an interval of integers, `Real` to define an interval of float numbers, and `Categorical` to define a list of possible (categorical) values. Note that not all the available hyperparameters are tuned, but only the most relevant ones due to the time complexity of the Bayesian Optimization algorithm. The definition of hyperparameter space for each model is described further in the following subsubsections.

Logistic Regression

The first hyperparameter **`Intercept`** allows us to specify whether the intercept should be estimated or not. C is the regularization strength, and it is used to specify the regularization factor, which is further specified with the hyperparameter **`Penalty`**. If the penalty is set to $ElasticNet$, we can also tune $L1$ ratio, which refers to the α parameter in in Equation 2.8 Moreover, the **`Class weight`** hyperparameter is a hyperparameter that allows us to assign higher importance to minority class instances during the training process.

`Solver` hyperparameter is used to specify which optimization algorithm should be used to estimate the parameters. According to Hale (Hale 2019), Scikit-learn provides five solvers for Logistic Regression, namely *lbfgs* (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) which approximates the second derivative matrix updates with gradient evaluations, *liblinear* (Library for Large Linear Classification) which uses the coordinate descent algorithm, *newton-cg*, *sag* (Stochastic Average Gradient descent) as a variation of gradient descent and incremental aggregated gradient approaches using a random sample of previous gradient values, and *saga* as an extension of *sag* which allows $L1$ regularization. For more information, please refer to the Scikit-learn documentation (scikit-optimize 2023b). The last hyperparameter of Logistic Regression to tune in this thesis is the **`Intercept scaling`** which allows to scale the intercept.

Table 4.9: Logistic Regression - Hyperparameter Space

Hyperparameter	**Space**
Intercept	True, False
C factor	$<1 \times 10^{-6}$, 5>
Penalty	L1, L2, Elastic Net, None
Solver	lbfgs, liblinear, newton-cg, sag, saga
Class weight	None, balanced
L1 ratio	<0, 1>
Intercept scaling	True, False

Source: Author's results in Python

Decision Tree

The first hyperparameter of Decision Tree is the **`Criterion`** which allows us to specify the function to measure the quality of a split - either Gini or Entropy impurity function. Another important hyperparameter is the **`Max depth`** which allows us to specify the maximum depth of the tree. The last hyperparameter is the **`Max features`** which allows to specify the number of features to consider when looking for the best split (scikit-learn 2023a). Note, that **`Max features`** has a variable range of values depending on the number of features, which is dependent on the number of features selected during feature selection (where the features are iteratively selected and the model is trained and evaluated on each iteration).

Table 4.10: Decision Tree - Hyperparameter Space

Hyperparameter	**Space**
Criterion	Gini, Entropy
Max depth	<1, 10>
Max features	<1, `len(X.columns)`>

Source: Author's results in Python

Gaussian Naive Bayes

In the case of Gaussian Naive Bayes, the only hyperparameter to tune is the **`Variance smoothing`** which allows to specify the portion of the largest variance of all features to be added to variances for calculation stability (scikit-learn 2023b), in order to smooth out the variances of each feature in case when the variance is zero.

Table 4.11: Gaussian Naive Bayes - Hyperparameter Space

Hyperparameter	**Space**
Variance smoothing	$<1 \times 10^{-9}, 1 \times 10^{-6}>$

Source: Author's results in Python

K–Nearest Neighbors

In KNN, it is crucial to specify the number of neighbors to consider during the classification process, which is depicted in the **`# neighbors`** hyperparameter. Further, it is needed to select the optimal distance measure, i.e., Euclidean, Manhattan, or Minkowski, as described in Subsection 2.3.4 - such selection refers to the **`Metric`** hyperparameter. Pertaining to the Minowski distance, we also tune the **`Norm order`** hyperparameter which allows us to specify the power parameter within the distance calculation. Last but not least, KNN also allows us to tune the **`Weights`** hyperparameter which allows us to specify the weight function used in prediction - either uniform or distance. With the former function, all the points within each neighborhood are weighted uniformly, and with the latter approach, the points are weighted inversely with respect to their distance. (scikit-learn 2023d).

Table 4.12: K–Nearest Neighbors - Hyperparameter Space

Hyperparameter	**Space**
# neighbors	<5, 20>
Metric	Euclidean, Manhattan, Minkowski
Norm order	<1, 5>
Weights	Uniform, Distance

Source: Author's results in Python

Random Forest

In the Random Forest, we tune the number of base trees that are trained in parallel way in the ensemble, i.e., the **`# estimators`** hyperparameter. Similarly to Decision Tree, Random Forest allows to select the optimal split measure, i.e., Gini, Entropy or additionally even Log Loss, which is depicted in the **`Criterion`** hyperparameter. Likewise Decision Tree, Random Forest also allows us to specify the maximum depth of the tree as well as the number of features to consider when looking for the best split, i.e., the **`Max depth`** hyperparameter and **`Max features`**, respectively. The **`Bootstrap`** hyperparameter allows us to specify whether bootstrap samples are used when training the tree estimators.

Similarly to Logistic Regression, Random Forest also has the **`Class weight`** hyperparameter which allows to specify the weight of each class in the classification process - *balanced* function takes the target variable to adjust the weights, which are inversely proportional to class frequencies, whereas *subsample balanced* does almost the same, but the weights are computed based on the bootstrap samples. Last but not least, Random Forest also allows to tune the **`CCP alpha`** hyperparameter which allows to specify the complexity parameter used for Minimal Cost-Complexity Pruning (CCP), in order to reduce the size of the tree estimators and avoid overfitting by removing subtrees from the tree estimators that do not improve the performance (scikit-learn 2023f).

Table 4.13: Random Forest - Hyperparameter Space

Hyperparameter	**Space**
# estimators	<100, 1000>
Criterion	Gini, Entropy, Log Loss
Max depth	<1, 10>
Max features	<1, `len(X.columns)`>
Bootstrap	True, False
Class weight	None, balanced, subsample balanced
CCP alpha	$<1 \times 10^{-12}$, 0.5>

Source: Author's results in Python

Gradient Boosting

Likewise Random Forest, Gradient Boosting also allows us to tune the number of base trees that are trained in a sequential way in the ensemble, i.e., the **`# estimators`** hyperparameter, as well as the maximum depth of the tree and the number of features to consider when looking for the best split, i.e., the **`Max depth`** hyperparameter and **`Max features`**, respectively. Is it also necessary to tune the **`Learning rate`** hyperparameter which shrinks the contribution of each tree, in order to prevent overfitting. We can also select the optimal loss function of Gradient Boosting (**`Loss`** hyperparameter), either Log Loss or Exponential loss function - in the latter case, the model is equivalent to the AdaBoost algorithm (scikit-learn 2023c). Moreover, the **`Criterion`** hyperparameter allows to specify the measurement method of the split quality, i.e., MSE or Friedman MSE, which improves the MSE with Friedman scores (scikit-learn 2023c).

Table 4.14: Gradient Boosting - Hyperparameter Space

Hyperparameter	**Space**
# estimators	<100, 1000>
Max depth	<1, 10>
Max features	<1, `len(X.columns)`>
Learning rate	<0.0001, 0.2>
Loss	Log Loss, Exponential
Criterion	MSE, Friedman MSE

Source: Author's results in Python

Support Vector Machine

Likewise Logistic Regression, SVM also allows us to tune the **`C`** hyperparameter for the regularization, as well as the **`Kernel`** hyperparameter which specifies the kernel type to be used in the algorithm in to map the data into higher dimensional space in order to find the optimal hyperplane that separates the classes. If the kernel is d-degree polynomial, we can also tune the **`Degree`** hyperparameter, which specifies the degree of the polynomial kernel function. SVM also has the **`Class weight`** hyperparameter, which assigns the weights to the input data based on the class frequencies in order to balance the classes.

Table 4.15: Support Vector Machine - Hyperparameter Space

Hyperparameter	**Space**
C factor	$<1 \times 10^{-6}$, 5>
Kernel	Linear, Poly, RBF, Sigmoid
Degree	<1, 10>
Class weight	balanced, None

Source: Author's results in Python

Multi–Layer Perceptron

Regarding the MLP, we use one hidden layer where we tune the number of units as represented by the **`Hidden layer size`** hyperparameter. We also tune the **`Activation function`** applied in the hidden layer, namely Logistic, ReLU, and Tanh, which were already described in Subsection 2.3.8, and further Identity, which is just a linear activation function, hence $f(z) = z$. Regarding the optimization algorithm, which refers to **`Solver`** hyperparameter, we can choose between *lbgfs*, *sgd* (Stochastic Gradient Descent), and *adam* (Adaptive Moment Estimation). The **`Learning rate`** can be either constant (i.e., 0.001), inversely scaled, i.e., gradually and inversely decreased with an inverse scaling exponent t (where t refers to the time step, by default $t = 0.5$), or adaptive, i.e., the learning rate is kept constant as long as the training loss keeps decreasing, otherwise it is divided by 5 (scikit-learn 2023e).

Table 4.16: Multi Layer Perceptron - Hyperparameter Space

Hyperparameter	**Space**
Hidden layer size	<5, 500>
Activation function	Identity, Logistic, Tanh, ReLU
Solver	Adam, sgd, lbfgs
Learning rate	Constant, Adaptive, Invscaling

Source: Author's results in Python

4.4.2 Sequential Feature Selection

As the feature selection approach, Forward Sequential Feature Selection (henceforth SFS) is employed in order to choose the optimal set of features as described in Section 2.8. Within machine learning implementation, instead of fitting Forward SFS only with one model, Forward SFS is fitted to each input model in order to obtain the best subset of features for each model, assuming the importance of each features varies across the models. Instead of using input models with default hyperparameters, each model is tuned with Bayesian Optimization in order to obtain the optimal hyperparameters for each model, which would further improve the performance of each model within SFS and therefore, it would lead to the more optimal selection of features. The custom feature selection algorithm is stated in Algorithm 1, thus, when having n input models, it returns n subsets of optimal features, one per each model:

Algorithm 1 Feature Selection Algorithm

1: **for** *model* $\in$ *models* **do**
2: *optimized_model* $\leftarrow$ BayesianOptimization(*model*)
3: *best_features* $\leftarrow$ ForwardSFS(*optimized_model*)
4: **end for**

Particularly, each input model is first tuned on the training set with Bayesian Optimization with 50 iterations and 10–fold stratified cross validation (in order to preserve the target variable distribution across the folds) while maximizing the F1 score - the author's machine learning implementation, his custom function `bayesian_optimization()` is used. Once the model is tuned, the Forward SFS is fitted with such tuned model on the same training set with 10–fold stratified cross validation while maximizing the F1 score. Instead of selecting the fixed number of features, Scikit-learn's `SequentialFeatureSelector` class allows to set a stop criterion (`tol` parameter) which stops adding features if the objective score function is not increasing at least by `tol` between two consecutive feature additions (scikit-learn 2023g). Such parameter is set to a value that is close to zero, therefore, the feature selection stops when the objective score function is not increasing anymore.

Such feature selection algorithm is wrapped into author's custom function `SFS_feature_selection()`. This function iteratively prints the process of the feature selection as can be seen in Figure 4.15, including the current step (Bayesian Optimization or Feature Selection), the execution time in minutes, and the selected features per each model. Since we have 8 input models, we get 8 optimal subsets of features.

Figure 4.15: Feature Selection Print Statement

```
--------------------------------------------------------------------------------
---------------------------------------- 2/8 -----------------------------------
--------------------------------------------------------------------------------
------------------------------ FEATURE SELECTION WITH DT -------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

1/4 ... Starting Bayesian Optimization on the whole set of features
2/4 ... Bayesian Optimization finished
3/4 ... Starting Forward Sequential Feature Selection
4/4 ... Forward Sequential Feature Selection with finished

Execution time: 0.908 minutes

9 features selected: VALUE, JOB, YOJ, DEROG, DELINQ, CLAGE, NINQ, CLNO, DEBTINC

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Source: Author's results in Python

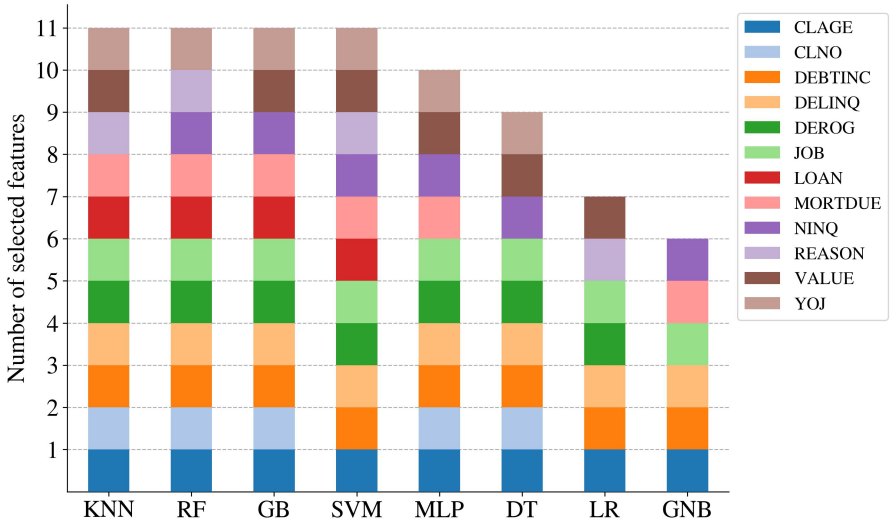
The following Figure 4.16 depicts the recurrence of the selected features. As can be seen, features such as `CLAGE`, `DEBTINC`, `DELINQ` and `JOB` were selected by each model. On the other hand, features such as `LOAN` and `REASON` were selected only four times. Therefore, it can be expected that such features, which were selected every time, will have a significant impact on predictions.

Figure 4.16: Reccurrence of Selected Features

Source: Author's results in Python

According to Figure 4.17, models such as KNN, Random Forest, Gradient Boosting and SVM chose almost all the features, as only one feature was eliminated. On the other hand, Gaussian Naive Bayes chose only 6. It seems that most of the features are important, as each model has selected a higher number of features. It is evident that more complex black–box models require more features in contrast to transparent models such as Logistic Regression or Gaussian Naive Bayes.

Figure 4.17: Distribution of Selected Features per Model



Source: Author's results in Python

4.4.3 Model Selection

In combination with the pre–selected subsets of features, the next step regards the selection of the final model, which will be further used within an evaluation and deployment. The algorithm process is described in Algorithm 2 below:

Algorithm 2 Model Selection Algorithm

```
for model ∈ models do
    for features ∈ features_subsets do
        optimized_model ← BayesianOptimization(model, features)
        for metric ∈ evaluation_metrics do
            performance ← Evaluation(optimized_model, metric)
        end for
    end for
end for
```

Hence, each input model is tuned on each subset of features selected within feature selection on the training set, and subsequently, the optimized model is evaluated on the validation set. Thus, when having n input models and m subsets of selected features, we get $n \times m$ tuned models. $m \leq n$ because we exclude duplicated subsets of selected features, which can occur when more than one model chooses the same subset(s) of features. Since there are 8 input models and 8 unique subsets of selected features, the total number of tuned models is 64.

Metrics Ranking Space

When evaluating classification models using class-based metrics such as F1 score, Precision, Recall, Accuracy, and Matthews Correlation Coefficient, a standard classification threshold of 0.5 is often used. This threshold separates predicted classes based on whether the predicted probability score is higher or lower than 0.5. However, in real-world use cases, the 0.5 classification threshold may not be appropriate, and according to (Esposito *et al.* 2021), such threshold is not appropriate when having imbalanced data.

Therefore, it is recommended to calculate an optimal threshold rather than relying on the standard one. In such case, the Youden index is employed, which is derived from the ROC curve and enables the selection of an optimal classification threshold value. The Youden index searches for such threshold that

maximizes the sum of True Positive Rate and True Negative Rate, decreased by 1 (Fluss *et al.* 2005), thus:

$$J = TPR + TNR - 1 \tag{4.5}$$

Mathematically, the optimal threshold using the Youden index is derived as follows:

$$T_{opt} = \text{argmax}_{t\in[0,1]}\,(J) \tag{4.6}$$

In Python, the `roc_curve` function from `Scikit-learn` returns False Positive Rate instead of the True Negative Rate. Nevertheless, we can derive the True Negative Rate from False Positive Rate as follows:

$$TNR = 1 - FPR \tag{4.7}$$

Therefore:

$$T_{opt} = \text{argmax}_{t\in[0,1]}\,(TPR + (1 - FPR) - 1) \tag{4.8}$$

Note, that the optimal threshold is calculated based on the training set and henceforth applied within the evaluation of the validation set.

In order to ensure a more comprehensive and unbiased evaluation of a model's performance, it is recommended to consider multiple metrics rather than relying on a single metric alone. This approach provides a more generalized overview of the model's performance across different aspects and helps to prevent any bias towards a single metric. To accomplish this, models can be ranked based on their performances on each individual metric, where a higher score or a lower loss indicates a better model, resulting in a higher rank for that metric. Subsequently, for each model, a rank score is calculated in order to determine the final rank. The lower rank score indicates a better model's performance. For a particular model M, the rank score can be calculated as a weighted average of the individual ranks as follows:

$$RankScore_M = \frac{\sum_{i=1}^{k} r_i \times w_i}{\sum_{i=1}^{k} w_i} \tag{4.9}$$

where k is the number of metrics used to evaluated the model M, r_i is the rank order of i–th metric for given model M and w_i is the weight assigned to the i–th metric. Such metric lies in interval $< 1, k >$, where 1 would indicate a perfect model' performance across the all metrics. Based on the rank scores,

we assign the final ranks where rank of 1 indicates the best performing model whereas rank k determines the worst performing model.

The weights have been set expertly and are summarized in Table 4.17. Specifically, the highest weight (1.5) is assigned to the F1 score, which provides a balanced measure of a model's performance with respect to both False Positives and False Negatives. This metric is commonly used in classification tasks, particularly in imbalanced data sets, such as the validation set in our case, which has not been oversampled. In addition to the F1 score, higher weight is assigned to the Recall score as well (1.2), which is a metric that penalizes False Negatives. False Negatives occur when the model predicts a negative result (i.e., no default) for an instance that is actually positive (i.e., default). In the context of loan applications, one may prefer to reject a loan applicant who would not have defaulted (False Positive) rather than approve the application of a client who would have defaulted (False Negative). Therefore, it is appropriate to give higher weight to Recall in order to reduce the likelihood of False Negatives. Henceforth, the weights are assigned to different metrics based on their relevance to the models' ranking, with the highest weight given to F1 score and additional weight given to Recall to ensure that False Negatives are minimized.

Table 4.17: Model Ranking Weights

Metric	Weight
F1 score	1.5
Recall	1.2
Precision	1
Accuracy	1
AUC	1
Somers' D	1
Kolmogorov Smirnov	1
Matthews Correlation Coefficient	1
Brier Score Loss	1

Source: Author's results in Python

Model Selection Results

The custom function `model_selection()` iteratively prints the process of the model tuning and evaluation on each subset of features, in order to keep the track of such process as it is depicted in Figure 4.18. Particularly, it prints which model on which features is being tuned and evaluated, execution time, optimal threshold, F1 score on the validation set, and the best hyperparameters.

Figure 4.18: Model Selection Print Statement

```
--------------------------------------------------------------------------------
---------------------------------------- 7/64 ----------------------------------
--------------------------------------------------------------------------------
---------------------------- BAYESIAN OPTIMIZATION OF LR ----------------------------
--------------------------- WITH FEATURES SELECTED BY SVM ---------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------

1/2 ... Starting Bayesian Optimization on the subset of features (11 features):
        LOAN, MORTDUE, VALUE, REASON, JOB, YOJ, DEROG, DELINQ, CLAGE, NINQ, DEBTINC
2/2... Bayesian Optimization finished

Execution time: 1.5373 minutes

F1 Score on Validation set: 0.6160919540229886

Optimal classification threshold: 0.4003

Tuned hyperparameters of LR:

        C: 0.2768320429063577
        class_weight: balanced
        dual: False
        fit_intercept: True
        intercept_scaling: 25.603205585229382
        l1_ratio: 0.5230523280543836
        max_iter: 100
        multi_class: auto
        n_jobs: -1
        penalty: elasticnet
        random_state: 42
        solver: saga
        tol: 0.0001
        verbose: 0
        warm_start: False

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Source: Author's results in Python

The final output of the function `model_selection()` is a table that summarizes the model's computed metrics as depicted in Table 4.18. For a reference **Tuned model** refers to the model over which the model selection algorithm is iterated, **FS model** refers to the model that was used as an input estimator in Forward SFS in order to select a subset of optimal features, **# Features** refers to the number of selected features on which the model is tuned, **Exec. Time** refers to the optimalization and training time of the tuned model, and **Thres.** refers to the optimal threshold computed by the Youden index (Equation 4.8).

The next columns represent the evaluation metrics, namely F1 score (**F1**), Precision (**Prec.**), Recall (**Rec.**), Accuracy (**Acc.**), AUC, Somers' D (**SD**), Kolmogorov-Smirnov Distance (**KS**), Matthews Correlation Coefficient (**MCC**), Brier Score Loss (**BS**), and Log Loss. The last two columns represent the rank score of the model computed according to Equation 4.9 (**Score**), and the last column (**Rank**) is the final rank based on which we determine the final model.

As can be seen, the best models in terms of ranking are the Gradient Boosting models, which in general have the highest score metrics and the lowest loss metrics, followed by another ensemble mode, Random Forest. On the other hand, the worst-performing models are Gaussian Naive Bayes models. In the first ten worst performing models, we can also observe that Decision Tree and Logistic Regression, as white–box models, perform poorly.

Table 4.18: Model Selection Results

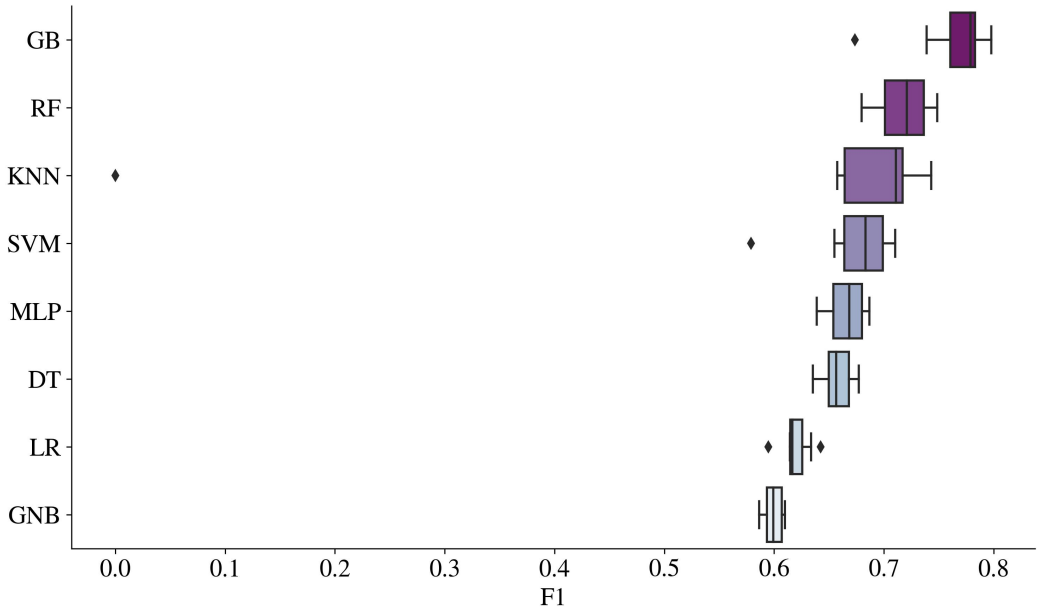
Tuned model	FS model	# Features	Exec. Time	Thresh.	F1	Prec.	Rec.	Acc.	AUC	SD	KS	MCC	BS	Log Loss	Score	Rank
GB	MLP	10	12.57	0.4955	0.7809	0.7853	0.7765	0.9128	0.9515	0.9030	0.7751	0.7265	0.0666	0.2487	3.30	1
GB	RF	11	14.10	0.4973	0.7775	0.7841	0.7709	0.9117	0.9541	0.9081	0.7961	0.7225	0.0669	0.2803	3.97	2
GB	KNN	11	14.18	0.5072	0.7978	0.7912	0.8045	0.9184	0.9587	0.9175	0.7989	0.7467	0.0687	0.4135	4.08	3
GB	SVM	11	15.33	0.5053	0.7896	0.8155	0.7654	0.9184	0.9555	0.9109	0.7961	0.7397	0.0718	0.3486	4.32	4
GB	GB	11	11.86	0.4132	0.7799	0.7778	0.7821	0.9117	0.9543	0.9086	0.7989	0.7247	0.0703	0.3372	4.39	5
RF	KNN	11	6.01	0.4725	0.7486	0.7326	0.7654	0.8972	0.9226	0.8452	0.7109	0.6843	0.0923	0.3232	8.48	6
GB	DT	9	12.91	0.4729	0.7675	0.7697	0.7654	0.9073	0.9407	0.8814	0.7556	0.7096	0.0808	0.4693	8.80	7
RF	GB	11	5.18	0.4517	0.7385	0.7135	0.7654	0.8916	0.9200	0.8400	0.7123	0.6709	0.0886	0.3135	9.22	8
RF	MLP	10	6.72	0.4761	0.7357	0.7181	0.7542	0.8916	0.9179	0.8358	0.7081	0.6679	0.0879	0.3084	10.20	9
GB	LR	7	17.58	0.4362	0.7388	0.7000	0.7821	0.8894	0.9219	0.8438	0.7081	0.6706	0.0886	0.3925	11.03	10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
GNB	DT	9	0.40	0.2241	0.6063	0.5095	0.7486	0.8056	0.8467	0.6935	0.5768	0.4991	0.1316	0.6370	49.56	55
GNB	MLP	10	0.39	0.2194	0.6013	0.5000	0.7542	0.8000	0.8493	0.6986	0.5768	0.4930	0.1319	0.6360	49.68	56
GNB	KNN	11	0.35	0.3588	0.6093	0.5219	0.7318	0.8123	0.8479	0.6957	0.5698	0.5024	0.1367	0.6686	50.58	57
DT	MLP	10	0.87	0.5000	0.6427	0.6374	0.6480	0.8559	0.8133	0.6266	0.5810	0.5524	0.1254	2.0008	52.10	58
SVM	GNB	6	6.66	0.1821	0.5788	0.4718	0.7486	0.7821	0.8393	0.6786	0.5768	0.4633	0.1355	0.4354	53.11	59
DT	GNB	6	0.85	0.5000	0.6354	0.6284	0.6425	0.8525	0.8187	0.6375	0.5838	0.5430	0.1251	2.1679	53.31	60
GNB	SVM	11	0.36	0.1991	0.5885	0.4872	0.7430	0.7922	0.8425	0.6850	0.5712	0.4756	0.1345	0.6721	53.65	61
GNB	GNB	6	0.36	0.4787	0.5950	0.5039	0.7263	0.8022	0.8356	0.6711	0.5559	0.4835	0.1470	0.5280	53.79	62
LR	GNB	6	1.55	0.4561	0.5948	0.5121	0.7095	0.8067	0.8379	0.6758	0.5531	0.4831	0.1319	0.4180	54.02	63
GNB	RF	11	0.36	0.3413	0.5864	0.4943	0.7207	0.7966	0.8288	0.6577	0.5545	0.4720	0.1486	0.7040	57.08	64

Source: Author's results in Python

In order to gain a more detailed understanding of the model selection results, the distribution of F1 score is plotted per each model as depicted in Figure 4.19. An outlier can be observed in KNN where the F1 score is 0. This is caused by the derived optimal threshold using Youden index which happens to be 1. Therefore, based on this threshold, the model predicts everything as non–default. After a detailed inspection, we observed that this particular KNN has predicted probability scores close to 0 or 1, respectively, as can be seen in Figure A.3 in Appendix A. This indicates that the KNN is very confident about distinguishing defaulters from non–defaulters.

Referring to the Youden index, which tries to maximize the TPR and TNR, we also inspect the ROC curve of such KNN model in Figure A.4 in Appendix A. It can be seen that the ROC curve reaches the top left corner, which might be considered as a good model. However, this leads to the result that the maximum of the sum of TPR and TNR corresponds to the threshold of 1. In such case, a threshold of 0.5 would be more appropriate since the probability scores are clearly separable. Therefore, we recommend adjusting the optimal threshold calculation by imposing a constraint on the thresholds derived from the ROC curve, particularly, if the thresholds lie close to 0 or 1 only, then the threshold should be set to 0.5. This would prevent the model from predicting everything as non–default.

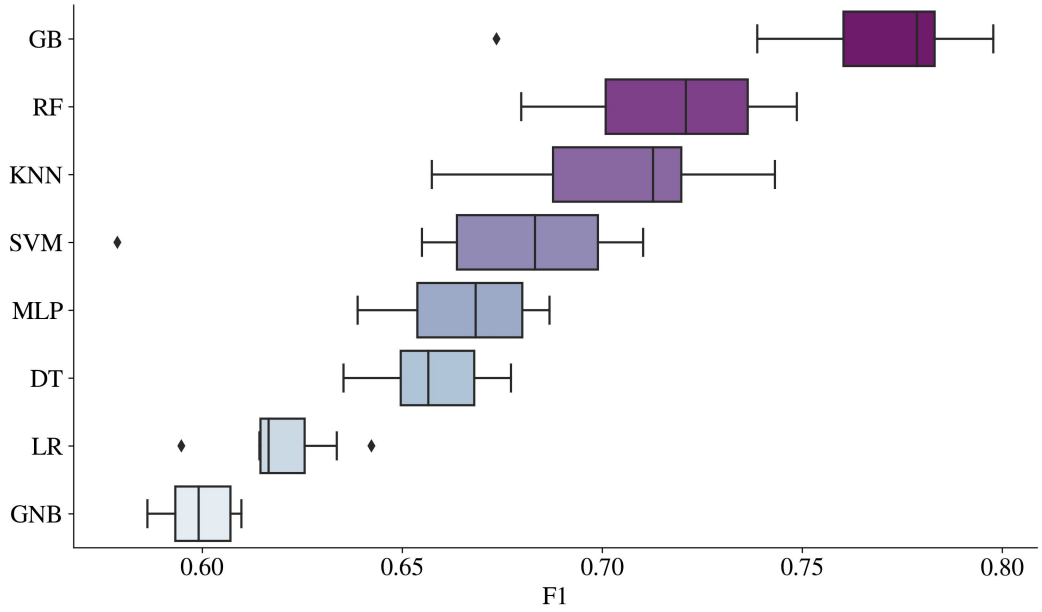
Figure 4.19: F1 Score Distribution



Source: Author's results in Python

Such outlier is removed in Figure 4.20 to gain a more general insight into the F1 score distribution. It can be observed that Gradient Boosting models have the highest F1 scores of around 80 %. Another tree ensemble model, Random Forest, is performing well as the second best. However, more transparent models such as Logistic Regression and Naive Bayes are performing poorly, having F1 scores around 60 %.

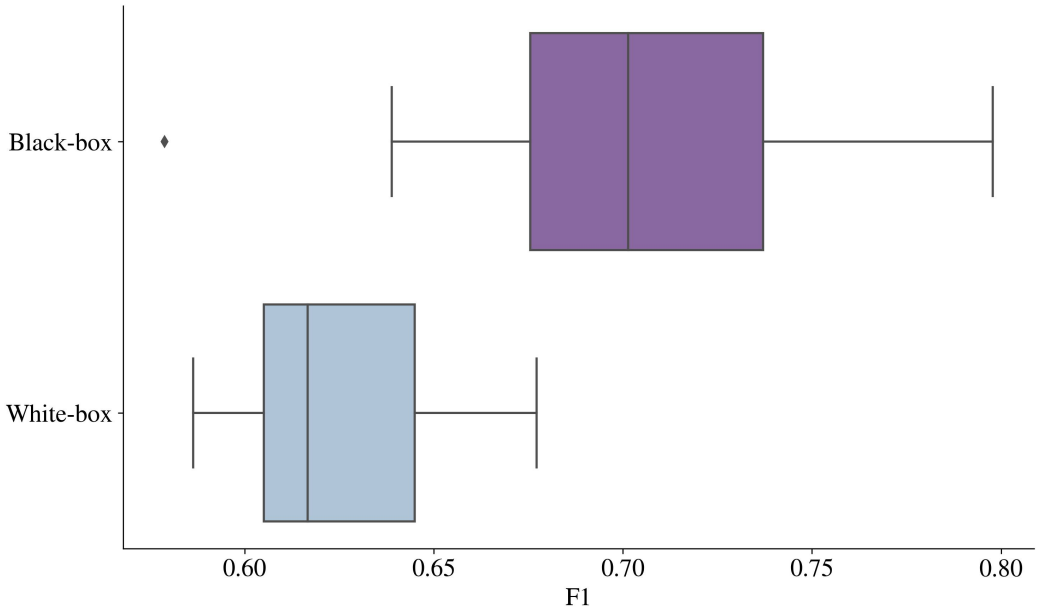
Figure 4.20: F1 Score Distribution - *without outlier*



Source: Author's results in Python

As depicted in Figure 4.21, it visualizes the distribution of F1 score (without the outlier) across the model type. Although, both score distributions overlap, it is evident, that black–box models outperform white–box models in terms of F1 score in overall as expected.

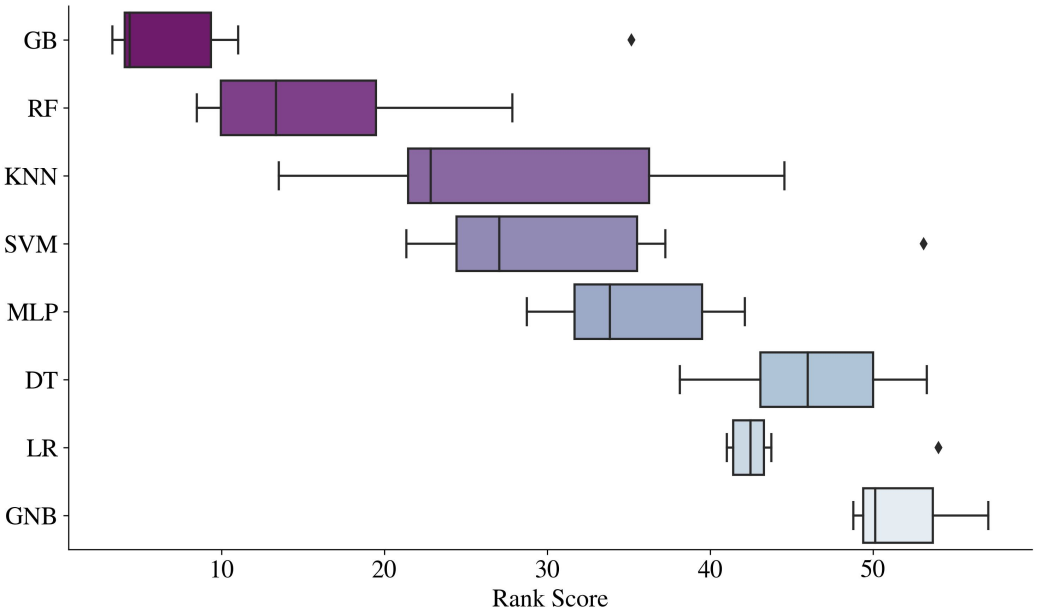
Figure 4.21: F1 Score Distribution (Black–box/White–box dimension) - *without outlier*



Source: Author's results in Python

We can also consider not only the F1 score but also other metrics, which is quantified in the rank score calculated according to Equation 4.9. As shown in Figure 4.22, we can observe that the order of models ranked by the rank score is identical as the order of the models ranked by the F1 score in Figure 4.19. Hence, across all the metrics, the Gradient performs the best in overall while Gaussian Naive Bayes does not perform well at all.

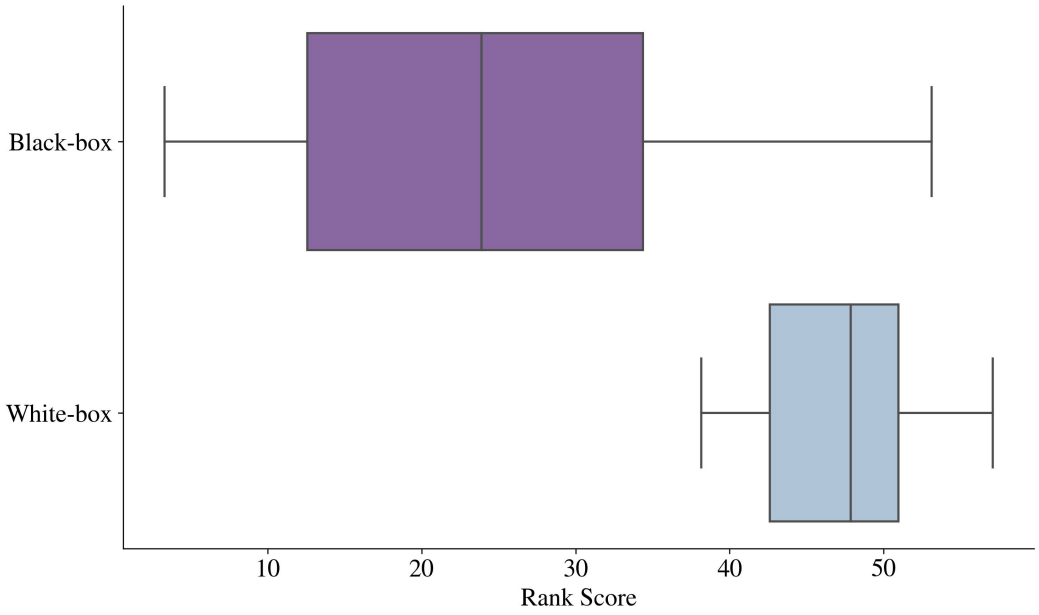
Figure 4.22: Rank Score Distribution



Source: Author's results in Python

As expected, also, the black–box models are ranked higher than the white–box models according to the rank score as shown in Figure 4.23. This is evident also from Table 4.18 where the black–box models, namely Gradient Boosting and Random Forest, dominated amongst the first 10 ranked models. Other metrics' distribution analyses are shown in Section A.1 in Appendix A.

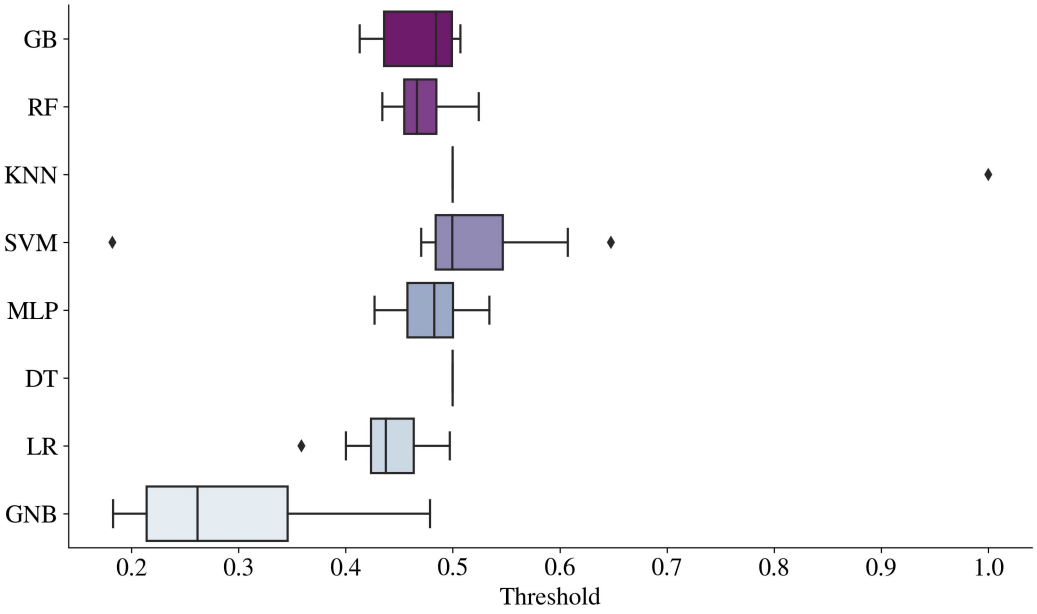
Figure 4.23: Rank Score Distribution (Black–box/White–box dimension)



Source: Author's results in Python

The optimal threshold distribution for each base model is presented in Figure 4.24. We can observe an outlier in KNN having a threshold value of 1, which explains the F1 score outlier found previously in Figure 4.19.

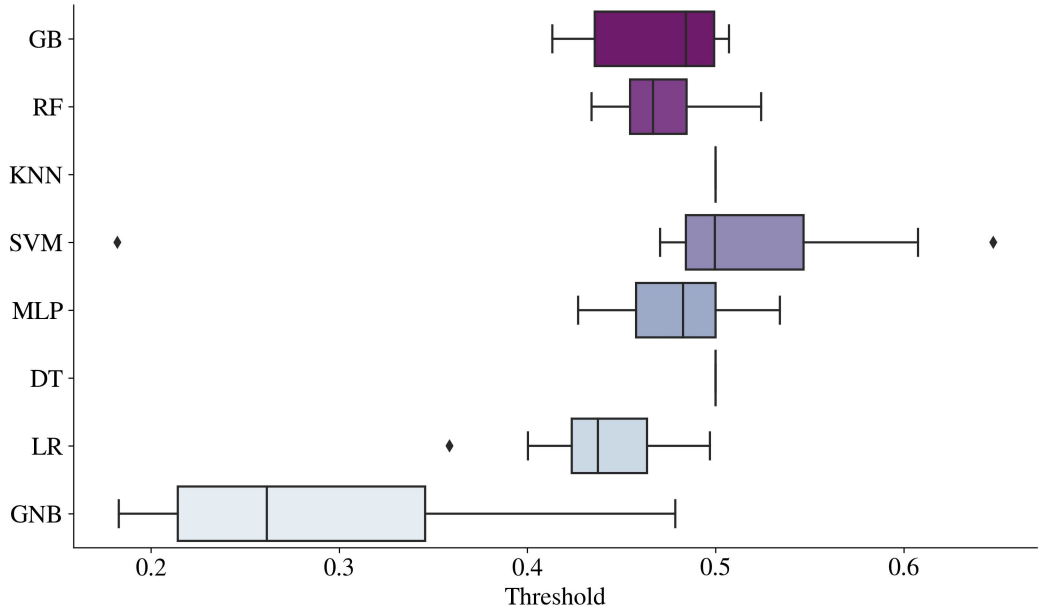
Figure 4.24: Threshold Distribution



Source: Author's results in Python

In order to obtain a better insight into the distribution of optimal thresholds, the outlier in KNN is excluded, resulting in the threshold distribution depicted in Figure 4.25. The optimal threshold values are mostly distributed below 0.5, indicating that the models are generally more conservative. The most conservative model is Gaussian Naive Bayes, which has a median threshold value of around 0.25.

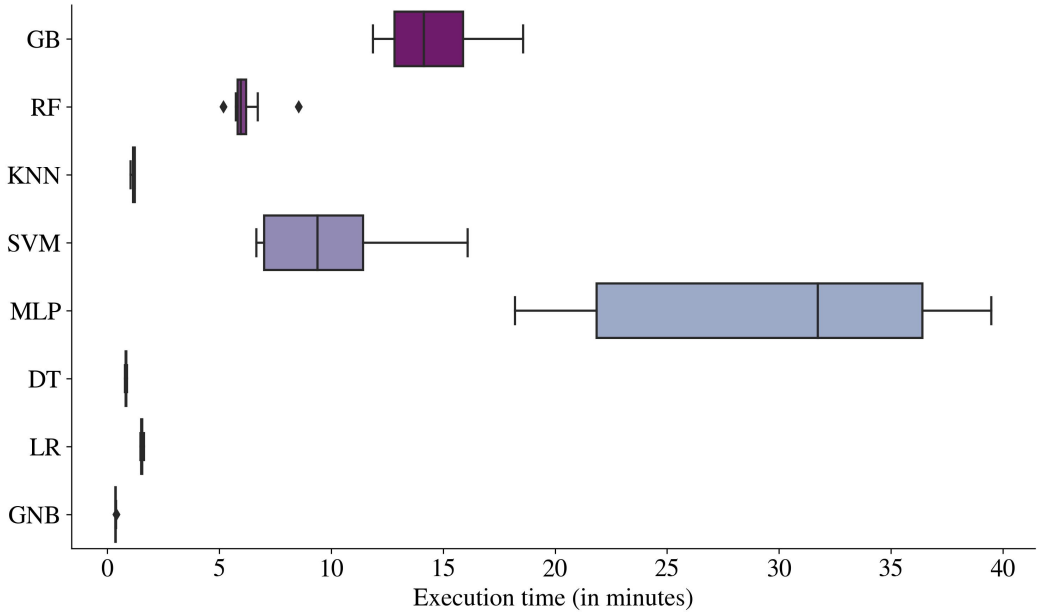
Figure 4.25: Threshold Distribution - *without outlier*



Source: Author's results in Python

Upon examining the execution time of each model, it can be observed that the transparent and non-complex models such as Logistic Regression, Gaussian Naive Bayes, or even Decision Tree, which take around only 1 minute to optimize themselves, also perform poorly, as already inspected in Figure 4.20. Conversely, the most time-consuming models are undoubtedly the Neural Network models, which take around 30 minutes to optimize themselves. Other time-consuming models include Gradient Boosting and Support Vector Machine, which take around 13 and 11 minutes to optimize themselves, respectively. This finding suggests that longer execution time does not necessarily lead to better performance, as the Neural Network models are significantly outperformed by several other models.

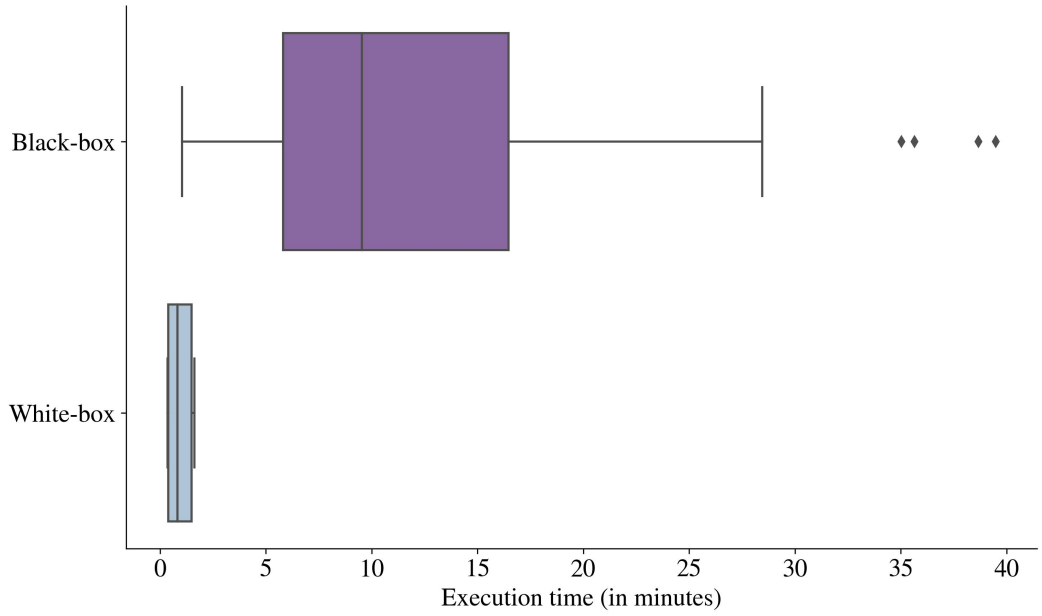
Figure 4.26: Execution Time Distribution



Source: Author's results in Python

We can inspect the execution time from another perspective as shown in Figure 4.27. It can be observed that black–box models take a significantly longer amount of time to optimize than the white–box models, which is due to the complexity of the black–box models. Further we can notice that some of the black–box models took only a few minutes to optimize themselves. This corresponds to KNN models, and it can be attributed to the small sample size of the data set as well as the lower data set dimensionality (i.e., low number of features) and further given the lower number of k neighbors (which is constrained to 10 at most as depicted in Table 4.12).

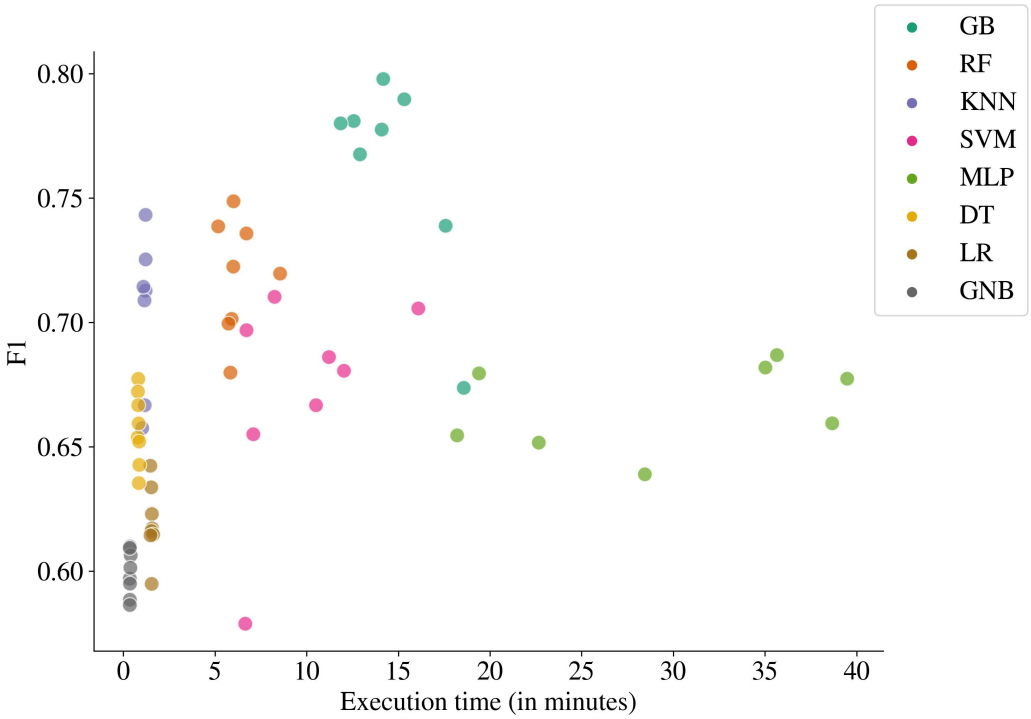
Figure 4.27: Execution Time Distribution (Black–box/White–box dimension)



Source: Author's results in Python

The execution time and the F1 score are inspected together using a scatterplot, as shown in Figure 4.28. A cluster of non-complex and transparent models, such as Logistic Regression, Gaussian Naive Bayes, and Decision Tree, can be observed around the vertical line near 0 execution time. These models are quick to optimize, but their F1 scores are generally low. Further, their variance in execution time is quite low, regardless of the feature subset they are optimized on. On the other hand, the Neural Network models always perform poorly, regardless of the length of the execution time. Furthermore, the variance of the F1 scores is quite low for these models, indicating that the execution time does not have a significant impact on the F1 score in the case of Neural Networks.

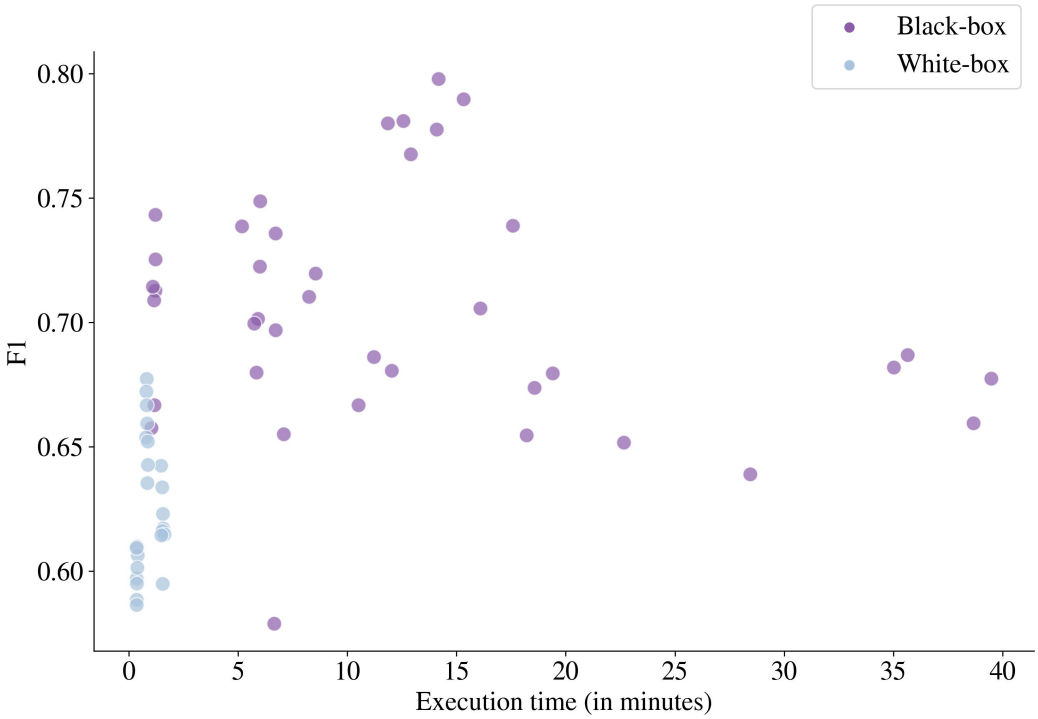
Figure 4.28: Execution Time vs. F1 Scatterplot - *without outlier*



Source: Author's results in Python

Such separation of the black–box and white–box models is more evident from Figure 4.29 where the white–box models points are light blue-colored and the black–box models are purple-colored. While the score of white–box models seems to be constant regardless of the execution time length, the points of black–box models are more dispersed across the execution time–F1 score dimension.

Figure 4.29: Execution Time vs. F1 Scatterplot (Black–box/White–box dimension) - *without outlier*



Source: Author's results in Python

To summarize this subsection, the best and final model is the **Gradient Boosting Classifier** which was optimized on the subset of features selected by **Multi–Layer Perceptron** - both the model information and its final hyperparameters' values are described in Table 4.19. Such model is then used in the next modelling steps, including calibration, evaluation, and deployment.

Table 4.19: Final Model Information

Final Model	Gradient Boosting
FS Model	Multi–Layer Perceptron
Final Features	MORTDUE, VALUE, JOB, YOJ, DEROG, DELINQ, CLAGE, NINQ, CLNO, DEBTINC
Threshold	0.4955
F1	0.7809
# estimators	1,000
Criterion	Friedman MSE
Max depth	10
Max features	1
Loss	Log loss
Learning rate	0.0150

Source: Author's results in Python

4.4.4 Model Recalibration

In order to ensure that the final model performs well on unseen data, it is desired practice to employ the recalibration approach, which involves retraining the model on both the training and validation sets. By doing so, the sample size used for training is increased. As such, the re-training (or so called recalibration) model leads (or is expected to lead) to a markedly improved model's performance (de Hond *et al.* 2023). The recalibrated model is then used to evaluate the performance of the final model on the test set, which is the ultimate measure of a model's performance.

In addition to recalibrating the final model, it is deemed appropriate to recalibrate the threshold value for assigning class labels based on predicted probabilities. The optimal threshold value can be determined using the training and validation sets. Such threshold is recalibrated based on the training and validation sets. In this thesis, the optimal threshold value is found to be **0.45109**, which is then used for evaluating the final model's performance on the test set. We can observe that the model is more conservative as its optimal classification threshold has decreased. The model recalibration impact on the model's performance is further assessed in Subsection 4.5.1.

Moreover, the recalibration process helps to mitigate overfitting issues, which occur when the model is only trained on the training set. By incorporating the validation set into the training process, the recalibrated model can better generalize to new data and improve its overall performance on the test set. The inclusion of the validation set during the recalibration process does not cause any data leakage issues since this set was already used during model selection to evaluate each model's performance. Therefore, using the validation data for recalibration is a sound practice that helps to ensure the reliability and accuracy of the final model.

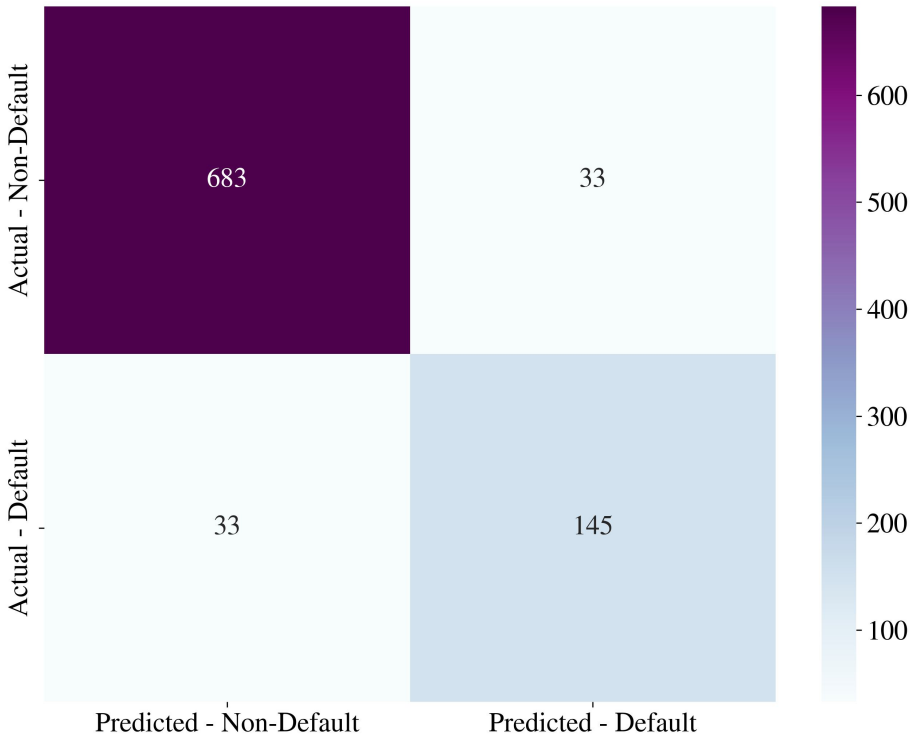
4.5 Model Evaluation

After recalibrating the model and threshold, the final step in evaluating the model's performance is to test it on previously unseen data, specifically the test set. This evaluation is critical to determining whether the model can generalize well to new data beyond the training, feature selection, and model selection phases. During the evaluation, the recalibrated classification threshold of 0.45109 is also used.

4.5.1 Model Performance Assessment

In Figure 4.30, the confusion matrix for the final model, based on the test set and using the recalibrated threshold, is presented. The matrix shows that the model is generalizing well, having correctly predicted 145 defaults and misclassified only 33 defaults, and further, correctly predicted 683 non–defaults and misclassified only 33 non–defaults. Such a result indicates that the model is a good fit for the data and can provide useful predictions.

Figure 4.30: Confusion Matrix



Source: Author's results in Python

In order to obtain a better understanding of the model's performance on previously unseen data, we computed several metrics that were used during the model selection process. These metrics are presented in Table 4.20. The results indicate that the model performs well on the unseen data, with most of the scores metrics around 80 % to 90 %. Furthermore, the loss metrics are relatively low, indicating that the model can effectively distinguish between defaults and non-defaults. Particularly, out of all test instances, the model predicts correctly 92.62 % default instances. The model also predicts 81.46 % instances out of all the actual default instances. Moreover, out of all predicted default instances, the model correctly predicts 81.46 % actual default instances. Overall, these results suggest that the model is performing well and is suitable for predicting defaults. The results suggest that the model has a good balance between correctly identifying defaults and non-defaults, as well as minimizing False Positives and False Negatives. This further confirms the model's ability to accurately predict defaults.

Table 4.20: Metrics Evaluation

Metric	Value
F1	0.8146
Precision	0.8146
Recall	0.8146
Accuracy	0.9262
AUC	0.9564
Somers' D	0.9128
KS	0.7915
MCC	0.7685
Brier Score Loss	0.0594
Log Loss	0.2163

Source: Author's results in Python

The following Table 4.21 shows the impact of recalibration on the model's performance metrics. M_{NR} denotes the final model that was not recalibrated at all (i.e., trained only on the training set), whereas M_R denotes the recalibrated model (i.e., trained on the joined training and validation sets). Respectively, M_{NR} uses its threshold computed based on the training set (0.4955), and M_R uses its threshold computed on the joined training and validation set(0.45109). As can be seen, the recalibration indeed has a positive impact on the metrics

evaluation, as all the metric scores have increased, while the metric loss functions have decreased. We can observe the most significant decrease in the Log Loss function, which has decreased by 8.09 %, while the objective function F1 score has increased by 3.30 % thanks to increases in both Precision and Recall. Therefore, the recalibration process is deemed desirable and appropriate in terms of the model's performance.

Table 4.21: Recalibration Impact on Metrics Evaluation

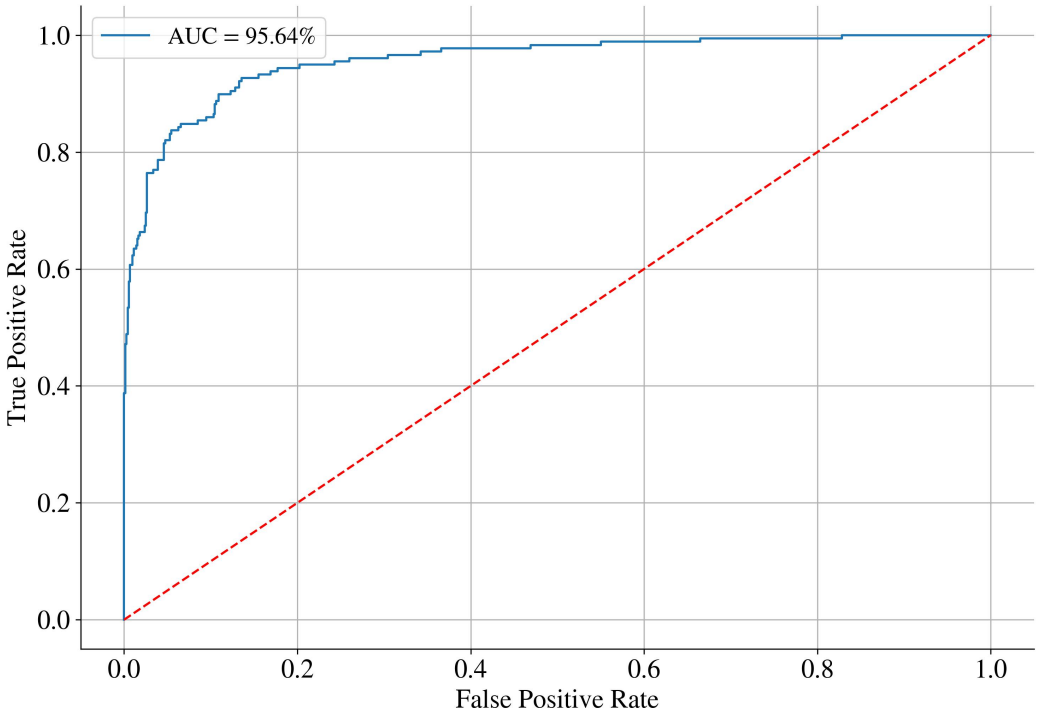
Metric	M_{NR}	M_R	**Diff.**
F1	0.7886	0.8146	3.30 %
Precision	0.8023	0.8146	1.53 %
Recall	0.7753	0.8146	5.07 %
Accuracy	0.9172	0.9262	0.98 %
AUC	0.9518	0.9564	0.48 %
Somers' D	0.9037	0.9128	1.01 %
KS	0.7845	0.7915	0.89 %
MCC	0.7373	0.7685	4.23 %
Brier Score Loss	0.0632	0.0594	-6.11 %
Log Loss	0.2353	0.2163	-8.09 %

Source: Author's results in Python

To further evaluate the performance of the recalibrated final model, we can visualize the ROC curve, as presented in Figure 4.31. The curve illustrates the trade-off between the True Positive Rate and the False Positive Rate at various classification thresholds. An ideal ROC curve should have an Area Under the Curve (AUC) value of 100 %, indicating a perfect classifier, while a random classifier would have an AUC of 50 %.

From the ROC curve plot, we observe that the AUC value of the model is 95.64 %, indicating a high degree of accuracy in distinguishing between defaults and non-defaults. The curve covers most of the area above the diagonal line, indicating that the model is performing well in differentiating the two classes. Therefore, the results suggest that the model is performing well and is capable of accurately identifying potential defaulters.

Figure 4.31: ROC Curve



Source: Author's results in Python

4.5.2 Model Explainability

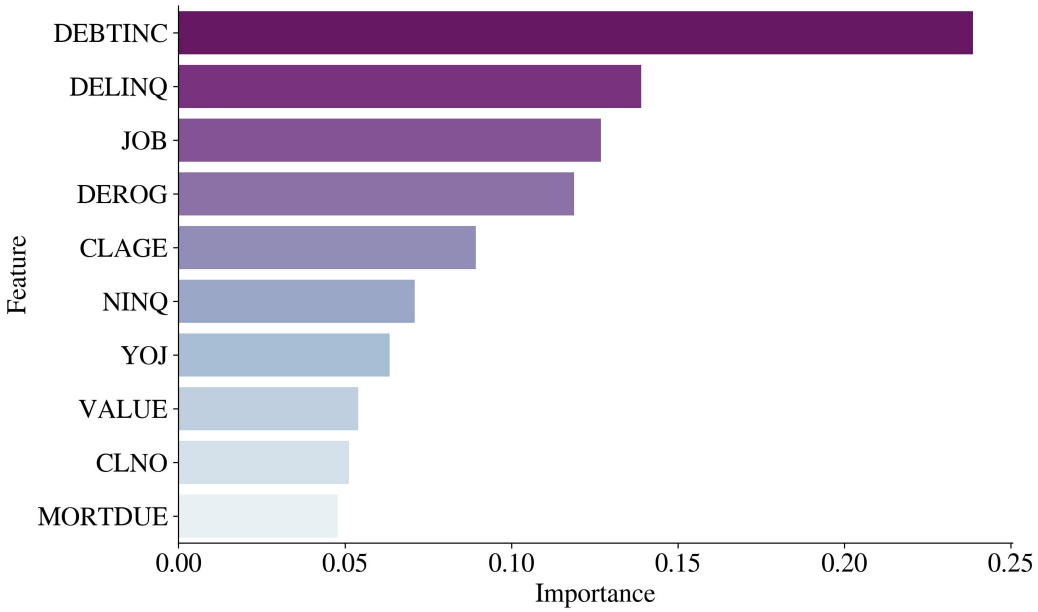
To gain insights into the impact of the features used in the final model, we can inspect the feature importances of the final model, which is a part of a family of tree ensemble algorithms. As the name indicates, it is the score value representing the importance of the features, i.e., the higher the score, the more important the feature is. Basically, it is a measure of how much a feature contributes to the overall performance of the model. Overall, the feature importance plot provides valuable insights into the factors that are most important in predicting loan defaults. It can be used to identify which features are contributing the most to the model's accuracy and to guide future feature selection efforts. According to Bonaccorso (Bonaccorso 2020), feature importance is the measure proportional to the impurity reduction that a particular feature allows us to achieve, and is defined as:

$$\mathrm{FI}\left(\bar{x}^{(i)}\right) = \frac{1}{N}\sum_{k=1}^{N}\sum_{j=1}^{L}\frac{n(j)}{M}\delta I_j^i \tag{4.10}$$

where FI $\left(\bar{x}^{(i)}\right)$ refers to the feature importance of the feature i, $n(j)$ is the number of samples reaching the node j, δI_j^i represents the impurity reduction at node j after the splitting using the feature j, M is total number of samples in the data set used to built the model, and N refers to the number of tree estimators used within an ensemble model.

The following Figure 4.32 depicts the feature importances of all the selected features on which the final model was trained. The two most important features used in the final model are `DEBTINC` and `DELINQ`, which are crucial debt and delinquency indicators in determining whether a borrower would be able to repay their loan. These two features have a significant impact on the model's ability to accurately predict loan defaults, with high feature importance scores. This is also in line with the findings from the exploratory analysis, WoE distribution, or feature selection.

Figure 4.32: Feature Importance



Source: Author's results in Python

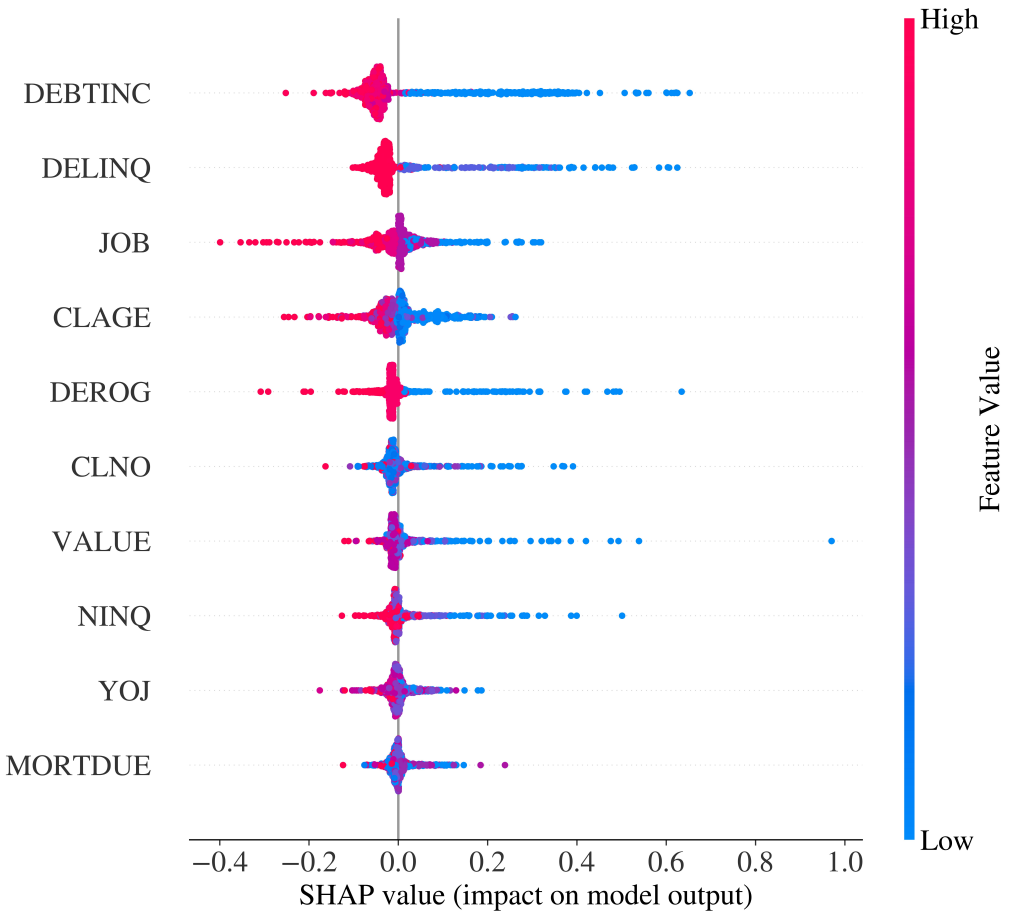
Thus, understanding the impact of individual features on model's performance can be useful in identifying areas for improvement, as well as identifying the most significant factors that drive loan defaults. By focusing on these important features, lenders and policymakers can better understand and address the underlying factors that contribute to default risk, ultimately leading to better lending decisions and improved outcomes for borrowers and lenders alike.

To gain further insights into the impact of the features on the final model's predictions, the SHapley Additive exPlanations (henceforth SHAP) values can be calculated. Particularly, the SHAP value is defined as the mean marginal contribution of each feature value across all possible values in the feature space, while considering feature importance (Bhattacharya 2022). The following Figure 4.33 depicts the SHAP summary plot, which provides a clear visualization of the contribution each feature makes to a prediction and the global explainability of the model predictions. Each dot in the plot represents a feature and its corresponding SHAP value. The color of the dot represents the feature's value, with red indicating high values and blue indicating low values. The position of the dot on the x-axis represents the impact of the feature on the prediction, with features on the right-hand side contributing more positively to the prediction, and features on the left-hand side contributing more negatively.

Since our data points are encoded in WoE values, the higher (the more positive) value, the larger the distribution of non-defaulters compared to defaulters, and vice versa, the lower (the more negative) value, the larger the distribution of defaulters compared to non-defaulters. For the most important features, we can observe that the blue values (negative WoE values) and red values (positive WoE values) are quite separable. Negative WoE values are positively contributing to the predictions, and positive WoE values are negatively contributing to the predictions. This means that the more negative the WoE value, the more likely the borrower is to default, and vice versa.

Overall, the SHAP summary plot provides a valuable tool for interpreting and understanding the complex decision-making process of the final model. The plot enables us to examine the impact of individual features on the model's output and helps us identify which features are most influential in the model's decision-making process. By understanding the relative importance of each feature, we can gain deeper insights into the creditworthiness assessment process and make more informed decisions in the lending industry.

Figure 4.33: SHAP Summary Plot



Source: Author's results in Python

4.6 Machine Learning Deployment

This section describes the process of taking a trained machine learning model and making it available for use in the real world. It involves taking the model from a development environment and integrating it into a production environment, where it can be used to make predictions or decisions based on new data. In this thesis, the machine learning model is deployed as a web application using Flask and HTML.

4.6.1 Final Model Recalibration

Before deploying the model in a production setting, we undertake the final recalibration of the model using the entire data set, comprising the training, validation, and test sets. The purpose of this recalibration is to fine-tune the model parameters, thereby maximizing its ability to generalize and yield accurate predictions.

The test set, which has been employed previously during the model evaluation phase, is incorporated into the recalibration process without any risk of data leakage. By including the test set, the sample size for training is expanded, thereby increasing the model's capacity to generalize effectively.

Upon completion of the recalibration, the final classification threshold for deployment is determined to be **0.3358**. We believe that these recalibrated final model's parameters and threshold are optimal for implementation in production settings and will exhibit strong generalization capabilities.

4.6.2 Flask and HTML Web Application

In this case, the machine learning model is deployed into a web application using Flask and HTML. The application is temporarily deployed on the Cloud server on the **PythonAnywhere** platform and is accessible here: `http://ml-credit-risk-app-petrngn.pythonanywhere.com/`. However, the application will be shut down after the thesis defense and will not be available online anymore due to budgetary reasons. Nevertheless, the code for the application is available in the GitHub repository, and the application can be run locally using any Python compiler.

Furthermore, prior to deployment, we need to prepare several Python inputs for the web application, including:

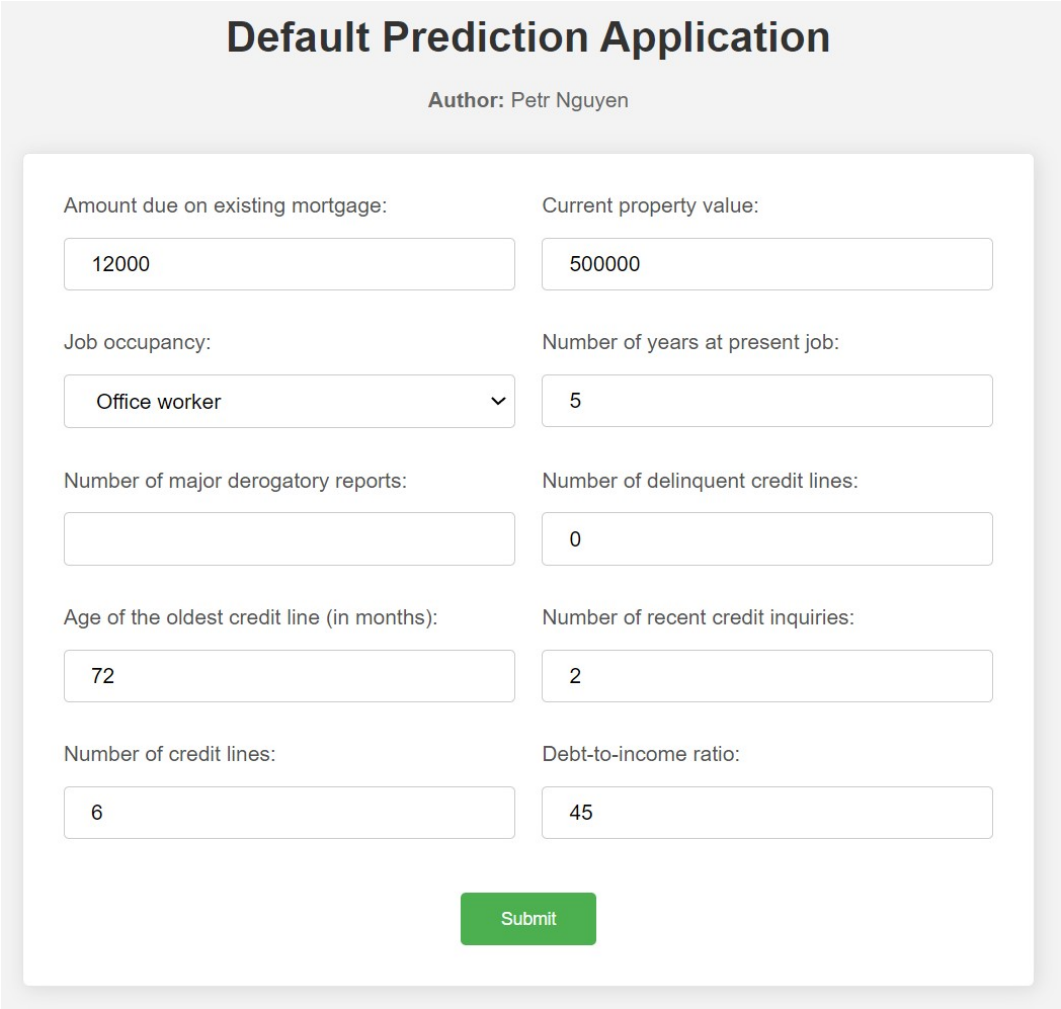
- **Model** - The final model recalibrated on the training, validation, and test sets.
- **Threshold** - The final classification threshold recalibrated on the training, validation, and test sets.
- **Features** - The final features used in the final model.
- **Data Frame** - The input tabular object used in the web application to store the loan applicant's inputs.
- **Optimal Binning Transformator** - Fitted `BinningProcess` object for binning and WoE-encoding of the loan applicant's inputs.
- **WoE Bins** - Set of bins and WoE values used for mapping the missing values to WoE values.
- **LIME explainer** - Fitted `LimeTabularExplainer` object for local explainability of the model's prediction.

Such inputs required for the machine learning application are exported in the `.pkl` format using the `dill` module. This format allows for efficient and easy-to-use serialization and deserialization of the inputs. The pickled file is then loaded directly into the Flask application.

For the back-end of the web application, Flask is used to deploy the machine learning model. The Flask application is written in the `app.py` file, which is stored in the `flask_app` directory. The front-end of the web application is coded in HTML, with CSS elements used to enhance the user interface.

The web application first renders a HTML page, as shown in Figure 4.34. This page contains a loan application form, in which the user or the loan applicant fills in the respective field values that correspond to the features on which the machine learning model was trained. The form is designed to capture the necessary information required for the model to make a prediction about the loan applicant's application. None of the fields in the loan application form need to be filled out. This is because missing values in certain features may indicate a higher risk of default. Conversely, one may choose to impose a restriction on the form, requiring all fields to be filled out. However, this could result in a lower number of received applications from delinquent clients, as they may not have all the necessary information to complete the form.

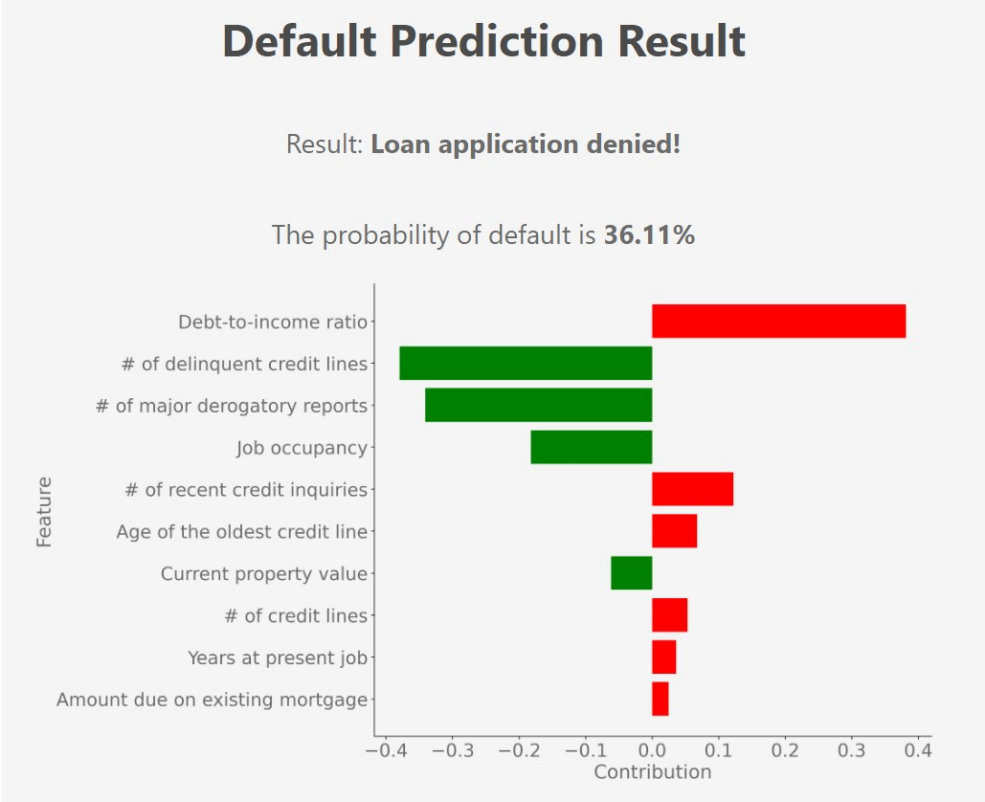
Figure 4.34: Flask Web Application Form



Source: Author's results in Python

Once, the loan application form is submitted, the web application uses the pickled input to transform the data from the loan application form and use it in the recalibrated model in order to get the result, whether the given loan applicant would repay his loan based on a predetermined threshold. The result is then displayed in the web application, as shown in Figure 4.35. Particularly, the web application returns whether the loan application would be denied or approved based on the model's output, and also the probability score of default.

Figure 4.35: Flask Web Application - Prediction Result



Source: Author's results in Python

Besides the prediction results, it also displays the Local Interpretable Model-Agnostic Explanations (henceforth LIME) of the black–box model with respect to the inputs submitted within the form. LIME focuses on the local explainability of the black box model around the black–box prediction as it generates a new data set consisting of perturbed samples around the given prediction and then trains a surrogate linear model on the new data set. Such local interpretable, surrogate model should be a good approximation of the black box model in the vicinity of the given prediction, i.e., the local interpretable model is then used to explain the prediction of the black box model (Ribeiro *et al.* 2016).

The LIME explanation of input instances x is given as follows:

$$\xi(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g) \tag{4.11}$$

where f is the original black–box model, g is the surrogate model, L is the loss function measuring how far the explanation $\xi(x)$ is from the prediction

produced by the black–box model f, and $\Omega(g)$ is the complexity of the surrogate model g.

The explanation is given in terms of the feature importance, which is represented by the magnitude of the feature's coefficient in the local interpretable model. The higher the magnitude of the coefficient, the more important the feature is in the prediction of the black box model. Therefore, as shown in Figure 4.35, the red bars indicate a positive contribution to the probability of default, whereas the green bars indicate a negative contribution to the probability of default. In other words, the features with red bars indicate that the client would probably not repay his loan, and vice versa. The contributions' magnitudes are in line with the findings from feature importance or SHAP values, which are focused on the global explainability of the black–box model (not local explainability), that features such as debt–to–income ratio (`DEBTINC`) or number of delinquent credit lines (`DELINQ`) have the largest impact on the probability of default. Particularly, a high debt–to–income ratio causes a higher probability of default, whereas no delinquent credit lines lead to a lower probability of default.

Chapter 5

Summary of Results

In this chapter, we evaluate the hypotheses presented in Section 5.1, as outlined in Section 3.1. These evaluations are based on the results obtained from the implementation of machine learning techniques described in Chapter 4. Furthermore, we compare our results with those of previous studies on HMEQ data set (Aras 2021; Zurada *et al.* 2014) in Section 5.2.

Additionally, we provide a summary of the key findings derived from the machine learning implementation from Chapter 4 in Section 5.3. Moreover, we delve into the author's contributions to the field of credit risk modelling and machine learning in Section 5.4. Lastly, we address the key recommendations to improve this thesis and its machine learning implementation or for future research in Section 5.5.

5.1 Hypotheses' Testing

In this section, we evaluate the hypotheses presented in Section 3.1 based on the results obtained from the machine learning implementation described in Chapter 4. The results of hypotheses' testing are summarized in Table 5.1. As can be seen, out of five proposed hypotheses, we reject two hypotheses and fail to reject three hypotheses. Such results are further described in the following paragraphs.

Table 5.1: Hypotheses' Results

#	Hypothesis	Rejected
H1	*The recalibration of the model enhances the model's performance on HMEQ data set.*	NO
H2	*Either Neural Network or KNN model outperforms all the models on HMEQ data set.*	YES
H3	*Black–box models perform better than white–box models on HMEQ data set.*	NO
H4	*The longer execution time of a model, the better its performance on HMEQ data set.*	YES
H5	*The main default drivers are the debt and/or delinquency features on HMEQ data set.*	NO

Source: Author's Results

Result #1: *The recalibration of the model DOES enhance the model's performance on HMEQ data set*

We fail to reject the **Hypothesis #1** because we observe improvements across all the metrics within evaluation on the test set when the final model was re-trained on the joined training and validation set instead of solely on the training set. In particular, all the score metrics and loss metrics have increased and decreased, respectively, as can be seen in Table 4.21. Hereby, we can confirm that the recalibration of the model truly boost the model's predictive power on HMEQ data set.

Result #2: *Neural Network and KNN model DO NOT outperform all the models on HMEQ data set.*

We reject the **Hypothesis #2** as the final, best performing, model is the Gradient Boosting as described in Table 4.19. KNN was also outperformed by the Random Forest and in the case of Neural Network, it was outperformed by SVM as well, as shown in Figure 4.22. If we aggregate the model selection results by looking at the highest rank per each model, we can observe that the best KNN model had the 12th highest rank, while Neural Network (MLP) exhibited weak performance with the 26th highest rank, as depicted in Table 5.2. Therefore, our results are not in with line with the outcomes of respective studies (Aras 2021; Zurada *et al.* 2014) which reported that Neural Network and KNN model outperformed all the models on HMEQ data set, respectively.

Table 5.2: Max–Aggregated Ranks of Models

Model	Rank
GB	1
RF	6
KNN	12
SVM	17
MLP	26
DT	35
LR	39
GNB	52

Source: Author's results in Python

Result #3: *Black–box models DO perform better than the white–box models on HMEQ data set.*

We fail to reject the **Hypothesis #3** as the black–box models truly outperformed the white–box models, as can be seen in Figure 4.23, which depicts the distribution of ranks for both black–box and white–box models. Most of the white–box models were ranked in the bottom half of the model selection, whereas the 10 best performing models consisted of black–box models only, namely Gradient Boosting and Random Forest. Even though some black–box models' performances were weak and other white–box models exhibited relatively high ranks, we can still conclude that the black–box models outperformed the white–box models on average, particularly on HMEQ data set.

Result #4: *The longer execution time of a model DOES NOT indicate better performance on HMEQ data set.*

We reject the **Hypothesis #4** as according to Figure 4.26, even though the Neural Network (MLP) model had the longest execution time on average, it underperformed several models such as Gradient Boosting, Random Forest, KNN and SVM which took significantly less time to execute. The same can be observed from Figure 4.28, where Neural Network performed poorly regardless of the length of the execution time.

Result #5: *Debt and delinquency features ARE the main default drivers on HMEQ data set.*

We fail to reject the **Hypothesis #5** according to Figure 4.32 which depicts the feature importance of the final model (Gradient Boosting). As can be seen, the most important features are debt–to–income ratio `DEBTINC` and number of delinquent credit lines `DELINQ`. Based on the SHAP values depicted in Figure 4.33, we can observe that the negative values of `DEBTINC` and `DELINQ` positively contribute to the model's predictions of the target variable, whereas the positive values negatively contribute to the model's predictions. In other words, the higher the value of such features, the higher the probability of default, and vice versa. Since the features' values are encoded as WoE, the negative WoE values indicate a larger distribution of defaulters compared to non–defaulters in given bins. Based on the WoE bins distribution in Figure 4.12 and Figure 4.13, respectively, we can observe negative WoE values for bins where the debt–to–income ratio is extremely high or is missing, and regarding the number of delinquent credit lines, we can observe a negative WoE value for bin corresponding to the relatively high number of delinquent credit lines. Therefore, if the loan applicant has either a high or missing debt–to–income ratio and/or a relatively high number of delinquent credit lines, he would be likely to default.

5.2 Comparison with other HMEQ Studies

Within the literature review in Chapter 3, we discussed the studies of Aras (Aras 2021) and Zurada (Zurada *et al.* 2014) which analyzed the HMEQ data set, i.e., the same data set as in this thesis. In this section, we contrast their results with those of this thesis. Particularly, we compare the models' rankings from Table 3.2 and Table 3.4, respectively, with the thesis' author's rankings derived from Table 5.2, which is summarized in the following Table 5.3. Upon examination, it is apparent that none of the studies considered the Gradient Boosting model, which happens to be the best-performing model in this thesis.

With respect to the study of Aras (Aras 2021), when excluding our Gradient Boosting model, we can agree that Random Forest and KNN are the top-performing models. However, there is a discrepancy in the order of their performances. Aras found that KNN outperformed Random Forest, whereas in this thesis, Random Forest outperformed KNN. Furthermore, it is worth noting that the worst-performing models, namely Decision Tree, Logistic Regression, and Gaussian Naive Bayes, aligned with both this thesis and Aras' study. Specifically, Aras identified Logistic Regression as the worst-performing model, while in this thesis, Gaussian Naive Bayes exhibited the poorest performance.

Regarding Zurada's study (Zurada *et al.* 2014), we again concur that Logistic Regression is one of the worst performing models. However, there are significant disparities in the rankings of other models compared to the results obtained in this thesis. It is evident from the findings that MLP was identified as the best performing model in Zurada's study, whereas in this thesis, MLP was ranked as the 5th best performing model, placing it in the second half of the ranked models. In contrast, while KNN was ranked as the 3rd best performing model in this thesis, it received the position of the 2nd worst-performing model according to Zurada. Conversely, Zurada ranked Decision Tree as the second best-performing model, while in this thesis, it was positioned as the 6th best performing model. However, both Zurada's and Aras' studies, consistent with this thesis, ranked the SVM model in the middle range, aligning with our findings.

Table 5.3: Ranking Results Comparison based on HMEQ Data Set

Model	This Thesis	(Aras 2021)	(Zurada *et al.* 2014)
GB	1	-	-
RF	2	2	-
KNN	3	1	4
SVM	4	3	3
MLP	5	-	1
DT	6	4	2
LR	7	6	5
GNB	8	5	-

Source: Author's results in Python, Rankings of (Aras 2021), (Zurada *et al.* 2014)

It is important to note that the author of the thesis, Aras, and Zurada employed distinct data preprocessing techniques and different data splits, leading to variations in the training, tuning, and evaluation samples, resulting in disparate outcomes. While Zurada and Aras utilized Grid Search with cross-validation for hyperparameter tuning, the author of the thesis employed Bayesian Optimization with stratified cross-validation. Besides, each author used different hyperparameters' spaces.

Additionally, each author utilized different evaluation metrics. Aras utilized Accuracy, Recall, Precision, F1, and Matthews Correlation Coefficient, while Zurada employed Accuracy, Recall, and AUC. In contrast, the author of the thesis employed a broader range of metrics, including Accuracy, Recall, Precision, F1, Matthews Correlation Coefficient, AUC, Kolmogorov-Smirnov Distance, Somers' D, Brier Score Loss, and Log Loss. When ranking the models, Aras' and Zurada's results were explicitly ranked based on the simple average of individual ranks (i.e., uniform weights). While the author of this thesis ranked the models based on predetermined weights set by himself. Moreover, it is worth mentioning that the authors may have used different software tools. The thesis' author utilized Python, while Zurada employed Weka, and the software used by Aras is unknown. These various factors contribute to the discrepancies in the obtained results among the studies and the thesis. Therefore, it is advisable to refrain from making direct comparisons due to the incomparability of the results.

5.3 Key Findings

The key findings of this thesis are summarized in the following list. We exclude such findings that are derived from the hypotheses' testing in Section 5.1, namely that the most important features for predicting default are the number of delinquent credit lines `DELINQ` and the debt–to–income ratio `DEBTINC`, and that the Gradient Boosting model outperforms all other models in terms of the majority of evaluation metrics, etc.

- ADASYN oversampling generates more synthetic default–case instances that are managers according to `JOB` feature as described in depicted in Table 4.8 and Figure 4.11. This is attributed to ADASYN nature as it generates more synthetic instances for such default–case instances, which are hard-to–learn, therefore, default clients who are managers are hard–to–learn according to ADASYN and its underlying KNN algorithm as described in Subsection 4.3.1. By generating more synthetic instances for instances that are hard–to–learn, ADASYN oversampling is expected to improve the model's performance.
- We can also observe the impact of missing values within default prediction, particularly in the case of debt–to–income ratio `DEBTINC`. As can be seen in Figure 4.12, the WoE corresponding to the bin capturing missing values has a negative value of (approx. -2), which indicates a larger distribution of defaulters compared to non–defaulters. Referring to the WoE values, the SHAP summary plot (Figure 4.33) depicts the global model explainability and the contribution of each feature to the predictions. As can be seen in case of `DEBTINC`, the blue values (i.e., negative WoE values) contribute positively to the default, hence, when the debt–to–income ratio is missing, the model is more likely to predict default for such instance. This also confirms our finding within exploration analysis, particularly in the association analysis where we inspected the association between the default and the missing values as presented in Table 4.6.
- Another finding regards the features `DEBTINC` and `DELINQ`. The higher the debt–to–income ratio and/or the higher the number of delinquent credit lines, the higher the probability of default. This is evident from the WoE values in Figure 4.12 and Figure 4.13, respectively, and the SHAP summary plot in Figure 4.33. Therefore, we confirm our findings from

the exploration analysis, where we assessed association analysis with the default in Table 4.4.

- Most of the models are conservative, as most of their optimal thresholds obtained using the Youden index are below the standard classification threshold of 0.5, as depicted in Figure 4.25.
- Ensemble models, such as Gradient Boosting and Random Forest, perform better than SVM or Neural Network (MLP), while transparent models such as Decision Tree, Logistic Regression, and Gaussian Naive Bayes perform the worst. This is depicted in Figure 4.22.
- Black–box models which are complex, tend to select more features than transparent white box–models within feature selection (Figure 4.16).

5.4 Contributions

The primary contribution of the thesis author involves the implementation of a customized machine learning framework on a credit risk dataset consisting of performing and non-performing US home equity loans, as described in Figure 4.1 from high–level point of view. This framework encompasses several key steps.

Firstly, the author conducts data exploration (Section 4.2), including distribution and association analysis. To address the imbalanced distribution of defaults, an advanced approach utilizing ADASYN oversampling is employed (Subsection 4.3.1). Additionally, a preprocessing step utilizing Optimal Binning is implemented to discretize feature values into bins optimized with respect to the default status, while also capturing the impact of extreme and missing values by converting them into Weight-of-Evidence values (Subsection 4.3.2).

The author then utilizes custom wrapped algorithms for both feature selection and model selection. These algorithms incorporate Bayesian Optimization for hyperparameter tuning based on the chosen classification models (Subsection 4.4.1). Within the feature selection, algorithm, each model is tuned and used within Forward Sequential Feature Selection to determine the subset of optimal features (Subsection 4.4.2). Within the model selection algorithm, each model is tuned and trained on subsets of selected features, determining an optimal threshold for classification using the Youden index, and subsequently evaluated on a validation set using a variety of metrics (Subsection 4.4.3). The

models are ranked based on these metrics, considering explicit weights defined by the author. The best performing model is selected for final evaluation on the test set and for deployment.

To enhance the model's performance and generalization ability, the author performs recalibration of the model and threshold prior to evaluation, by joining the training and validation sets into one set used for model recalibration (Subsection 4.4.4). The evaluation process encompasses various approaches, including the assessment of the model's performance using a confusion matrix, evaluation metric scores and losses, and ROC curve analysis (Subsection 4.5.1). The impact of recalibration on the evaluation is also examined. Additionally, the author assesses the explainability of the black-box model by analyzing feature importances and SHAP values (Subsection 4.5.2).

Another significant contribution involves deploying the model as a web application using Flask and HTML (Section 4.6). The deployed application is recalibrated alongside its threshold on the joined training, validation, and test sets to enhance model's ability to generalize. Such application is temporarily hosted and deployed on PythonAnywhere cloud platform and made publicly accessible. Users are required to complete a loan application form, with the fields corresponding to the features on which the model was trained. After submission, the application provides a result and default prediction, accompanied by explainability insights using LIME, which highlights the contribution of each feature to the prediction (Subsection 4.6.2).

5.5 Recommendations

Despite the complexity of the custom machine learning solution applied in this thesis, and the hundreds of hours invested by the author, it still has its limitations and drawbacks. We aim to address these issues in this section and provide suggestions for future studies.

1. **Use more relevant and actual data** - Given the nature of the data set analyzed in this thesis, it might not be the most relevant for the current market situation. Thus, using up-to-date data would either lead to a more realistic generalization (Kumar *et al.* 2021) or an improvement in the model's performance (Karatas *et al.* 2020).
2. **Use bigger data** - Since the real bank data sets often contain hundreds

of thousands or even millions of instances, and the number of features can be in the hundreds or even higher, it would be beneficial to use bigger data sets to train the models on. Therefore, by having a larger sample size for the training, we can enhance the model's performance (Ng *et al.* 2020), as the model would be able to learn more complex patterns in the data.

3. **Use behavioral scoring data** - The data set used in this thesis is based on application data only, however, behavioral scoring data is often used in real–world applications as well. Therefore, it would also be beneficial to use behavioral scoring data, which would dynamically capture the client's information about his behavior and help the bank make more informed decisions about how to deal with the existing clients (Li & Zhong 2012).
4. **Use macroeconomic data** - Using macroeconomic data such as GDP, unemployment rate, inflation rate, interest rate, etc. would allow to create macroeconomic forecasts, i.e., to take into account possible changes in the macroeconomic changes by incorporating forward–looking information (Jakubik *et al.* 2007). Thanks to that, one would be able to achieve more reliable estimates by dynamically capturing the current economic state.
5. **Use TensorFlow/Keras or PyTorch for NN development** - In this thesis, we used Neural Network (Multi–Layer Perceptron) from `Scikit-learn` module. However, in ML engineering or deep learning, the Neural Networks are mostly developed using TensorFlow/Keras or Keras modules, which are more efficient and provide more flexibility (Gevorkyan *et al.* 2019).
6. **Django for ML deployment** - In this thesis, the Flask framework was used for ML deployment as a web application. Although Flask is a great tool for ML deployment when it comes to small projects or micro-framework solutions, it is not suitable for production environment. Therefore, we recommend using the Django framework for ML deployment, which is a full–stack Python–based web application framework and is more suitable for production environment and creating larger and more complex database–backed websites and applications (Khatri & Jonhs 2023).
7. **ML application in other credit risk modelling components** - Besides applying ML prediction models in credit risk for default prediction

(i.e., PD), it would also be beneficial to apply ML models in other credit risk modelling components, such as LGD and EAD, or subsequently in ECL as well (Munkhdalai *et al.* 2019; Grzybowska & Karwanski 2020) or even for macroeconomic forecasting (Hall 2018).

8. **Hyperparameter Optimization with Optuna** - In this thesis, we used Bayesian Optimization from `Scikit-optimize` module for hyperparameter optimization. However, there is a more efficient framework, namely the Optuna module, which is deemed the next–generation hyperparameter optimization software, which allows to construct the hyperparameter search space dynamically, efficient implementation of searching and pruning strategies, and also provides a versatile architecture that can be deployed in scalable distributed computing or light–weight experiments conducted through an interactive interface (Akiba *et al.* 2019).

9. **H2O ML module** - H2O is an automated machine learning module that is more efficient than `Scikit-learn` and provides more flexibility in terms of faster scoring capabilities or producing high quality models suitable for deployment in an enterprise environment (LeDell & Poirier 2020). According to H2O's documentation, it provides wrapper functions that perform a large number of modelling-related tasks that would typically require many lines of code, and it can also be used for automating the machine learning workflow, including the automatic training and tuning of many models (H2O.ai 2023).

Chapter 6

Conclusion

The main contribution of this thesis is to develop a custom machine learning implementation framework using Python which is further applied to the application scoring data set of US home equity loans (HMEQ). In this thesis, we employ 8 classification models: Logistic Regression, Decision Tree, Gaussian Naive Bayes, K–Nearest Neighbors, Random Forest, Gradient Boosting, Support Vector Machine, and Neural Network. As evaluation metrics, we use F1, Precision, Recall, Accuracy, Matthews Correlation Coefficient, AUC, Kolmogorov–Smirnov Distance, Somers' D, Brier Score Loss and Log Loss. Prior the machine learning implementation, we cover theoretical background of both credit risk and machine learning (Chapter 2), and we also propose five hypotheses (Chapter 3), including the literature review with the main focus on HMEQ–based studies of Aras (Aras 2021) and Zurada (Zurada *et al.* 2014).

Within the empirical analysis (Chapter 4) we conduct machine learning implementation on HMEQ data set, which includes data exploration, data preprocessing, hyperparameter tuning, feature selection, model selection, model recalibration, model evaluation and final machine learning deployment. In the data exploration (Section 4.2), we inspect the distributions of the variables and their associations. Within the data preprocessing phase (Section 4.3), we perform 3 operations: (1) Stratified split into training, validation and test set with 70:15:15 ratio, (2) ADASYN oversampling for balancing default distribution, and (3) Optimal Binning of features and Weight–of–Evidence transformation.

In the ML modelling (Section 4.4), we establish hyperparameter tuning process using Bayesian Optimization with 50 iterations, stratified 10–fold cross validation while maximizing F1 score (Subsection 4.4.1). Such process enters

into 2 following steps: feature selection and model selection. In the former step (Subsection 4.4.2), each model is tuned with Bayesian Optimization and is further used within Forward Sequential Feature Selection, which returns a subset of optimal features. The features are selected while maximizing F1 score using the stratified 10–fold cross validation. Since we have 8 models, we get 8 subsets of features. In the latter step (Subsection 4.4.3), each model is tuned on each subset of selected features, and then evaluated on the validation set by computing a range of metrics mentioned above. Instead of using standard classification threshold 0.5, we calculate an optimal threshold using Youden index. Since we have 8 models and 8 subsets of features, we obtain 64 models.

Each metric is used to rank all the models, and the rank score is then calculated as a weighted average of the individual rankings, with the author's explicit weights. We calculate the final rank based on the rank score. The final model chosen for evaluation and deployment is the one with the highest rank (rank of 1). The final model is **Gradient Boosting** which was trained on the features selected by **Neural Network**. Such final model and its threshold are recalibrated on the joined training and validation set prior the evaluation (Subsection 4.4.4). In the evaluation part (Section 4.5), we assess the model's performance on test set, inspect the impact of the recalibration on the model's performance and delve into black–box model explainability using feature importances and SHAP values. It is evident that debt–to–income ratio and number of delinquent credit lines are the main default drivers.

Prior the deployment of the model into a production (Section 4.6), the final model and its optimal threshold are recalibrated on the whole data set in order to maximize the predictive power of the model. Particularly, we develop a web application using Flask and HTML, which is temporarily deployed on PythonAnywhere cloud platform and available via the following link: `http://ml-credit-risk-app-petrngn.pythonanywhere.com/`. Such application requires to fill in the loan application form (Figure 4.34) and once the form is submitted, the application processes the inputs and returns (1) loan approval result (2), predicted probability of default and (3) local black–box model explainability of the prediction using LIME (Figure 4.35).

Lastly, we summarize our results (Chapter 5), particularly we assess the hypotheses' testing in conduct a comparison with the HMEQ–based studies, outline key findings and main contributions, and propose several policy recommendations for feature research.

Bibliography

Adeodato, P. & S. Melo (2016): "On the equivalence between Kolmogorov-Smirnov and ROC curve metrics for binary classification." .

Akiba, T., S. Sano, T. Yanase, T. Ohta, & M. Koyama (2019): "Optuna: A Next-generation Hyperparameter Optimization Framework." *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* pp. 2623–2631.

Aniceto, M. C., F. Barboza, & H. Kimura (2020): "Machine learning predictivity applied to consumer creditworthiness." *Future Business Journal* **6**: pp. 1–14.

Aras, S. (2021): "Bagging and Boosting Classifiers for Credit Risk Evaluation." *Reflection of Economic Development* pp. 121–150.

Ayyadevara, V. K. (2018): *Pro Machine Learning Algorithms: A Hands-On Approach to Implementing Algorithms in Python and R.* Apress.

Baesens, B., D. Roesch, & H. Scheule (2016): *Credit risk analytics: Measurement techniques, applications, and examples in SAS.* John Wiley & Sons.

Bathaee, Y. (2018): "The Artificial Intelligence Black Box and the Failure of Intent and Causation." *Harvard Journal of Law & Technology* **31**: p. 889.

Beerbaum, D. (2015): "Significant increase in credit risk according to IFRS 9: Implications for financial institutions." *International Journal of Economics & Management Sciences* **04(09)**.

Bellini, T. (2019): *IFRS 9 and CECL Credit Risk Modelling and Validation: A Practical Guide with Examples Worked in R and SAS.* Academic Press.

Bera, D., R. Pratap, & B. D. Verma (2021): "Dimensionality reduction for categorical data." *IEEE Transactions on Knowledge and Data Engineering* .

Berry, M. W., A. Mohamed, & B. W. Yap (editors) (2020): *Supervised and Unsupervised Learning for Data Science*. Springer Nature, 1st edition.

Bessis, J. (2015): *Risk Management in Banking*. John Wiley & Sons, 4th edition.

Bhattacharya, A. (2022): *Applied Machine Learning Explainability Techniques: Make ML models explainable and trustworthy for practical applications using LIME, SHAP, and more*. Packt Publishing Ltd.

Bischl, B., M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix *et al.* (2023): "Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* **13(2)**: p. e1484.

Bolón-Canedo, V., N. Sánchez-Maroño, & A. Alonso-Betanzos (2015): *Feature selection for high-dimensional data*. Springer.

Bonaccorso, G. (2020): *Mastering Machine Learning Algorithms: Expert techniques for implementing popular machine learning algorithms, fine-tuning your models, and understanding how they work*. Packt Publishing Ltd, 2nd edition.

Boughorbel, S., F. Jarray, & M. El-Anbari (2017): "Optimal classifier for imbalanced data using Matthews Correlation Coefficient metric." *PLOS ONE* **12(6)**: pp. 1–17.

Brabec, J., T. Komárek, V. Franc, & L. Machlica (2020): "On Model Evaluation Under Non-Constant Class Imbalance." *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part IV 20* pp. 74–87.

Brezigar-Masten, A., I. Masten, & M. Volk (2021): "Modeling credit risk with a Tobit model of days past due." *Journal of Banking & Finance* **122**.

Brownlee, J. (2020): "Recursive feature elimination (RFE) for feature selection in Python." MachineLearningMastery.com. Accessed on: April 28, 2023. Available at: `https://machinelearningmastery.com/rfe-feature-selection-in-python/`.

Brownlee, J. (2021): "Failure of Classification Accuracy for Imbalanced Class Distributions." MachineLearningMastery.com. Accessed on: April 30, 2023. Available at: https://machinelearningmastery.com/failure-of-accuracy-for-imbalanced-class-distributions/.

Charu, C. A. (2018): *Neural Networks and Deep Learning: A Textbook*. Spinger.

Chicco, D. & G. Jurman (2020): "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation." *BMC genomics* **21**.

Cichosz, P. (2015): *Data Mining Algorithms: Explained Using R*. John Wiley & Sons.

Comotto, F. (2022): "Evaluation Metrics: Leave Your Comfort Zone and Try MCC and Brier Score." TowardsDataScience.com. Accessed on: April 30, 2023. Available at: https://towardsdatascience.com/evaluation-metrics-leave-your-comfort-zone-and-try-mcc-and-brier-score-86307fb1236a.

Dembla, G. (2020): "Intuition behind Log-Loss score." TowardsDataScience.com. Accessed on: April 30, 2023. Available at: https://towardsdatascience.com/intuition-behind-log-loss-score-4e0c9979680a.

Dias, P., S. Experian, B. Melissa Forti, M. Witarsa, & S. Experian (2018): "A comparison of Gradient Boosting with Logistic Regression in Practical Cases."

Dilmegani, C. (2022): "Dark side of neural networks explained [2023]." AIMultiple.com. Accessed on: April 30, 2023. Available at: https://research.aimultiple.com/how-neural-networks-work/.

Doumpos, M., C. Lemonakis, D. Niklis, & C. Zopounidis (2019): *Analytical Techniques in the Assessment of Credit Risk: An Overview of Methodologies and Applications*. Springer.

Drahokoupil, J. (2022): "Application of the XGBoost algorithm and Bayesian optimization for the Bitcoin price prediction during the COVID-19 period." *FFA Working Papers* **4**: p. Article 2022.006.

Esposito, C., G. A. Landrum, N. Schneider, N. Stiefl, & S. Riniker (2021): "GHOST: adjusting the decision threshold to handle imbalanced data in machine learning." *Journal of Chemical Information and Modeling* **61(6)**: pp. 2623–2640.

Fawcett, T. (2006): "An introduction to ROC analysis." *Pattern Recognition Letters* **27(8)**: pp. 861–874.

Fluss, R., D. Faraggi, & B. Reiser (2005): "Estimation of the Youden Index and its associated cutoff point." *Biometrical Journal: Journal of Mathematical Methods in Biosciences* **47(4)**: pp. 458–472.

Forsyth, D. (2019): *Applied Machine Learning*. Springer.

Gauhar, N. (2020): "Decision Tree: A Classification Algorithm." LearningWithGauhar.com. Accessed on: April 30, 2023. Available at: `https://learnwithgauhar.com/decision-tree-a-classification-algorithm/`.

Gevorkyan, M. N., A. V. Demidova, T. S. Demidova, & A. A. Sobolev (2019): "Review and comparative analysis of machine learning libraries for machine learning." *Discrete and Continuous Models and Applied Computational Science* **27(4)**: pp. 305–315.

Goethals, S., D. Martens, & T. Evgeniou (2022): "The non-linear nature of the cost of comprehensibility." *Journal of Big Data* **9(1)**: p. 30.

Gornjak, M. (2017): "Comparison of IAS 39 and IFRS 9: The Analysis of Replacement." *International Journal of Management, Knowledge and Learning* **6(1)**: pp. 115–130.

Gregory, J. (2012): *Counterparty Credit Risk and Credit Value Adjustment: A Continuing Challenge for Global Financial Markets*. John Wiley & Sons, 2nd edition.

Grzybowska, U. & M. Karwanski (2020): "Application of Machine Learning Method under IFRS 9 Approach to LGD Modeling." *Acta Physica Polonica, A.* **138(1)**: pp. 116–122.

H2O.ai (2023): "AutoML: Automatic Machine Learning." AutoML: Automatic Machine Learning - H2O 3.40.0.4 documentation. Accessed on: March 28, 2023. `ttps://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html`.

Hale, J. (2019): "Don't Sweat the Solver Stuff." TowardsDataScience.com. Accessed on: May 1, 2023. Available at: `https://towardsdatascience.com/dont-sweat-the-solver-stuff-aea7cddc3451`.

Hall, A. S. (2018): "Machine Learning Approaches to Macroeconomic Forecasting." *The Federal Reserve Bank of Kansas City Economic Review* **103**: pp. 63–81.

Han, J., M. Kamber, & J. Pei (2011): *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3rd edition.

He, H., Y. Bai, E. A. Garcia, & S. Li (2008): "ADASYN: Adaptive synthetic sampling approach for imbalanced learning." *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)* pp. 1322–1328.

He, H. & Y. Ma (2013): *Imbalanced Learning: Foundations, Algorithms, and Applications*. Wiley-IEEE Press.

de Hond, A. A., I. M. Kant, M. Fornasa, G. Cinà, P. W. Elbers, P. J. Thoral, M. S. Arbous, & E. W. Steyerberg (2023): "Predicting Readmission or Death After Discharge From the ICU: External Validation and Retraining of a Machine Learning Model." *Critical Care Medicine* **51(2)**: pp. 291–300.

Hsu, C.-W. & C.-J. Lin (2002): "A comparison of methods for multiclass support vector machines." *IEEE transactions on Neural Networks* **13(2)**: pp. 415–425.

Igareta, A. (2021): "Stratified sampling: You may have been splitting your dataset all wrong." TowardsDataScience.com. Accessed on: April 17, 2023. Available at: `https://towardsdatascience.com/stratified-sampling-you-may-have-been-splitting-your-dataset-all-wrong-8cfdd0d32502`.

Jahromi, A. H. & M. Taheri (2017): "A non-parametric mixture of Gaussian naive Bayes classifiers based on local independent features." *2017 Artificial intelligence and signal processing conference (AISP)* pp. 209–212.

Jakubik, P. *et al.* (2007): "Macroeconomic environment and credit risk." *Czech Journal of Economics and Finance (Finance a uver). Charles University Prague, Faculty of Social Sciences* **57(1-2)**: pp. 60–78.

Janitza, S., C. Strobl, & A.-L. Boulesteix (2013): "An AUC-based permutation variable importance measure for random forests." *BMC bioinformatics* **14**.

Japkowicz, N. & M. Shah (2014): *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 1st edition.

Joseph, C. (2013): *Advanced Credit Risk Analysis and Management*. John Wiley & Sons.

Karatas, G., O. Demir, & O. K. Sahingoz (2020): "Increasing the Performance of Machine Learning-Based IDSs on an Imbalanced and Up-to-Date Dataset." *IEEE access* **8**: pp. 32150–32162.

Kaushik, S. (2023): "Introduction to Feature Selection methods with an example (or how to select the right variables?)." Analytics Vidhya. Accessed on: April 30, 2023. Available at: `ttps://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/`.

Kazemi, H. R., K. Khalili-Damghani, & S. Sadi-Nezhad (2022): "Estimation of optimum thresholds for binary classification using genetic algorithm: An application to solve a credit scoring problem." *Expert Systems* **40(3)**.

Khatri, V. S. & R. Jonhs (2023): "Flask vs Django: Which Python Web Framework to Use in 2023?" hackr.io. Accessed on: May 20, 2023. Available at: `https://hackr.io/blog/flask-vs-django`.

Konno, Y. & Y. Itoh (2016): "An alternative to the standardized approach for assessing credit risk under the Basel Accords." *Cogent Economics & Finance* **4(1)**.

Kornbrot, D. (2015): "Point Biserial Correlation." *Encyclopedia of Statistics in Behavioral Science* .

Kumar, R., A. A. Khan, J. Kumar, N. A. Golilarz, S. Zhang, Y. Ting, C. Zheng, W. Wang *et al.* (2021): "Blockchain-Federated-Learning and Deep Learning Models for COVID-19 detection using CT Imaging." *IEEE Sensors Journal* **21(14)**: pp. 16301–16314.

LeDell, E. & S. Poirier (2020): "H2O AutoML: Scalable Automatic Machine Learning." *7th ICML Workshop on Automated Machine Learning (2020)* .

Li, X.-L. & Y. Zhong (2012): "An Overview of Personal Credit Scoring: Techniques and Future Work." *International Journal of Intelligence Science* **2(4A)**: pp. 181–189.

Lin, H.-T., C.-J. Lin, & R. C. Weng (2007): "A note on Platt's probabilistic outputs for support vector machines." *Machine learning* **68**: pp. 267–276.

Loyola-Gonzalez, O. (2019): "Black-Box vs. White-Box: Understanding Their Advantages and Weaknesses From a Practical Point of View." *IEEE access* **7**: pp. 154096–154113.

Malley, J., J. Kruppa, A. Dasgupta, K. Malley, & A. Ziegler (2011): "Probability Machines Consistent Probability Estimation Using Nonparametric Learning Machines." *Methods of information in medicine* **51**: pp. 74–81.

Marinov, D. & D. Karapetyan (2019): "Hyperparameter Optimisation with Early Termination of Poor Performers." *2019 11th Computer Science and Electronic Engineering (CEEC)* pp. 160–163.

Meyer, D. (2009): "Support Vector Machines: The Interface to libsvm in package e1071." *FH Technikum Wien, Austria* .

Mucherino, A., P. Papajorgji, & P. M. Pardalos (2009): *Data Mining in Agriculture*. Springer.

Munkhdalai, L., T. Munkhdalai, O.-E. Namsrai, J. Y. Lee, & K. H. Ryu (2019): "An Empirical Comparison of Machine-Learning Methods on Bank Client Credit Assessments." *Sustainability* **11(3)**: p. 699.

Narkhede, S. (2018): "Understanding AUC-ROC Curve." TowardsDataScience.com. Accessed on: April 28, 2023. Available at: `https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5`.

Navas-Palencia, G. (2020): "Optimal binning: mathematical programming formulation." .

Newson, R. (2002): "Parameters behind "Nonparametric" statistics: Kendall's tau, Somers' D and Median Differences." *The Stata Journal* **2(1)**: pp. 45–64.

Newson, R. (2014): "Interpretation of Somers' D under four simple models."

Ng, W., B. Minasny, W. d. S. Mendes, & J. A. M. Demattê (2020): "The influence of training sample size on the accuracy of deep learning models for the prediction of soil properties with near-infrared spectroscopy data." *SOIL* **6(2)**: pp. 565–578.

Nian, R. (2018): "Fixing Imbalanced Datasets: An Introduction to ADASYN (with code!)." Medium.com. Accessed on: May 2, 2023. Available at: `https://medium.com/@ruinian/an-introduction-to-adasyn-with-code-1383a5ece7aa`.

Owen, L. (2022): *Hyperparameter Tuning with Python*. Packt Publishing, 1st edition. Original work published 2022.

Owusu, E., R. Quainoo, S. Mensah, & J. K. Appati (2023): "A Deep Learning Approach for Loan Default Prediction Using Imbalanced Dataset." *International Journal of Intelligent Information Technologies (IJIIT)* **19(1)**: pp. 1–16.

Park, H.-A. (2013): "An introduction to logistic regression: from basic concepts to interpretation with particular attention to nursing domain." *Journal of Korean Academy of Nursing* **43(2)**: pp. 154–164.

Patle, A. & D. S. Chouhan (2013): "SVM kernel functions for classification." *2013 International Conference on Advances in Technology and Engineering (ICATE)* pp. 1–9.

Pintelas, E., I. E. Livieris, & P. Pintelas (2020): "A Grey-Box Ensemble Model Exploiting Black-Box Accuracy and White-Box Intrinsic Interpretability." *Algorithms* **13(1)**: p. 17.

Platt, J. *et al.* (1999): "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods." *Advances in Large Margin Classifiers* .

Porretta, P., A. Letizia, & F. Santoboni (2020): "Credit risk management in bank: Impacts of IFRS 9 and Basel 3." *Risk Governance and Control: Financial Markets & Institutions* **10(2)**: pp. 29–44.

Pramoditha, R. (2021): "How to Mitigate Overfitting with Regularization." TowardsDataScience.com. Accessed on: May 1, 2023. Available at: https://towardsdatascience.com/how-to-mitigate-overfitting-with-regularization-befcf4e41865.

Prati, R. C., G. E. Batista, & M. C. Monard (2009): "Data mining with imbalanced class distributions: concepts and methods." *Indian International Conference on Artificial Intelligence (IICAI)* .

Provost, F. & T. Fawcett (2013): *Data Science for Business: What You Need to Know about Data Mining and Data-Analytic Thinking*. O'Reilly Media, 1st edition.

PwC (2023): "Risk management and modelling." PwC. Accessed on: May 20, 2023. Available at: https://www.pwc.com/cz/cs/assets/Risk-management-and-Modelling-eBook-A4.pdf.

Ribeiro, M. T., S. Singh, & C. Guestrin (2016): ""Why Should I Trust You?": Explaining the Predictions of Any Classifier." *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* pp. 1135–1144.

scikit-learn (2023a): "DecisionTreeClassifier." Scikit-learn. Accessed on: April 28, 2023. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier.

scikit-learn (2023b): "GaussianNB." Scikit-learn. Accessed on: April 28, 2023. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB.

scikit-learn (2023c): "GradientBoostingClassifier." Scikit-learn. Accessed on: April 28, 2023. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html#sklearn.ensemble.GradientBoostingClassifier.

scikit-learn (2023d): "KNeighborsClassifier." Scikit-learn. Accessed on: April 28, 2023. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.

SCIKIT-LEARN (2023e): "MLPClassifier." Scikit-learn. Accessed on: April 28, 2023. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier.

SCIKIT-LEARN (2023f): "RandomForestClassifier." Scikit-learn. Accessed on: April 28, 2023. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier.

SCIKIT-LEARN (2023g): "SequentialFeatureSelector." Scikit-learn. Accessed on: April 28, 2023. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SequentialFeatureSelector.html#sklearn.feature_selection.SequentialFeatureSelector.

SCIKIT-OPTIMIZE (2023a): "BayesSearchCV." Scikit-optimize. Accessed on: April 28, 2023. Available at: https://scikit-optimize.github.io/stable/modules/generated/skopt.BayesSearchCV.html#.

SCIKIT-OPTIMIZE (2023b): "LogisticRegression." Scikit-learn. Accessed on: April 28, 2023. Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression.

SHAKDWIPEE, P. & M. MEHTA (2017): "From Basel I to Basel II to Basel III." *International Journal of New Technology and Research (IJNTR)* **3(1)**: pp. 66–70.

SUBASI, A. (2020): *Practical Machine Learning for Data Analysis Using Python*. Academic Press.

TATSAT, H., S. PURI, & B. LOOKABAUGH (2020): *Machine Learning and Data Science Blueprints for Finance: From Building Trading Strategies to Robo-Advisors Using Python*. O'Reilly Media, 1st edition.

TEMIM, J. (2016): "The IFRS 9 impairment model and its interaction with the Basel framework." *Moody's Analytics Risk Perspectives: The Convergence of Risk, Finance, and Accounting: CECL* **8(1)**.

Teplý, P. & M. Polena (2020): "Best classification algorithms in peer-to-peer lending." *The North American Journal of Economics and Finance* **51**.

Van Gestel, T. & B. Baesens (2009): *Credit Risk Management: Basic concepts: Financial risk components, Rating analysis, models, economic and regulatory capital*. Oxford University Press.

Verma, V. (2022): "A comprehensive guide to Feature Selection using Wrapper methods in Python." Analytics Vidhya. Accessed on: April 30, 2023. Available at: https://www.analyticsvidhya.com/blog/2020/10/a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python/.

Verma, Y. (2021): "A Complete Guide to Sequential Feature Selection." Analytics India Magazine. Accessed on: April 28, 2023. Available at: https://analyticsindiamag.com/a-complete-guide-to-sequential-feature-selection/.

Wang, W. (2020): "Bayesian optimization concept explained in Layman terms." TowardsDataScience.com. Accessed on: April 29, 2023. Available at: https://towardsdatascience.com/bayesian-optimization-concept-explained-in-layman-terms-1d2bcdeaf12f.

Wendler, T. & S. Gröttrup (2021): *Data Mining with SPSS Modeler: Theory, Exercises and Solutions*. Cham: Springer, 2nd edition.

White, J. (2022): "What Does 'Derogatory' Mean on a Credit Report?" Experian. Accessed on: March 21, 2023. https://www.experian.com/blogs/ask-experian/what-the-term-derogatory-means-in-a-credit-report/.

Witten, I. H., E. Frank, M. Hall, & C. Pal (2016): *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 4th edition.

Witzany, J. (2017): *Credit Risk Management: Pricing, Measurement, and Modeling*. Springer.

Wu, Z., L. Jiang, Z. Jiang, B. Chen, K. Liu, Q. Xuan, & Y. Xiang (2018): "Accurate indoor localization based on CSI and visibility graph." *Sensors (Basel, Switzerland)* **18(8)**: p. 2549.

Zaidi, A. (2022): "Mathematical justification on the origin of the sigmoid in logistic regression." *Central European Management Journal* **30(4)**: pp. 1327–1337.

Zeng, G. (2014): "A Necessary Condition for a Good Binning Algorithm in Credit Scoring." *Applied Mathematical Sciences* **8(65)**: pp. 3229–3242.

Zurada, J., N. Kunene, & J. Guan (2014): "The classification performance of multiple methods and datasets: Cases from the loan credit scoring domain." *Journal of International Technology and Information Management* **23(1)**.

Appendix A

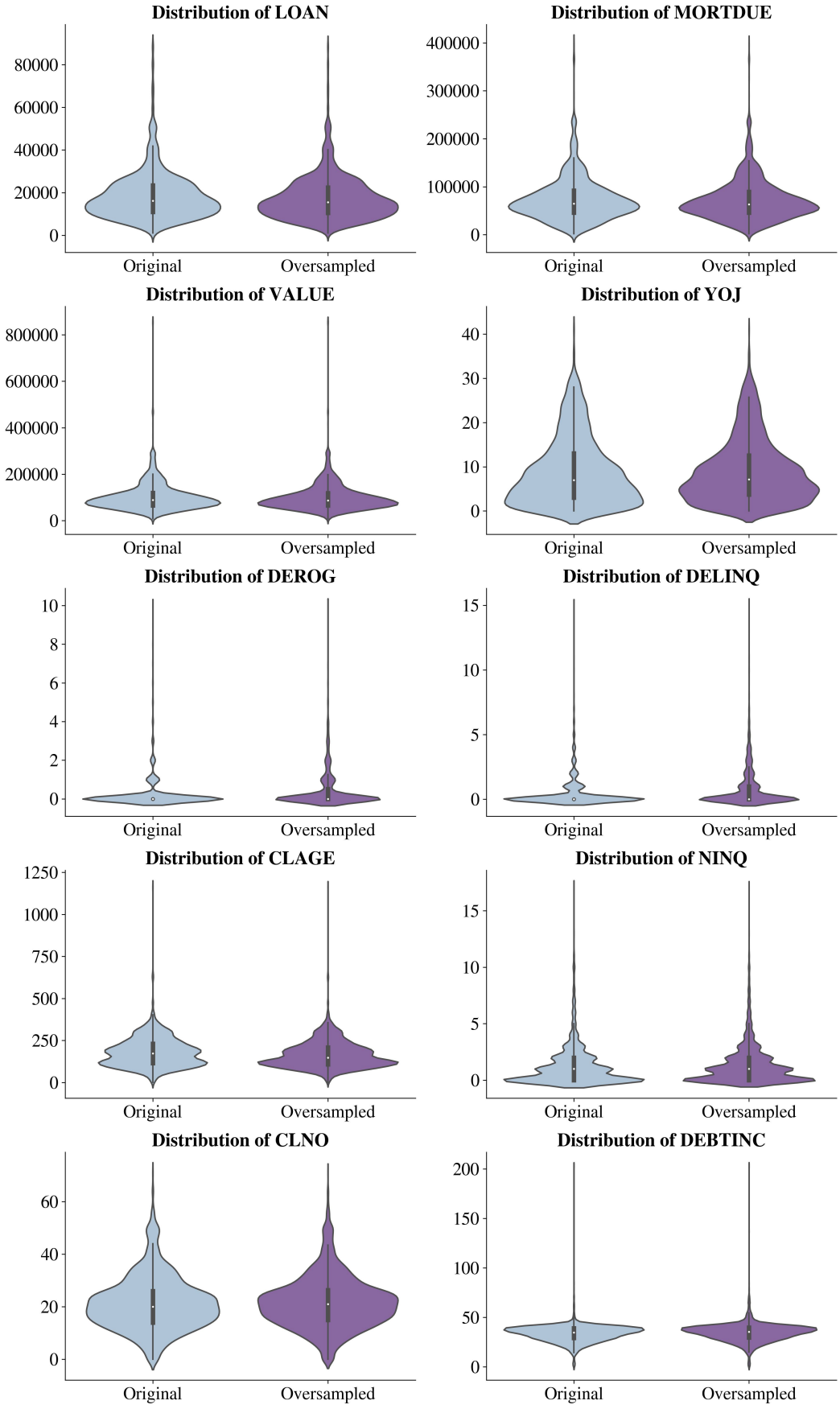
Additional Figures and Tables

Table A.1: Normality Test - Shapiro–Wilk

Feature	Shapiro-Wilk	Significance	Result
LOAN	0.8519	***	Not normally distributed
MORTDUE	0.8802	***	Not normally distributed
VALUE	0.8100	***	Not normally distributed
YOJ	0.9078	***	Not normally distributed
DEROG	0.3360	***	Not normally distributed
DELINQ	0.4578	***	Not normally distributed
CLAGE	0.9346	***	Not normally distributed
NINQ	0.6909	***	Not normally distributed
CLNO	0.9665	***	Not normally distributed
DEBTINC	0.8272	***	Not normally distributed

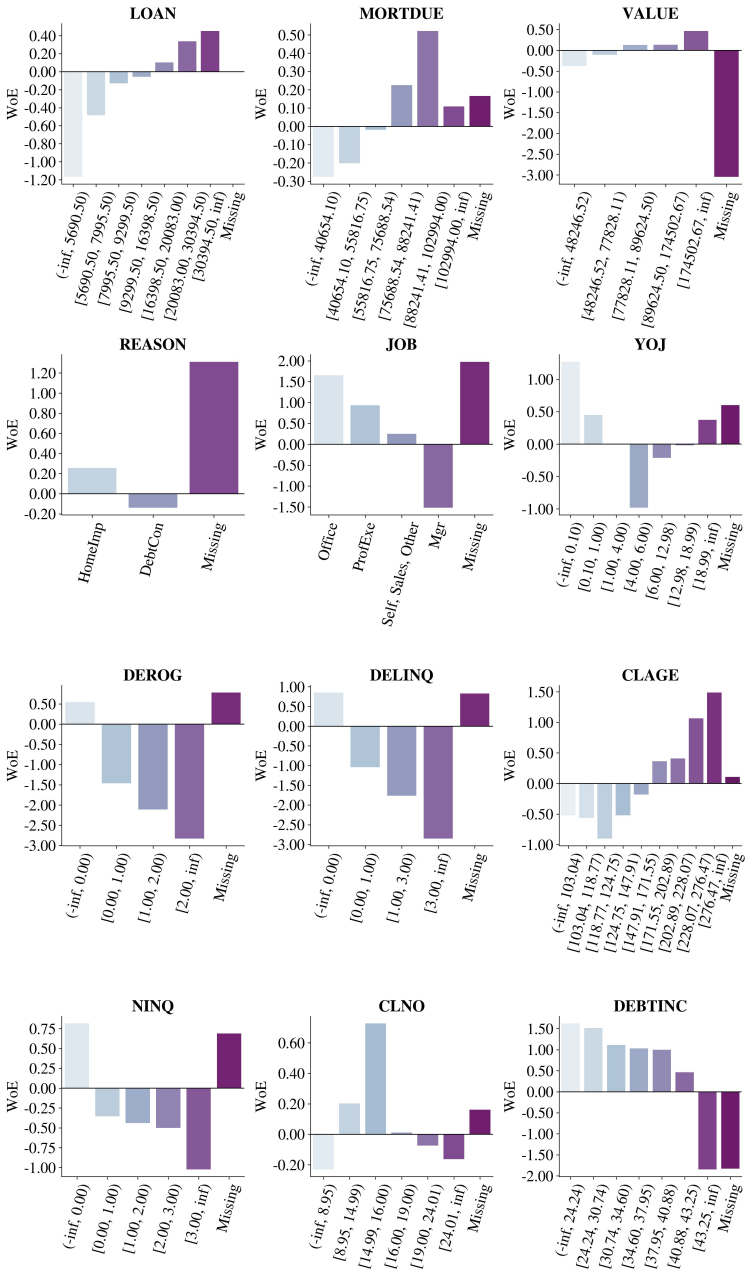
Source: Author's results in Python

Figure A.1: ADASYN Impact of Numeric Features' Distribution



Source: Author's results in Python

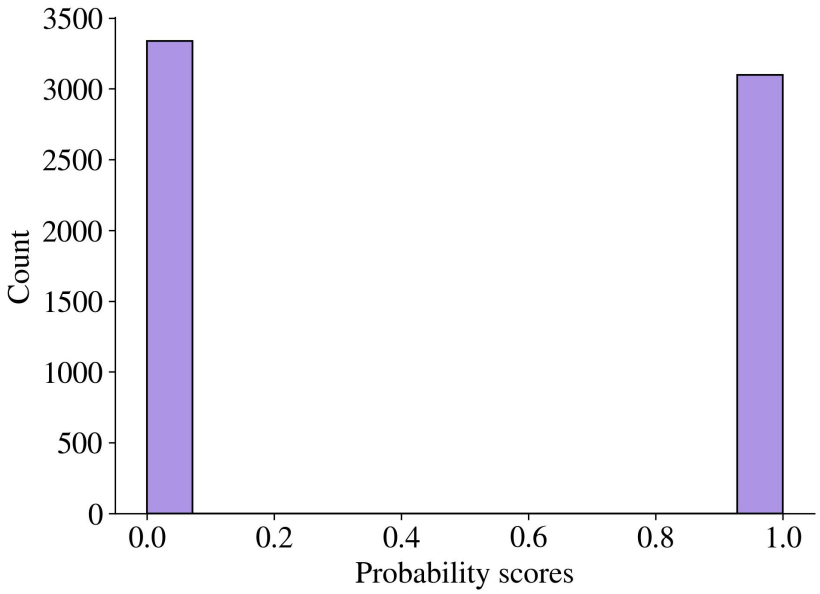
Figure A.2: WoE Bins Distribution



Source: Author's results in Python

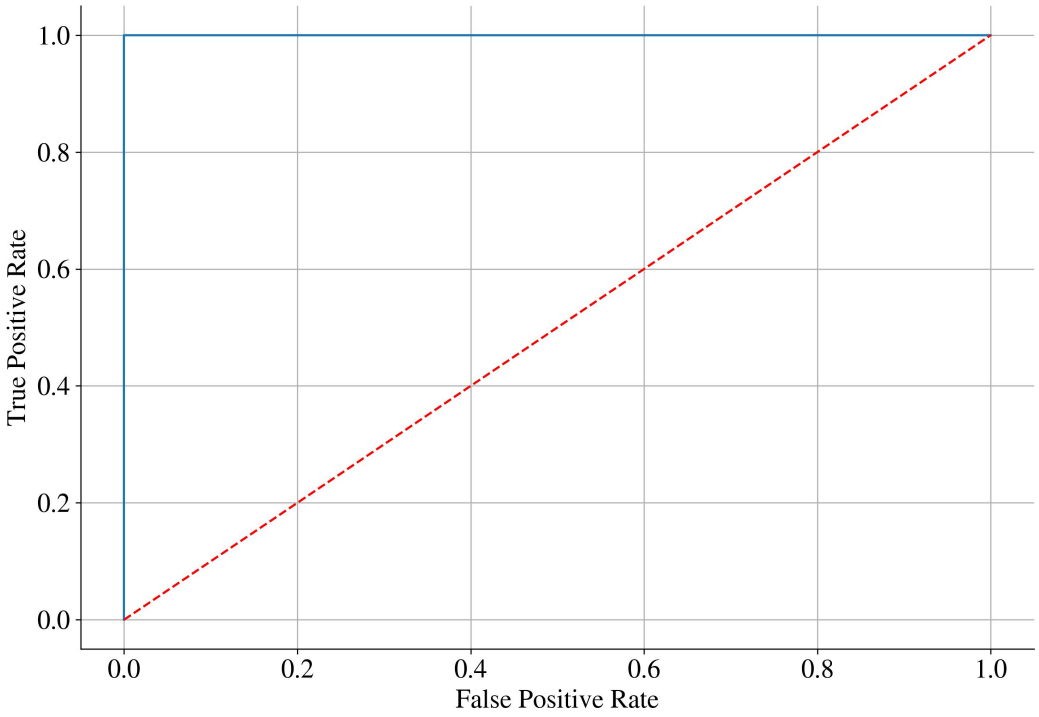
A.1 Model Selection Results

Figure A.3: Special Case of KNN - Probability Scores Distribution



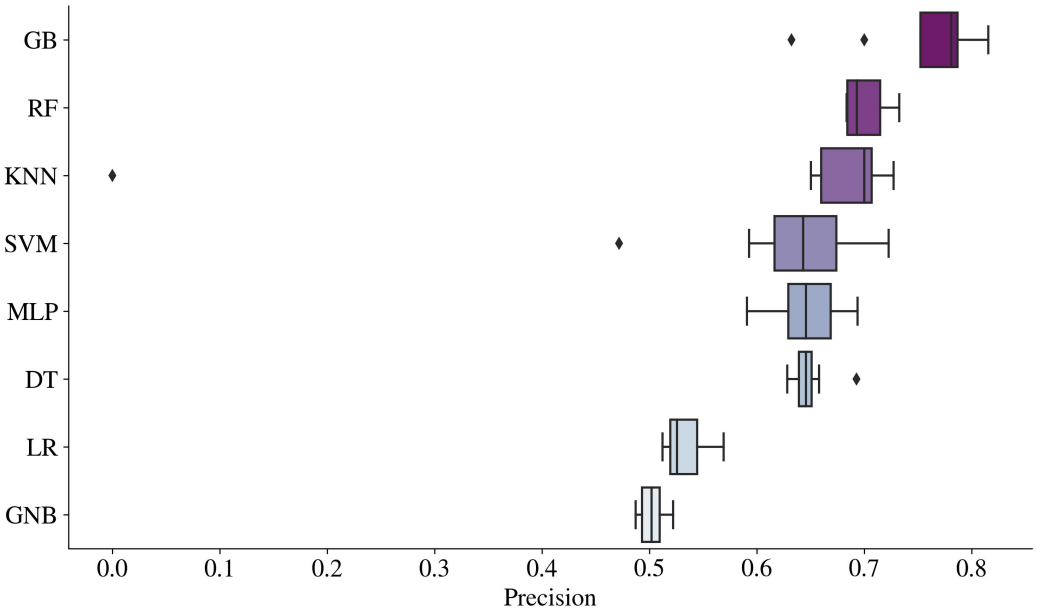
Source: Author's results in Python

Figure A.4: Special Case of KNN - ROC Curve



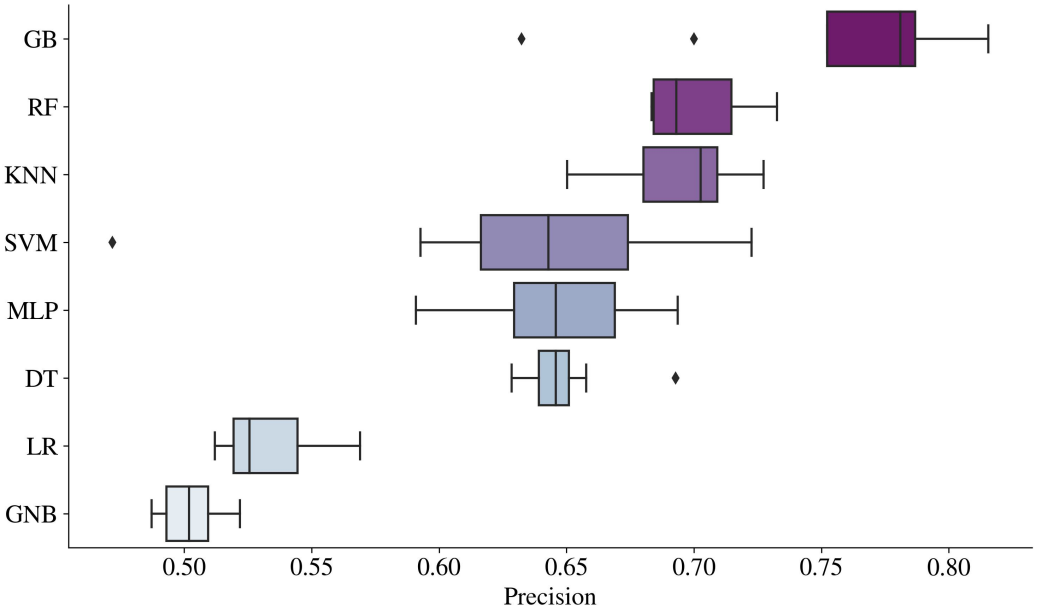
Source: Author's results in Python

Figure A.5: Precision Distribution



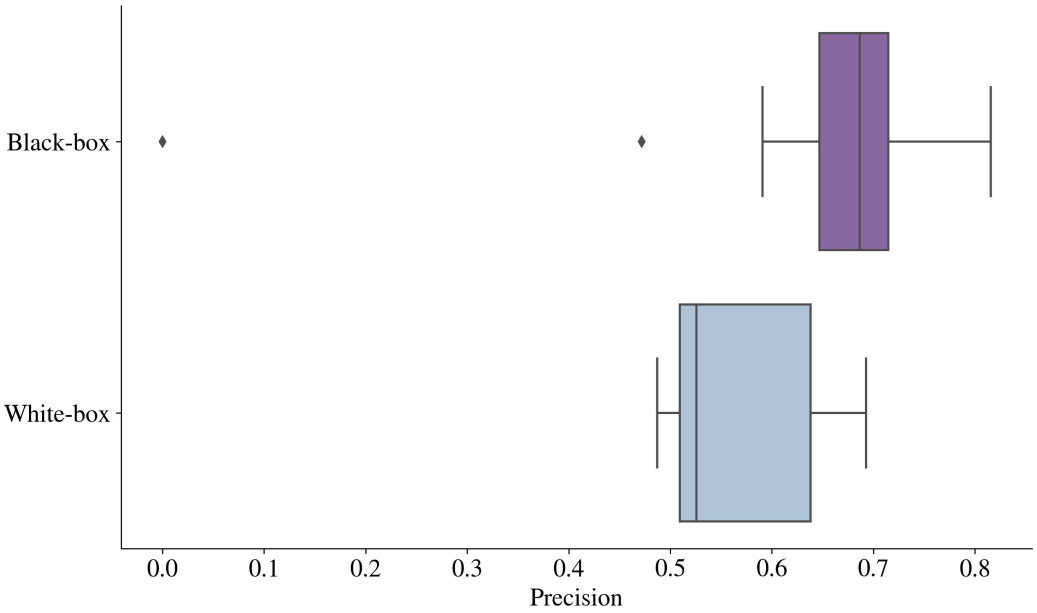
Source: Author's results in Python

Figure A.6: Precision Distribution - *without outlier*



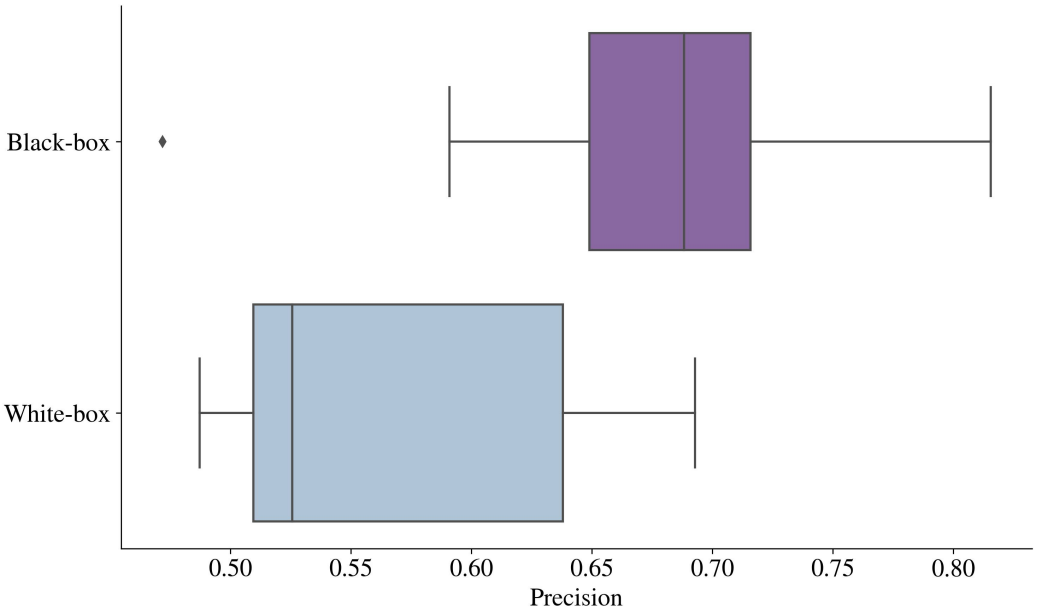
Source: Author's results in Python

Figure A.7: Precision Distribution (Black–box/White–box dimension)



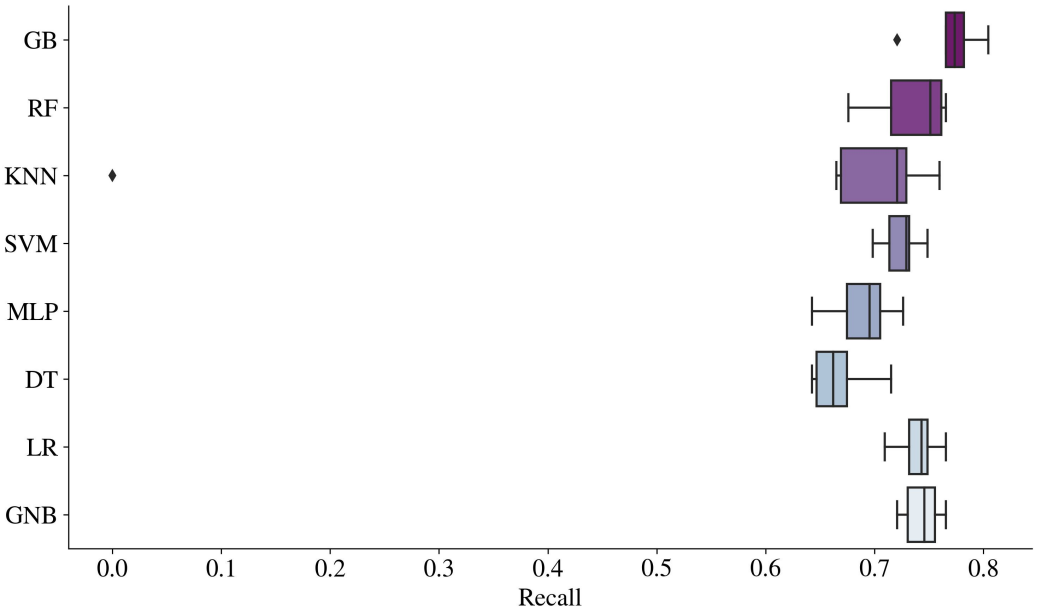
Source: Author's results in Python

Figure A.8: Precision Distribution (Black–box/White–box dimension) - *without outlier*



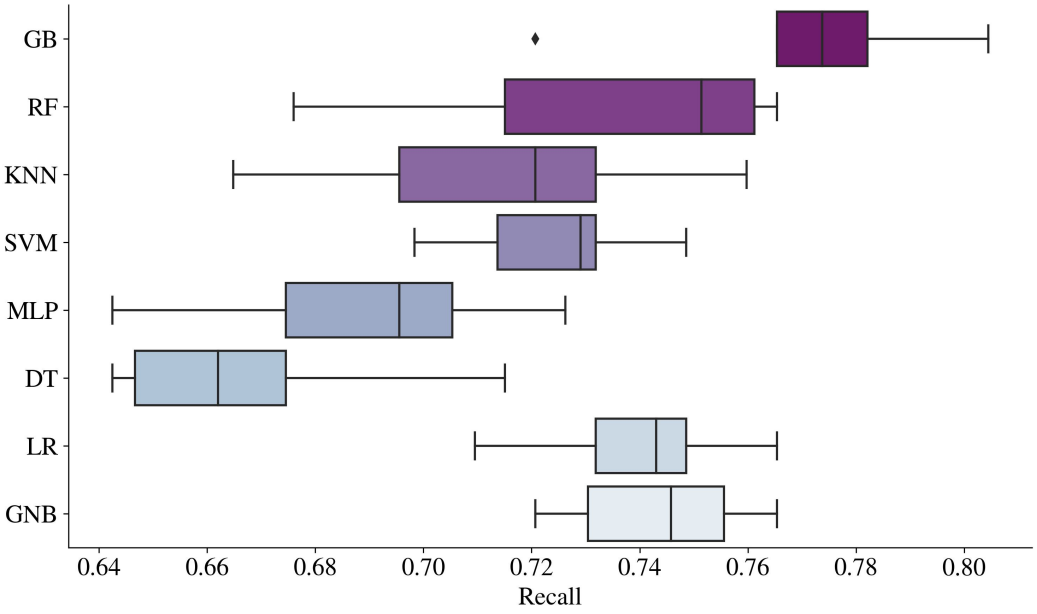
Source: Author's results in Python

Figure A.9: Recall Distribution



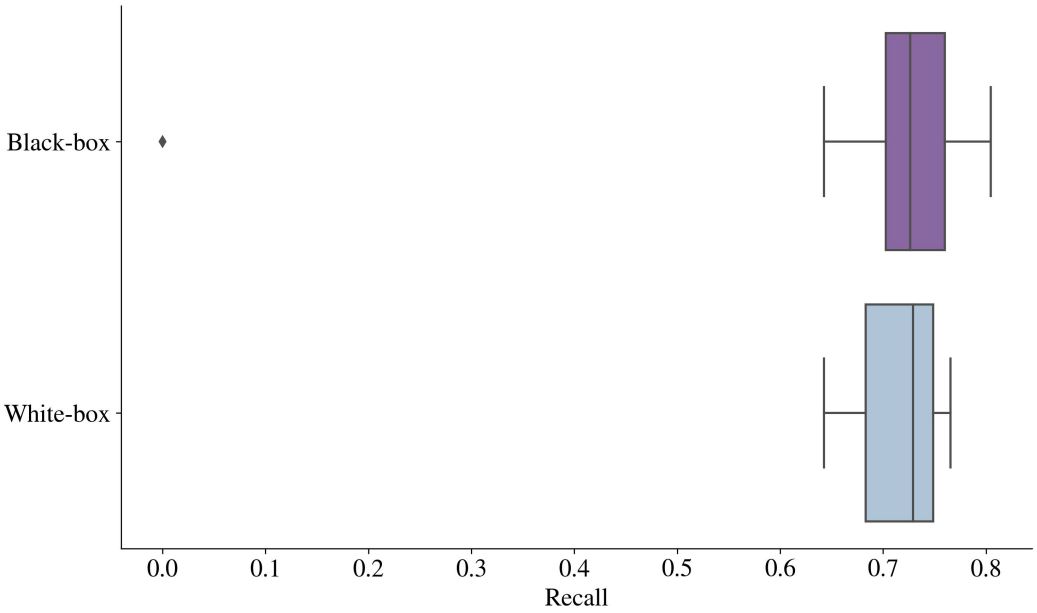
Source: Author's results in Python

Figure A.10: Recall Distribution - *without outlier*



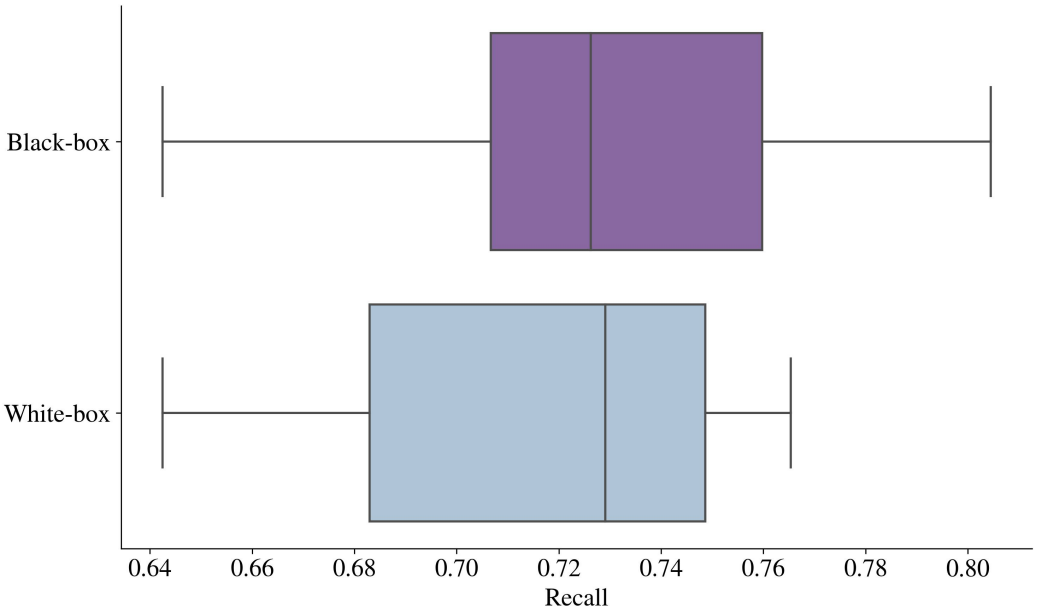
Source: Author's results in Python

Figure A.11: Recall Distribution (Black–box/White–box dimension)



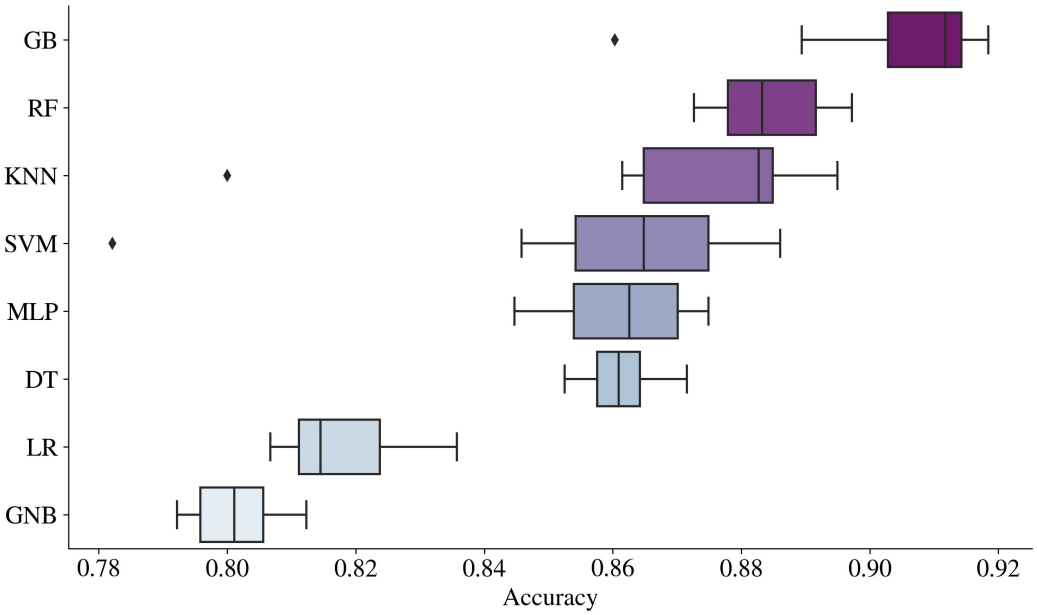
Source: Author's results in Python

Figure A.12: Recall Distribution (Black–box/White–box dimension) - *without outlier*



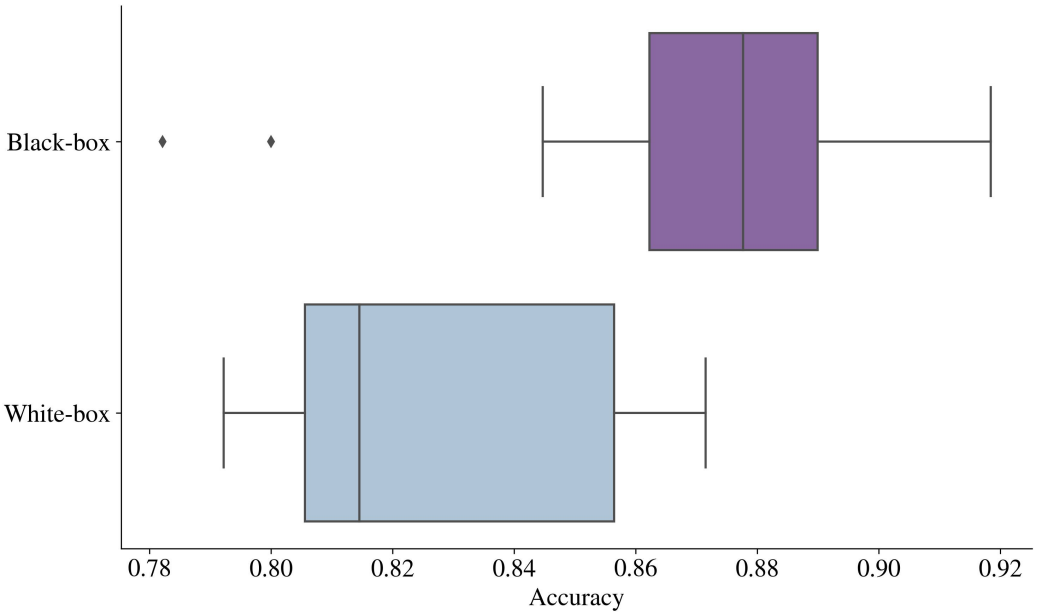
Source: Author's results in Python

Figure A.13: Accuracy Distribution



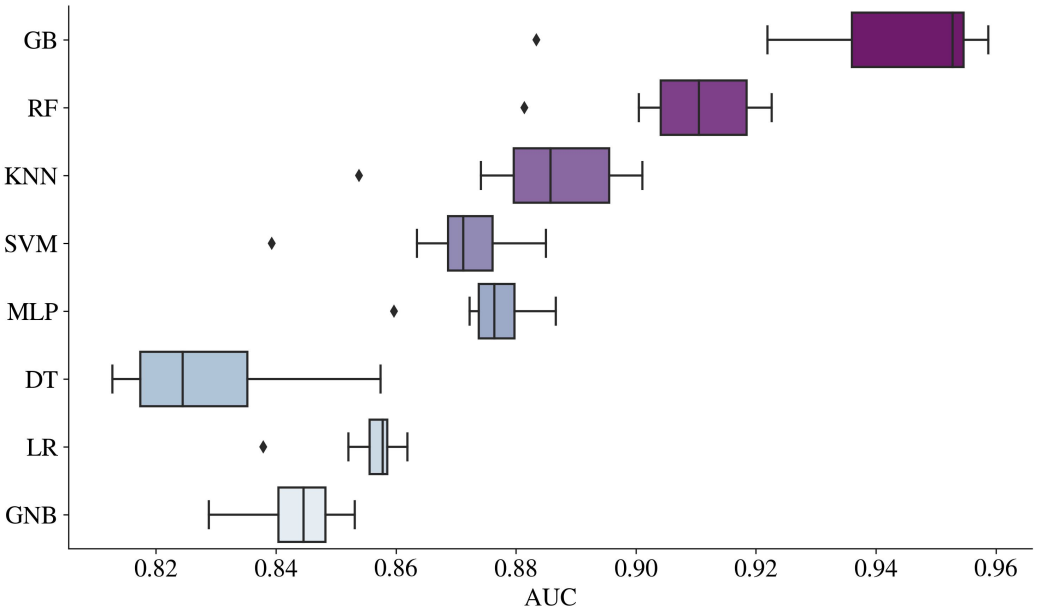
Source: Author's results in Python

Figure A.14: Accuracy Distribution (Black–box/White–box dimension)



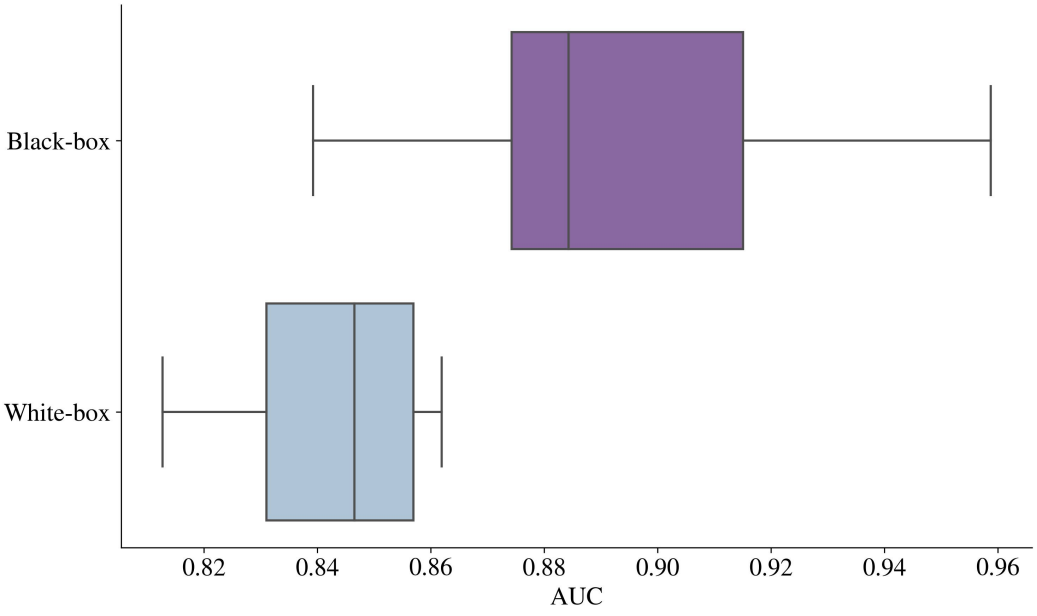
Source: Author's results in Python

Figure A.15: AUC Distribution



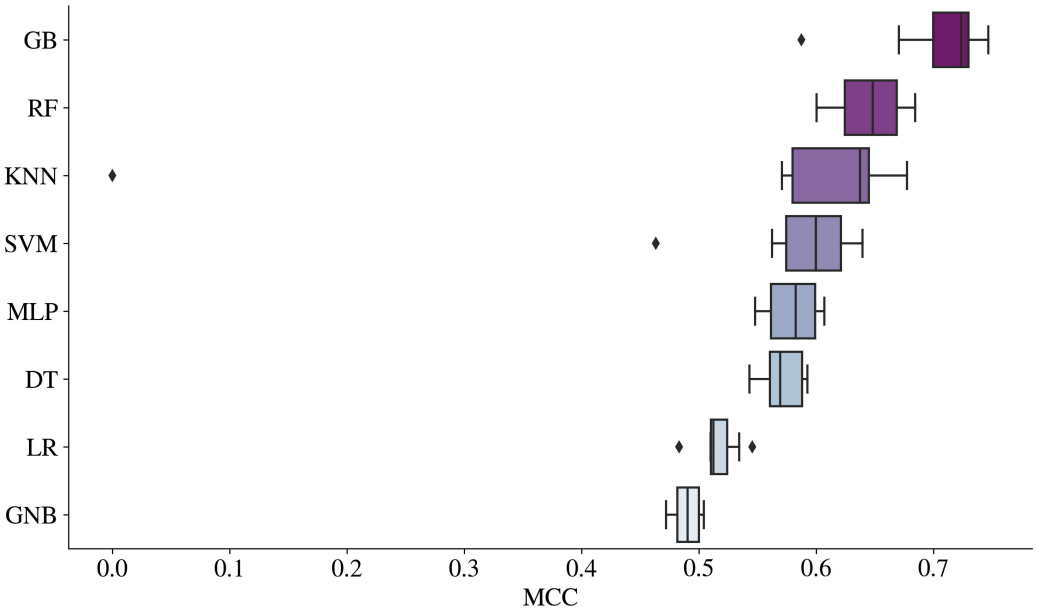
Source: Author's results in Python

Figure A.16: AUC Distribution (Black–box/White–box dimension)



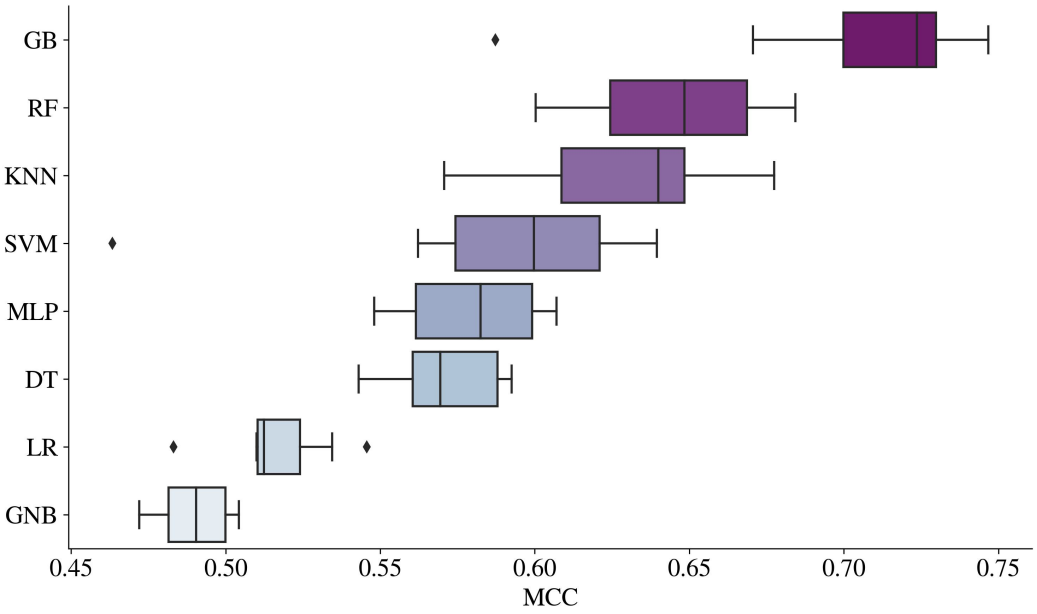
Source: Author's results in Python

Figure A.17: Matthews Correlation Coefficient Distribution



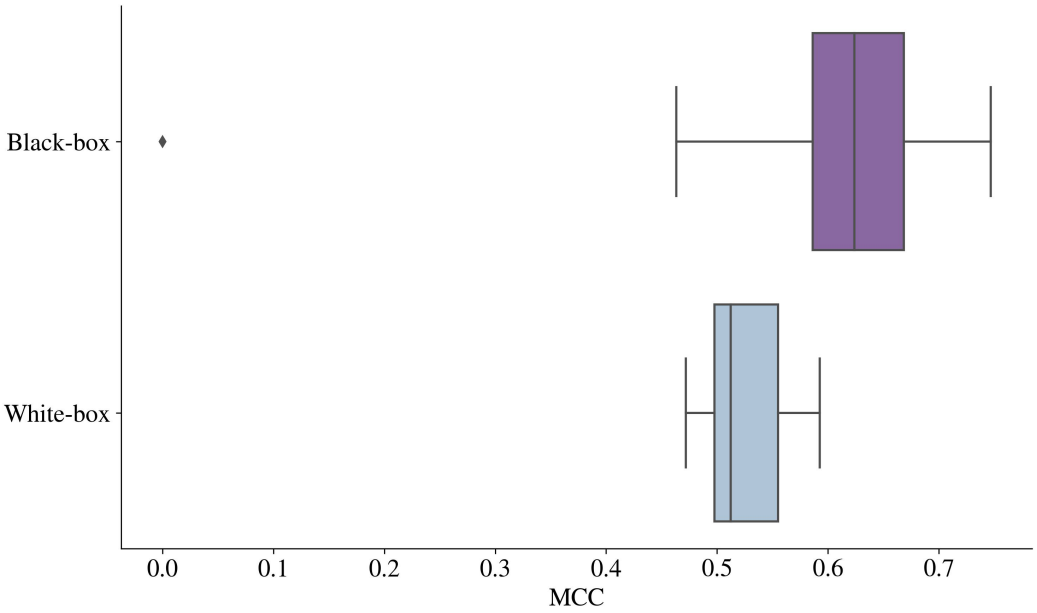
Source: Author's results in Python

Figure A.18: Matthews Correlation Coefficient Distribution - *without outlier*



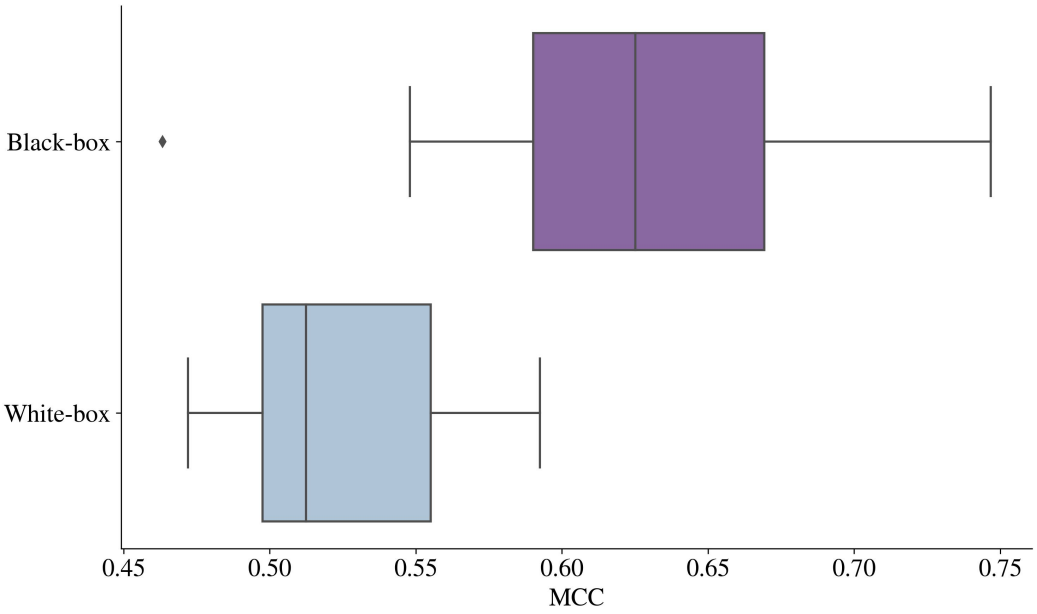
Source: Author's results in Python

Figure A.19: Matthews Correlation Coefficient Distribution (Black–box/White–box dimension)



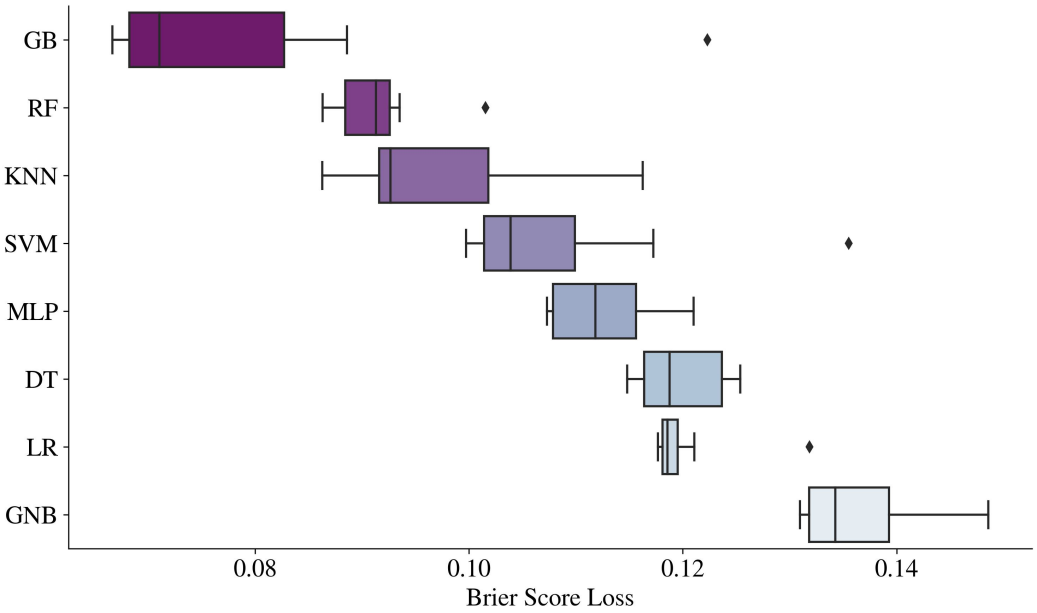
Source: Author's results in Python

Figure A.20: Matthews Correlation Coefficient Distribution (Black–box/White–box dimension) - *without outlier*



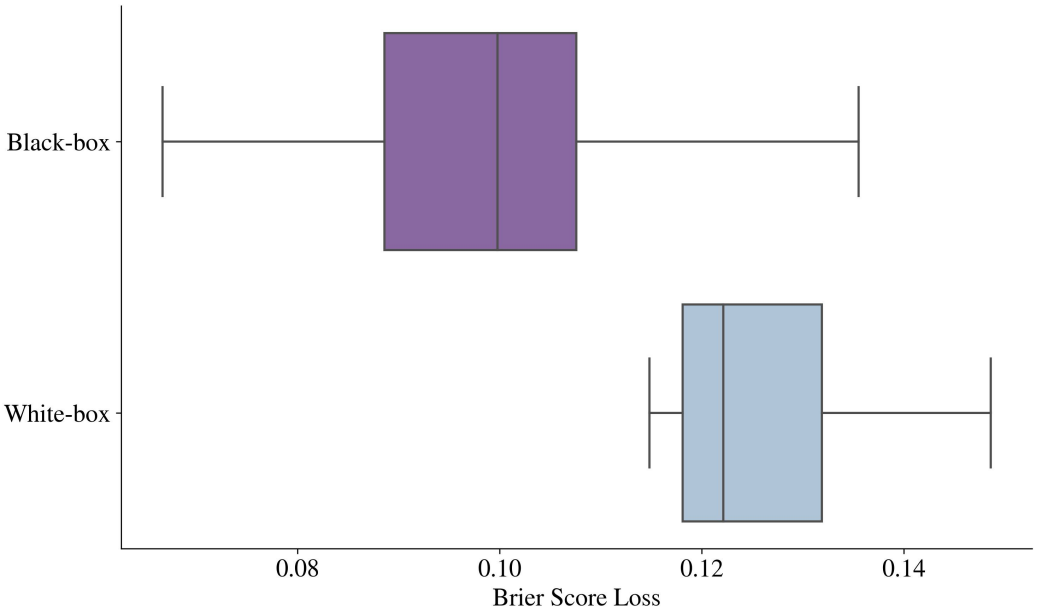
Source: Author's results in Python

Figure A.21: Brier Score Loss Distribution



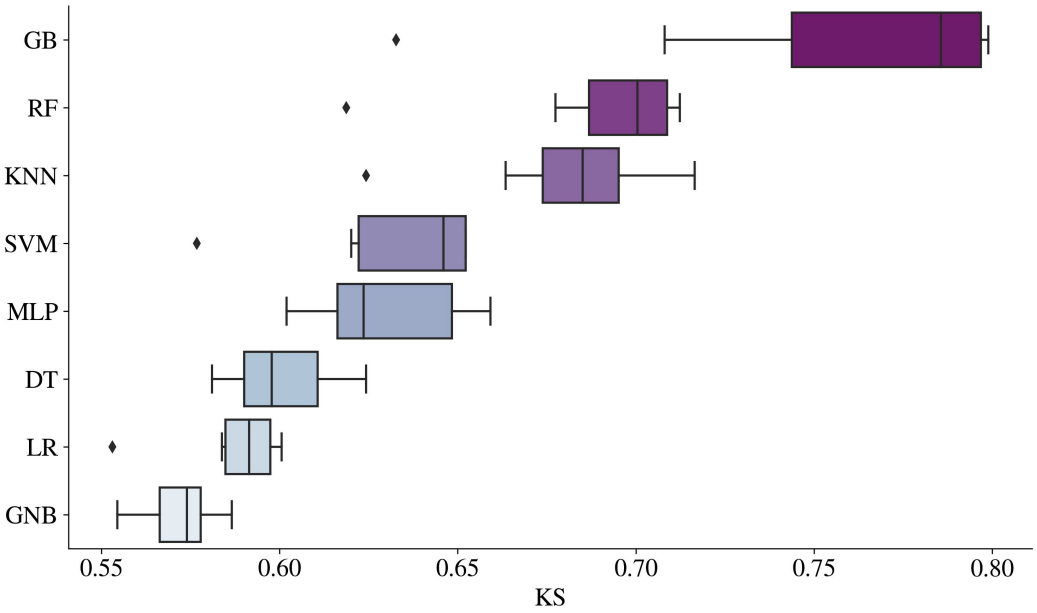
Source: Author's results in Python

Figure A.22: Brier Score Loss Distribution (Black–box/White–box dimension)



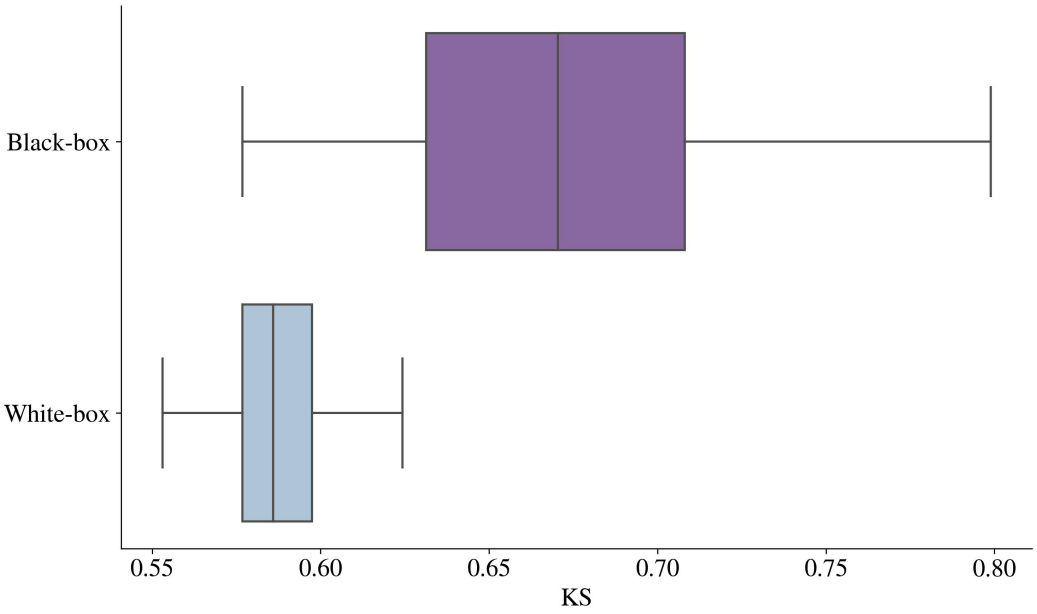
Source: Author's results in Python

Figure A.23: Kolmogorov–Smirnov Distance Distribution



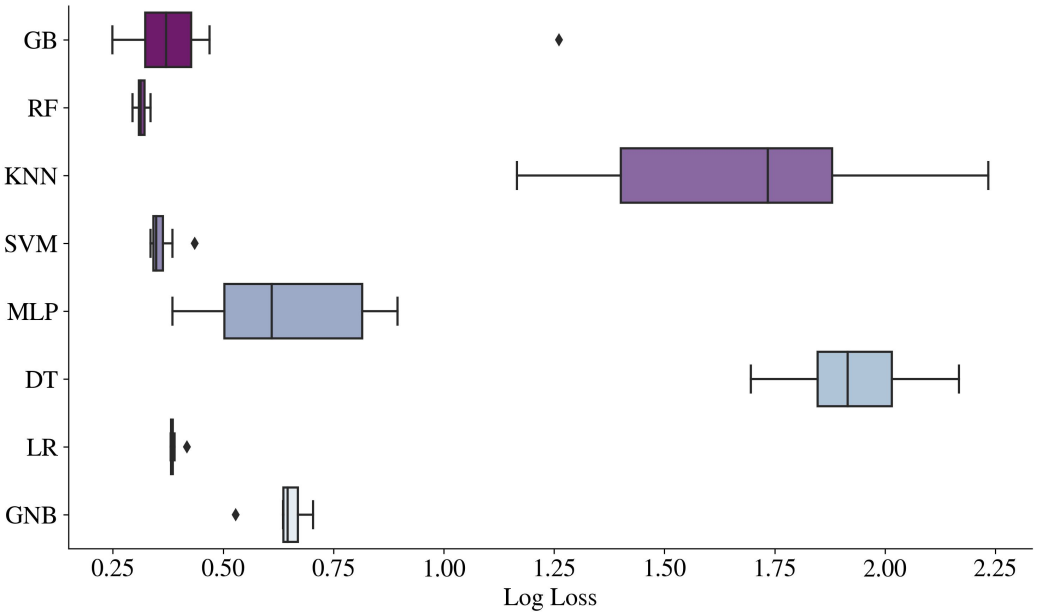
Source: Author's results in Python

Figure A.24: Kolmogorov–Smirnov Distance Distribution (Black–box/White–box dimension)



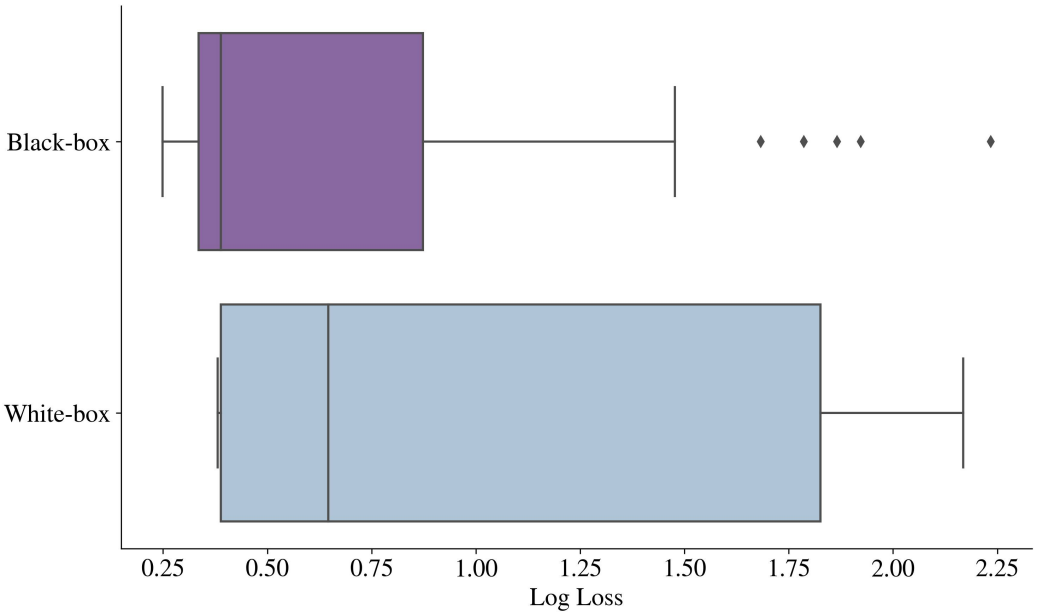
Source: Author's results in Python

Figure A.25: Log Loss Distribution



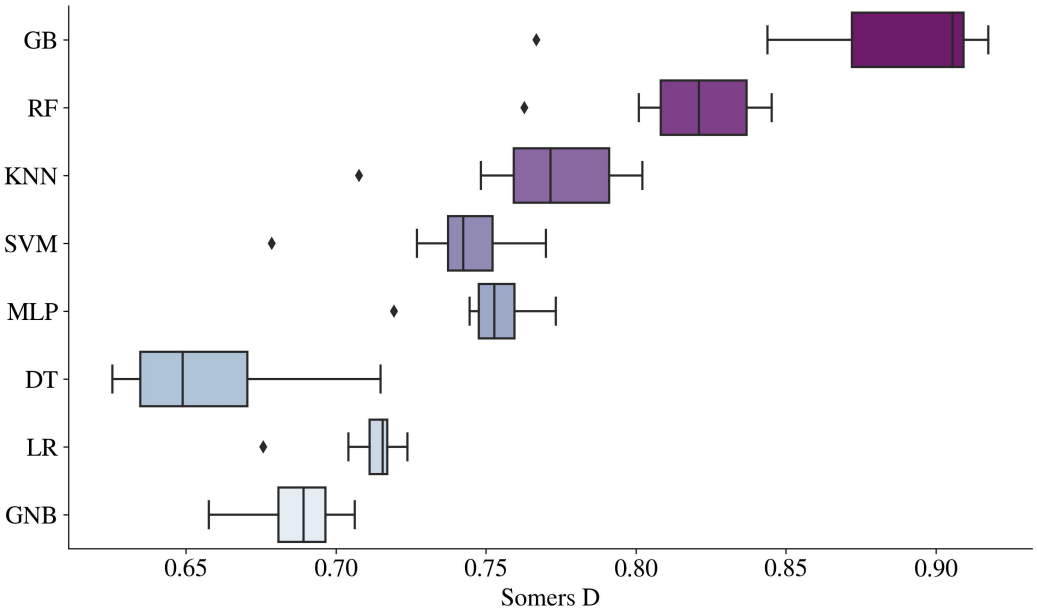
Source: Author's results in Python

Figure A.26: Log Loss Distribution (Black–box/White–box dimension)



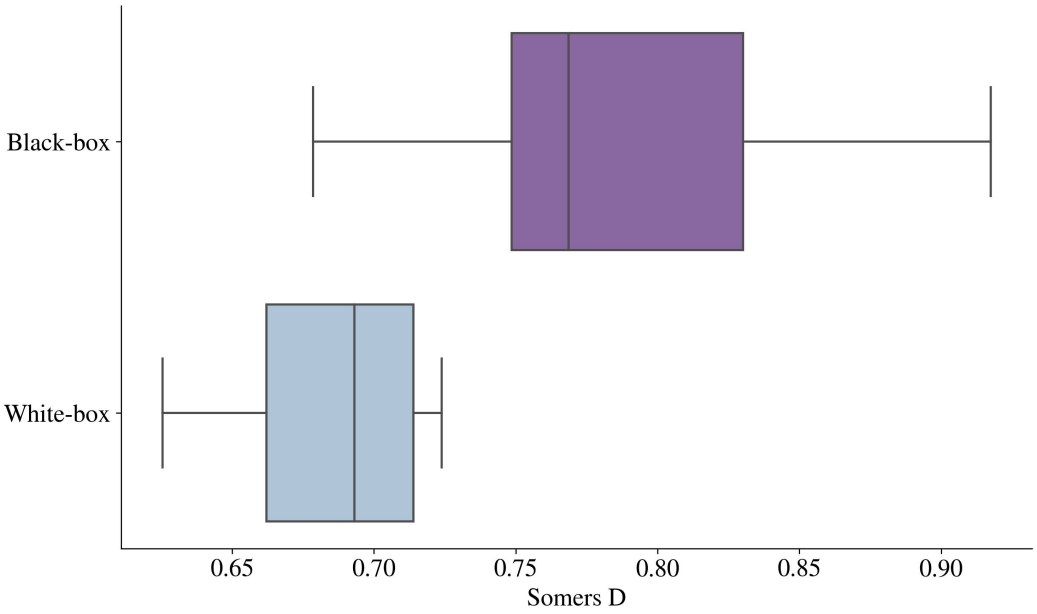
Source: Author's results in Python

Figure A.27: Somers' D Distribution



Source: Author's results in Python

Figure A.28: Somers' D Distribution (Black–box/White–box dimension)



Source: Author's results in Python