universität wien

MASTERARBEIT / MASTER’S THESIS

Titel der Masterarbeit / Title of the Master’s Thesis

„Bank Fraud Prediction using Machine Learning“

verfasst von / submitted by

Tomáš Tax

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 974
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Banking and Finance
Betreut von / Supervisor:	Dipl.-Inform. Dipl.-Volksw. Dr. Christian Westheide

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dipl.-Inform. Dipl.-Volksw. Dr. Christian Westheide for his time, guidance, and valuable comments.

Abstract

EN Machine learning is the main approach to financial fraud detection. Fraud datasets, however, typically bring several challenges, including a high class imbalance, problematic model evaluation, and limited interpretability. The novel Bank Account Fraud dataset examined in this thesis is no different. This thesis empirically examines Random Oversampling, Synthetic Minority Oversampling (SMOTE), Random Undersampling, and Cost-sensitive Learning as possible solutions to the high-imbalance problem training four machine learning classification algorithms in the process. Our results show Random Undersampling as the best-performing imbalance-handling technique and Logistic Regression together with Extreme Gradient Boosting Classifier as the best-performing model in terms of a proposed evaluation metric. Lastly, SHapley Additive exPlanations (SHAP) and Local Interpretable Model-agnostic Explanations (LIME) frameworks are discussed as a possible solution to explaining black-box models.

DE Machine Learning ist der wichtigste Ansatz zur Erkennung von Finanzbetrug. Datensätze zur Betrugserkennung bringen jedoch in der Regel mehrere Herausforderungen mit sich, darunter ein hohes Ungleichgewicht zwischen den Klassen ("High Class Imbalance"), eine problematische Modellbewertung und eine begrenzte Interpretierbarkeit. Der in dieser Arbeit untersuchte neuartige Datensatz zum Bankkontobetrug ist keine Ausnahme. In dieser Arbeit werden Random Oversampling, Synthetic Minority Oversampling (SMOTE), Random Undersampling und Cost-sensitive Learning als mögliche Lösungen für das High-Imbalance-Problem empirisch untersucht und dabei vier Machine-Learning Klassifizierungsalgorithmen trainiert. Unsere Ergebnisse zeigen, dass Random Undersampling das beste Verfahren zur Behandlung von High-Imbalance ist und dass die logistische Regression zusammen mit dem Extreme Gradient Boosting Klassifizierer das beste Modell in Hinblick auf eine vorgeschlagene Evaluierungsmetrik ist. Abschließend werden SHapley Additive exPlanations (SHAP) und Local Interpretable Model-agnostic Explanations (LIME) als mögliche Lösungen zur Erklärung von Black-Box-Modellen erläutert.

Contents

1. Introduction and Motivation

Financial fraud can be defined as a "deliberate act of deception intended for personal gain or to cause a loss to another party" (OLAF, 2017)[1]. With advancing technology and financial operations shifting to the online environment, the volume of fraud has been inevitably increasing over the past twenty years. Federal Trade Commission (FTC), an institution that collects information about consumer reports of fraud through its Sentinel Consumer Network, received 2.4 million reports of fraud which resulted in financial losses of almost 9 billion dollars in 2022. Moreover, they registered more than a million cases of identity theft in the same year (FTC, 2023). For illustration, in 2002, the same institution received 243 and 162 thousand reports of fraud and identity theft, respectively. This is equivalent to a 750 percent increase in the number of fraudulent activities reported over the past twenty years. According to the same report, in 2022, more than half of the identity theft reports consider credit cards or bank fraud. Moreover, the majority of both credit card and bank account fraud cases are classified as "new account fraud" with their quantity increasing by 13 and 32 percent year to year, respectively. Awoyemi et al. (2017) state that the dramatic increase in fraudulent activities has led to extensive research in the field of fraud detection using machine learning (ML) techniques. Thennakoon et al. (2019) add that since the long-established manual methods are ineffective, the ML approach remains the main modern solution when detecting fraud. However, there are several challenges to using machine learning.

First, Thennakoon et al. (2019) mention the lack of data available for research purposes due to high sensitivity and privacy issues to be the main issue. Jesus et al. (2022a) state that there are only a few data sets concerning credit card fraud available today due to these reasons. They mention Pozzolo et al. (2015), a data set containing transaction data about almost 300 thousand European cardholders from 2013[2], to be the most popular data source for training machine learning algorithms for fraud detection purposes. However, there is a major drawback to this data set as it contains only two interpretable features and the remaining 28 are principal components obtained from PCA in order to anonymize the data. It is therefore impossible to interpret models sensibly. Another dataset Jesus et al. (2022a) mention is Lopez-Rojas et al. (2016). However, neither one of the two datasets contains information about account opening fraud which, as mentioned above, is one of the fastest-growing and most common types of identity theft.

[1] See a full legal definition at `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32017L1371`

[2] Available at `https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud`

Motivated by the lack of available and informative data on new-account fraud, Jesus et al. (2022a) recently published a novel Bank Account Fraud (BAF) dataset(s) which they claim to be "realistic, based on a present-day real-world dataset for fraud detection". The recent availability of such an informative source motivates further research of machine learning applications for new account fraud such as this thesis. This paper explores and assesses different machine-learning algorithms and their application on the BAF dataset. However, even when having a dataset of satisfactory quality, doing machine learning for fraud detection purposes brings yet another major challenge: data imbalance.

High class imbalance, meaning that most of the observations in the data contain legitimate transactions/applications and only relatively few cases of fraud are present, makes it difficult to detect the few anomalies (Zojaji et al., 2016). The BAF dataset isn't different with only approx. one percent of all applications being fraudulent. Moreover, class imbalance calls for more advanced evaluation metrics as traditional measures like accuracy might fail to identify the desired model correctly. Although it is a complication, the class imbalance can be handled in several ways which will be the main point of focus of this thesis. This paper will explore and empirically assess the benefits of the use of undersampling and oversampling methods together with cost-sensitive learning and ensemble methods.

Finally, one of the main recent trends in Artificial intelligence and ML is to ensure model explainability. According to Psychoula et al. (2021), being able to provide reasoning behind ML model's decisions to end users is highly important and has become a regulatory requirement in many domains. Moreover, transparency about model mechanics helps to reduce model biases and remove potential discrimination. Cirqueira et al. (2020) add that explainable models help the end users to build more trustworthy relationships with AI. Furthermore, the authors stress the importance of explainable AI in fraud reviews as falsely flagged fraudulent cases cause financial harm to the customer. Therefore, after selecting the most suitable ML algorithms and imbalance-handling techniques, this paper will also pay attention to explainability of the trained models.

To sum up, with more financial operations happening online, financial fraud has become more common with new-account credit card fraud being one of the most common types of identity theft. As a consequence, machine learning has become the leading approach to fraud detection. Recently, Jesus et al. (2022a) published the Bank Account Fraud (BAF) dataset which contains information about applications for new credit cards and contains informative variables enabling better interpretation of the ML algorithms. This thesis takes advantage of this novel data source and uses it to achieve the following research goals:

1. Train and evaluate different machine learning algorithms for fraud detection purposes; namely Logistic Regression (LR), Decision Tree (DT), Random Forest Classifier (RF), and Extreme Gradient Boosting Classifier (XGBoost). For all of these models, the most important hyperparameters are to be identified and optimized through

randomized search to find the best model specification.

2. Explore and test different methods of handling high class imbalance which is present in the BAF dataset. These methods include cost-sensitive learning and resampling methods; namely Synthetic Minority Oversampling Technique (SMOTE), random oversampling (ROS), and random undersampling (RUS). These methods will be applied to the four models from above to see which method(s) is the most appropriate for each of the models.

3. Finally, as the BAF dataset comprises informative features, all trained models are to be explained to increase their trustworthiness and transparency. Logistic regression and Decision Trees do not require any additional framework as they are explainable by their nature. However, in the case of Random Forest and XGBoost, Shapley values and the LIME framework will be used to provide model explanations.

This text proceeds with chapter 2 which provides an introduction to fraud detection with machine learning, briefly presents the machine learning algorithms used in this thesis, and familiarises the reader with different ways of handling class imbalance together with techniques used for model explanation. Next, chapter 3 first describes the dataset in more detail and explains how the empirical part of this research proceeds. Once the reader understands what exactly was done, chapter 4 presents the empirical results of this research. Finally, chapter 5 summarizes the research.

2. Literature Review

2.1. Credit Card Fraud

As mentioned in the introduction, financial fraud has been on the rise for the last twenty years. A significant portion of fraudulent activities fall under credit card or bank account fraud (FTC, 2023). In the recent past, a lot of attention on credit card fraud concerned transactions, i.e., fraud happening to existing cardholders. According to Le Borgne et al. (2022) for transactional credit card fraud, multiple scenarios exist, categorized as card-present (CP) and card-not-present (CNP) frauds. CP frauds involve physical card usage, such as at point-of-sale terminals or ATMs. These encompass lost or stolen cards, and cards not received. In lost or stolen card scenarios, legitimate cards fall into the hands of fraudsters through theft or loss and card-not-received incidents involve fraudsters intercepting new cards sent to customers. On the other hand, CNP fraud occurs remotely through mail, phone, or online transactions, utilizing card information without requiring a physical card. They are often a result of illegally acquired payment credentials, typically obtained from data breaches or cardholders themselves through phishing. Criminal groups then sell this data on the dark web for fraudulent use. Such fraud predominantly relies on stolen card numbers, expiration dates, security codes, and personal billing information. All of the above is transactional credit card fraud.

The focus of researchers (such as Awoyemi et al. (2017), Thennakoon et al. (2019), Varmedja et al. (2019), and Dornadula and Geetha (2019)) on transactional credit card data might have been also connected to the fact, that there were only a few publicly available datasets about fraud, most of which concern, according to Zojaji et al. (2016), transaction data. Jesus et al. (2022a) mention Pozzolo et al. (2015)[3] and Lopez-Rojas et al. (2016) as the two most popular credit card datasets - both contain transactional data.

Recently, however, a novel dataset containing data about online applications for credit cards within a major consumer bank was published by Jesus et al. (2022a)[4]. This new Bank Account Fraud (BAF) dataset enables researchers to turn their focus more towards application fraud. In the context of online credit applications, fraudsters attempt identity theft or fabricate fictional personas to gain unauthorized access to banking services. Once granted access to a new account, fraudsters immediately exploit the available credit or

[3] Available at: `https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud`. This is probably the most common public dataset. It is used, for example, by Awoyemi et al. (2017), Varmedja et al. (2019), Dornadula and Geetha (2019)

[4] Available at `https://www.kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022`

employ the account for illicit transactions. The financial burden falls entirely on the bank, as the fraudster's true identity remains untraceable.

Jesus et al. (2022a) point out that bank account application fraud is a high-stakes domain of machine learning application because indicating fraud results in the rejection of a customer's bank account application, while a negative prediction (predicting application as legitimate) grants access to a new bank account and its associated credit card. The authors emphasize that in this particular case of fraud prediction, even a small decline in model predictive performance might translate into substantial losses for the bank. While the authors point out that they made the BAF dataset as informative (interpretable features)[5] as possible to allow mainly for fairness evaluation[6], the fact that they include meaningful real-world variables also benefits users in the form of the possibility of model explainability (explainable AI) which is of concern to this paper and is explained in more detail later in section 2.5.

2.2. Fraud Detection with Machine Learning

As mentioned in the introduction, because of the increasing amount of data, it is currently nearly impossible for a human specialist to discern meaningful patterns in transaction data. As a result, machine learning techniques are increasingly widely used in the field of fraud detection. In recent years, the capacity of machine learning methods to effectively solve the difficulties highlighted by credit card fraud detection has resulted in a significant increase in research in this field. Le Borgne et al. (2022) states that the number of published articles related to fraud detection with ML increased from not even 300 in 2010 to about 1500 in 2020. In his book, Le Borgne et al. (2022) mentions 10 recent surveys published between 2015 and 2021 which review the state of the current research. The most recent reviews mentioned by the author are Lucas and Jurgovsky (2020), Priscilla and Prabha (2020), or Yousefi et al. (2019). The majority of the reviews focus on two topics: an overview and comparison of machine learning methods that can be used to detect fraud and the challenges which fraud detection comes with. A brief overview of both methods and challenges is provided in the following two sections.

[5] The prior-mentioned Credit Card Fraud Detection dataset consist of 2 informative variables (amount and time) and 28 non-interpretable variables which come from the Principal Component Analysis. A dataset consisting of principal components is of no use for fairness or model explainability purposes.

[6] According to Begley et al. (2020), machine learning is considered unfair in the case of algorithmic bias, i.e., when the model discriminates with respect to a protected attribute (e.g. race, sex, or age). Models get their discriminatory bias from past data. The field of fair ML emerged to ensure that important decisions made based on machine learning are not discriminatory. If interested in Fair ML, see, for example, Begley et al. (2020), Corbett-Davies and Goel (2018), Barocas and Selbst (2016), Caliskan et al. (2017), or Bolukbasi et al. (2016).

2.2.1. Challenges

The challenges accompanying fraud detection tasks which are mostly often stated in the previously-mentioned reviews are:

1. *Skewed distribution/class imbalance.* High class imbalance is usually present in fraud data as, in general, there are more legitimate actions happening relative to fraudulent activities. This is an issue as some algorithms might not be able to learn the patterns behind the minority class and therefore predict solely the majority class. High class imbalance can be handled, typically by sampling or cost-weighting approaches (Le Borgne et al. (2022), Lucas and Jurgovsky (2020), Mekterović et al. (2018), Adewumi and Akinyelu (2017)). Imbalance handling is the main research topic of this paper. The techniques used in the empirical part of this research are discussed in section 2.4.

2. *Performance measures.* Class imbalance complicates model evaluation as some traditional metrics (such as accuracy or AUC) might fail to choose a desirable model (Le Borgne et al. (2022), Mekterović et al. (2018)). Confusion-matrix-based measures which account for both costs of false positives (rejecting legitimate applications) and false negatives (not detecting fraud) are usually used. Mekterović et al. (2018) suggest, for example, F-measure, G-mean, or Mathew's Correlation Coefficient to be well suited for imbalanced tasks. Lucas and Jurgovsky (2020) point out that ML algorithms like Logistic regression, SVM, Decision trees, Ensemble methods, or neural networks return a probability of a prediction belonging to a positive class which is then translated into a 0/1 prediction according to some threshold. In the case of these techniques, it might be advantageous to use the ROC curve or Precision-Recall curve as an evaluation measure as by varying the value of a threshold parameter over all possible probabilities obtained with the machine learning classifier, one may track the prediction efficiency for each case, from the most obvious to classify to the least obvious. Indeed, Jesus et al. (2022a), the authors of the dataset used in this research, recommend maximizing recall at a fixed false-positive rate which can be most easily done through the ROC curve. For this reason, the ROC curve was selected as the main evaluation metric for the empirical part of this thesis. It is discussed in more detail in section 2.3.5.

3. *Interpretability.* Lucas and Jurgovsky (2020) claim that ensuring the interpretability of machine learning algorithm decisions holds significant importance for users from the targeted group. As mentioned in the introduction, model explainability is one of the recent trends in AI which increases the transparency and trustworthiness of ML models (Cirqueira et al., 2020). Moreover, there is a trend toward including explainability in the regulatory structures (Psychoula et al., 2021). For these reasons, this research also pays attention to explainability techniques which are further discussed in section 2.5.

4. *Categorical features.* According to Le Borgne et al. (2022), credit card data usually contain many categorical features. This poses a problem as many ML algorithms

cannot handle categorical variables which then must be translated into numbers (Lucas and Jurgovsky, 2020). Commonly, categorical variables are transformed into dummy variables which can significantly increase the dataset dimension posing yet another challenge to the data scientist.

5. *Dataset/concept drift.* Lucas and Jurgovsky (2020) mention that fraudsters adapt their methods over time, rendering some strategies outdated due to the ongoing interplay between them and experts. Additionally, as stated in Le Borgne et al. (2022), emerging technologies can expose new vulnerabilities that fraudsters exploit. Furthermore, credit card users' spending behaviors change during different time periods. These shifts, alongside evolving transaction and fraud patterns, contribute to what is known as concept drift – where the distributions of transactions and fraud cases change over time. This phenomenon, however, is not of interest to this research and, therefore, won't be discussed here any further.

6. *Lack of data.* As mentioned before, there is a shortage of real-life public datasets which poses a problem to fraud detection as a whole field. However, this research uses a novel dataset published by Jesus et al. (2022a) and thus does not suffer from this issue.

2.2.2. Common Machine Learning Techniques in Fraud Detection

The following part gives a brief overview of ML methods that are commonly used for fraud detection purposes as provided by Le Borgne et al. (2022), Yousefi et al. (2019), and Mekterović et al. (2018). ML methods can be generally divided into supervised and unsupervised learning.

Supervised Learning. As stated in Le Borgne et al. (2022), supervised learning can be split into two stages. The initial stage involves constructing a predictive model using a collection of labeled historical data. This process is termed supervised learning, as it involves knowing the class labels (legitimate or fraudulent). Subsequently, in the second stage, the predictive model derived from the previous process is deployed to predict the labels of unseen observations. The procedure commonly includes training a range of models and assessing their predictive performances using a validation set. Ultimately, the model with the presumed best performance is chosen for making predictions on new transactions.

Le Borgne et al. (2022), Mekterović et al. (2018), and Yousefi et al. (2019) come to approximately the same conclusions about the most commonly used ML algorithms. All three papers state Logistic regression (LR), Decision trees (DT), Artificial Neural Networks (ANN), Support Vector Machines (SVM), and Naïve Bayes. Moreover, Mekterović et al. (2018) and Yousefi et al. (2019) name Random Forest (RF), one of the popular ensemble techniques, to be another frequent model of choice. Le Borgne et al. (2022) puts ensemble learning as a separate group of ML tasks, however, he admits that, technically, ensemble methods fall under supervised learning as they typically require labeled data. One algorithm that all three papers name, yet claim it not to be so common is K-Nearest Neighbor. For the purposes of the empirical part of this thesis, LR, DT, and RF were selected. Moreover, one not-so-common but powerful technique - Extreme Gradient Boosting - is considered. All four methods used in this research are described in more detail in the following section (2.3).

Unsupervised Learning. As stated in Yousefi et al. (2019), unsupervised techniques identify alterations in behavior or atypical transactions. Within these methods, a model of legitimate user behavior is established, and any activities significantly deviating from this norm are identified as fraudulent. An advantage of unsupervised approaches lies in their greater capability to detect novel forms of fraud compared to supervised methods. Furthermore, these techniques prove valuable in scenarios lacking labeled data, as they do not require pre-existing information, offering applicability in situations with limited or no prior knowledge. However, this comes at a cost of higher computational complexity relative to the supervised method and typically with less accurate predictions, according to Le Borgne et al. (2022). Both, Yousefi et al. (2019) and Le Borgne et al. (2022) determine the K-Means and Hidden Markov Model to be the most common among unsupervised techniques.

2.3. Machine Learning Algorithms

This research aims to predict whether an application is fraudulent or not. Therefore, we are concerned with a classification task. For the purpose of this paper, four classification algorithms were applied: Logistic Regression (LR), Decision Tree (DT), Random Forest (RF), and Extreme Gradient Boosting (XGBoost). This chapter provides an introduction to each of the models' mechanics. Finally, a selected evaluation metric (the Reciever Operating Characteristic) which is to be used to get insight in the models' predictive performance is mentioned at the end of this section.

2.3.1. Logistic Regression

Logistic regression is based on linear regression, however, it is suited for classification tasks as it predicts class probability, not a numeric value. As logistic regression predicts probability rather than just class (0/1), it is fitted using maximum likelihood (see Aggarwal (2015)). Similar to linear regression, logistic regression is a linear additive model, which can be expressed as

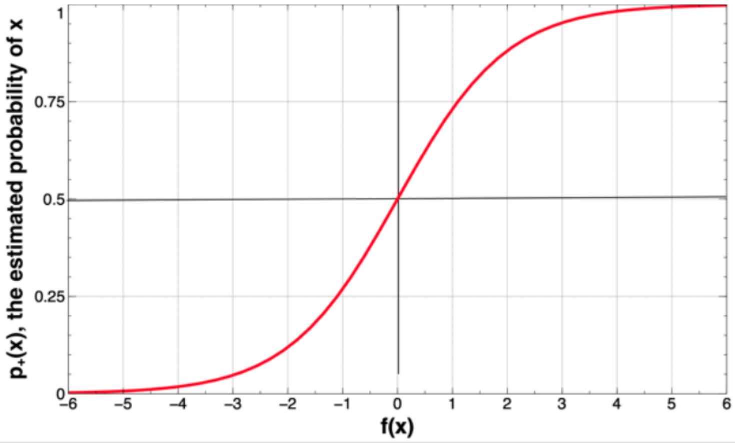
$$log\left(\frac{p_+(x)}{1-p_+(x)}\right) = f(x) = w_0 + \sum_{i=1}^{n} w_i x_i \tag{2.1}$$

with $log\left(\frac{p_+(x)}{1-p_+(x)}\right)$ being a log odd of an observation being of the positive class given feature vector x. From 2.1, the probability of membership to a positive class can be extracted using basic algebra:

$$p_+(x) = \frac{1}{1 + e^{-f(x)}} \tag{2.2}$$

When plotting $p_+(x)$ from 2.2 as a function of log odds ($f(x)$), one gets a sigmoid function plotted below in Figure 2.1. Let's clarify the relation between (log)odds and probabilities by giving two numeric examples from Provost and Fawcett (2013). First, assume that, based on past data, observation is equally likely to belong to a positive as to a negative class. The odds (x) of this observation thus are 50:50 (=1). Taking the log of this odd, we get a log-odd of zero ($f(x) = 0$ as $log(1) = 0$). When plugging $f(x) = 0$ to the sigmoid function or into the formula in equation 2.2, we get an estimated probability $p_+(x)$ of 0.5. Let's continue with the example, this time with the observed probability being 0.9. Then the odds (x) are 90:10 or 9. Again, we can take the logarithm of 9 and get $f(x) = log(9) \approx 2.19$ which, after plugging into 2.2 or the sigmoid function, gives us a probability estimate of 0.9.

Figure 2.1.: Logistic Regression: Sigmoid Function



Source: Provost and Fawcett (2013), page 100.

According to Provost and Fawcett (2013), the sigmoid function plots the estimated probability as a function of the distance of an observation from the decision boundary. In other words the further away from the decision boundary (i.e. where $f(x) = 0$) an observation is, the further is the estimated probability from 0.5 ($p_+(x)$ corresponding to $f(x) = 0$). Provost and Fawcett (2013) point out that the estimated probability grows approximately linearly near the decision boundary, but intuitively then converges to certainty when farther away.

According to Aggarwal (2015), logistic regression can suffer from overfitting[7], especially with highly-dimensional data. The authors add, that one way to prevent overfitting of logistic regression is through regularization. This means adding a regularization term

[7] Overfitting happens when model describes noise instead of the underlying patterns in the data. In general, it results in poor performance on a validation/test set (Aggarwal, 2015).

to penalize the log-likelihood in case the model parameters are high. See Aggarwal (2015) for more detail. In Python, Logistic regression is offered in the Scikit learn library (Pedregosa et al., 2011) and comes in the regularized form by default. See the Scikit Learn Documentation for more detail.

2.3.2. Decision Tree

Decision trees (DT) are a non-parametric method that uses simple decision rules obtained from the data to estimate the class probability or a numeric value (Louppe, 2015). They are characterized by a sequence of branching nodes that partition data based on feature attributes, ultimately leading to terminal nodes representing class labels or outcomes (Breiman, 2017). At each internal node, a decision criterion is applied to split the data into subsets, guided by a specific attribute, aiming to maximize information gain or purity. Aggarwal (2015) mention two most common impurity measures (to be minimized). The first mentioned is the Gini index:

$$G(N) = 1 - \sum_{i=1}^{k} p_i^2 \tag{2.3}$$

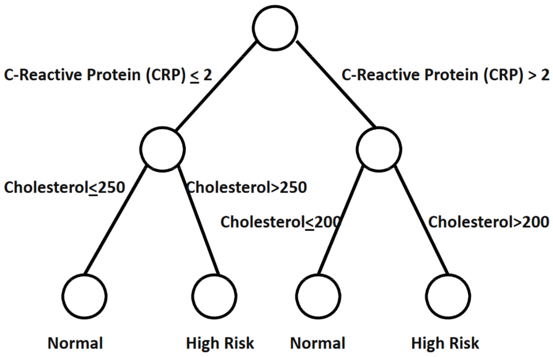
where $G(N)$, the gini index at node N, lies between 0 and $1 - 1/k$ where k is the number of distinct classes present at node N and p_j is a fraction of class j at node N. A second impurity measure mentioned by Aggarwal (2015) is Entropy:

$$E(N) = -\sum_{i=1}^{k} p_i \cdot log(p_i) \tag{2.4}$$

Entropy ranges from 0 to $log(k)$. At entropy's maximum, classes present in the node are in perfect balance. At each node of a DT, a decision criterion is selected such that the selected impurity measure is minimized. Through these iterative binary divisions, DTs construct a predictive model capable of capturing complex decision-making processes within a dataset, facilitating the classification of new instances by traversing the tree from the root to an appropriate leaf node, thereby offering interpretability and insight into the underlying decision logic (Pedregosa et al., 2011).

A simple example provided by Aggarwal (2015) helps to understand DT mechanics quite well. First, DT was built as follows: At the first node, all observations are considered. Look at all features of the data and build different decision rules (for example, age > 50, age > 70, CRP > 2, CRP > 3, CRP > 4, etc.). Select the one decision criterion which leads to the highest purity gain given a selected impurity measure. In the example 2.2, the most successful split criterion is "C-Reactive Protein (CRP) ≤ 2". Now, use this decision criterion to split the dataset into two subsets. Next, repeat the same procedure for both of the subsamples. It can be seen from the figure, that for both nodes at depth 2, the split criterion is based on the feature "Cholesterol", however, a different value is used at each node. Now when the decision tree is built, we can provide an example of how

Figure 2.2.: Decision Tree Example



Source: Aggarwal (2015), page 9.

it makes predictions. Assume an individual with CRP of 1.5, Cholesterol value of 190, age of 70, and possibly also other features. We proceed from the root of the tree toward the leaves (from top to bottom). First, since the observed CRP is 1.5, the individual continues to the left node. At this node, the level of cholesterol is used for splitting. Since the observed level of cholesterol is 190 which is lower than the critical value, again, we continue left. As this tree has only two layers (depth = 2), we arrive at the left-most leaf which translates into a prediction of "Normal Risk" for the examined individual.

Finally, Aggarwal (2015) points out that by default, a DT will keep splitting the nodes until it arrives at pure leaves (with only one class present). This, however, leads to overfitting, i.e., the DT fails to perform well on a testing set. Moreover, the author mentions that, even in a training set, having a large tree might bring only a limited performance gain compared to a small one. A solution to overfitting is tree *pruning*, i.e., replacing redundant subtrees (with already high purity) with a single leaf. Pyton's Scikit Learn default values for training DTs lead to fully grown trees. Pedregosa et al. (2011) recommend tuning hyperparameters `max_depth`[8], `min_samples_split`[9], and `min_samples_leaf`[10] in order to avoid overfitting and reduce computational complexity.

2.3.3. Random Forest

Building upon the principles of decision trees, Random Forests constitute an ensemble learning technique that combines multiple decision trees to enhance predictive accuracy and robustness. Aggarwal (2015) explain that Random Forests introduce an element of diversity by creating a multitude of decision trees, each trained on a different random subset of the original dataset. Through this process of bootstrapping and random feature

[8] Specifies the maximum depth of a tree (Pedregosa et al., 2011).

[9] Specifies the minimum number of observations in a node to continue splitting (Pedregosa et al., 2011).

[10] Similar to `min_samples_split` but considers observations in each leaf/subtree after the split instead of before (Pedregosa et al., 2011).

selection, Random Forests aim to reduce overfitting and improve generalization to new data.

When making predictions, each tree in the Random Forest independently classifies an instance, and the final prediction is determined through a majority vote or averaging mechanism (Breiman, 2001). This ensemble approach contributes to better overall performance by collectively capturing a wider range of patterns and relationships within the data (Louppe, 2015). The strength of RFs lies in their ability to handle complex datasets and mitigate issues associated with individual decision trees, such as high variance and sensitivity to small variations in data. Moreover, as stated in Breiman (2017), RFs offer insights into feature importance by measuring the contribution of each attribute in the prediction process, aiding in understanding the underlying factors driving the model's decisions. On the other hand, as RF selects a sample of features randomly, they might underperform in datasets with overall many variables but only a few informative features as RF might fail to choose them.

In the empirical part of this research, RF Classifier from the Scikit Learn library will be used. For Python documentation, visit Scikit Learn web page.

2.3.4. Extreme Gradient Boosting

Extending the framework of ensemble learning, the Extreme Gradient Boosting (XGBoost) classifier represents a prominent advancement in machine learning techniques. XGBoost harnesses the power of boosting, a sequential method where weak learners (in this case, decision trees) are progressively refined to form a robust and accurate predictive model. Brownlee (2016) argues that XGBoost introduces innovative mechanisms to optimize performance, making it particularly useful for handling complex and high-dimensional data. The author claims that XGBoost is a go-to model which he supports by quoting several Kaggle competition winners who used XGBoost.

Similarly to Random Forest, XGBoost is also an ensemble technique, i.e., it decides based on majority voting of many built trees. However, on contrary to RF, it doesn't build the decision trees by random sampling. As explained by Brownlee (2016), in each iteration, XGBoost constructs a new decision tree that focuses on addressing the errors or misclassifications made by the ensemble thus far (simply put, it learns from its mistakes). Its name Gradient Boosting comes from the fact that the algorithm uses gradient descent to fix the errors made by the model in the previous iteration. This iterative process directs the model's emphasis towards challenging instances, ultimately leading to an ensemble that excels at capturing intricate relationships within the data.

To strike a balance between predictive power and model complexity, XGBoost introduces novel techniques for regularization. These include controlling tree depth, employing a sparsity-aware split finding algorithm, and applying L1 and L2 regularization terms on leaf scores (Chen and Guestrin, 2016). Such measures prevent overfitting, ensuring that the model generalizes well to unseen data. As the name implies, XGBoost leverages the power of extreme gradients to elevate the performance of ensemble models. (Brownlee, 2016).

In the empirical part, XGBoost algorithm from the `xgboost` library will be used as, according to (Chen and Guestrin, 2016), it is the best-performing package compared to other libraries which provide tree-boosting systems. See the XGBoost documentation for more detail on the library.

Probability estimates. All four algorithms are well-suited for classification tasks. Moreover, all of the models return probabilities of an observation belonging to a positive class. Logistic regression (2.3.1) returns these probabilities directly. Decision Trees (2.3.2) estimate the probability as a ratio of the positive class to the overall number of observations in the selected leaf of the tree. That is if the observation for which we make a prediction ends up in a leaf that, in the training phase, consisted of 10 samples out of which 3 were positive, the DT predicts a 0.3 probability of the observation being positive. It's necessary to point out that if a leaf contains a small number of samples, the probability estimate might be inaccurate. Next, Random Forests (2.3.3) estimate the probabilities according to the number of tree estimators that voted for the prediction to be positive. For example, if in a forest of 100 trees 75 vote for the prediction to be positive, RF yields a probability of 0.75. Finally, XGBoost (2.3.4) models the predicted probabilities as continuous values directly during training as it uses the sigmoid function to transform these continuous values into probabilities.

The possibility of getting probability estimates of these models allows for a unified evaluation approach using the Receiver Operating Characteristic which is defined only for probabilistic models, not for discrete classifiers (Aggarwal, 2015).

2.3.5. Receiver Operating Characteristic: The Evaluation Metric

Before proceeding to the Receiver Operating Characteristic (ROC), the basis of classification evaluation in the form of a confusion matrix needs to be laid down. As claimed by Aggarwal (2015), the confusion matrix (Table 2.1) offers a comprehensive breakdown of a classification model's predictions. It categorizes outcomes into four distinct scenarios: True Positives (TP), where the model correctly identifies positive instances; True Negatives (TN), where negative instances are correctly identified; False Positives (FP), where the model incorrectly labels negative instances as positive; and False Negatives (FN), where positive instances are incorrectly classified as negative. These elements form the foundation for metrics like accuracy, precision, recall, and the ROC curve, providing a nuanced understanding of a model's performance across various outcomes and aiding in making informed decisions regarding its effectiveness.

Fan et al. (2006) claim the ROC curve to be a fundamental tool in evaluating and comparing the performance of classification models as it provides valuable insights into

		Actual Class	
		Positive	Negative
Predicted Class	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Table 2.1.: Confusion Matrix

the trade-off between true positive rate (TPR)[11] and false positive rate (FPR)[12]. In the context of credit card fraud, TPR tells us how many of the actual fraudsters the model correctly flagged, whereas the FPR provides information on how many "innocent" applications were incorrectly rejected.

$$TPR = recall = \frac{TP}{TP + FN} \tag{2.5}$$

$$FPR = falarm = \frac{FP}{FP + TN} \tag{2.6}$$

In essence, the ROC curve illustrates how well a model can distinguish between positive and negative instances at various probability thresholds by plotting TPR and FPR against each other in a two-dimensional graph. Probabilistic classifiers need probability thresholds to make a final decision of a class.[13] Note that for a selected probability threshold, we get only one point in the TPR-FPR space. However, by iteratively shifting the threshold from 0 to 1, we get a continuum of points - the ROC curve shown in Figure 2.3 below.

The ROC curve showcases the model's ability to correctly identify true positives while minimizing the number of false positives. The higher the TPR for a given FPR, the better the model, therefore, the goal of any classifier should be to shift the ROC curve as much to the upper-left corner as possible as an algorithm touching the (0,1) point is a perfect classifier. On the contrary, point (0,1) marks the worst possible model (Aggarwal, 2015). A straight line connecting points (0,0) and (1,1) corresponds to a so-called random classifier as here TPR always equals FPR and the decision is 50:50, i.e., random. To quantify classifier quality and allow for easier comparison across different classifiers, the area under the ROC curve (AUC-ROC) serves as a quantitative metric summarizing the model's overall discriminatory performance; a higher AUC-ROC implies a better-performing model.

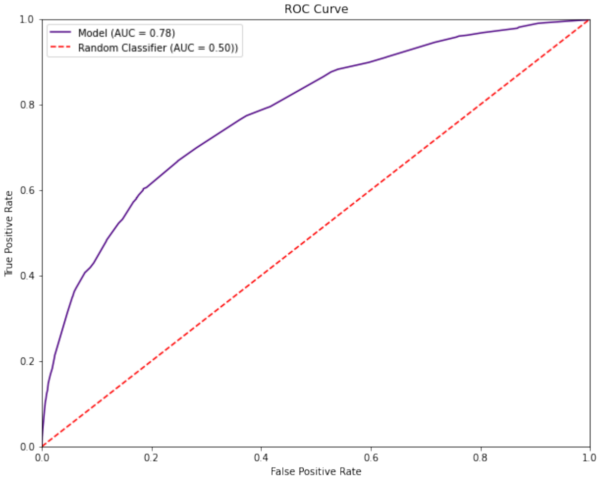
Another reason why ROC was selected as a primary evaluation metric is due to its relevance for imbalanced classification such as fraud detection (Aggarwal, 2015). The ROC curve visually illustrates how models navigate the TPR-FPR trade-off in cases

[11] TPR is also called Recall.

[12] FPR is sometimes called false alarm rate or falarm for short (Aggarwal, 2015).

[13] Example: Assume a model returns a probability estimate of 0.25. By default, this translates into a prediction of 0 as the estimated probability is lower than 50. However, this threshold can be set to, for example, 0.1 in which case, the algorithm would predict a positive class. By changing this threshold, the algorithm changes its predictions which shifts it along the ROC curve.

Figure 2.3.: The Receiver Operating Characteristic Curve



where one class highly overweights the other. It shows the model's capacity to detect the minority class (frauds) while effectively managing the majority class (genuine transactions). This makes the ROC curve a valuable tool for assessing the model's ability to handle imbalanced data and to guide the selection of a suitable threshold that balances sensitivity and specificity. Indeed, Jesus et al. (2022a), the authors of the BAF dataset, recommend optimizing for a model that maximizes recall at 5% FPR. Such a task requires training a model and, later, tuning the probability threshold in a way the model reaches the 5% FPR in the ROC.

To summarize, the ROC curve offers a comprehensive perspective on the performance of probabilistic models, particularly in imbalanced classification scenarios. It showcases their capability to distinguish between classes and assists in identifying an optimal threshold for achieving desired trade-offs between true positives and false positives, making it a useful tool for robust evaluation and model selection in credit card fraud detection.

Precision-Recall Curve. A useful alternative to ROC, yet not so popular, is the Precision-Recall Curve which plots Recall (TPR) against Precision (2.7). As mentioned by Cook and Ramadas (2020), its advantage lies in its sensitivity to class imbalance, as it focuses on the performance of the minority class, providing insights into the model's ability to identify positive instances accurately. However, a drawback of the precision-recall curve is that it does not capture the overall performance across varying decision thresholds as comprehensively as the ROC curve.

$$Precision = \frac{TP}{FP + TP} \tag{2.7}$$

2.4. Imbalanced Classification

Chapter 2.2 mentions that fraud detection typically brings a challenge in the form of imbalanced data. The dataset published by Jesus et al. (2022a) is not any different in this manner. Not only that class imbalance calls for the need for an imbalance-robust evaluation technique such as ROC, but it also requires imbalance-handling techniques to be employed. Aggarwal (2015) claims Cost-Sensitive Learning and Resampling to be the leading techniques of imbalance-handling. Both of these approaches are discussed in the upcoming section as both are employed in the empirical part of this research.

2.4.1. Cost Sensitive Learning

In imbalanced scenarios, training the model with the objective being maximizing its overall accuracy, might not bring the desired fruit. Aggarwal (2015) mentions that optimizing accuracy does not correspond to the real-world objective since in the case of credit card fraud, misclassifying a fraudulent transaction bares a different (likely higher) cost than falsely identifying a legitimate transaction as a fraud. The cost-sensitive learning method addresses this fact by making the algorithm take into account the costs associated with misclassifying different classes. It aims to minimize these costs by adjusting the model's predictions and decision boundaries. By assigning higher penalties for misclassifying the minority class, cost-sensitive learning ensures that the model pays more attention to accurately predicting the rare class.

The question remains, what weights to assign to each of the classes? According to Aggarwal (2015), multiple heuristic methods exist to answer this question. The simplest, however, is to weight proportional to relative class frequencies. For example, when having 100 samples in total, 5 being of class 1, the rest of class 0. The simple approach would multiply the misclassification cost of class 1 by $(1 - \frac{5}{100})$ to achieve the same misclassification cost for both classes. Indeed, this approach is also incorporated in most of Scikit Learn's classifiers by using the `class_weight` argument (Pedregosa et al., 2011). The same holds for the XGBoost library (Chen and Guestrin, 2016) and, therefore, this approach will be used in the practical part of this research. If interested in more advanced weighting methods, see Aggarwal (2015), chapter 17.2.

2.4.2. Resampling Techniques

Resampling techniques are employed in imbalanced classification scenarios to modify the class distribution (directly at the dataset level), with the aim of increasing the influence of the rare class on the algorithm. As mentioned by Aggarwal (2015), in general, resampling is done by oversampling the rare class, undersampling the frequent class, or a combination of both approaches. Subsequently, the model is trained[14] on the re-sampled data. This approach ensures an elevated representation of the rare class within the learning sample,

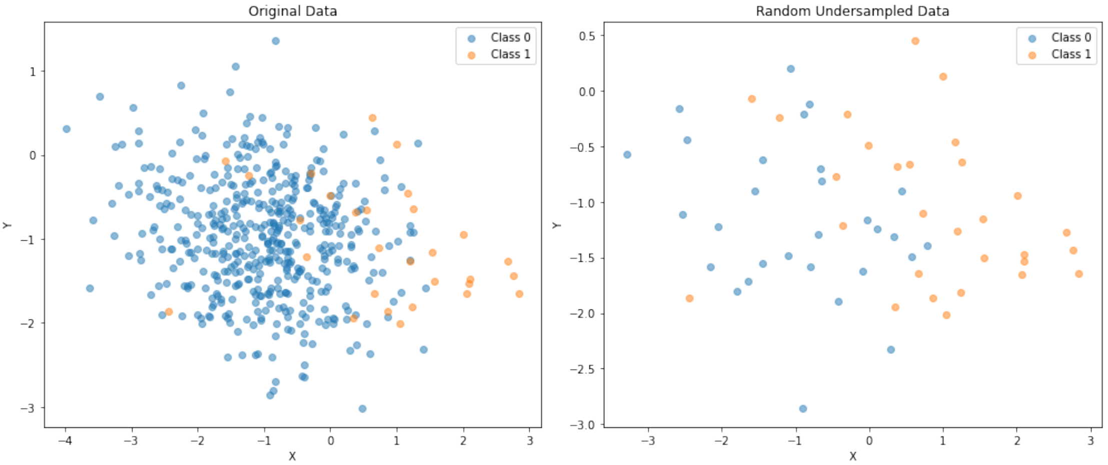
[14] The resampled dataset is used only for training not for validation or evaluation purposes to avoid being exposed to real-life data with the original distribution (Kuhn and Silge, 2023).

emphasizing its significance.

Drummond and Holte (2003) have indicated the advantages of under-sampling over over-sampling. Under-sampling notably reduces the size of the training dataset compared to the original, enabling more efficient model construction. This efficiency arises from two key factors: firstly, the model construction phase for a reduced training dataset demands less time, and secondly, the focus on the rare class enhances modeling effectiveness by minimizing the impact of discarded instances from the normal class, which is deemed less critical for modeling purposes (Aggarwal, 2015).

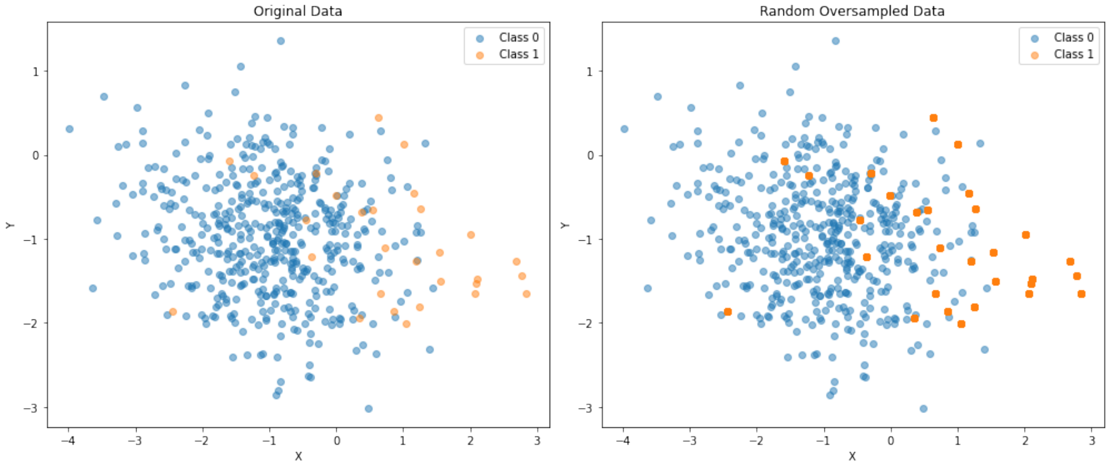
Random Undersampling. Random undersampling (RUS) is done by randomly dropping instances of the majority class. The example provided in Figure 2.4 illustrates such an approach. On the left, we start with 500 randomly generated data points out of which, only 25 belong to the positive class. Next, 450 points belonging to the majority class are dropped at random leaving us with a balanced dataset as shown in the right-hand-side picture. As discussed, this simple method often proves sufficiently effective. Moreover, as we drop information, the risk of overfitting reduces, however, discarding majority class instances potentially leads to the loss of valuable data patterns (Mohammed et al., 2020).

Figure 2.4.: Random Undersampling with Random Data



Random Oversampling. On the opposite of RUS, random oversampling (ROS) randomly selects and later duplicates points that belong to the minority class. Again, Figure 2.5 provides a visual example. Although the two pictures might look the same, the figure on the right contains 450 more observations than the left one. These are the 450 copies of the original points belonging to the rare class which are stacked on top of each other in the graph. Compared to RUS, oversampling retains all available instances, preventing the loss of valuable information. On the other hand, Menardi and Torelli (2014) claim that ROS comes with an increased risk of overfitting especially when the sampling rate increments. Drummond and Holte (2003) argue that oversampling might underperform compared to undersampling with models that do not handle complex datasets well (e.g. Decision trees) as oversampling further increases data complexity.

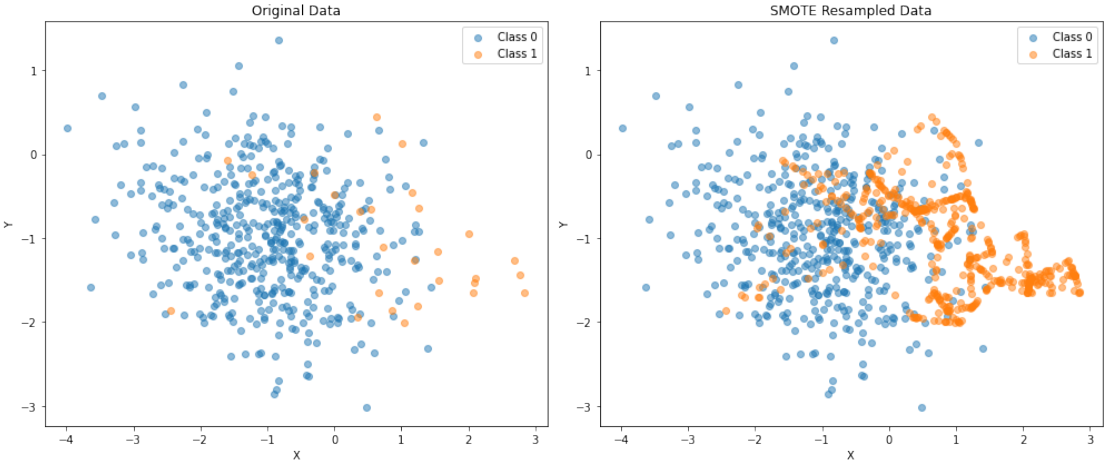
Figure 2.5.: Random Oversampling with Random Data



Synthetic Minority Oversampling Technique. Synthetic Minority Oversampling Technique (SMOTE) is an oversampling technique introduced by Chawla et al. (2002) to address the class imbalance by generating synthetic instances for the minority class. Figure 2.6 illustrates the mechanics of this approach. For each minority class instance, SMOTE selects its k nearest neighbors. It then creates new synthetic instances by interpolating between the chosen instance and its neighbors. The synthetic instances are generated by selecting a point along the line connecting the instance and one of its neighbors and repeating this process for a desired number of synthetic instances. This effectively "fills in" the gaps in the feature space of the minority class, forcing the decision region of the minority class to be generalized more easily (Chawla et al., 2002).

Strategy Selection. Three resampling techniques were introduced above. The question remains; which one is the most suitable for the BAF dataset? Drummond and Holte (2003) suggest RUS to be the best-performing strategy in their experience. On the other hand, Mohammed et al. (2020) show that in their research, oversampling strategies significantly outperformed undersampling. Wongvorachan et al. (2023) argue that oversampling is useful in cases of mild class imbalance, however, tends to overfit as the imbalance get more extreme. In the extreme cases, they suggest RUS as a more viable option. Moreover, they show that combining both over and undersampling techniques might be also beneficial in some cases. To sum up, there is no universal guideline as to which strategy to choose. On the contrary, the most suitable strategy seems to be specific to the data and algorithms used and needs to be established through empirical research. The goal of this is to experiment and provide arguments to which resampling strategy is best suited for the BAF dataset recently published by Jesus et al. (2022a).

Figure 2.6.: Synthetic Minority Oversampling with Random Data



In practice, not only a suitable strategy needs to be determined but also a desired target balance (Wongvorachan et al., 2023). In the empirical part, Python library Imbalanced-learn (Lemaître et al., 2017) which can handle all the three resampling techniques, will be used. In Imbalanced-learn the target ratio (α) is determined by a `sampling_strategy` parameter and is defined as follows:

$$\alpha = \frac{N_{minority}}{N_{majority}} \tag{2.8}$$

where $N_{minority}$ is the number of samples in the minority class and $N_{majority}$ is the number of samples in the majority class (after resampling). That is if set to 1, the dataset is to be sampled to a perfect balance (Lemaître et al., 2017). Optimizing this argument for each of the resampling techniques is also one of the objectives of this thesis.

2.5. Explainable Artificial Intelligence

The need for explainability of ML algorithms in fraud applications was explained in chapter 2.2.1. For this reason, ensuring that all models trained in the empirical part can be sufficiently interpreted is one of the research goals outlined in chapter 1. This section provides a theoretical background for methods used to interpret not only logistic regression and decision trees but also random forest and XGBoost which belong to so called "black-box" models.

2.5.1. Black-box vs. Glass-box Models

The field of explainable AI can be divided into models interpretable by their nature (glass-box models) and those necessitating supplementary explainable AI (XAI) techniques in order to be interpreted (black-box models). Glass-box models inherently incorporate explanations within their decision-making framework. They, therefore, establish transparent and understandable mappings from inputs to outputs, providing users with valuable insights (Holzinger et al., 2022). In the context of this study, the interpretable models are logistic regression and decision trees. When trained, LR fits feature weights which are later used for predictions by plugging desired feature values into the fitted weights. LR, therefore, enables us to see both, the macro setup of the model as well as how individual predictions are made. DTs are probably the most intuitive to interpret as they consist of simple decision rules which can be easily explained to non-practitioners (Breiman, 2017). In contrast, the black-box models, often very complex, do not provide sufficient (or any) insight into their decision-making. For RF and XGBoost, XAI methods like LIME or SHAP need to be used to obtain explainability. These methods are introduced later in this chapter.

2.5.2. Local vs. Global Explainability

Explanations of model behavior can be classified as local or global. Hacker and Passoth (2021) explain that local explanations provide interpretation of individual decisions/predictions made by the model. An example relevant to this thesis might be an individual coming to a bank asking why his credit card application was denied. Local XAI provides us with such individual-specific information. Global explanations, on the other hand, provide an overarching characterization of the entire model's behavior across the whole dataset. It involves extracting insights about feature importance, relationships between variables, and general patterns that the model has learned. Global explainability methods provide a higher-level view of how the model operates and help identify recurring patterns, trends, and biases in its predictions, contributing to a broader understanding of the model's functioning and potential limitations (Ribeiro et al., 2016). In the context of credit card application data, global explainability would help us ensure, for example, that the model uses sensitive features like race or sex fairly across all the target groups and that the model does not discriminate.

2.5.3. Shapley Additive Explanations

SHapley Additive exPlanations (SHAP) values are a concept and methodology in the field of machine learning interpretability that provides insights into how the input features of a model contribute to its predictions. They offer a systematic way to understand the importance of each feature in influencing the model's output, thereby enhancing understanding of the model's decision-making process. SHAP values are grounded in cooperative game theory and seek to fairly allocate the contribution of each feature to the overall prediction (Lundberg and Lee, 2017).

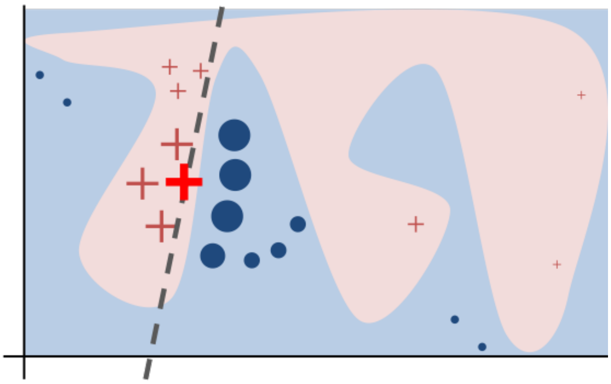
The calculation of SHAP values involves a combination of mathematical principles and computational techniques which are described in detail by Lundberg and Lee (2017). In the context of machine learning, SHAP values quantify the marginal contribution of a feature to the difference between a specific prediction and the average prediction made by the model. This involves considering all possible combinations of features and calculating the average effect of adding or removing a particular feature. As the exact computation for all feature combinations can be computationally intensive, Lundberg and Lee (2017) introduce various approximation techniques, such as KernelSHAP, Shapley sampling values, or model-specific techniques like TreeSHAP or DeepSHAP.

Interpreting SHAP values involves understanding the direction and magnitude of a feature's impact on a specific prediction. A positive SHAP value for a feature indicates that its presence increases the prediction relative to the average, while a negative value suggests the opposite effect. The magnitude of the SHAP value signifies the extent of the feature's influence, with larger values indicating stronger contributions (Lundberg, 2018).

2.5.4. Local Interpretable Model-agnostic Explanations

Local Interpretable Model-agnostic Explanations (LIME) is a method designed to provide insights into the decision-making process of intricate machine learning models offering their local explainability. LIME approximates the model's behavior around a particular prediction (locally) using a more comprehensible interpretable model. Ribeiro et al. (2016) describe that, in practice, LIME creates a new dataset comprised of altered samples and the black box model's predictions. This dataset is then employed to train a simple interpretable model, such as linear regression or decision tree (Molnar, 2022). The interpretable model's objective is to mimic the complex model's behavior within the vicinity of the chosen prediction. Consequently, LIME produces a localized explanation by highlighting the features that exert the most influence on the prediction of interest. This process offers a transparent and intuitive insight into why the complex black-box model arrived at a specific decision for that particular instance.

Figure 2.7.: Intuition behind LIME



Source: Ribeiro et al. (2016)

Figure 2.7 shows a complex decision boundary (blue and pink regions) of a black-box model. This decision boundary cannot be globally approximated by a linear model.

However, in the neighborhood of the point of interest (red cross), a simple linear model can be fitted which provides us with approximate explanations of the decision boundary locally around the particular observation.

2.6. Chapter Summary

First, this chapter introduced the reader to different kinds of credit card fraud while explaining why this research focuses on fraud happening during application for a new bank account. As machine learning is one of the main approaches to detecting fraud, challenges that ML algorithms typically face when trained to tackle fraud were discussed. The main challenges that this research addresses are (1) high class imbalance which makes model training and evaluation difficult, (2) selection of an appropriate evaluation approach (the ROC curve), and finally (3) model interpretability which ensures that models are fair and sensible.

Second, in section 2.3 the ideas of machine learning algorithms to be used in the empirical part of this research were laid out. In this research, two simple models - logistic regression and decision trees - will be used. Moreover, random forest which builds up on the decision tree frameworks is considered together with Extreme Gradient Boosting which is the most complex method used. After explaining the logic behind these models, the paper proceeds by introducing the Reciever Operating Characteristic which is used for evaluation of the above-mentioned algorithms.

Next, section 2.4 dug deeper into the class imbalance problem. Two possible approaches to imbalance handling were discussed. First, cost-sensitive learning deals with imbalance directly in the model by weighting the costs of misclassification differently for different classes making the model pay more attention to the rare class. Secondly, resampling techniques which work at the dataset level, were introduced. The three outlined resampling strategies are (1) random undersampling which randomly deletes observations from the majority class, (2) random oversampling which copy-pastes randomly selected instances of the minority class and (3) SMOTE algorithm which creates synthetic new instances by interpolating between minority observations.

Finally, section 2.5 touched on the explainability challenge mentioned earlier. First, the difference between black-box and glass-box models was explained. The glass-box models are interpretable by their nature whereas black-box models need an additional framework in order to provide sensible intuition to their users. The paper then continued by explaining that local explanations are important to explain model decision-making in the case of individual predictions whereas global explanations provide insight into the overall functioning and reasoning of a model. The chapter ends by introducing two methods for the explainability of black-box models. First, LIME approximates a local decision boundary of a complex algorithm by a simpler, interpretable model. SHAP considers all possible combinations of features and by calculating the average effect of adding or removing a particular feature allows to quantify feature importance both local and global.

The following chapter proceeds with description of the dataset and the methodology.

3. Methodology and Data

This section will introduce and describe in detail the dataset and the methodology used in this research to answer the research questions, i.e., to find a best-performing imbalance-handling technique and compare across ML models to find the one that provides the most accurate predictions of bank fraud.

3.1. Data

This research uses a Bank Account Fraud (BAF) dataset published by Jesus et al. (2022a). The Data section first provides a brief summary of the dataset description based on the original paper and on the accompanying datasheet Jesus et al. (2022b). Later, it discusses insights obtained by data exploration techniques and comments on operations that were undergone before proceeding to the ML modeling part.

3.1.1. Data Description

The BAF data suite comprises six datasets with different kinds of biases. In this research, the original data without any fabricated bias is used. The dataset is generated by Generative Adversarial Network (GAN) trained on a real-world online bank account opening fraud detection dataset which was further anonymized by injection of Laplacian noise prior to the training. The dataset provides thirty informative features which were selected from the original dataset by obtaining feature importances of multiple LightGBMs. The original dataset comprised 2.5M applications which were reduced to 1M to enable better computational efficiency while preserving the original distribution of the fraud in the data. The quality of the generated data and its faithfulness to the original sample were tested by several statistical and distance-based tools as well as by training ML algorithms on the original data and testing them on the generated data to compare their performance on both (Jesus et al. (2022a); Jesus et al. (2022b)).

As mentioned before, the BAF consists of 1M observations, each representing an individual application for a bank account on an online platform. The data was collected during eight months (February to September) with fraud prevalence ranging from 0.85% to 1.5% in a month. The distribution of applications across months is non-uniform with some months containing 10% of observations while others up to 15%. This corresponds to the distribution in the original dataset. The thirty features speak for observed properties of the applications and can be split into three categories:

1. Features obtained directly from the applicant; e.g., `emplyment_status`, `income`, `proposed_limit`, or `age`.
2. Information derived from the provided information; e.g., `phone_number_valid`, `name_email_similarity`, or `has_other_cards`.
3. Data aggregations; different velocity measures based on the bank's data.

A variable called `month` is intended to be used for sampling purposes. The target variable `is_fraud` is a binary variable reaching a value of one if the application was deemed to be fraudulent and zero otherwise. For a detailed description of all features present in the dataset, see Jesus et al. (2022b) or visit BAF Kaggle website.

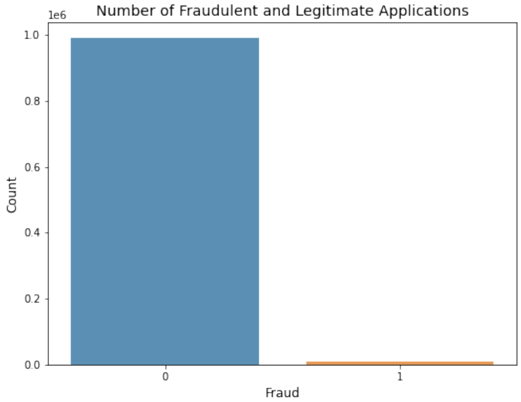
The authors of the dataset also mention several drawbacks to the BAF dataset. The main issue they mention is selective labeling, i.e., labels are known (and therefore the applications are included) only for applications that were successful. Applications rejected due to pre-screening or based on some ML technique are missing in the dataset. However, the authors claim that in the case of the original dataset, only a minimal pre-screening was conducted (e.g. AML regulation compliance) (Jesus et al., 2022a).

3.1.2. Data Exploration

Data exploration brought several deeper insights into the dataset:

1. *Categorical features.* The dataset contains five categorical variables (`payment_type`, `employment_status`, `housing_status`, `source`, and `device_os`) with overall 26 distinct values. These variables need to be OneHot encoded as a part of preprocessing.
2. *Low correlation across numeric features.* There are no two variables with a Pearson correlation coefficient bigger than 0.61. The majority of the features have correlation coefficients close to zero. The conclusion of this analysis is that there are no redundant highly correlated variables in the data and it is, therefore, advisable to keep all of them for ML. See Appendix A.1 for a correlation matrix of all features in BAF.
3. *Five variables contain missing values.* Jesus et al. (2022b) explain that missing values from the original dataset translate to negative values in the BAF. Any negative value thus represents a missing value that needs to be encoded. After analyzing the data, missing values were identified in five numeric variables. Handling of missing values is further discussed in section 3.1.3.
4. *High class imbalance.* The BAF dataset contains 1M instances of which only 11K (i.e. 1.1%) are labeled as fraudulent. Figure 3.1 illustrates such imbalance graphically. Challenges posed by high class imbalance were extensively discussed in Chap 2. The approach undertaken by this research to tackle extreme class imbalance is further discussed in section 3.2.

Figure 3.1.: Dataset Imbalance



5. *Numeric variables have significantly different scales.* The scales of numeric features differ quite a lot across variables. Some come on a scale from zero to one, some, on the other hand, have a maximum of up to 16.7K. This poses a challenge to SMOTE as it uses KNN, a distance-based algorithm, to identify neighboring points. In order to calculate distance consistently, all features should be scaled to a uniform scale Provost and Fawcett (2013).

6. *One column is constant.* Variable `device_fraud_count` contains the same value for all observations. It, therefore, doesn't contain any useful information for training ML models. For this reason, the variable was dropped.

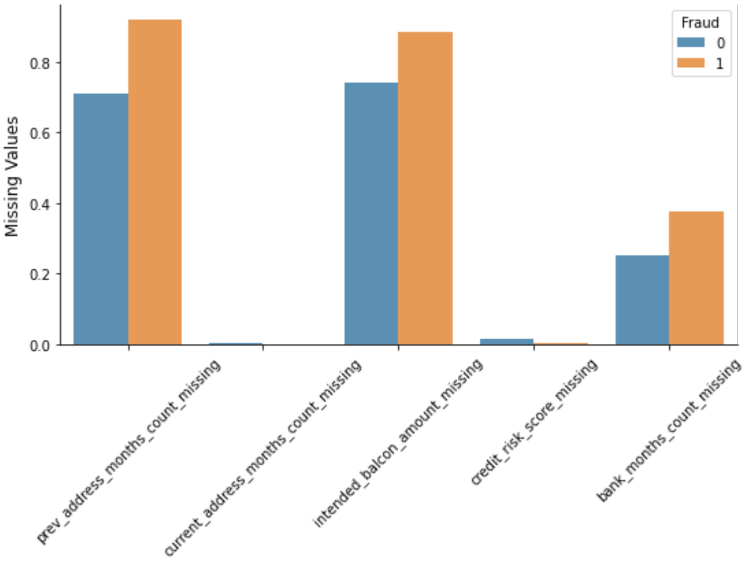
3.1.3. Feature Induction and Missing Values Handling

As mentioned earlier, five features contain missing values. Figure 3.2 provides a visual insight into these five variables. In two features, about 75% of the data is missing, one feature lacks around 25% of instances while in last two have only a few missing values. The figure, however, also illustrates that in the three variables with many missing values, fraudulent applications exhibit significantly more missing values than legitimate transactions. This suggests that the fact that a variable value is missing carries some explanatory value. For this reason, dummy variables that take the value of 1 if a value is missing and 0, if the numeric value is present, were induced in the dataset. This proved to be a favorable strategy as two of these dummies[15] proved to be relatively important features for XGBoost, RF, and LR.

After the dummy variables for missing values were encoded, missing values were imputed as a median value of the known sample. Appendix A.2 provides a visual illustration of the effect of such an operation on the feature distribution.

[15] For intended balcony amount and for month count at a previous address.

Figure 3.2.: Frequency of Missing Values by Class



3.1.4. Preprocessing

Handling missing values is not the only desirable preprocessing step. As mentioned in section 3.1.2, five columns contain categorical values. These need to be OneHot (dummy) encoded for two reasons: (1) Logistic regression cannot handle categorical data. Neither does (2) SMOTE which used a distance measure to identify neighbors and numeric values are a must for distance calculation.

Moreover, as SMOTE relies on distance measures, it is recommended to scale numerical variables to a common range (Provost and Fawcett, 2013). A MinMax transformation[16] specified by equation 3.1 is used for this purpose. By using MinMax Scaler, all numeric features are projected into a zero-one range which is particularly useful as the dataset contains boolean features that work with the same scale. Moreover, having features scaled to a uniform scale makes model interpretation (especially LR) easier.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

The final preprocessing step is rebalancing the dataset to face the extreme class imbalance by using over or undersampling techniques. Although these definitely fall under the preprocessing stage, resampling will be discussed later in the Methodology part as it is a critical point of this research.

[16] See the Scikit Learn documentation of MinMax Scaler.

3.1.5. Sampling

The empirical part of this research can be, in general, divided into three parts: (1) the part which seeks to find the optimal resampling technique and tune model hyperparameters through repeated cross-validation of each model and (2) the final training phase, using the results from part 1 and all the data available builds the final models to be tested and deployed and finally (3) explainability part. Different samples of the data will be needed for the first two phases, for explainability the same data as for part 2 is needed. The month variable is used to split the data according to our needs.

1. *"Cross-validation dataset".* The Cross-validation (CV) dataset is built using the first six months of the data (same as the Full training set). The last two months are excluded as the model should never see any of the data used for final evaluation prior to it. Moreover, to make the repeated experiment computationally achievable, we randomly drop half of the observations from the first six months. As a result, the CV dataset contains approximately 400K observations. To ensure the same class balance as in the initial data, stratified sampling[17] was used to drop the 50% of the data.

2. *"Full training set".* The final models are to be trained on all data from the first six months, i.e., where `month` ≤ 6. We call this dataset "FT set" for short.

3. *"Full testing set".* The models trained on the Full set are to be tested and evaluated using the last two months of the data, i.e. `month` > 6. We call this dataset "FV set" as a shortcut for the Full Validation set.

3.2. Methodology

Section 3.1.5 briefly touches the three phases of the empirical research. This section gives more detail on them. First, the strategy of establishing which resampling technique is best suitable for each model together with procedure of hyperparameter tuning is discussed in chapter 3.2.1. Next, section 3.2.2 discussed the training and evaluation of the full models together with the probability treshold tuning. Finally, application of SHAP for XAI is examined in the last section (3.2.3) of this chapter.

3.2.1. Phase 1: Resampling Techniques Optimization, Hyperparameter Tuning

The goal of Phase 1 is to identify the best-performing resampling technique (and the extent of it, i.e., the optimal target balance) and hyperparameter set of each algorithm. The diagram in Figure 3.3[18] and the pseudocode on the next page are intended to provide a visual aid for understanding the process described in this section. The strategy to

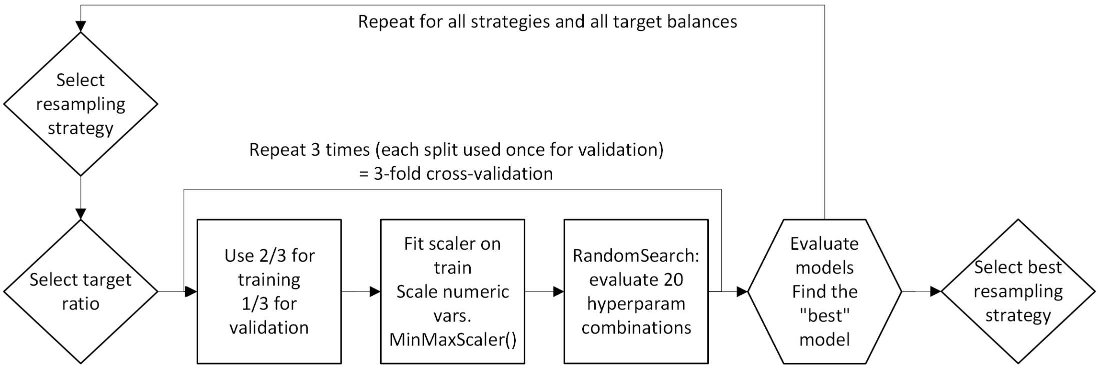
[17] Scikit Learn function `train_test_split` with a `stratify` argument was used for this purpose.

[18] Diagrams were created in Microsoft Visio.

achieve the above-stated goal follows.

Consider only one of the algorithms. First, from a pre-defined space of selected hyperparameters[19] (see Appendix A.3), randomly select twenty combinations of hyperparameter values[20]. Now, run a nested 3-fold cross-validation (CV)[21] to find the best model specification (hyperparameter combination) without any resampling taking place - a baseline model and its performance. Next, select a resampling strategy and a target ratio (i.e., how much to resample). Run the 3-fold cross validation again, using the same twenty combinations of hyperparameter values (use the same random seed), however, this time, apply the resampling technique to the two folds which are used for training. Results of these twenty CVs give us the highest achieved AUC ROC and the set of hyperparameters that were used to achieve it for the used resampling technique and the target ratio. Next, repeat the same procedure for all target ratios of the given resampling technique and later for the remaining sampling strategies. Compare the results to see which resampling strategy achieved the best results. Finally, repeat the whole procedure also for the remaining models to allow for comparison across models.

Figure 3.3.: Phase 1: Workflow Diagram


Repeat for all strategies and all target balances
Select resampling strategy
Repeat 3 times (each split used once for validation)
= 3-fold cross-validation
Select target ratio
Use 2/3 for training 1/3 for validation
Fit scaler on train Scale numeric vars. MinMaxScaler()
RandomSearch: evaluate 20 hyperparam combinations
Evaluate models Find the "best" model
Select best resampling strategy

By repeating the cross-validation for each combination of the resampling strategy and the target ratio using the *same twenty combinations of hyperparameter values*, we ensure two crutial properties. First, using the same twenty combinations allows for comparison across the target ratios and target ratios. If different combinations were allowed, the best model performance obtained through CV might be caused by the choice of the hyperparameters instead of the choice of resampling technique. Second, it might be tempting to use the same set of hyperparameters obtained in the first CV to save resources. However, the optimal hyperparameter set might depend on the resampling strategy used. Therefore,

[19] For all models, hyperparameter `class_weight` with options `balanced` and `None` is included to include cost-sensitive learning (chapter 2.4) in the research.

[20] Scikit Learn's RandomizedSearchCV library was used.

[21] See, for example, Provost and Fawcett (2013), chapter 5 or Kuhn and Silge (2023) for a detailed introduction to cross-validation.

Pseudocode of Phase 1

```
Data: Cross-validation dataset (described in section 3.1.5)
Result: Best performing target balance, resampling technique, and
        hyperparameter set of each algorithm

foreach model in [LR, DT, RF, XGB] do
    foreach resampling technique in [RUS, ROS, SMOTE] do
        foreach target ratio in [0.0112, .0168, 0.025, 0.05, 0.075, 0.1, 0.2, 0.5, 1]*
          do
            Run RandomizedSearch** over a selected space of hyperparameters.
        end
        Output: best-performing target balance for the given technique;
    end
    Output: best performing resampling technique for the given model;
end
Output: best-performing hyperparameter set of the model
```

* For RF and XBG, a reduced list [0.0112, 0.025, 0.05, 0.1, 1] is considered due to computational limitations.

**Randomized search parameters: stratified 3 fold cross-validation, 20 iterations, evaluated by AUC ROC. In each fold, MinMax Scaler and the corresponing resampling technique is applied to the training set.

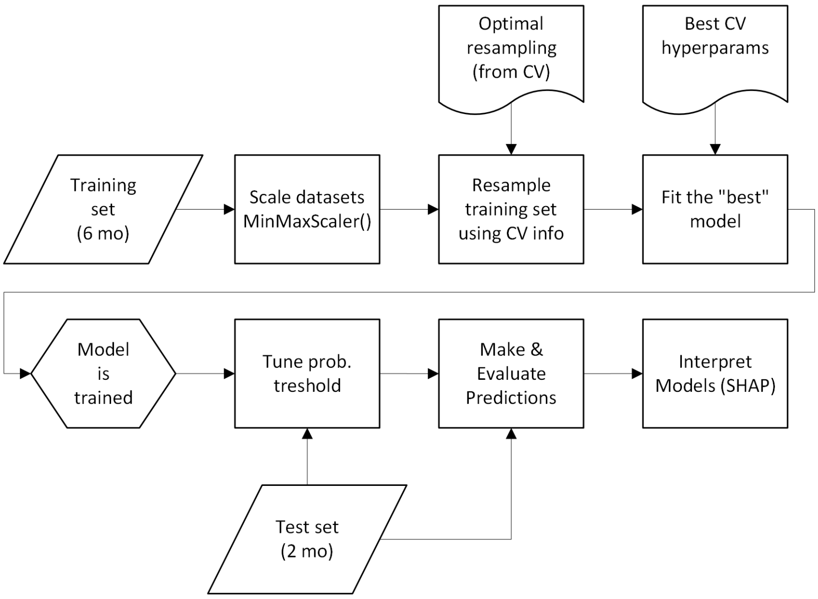
some freedom of choice should be given to the Randomized Search to ensure that for each resampling strategy, the best model specification was obtained.

Finally, as discussed in section 3.1.4, data is to be scaled using the MinMax scaler. According to Brownlee (2020a), scaling must happen inside each fold of the cross-validation in order to avoid data leakage. If data was scaled before the CV, minimum and maximum of the whole dataset would be used to fit the scaler. This, however, is wrong because when scaling, we can only use the "known information", i.e., only information coming from the training set. Since training and test sets differ in each fold of the CV, the scaler has to be fitted and applied over and over in each iteration. The same holds for imputing missing values and resampling. When, for example, oversampling, new instances are generated based exclusively on the training set.

3.2.2. Phase 2: Final Model Training, Tuning, and Evaluation

Once the best-performing hyperparameters together with ideal resampling strategies are determined for each algorithm, this information is used in Phase 2 to train the model on the full (resampled) training set (containing 6 months of data). These models then need to be tested and fine-tuned by adjusting the probability threshold such that the models achieve the desired 5% FPR. The diagram in Figure 3.4 gives a visual overview of Phase 2. Notice that this part uses a different data sample than Phase 1 - here, FT set described in section 3.1.5 is used for fitting the algorithms and FV set for evaluation.

Figure 3.4.: Phase 2: Workflow Diagram



First, missing values are imputed to both FT and FV datasets using the median value based on the training data alone as instructed by Brownlee (2020a). Next, all the data was scaled using the MinMax scaler, again, using minimum and maximum values obtained from the FT set only. These two preprocessing steps are common to all models. Resampling, on the other hand, is specific to each model as at this stage, only the most successful technique from Phase 1 is used for each model. It might therefore happen, that each algorithm requires a completely different resampling approach.

Second, we fit each model using the respectively-resampled FT set and the "optimal" set of hyperparameters obtained in Phase 1. Once the training is done, we make all models predict probabilities for the FV test set. Moreover, by using knowledge from section 2.3.5, the Reciever Operating Characteristic curve for all the models can be constructed by varying the probability threshold. Figure 3.5 provides an intuition for how ideas probability threshold tuning interact with the ROC on an example of logistic regression (its sigmoid function). As explained in chapter 2.3, when a model predicts a probability lower than the threshold, it makes a negative prediction, i.e., predicts the application to be legitimate. Therefore, as the threshold decreases, we make the models predict more applications to be of fraudulent nature. This translates into increasing TPR as we correctly identify more fraud. However, not all the newly identified applications are fraudulent in reality, thus also increasing FPR. In general, decreasing the probability threshold makes the model more sensitive toward the positive class and makes the model move along the ROC curve in the top-right direction. In the case of this research, the goal of this exercise is to calibrate the probability threshold such that we get the highest TPR at (at most) 5% FPR as suggested by Jesus et al. (2022a).[22]

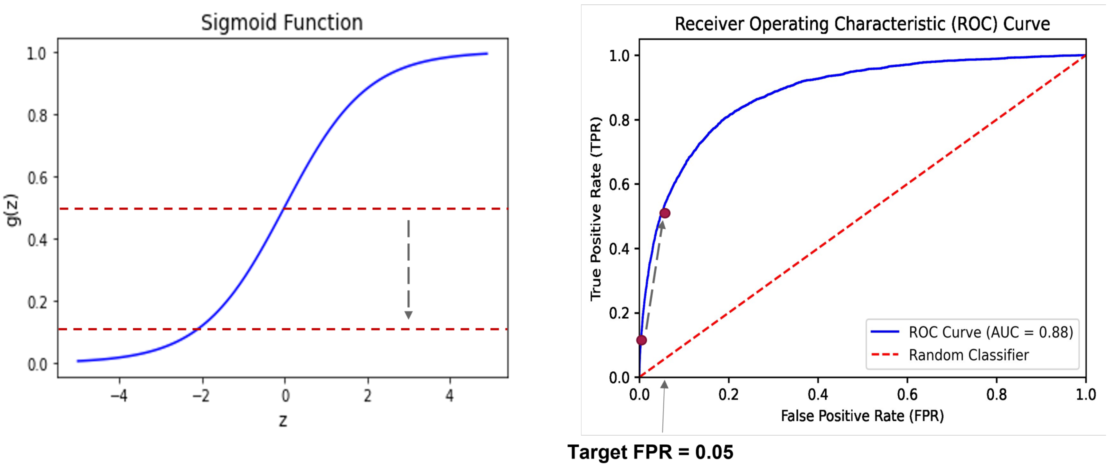
Finally, when the desirable probability threshold is determined, it is used to make final predictions of each of the algorithms which can be evaluated on the FV test set. Once evaluated, having the TPR at 5% FPR metric enables for comparison of performance across the algorithms. To allow for a comparison of the performance of an algorithm with and without the selected resampling technique, baseline algorithms without any resampling strategy were also trained, tuned, and evaluated using the same methodology. Therefore, at the end of this part, an overall of eight different models are trained and compared. Four models where only hyperparameters were tuned ("baseline" models) and four where optimal resampling was undergone ("resampled" models). This was the main body of this research, now, one last step remains; to explain and interpret the trained models.

3.2.3. Phase 3: Explaining the Models; Feature Importance and SHAP

Phase 3 can be separated into two parts; global and local explainability as all models enable for global feature importance, however, for local explanations of RF and XGBoost,

[22] Jesus et al. (2022a) explain that fraud detection clients commonly establish this measure, as it achieves an equilibrium between identifying fraud (TPR) and minimizing customer attrition. Each false positive represents a customer who might switch their bank due to an erroneous fraud designation.

Figure 3.5.: Phase 2: Probability Threshold Tuning



SHAP needs to be implemented.

First, to explain logistic regression, the coefficient load can be directly used as a feature importance as all variables were scaled to the same value range (Brownlee, 2020b). Moreover, by dividing each coefficient by the sum of the absolute values of all the coefficients, we get a relative feature importance. As discussed in section 2.5, in the case of logistic regression, its fitted coefficents provide both local and global explanations.

Second, local explanations of decision trees are quite forward, it suffices to walk through the tree from the root to leafs to see how a prediction is made. However, according to Brownlee (2020b), decision trees provide importance metric based on the reduction in the criterion (Gini or Entropy) used to split nodes.[23]

Finally, when modeling random forests and XGBoost, a `feature_importances_` property can be retrieved easily to provide global explainability based on the same approach as in the DT case. However, for local explanation, SHAP values are retrieved for the desired instances in data. Moreover, calculating many SHAP values and taking an average over their absolute values enables another approach to global explainability. To provide better intuition, for each model, graphics provided by the SHAP library[24] will be generated. Additionally, we will use LIME[25] to provide a different angle to local explanations.

[23] See Brownlee (2020b) for an introduction to feature importances with Python for all the models discussed in this research.

[24] See `https://shap.readthedocs.io/en/latest/index.html` for complete documentation.

[25] We use the InterpretML and LIME packages to implement the LIME method in Python. See `https://interpret.ml/docs/index.html` and `https://lime-ml.readthedocs.io/en/latest/lime.html#module-lime` for documentation of these packages.

3.3. Chapter Summary

To conclude, this chapter first discussed the BAF dataset and its characteristics (section 3.1). First, a brief summary of the dataset description provided by Jesus et al. (2022a) was given. Second, a data exploration was conducted and several insights to the data were discussed such as missing values, high class imbalance or need for scaling. Next, handling of missing values and scaling approach were debated. Finally, the last part introduced sampling of the data into three different datasets used in different stages of the data mining process.

Finally, the methodology part (section 3.2) introduced three phases of the modeling process. The goal of the first phase is to obtain the best hyperparameter set together with an optimal resampling technique for each algorithm through repeated randomized search. The second phase uses the results of Phase 1 to train the definite models on the whole training dataset. After the final models are trained, their probability thresholds are tuned to reach the desired 5% FPR which enables a final model evaluation. Last, Phase 3 includes explaining the models both globally and locally using feature importance (for global explanations) and SHAP values (for local explanations of RF and XGBoost).

4. Results

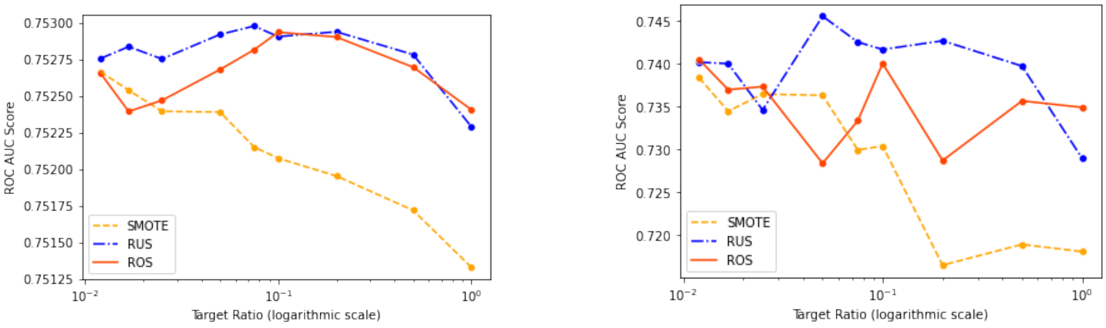
This chapter discusses the results of the research described in chapter 3 phase by phase. First, section 4.1 shows how well each resampling technique performs with respect to the target ratio. This information enables us to choose the best-performing resampling strategy. Moreover, it also presents the best hyperparameter set found through the Randomized Search for each algorithm. Knowledge obtained in this part is then used in Phase 2 to resample the full training (FT) set and train the final models. Results of that are presented in section 4.2 which also compares the performance of the models trained on resampled data to the baseline models (based on the original data). Moreover, this section also provides the results of the probability threshold calibration necessary to tune the model to achieve the maximum Recall at 5% FPR (the desired evaluation metric). Next, evaluating all the models using the full test set (FV set) allows us to compare all the models to see which models performed the best overall. Finally, section 4.3 gives examples of how to interpret and explain all four kinds of ML algorithms used in this research.

4.1. Phase 1: Resampling Technique Selection

The output of this section is, for each algorithm, the best resampling method together with the optimal target ratio (α) to which resampling should be done. Note, that resampling is evaluated based on the tuned set of hyperparameters. That is, for each resampling strategy performance you can see in Figures 4.1 and 4.2, a different "best" set of optimized hyperparameters might be chosen. After determining which resampling strategy performs the best overall, a tuned hyperparameter set that was used to obtain these results is reported in this section as well. Finally, models after resampling are compared to their baseline versions in terms of their hyperparameters and runtime.

4.1.1. Logistic Regression

Resampling Methods and Target Ratios. Two of the resampling strategies seem to work out in the case of LR; RUS and ROS. In the case of RUS, the area under the ROC curve stays approximately the same for target ratios from 0.012 to 0.2 and then drops significantly. For ROS, model performance is getting better until it reaches α of 0.1, then, again, decreases. For both RUS and ROS, the optimal target ratio seems to be moving around 0.05 to 0.2. Even though both RUS and ROS perform almost identically, since RUS saves computational resources by decreasing dataset dimension, it is identified as the preferable resampling strategy. As the results suggest, the optimal α for RUS moves between 0.05 to 0.2. Again, since more undersampling saves training time, the target ratio of 0.1 (good performance for both RUS and ROS) is going to be applied in Phase 2. Applying RUS @ 0.1 α means dropping approx. 90% of the legitimate transactions and translates into approx. 75% decline in runtime when running the Randomized Search.



(a) Logistic Regression

(b) Decision Tree

Figure 4.1.: Model performance as a function of resampling strategy and target balance Note: x-axis on a logarithmic scale; x-values (α): 0.012, 0.0168, 0.025, 0.05, 0.075, 0.1, 0.2, 0.5, 1.0.

Hyperparameter Tuning. Table 4.1 presents the results of hyperparameter tuning. The only difference between the model specifications is in C (inverse of regularization strength) meaning that when RUS is applied, more regularization is required in the training phase. Interestingly, the class_weight parameter is "None" for both models suggesting that cost-sensitive learning (see section 2.4.1) does not add positively to model performance and initial class weights should thus be used.

Hyperparameter Values (LR)	C	class_weight	penalty	solver
Baseline Model	1.33	None	None	saga
Resampled Model (RUS @ 0.1)	0.18	None	None	saga

Table 4.1.: Best Hyperparameter Sets for Logistic Regression

4.1.2. Decision Tree

Resampling Methods and Target Ratios. As presented in Figure 4.1 (b), similar to Logistic Regression, SMOTE seems to steadily underperform RUS and ROS in the case of Decision Tree as its performance decreases steadily. ROS's performance is quite volatile in the sense that there is only one target ratio (0.1) performing similarly well as the initial setup whereas in all the other cases ROS underperforms the imbalanced model. Only RUS seems to improve AUC ROC at the target balance of 0.05. The RUS strategy at 0.05 α is, therefore, identified as the most suitable resampling approach in the case of a decision tree.

Hyperparam. Values	criterion	class _weight	max _depth	max _features	min _leaf	min _split
Baseline Model	gini	None	8	sqrt	8	6
Resampled Model (RUS @ 0.05)	gini	None	8	log2	9	2

Table 4.2.: Best Hyperparameter Sets for Decision Tree

Hyperparameter Tuning. In the case of a decision tree, a model of the same depth and same criterion resulted from the Randomized Search. However, different values for min_split and min_leaf were obtained. What is most important, however, is that neither one of the two models uses cost-sensitive learning as both models perform better when class_weight is set to "None".

4.1.3. Random Forest

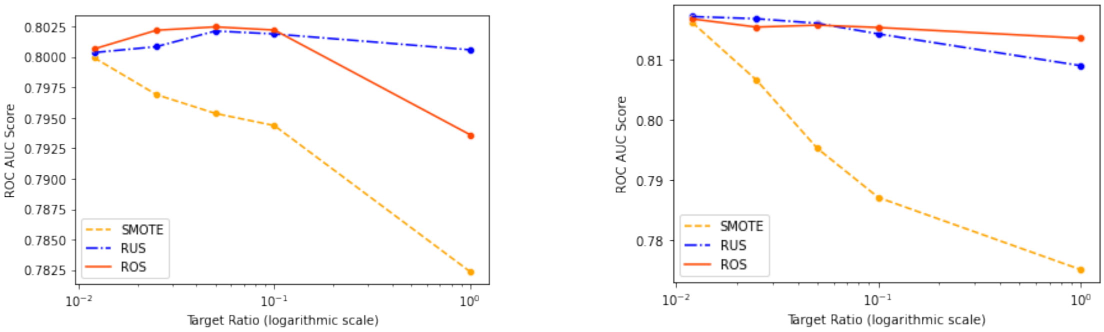
Resampling Methods and Target Ratios[26]. Random Forest (Figure 4.2 (a)) follows the LR and DT example with SMOTE steadily decreasing model predictive performance. In the case of Random Undersampling, one can see that for all the tested target ratios, the model's performance stagnates on the same level. The same holds for ROS with the exception of a target ratio of 1 at which the performance decreases. Since ROS and RUS show practically the same pattern and values, we use the same reasoning as with logistic regression (saving computational resources) and choose RUS @ 0.05 α to be the resampling strategy implemented in the later phases of the research. Indeed, performing RUS @ 0.05 α (i.e. randomly dropping 80% of the majority class) decreases the runtime of the final training by approx. 50% while ROS would increase it with respect to the baseline model.

Hyperparam. Values Random Forest	max _depth	n_estim	max_ features	criterion	class _wight
Baseline Model	8	200	log2	entropy	None
Resampled Model (RUS @ 0.05)	8	200	log2	entropy	None

Table 4.3.: Best Hyperparameter Sets for Random Forest

Hyperparameter Tuning. The hyperparameters of the random forest are the same for the baseline and resampled models as both forests consist of 200 decision trees of max 8 layers and entropy as a splitting criterion. Same as with LR and DT, no cost weighting is applied as the class_weight parameter is established to be "None" through the randomized search.

[26] Note that due to computational constraints, in the case of Random Forest and XGBoost, fewer target ratios (0.012, 0.025, 0.05, 0.1, and 1) were considered for the analysis.



(a) Random Forest

(b) XGBoost

Figure 4.2.: Model performance as a function of resampling strategy and target balance Note: x-axis on a logarithmic scale; x-values (α): 0.012, 0.025, 0.05, 0.1, 1.0

4.1.4. XGBoost

Resampling Methods and Target Ratios.

As one can see in Figure 4.2 (b), XGBoost results are almost identical to the ones of an RF. SMOTE fails to perform while ROS and RUS exhibit a similar slightly decreasing pattern. For this reason, the same resampling strategy and target ratio (RUS @ 0.05 α) as for random forest are selected to be applied in the upcoming phase of the research. Again, training the final model on the randomly undersampled data takes approximately one-fifth of the time needed to train the baseline model.

Hyperparam. Values XGBoost	max _depth	n_estim	learning _rate	sub sample	colsample _bytree	scale_pos _weight
Baseline Model	2	100	0.3	1	0.8	1
Resampled Model (RUS @ 0.05)	4	150	0.1	0.8	0.6	1

Table 4.4.: Best Hyperparameter Sets for XGBoost

Hyperparameter Tuning. XGBoost uses similar, yet slightly different hyperparameters[27] to the previous models. Table 4.4 presents the most important parameters tuned during Phase 1. The most interesting one for this research is the scale_pos_weight parameter which controls the cost-weighting, i.e., whether cost-sensitive learning is applied or not. Since this parameter is 1 for both, the baseline and resampled models, no cost-sensitive learning was applied for XGB in agreement with all the other tested algorithms. However, apart from that, XGB exhibits the biggest difference between the baseline and the resampled model. Random undersampling seems to allow the model to identify complex patterns as it builds more and deeper trees but at a more conservative learning rate.

[27] See XGBoost Documentation for a detailed hyperparameter description.

Nonetheless, both models' performance is almost identical as can be seen in Figure 4.2 (b).

4.1.5. Phase 1: Conclusions

Looking at the results of Phase 1, we can derive three main overarching patterns. First, SMOTE performed poorly for all the evaluated models. A possible explanation for this is that SMOTE invents new data points outside the existing data patterns. In the case of this dataset, it seems that SMOTE fails to strengthen the existing patterns in the data and instead introduces fraudulent transactions where they wouldn't occur naturally. That is, our models overfit when applied to SMOTE-oversampled datasets.

Second, the go-to resampling strategy is Random Undersampling for all the models. This is in line with, e.g., Wongvorachan et al. (2023) who suggest that for datasets with extreme imbalance, ROS tends to overfit. In the case of Logistic Regression and Decision Tree, RUS outperformed ROS at majority of the target ratios and for this reason, it was selected to be the optimal resampling strategy to be applied in Phase 2. In the case of Random Forest and XGBoost, RUS and ROS perform very similarly. However, RUS was chosen for both models over ROS since RUS decreases data dimensionality and thus also the runtime of the algorithms which is advantageous since both RF and XGB are computationally more demanding than simple models like LR and DT.

Finally, we argued that for none of the models (baseline and resampled) cost-sensitive learning was chosen by Randomized Search. This is an important result since one of the research goals is to suggest the best ways to handle the class imbalance in the BAF dataset. Cost-weighting was one of the methods we wanted to evaluate in this research and our results show that cost-sensitive learning is not the ideal solution to the class imbalance in the case of the BAF dataset.

4.2. Phase 2: Final Model Training

This section presents the outputs of the Phase 2 as described in section 3.2.2. Phase 2 trains final models on the first 6 months of the data using the resampling strategies selected in Phase 1 and tests them on the last 2 months of the data. For better comparison, it compares these final "resampled" models to the baseline models which are trained on the full first 6 months of the data prior to resampling. The results of Phase 2 are the ROC curve which is constructed by iteratively adjusting the probability threshold of all final models, the probability threshold which was optimized to maximize TPR at 5% FPR and, finally, the evaluation metric (TPR at 5% FPR) itself. Last, all models are compared based on this evaluation metric to find the overall best final model.

4.2.1. Logistic Regression

Logistic regression performed surprisingly well compared to other models considering the fact that is one of the less complex methods introduced here. Both the baseline and the resampled LR models achieved AUC ROC of 0.88 which is as good as the same metric of the XGBoost model which is way more complex and computationally demanding. The probability threshold was tuned to 0.25 after RUS and to 0.03 in the baseline model. RUS thus slightly decreases the need for probability threshold tuning making the default model more sensitive towards the minority class. Both models achieved a TPR of 51% at 0.05 FPR. That is, at the cost of falsely denying 5% of legitimate applications, LR models manage to correctly identify more than half of fraudulent attempts. The resampled model performs equally well as the baseline model, however, in a significantly (80%) shorter training time. Applying RUS is thus advantageous compared to not taking any resampling steps.

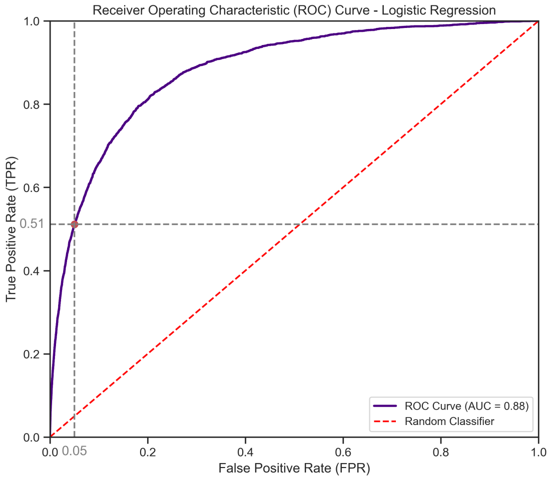


Figure 4.3.: ROC curve of the final Logistic Regression model;
Prob. threshold @ 5% FPR: 0.25, TPR reached: 0.51

4.2.2. Decision Tree

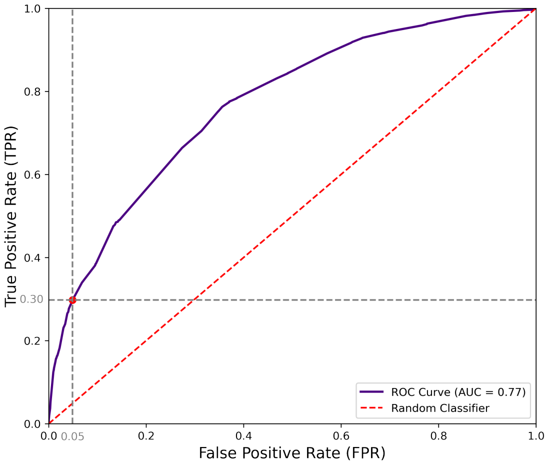


Figure 4.4.: ROC curve of the final Decision Tree model;
Prob. threshold @ 5% FPR: 0.17, TPR reached: 0.30

Decision Tree performed the worst of all the models with AUC ROC being 0.78 and 0.77 for the baseline and resampled models respectively. Even though the AUC ROC curve is almost equal for both models, after calibrating the probability threshold (0.04 for the baseline model and 0.17 for the resampled model) we get slightly different values for Recall at the desired FPR level. In practice, this means that the ROC curve has a slightly different shape even though the AUC is almost the same. The baseline model managed to achieve 29% TPR at 0.05 FPR while the model after resampling achieved 30%. This shows that random undersampling enabled the decision tree to be slightly more precise at low FPRs. Note, however, that the resampled model is preferable as long as only a few false positives are acceptable. The initial model could, on the other hand, perform better in areas with high FPR. Note that the fact that the resampled model performs slightly better might potentially be due to a different set of hyperparameters and not solely due to data undersampling. The statement that the resampled DT is more precise at low FPRs is supported by the high probability threshold meaning that we didn't have to adjust the model's initial setup as much as in the baseline case to flag mode fraudulent cases.

4.2.3. Random Forest

As one can expect from the RF results in Phase 1, the baseline and the resampled random forests performed almost equally. In Phase 1, the resampling strategy choice had very little impact on the AUC ROC. The results of Phase 2 follow the same logic with both models achieving an AUC ROC of 0.87 and a TPR of 0.477 after the probability threshold calibration. However, similar to the decision tree and logistic regression, undersampling the data allowed for a higher probability threshold (0.12) compared to the initial state

(0.03). Even though the Random Forest, an ensemble of 200 decision trees, outperforms a primitive decision tree, it did not manage to achieve better TPR @0.05 FPR than logistic regression which is considered to be a more straightforward algorithm.

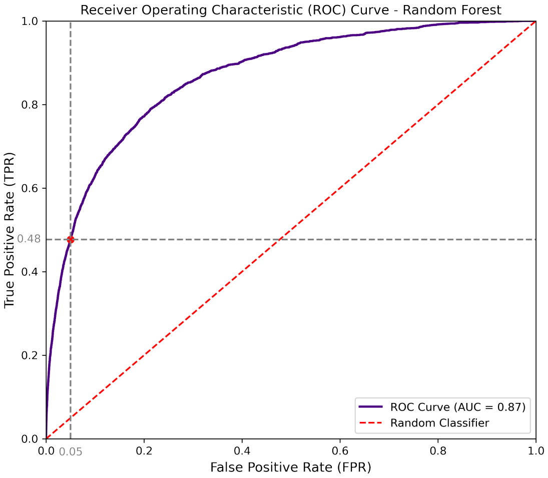


Figure 4.5.: ROC curve of the final Random Forest model;
Prob. threshold @ 5% FPR: 0.12, TPR reached: 0.48

4.2.4. XGBoost

XGBoost shows the most promising results for all trained models with the area under the ROC curve of 0.89 for the resampled and baseline models. Both models also perform equally well in terms of ROC and in terms of Recall @ 0.05 FPR of 54% which is the best achieved result overall. XGBoost therefore follows the same pattern as the other three models; the evaluation performance is equal for both models, however, after RUS all the algorithms require less probability threshold calibration as they are more sensitive towards the minority class by their setup.

4.2.5. Phase 2: Conclusions

This section presented the results of Phase 2, i.e., of model final training and evaluation with the optimal resampling strategy established in Phase 1 and without any resampling applied. Table 4.5 gives an overview of all the final models developed. The overall best model in terms of TPR at 5% FPR is XGBoost which would manage to correctly identify more than 54% of fraudulent applications at the given FPR. Surprisingly, logistic regression ranks as the second best with a Recall of 51% followed by Random Forest with a TPR of 0.48. The decision tree ranks clearly last with a significantly lower Recall (29 to 30 percent) compared to the other models. For LR, RF, and XGB, resampling with RUS does not influence the evaluation results, however, decreases training time significantly. Moreover, RUS decreases the need for probability threshold calibration as for

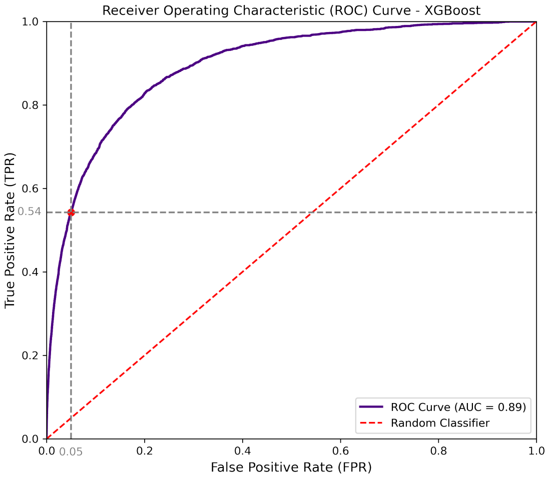


Figure 4.6.: ROC curve of the final XGBoost model;
Prob. threshold @ 5% FPR: 0.17, TPR reached: 0.54

all resampled models, the desired threshold is higher than in their baseline counterparts. In the case of a decision tree, RUS not only increases the prob. threshold but also slightly increases the model TPR @0.05 FPR by one percentage point.

For all the models, the resampling approach of choice was random undersampling to target ratios moving from 0.05 to 0.1 (i.e. dropping 80 to 90% of the observations belonging to the majority class). As mentioned by Brownlee (2020c), RUS may perform well when there are enough minority observations yet class imbalance present, which, given the relatively big sample, seems to be our case too. Interestingly, RUS had a different effect on the evaluation results of different models. Such results align with the findings of, for example, Akyol and Atila (2020), who show that resampling strategy can have a different effect on different ML algorithms. LR, RF, and XGBoost seem robust to the selected resampling strategy yielding similar results for the baseline and resampled models. However, in such cases, it might be advantageous to perform RUS to save computational resources. Next, the decision tree's performance increased after undersampling the data. One might be disappointed that RUS did not increase model performance compared to the baseline models. Yet, as shown by Akyol and Atila (2020), resampling data might even lead to underperformance of ML algorithms after resampling compared to their baseline versions. Considering that resampling did not hurt model performance and at the same time decreased the computational complexity of the analysis, it might be still useful to apply RUS to the BAF dataset.

Model	Resampling Method	Target Ratio	ROC AUC	Probablity Threshold	TPR at 5% FPR	Overall Rank
Baseline Models						
LR	-	-	0.88	0.03	0.51	4
DT	-	-	0.78	0.04	0.29	8
RF	-	-	0.87	0.03	0.48	6
XGB	-	-	0.89	0.04	0.54	2
Resampled Models						
LR	RUS	0.10	0.88	0.25	0.51	3
DT	RUS	0.05	0.77	0.17	0.30	7
RF	RUS	0.05	0.87	0.12	0.48	5
XGB	RUS	0.05	0.89	0.17	0.54	1

Table 4.5.: Performance of the Baseline Models and Models after Resampling

4.3. Phase 3: Explaining the Models

This section provides examples of how to explain the models trained in the previous phases of the research. First, a basic feature importance is introduced as a method of interpreting logistic regression. Feature importance framework can be also obtained for all the other algorithms to provide insight into the global mechanics of all the models. We provide a comparison of feature importance for all resampled models in section 4.3.5. Second, a decision tree trained in Phase 2 is visualized and described. Next, in the example of Random Forest, global and local explanations using the SHAP framework are introduced. Finally, we use the XGBoost algorithm from Phase 2 to provide an example of how LIME can be used to explain black-box models locally.

4.3.1. Logistic Regression: Feature Importance

In logistic regression, feature importance is inferred based on the coefficients associated with each feature in the logistic regression equation. Since the dataset was rescaled to a common scale, the magnitude of coefficients can be directly used as feature importance (Brownlee, 2020b). A positive coefficient indicates that as the feature (scaled) value increases, the log odds of the target variable being 1 increase, and a negative coefficient indicates the opposite. Figure 4.7 presents the 20 most influential features resulting from this research according to the logistic regression. Features that add the most to the probability of an application being classified as fraudulent are, for example, customer age or a missing value in the variable containing the count of months at a previous address which was induced in preprocessing. On the other hand, having a "BE" or "BF" housing status[28] decreases the likelihood of a fraudulent transaction. The feature importance framework can be useful when the researcher wishes to reduce the number of features used in the model or select the most important ones to the model.

[28] A more detailed description of these features is unavailable due to anonymization of the dataset.

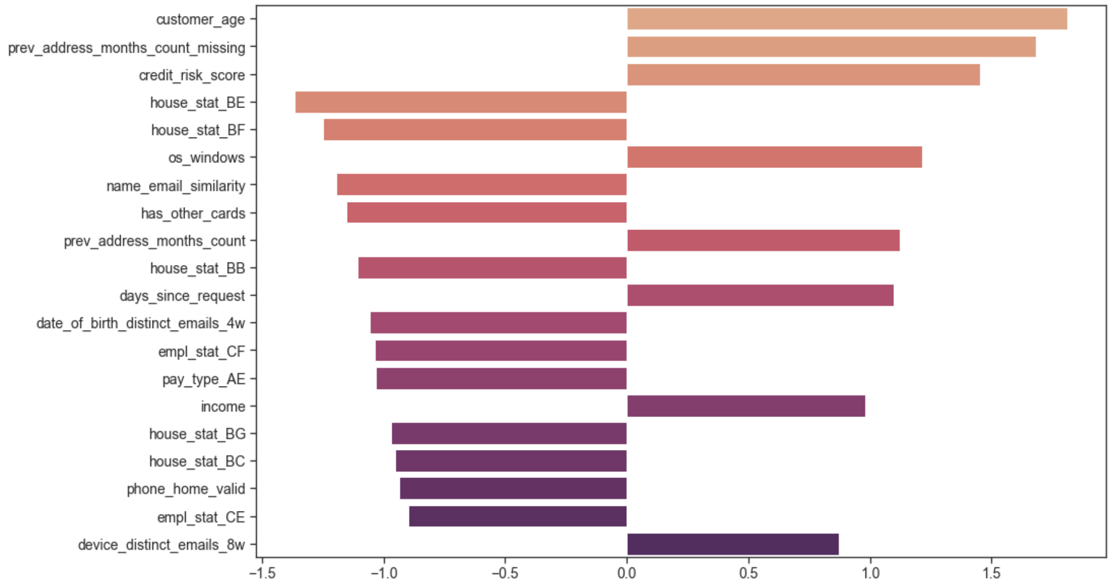


Figure 4.7.: Feature Importance (Model Coefficients) of the Logistic Regression

4.3.2. Decision Tree

As mentioned in Chapter 2, decision trees offer probably the most intuitive interpretation. Figure 4.8 presents a visualization of the first three layers of the resampled decision tree from section 4.2. What information can we obtain from such a figure? The first line of each node represents the splitting criterion. If fulfilled, we move to the left branch. On the contrary, if the observation does not meet the splitting criteria, it moves to the subsequent node on the right side of the current node. The second line provides information about the impurity measure used for branching and its value at the given node. By comparing a node to its two subsequent nodes, one can see the gain in purity of the subsamples obtained by the split. The third line tells us what percentage of the original sample is assessed at the respective tree node. The fourth line tells us what percentage of the samples in the node belong to each class and, finally, the last line gives us a prediction of the node based on the selected probability threshold.

Example. Assume an applicant with two distinct emails logged to their device, using Windows on their PC. Moreover, this applicant's emails do not contain their name so their name_email_similarity factor is low. We start from the topmost node. We see that the Gini coefficient is 0.02 at the beginning with the class balance being 0.99 to 0.01. Since the person has two emails, the first decision rule evaluates as False leading us towards the right node in the second layer. Now we can observe that only 2.8% of the initial sample has more than one distinct email and that this subsample is approximately four times less pure than the initial sample as the Gini impurity index grew four times. This might be counterintuitive, however, in this extremely imbalanced case, an increase

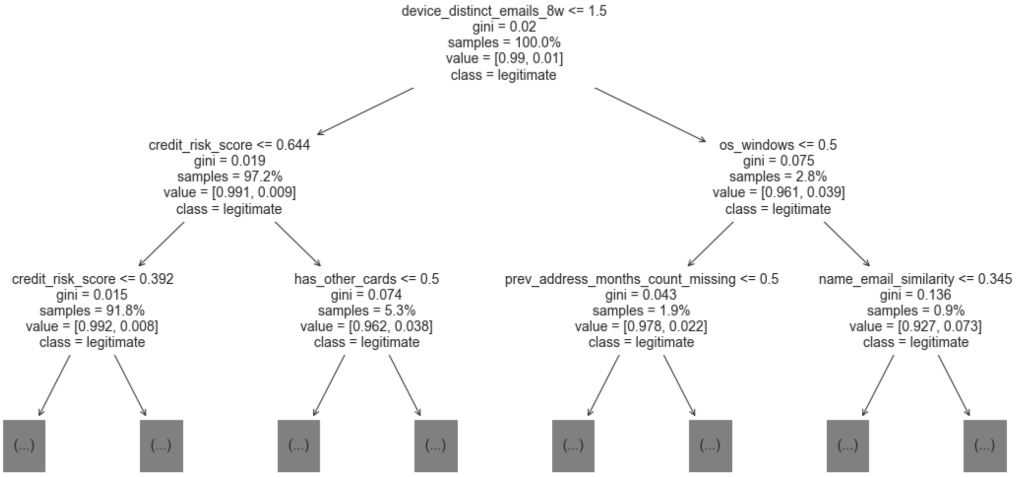


Figure 4.8.: Decision Tree (Baseline Model): First Three Layers

in impurity means that we get rid of some of the majority class and move more towards the desired minority class in our sample. Next, since the person uses Windows as their operation system, the logical expression $os\_windows \leq 0.5$ evaluates as False. Therefore, again, we go to the right subnode. Following the same logic, we continue to the left based on name_email_similarity. We repeat this step-by-step process until we arrive at a leaf node of the tree providing us with a final probability estimate and class prediction.

Note that decision tree models also allow for feature importance calculations. In Python, decision tree classifiers have the $feature\_importances\_$ property. According to Pedregosa et al. (2011), "feature importance is computed as the (normalized) total reduction of the criterion brought by that feature." The visualization and interpretation of the feature importance are highly similar to the example given for logistic regression and allow comparison of all the algorithms (see section 4.3.5 and Appendix A.4).

4.3.3. Random Forest: SHAP

Ensemble methods such as Random Forest and XGBoost also offer insight into feature importance. In an ensemble of decision trees, feature importance can be obtained by aggregating the importances of individual trees. See section 4.3.5 or Appendix A.4 for the feature importance of the RF. On the example of a random forest classifier, we will explore selected outputs of the SHAP framework (Lundberg and Lee, 2017) for local and global explanations which we introduced in section 2.5.

The explanation in Figure 4.9 provides a visual representation of how individual features influence the model's output. In this representation, each feature is depicted as either

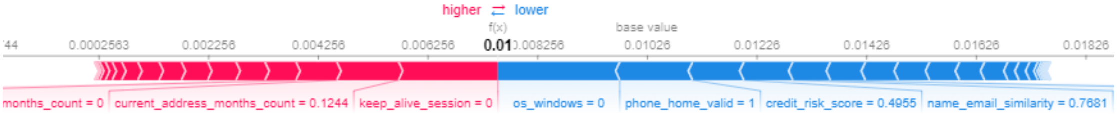


Figure 4.9.: SHAP value decomposition

increasing (shown in red) or decreasing (shown in blue) the model's prediction from a starting point, which is typically the average model output on the training dataset. This visualization helps illustrate how each feature contributes to the model's final predictions. Note that such a figure is specific for each observation from the training sample as it is based on each individual's value (which can be found next to the feature name in the plot). Looking at individual SHAP value decomposition enables for local explainability of black-box models.

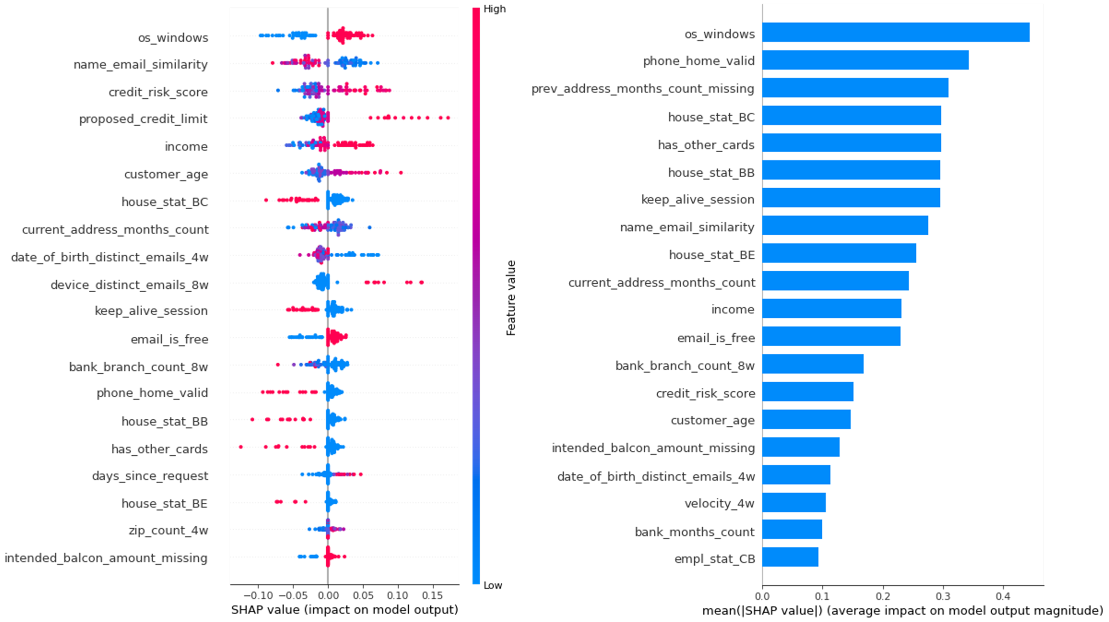


Figure 4.10.: SHAP values (left) and Average impact of a feature (right) for a Random Forest

Computing SHAP values for many (ideally for all) observations from the training sample, we can then plot them into a "beeswarm" graph (Figure 4.10 left) to summarize the effect of selected features. Note that in practice, computing all values might get computationally quite complex when working with big datasets. For this reason, only a sample of 500 observations was used to generate the graph on the right of Figure 4.10. To generate the beeswarm graph on the left of Figure 4.10, only 100 SHAP values were computed for better readability.[29] Let's explain how to read the figure using the proposed_credit_limit (fourth line) as an example. For high values of this feature (high proposed limit), we get a bright red color in the graph. We can see that all the red dots have a highly positive effect on the model output, i.e., the likelihood of being classified as fraud. On the other hand, for average or low feature values of proposed_credit_limit, the impact on the model's prediction is negligible. Therefore, the model sees applications with extremely high proposed credit limits as more suspicious but does not care much about other than very high values.

Finally, we can take an absolute value of the values from the beeswarm graph and take a mean of each feature to construct the figure on the right-hand side of Fig. 4.10. Such a graphic enables a similar global interpretation of feature importance as it represents an average effect of each variable on the model's output. The most influential features are at the top of the graph. In general, the ordering of the features is the same for the summary and beeswarm plots. In our case, however, the ordering differs as different samples were used to generate the plots.

4.3.4. XGBoost: LIME

As mentioned above, the feature importance and SHAP frameworks are also applicable to XGBoost. We provide feature importance in the following section. However, in this section, we introduce another approach to the local interpretation; LIME (Ribeiro et al., 2016), implemented by the InterpretML package and previously discussed in section 2.5. Figure 4.11 enables an understanding of how a machine learning model makes predictions for a specific instance.

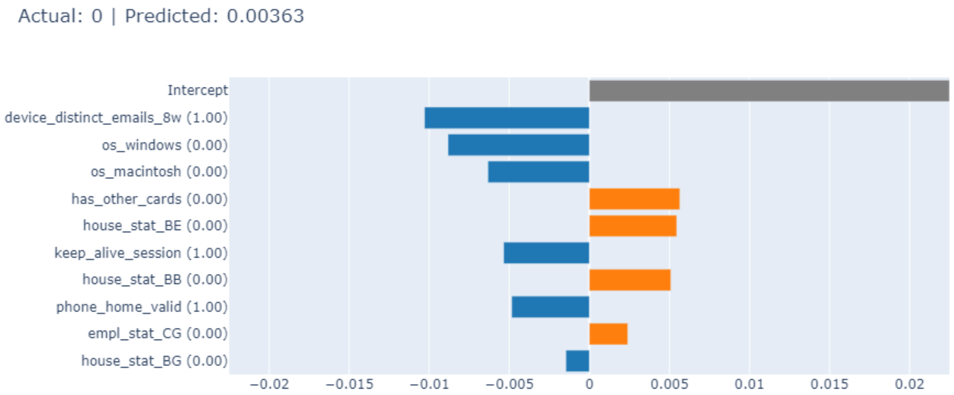


Figure 4.11.: LIME Tabular Graph of XGBoost

[29] To illustrate the computational complexity, generating both graphs with TreeSHAP (algorithm optimized for tree-based models) took approximately 2 minutes on a PC with 16GB RAM and AMD Ryzen 7 PRO 3700U w/ Radeon Vega Mobile Gfx 2.30 GHz processor. Calculating SHAP values for all the observations in the dataset thus was not plausible for us.

The plot displays the importance and direction of the impact of each feature's value (visible next to the feature name) on the model's prediction for that instance. Longer bars indicate greater feature importance, and the direction (right or left) signifies whether increasing a feature leads to a higher or lower prediction (higher prediction meaning higher likelihood of fraudulent application). By analyzing the plot, we gain insight into why the model made a particular decision for that instance, making it easier to understand the local behavior of the model. However, LIME explanations are instance-specific and may not represent the model's global behavior. If one is interested in global explanations, the feature importance or the SHAP framework needs to be used instead.

4.3.5. Feature Importance Across Algorithms

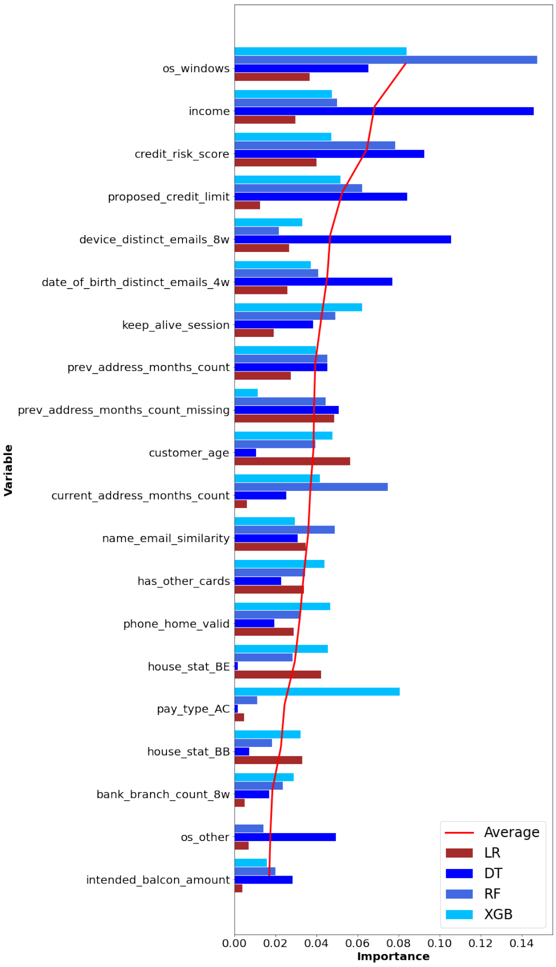


Figure 4.12.: Absolute Feature Importance Across Trained Algorithms

Absolute importance. The feature importance is perfectly comparable across tree-based models and also somewhat comparable to the one obtained from LR coefficients. Comparing feature importances across multiple machine learning models can offer valuable insights into the significance of different features within the BAF dataset and how each model perceives their relevance. We look for features that consistently show high importance across all models. Consistent high importance suggests these features strongly influence predictions, making them potentially crucial for the model's performance. Figure 4.12 makes such a comparison, ranking the 20 most-important variables by their average absolute importance to the four models. One can see that models agree on some features more than on others. For example, all four models use credit_risk_score, income, or os_windows very extensively to make predictions. These features should be always included in any ML tasks. On the other hand, there are variables such as pay_type_AC or os_other that are important only to one of the models (XGB and DT respectively) or, on the contrary, important to all models except one such as house_stat_BE (not used by the DT).

Relative importance. Take the variable os_other, for example. This feature is not used by XGBoost, has low importance to RF and LR and only occurs in the top 20 due to its high importance to the decision tree. We might want to exclude this variable from the top 20 in certain cases[30] since it is important only to the worst-performing model and not to the others. As mentioned above, the scale of importance might differ between models, so the relative importance of features within the same model might be useful. Therefore, focusing on the relative ranking of features within each model might bring additional insight into feature selection. When deciding based on relative importance, os_other would not rank 19th as in Figure 4.12 but 28th and would thus be excluded from the variable selection. See Appendix A.4 for a table containing all absolute and relative importance for all four algorithms. Note that the seven features ranking last were not used by any of the tree-based algorithms, it might be thus advantageous to drop these variables for further modeling purposes (if there were any in this research) even though they are to some extent important to the logistic regression.

4.3.6. Phase 3: Conclusions

In this section, we provided several options for local and global model interpretability. The most straightforward to interpret is the decision tree classifier where it suffices to follow the splitting criteria based on the feature values and move through the decision tree, finally arriving at a prediction. Next, the feature importance for global explanations was discussed in the example of a logistic regression. In the case of logistic regression, feature importances are derived from the regression coefficients. Feature importances are also possible to calculate for decision trees and tree ensemble methods such as Random Forest and XGBoost based on metrics like Gini impurity to determine how well a particular feature splits the data. We provided a comparison of feature importance across models in section 4.3.5. However, an additional framework is needed for two reasons: (1) feature importance does not offer an insight into local explanations of black-box models, and (2) some algorithms like Artificial Neural Networks, Support Vector Machines or Naïve Bayes do not allow for straightforward feature importance.

Examples of two advanced techniques to explain black-box models were provided using the RF and XGB models trained in Phase 2: SHAP and LIME. First, individual SHAP values were presented separately to allow for local explanability. Later, we put SHAP values for multiple observations into one plot thus creating a beeswarm graph. Next, by taking an absolute value and a mean of the beeswarm plot, the mean global impact of each feature on the model's output (similar to feature importance) was obtained. Last, in the XGBoost example, we show how LIME from the InterpretML package can be implemented and interpreted to provide local interpretability of black-box models.

[30] Imagine a hypothetical scenario where only 15 to 20 features may be used to predict fraud in order to, e.g., save database memory or decrease computational requirements.

4.4. Chapter Summary

To conclude this chapter, we summarize the results of the three phases of the empirical part of this thesis. First, the results of Phase 1 include the evaluation of the three resampling techniques together with cost-sensitive learning for each of the algorithms. On top of the best resampling strategy and target balance, this phase provides tuned values of hyperparameters for all models. To summarize the results of this phase, Random Undersampling was found to be the go-to resampling strategy in the case of the BAF dataset for all the assessed algorithms. Moreover, Randomized search results suggest that RUS should be ideally employed at target ratios ranging from 0.05 to 0.1. Random Oversampling performed almost as well as undersampling for the majority of the models. However, since ROS adds dimensions to the training data, it makes the whole training procedure more costly. On the contrary, RUS cuts dimensions and therefore also improves computational efficiency. For this reason, undersampling is preferred to oversampling in our case. Last, SMOTE failed to perform well at any target ratios ultimately damaging predictive performance of all the tested models. Finally, our results show that cost-sensitive learning is not a viable solution to the class imbalance in our case.

Second, Phase 2 uses the above-mentioned results to train final prediction models using the first 6 months of the data. It then evaluates them on the last 2 months and presents the results here. All four models based on the resampled data are compared based on their TPR @ 5% FPR to their baseline versions and to all other models evaluated in this part. We found that the overall best-performing model is the XGBoost with a Recall of 0.54. Interestingly, logistic regression ranked second best with its resample version having similar results to its baseline version. The overall worst model is, according to our research, a decision tree. A comparison of baseline and resampled models brought an interesting insight; resampling increased the model performance only in the case of a decision tree and had an immaterial effect on logistic regression, XGboost, and random forest. Such findings are in line with, e.g., Akyol and Atila (2020), who show that in their empirical experiment, resampled models outperformed their baseline counterparts in only a few cases. Even though RUS does not increase model performance drastically, it might prove useful by decreasing computational time.

Finally, in Phase 3, we provide empiric examples of how to interpret the models trained in this research. For global explanations, we employ the feature importance framework which can be used for any of the four algorithms. Feature importances (both absolute and relative to each model) were compared for all the four ML algorithms trained in this research. Additionally, we also comment on how to provide global explanations with SHAP. For local explanations of black-box models, SHAP and LIME examples are provided to explain the random forest and XGBoost trained in the previous stages of the research. In the case of a decision tree, the three graph is used as a local explainability technique. For logistic regression, the global and local explanations are more or less equivalent as both use regression coefficients.

5. Conclusion

5.1. Findings Summary

To detect financial fraud, machine learning algorithms are often used. Fraud datasets, however, bring several challenges to machine learning. In this thesis, we address two challenges that usually accompany machine learning tasks in the fraud detection field: high class imbalance which makes it difficult to train and evaluate models, and interpretability of the complex black-box algorithms.

First, we evaluated the possible use of four imbalance-handling techniques (Random Undersampling, Random Oversampling, Synthetic Minority Oversampling, and Cost-sensitive Learning) for training of four classification algorithms (logistic regression, decision tree, random forest, and XGBoost). As class imbalance is present in the dataset, simple evaluation metrics such as accuracy fail to measure model performance reliably. For this reason, the area under the ROC curve and Recall @0.05 False Positive Rate were used to evaluate models. According to our analysis, the most successful technique to handle class imbalance is Random Undersampling, i.e., randomly dropping observations that belong to the majority class. We have shown that for all of the models, dropping 80 to 90 percent of the minority class instances leads to an increased prediction power of the algorithms on the validation set. However, in the second part of the analysis, it was shown that when evaluating the final models on the last two months of the data (the test set), RUS increased performance only in the case of the decision tree. In the case of the other three algorithms, undersampling the data had an insignificant effect on the final model performance, yet decreased the training time significantly. Finally, when comparing all the models' performance, we found that XGBoost performs the best, overall, with a TPR of 0.54 at 0.05 FPR. Interestingly, logistic regression ranked second best with only a slightly worse result (0.51) than XGBoost which is surprising when taking into consideration that XGBoost is a much more advanced and complex algorithm. According to our results, the overall worst model is a decision tree.

Finally, we provide empirical examples of how the models generated through this study can be interpreted. We discuss the feature importance framework, which may be used in all four models to assist with global explanations. Furthermore, we investigate the methods for providing global explanations for black-box models using SHAP and present instances of local explanations using SHAP and LIME to explain individual predictions. To explain predictions of a decision tree, a three-graph is used to go through the tree step by step until we arrive at a prediction.

5.2. Limitations and Further Research

Several areas where this research might be further developed come to mind. First, the biggest limitation in our case was limited computing power. As a result, the Randomized search which was used for hyperparameter tuning and target ratio selection consisted of only 20 iterations (hyperparameter sets). As we searched over four to six hyperparameters with numerous possible values, the selected hyperparameter set might be far from an optimal one. This makes the comparison across models less reliable as it could happen that one model outperforms the other simply because we were more "lucky" when randomly searching for its hyperparameters compared to the other model. Additionally, taking into consideration a wider range of hyperparameters to tune could also be advantageous. For example, a forest consisting of 1000 trees might perform significantly better than the one we trained as we didn't consider such a high number to save runtime. Another consequence of small computing capacity was the limited number of target ratios considered in the analysis. In the case of LR and DT, we resampled the data to 8 target ratios while computational constraints allowed for only 4 target ratios in the case of ensemble methods. An increase in computing power would allow us to conduct the same analysis with higher granularity offering better insight into how models react to different resampling strategies. To sum up this point, scaling the whole analysis up would provide us with more robust results enabling us to draw more reliable conclusions from the analysis.

Second, in this research, we use the test set to calibrate the probability threshold in Phase 2. However, this is, to some extent a case of data leakage as we calibrate the trained model based on a dataset it should have never seen before making final predictions. This approach was chosen to enable for precise performance comparison across models. Probably the more applicable-in-practice approach would be to calibrate the probability threshold on a validation set (based on first 6 months of the data) and use it to make final predictions. This approach might, however, yield different evaluation metrics for each model on the test set (models might achieve different FPRs, i.e., one might have TPR at 4% FPR and another might have TPR at 7% FPR). Since the main goal was to make evaluations comparable among all models and probability threshold calibration does not change the models' inner setup, we decided to calibrate the threshold based on the test data.

Third, more advanced machine learning algorithms such as Artificial Neural Networks might be more successful when dealing with the BAF dataset. Also, more complex resampling techniques might be more successful than the ones used in this research. Some of the possible resampling techniques might be, e.g., Tomek Links, Adaptive Synthetic Sampling (ADASYN), or Edited Nearest Neighbors (ENN). However, as resampling generally proved unsuccessful in our research, the aforementioned methods might also fail to perform well.

Finally, if interested, fairness evaluation could be also included in the research. As briefly mentioned in this thesis, ensuring that machine learning models do not discriminate based on protected attributes such as race or sex is crucial. As mentioned by Jesus et al. (2022a), the BAF dataset is an optimal source for fair ML evaluation as the dataset contains informative features.

Bibliography

A. O. Adewumi and A. A. Akinyelu. A survey of machine-learning and nature-inspired based credit card fraud detection techniques. *International Journal of System Assurance Engineering and Management*, 8(2):937–953, Nov. 2017. ISSN 0976-4348. doi: 10.1007/s13198-016-0551-y. URL `https://doi.org/10.1007/s13198-016-0551-y`.

C. C. Aggarwal, editor. *Data Classification Algorithms and Applications*. Chapman & Hall Book/CRC Press, 2015. ISBN 978-1-4665-8675-8.

K. Akyol and Atila. EEG Sinyallerinden Epileptik Nöbet Tahmini Üzerine Bir Çalışma. *Academic Platform Journal of Engineering and Science*, 8(2):279–285, May 2020. ISSN 2147-4575. doi: 10.21541/apjes.569553. URL `https://dergipark.org.tr/en/doi/10.21541/apjes.569553`.

J. O. Awoyemi, A. O. Adetunmbi, and S. A. Oluwadare. Credit card fraud detection using machine learning techniques: A comparative analysis. In *2017 International Conference on Computing Networking and Informatics (ICCNI)*, pages 1–9, Oct. 2017. doi: 10.1109/ICCNI.2017.8123782.

S. Barocas and A. D. Selbst. Big Data's Disparate Impact. *California Law Review*, 104 (3):671–732, 2016. ISSN 0008-1221. URL `https://www.jstor.org/stable/24758720`. Publisher: California Law Review, Inc.

T. Begley, T. Schwedes, C. Frye, and I. Feige. Explainability for fair machine learning, Oct. 2020. URL `http://arxiv.org/abs/2010.07389`. arXiv:2010.07389 [cs, stat].

T. Bolukbasi, K.-W. Chang, J. Y. Zou, V. Saligrama, and A. T. Kalai. Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL `https://proceedings.neurips.cc/paper_files/paper/2016/hash/a486cd07e4ac3d270571622f4f316ec5-Abstract.html`.

L. Breiman. Random forests. *Journal Machine Learning Archive*, 45(1):5–32, 2001.

L. Breiman. *Classification and Regression Trees*. Routledge, New York, Oct. 2017. ISBN 978-1-315-13947-0. doi: 10.1201/9781315139470.

J. Brownlee. *XGBoost With Python: Gradient Boosted Trees with XGBoost and scikit-learn*. Machine Learning Mastery, Aug. 2016. Google-Books-ID: HgmqDwAAQBAJ.

J. Brownlee. Data Leakage in Machine Learning, Aug. 2020a. URL `https://machinelearningmastery.com/data-leakage-machine-learning/`.

J. Brownlee. How to Calculate Feature Importance With Python, Mar. 2020b. URL https://machinelearningmastery.com/calculate-feature-importance-with-python/.

J. Brownlee. Random Oversampling and Undersampling for Imbalanced Classification, Jan. 2020c. URL https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/.

A. Caliskan, J. J. Bryson, and A. Narayanan. Semantics derived automatically from language corpora contain human-like biases. *Science*, 356(6334):183–186, Apr. 2017. ISSN 0036-8075, 1095-9203. doi: 10.1126/science.aal4230. URL http://arxiv.org/abs/1608.07187. arXiv:1608.07187 [cs].

N. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002. ISSN 1076-9757. doi: 10.1613/jair.953. URL http://arxiv.org/abs/1106.1813. arXiv:1106.1813 [cs].

T. Chen and C. Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, Aug. 2016. doi: 10.1145/2939672.2939785. URL http://arxiv.org/abs/1603.02754. arXiv:1603.02754 [cs].

D. Cirqueira, D. Nedbal, M. Helfert, and M. Bezbradica. Scenario-Based Requirements Elicitation for User-Centric Explainable AI. In A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. Weippl, editors, *Machine Learning and Knowledge Extraction*, Lecture Notes in Computer Science, pages 321–341, Cham, 2020. Springer International Publishing. ISBN 978-3-030-57321-8. doi: 10.1007/978-3-030-57321-8_18.

J. Cook and V. Ramadas. When to consult precision-recall curves. *The Stata Journal*, 20(1):131–148, Mar. 2020. ISSN 1536-867X. doi: 10.1177/1536867X20909693. URL https://doi.org/10.1177/1536867X20909693. Publisher: SAGE Publications.

S. Corbett-Davies and S. Goel. The Measure and Mismeasure of Fairness: A Critical Review of Fair Machine Learning, Aug. 2018. URL http://arxiv.org/abs/1808.00023. arXiv:1808.00023 [cs].

V. N. Dornadula and S. Geetha. Credit Card Fraud Detection using Machine Learning Algorithms. *Procedia Computer Science*, 165:631–641, 2019. ISSN 18770509. doi: 10.1016/j.procs.2020.01.057. URL https://linkinghub.elsevier.com/retrieve/pii/S187705092030065X.

C. Drummond and R. C. Holte. C4.5, Class Imbalance, and Cost Sensitivity: Why Under-Sampling beats Over-Sampling. 2003.

J. Fan, S. Upadhye, and A. Worster. Understanding receiver operating characteristic (ROC) curves. *CJEM*, 8(01):19–20, Jan. 2006. ISSN 1481-8035, 1481-8043. doi: 10.1017/S1481803500013336. URL https://www.cambridge.org/core/product/identifier/S1481803500013336/type/journal_article.

FTC. Consumer Sentinel Network Data Book 2022. Technical report, 2023. URL `https://www.ftc.gov/system/files/ftc_gov/pdf/CSN-Data-Book-2022.pdf`.

P. Hacker and J.-H. Passoth. Varieties of AI Explanations under the Law. From the GDPR to the AIA, and Beyond, Aug. 2021. URL `https://papers.ssrn.com/abstract=3911324`.

A. Holzinger, R. Goebel, R. Fong, T. Moon, K.-R. Müller, and W. Samek, editors. *xxAI - Beyond Explainable AI: International Workshop, Held in Conjunction with ICML 2020, July 18, 2020, Vienna, Austria, Revised and Extended Papers*, volume 13200 of *Lecture Notes in Computer Science*. Springer International Publishing, Cham, 2022. ISBN 978-3-031-04082-5 978-3-031-04083-2. doi: 10.1007/978-3-031-04083-2. URL `https://link.springer.com/10.1007/978-3-031-04083-2`.

S. Jesus, J. Pombal, D. Alves, A. Cruz, P. Saleiro, R. P. Ribeiro, J. Gama, and P. Bizarro. Turning the Tables: Biased, Imbalanced, Dynamic Tabular Datasets for ML Evaluation, Nov. 2022a. URL `http://arxiv.org/abs/2211.13358`. arXiv:2211.13358 [cs].

S. Jesus, J. Pombal, D. Alves, A. F. Cruz, P. Saleiro, R. P. Ribeiro, J. Gama, and P. Bizarro. BAF Dataset Suite Datasheet. 2022b. URL `https://github.com/feedzai/bank-account-fraud/blob/main/documents/datasheet.pdf`.

M. Kuhn and J. Silge. *Tidy Modeling with R*. O'Reilly, 2023. URL `https://www.tmwr.org/`.

Y.-A. Le Borgne, W. Siblini, B. Lebichot, and G. Bontempi. *Reproducible Machine Learning for Credit Card Fraud Detection - Practical Handbook*. Université Libre de Bruxelles, 2022. URL `https://github.com/Fraud-Detection-Handbook/fraud-detection-handbook`.

G. Lemaître, F. Nogueira, and C. K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL `http://jmlr.org/papers/v18/16-365.html`.

E. A. Lopez-Rojas, A. Elmir, and S. Axelsson. PAYSIM: A FINANCIAL MOBILE MONEY SIMULATOR FOR FRAUD DETECTION. 2016.

G. Louppe. Understanding Random Forests: From Theory to Practice, June 2015. URL `http://arxiv.org/abs/1407.7502`. arXiv:1407.7502 [stat].

Y. Lucas and J. Jurgovsky. Credit card fraud detection using machine learning: A survey, Oct. 2020. URL `http://arxiv.org/abs/2010.06479`. arXiv:2010.06479 [cs].

S. Lundberg. SHAP documentation, 2018. URL `https://shap.readthedocs.io/en/latest/index.html`.

S. Lundberg and S.-I. Lee. A Unified Approach to Interpreting Model Predictions, Nov. 2017. URL `http://arxiv.org/abs/1705.07874`. arXiv:1705.07874 [cs, stat].

I. Mekterović, L. Brkić, and M. Baranović. A Systematic Review of Data Mining Approaches to Credit Card Fraud Detection. 15, 2018.

G. Menardi and N. Torelli. Training and assessing classification rules with imbalanced data. *Data Mining and Knowledge Discovery*, 28(1):92–122, Jan. 2014. ISSN 1573-756X. doi: 10.1007/s10618-012-0295-5. URL `https://doi.org/10.1007/s10618-012-0295-5`.

R. Mohammed, J. Rawashdeh, and M. Abdullah. Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. In *2020 11th International Conference on Information and Communication Systems (ICICS)*, pages 243–248, Apr. 2020. doi: 10.1109/ICICS49469.2020.239556. ISSN: 2573-3346.

C. Molnar. *Interpretable Machine Learning*. 2 edition, 2022. URL `https://christophm.github.io/interpretable-ml-book`.

OLAF. Directive (EU) 2017/1371 of the European Parliament and of the Council of 5 July 2017 on the fight against fraud to the Union's financial interests by means of criminal law. Technical report, European Parliament, July 2017. URL `https://anti-fraud.ec.europa.eu/olaf-and-you/report-fraud_en`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166, Dec. 2015. doi: 10.1109/SSCI.2015.33.

C. V. Priscilla and D. P. Prabha. Credit Card Fraud Detection: A Systematic Review. volume 9, pages 290–303, Cham, 2020. Springer International Publishing. ISBN 978-3-030-38500-2 978-3-030-38501-9. doi: 10.1007/978-3-030-38501-9_29. URL `http://link.springer.com/10.1007/978-3-030-38501-9_29`. Book Title: Intelligent Computing Paradigm and Cutting-edge Technologies Series Title: Learning and Analytics in Intelligent Systems.

F. Provost and T. Fawcett. Data Science for Business. *O'Reilly Media*, 2013.

I. Psychoula, A. Gutmann, P. Mainali, S. H. Lee, P. Dunphy, and F. Petitcolas. Explainable Machine Learning for Fraud Detection. *Computer*, 54(10):49–59, Oct. 2021. ISSN 1558-0814. doi: 10.1109/MC.2021.3081249. Conference Name: Computer.

M. T. Ribeiro, S. Singh, and C. Guestrin. "Why Should I Trust You?": Explaining the Predictions of Any Classifier, Aug. 2016. URL `http://arxiv.org/abs/1602.04938`. arXiv:1602.04938 [cs, stat].

A. Thennakoon, C. Bhagyani, S. Premadasa, S. Mihiranga, and N. Kuruwitaarachchi. Real-time Credit Card Fraud Detection Using Machine Learning. In *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 488–493, Jan. 2019. doi: 10.1109/CONFLUENCE.2019.8776942.

D. Varmedja, M. Karanovic, S. Sladojevic, M. Arsenovic, and A. Anderla. Credit Card Fraud Detection - Machine Learning methods. In *2019 18th International Symposium INFOTEH-JAHORINA (INFOTEH)*, pages 1–5, Mar. 2019. doi: 10.1109/INFOTEH.2019.8717766.

T. Wongvorachan, S. He, and O. Bulut. A Comparison of Undersampling, Oversampling, and SMOTE Methods for Dealing with Imbalanced Classification in Educational Data Mining. *Information*, 14(1):54, Jan. 2023. ISSN 2078-2489. doi: 10.3390/info14010054. URL `https://www.mdpi.com/2078-2489/14/1/54`. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

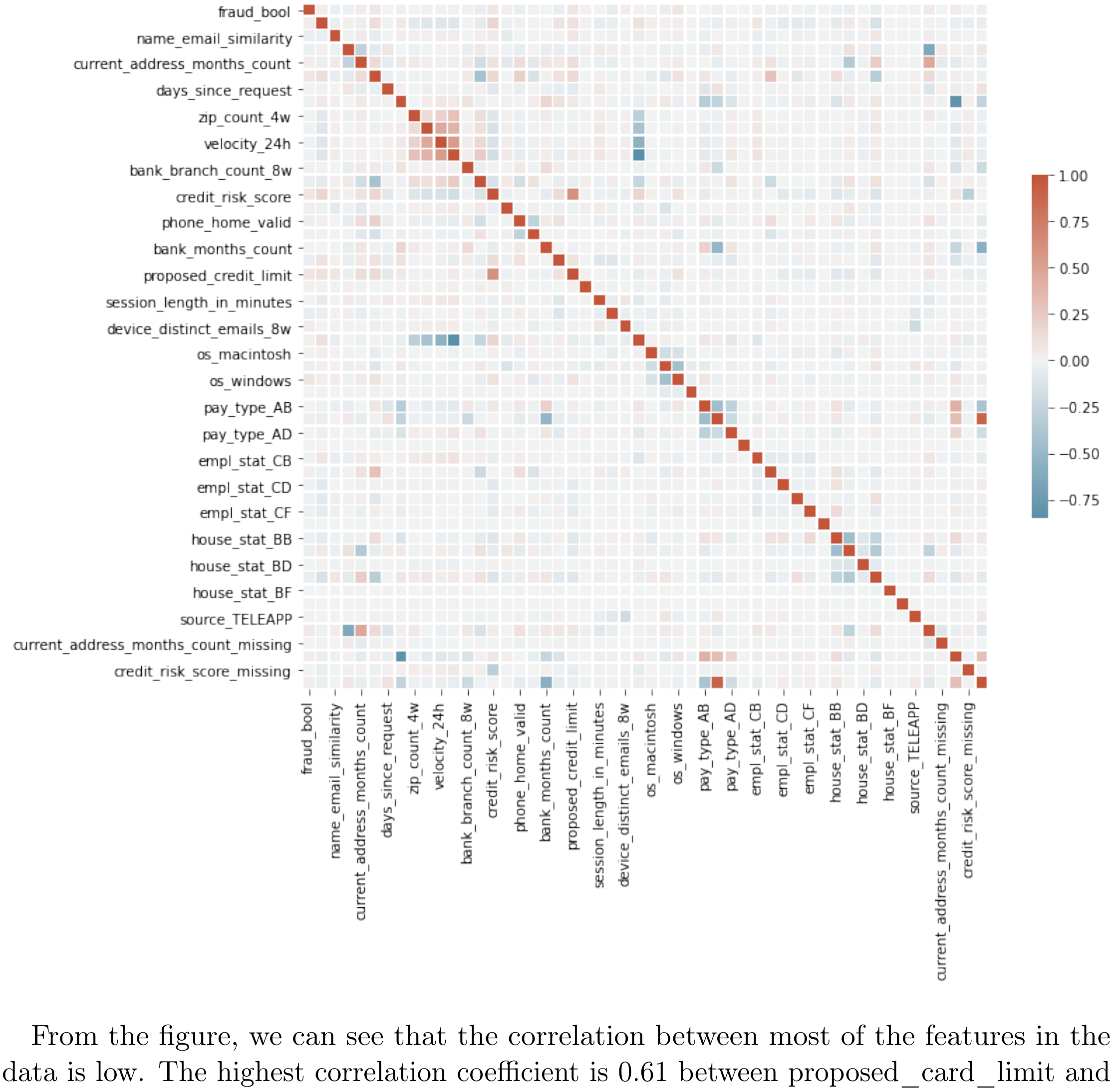
N. Yousefi, M. Alaghband, and I. Garibay. A Comprehensive Survey on Machine Learning Techniques and User Authentication Approaches for Credit Card Fraud Detection, Dec. 2019. URL `http://arxiv.org/abs/1912.02629`. arXiv:1912.02629 [cs].

Z. Zojaji, R. E. Atani, and A. H. Monadjemi. A Survey of Credit Card Fraud Detection Techniques: Data and Technique Oriented Perspective. 2016.

A. Appendix

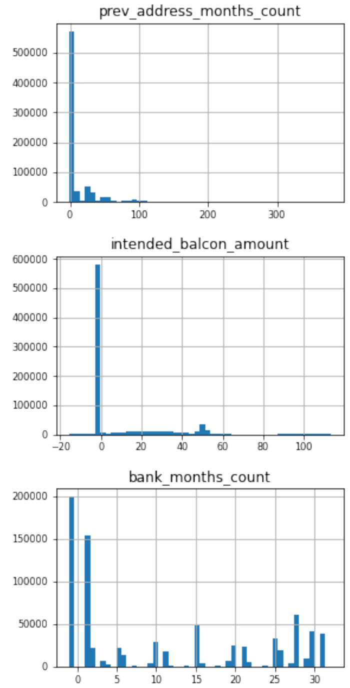
A.1. Correlation Analysis

Figure A.1.: Correlation Matrix

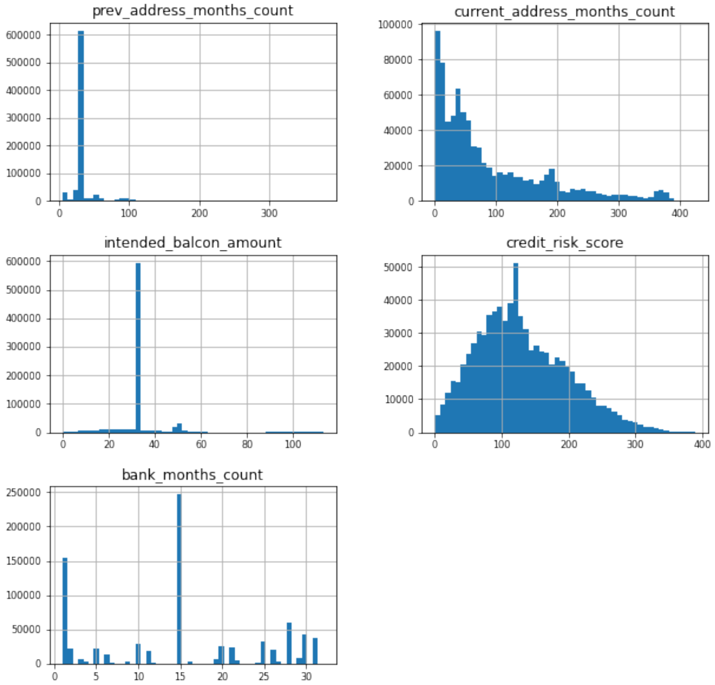


From the figure, we can see that the correlation between most of the features in the data is low. The highest correlation coefficient is 0.61 between proposed_card_limit and credit_risk_score. From this analysis, we conclude that all features should be considered for modeling.

A.2. Missing Values Imputation



(a) Features with missing values prior to NA imputation Median strategy



(b) Features with missing values after NA imputation Median strategy

A.3. Hyperparameters of Each Algorithm Included in the Randomized Search and Their Value Range

For a detailed description of each hyperparameter, see the Scikit Learn Documentation and the XGBoost documentation.

Logistic Regression

- *solver*: 'saga';
- *penalty*: 'none', 'l2', 'l1', 'elasticnet';
- *C*: stats.loguniform(1e-5, 100);
- *class_ weight*: None, 'balanced'.

Decision Tree

- *max_ depth*: 2, 4, 6, 8, 10;
- *min_ samples_ split*: stats.randint(2, 11);
- *min_ samples_ leaf*: stats.randint(1, 11);
- *max_features*: 'sqrt', 'log2';
- *criterion*: 'gini', 'entropy';
- *class_ weight*: 'balanced', None.

Random Forest

- *n_ estimators*: 50, 75, 100, 150, 200;
- *criterion*: 'gini', 'entropy';
- *max_ depth*: 2, 4, 6, 8, 10;
- *max_features*: 'sqrt', 'log2';
- *class_ weight*: 'balanced', 'balanced_subsample', None.

XGBoost

- *n_ estimators*: 50, 75, 100, 150;
- *max_ depth*: 2, 4, 6, 8, 10;
- *learning_ rate*: 0.01, 0.1, 0.2;
- *subsample*: 0.6, 0.8, 1.0;
- *colsample_ bytree*: 0.6, 0.8, 1.0;
- *scale_pos_ weight*: 1, 10, 100

A.4. Feature Importance Relative to each Model

Variable (1 to 22)	LR	DT	RF	XGB	rank_LR	rank_DT	rank_RF	rank_XGB	Average_rank
os_windows	0.037	0.065	0.148	0.084	5	6	1	1	3.25
credit_risk_score	0.040	0.093	0.078	0.047	4	3	2	7	4.00
income	0.030	0.146	0.050	0.048	11	1	5	5	5.50
customer_age	0.056	0.011	0.039	0.048	1	19	11	5	9.00
name_email_similarity	0.035	0.031	0.049	0.029	6	11	6	17	10.00
keep_alive_session	0.019	0.038	0.049	0.062	23	10	6	3	10.50
prev_address_months_count_missing	0.049	0.051	0.044	0.012	2	7	9	24	10.50
has_other_cards	0.034	0.023	0.034	0.044	7	14	12	10	10.75
prev_address_months_count	0.028	0.045	0.045	0.040	16	9	8	12	11.25
proposed_credit_limit	0.013	0.084	0.062	0.052	34	4	4	4	11.50
date_of_birth_distinct_emails_4w	0.026	0.077	0.041	0.037	19	5	10	13	11.75
device_distinct_emails_8w	0.027	0.106	0.022	0.033	17	2	16	14	12.25
phone_home_valid	0.029	0.020	0.032	0.047	14	16	13	7	12.50
house_stat_BE	0.042	0.002	0.028	0.046	3	29	14	9	13.75
house_stat_BB	0.033	0.007	0.018	0.032	8	21	18	15	15.50
current_address_months_count	0.006	0.025	0.075	0.042	42	13	3	11	17.25
house_stat_BC	0.029	0.002	0.013	0.019	14	29	20	19	20.50
days_since_request	0.030	0.004	0.008	0.007	11	24	25	28	22.00
email_is_free	0.017	0.009	0.009	0.029	27	20	24	17	22.00
bank_branch_count_8w	0.005	0.017	0.024	0.029	45	18	15	17	23.75
intended_balcon_amount	0.004	0.028	0.020	0.016	47	12	17	20	24.00
pay_type_AC	0.005	0.002	0.011	0.081	45	29	22	2	24.50

Variable (23 to 50)	LR	DT	RF	XGB	rank_LR	rank_DT	rank_RF	rank_XGB	Average_rank
bank_months_count	0.017	0.004	0.011	0.009	27	24	22	26	24.75
velocity_4w	0.015	0.005	0.008	0.008	30	22	25	27	26.00
intended_balcon_amount_miss	0.019	0.019	0.012	0.000	23	17	21	44	26.25
empl_stat_CF	0.033	0.000	0.003	0.012	8	41	33	24	26.50
os_other	0.007	0.049	0.014	0.000	39	8	19	44	27.50
os_macintosh	0.027	0.000	0.003	0.016	17	41	33	20	27.75
zip_count_4w	0.014	0.003	0.007	0.007	32	26	28	28	28.50
session_length_in_minutes	0.009	0.005	0.007	0.006	37	22	28	30	29.25
empl_stat_CD	0.020	0.002	0.001	0.005	21	29	38	33	30.25
empl_stat_CB	0.017	0.000	0.003	0.014	27	41	33	22	30.75
empl_stat_CE	0.029	0.000	0.001	0.005	14	41	38	33	31.50
foreign_request	0.017	0.000	0.001	0.012	27	41	38	24	32.50
empl_stat_CC	0.006	0.023	0.002	0.000	42	14	35	44	33.75
velocity_24h	0.005	0.003	0.006	0.004	45	26	30	35	34.00
house_stat_BF	0.031	0.000	0.000	0.000	10	41	45	44	35.00
house_stat_BD	0.020	0.000	0.000	0.005	21	41	45	33	35.00
os_x11	0.022	0.000	0.000	0.003	20	41	45	37	35.75
bank_months_count_missing	0.013	0.000	0.007	0.000	34	41	28	44	36.75
phone_mobile_valid	0.008	0.000	0.001	0.006	38	41	38	30	36.75
velocity_6h	0.001	0.000	0.006	0.004	50	41	30	35	39.00
pay_type_AB	0.007	0.000	0.001	0.002	39	41	38	38	39.00
source_TELEAPP	0.017	0.000	0.000	0.000	27	41	45	44	39.25
house_stat_BG	0.015	0.000	0.000	0.000	30	41	45	44	40.00
pay_type_AE	0.013	0.000	0.000	0.000	34	41	45	44	41.00
empl_stat_CG	0.011	0.000	0.000	0.000	36	41	45	44	41.50
pay_type_AD	0.006	0.000	0.000	0.000	42	41	45	44	43.00
credit_risk_score_missing	0.004	0.000	0.000	0.000	47	41	45	44	44.25
current_address_months_count_miss	0.002	0.000	0.000	0.000	49	41	45	44	44.75