December 2018

Application of Machine Learning Techniques in Credit Card Fraud Detection

Ronish Shakya

APPLICATION OF MACHINE LEARNING TECHNIQUES IN CREDIT CARD FRAUD DETECTION

By

Ronish Shakya

Bachelor Degree In Computer Engineering
Tribhuvan University, Kathmandu, Nepal
2013

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

Department of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
December 2018

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

November 9, 2018

This thesis prepared by

Ronish Shakya

entitled

Application of Machine Learning Techniques in Credit Card Fraud Detection

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
Department of Computer Science

Fatma Nasoz, Ph.D.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Interim Dean

Laxmi Gewali, Ph.D.
Examination Committee Member

Justin Zhan, Ph.D.
Examination Committee Member

Mehmet Erdem, Ph.D.
Graduate College Faculty Representative

Abstract

Credit card fraud is an ever-growing problem in today's financial market. There has been a rapid increase in the rate of fraudulent activities in recent years causing a substantial financial loss to many organizations, companies, and government agencies. The numbers are expected to increase in the future, because of which, many researchers in this field have focused on detecting fraudulent behaviors early using advanced machine learning techniques. However, the credit card fraud detection is not a straightforward task mainly because of two reasons: (i) the fraudulent behaviors usually differ for each attempt and (ii) the dataset is highly imbalanced, i.e., the frequency of majority samples (genuine cases) outnumbers the minority samples (fraudulent cases).

When providing input data of a highly unbalanced class distribution to the predictive model, the model tends to be biased towards the majority samples. As a result, it tends to misrepresent a fraudulent transaction as a genuine transaction. To tackle this problem, data-level approach, where different resampling methods such as undersampling, oversampling, and hybrid strategies, have been implemented along with an algorithmic approach where ensemble models such as bagging and boosting have been applied to a highly skewed dataset containing 284807 transactions. Out of these transactions, only 492 transactions are labeled as fraudulent. Predictive models such as logistic regression, random forest, and XGBoost in combination with different resampling techniques have been applied to predict if a transaction is fraudulent or genuine. The performance of the model is evaluated based on recall, precision, f1-score, precision-recall (PR) curve, and receiver operating characteristics (ROC) curve. The experimental results showed that random forest in combination with a hybrid resampling approach of Synthetic Minority Over-sampling Technique (SMOTE) and Tomek Links removal performed better than other models.

Acknowledgements

"Foremost, I would like to express my sincere gratitude to my advisor, Dr. Fatma Nasoz, for her valuable and constructive suggestions during the planning and development of this research work. She consistently guided me in the right direction in this research work as well as my Master's program.

I would also like to thank Dr. Laxmi Gewali, Dr. Justin Zhan and Dr. Mehmet Erdem for their continuous support and for being a part of my thesis committee. Furthermore, I am truly grateful to Dr. Ajoy K Datta who has always supported me since the beginning of my Master's program.

I must express my profound gratitude to my parents Ram Kaji Shakya and Sarojana Shakya and to my elder brothers Rosish Shakya and Rojish Shakya for providing me with unfailing support and continuous encouragement throughout my years of study.

Finally, I would like to thank all my friends especially Pradip Maharjan, Ashish Tamrakar, Bibek Bhattarai and Neha Bajracharya who made my time here in UNLV a memorable one."

Ronish Shakya

University of Nevada, Las Vegas

December 2018

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

E-commerce has come a long way since its inception. It has become an essential means for most organizations, companies, and government agencies to increase their productivity in global trade. One of the main reason for the success of e-commerce is the easy online credit card transaction [LBCS16]. Whenever we talk about monetary transactions, we also have to take financial fraud into consideration. Financial fraud is an intentional crime in which a fraudster benefits himself/herself by denying a right to a victim or by obtaining financial gain [AAO17]. As credit card transaction is the most common method of payment in the recent years, the fraud activities have increased rapidly.

Enterprises and public institutions are facing a massive problem as huge amount of financial loss are caused by fraud activities. According to The Nilson Report [nila], the losses due to the credit card, debit card, and prepaid card fraud reached $16.31B worldwide in 2015. And the recent report by The Nilson Report [nila] shows that the gross fraud loss has reached $22.8B in 2018 which is 4% more than that in 2015 and it is expected to exceed by an even more significant amount in the coming years. According to Statista [sta], as shown in figure 1.1, the gross fraud reached $5.6B in 2012, whereas in 2018, the fraud loss has reached $9.1B, which is approximately two-fifths of the total loss. In particular, 70% of these frauds are Card-Not-Present (CNP) frauds (i.e., frauds conducted online or over the telephones), 20% are counterfeits and remaining 10% cases are related to losses due to lost or stolen cards [sta].

The solutions to the fraud can be categorized into prevention, which involves preventing the fraud in the source itself and detection, which is the action taken after the occurrence of the event. The technologies like the Address Verification System (AVS) and Card Verification System (CVM) are usually operated to prevent fraud. Basically, rule-based filter and data mining methods are

used for the prevention [JAH+08].

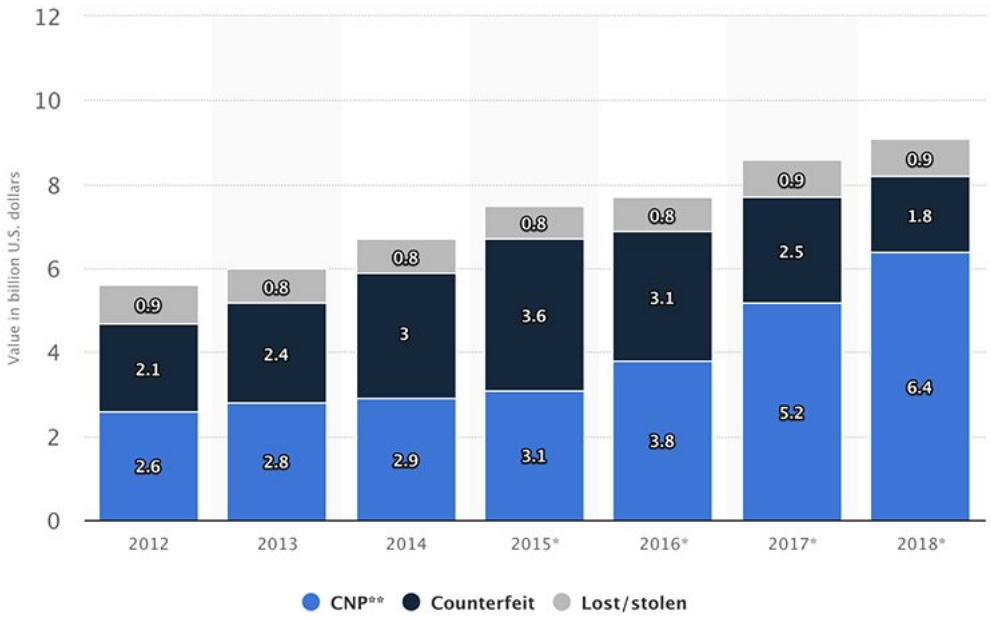

Figure 1.1: Losses due to Card fraud in the U.S. during 2012-2018 as reported by [sta]

However, when fraud cannot be prevented from occurring, then it has to be detected as soon as possible, and necessary actions should be taken against it. Fraud detection is the process of detecting whether a transaction is legitimate or not [MTVM93]. Automated fraud detection systems are required especially considering the huge traffic of transaction data, and it is not possible for humans to check manually every transaction one by one if it is fraudulent or not. This thesis is based on the automatic fraud detection system using machine learning techniques.

1.1 Fraud detection process

The transactions are first checked at the terminal point to be valid or not, which is shown in figure 1.2. At the terminal point, certain essential conditions such as sufficient balance, valid PIN (Personal Identification Number), etc. are validated and the transactions are filtered accordingly. All the valid transactions are then scored by the predictive model, which then classifies the transactions as genuine or fraudulent. The investigators investigate each fraudulent alert and provide feedback to the predictive model to improve the model's performance [Poz15]. This thesis only deals with the predictive model.

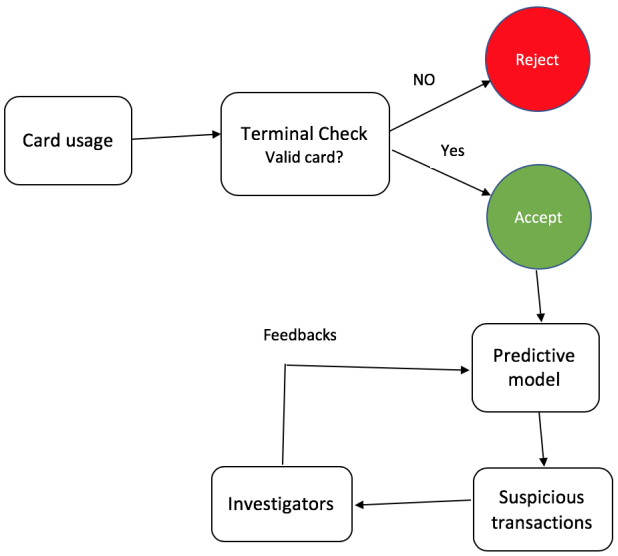

Figure 1.2: Fraud detection process

1.2 Challenges in fraud detection

Building a fraud detection system is not as straightforward as it looks. The practitioner needs to determine which learning strategy to use (e.g., supervised learning or unsupervised learning), which algorithms to use (e.g., Logistic regression, decision trees, etc.), which features to use, and most importantly, how to deal with the class imbalance problem (fraudulent cases are extremely sparse as compared to the legitimate cases) [Poz15]. Class imbalance is not only the major concern in fraud detection system. Overlapping of the genuine and fraudulent classes due to limited information about the transaction records is another problem in the classification task [HAP89], and most machine learning algorithms underperform under these scenarios [JS02].

In a real-life scenario, a fraud detection model predicts the nature of class (genuine or fraudulent) and gives the alert for the most suspicious transaction to the investigators. Investigators then perform a further investigation and provide feedback to the fraud detection system to improve its performance. However, this process can be an overhead for the investigators due to which only a few transactions are validated on time by the investigators. In such a case, just a few feedbacks are provided to the predictive model, which generally results in a lesser accurate model [PBC+15].

Lastly, as financial institutes very rarely disclose the customer data to the public due to confidentiality issues, the real financial datasets are very hard to find. This is one of the major challenges in fraud detection research work [PCB+14].

1.3 Objective

The main objective of this thesis is to perform predictive analysis on credit card transaction dataset using machine learning techniques and detect the fraudulent transactions from the given dataset. The focus is to identify if a transaction comes under normal class or fraudulent class using predictive models. Different sampling techniques will be implemented to tackle the class imbalance problem and series of machine learning algorithms like logistic regression, random forest and xgboost will be implemented on the dataset, and the results will be reported.

1.4 Outline

In chapter 1, we discussed the impacts of fraud in the financial market, an overview of the fraud detection process, challenges in fraud detection system, and the proposed approach to build a fraud detection system.

Chapter 2 will focus on some of the previous research works performed on the credit card fraud detection and then we will move on to some background knowledge of Machine Learning.

In chapter 3, we will describe the methodology that has been used for the analysis.

In chapter 4, we will look into the experimental comparisons of several models for the unbalanced streams of data.

Finally, In chapter 5, we will summarize the results and end with some insights about future work.

Chapter 2

Background and Preliminaries

2.1 Related Work

Since credit card fraud has become a huge problem to the financial market, many financial institutions have spent a huge amount of money and gathered teams of the human expert to develop fraud detection systems. Many researchers have been actively working on to mitigate the problems that are faced while building a fraud detection system (FDS) such as class imbalance problem, class overlapping, change in fraud behaviors and so on. In this chapter, we will talk about some of the past research works in this field.

In a recent study, Pozzolo et al. [PCB+14] have focussed on two different ways of fraud detection: static approach, where a detection model is trained in a seasonal manner (e.g., once a month or year) and online approach, where the model is updated immediately after the new transaction data arrives. They have stated that the online learning approach is a better approach since the fraud behaviors change time by time. Pozzolo et al. [PCB+14] also proposed that the Average Precision (AP), Area Under Curve (AUC), and PrecisionRank are the best measures for the fraud detection task. In another study, Pozzolo et al. [PBC+15] have concluded that random forest is the best approach in the fraud detection task.

Awoyemi et al. [AAO17] have performed a data analysis of K-nearest neighbor, logistic regression, and Naive Bayes when applied on a credit card transaction data, which was further resampled using Synthetic Minority Over-sampling Technique (SMOTE). The result showed that K-nearest neighbor outperformed the other two. The performance was measured in terms of recall, precision, balanced classification rate, specificity, and Mathews correlation coefficient.

In a study, Lebichot et al. [LBCS16] have proposed a graph-based approach to develop a fraud

detection system. In this approach, a collective inference algorithm was used to pass the fraudulent influence in a network by using some fraudulent transaction data. Lebichot et al. [LBCS16] have made several improvements in an existing fraud detection system: APATE (Anomaly Prevention using Advanced Transaction Exploration), to achieve their goal.

Carcillo et al. [CPB$^+$18] have proposed a unique solution to the fraud detection task. They have explored the possibilities of big data technology in fraud detection domain using big data tools such as Cassandra, Spark, and Kafka and came up with a detection system called Scalable Real-time Fraud Finder (SCARFF). They mainly emphasized the fact that as fraud detection demands a real-time setting and because of the massive amount of transaction data, a system with higher scalability, fault-tolerant, and accuracy is needed and big data technology has all these advantages over conventional system architecture.

Domingos [Dom99] has proposed a cost-sensitive model called MetaCost. In his study, he has stated that the cost of all the misclassification errors is not the same. So he has proposed a method to wrap a cost-minimizing procedure in the classifier which considers the misclassification cost. From the experiment, he concluded that there is a systematic decrease in the total cost while using MetaCost as compared to other in-sensitive classifiers.

Aleskerov et al. [AFR97] have proposed a database mining system that uses neural networks and is used for fraud detection. Srivastava et al. [SKSM08] have proposed a Hidden Markov Model (HMM) in which the spending habit of the customer is analyzed in order to detect the fraudulent behavior. Wheeler and Aitken [WA00] have investigated multiple algorithms to detect fraud, and the results showed that the adaptive approach could filter and order the fraud cases that will reduce the number of fraud investigations. In another research study, Baader and Krcmar [BK18] have proposed an automated feature engineering approach to reduce the higher false positive rate that is usually observed in the fraud prediction results.

Although, there have been multiple reseach studies carried out on the different aspects of fraud detection such as tracking the cardholder behavior, improving the fraud detection processing time, addressing the misclassification costs and various data mining techniques, very little research has been performed in the methods to tackle the class imbalance problem in the fraud detection task. Therefore, our thesis aims to perform predictive analysis on the credit card fraud detection task that mainly focuses on various techniques such as resampling, ensemble and hybrid approaches to deal with the class imbalance problem.

2.2 Preliminaries

Before going further into the specific application domain, it is better to get familiar with some of the vital machine learning theories. This chapter will help the reader to understand the proposed fraud detection model using machine learning. More specifically, we will be focussing on the supervised learning, in which the model will be trained with previously labeled data.

2.2.1 Machine Learning

In general context, machine learning can be defined as a field in artificial intelligence that provides the system the capability to learn from the experience automatically without the human intervention and aims to predict the future outcomes as accurate as possible utilizing various algorithmic models. Machine Learning is very different than the conventional computation approaches, where systems are explicitly programmed to calculate or solve a problem. Machine learning deals with the input data that are used to train a model where the model learns different patterns in the input data and uses that knowledge to predict unknown results. The application of machine learning is incredibly vast. It is used in various applications like the spam filter, weather prediction, stock market prediction, medical diagnosis, fraud detection, autopilot, house price prediction, face detection, and many more.

Typically, machine learning has three categories: supervised, unsupervised and reinforcement learning. This thesis deals with supervised learning and we will discuss it in the next section. For now, we can define supervised learning as the approach where the model is trained with both input and output labels. In contrast, unsupervised learning is where the dataset has input labels, (i.e., a model is trained with unlabeled data), from which it learns different patterns and structures. Usually, it is implemented in applications like visual recognition, robotics, speech recognition and so on. Reinforcement learning deals with learning how to obtain a complex goal by maximizing along a specific dimension step by step, (e.g., maximizing the points won in every round [sky]).

2.2.2 Supervised Learning

Supervised learning can be defined as a machine learning approach in which both input and output labels are provided to the model to train. The supervised model uses the input and output labeled data for training, and it extracts the patterns from the input data. These extracted patterns are used to support future judgments. Supervised learning can be formally represented as follows:

$$Y = f(x)$$

where x represent the input variables, Y denotes an output variable and f(X) is a mapping function. The goal is to approximate mapping function such that when an unseen input is given to the mapping function, it can predict the output variable (Y) correctly. Furthermore, supervised learning has two sub-categories: classification and regression. In a classification problem, the output variable is a category, (e.g., fraud or genuine, rainy or sunny, etc.). In a regression problem, the output variable is a real value, (e.g., the price of a house, temperature, etc.). This thesis only deals with the classification problem.

2.2.3 Classification

Classification problem in machine learning can be defined as the task of predicting the class label of a given data point. For example, fraud detection can be identified as a classification problem. In this case, the goal is to predict if a given transaction is fraud or genuine. Generally, there are three types of classification: binary classification, where there are two output labels (e.g., classifying a transaction which may be fraud or genuine), multi-class classification, where there are more than two output labels (e.g., classifying a set of images of flowers which may be Rose or Lilly or Sunflower) and multi-label classification, where the data samples are not mutually exclusive and each data samples are assigned a set of target labels (e.g., classifying a crab on the basis of the sex and color in which the output labels can be male/female and red/black). This thesis deals with the binary classification problem where the output label is either normal or fraud.

2.2.4 Class Imbalance problem

Most real-world applications possess unbalanced class distribution where the number of a class label heavily dominates the count of another class label. One of the best example to explain class imbalance problem is the fraud detection task, where the number of fraud class label is very low as compared to the normal class label. Most machine learning algorithms work poorly in the presence of unbalanced class distribution (i.e., the predictive model tends to classify the minority example as the majority example). So some questions may arise such as: (i) How to tackle the class imbalance problem? (ii) Which machine learning algorithms should be applied in the presence of unbalanced class distribution? (iii) What evaluation metrics should be used to assess the performance of a predictive model when the dataset is highly unbalanced? In the next sections, we will discuss the solutions to these questions.

2.2.5 Handling class imbalance problem

This section explains various approaches that can solve the class imbalance problem. We can roughly classify the approaches into three categories: resampling approach, ensemble-based approach, and cost-sensitive learning approach. This thesis only deals with resampling approach and ensemble-based approach which we will go through them in details in the next sections. Just to give a brief overview, cost-sensitive learning takes misclassification costs into consideration. For example, in medical diagnosis of cancer, the misclassification cost of missing a cancer is much higher than the cost of predicting that a healthy person has cancer. Hence, by considering the misclassification cost of minority class more heavily than that of majority class, the true positive rate of the model can be improved.

2.2.5.1 Resampling approach

Most of the predictive model works worst in the presence of unbalanced class distribution. Therefore, some data preprocessing task has to be performed before providing data as an input to the model. In the case of class imbalance problem, such data preprocessing is performed using a data level approach, which is called resampling approach. Basically, there are three resampling approaches: undersampling, oversampling, and hybrid.

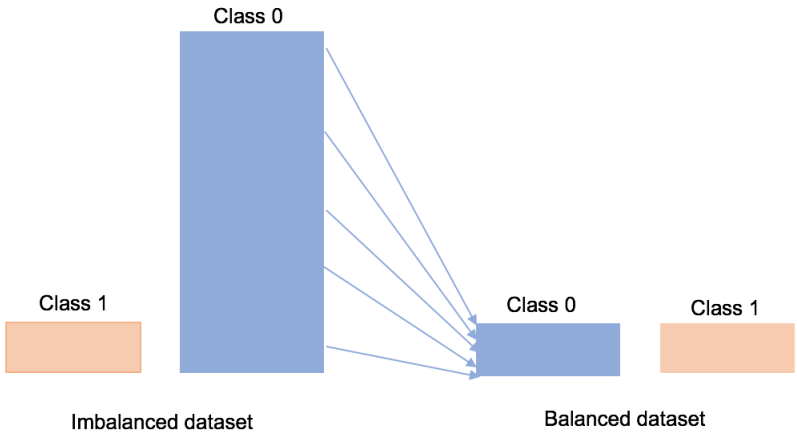

Figure 2.1: Undersampling approach

In the undersampling method, the majority class is reduced in order to make the dataset balanced, which is shown in figure 2.1. It is best to implement when the size of the dataset is huge and reducing the majority samples can greatly boost the runtime and reduce the storage troubles. An oversampling method is exactly opposite to the undersampling method. This method works

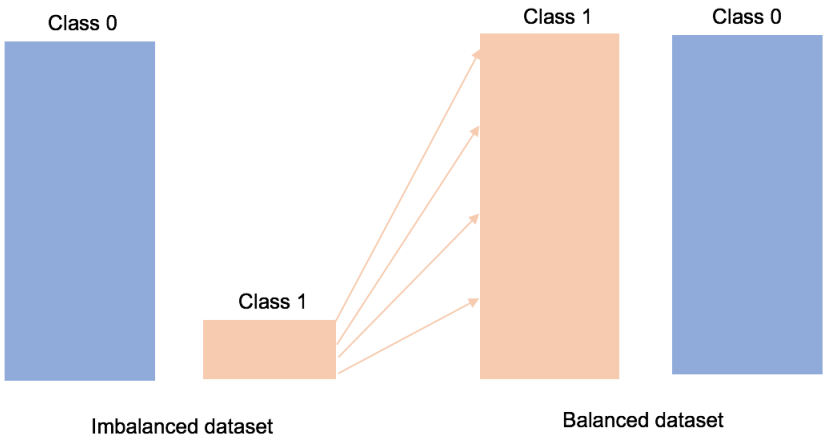

Figure 2.2: Oversampling approach

with the minority class. It replicates the observations from minority class to balance the ratio between majority and minority sample, which is shown in figure 2.2. At last, a hybrid method applies both undersampling and oversampling method for rebalancing purpose. We will discuss some of the resampling approaches in the following subsections.

2.2.5.1.1 Random Undersampling This method randomly eliminates the majority samples in order to make the dataset balanced. This method is best to use when the size of the training data is huge. By reducing the frequency of majority samples, it improves the runtime and also reduces the storage troubles. However, the disadvantage of using such an approach is that some useful information may be eliminated in the process of eliminating majority samples. As a result, the classifier prediction may not be very accurate.

2.2.5.1.2 Tomek Link Removals Tomek Link is a pair of examples of different classes which are each other's nearest neighbors. Given two samples E1 and E2 belonging to different classes, a pair (E1, E2) is a Tomek Link if there's not a sample E3 such that the distance between E1 and E3 is less than that of E1 and E2 or the distance between E2 and E3 is less than that of E1 and E2 [BPM04]. Removing Tomek links can be considered as an undersampling approach, where the majority sample in the Tomek link is eliminated.

2.2.5.1.3 Random Oversampling This method randomly replicates the minority samples to make the dataset balanced. Unlike random undersampling, this approach does not lead to information loss. However, there's a high possibility of overfitting the data since it replicates the

minority samples.

2.2.5.1.4 Synthetic Minority Over-sampling Technique (SMOTE) SMOTE is a popular technique used to rebalance the dataset and it was developed by Chawla [BCHK02]. It aims to create new minority class examples (synthetic instances) by interpolating between several nearest minority examples rather than by oversampling with replacement, which is shown in figure 2.3. As a result, it diminishes the problem of overfitting of the training data. Depending upon the amount of oversampling required, nearest neighbors of minority examples are randomly selected.

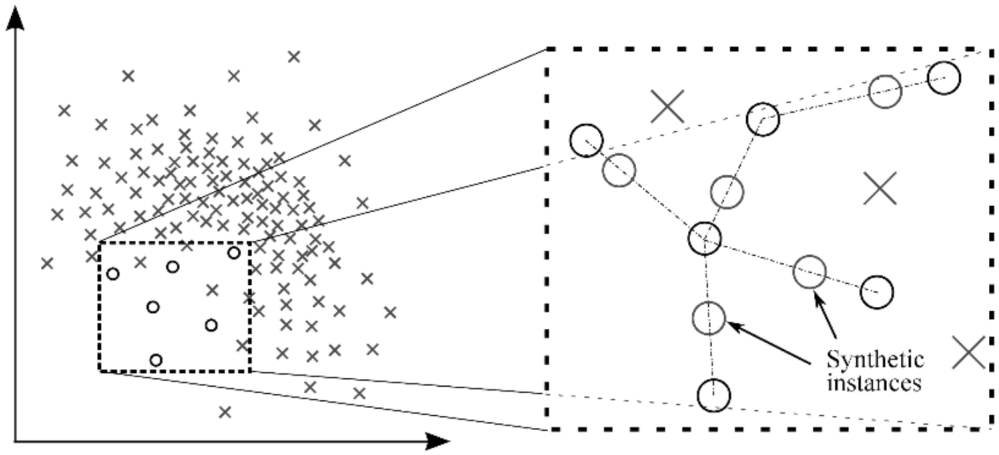

Figure 2.3: Generation of synthetic examples using SMOTE [Blo18]

2.2.5.1.5 Combination of SMOTE and Tomek Link removal SMOTE is a powerful approach to balance the class distributions. However, while creating new synthetic minority examples, the minority class cluster might invade the majority class space. Providing such data to the model can lead to overfitting. Hence, to mitigate such a situation, both SMOTE and Tomek Link removal approach can be applied for balancing the class distribution. In this process, the original training dataset is first oversampled using SMOTE, and then Tomek Link removal is applied to the resulting dataset producing a balanced dataset.

2.2.5.2 Ensemble approach

In the previous section, we discussed the data-level approach in which resampling techniques are used to balance the class distributions. In this section, we will discuss the algorithmic approach which is called the ensemble approach. Ensemble approach deals with modifying existing classification algorithms to tackle the unbalanced class distribution. In general, an ensemble approach is

a learning algorithm that assembles a set of classifiers and applies them for classification by taking a vote of their predictions [Die00], which is also shown in figure 2.4. Typically, there are two types of the ensemble approach: bagging and boosting.

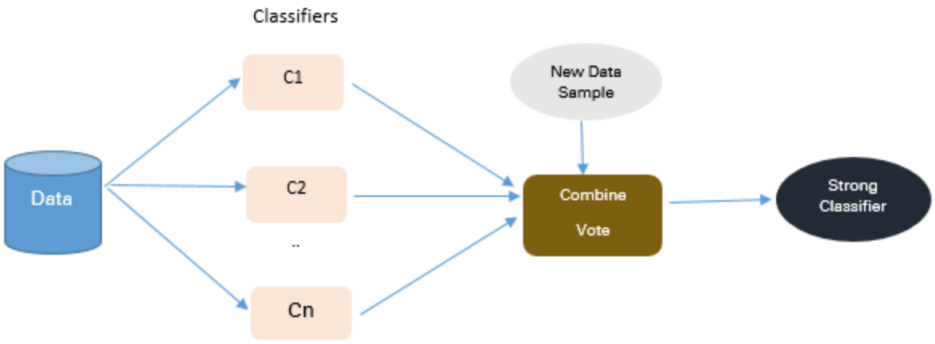

Figure 2.4: Ensemble approach [Blo18]

Before moving forward to bagging, let's first talk about an essential concept of bootstrapping which is used in bagging algorithms. In machine learning, bootstrapping is the process of sampling the training data randomly with replacement. Each bootstrap sample is sampled in such a way that each sample will have different characteristics as shown in Figure 2.5. When the models use these samples for training, they can learn various aspects of the data and can improve the prediction performance.

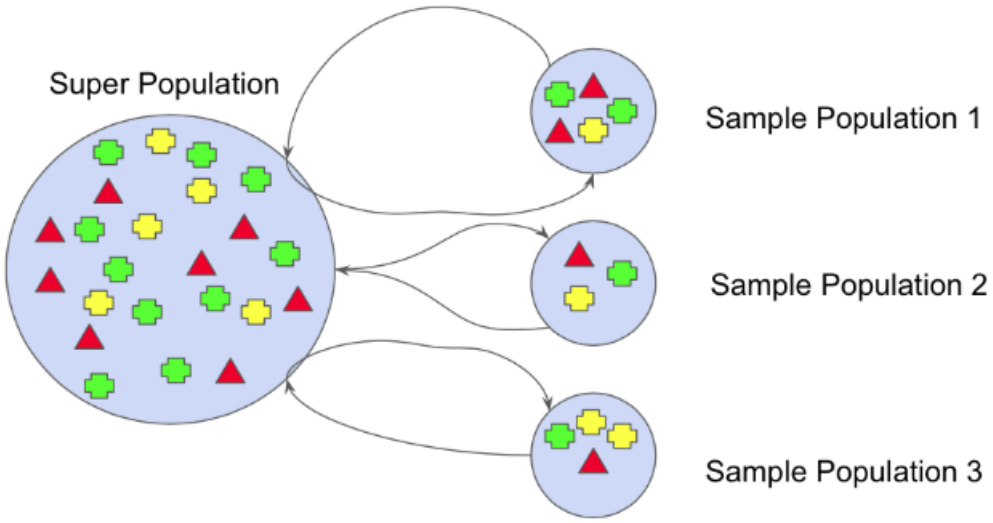

Figure 2.5: Bootstrapping [Sea17]

2.2.5.2.1 Bagging Bagging which is an abbreviation form of Bootstrap Aggregation is a simple yet very powerful ensemble technique. This method involves bootstrapping that generates new training samples from the original training set with replacement. These new training samples are called bootstrap training samples. Each bootstrap sample is used for training the individual model separately, which are then used for prediction. Finally, the predictions from all the bootstrapped models are aggregated by averaging the output (for regression) or voting (for classification). Figure 2.6 gives a better picture of bagging. It helps to reduce overfitting and variance. Decision trees are usually used as a base model in bagging. However, other types of methods can also be used. This thesis deals with the random forest algorithm as a bagging method.

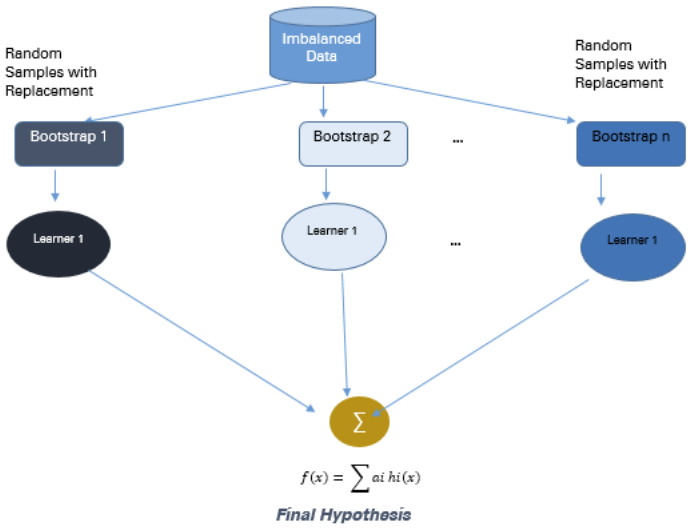

Figure 2.6: Overview of bagging [Blo18]

2.2.5.2.2 Boosting Boosting is another very powerful ensemble technique. It involves combining weak learners, which are also called base learners, to create a strong learner that can give a better result as compared to the results generated by an individual learner. Unlike bagging in which each model runs parallelly and then the outputs are combined at the end, the boosting deals with training the weak learners sequentially, such that each learner tries to correct its predecessor by adding more weights to the samples that were previously misclassified. Therefore, the future weak learner will focus more on the misclassified cases. Figure 2.7 gives a better picture of boosting. It also uses bootstrapping due to which it also avoids overfitting and variance. There are many examples of boosting algorithm like ada boost, gradient boost, xgboost, etc. This thesis deals with xgboost.

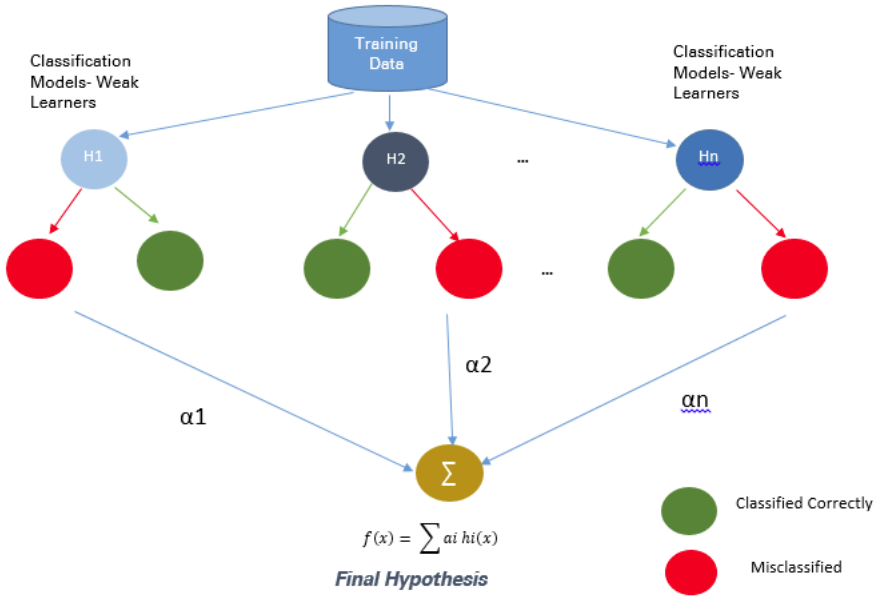

Figure 2.7: Overview of boosting [Blo18]

2.2.6 Selected models

In this section, we will discuss the different models selected for the predictive analysis. Depending on the nature of the classification problem, we chose three very popular predictive model. They were logistic regression, random forest, and xgboost.

2.2.6.1 Logistic regression

Logistic regression is one of the most popular machine learning algorithms that is used for classification. Although the term 'regression' appears in the name, it is not a regression algorithm. Logistic regression has its name as it was built on another very popular machine learning algorithm, linear regression, which is used for regression problem. In logistic regression, the prediction is expressed in terms of probability of outcome belonging to each class. In a linear regression model, the real-valued outputs are predicted by combining the input variables (x) with the weights. To be more clear, consider there is just one input or independent variable 'x' and a dependent variable 'y'. Then the hypothesis of linear regression can be expressed as

$$y = a0 + a1 * x \tag{2.1}$$

where a0 is a bias term, and a1 is the weight for single input variable x. These weights are learned during the training. In this case, the value of the hypothesis can be less than 0 or greater than 1.

Logistic regression also uses such a linear equation. However, since it should predict the probability of the outcome of belonging to each class, it uses a sigmoid function or a logistic function which is shown in equation 2.2, to squash the predicted real values between the range of 0 and 1.

$$sigm(z) = \frac{1}{1 + e^{-}z} \tag{2.2}$$

Figure 2.8 shows how the sigmoid function looks like. In a classification problem where we have one independent variable 'x' and one dependent variable 'y', logistic regression can be represented as in equation 2.3. By default, logistic regression uses a threshold of 0.5 such that any probability below 0.5 is classified as class 0, and any probability above 0.5 is classified as class 1. This threshold can be adjusted according to the needs.

$$P(y = 1) = sigm(a0 + a1 * x) \tag{2.3}$$

where a0 and a1 are the parameters of the logistic regression model that are learned during training. Therefore, with a threshold of 0.5, the predicted output can be represented as follows.

$$\text{y} = 1 \text{ if P(y=1)} \geq 0.5$$

$$\text{y= 0 if P(y=1)} < 0.5$$

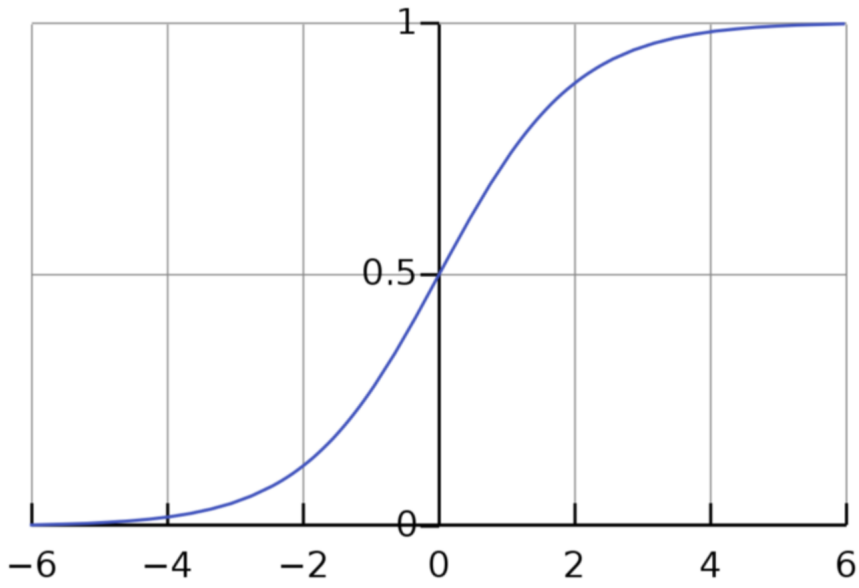

Figure 2.8: Sigmoid function graph [Gan18]

2.2.6.2 Random forest

Random forest is an ensemble learning algorithm, which can be used for both regression and classification task. Specifically, it is an extension of bagging. In section 2.2.5.2.1, we explained that the bagging method involves the combination of many weak learners. In a random forest, these weak learners are decision trees. So, before going into the details of the random forest, we will try to understand the basics of decision trees.

A decision tree is a supervised learning algorithm, which can be used for both regression and classification. But, it is mostly used in classification problems. It is composed of several internal nodes where each node represents a test in an attribute (e.g., whether tomorrow's weather is sunny or overcast or rainy). Each branch in the tree represents the outcome of the test and leaf nodes represent the final outcome (class label). It involves breaking down a training set into several subsamples.

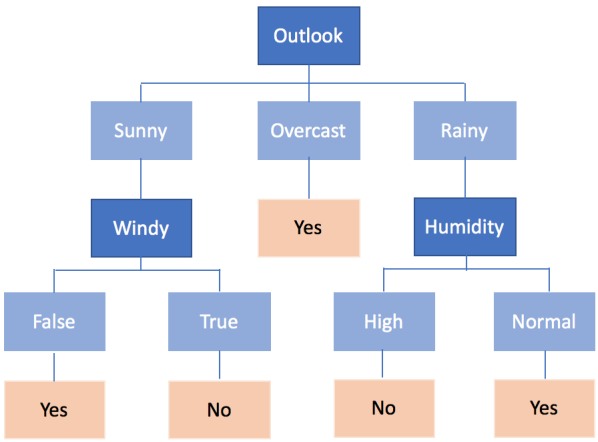

Figure 2.9: An example of decision tree [Lib17]

Figure 2.9 shows a simple example of a decision tree where it tries to decide whether to play golf tomorrow or not. It starts with an outlook which has three choices: sunny, overcast and rainy. If it is sunny, then also check if it is windy (true or false). If it is true, then we decide not to play golf that day. If the answer is false, then we choose to play. If it is overcast, then we can choose to play that day. If the outlook is rainy, we should also check the humidity. If humidity is high, we choose not to play, but if the humidity is normal, then we can play golf that day. Hence, this example clearly shows that a decision tree involves a series of if-then conditions that are used to make the final decision.

The construction of a decision tree involves splitting of entire training data into subsets, which

are performed in each internal node based on some requirement. The decision tree algorithm determines the best split of each node on the basis of metrics such as Gini impurity and Information gain. Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. Information gain is used for deciding which feature to split on at each step in building the tree. The process of splitting continues until the internal node has the class label value. Although decision trees are easy to understand and perform well in some datasets, they tend to have a high variance because of the greedy approach of the algorithm where the tree tends to always select the best split at each level and it cannot see far behind the current level. Due to this reason, there may be the possibility of overfitting, where the model only performs better in the training set and fails to perform well in test sets.

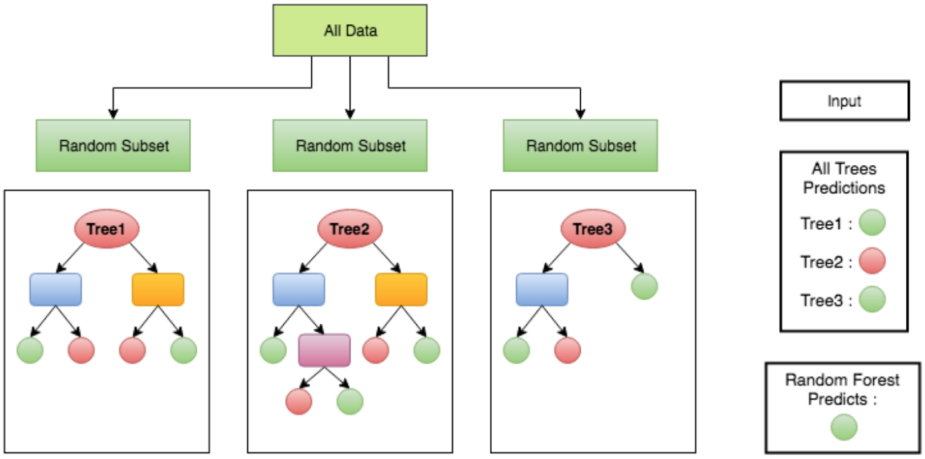

Figure 2.10: An example of random forest [Lib17]

Random forest algorithm mitigates the overfitting problem well by using the bootstrap concept which we learned in section 2.2.5.2. In simple language, the random forest builds multiple decision trees and combines them to improve the performance of the model as a whole. As we discussed earlier, bootstrapping is the process of sampling the training data randomly with replacement. Random forest utilizes bootstrapping such that each decision tree will be trained with different subsamples of data.

Moreover, the random forest uses random subsets of features. For example, if there are 50 features in the data, random forest will only choose a certain number of them, let's say 10, to train on each tree. Thus, each tree will have 10 random features that will be used for training including

finding the best split of each node of the tree. Once we have the collection of decision trees, the results of each tree will be aggregated to get the final result (vote). The model trained in such a way will ensure generalization since not one, but multiple decision trees are used for making the decision, and moreover, each tree is trained with different subsections of data. Figure 2.10 shows a clear explanation of the construction of the random forest from the given input data.

2.2.6.3 XGBoost

XGBoost is the abbreviation form of eXtreme Gradient Boosting. It is an advanced implementation of gradient boosting proposed by Chen and Guestrin [CG16]. Since XGBoost improves on gradient boosting algorithm, we will briefly explain it. Gradient boosting is also a boosting algorithm, which tries to combine the weak learners to form a strong learner. It generates weak learners during the learning process. At each level of the process, the weak learner predicts the values or class label and then calculates the loss, (i.e., the difference between real value and the predicted value). Depending upon the loss, it creates a new weak learner and then the weak learner trains on the remaining errors. This process continues until a certain threshold. This process is called gradient descent optimization problem and therefore this algorithm is called gradient boosting. It is another way for giving high preference to the misclassified samples whereas in boosting algorithm like Ada boost, the weights of the misclassified samples are given higher values, so that next weak learner works on it.

As we mentioned earlier, xgboost is an advanced implementation of gradient boosting algorithm. It uses decision trees as the weak learners and it has many advantages over standard gradient boosting algorithm. XGBoost has better regularization than gradient boosting. Therefore, it reduces overfitting. XGboost allows parallel processing, so it is much faster than standard gradient boosting. XGBoost has the inbuilt capability to handle missing data. Gradient boosting is a greedy algorithm since it stops splitting the node as soon as it encounters a negative loss in the split whereas xgboost splits up to the maximum depth specified. XGBoost has built-in cross-validation feature, so it is easier to determine the number of boosting rounds at each run. There are quite a few hyperparameters that need to be tuned in order to get the best result from xgboost algorithm.

2.2.7 Evaluation metrics

In machine learning, we train the model with the training data, and then we check the generalization capability of the model. In simple terms, we examine how the model performs when tested on data

that was unseen. So how do we measure the performance of the model? We use evaluation metrics for evaluating the performance of the model depending on the nature of the problem (whether it is a regression or classification). In this section, we will only discuss the evaluation metrics related to the classification problem.

2.2.7.1 Confusion matrix

It is the most commonly used evaluation metrics in predictive analysis mainly because it is very easy to understand and it can be used to compute other essential metrics such as accuracy, recall, precision, etc. It is an NxN matrix that describes the overall performance of a model when used on some dataset, where N is the number of class labels in the classification problem. For binary classification, we have a 2x2 confusion matrix as shown in figure 2.11.

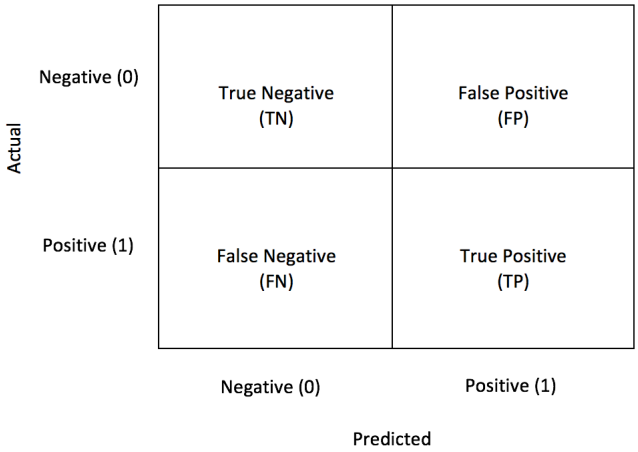

Figure 2.11: Confusion matrix

A confusion matrix is composed of statistics such as True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) which are calculated using the combination of actual and predicted values.

True Positive (TP) is a case where the actual value was positive (e.g., fraud) and the predicted value is also positive.

False Positive (FP) is a case where the actual value was negative (e.g., normal) but the predicted value is positive.

True Negative (TN) is a case where the actual value was negative (e.g., normal) and the predicted value is also negative.

False Negative (FN) is a case where the actual value was positive (e.g., fraud) but the predicted value is negative.

2.2.7.2 Recall

Recall, also known as sensitivity, is the fraction of true positives to the actual positive cases, which is shown in equation 2.4. In simple terms, recall is how many of true positives were found (recalled) out of all the true positive cases.

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

For example, consider a picture containing 12 apples and some oranges. Suppose a model identifies 8 apples in the picture, and out of 8 recognized as apples, only 5 actually were apples (TP), while rest were oranges (FP). In this case, recall is 5/12.

2.2.7.3 Precision

It is the fraction of true positives over the true positives and false positives, which is shown in equation 2.5. In simple terms, precision is how many of the found cases were true positives.

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

In the above example of identifying apples, the precision value is 5/8.

2.2.7.4 F_1 Score

F_1 Score also called F score or F-measure is the harmonic mean of the recall and precision. Its value ranges from 0 to 1, where 0 is considered worst, and 1 is considered best. It can be calculated as follows.

$$F_1 = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{2.6}$$

2.2.7.5 Area Under Receiver Operating Characteristic curve

Area Under Receiver Operating Characteristic curve is one of the most widely used evaluation metrics in predictive analysis. It tells us how good a model performs when used at different probability thresholds. By default, a probability threshold of 0.5 is used for the classification problem. It is a plot between True positive rate (TPR), which is also called sensitivity and False Positive Rate (FPR). FPR is calculated as (1-Specificity). The equations of sensitivity and specificity are given in equations 2.7 and 2.8 respectively.

$$Sensitivity = \frac{TP}{TP + FN} \tag{2.7}$$

$$Specificity = \frac{TN}{TN + FP} \tag{2.8}$$

Sensitivity and specificity are inversely related as we change the probability threshold, (i.e., when we decrease the threshold, sensitivity increases while specificity decreases and when we increase the threshold, sensitivity decreases while specificity increases). From the ROC curve, we can calculate the area under the curve which is the probability that a model will rank a randomly chosen positive instance higher than a randomly chosen negative one [Bra97]. Figure 2.12 shows an example of a ROC curve where the blue curve shows the ROC of the model and AUC is 0.966. Whereas, the red dotted line shows the ROC of a random model whose AUC is 0.5.

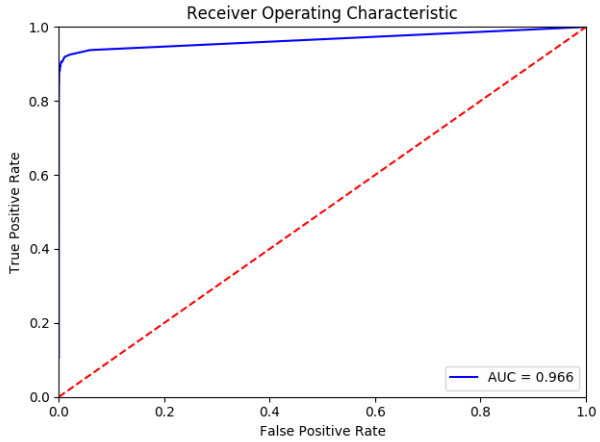

Figure 2.12: An example of ROC curve

2.2.7.6 Area Under Precision Recall Curve

Area Under Precision Recall Curve is another crucial evaluation metrics to measure the performance of a predictive model especially when there is unbalanced class distribution. It is based on two evaluation metrics, precision and recall, and the plot shows the values of precision and recall at different probability thresholds. The area under PR curve can be used to examine the performance of the model. Figure 2.13 shows an example of the PR curve.

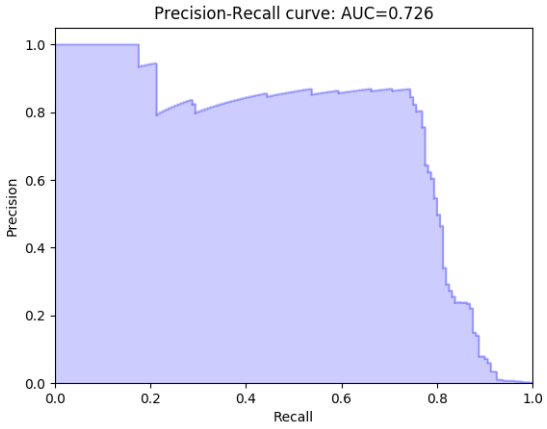

Figure 2.13: An example of PR curve

Chapter 3

Methodology

3.1 Data description

Datasets are an integral part in the field of machine learning. Gathering data is one of the hardest tasks, especially when it is related to the financial domain like credit card fraud. The dataset used in this thesis was originally used for a research project [PCJB15] carried out by Worldline and the Machine Learning Group of ULB (Universit Libre de Bruxelles), and it was also released in Kaggle, a community of data scientists and machine learners. It contains the record of credit card transactions made by European cardholders and occurred in two days in September 2013. The dataset contains 284,807 transactions out of which only 492 are fraudulent. The dataset is highly unbalanced as the positive class accounts for only 0.172% of the total transactions. The unbalanced class distribution can be visualized in a bar diagram given in figure 3.1.

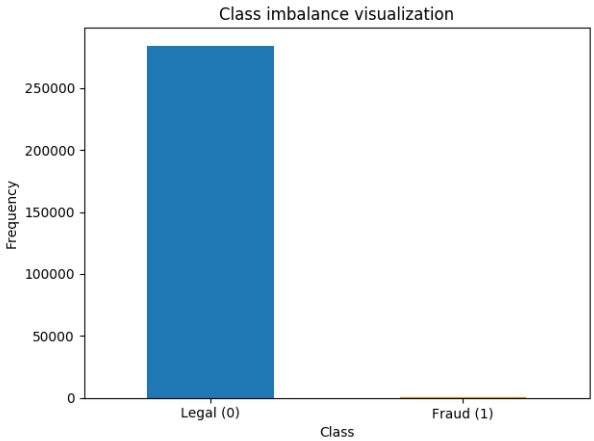

Figure 3.1: A visualization of highly unbalanced class distribution

The dataset contains the numerical values generated from the Principal Component Analysis (PCA)

transformation. However, due to the confidentiality issue, the original features have not been disclosed. There are total 30 features out of which 28 features have been generated by principal component analysis. PCA is a dimensionality-reduction technique in which a large number of original variables are condensed into a smaller subset of feature variables. The only features that have not been transformed into principal components are 'Amount' and 'Time'. Table 3.1 shows the definitions of variables used in the dataset.

Table 3.1: Description of the variables in the dataset

Variable	**Type**	**Description**
Time	Integer	Time elapsed between each transaction and the first transaction
Amount	Double	Transaction amount
Class	Integer	Response variable (1=Fraudulent and 0=Legitimate)
V1	Double	First principal component
V2	Double	Second principal component
V3	Double	Third principal component
V4	Double	Fourth principal component
...	...	...
...	...	...
V28	Double	Last principal component

In a predictive analysis, before diving directly into the implementation, it is better to visualize the data first. We can visualize the data in terms of the correlation between variables, where correlation refers to the mutual relationship between the variables. Generally, feature variables with higher correlation with response variable have a more significant impact during the training phase. Figure 3.2 shows a correlation matrix that gives an interpretation of the pairwise correlation between each variable. The given correlation matrix shows that none of the V1 to V28 principal components have any correlation to each other. However, if we observe further, response variable 'Class' has some form of positive and negative correlations with the principal components, but it does not correlate with 'Time' and 'Amount'.

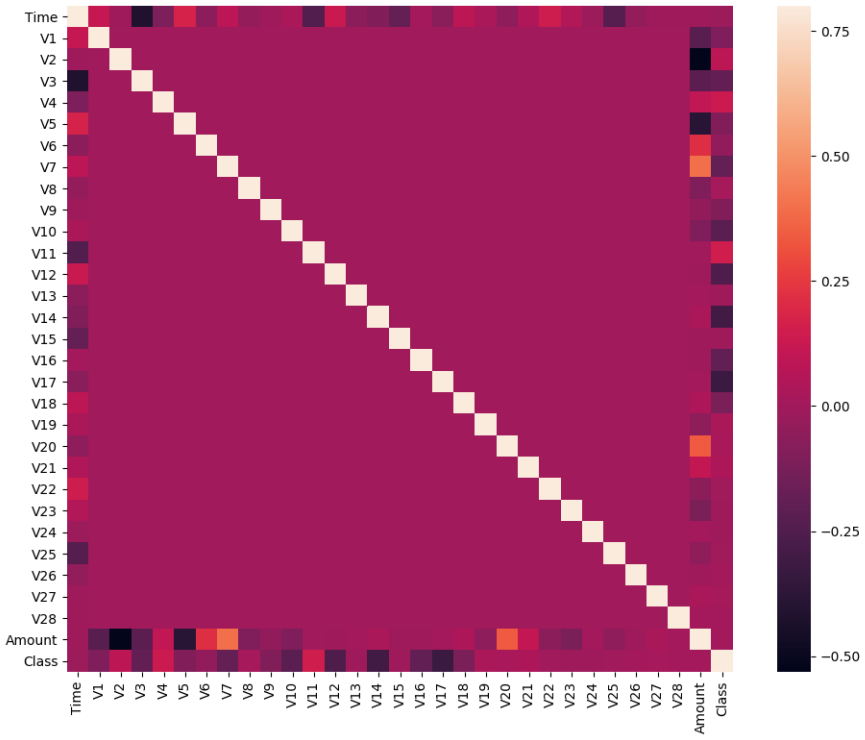

Figure 3.2: A correlation matrix showing the correlations in the data

3.2 Data standardization

Standardizing the features refers to rescaling the features so that they will have the properties of a standard normal distribution with a mean of 0 and standard deviation of 1. It is a common requirement for many machine learning models that the features should be standardized before applying the machine learning techniques. If standardization is not performed, then it might affect the performance of the model. We performed standardization on the 'Amount' feature using StandardScalar in the scikit-learn library. Standardization can be achieved as follows.

$$z = \frac{x - \mu}{\sigma} \tag{3.1}$$

where

$$Mean(\mu) = \frac{1}{N} \sum_{i=1}^{N} (x_{\mathrm{i}}) \tag{3.2}$$

$$Standard\,deviation(\sigma) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (x_{\mathrm{i}} - \mu)^2} \tag{3.3}$$

3.3 Data splitting

For each experiment, we split the entire dataset into 70% training set and 30% test set. We used the training set for resampling, hyperparameter tuning, and training the model and we used test set to test the performance of the trained model. While splitting the data, we specified a random seed (any random number), which ensured the same data split every time the program executed.

3.4 Data resampling

As mentioned earlier, the dataset is highly unbalanced. The number of legitimate transactions outnumbers the number of fraudulent transactions. In this case, if we use this dataset to train our model, the model tends to be biased towards the legitimate transactions, and hence, it results in the poor performance of model when tested in an unseen data. To tackle this problem, we have used some resampling techniques such as random undersampling, random oversampling, SMOTE, Tomek links removal, a combination of SMOTE and Tomek links removal. We implemented these resampling techniques on the training data separately to make it balanced.

3.5 Hyperparameter tuning using 10-fold cross validation

Hyperparameter is a configuration that is external to the model whose value cannot be estimated from the training data. Hyperparameter should not be confused with the model parameter as a model parameter is a configuration that is internal to the model, and its value can be estimated during the training process. Since hyperparameter is external to the model, its value has to be set manually by the practitioner. But before that, the value needs to be rightly tuned to get the best performance from the model. The process here used for tuning the hyperparameter is cross validation technique. More specifically, we used K-fold cross validation where we set the value for K as 10.

In 10-fold cross-validation, we divide the training dataset into 10 folds, and for each fold, we choose the current fold as a test set and remaining folds as a training set. Then, we fit a model into the training set and evaluate it on the test set. Figure 3.3 gives a better picture of 10-fold cross validation. This cross-validation technique can be used for tuning hyperparameters as well. We implemented the best hyperparameter search using sckit learn's grid search function with cross-validation. Since each machine learning model has different hyperparameters, the overall search was different for each model.

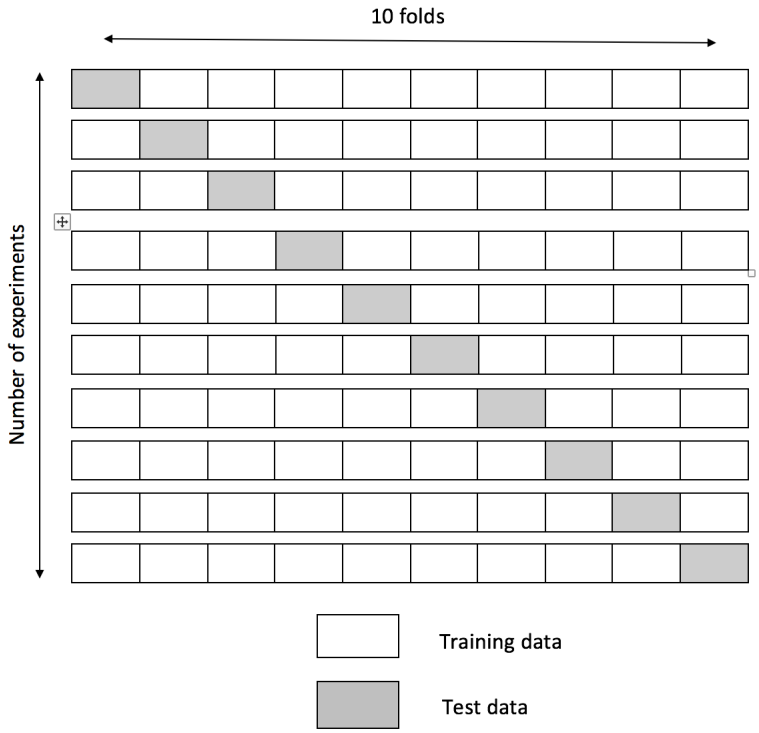

Figure 3.3: 10-fold cross validation

3.5.1 Hyperparameters search in logistic regression

In the logistic regression model, the regularization parameter 'C' is an important hyperparameter that needs to be tuned carefully. The value of C directly affects the generalization ability of the model. For instance, for large values of C, the model tends to overfit the data, and for small values of C, the model tends to underfit the data. Given an initial list of C values, we performed GridSearchCV technique on the resampled training data to find the best C parameter for the logistic regression model.

3.5.2 Hyperparameters search in random forest

We performed grid search with 10-fold cross-validation technique on the resampled training dataset to find the best hyperparameters: n_estimators, which is the number of trees in the forest and max_features, which is the maximum number of features considered for splitting a node.

3.5.3 Hyperparameters search in xgboost

There are many essential hyperparameters in an xgboost model. We performed grid search with 10-fold cross-validation technique on the resampled training dataset to find the value of the following hyperparameter.

n_estimators defines the total number of trees or the total number of boosting rounds.

min_child_weight defines a minimum sum of weights of all observations required in a child. It controls the overfitting.

max_depth is the maximum depth of the tree. Typically, its value lies in the range of 3-10.

gamma is the minimum loss reduction required to make a split. The splitting in a node is done only when the resulting split gives a positive reduction in the loss function.

subsample defines the fraction of observations to be randomly sampled for each tree. Typically, its value lies in the range of 0.5-1.

colsample_bytree denotes the fraction of columns to be randomly sampled for each tree.

alpha denotes the regularization parameter which helps to reduce overfitting.

3.6 Training and testing phase

After tuning the hyperparameters for each predictive model, we set the hyperparameters into each model respectively, and then we passed the resampled training set to each model as the training data. Thus, the models learned different patterns in the resampled training data. Then, we used the test set, which we separated before while splitting the whole dataset, to test the performance of the model.

3.7 Performance evaluation of selected models

The final step of the predictive analysis is the performance evaluation of the model. In this thesis, we evaluated the performance of the models using recall, precision, f1-score, and area under precision-recall curve. One thing to notice that we have not used accuracy for the performance evaluation. The reason for this is accuracy usually gives a misleading conclusion when there is unbalanced class distribution. For example, suppose there are 100 transactions out of which only 3 are labeled as

fraudulent, and rest are legitimate transactions. Suppose, a model predicts all of the transactions as legitimate. In this case, the accuracy of the model is 97% which is great in number. However, the truth is that the model failed to predict all of the fraudulent transactions.

Since this is a fraud detection task, we don't want the predictive model to miss the fraudulent transactions, we want the recall score to be as higher as possible. However, we should not neglect the precision score as well, since we don't want to predict the transaction as fraudulent even if it is not. Since the f1 score is the harmonic mean of recall and precision, it is also a good metric to consider for this type of problem.

Usually, in a classification problem, the ROC curve is the go-to choice for evaluating the performance of the model at different thresholds. However, if there is unbalanced class distribution where the negative examples dwarf the number of positive examples, the precision-recall curve is going to more useful. For illustration, let's take an example of fraud detection problem, where we want to detect, say, 100 fraudulent transactions out of 1 million transactions. Let's say we have got two models for the performance comparison.

Model 1: predicted 100 transactions were fraudulent, but actual 90 transactions were fraudulent.

Model 2: predicted 2000 transactions were fraudulent, but actual 90 transactions were fraudulent.

Clearly, the result of model 1 is preferable since model 2 brings a lot of false positives with it. The TPR and FPR in ROC reflect that, however, since the legitimate transactions outnumber the fraudulent transaction, the difference is mostly lost.

Model 1: 0.9 True Positive Rate, 0.00001 False Positive Rate

Model 2: 0.9 True Positive Rate, 0.00191 False Positive Rate (difference of 0.0019)

Precision and recall don't consider true negatives, hence are not affected by relative imbalance.

Model 1: 0.9 precision, 0.9 recall

Model 2: 0.9 precision, 0.045 recall (difference of 0.855)

Obviously, these are just single points in the ROC and PR spaces, but if these differences remain in various thresholds, we would see a very small difference in the performance of two models while using Area Under Receiver Operating Characteristic curve, whereas Area Under Precision Recall Curve would show a significant difference in the performance.

Chapter 4

Results

4.1 Logistic regression (LR)

No resampling- Best C-Parameter: 0.1

Remarks

The logistic regression performed very well in classifying the legitimate samples even without resampling with precision, recall and f1 scores of 0.9991, 0.9995 and 0.9993 respectively as shown in table 4.1. This was expected since we're dealing with the imbalanced class. However, the performance of this model was not satisfactory when dealing with fraudulent class, where the precision and recall were 0.6774 and 0.5250. In addition, PR AUC (area under precision recall curve) and ROC AUC (area under receiver operating characteristic curve) values, which is shown in figure 4.2 and figure 4.3 respectively, are also not very good. Figure 4.1 shows the confusion matrix of the logistic regression model when none of the resampling methods are used.

Class	Precision	Recall	F1 score	Support
0	0.9991	0.9995	0.9993	85283
1	0.6774	0.5250	0.5915	160
avg/total	0.9985	0.9986	0.9986	85443

Table 4.1: Evaluation metrics-LR-no resampling

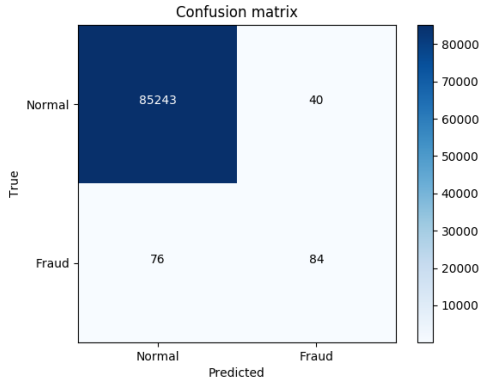

Figure 4.1: Confusion matrix-LR-no resampling

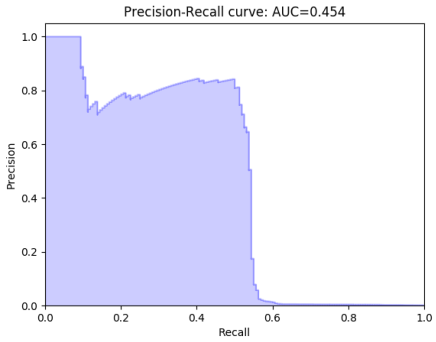

Figure 4.2: PR curve-LR-no resampling

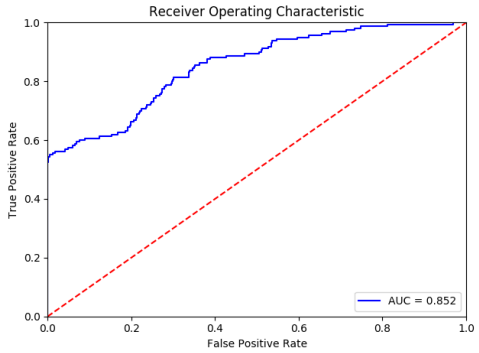

Figure 4.3: ROC curve-LR-no resampling

Random undersampling

Best C-Parameter: 0.1

Remarks

As expected, the logistic regression performed very well in classifying the negative class, when random undersampling was used, which is shown in table 4.2. However, in the case of the positive class, the precision decreased to 0.0857, which is very poor. Even though the recall of 0.9062 is a very good score, we cannot neglect the precision. Also, figure 4.6 shows a ROC curve in which ROC AUC is 0.973, which is very good but PR curve as shown in figure 4.5, gives a different picture in which the precision is very low when the recall score is above 0.90. Figure 4.4 shows the confusion matrix of the logistic regression model when the random undersampling method is used.

Class	Precision	Recall	F1 score	Support
0	0.9998	0.9819	0.9908	85283
1	0.0857	0.9062	0.1566	160
avg/total	0.9981	0.9817	0.9892	85443

Table 4.2: Evaluation metrics-LR-random undersampling

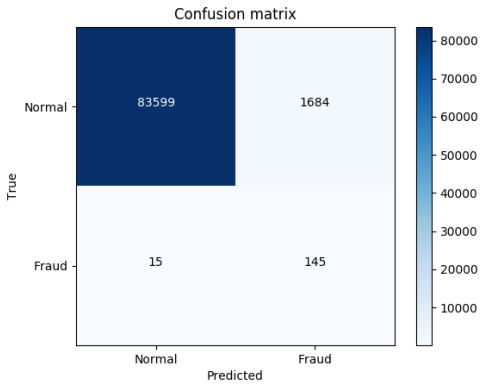

Figure 4.4: Confusion matrix-LR-random undersampling

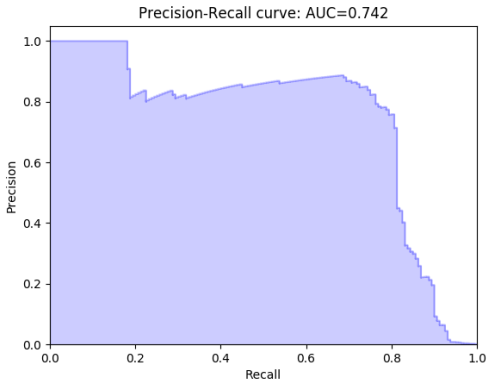

Figure 4.5: PR curve-LR-random undersampling

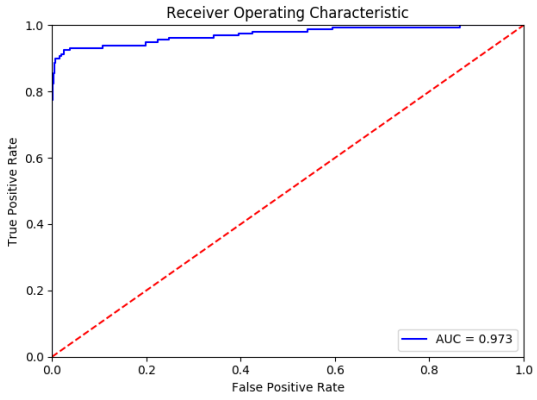

Figure 4.6: ROC curve-LR-random undersampling

Tomek links removal

Best C-Parameter: 0.1

Remarks

Table 4.3 shows the evaluation metrics of the logistic regression when tomek links removal method is used. Similarly, figure 4.7, figure 4.8, and figure 4.9 show the confusion matrix, the PR curve, and the ROC curve respectively. Just like with the previous two models, the logistic regression with tomek links removal performed very well in classifying the negative class. As compared to random undersampling, the precision improved quite a lot but the recall score decreased by quite a big margin.

Class	Precision	Recall	F1 score	Support
0	0.9991	0.9995	0.9993	85283
1	0.6746	0.5312	0.5944	160
avg/total	0.9985	0.9986	0.9986	85443

Table 4.3: Evaluation metrics-LR-tomek links removal

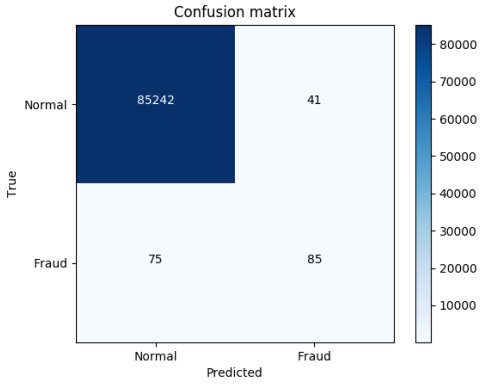

Figure 4.7: Confusion matrix-LR-tomek links removal

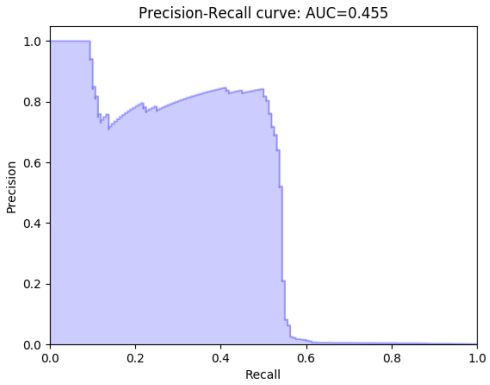

Figure 4.8: PR curve-LR-tomek links removal

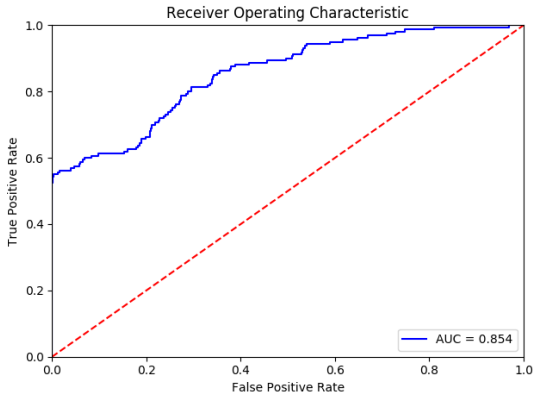

Figure 4.9: ROC curve-LR-tomek links removal

Random oversampling

Best C-Parameter: 0.001

Remarks

Table 4.4 shows the evaluation metrics of the logistic regression when the random oversampling method is used. Similarly, figure 4.10, figure 4.11, and figure 4.12 show the confusion matrix, the PR curve, and the ROC curve respectively. The logistic regression with random oversampling performed very well in terms of the recall but performed worst in terms of precision, thus giving a poor f1 score of 0.1340.

Class	Precision	Recall	F1 score	Support
0	0.9998	0.9784	0.9890	85283
1	0.0724	0.9000	0.1340	160
avg/total	0.9981	0.9782	0.9874	85443

Table 4.4: Evaluation metrics-LR-random oversampling

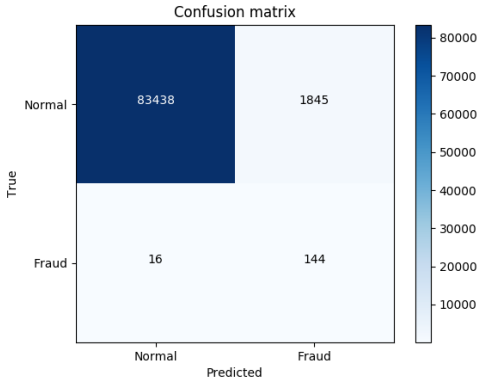

Figure 4.10: Confusion matrix-LR-random oversampling

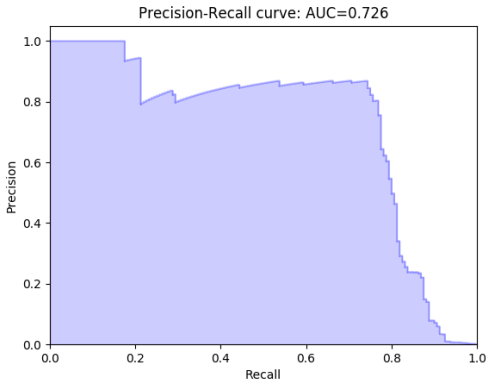

Figure 4.11: PR curve-LR-random oversampling

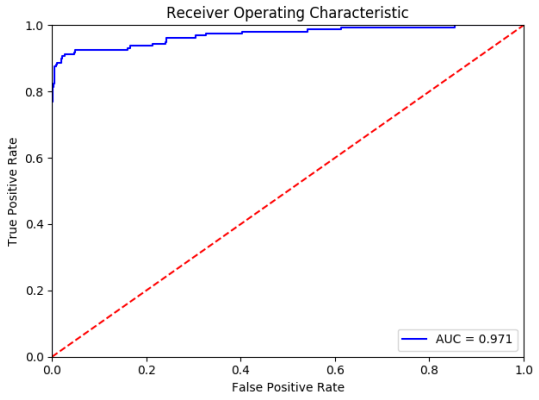

Figure 4.12: ROC curve-LR-random oversampling

SMOTE

Best C-Parameter: 0.01

Remarks

Table 4.5 shows the evaluation metrics of the logistic regression when Synthetic Minority Oversampling Technique (SMOTE) is used. Similarly, figure 4.13, figure 4.14, and figure 4.15 show the confusion matrix, the PR curve, and the ROC curve respectively. Using synthetic minority over-sampling technique with logistic regression also did not help to improve the performance as the f1 score was only 0.1695.

Class	Precision	Recall	F1 score	Support
0	0.9998	0.9840	0.9918	85283
1	0.0938	0.8812	0.1695	160
avg/total	0.9981	0.9838	0.9903	85443

Table 4.5: Evaluation metrics-LR-SMOTE

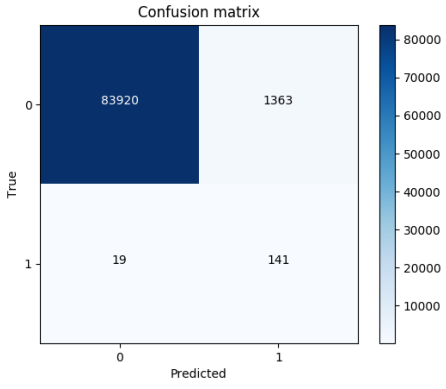

Figure 4.13: Confusion matrix-LR-SMOTE

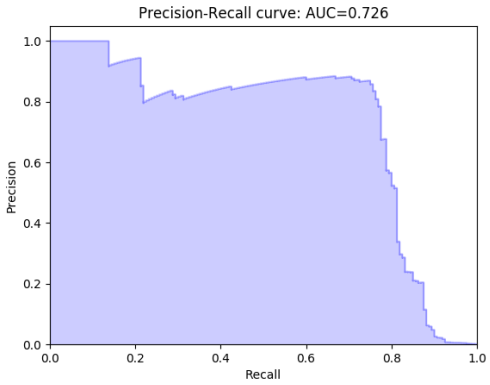

Figure 4.14: PR curve-LR-SMOTE

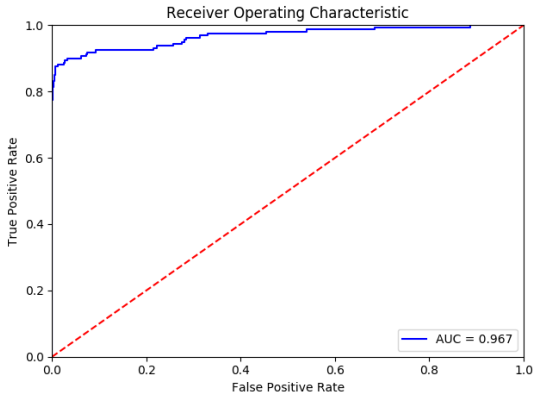

Figure 4.15: ROC curve-LR-SMOTE

SMOTE & Tomek links removal

Best C-Parameter: 0.01

Remarks

Table 4.6 shows the evaluation metrics of the logistic regression when a combination of Synthetic Minority Over-sampling Technique (SMOTE) and tomek links removal is used. Figure 4.16, figure 4.17, and figure 4.18 show the confusion matrix, the PR curve, and the ROC curve respectively. Similarly, using the combination of SMOTE and Tomek links removal also didn't improve the model performance at all, as the f1 score was just 0.15.

Class	Precision	Recall	F1 score	Support
0	0.9998	0.9803	0.9899	85283
1	0.0793	0.9062	0.1458	160
avg/total	0.9981	0.9801	0.9884	85443

Table 4.6: Evaluation metrics-LR-SMOTE % tomek link removal

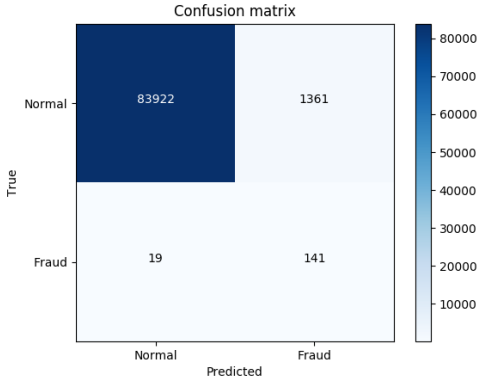

Figure 4.16: Confusion matrix-LR-SMOTE & tomek links removal

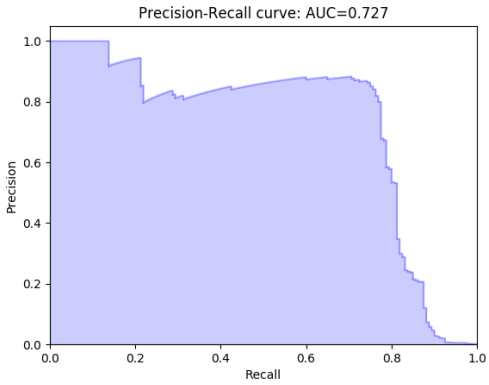

Figure 4.17: PR curve-LR-SMOTE & tomek links removal

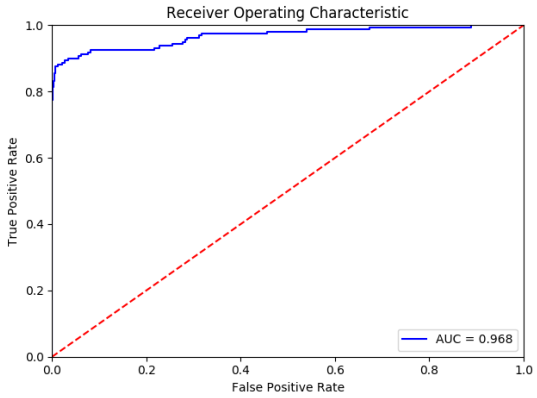

Figure 4.18: ROC curve-LR-SMOTE & tomek links removal

4.2 Random forest (RF)

No resampling- Best hyperparameters: total trees: 400, max features: auto

Remarks

Table 4.7 shows the evaluation metrics of the random forest when none of the resampling methods are used. Figure 4.19, figure 4.20, and figure 4.21 show the confusion matrix, the PR curve, and the ROC curve respectively. The random forest performed very well in classifying both the negative class and positive class with an overall f1 score of 0.87 even without resampling. This clearly shows the power of ensemble technique when used on the imbalanced dataset. However, still, we would like to have recall score a little higher than 0.7937.

Class	Precision	Recall	F1 score	Support
0	0.9996	0.9999	0.9998	85283
1	0.9621	0.7937	0.8699	160
avg/total	0.9995	0.9996	0.9995	85443

Table 4.7: Evaluation metrics-RF-no resampling

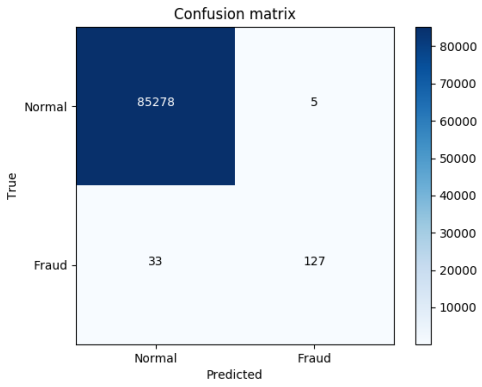

Figure 4.19: Confusion matrix-RF-no resampling

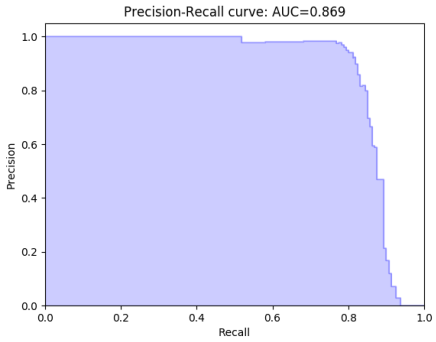

Figure 4.20: PR curve-RF-no resampling

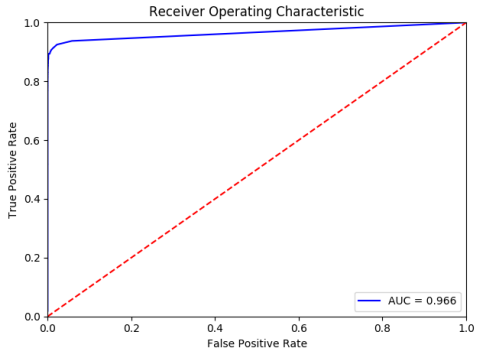

Figure 4.21: ROC curve-RF-no resampling

Random undersampling

Best hyperparameters: total trees: 400, max features: auto

Remarks

Table 4.8 shows the evaluation metrics of the random forest when random undersampling is used. Figure 4.22, figure 4.23, and figure 4.24 show the confusion matrix, the PR curve, and the ROC curve respectively. The random undersampling approach, when used with logistic regression, performed worst in terms of precision, but the result was the exact opposite in terms of recall. It happened to be the same case in case of the random forest as well that concludes it as a bad classifier.

Class	Precision	Recall	F1 score	Support
0	0.9998	0.9637	0.9814	85283
1	0.0453	0.9187	0.0864	160
avg/total	0.9981	0.9636	0.9798	85443

Table 4.8: Evaluation metrics-RF-random undersampling

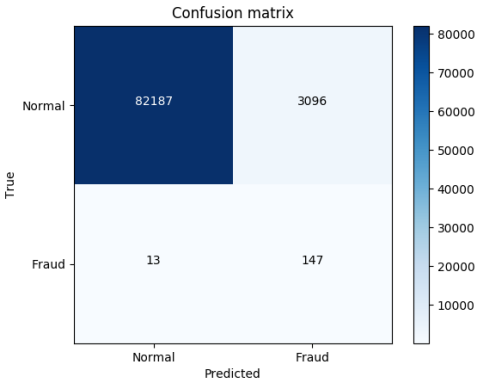

Figure 4.22: Confusion matrix-RF-random undersampling

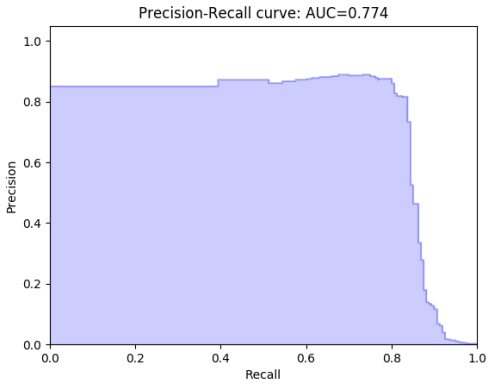

Figure 4.23: PR curve-RF-random undersampling

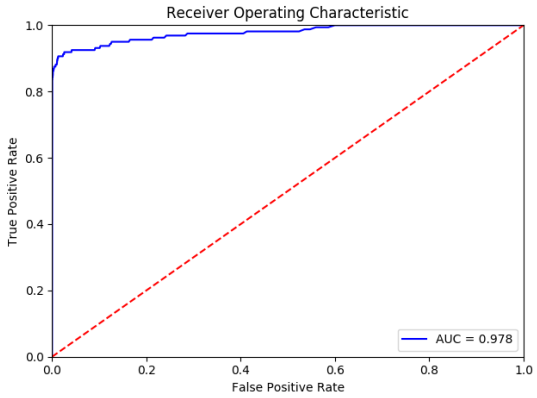

Figure 4.24: ROC curve-RF-random undersampling

Tomek links removal

Best hyperparameters: Number of trees: 600, max features: auto

Remarks

Table 4.9 shows the evaluation metrics of the random forest when tomek links removal method is used. Figure 4.25, figure 4.26, and figure 4.27 show the confusion matrix, the PR curve, and the ROC curve respectively. The performance of random forest when used with tomek links removal approach is very similar to the performance when no resampling was performed. The precision and recall were considerably high, but still, we want the recall score to be a bit higher than that. The area under the curve score for both the precision-recall curve and receiver operating characteristic was also considerably high.

Class	Precision	Recall	F1 score	Support
0	0.9996	0.9999	0.9997	85283
1	0.9185	0.7750	0.8407	160
avg/total	0.9994	0.9994	0.9994	85443

Table 4.9: Evaluation metrics-RF-tomek links removal

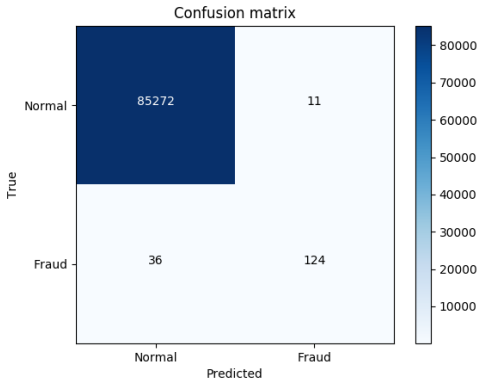

Figure 4.25: Confusion matrix-RF-tomek links removal

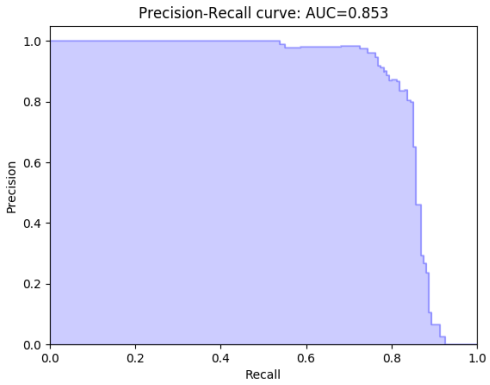

Figure 4.26: PR curve-RF-tomek links removal

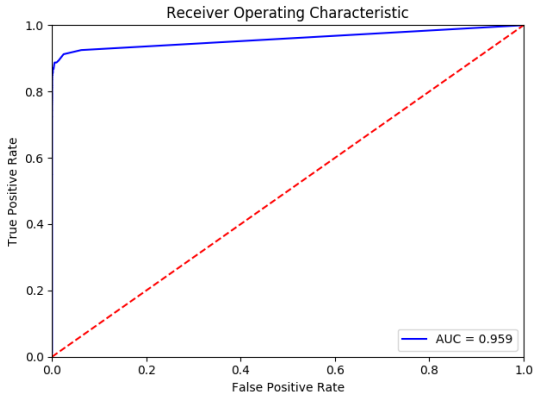

Figure 4.27: ROC curve-RF-tomek links removal

Random oversampling

Best hyperparameters: Number of trees: 800, max features: auto

Remarks

Table 4.10 shows the evaluation metrics of the random forest when random oversampling is used. Figure 4.28, figure 4.29, and figure 4.30 show the confusion matrix, the PR curve, and the ROC curve respectively. Similar to tomek links removal, the precision score was high and the recall score was decent. But we would like to have recall score a bit higher than that.

Class	Precision	Recall	F1 score	Support
0	0.9996	1.0000	0.9998	85283
1	0.9685	0.7688	0.8571	160
avg/total	0.9995	0.9995	0.9995	85443

Table 4.10: Evaluation metrics-RF-random oversampling

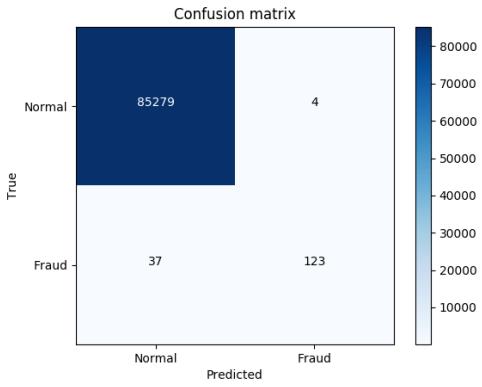

Figure 4.28: Confusion matrix-RF-random oversampling

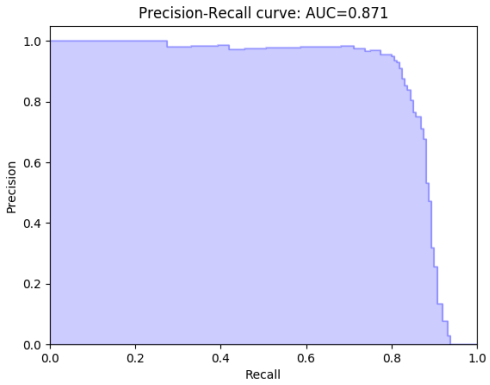

Figure 4.29: PR curve-RF-random oversampling

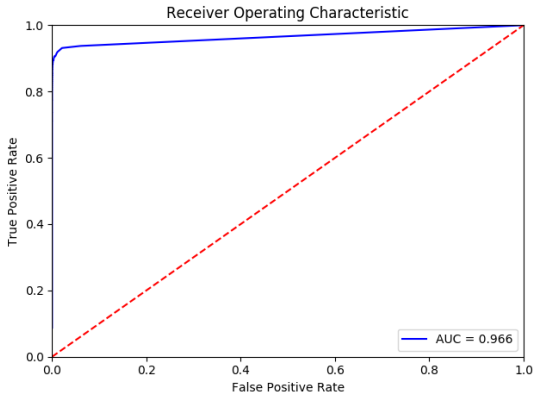

Figure 4.30: ROC curve-RF-random oversampling

SMOTE

Best hyperparameters: Total trees=600, max features=auto

Remarks

Table 4.11 shows the evaluation metrics of the random forest when SMOTE is used. Figure 4.31, figure 4.32, and figure 4.33 show the confusion matrix, the PR curve, and the ROC curve respectively. This time we got a higher recall score, which is great, however, the precision score decreased by quite a higher amount. We would want to have a balance between recall and precision.

Class	Precision	Recall	F1 score	Support
0	0.9998	0.9992	0.9995	85283
1	0.6814	0.8688	0.7637	160
avg/total	0.9992	0.9990	0.9991	85443

Table 4.11: Evaluation metrics-RF-SMOTE

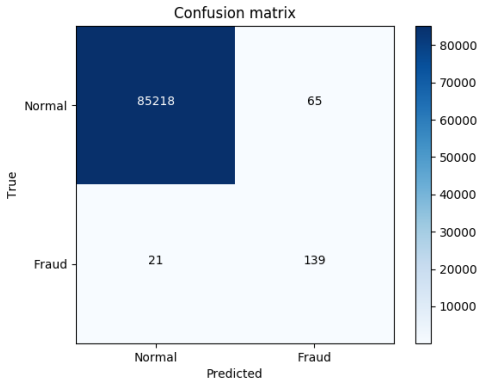

Figure 4.31: Confusion matrix-RF-SMOTE

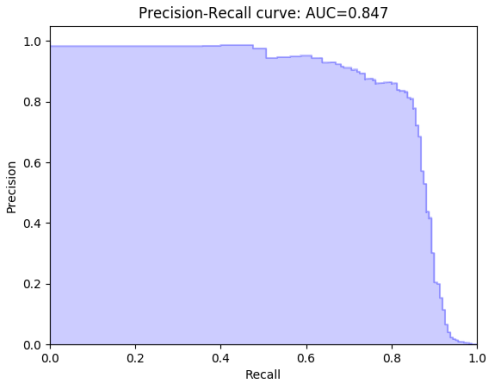

Figure 4.32: PR curve-RF-SMOTE

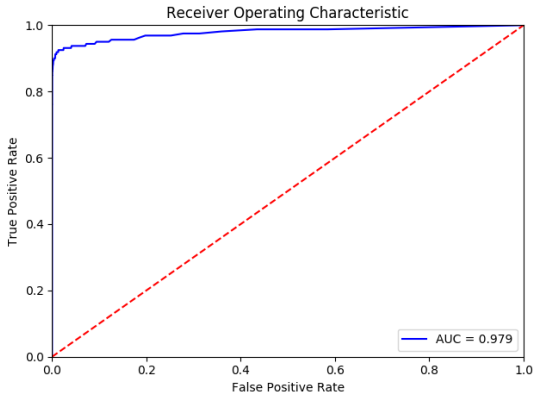

Figure 4.33: ROC curve-RF-SMOTE

SMOTE & Tomek links removal

Best hyperparameters: Total trees = 400, Maximum features = auto

Remarks

Table 4.12 shows the evaluation metrics of the random forest when a combination of SMOTE and tomek links removal is used. Figure 4.34, figure 4.35, and figure 4.36 show the confusion matrix, the PR curve, and the ROC curve respectively. The random forest, when used with a combination of SMOTE and tomek links removal, performed very well considering the precision score of 0.84 and recall score of 0.84, which can also be seen from the higher PR AUC and ROC AUC scores. The random forest performed very well when used with any resampling technique as compared to logistic regression. This shows that ensemble technique like random forest is very powerful when used to classify imbalanced data.

Class	Precision	Recall	F1 score	Support
0	0.9997	0.9997	0.9997	85283
1	0.8438	0.8438	0.8438	160
avg/total	0.9994	0.9994	0.9994	85443

Table 4.12: Evaluation metrics-RF-SMOTE & tomek link removal

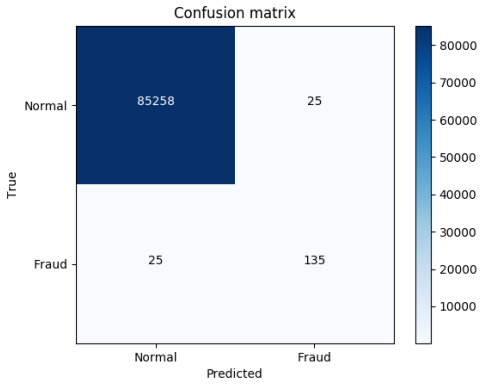

Figure 4.34: Confusion matrix-RF-SMOTE & tomek links removal

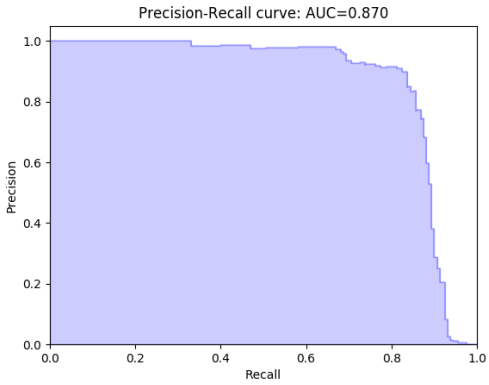

Figure 4.35: PR curve-RF-SMOTE & tomek links removal

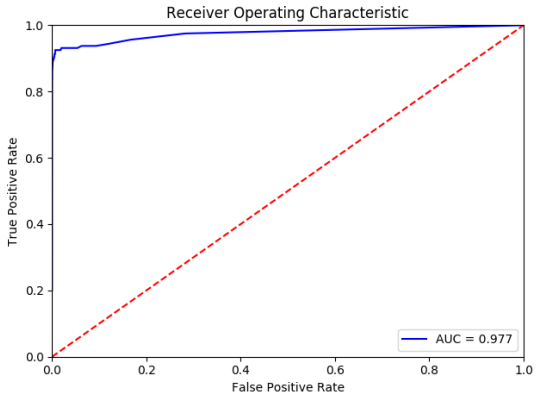

Figure 4.36: ROC curve-RF-SMOTE & tomek links removal

4.3 XGBoost (XGB)

No resampling- Best hyperparameters: Learning rate:0.1, total estimators:1000, maximum depth:6, minimum child weight:6, Sub-sample:0.85, colsample bytree:0.75, alpha:1e-5, gamma:0.0

Remarks

Table 4.13 shows the evaluation metrics of XGBoost when none of the resampling methods are used. Figure 4.37, figure 4.38, and figure 4.39 show the confusion matrix, the PR curve, and the ROC curve respectively. XGBoost performed very well in classifying the positive class in terms of precision even without resampling. The recall score is not so high as compared to random forest and it can definitely be improved.

Class	Precision	Recall	F1 score	Support
0	0.9995	0.9999	0.9997	85296
1	0.9291	0.7375	0.8223	147
avg/total	0.9994	0.9994	0.9994	85443

Table 4.13: Evaluation metrics-XGB-no resampling

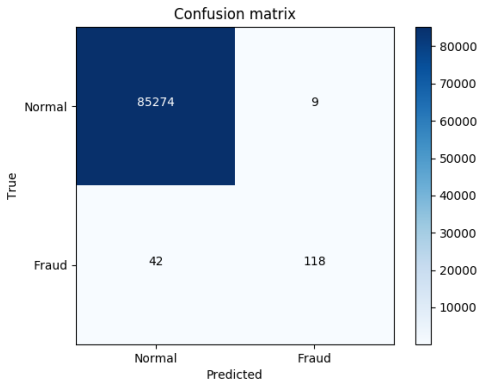

Figure 4.37: Confusion matrix-XGB-no resampling

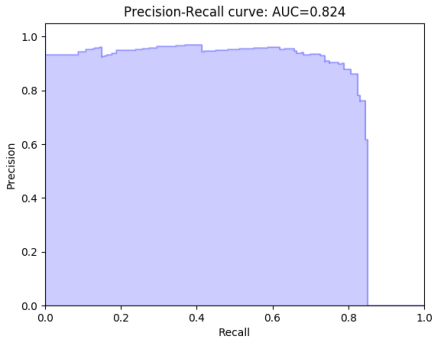

Figure 4.38: PR curve-XGB-no resampling

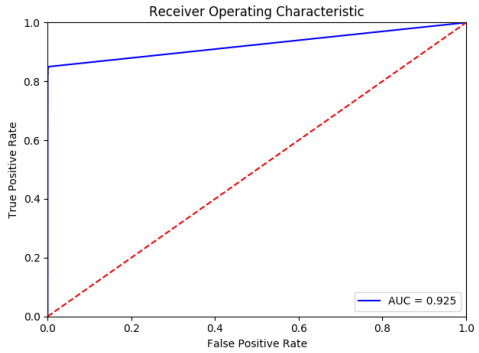

Figure 4.39: ROC curve-XGB-no resampling

Random undersampling

Best hyperparameters: Learning rate: 0.1, number of estimators:50, maximum depth:5, minimum child weight:4, subsample:0.85, colsample bytree:0.8, alpha:1, gamma:0.1

Remarks

Table 4.14 shows the evaluation metrics of XGBoost when the random undersampling method is used. Figure 4.40, figure 4.41, and figure 4.42 show the confusion matrix, the PR curve, and the ROC curve respectively. XGBoost, when used with random undersampling performed worst in terms of precision even though the recall score is very good. As we have seen so far, all the classifiers have shown similar results when random undersampling is used to tackle the class imbalance problem.

Class	Precision	Recall	F1 score	Support
0	0.9998	0.9594	0.9792	85283
1	0.0397	0.8938	0.0760	160
avg/total	0.9980	0.9593	0.9775	85443

Table 4.14: Evaluation metrics-XGB-random undersampling

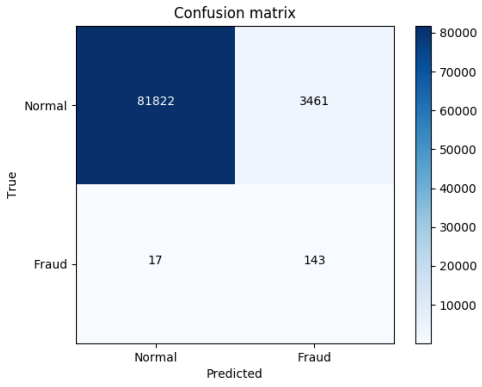

Figure 4.40: Confusion matrix-XGB-random undersampling

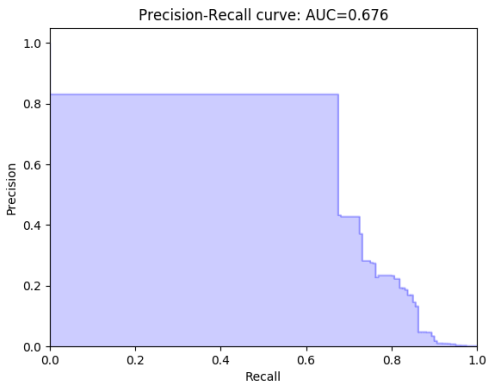

Figure 4.41: PR curve-XGB-random undersampling

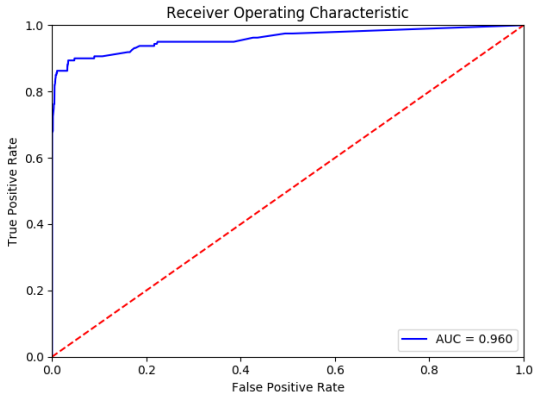

Figure 4.42: ROC curve-XGB-random undersampling

Tomek links removal

Best hyperparameters:Learning rate: 0.1, number of estimators:115, maximum depth:4, minimum child weight:5, subsample:0.65, colsample bytree:0.6, alpha: 1e-5, gamma: 0.2

Remarks

Table 4.15 shows the evaluation metrics of XGBoost when tomek links removal method is used. Figure 4.43, figure 4.44, and figure 4.45 show the confusion matrix, the PR curve, and the ROC curve respectively. There is a huge increase in precision when XGBoost is used with tomek links removal approach, as compared to random undersampling technique. However, the recall has decreased by quite a number.

Class	Precision	Recall	F1 score	Support
0	0.9995	0.9998	0.9996	85283
1	0.8702	0.7125	0.7835	160
avg/total	0.9992	0.9993	0.9992	85443

Table 4.15: Evaluation metrics-XGB-tomek links removal

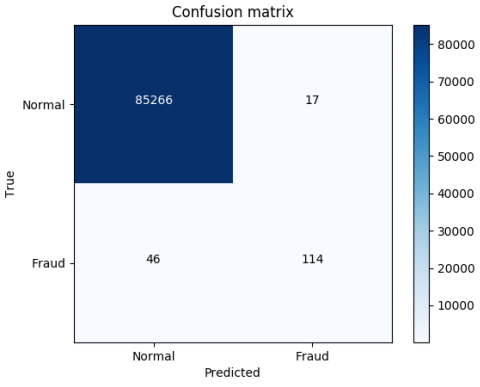

Figure 4.43: Confusion matrix-XGB-tomek links removal

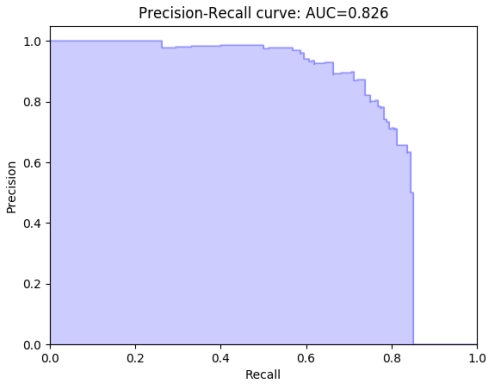

Figure 4.44: PR curve-XGB-tomek links removal

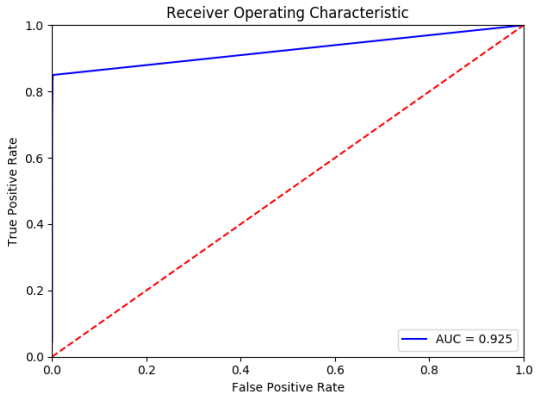

Figure 4.45: ROC curve-XGB-tomek links removal

Random oversampling

Best hyperparameters: Learning rate: 0.1, number of estimators: 572, maximum depth: 5, minimum child weight: 6, Subsample: 0.75, colsample bytree: 0.65, alpha:1e-5, gamma: 0.1

Remarks

Table 4.16 shows the evaluation metrics of XGBoost when the random oversampling method is used. Figure 4.46, figure 4.47, and figure 4.48 show the confusion matrix, the PR curve, and the ROC curve respectively. XGBoost with random oversampling performed quite well in terms of both the precision and recall scores.

Class	Precision	Recall	F1 score	Support
0	0.9997	0.9996	0.9997	85283
1	0.8024	0.8375	0.8196	160
avg/total	0.9993	0.9993	0.9993	85443

Table 4.16: Evaluation metrics-XGB-random oversampling

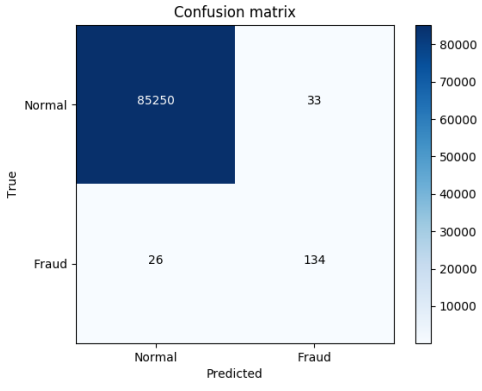

Figure 4.46: Confusion matrix-XGB-random oversampling

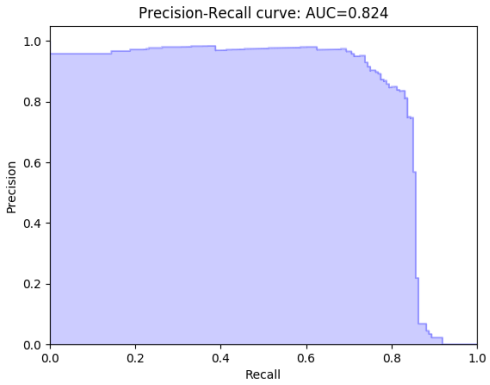

Figure 4.47: PR curve-XGB-random oversampling

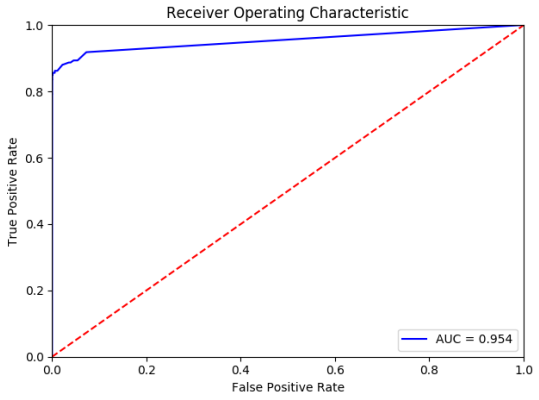

Figure 4.48: ROC curve-XGB-random oversampling

SMOTE

Best hyperparameters: Learning rate: 0.1, total estimators:870, maximum depth:10, minimum child weight:6, subsample:0.7, colsample bytree:0.95, alpha:1e-5, gamma:0.1

Remarks

Table 4.17 shows the evaluation metrics of XGBoost when SMOTE(Synthetic Minority Oversampling Technique) is used. Figure 4.49, figure 4.50, and figure 4.51 show the confusion matrix, the PR curve, and the ROC curve respectively. There was a high decrease in precision while using SMOTE with XGBoost. But the recall score is considerably high.

Class	Precision	Recall	F1 score	Support
0	0.9998	0.9985	0.9992	85283
1	0.5303	0.8750	0.6604	160
avg/total	0.9989	0.9983	0.9985	85443

Table 4.17: Evaluation metrics-XGB-SMOTE

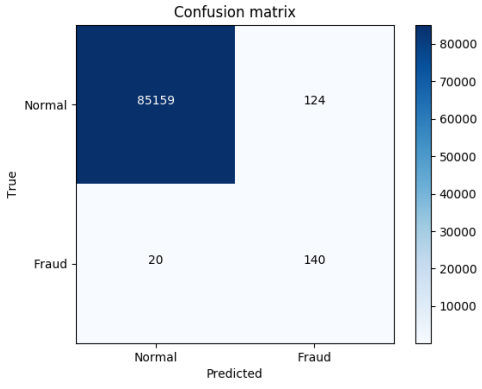

Figure 4.49: Confusion matrix-XGB-SMOTE

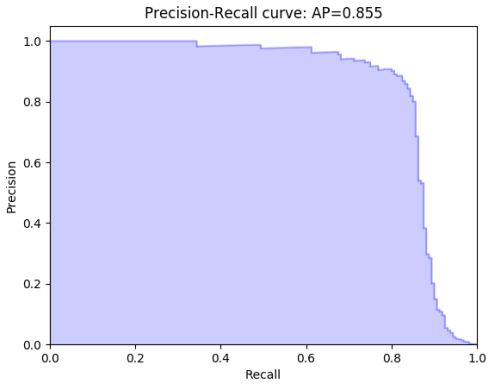

Figure 4.50: PR curve-XGB-SMOTE

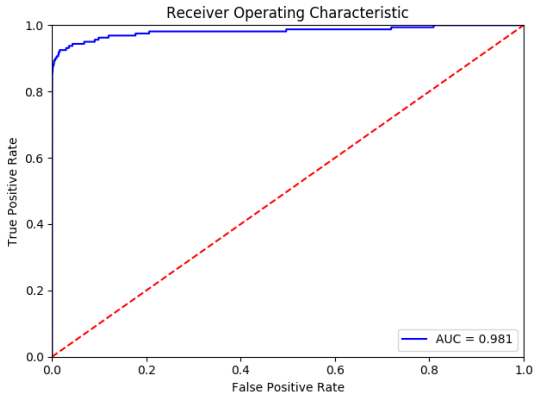

Figure 4.51: ROC curve-XGB-SMOTE

SMOTE & Tomek links removal

Best hyperparameters: Learning rate: 0.1, total estimators:600, maximum depth:7, minimum child weight:6, subsample:0.7, colsample bytree:0.85, alpha: 1e-5, gamma:0.1

Remarks

Table 4.18 shows the evaluation metrics of XGBoost when a combination of SMOTE and tomek links removal is used. Figure 4.52, figure 4.53, and figure 4.54 show the confusion matrix, the PR curve, and the ROC curve respectively. XGBoost with the combination of SMOTE and tomek links performed pretty well in terms of recall. However, the precision score is very low.

Class	Precision	Recall	F1 score	Support
0	0.9998	0.9934	0.9966	85283
1	0.2023	0.8875	0.3295	160
avg/total	0.9983	0.9932	0.9954	85443

Table 4.18: Evaluation metrics-XGB-SMOTE & tomek link removal

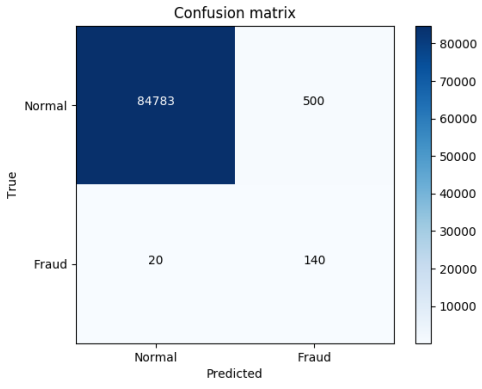

Figure 4.52: Confusion matrix-XGB-SMOTE & tomek links removal

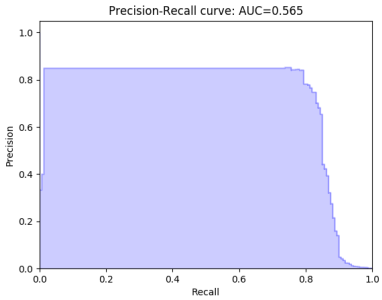

Figure 4.53: PR curve-XGB-SMOTE & tomek links removal

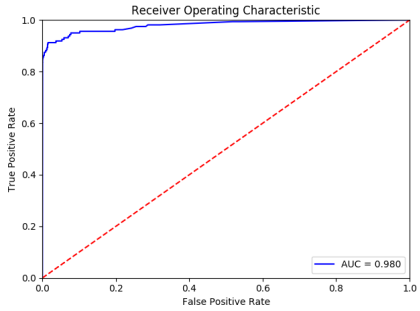

Figure 4.54: ROC curve-XGB-SMOTE & tomek links removal

4.4 Result summary

Classifier	Resampling method	Precision	Recall	F1 Score	PR	ROC
Logistic regression	No resampling	0.68	0.53	0.59	0.454	0.852
	Random undersampling	0.09	0.91	0.16	0.729	0.973
	Tomek links removal	0.67	0.53	0.59	0.455	0.854
	Random oversampling	0.07	0.90	0.13	0.726	0.971
	SMOTE	0.09	0.88	0.17	0.726	0.967
	SMOTE + Tomek links removal	0.08	0.91	0.15	0.727	0.973
Random forest	No resampling	0.96	0.79	0.87	0.869	0.966
	Random undersampling	0.05	0.92	0.09	0.774	0.978
	Tomek links removal	0.92	0.78	0.84	0.853	0.959
	Random oversampling	0.97	0.77	0.86	0.871	0.966
	SMOTE	0.68	0.87	0.76	0.847	0.979
	SMOTE + Tomek links removal	0.84	0.84	0.84	0.870	0.977
XGBoost	No resampling	0.93	0.74	0.82	0.824	0.925
	Random undersampling	0.04	0.89	0.08	0.676	0.960
	Tomek links removal	0.87	0.71	0.78	0.826	0.925
	Random oversampling	0.80	0.84	0.82	0.824	0.954
	SMOTE	0.53	0.88	0.66	0.855	0.981
	SMOTE + Tomek links removal	0.22	0.88	0.35	0.565	0.980

Table 4.19: Result summary

The above table clearly shows that random forest and XGBoost even without resampling performed better than the logistic regression model considering their overall f1 score. This shows the power of ensemble techniques that can give higher performance even in the presence of the class imbalance problem. Every model, when used with random undersampling, gave a good recall score but failed miserably in terms of precision. Comparing with the ensemble models without resampling, the precision and recall score did not improve in the case where tomek links removal was used. Both random forest and xgboost performed pretty well in terms of f1 score when random oversampling was used. SMOTE improved the recall scores of both random forest and xgboost but the precision scores decreased quite a bit. Finally, when the hybrid combination of SMOTE and tomek links removal was applied with random forest, it gave a balanced precision & recall scores of 0.84 and area under the PR curve of 0.870. Whereas when the same resampling technique was used, xgboost failed to improve the precision score. Also, we can see that there was very little difference in the area under the ROC curves between the models.

Chapter 5

Conclusion and Future Works

In this thesis, we applied machine learning techniques to predict whether a credit card transaction is fraudulent or not. For this, we collected a publicly available dataset provided by the machine learning group of ULB (Universit Libre de Bruxelles), which contains the record of credit card transactions made by European cardholders and occurred in two days in September 2013. It contains 284,807 transactions out of which only 492 are fraudulent. The dataset is highly unbalanced as the positive class accounts for only 0.172% of the total transactions.

When providing input data of a highly unbalanced class distribution to the predictive model, the model tends to be biased towards the majority samples. As a result, it tends to misrepresent a fraudulent transaction as a genuine transaction. To tackle this problem, we implemented a data-level approach which includes various resampling techniques namely, random undersampling, tomek links removal, random oversampling, Synthetic Minority Over-sampling Technique (SMOTE) and a hybrid resampling approach of SMOTE and tomek links removal. In addition, we implemented the algorithmic approaches such as bagging and boosting to tackle the class imbalance problem. For this, we selected random forest model as a bagging method and XGBoost as a boosting method. Besides these models, we chose logistic regression model to compare with other models. Then, we analyzed all three models with and without using resampling techniques. The comparison results revealed that the random forest in combination with a hybrid resampling approach of SMOTE and tomek links removal performed better than other models.

For future work, a cost-sensitive learning approach can be implemented by considering the misclassification costs. The cost for misclassifying a fraudulent class as a legitimate class (False Negative), which corresponds to the fraud amount (can be from few to thousands of dollars) is much higher than the cost for misclassifying a legitimate class as a fraudulent class (False Positive), which

corresponds to the cost related to analyzing the transaction and contacting the cardholder. So, this type of learning deals with classifying an example into a class that has the minimum expected cost.

Credit card fraud is related to the non-stationary nature of transaction distributions in which the fraudsters usually always comes with a new way to attempt the fraudulent activities. Therefore, it becomes essential to consider these changing behavior as well while developing a predictive model. Hence, a detailed study on dealing with non-stationary nature in credit card fraud detection can be performed. However, this study requires a huge amount of data.

Bibliography

[AAO17] John O. Awoyemi, Adebayo Olusola Adetunmbi, and Samuel Adebayo Oluwadare. Credit card fraud detection using machine learning techniques: A comparative analysis. *2017 International Conference on Computing Networking and Informatics (ICCNI)*, pages 1–9, 2017.

[AFR97] Emin Aleskerov, Bernd Freisleben, and R. Bharat Rao. Cardwatch: a neural network based database mining system for credit card fraud detection. In *CIFEr*, 1997.

[BCHK02] Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.*, 16:321–357, 2002.

[BK18] Galina Baader and Helmut Krcmar. Reducing false positives in fraud detection: Combining the red flag approach with process mining. *International Journal of Accounting Information Systems*, 2018.

[Blo18] Guest Blog. How to handle imbalanced classification problems in machine learning?, Apr 2018.

[BPM04] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations*, 6:20–29, 2004.

[Bra97] Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30:1145–1159, 1997.

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *KDD*, 2016.

[CPB+18] Fabrizio Carcillo, Andrea Dal Pozzolo, Yann-Aël Le Borgne, Olivier Caelen, Yannis Mazzer, and Gianluca Bontempi. Scarff: a scalable framework for streaming credit card fraud detection with spark. *Information Fusion*, 41:182–194, 2018.

[Die00] Thomas G. Dietterich. Multiple classifier systems. In *Lecture Notes in Computer Science*, 2000.

[Dom99] Pedro M. Domingos. Metacost: A general method for making classifiers cost-sensitive. In *KDD*, 1999.

[Gan18] Rohith Gandhi. Introduction to machine learning algorithms: Logistic regression, May 2018.

[HAP89] Robert C. Holte, L. Acker, and B. Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 813–818, Detroit, MI, 1989.

[JAH+08] Piotr Juszczak, Niall M. Adams, David J. Hand, Christopher Whitrow, and David John Weston. Off-the-peg and bespoke classifiers for fraud detection. *Computational Statistics Data Analysis*, 52:4521–4532, 2008.

[JS02] Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, pages 429–449, 2002.

[LBCS16] Bertrand Lebichot, Fabian Braun, Olivier Caelen, and Marco Saerens. A graph-based, semi-supervised, credit card fraud detection system. In *COMPLEX NETWORKS*, 2016.

[Lib17] Neil Liberman. Decision trees and random forests towards data science, Jan 2017.

[MTVM93] Sam Maes, Karl Tuyls, Bram Vanschoenwinkel, and Bernard Manderick. Credit card fraud detection using bayesian and neural networks. In *In: Maciunas RJ, editor. Interactive image-guided neurosurgery. American Association Neurological Surgeons*, pages 261–270, 1993.

[nila] Card and mobile payment industry news | the nilson report newsletter archive.

[PBC+15] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection and concept-drift adaptation with delayed supervised information. *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2015.

[PCB+14] Andrea Dal Pozzolo, Olivier Caelen, Yann-Aël Le Borgne, Serge Waterschoot, and Gianluca Bontempi. Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Syst. Appl.*, 41:4915–4928, 2014.

[PCJB15] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166, 2015.

[Poz15] Andrea Dal Pozzolo. *Adaptive Machine Learning for Credit Card Fraud Detection.* PhD thesis, 2015.

[Sea17] SeattleDataGuy. Boosting and bagging: How to develop a robust machine learning algorithm, Nov 2017.

[SKSM08] Abhinav Srivastava, Amlan Kundu, Shamik Sural, and Arun K. Majumdar. Credit card fraud detection using hidden markov model. *2011 World Congress on Information and Communication Technologies*, pages 1062–1066, 2008.

[sky] A beginner's guide to deep reinforcement learning.

[sta] U.s. payment card fraud losses by type 2018 | statistic.

[WA00] Richard Wheeler and J. Stuart Aitken. Multiple algorithms for fraud detection. *Knowl.-Based Syst.*, 13:93–99, 2000.

Curriculum Vitae

Graduate College
University of Nevada, Las Vegas

Ronish Shakya
ronishshk@gmail.com

Degrees:

Bachelor Degree in Computer Engineering 2013
Kantipur Engineering College, Tribhuwan University, Nepal

Thesis Title: Application of Machine Algorithm Techniques in Credit Card Fraud Detection

Thesis Examination Committee:

Chairperson, Dr. Fatma Nasoz, Ph.D.
Committee Member, Dr. Justin Zhan, Ph.D.
Committee Member, Dr. Laxmi Gewali, Ph.D.
Graduate Faculty Representative, Dr. Mehmet Erdem, Ph.D.