MASTER THESIS

Rohit Khanduri

Fraud detection in Credit Cards using Machine Learning technologies

Mittweida, 2020

Fakultät Angewandte Computer- und Biowissenschaften

MASTER THESIS

Fraud detection in Credit Cards using Machine Learning technologies

Author:
Rohit Khanduri

Course of Studies:
Applied Mathematics for Network and Data Sciences

Seminar Group:
Fakultät Angewandte Computer- und Biowissenschaften

First examiner:
Prof. Dr. rer. nat. habil. Thomas Villmann

Second examiner:
Dipl.-Math. Lars Nöbel

Submission:
Mittweida, 15.11.2020

Defence/Evaluation:
Mittweida, 2020

Fakultät Angewandte Computer- und Biowissenschaften

MASTER THESIS

Bibliographic description:
Khanduri, Rohit:
Fraud detection in Credit Cards using Machine Learning technologies -
2020 - Mittweida, Hochschule Mittweida, Fakultät Angewandte Computer- und
Biowissenschaften, Master Thesis, 2020

Contents

List of Figures **3**

List of Tables **4**

1 Abstract **5**

2 Acknowledgements **5**

3 Introduction **6**

4 Challenges in Fraud Detection **6**

5 Objective **6**

6 Outline **7**

7 Related work and Theory of Machine Learning **7**
7.1 Related Work . . . 7
7.2 Machine Learning . . . 8
7.2.1 Introduction . . . 9
7.2.2 Main challenges of Machine Learning . . . 10
7.2.3 Classification . . . 10
7.2.4 Feature Engineering . . . 10
7.2.5 Overfitting the Training Data . . . 11
7.2.6 Underfitting . . . 11
7.3 Predictive modelling . . . 12
7.3.1 Classification Predictive Modelling . . . 12
7.3.2 Regression Predictive Modelling . . . 12
7.4 Class Imbalance Problem . . . 13
7.5 Solutions to the Class imbalance problem . . . 13
7.6 Resampling . . . 13
7.7 Random Undersampling . . . 13
7.8 Tomek Link removals . . . 14
7.9 Random Oversampling . . . 14
7.10 Synthetic Minority Oversampling Technique (SMOTE) . . . 14
7.11 SMOTE and Tomek Link removal together . . . 14
7.12 Ensemble Learning . . . 14
7.12.1 Voting Classifiers . . . 15
7.12.2 Bagging . . . 16
7.12.3 Out-of-bag Error Estimation . . . 17
7.12.4 Boosting . . . 17
7.12.5 Boosting as a gradient descent . . . 18
7.12.6 Margin for classification . . . 18
7.13 Selected models . . . 18
7.13.1 Logistic Regression . . . 19
7.13.2 Decision Trees . . . 21

7.13.3 Random Forests . . . 22
7.13.4 XGBoost . . . 23
7.14 Evaluation Metrics . . . 25
7.14.1 Confusion Matrix . . . 25
7.14.2 Precision . . . 26
7.14.3 Recall . . . 26
7.14.4 Precision and Recall - The F1 Score . . . 26
7.14.5 Precision/Recall Trade-off . . . 27
7.14.6 Area under the Precision Recall Curve . . . 27
7.14.7 The ROC Curve . . . 27

8 Data and the method **28**
8.1 Data description . . . 28
8.2 Standardization of the data . . . 32
8.3 Splitting the data . . . 32
8.4 Data resampling . . . 32
8.5 10 fold cross validation . . . 32
8.6 Training and Testing . . . 33
8.7 Performance evaluation . . . 33

9 Results **33**
9.1 Logistic Regression . . . 33
9.1.1 No resampling . . . 33
9.1.2 Random Undersampling . . . 34
9.1.3 Tomek Links Removal . . . 36
9.1.4 Random Oversampling . . . 40
9.1.5 SMOTE . . . 40
9.1.6 SMOTE & Tomek Links removal . . . 42
9.2 Random Forest . . . 46
9.2.1 No resampling . . . 46
9.2.2 Random Undersampling . . . 48
9.2.3 Tomek Links Removal . . . 48
9.2.4 Random Oversampling . . . 50
9.2.5 SMOTE . . . 50
9.2.6 SMOTE & Tomek Links removal . . . 54
9.3 XGBoost . . . 58
9.3.1 No resampling . . . 58
9.3.2 Random Undersampling . . . 58
9.3.3 Tomek Links Removal . . . 60
9.3.4 Random Oversampling . . . 64
9.3.5 SMOTE . . . 64
9.3.6 SMOTE & Tomek Links removal . . . 66
9.4 Summary of results . . . 70

10 Conclusion **70**

Bibliography **72**

List of Figures

1 Precision Recall Curve . . . 28
2 Confusion Matrix . . . 29
3 A comparison of the number of Fraudulent transactions versus the number of normal transactions . . . 30
4 Heatmap for Correlation . . . 32
5 Logistic Regression Confusion Matrix - No Resampling . . . 34
6 Logistic Regression AUC-ROC - No Resampling . . . 35
7 Logistic Regression AUC-PR Curve - No Resampling . . . 35
8 Logistic Regression Confusion Matrix - Random Undersampling . . . 36
9 Logistic Regression AUC-ROC - Random Undersampling . . . 37
10 Logistic Regression AUC-PR Curve - Random Undersampling . . . 37
11 Logistic Regression Confusion Matrix - Tomek Links removal . . . 38
12 Logistic Regression AUC-ROC - Tomek Links removal . . . 38
13 Logistic Regression AUC-PR Curve - Tomek Links removal . . . 39
14 Logistic Regression Confusion Matrix - Random Oversampling . . . 40
15 Logistic Regression AUC-ROC - Random Oversampling . . . 41
16 Logistic Regression AUC-PR Curve - Random Oversampling . . . 41
17 Logistic Regression Confusion Matrix - SMOTE . . . 42
18 Logistic Regression AUC-ROC - SMOTE . . . 43
19 Logistic Regression AUC-PR Curve - SMOTE . . . 43
20 Logistic Regression Confusion Matrix - SMOTE and Tomek Links removal . . . 44
21 Logistic Regression AUC-ROC - SMOTE and Tomek Links removal . . . 44
22 Logistic Regression AUC-PR Curve - SMOTE and Tomek Links removal . . . 45
23 Random Forest Confusion Matrix - No Resampling . . . 46
24 Random Forest AUC-ROC - No Resampling . . . 47
25 Random Forest AUC-PR Curve - No Resampling . . . 47
26 Random Forest Confusion Matrix - Random Undersampling . . . 48
27 Random Forest AUC-ROC - Random Undersampling . . . 49
28 Random Forest AUC-PR Curve - Random Undersampling . . . 49
29 Random Forest Confusion Matrix - Tomek Links removal . . . 50
30 Random Forest AUC-ROC - Tomek Links removal . . . 51
31 Random Forest AUC-PR Curve - Tomek Links removal . . . 51
32 Random Forest Confusion Matrix - Random Oversampling . . . 52
33 Random Forest AUC-ROC - Random Oversampling . . . 52
34 Random Forest AUC-PR Curve - Random Oversampling . . . 53
35 Random Forest Confusion Matrix - SMOTE . . . 54
36 Random Forest AUC-ROC - SMOTE . . . 55
37 Random Forest AUC-PR Curve - SMOTE . . . 55
38 Random Forest Confusion Matrix - SMOTE and Tomek Links removal . . . 56
39 Random Forest AUC-ROC - SMOTE and Tomek Links removal . . . 56
40 Random Forest AUC-PR Curve - SMOTE and Tomek Links removal . . . 57
41 XGBoost Confusion Matrix - No Resampling . . . 58
42 XGBoost AUC-ROC - No Resampling . . . 59
43 XGBoost AUC-PR Curve - No Resampling . . . 59

44 XGBoost Confusion Matrix - Random Undersampling 60
45 XGBoost AUC-ROC - Random Undersampling 61
46 XGBoost AUC-PR Curve - Random Undersampling 61
47 XGBoost Confusion Matrix - Tomek Links removal 62
48 XGBoost AUC-ROC - Tomek Links removal 62
49 XGBoost AUC-PR Curve - Tomek Links removal 63
50 XGBoost Confusion Matrix - Random Oversampling 64
51 XGBoost AUC-ROC - Random Oversampling 65
52 XGBoost AUC-PR Curve - Random Oversampling 65
53 XGBoost Confusion Matrix - SMOTE 66
54 XGBoost AUC-ROC - SMOTE 67
55 XGBoost AUC-PR Curve - SMOTE 67
56 XGBoost Confusion Matrix - SMOTE and Tomek Links removal 68
57 XGBoost AUC-ROC - SMOTE and Tomek Links removal 68
58 XGBoost AUC-PR Curve - SMOTE and Tomek Links removal 69

List of Tables

1 Description of the dataset 29
2 Logistic Regression results for various threshold values for No Resampling 34
3 Logistic Regression results for various threshold values for Random Undersampling . 36
4 Logistic Regression results for various threshold values for Tomek Links removal . . 39
5 Logistic Regression results for various threshold values for Random Oversampling . . 40
6 Logistic Regression results for various threshold values for SMOTE 42
7 Logistic Regression results for various threshold values for SMOTE and Tomek Links removal 45
8 Random Forest results for various threshold values for No Resampling 46
9 Random Forest results for various threshold values for Random Undersampling . . . 48
10 Random Forest results for various threshold values for Tomek Links removal 50
11 Random Forest results for various threshold values for Random Oversampling 53
12 Random Forest results for various threshold values for SMOTE 54
13 Random Forest results for various threshold values for SMOTE and Tomek Links removal 57
14 XGBoost results for various threshold values for No Resampling 58
15 XGBoost results for various threshold values for Random Undersampling 60
16 XGBoost results for various threshold values for Tomek Links removal 63
17 XGBoost results for various threshold values for Random Oversampling 64
18 XGBoost results for various threshold values for SMOTE 66
19 XGBoost results for various threshold values for SMOTE and Tomek Links removal 69
20 Summary of the results 70

1 Abstract

Financial fraud for banks can be a reason for huge monetary losses. Studies have shown that, if not mitigated, financial fraud can lead to bankruptcy for big financial institutions and even insolvency for individuals. Credit card fraud is a type of financial fraud that is ever growing. In the future, these numbers are expected to increase exponentially and that's why a lot of researchers are focusing on machine learning techniques for detecting frauds. This task, however, is not a simple task. There are mainly two reasons

- varying behaviour in committing fraud
- high level of imbalance in the dataset (the majority of normal or genuine cases largely outnumbers the number of fraudulent cases)

A predictive model usually tends to be biased towards the majority of samples, in an unbalanced dataset, when this dataset is provided as an input to a predictive model.
In this Thesis this problem is tackled by implementing a data-level approach where different resampling methods such as undersampling, oversampling, and hybrid strategies along with bagging and boosting algorithmic approaches have been applied to a highly skewed dataset with 492 identified frauds out of 284,807 transactions.
Predictive modelling algorithms like Logistic Regression, Random Forest, and XGBoost have been implemented along with different resampling techniques to predict fraudulent transactions. The performance of the predictive models was evaluated based on Receiver Operating Characteristic-Area under the curve (AUC-ROC), Precision Recall Area under the Curve (AUC-PR), Precision, Recall, F1 score metrics.

2 Acknowledgements

I would like to express my sincere gratitude to my advisor, Anne Reichmuth, for her valuable and constructive suggestions and reviews during the planning and development of this research work. She consistently guided me in the right direction by giving me valuable tips and instructions to improve my work. I would also like to thank Mr. Lars Nöbel, Prof. Dr. Thomas Villmann and Prof. Dr. Marika Kaden for their continuous support and for being a part of my thesis committee.

ROHIT KHANDURI

Hochschule Mittweida, 2020

3 Introduction

Financial fraud is a serious white-collar crime which is often accompanied by strict punishment and monetary or non-monetary fines.
In the report 2018 True Cost of Fraud Study for the Retail Sector (LexisNexis, 2018), prepared and published by the LexisNexis Risk Solutions, estimates that fraud costed US eCommerce an average of 2.38 percent of revenue in 2018. These costs vary by sector - digital goods and physical goods. With the boom in the technological sector, there is an increased growth in digital payments leading to an extremely high rise in the number of digital financial fraud cases. According to the LexisNexis Report (LexisNexis, 2018) the losses due to credit card, debit and prepaid cards were $ 22.8 billion worldwide in 2018 which is a 4% increase from $ 16.31 billion in 2015.
The solutions to such fraud can be categorized as prevention and detection. The former involves preventing the fraud in the source itself and the latter is the action taken after the occurrence of the event.
The technologies like Address Verifcation System (AVS) and Card Verifcation System (CVM) are usually used to prevent fraud, as per the article by FIS Global (Global, 2019a, see). Fraud needs to be detected when it can not be prevented and necessary actions need to be taken. As per the article by FIS Global (Global, 2019b), fraud detection solutions include Predictive analytics, Experienced fraud analysts, Outlier models, Custom rule management, Global profiling, Mobile card controls, etc. to identify fraud.
This Thesis focuses on the automatic fraud detection system using machine learning technologies.

4 Challenges in Fraud Detection

Developing a Fraud detection system is a very complicated task. The developer needs to determine the learning strategy (supervised learning or unsupervised learning), the algorithm/s (Logistic regression, random forest, etc.), the features, and how to deal with the class imbalance problem (very few fraudulent cases compared to the normal cases) (Pozzolo and Bontempi, 2015). Unbalanced classes are not the only major concern for fraud detection systems but overlapping of classes due to limited transaction information is another problem for classification (Holte et al., 1989), and most machine learning algorithms do not perform well under these scenarios (Japkowicz and Stephen, 2002). A Fraud detection model predicts the fraudulent classes and alerts the investigating team about a probable fraud. This Investigating team will then perform a further investigation and provide feedback to the system to improve its performance. This is, however, a time taking process due to which only a few transactions can be validated and just a few feedbacks being provided to the predictive model, resulting generally in a less accurate model (Dal Pozzolo et al., 2015). 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 which poses as one of the major challenges in fraud detection related research (Dal Pozzolo et al., 2014).

5 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 whether a transaction belongs to the normal class or the fraudulent class using predictive models.
Different resampling techniques were implemented to tackle the class imbalance problem and a series of machine learning algorithms like logistic regression, random forest, and xgboost were tested to obtain the results.

6 Outline

- Chapter 1 is focused on fraud and its impact in the financial sector, an overview of a fraud detection process in general, the challenges faced by such systems, and finally an approach to develop a fraud detection system is proposed.
- Chapter 2 focuses on theoretical and background knowledge of Machine Learning for a better understanding of the proposed model.
- Chapter 3 focuses on the analysis methodology.
- Chapter 4 focuses on a comparison of the methods/models used for predicting frauds from the unbalanced dataset.
- Chapter 5 concludes the research and summarizes the results.

7 Related work and Theory of Machine Learning

7.1 Related Work

Credit card fraud is a major problem in the financial sector. Many researchers are actively working on to mitigate and find probable solutions for this problem. In this chapter we will discuss about some of the research work done in the past for this field.
Pozzolo et al. (Dal Pozzolo et al., 2014), in their study, have focused on basically two approaches towards detection of fraud:

- Static approach: where they train a detection model in a "seasonal manner" (once a month or once a year)
- Online approach: where they update the model immediately after the new transaction data arrives.

They stated that the online learning approach is a better approach towards detecting fraud as the behavior for any fraud changes from time to time. Pozzolo et al. (Dal Pozzolo et al., 2014) also proposed that the Average Precision (AP), Area Under Curve (AUC), and PrecisionRank are the best measures for detecting frauds.
In another study, Pozzolo et al. (Dal Pozzolo et al., 2015) concluded that random forest is the best approach in the fraud detection task.
A graph-based approach to develop a fraud detection system was proposed in a study by Lebichot et al. (Lebichot et al., 2017) where a collective inference algorithm was used to infer the

fraudulent behaviour in the dataset. This algorithm, known as Anomaly Prevention using Advanced Transaction Exploration (APATE), was significantly improved by Lebichot et al. (Lebichot et al., 2017) in their study to achieve their goal.
In their study, Awoyemi et al. (Awoyemi et al., 2017), analysed the data from K-nearest neighbor, logistic regression, and Naive Bayes applied on credit card transaction data, which was further resampled using Synthetic Minority Over-sampling Technique (SMOTE). The result showed that K-nearest neighbor performed better than the Logistic Regression and Naive Bayes measured in terms of recall, precision, balanced classification rate, specificity, and Mathews correlation coefficient.
Srivastava et al. (Srivastava et al., 2008) proposed a Hidden Markov Model (HMM) which analyses the spending habit of the customer in order to detect the fraudulent behavior.
In thier research, Wheeler and Aitken (Wheeler and Aitken, 2000) have investigated multiple algorithms to detect fraud, and showed that an adaptive approach filters and orders fraud cases and subsequently reduces the number of fraud investigations.
Carcillo et al. (Carcillo et al., 2017) proposed a unique solution in their study where they explored big data technology in fraud detection using big data tools such as Cassandra, Spark, and Kafka and developed a system called Scalable Realtime Fraud Finder (SCARFF). They established the fact that a highly scalable, fault-tolerant, and accurate system needs to be developed for fraud detection as such systems have to work on real-time transactions and because of the massive amount of transaction data, such a system has advantages and performs much better over conventional systems.
Domingos (Domingos, 1999), in his research, stated that the cost of all the misclassification errors is not constant and proposed a method to embed a procedure to minimizing cost in the classifier and named this cost-sensitive model, MetaCost. He found out a systematic decrease in the total cost with MetaCost compared to other non cost-sensitive classifiers.
Aleskerov et al. (Aleskerov et al., 1997), in their study, proposed a database mining system that uses neural networks for fraud detection.
In another research study, Baader and Krcmar (Baader and Krcmar, 2018) proposed an automated feature engineering approach to reduce the higher false positive rate which is usually observed in fraud prediction.
Even though there exists a lot of research in understanding the customer usage patterns, fraud processing time, reducing the number of fraud investigations, etc. there does not exists a lot of research focusing on the machine learning models best suitable for such a predictive analysis. This thesis, thus, aims to preform predictive analysis with a focus on various techniques such as sampling, ensemble and hybrid methods to tackle the problem of class imbalance.

7.2 Machine Learning

Machine Learning (ML) is a branch of Artificial Intelligence that lets machines (computers) learn tasks automatically without explicitly programming them (Valkov, 2019). In order to create a machine that can learn, one needs to follow the following steps:

- Define a computer understandable problem
- Choose a model (or a set of models) capable of solving the problem
- Provide the data to the model

- Assess the problem and improve the machine

7.2.1 Introduction

In machine learning, computers (models) are used to identify patterns in data automatically using statistical learning techniques. These techniques are, then, used to make highly accurate predictions about new and unseen data.
Dimensions in a data set are known as features which are sometimes also referred to as variables, or predictors. Identifying boundaries in data using mathematics is the core of statistical learning. These mathematical models should be trainable and creating such a model is also called training a model.
Machine Learning is used to find patterns in data which can later be identified in new and unseen data. These methods are used to identify patterns in data using statistical learning.
One common ML method is a decision tree. A decision tree uses if-then statements to identify patterns in data provided to it. These if-then conditional statements are called forks, which split the data into two branches based on some value and this value between the branches is called a split point. In an ideal scenario, at the best split, the results of each branch of a decision tree should be as homogeneous (or pure) as possible.
To add a new split point, the decision tree algorithm repeats the earlier process on the subsets of data. This repetition is called *recursion*, which is a frequently occurring term in Machine Learning terminology. Additional forks add new information that can increase a tree's prediction accuracy.
The ultimate (final) branches of the tree are called *leaf nodes*. The data provided to these models is called training data because it is used to train the model.
To test the tree's performance on new data, new data points are passed as parameters. This previously unused data is called test data.
There are four main types of learnings:

- Supervised - In such a learning setting there is a dataset, which is a collection of N labelled examples. Each example has a vector of features x_i and label y_i. The label y_i can belong to a finite set of classes $\{1, 2, ..., C\}$, a real number or even complex numbers. The goal of supervised learning models is to receive a feature x as an input and predict a correct label for it. (Valkov, 2019)

- Semi-supervised - In such a setting, the dataset contains both labelled and unlabelled examples. Usually, the quantity of unlabelled examples is much higher than labelled examples. The goal of this learning method is the same as that of Supervised learning.

- Unsupervised - In such a setting there is a dataset, which is a collection of N unlabelled examples. The goal is to receive a feature x as an input and transform it into another. Some practical examples include Clustering, Dimensionality reduction, Anomaly detection, etc.(Valkov, 2019)

- Reinforcement- This type of learning is basically concerned with building an agent to interact with an environment by fetching its state and executing an action. Actions provide rewards and change the state of the environment. The goal is to maximize the reward by learning a set of actions. (Valkov, 2019)

7.2.2 Main challenges of Machine Learning

The main things that can go wrong in the process of selecting a learning algorithm and training it are usually due to "bad algorithm" and "bad data".

- Insufficient Training Data - For ML models to work properly a lot of data is needed.There may be a need for thousands of examples even for very trivial problems, and may be even millions of examples for complex problems like image or speech recognition. (Valkov, 2019)

  As per the paper (Michele Banko, 2001), published by Microsoft Researchers Michele Banko and Eric Brill, relatively different ML Algorithms performed almost identically on a complex problem of Natural Language Disambiguation after providing the algorithms with enough data. As per the authors (Michele Banko, 2001)

  > these results suggest that we may want to reconsider the trade-off between spending time and money on algorithm development versus spending it on corpus development.

  This development further gained strength through the paper by Peter Novrig et al. (Alon Halevy, 2009) titled "The Unreasonable Effectiveness of Data"

- Bad Quality Data - A system is likely to perform bad if the training data is full of errors, outliers, and noise as this makes it hard for the system to detect patterns. Therefore, cleaning the data is a necessary activity for Machine Learning.

7.2.3 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.

7.2.4 Feature Engineering

The ML Model can only learn if the training data contains higher number of relevant features versus a low or even a very low number of irrelevant features. This is, in fact, a very critical part for a successful Machine Learning project and is known as *Feature Engineering*. Feature Engineering involves:

- Feature Selection
- Feature Extraction - combining features to produce more meaningful ones (Valkov, 2019)
- Gathering new data to create new features.

7.2.5 Overfitting the Training Data

The goal of a Machine Learning model is to learn patterns in data in order to identify these patterns in data which has not been previously exposed to the model. This process of learning patterns is known as *generalisation* and the goal of a Machine Learning model is to increase this generalisation which enables the model to make efficient predictions.
A *fit*, in statistics, is the approximation of a target function and the process of measuring this approximation is known as the *goodness of fit*.
Learning from a datset involves two concepts:

- Signal - A signal is a pattern which a ML model learns
- Noise - Noise is the randomness or irrelavant data in a dataset.

Overfitting occurs when the Machine Learning model generalizes too well which means that it learns patterns (signals) in a dataset leading to it eventually decreasing the performance on new data.
This can be mitigated, generally, by following the basic steps mentioned in the paper (López de Prado, 2020)

- decreasing the number of parameters required by the learning model
- selecting a model that requires fewer parameters
- more training data for training the model
- noise reduction in the data

According to the paper by (Li et al., 2018), "hyperparameters" are input parameters provided to a machine learning algorithm which control the performance on new data; hyperparameters are usually parameters that control the required amount of regularization, learning rates, and the learning process of an algorithm. The efficiency of a Machine Learning model depends on how these hyperparameters are configured or tuned.
How these hyperparameters interact with each other is still a topic under research.

7.2.6 Underfitting

Underfitting is the opposite of Overfitting. This occurs when it is too difficult for the model to learn the patterns in data i.e. it does not generalize the data.
As per the paper by (Allamy, 2014), this problem can be usually fixed by:

- Selecting a more powerful model, with more parameters
- Passing better features as parameters to the learning algorithm
- constraint reduction (e.g. reduction of the regularization hyperparameter)

7.3 Predictive modelling

Predictive modeling is the process of predicting outcomes based on data provided to a model in order to identify these predictions in new, unseen data.
Predictive modeling is a mathematical problem for approximating a mapping function f from input variables X to output variables y. This is also known as *Function Approximation*
These variables are usually Random Variables.

$$f(X) :\rightarrow Y \tag{1}$$

In general, function approximation tasks are divided into

- Classification tasks
- Regression tasks

7.3.1 Classification Predictive Modelling

Classification Predictive modeling is a mathematical problem for classifying data in classes based on what the model has learned from training data. It predicts a **discrete** output variable y from input variables X by approximating a mapping function f.
A Random Variable is discrete if

$$P(X = a_j \text{ for some } j) = 1 \tag{2}$$

, where a_j is a finite list of values for $j = 1, ..., n$ or an infinite list of values for $j = 1, ...$
If X is a discrete Random Variable, then the finite or infinite set of values a such that $P(X = a) > 0$ is called the *support* of X.
It is used for predicting the class or category for a given observation. The Credit Card Fraud Detection dataset is basically a classification problem of two classes viz. Fraud and Non-fraud, which is in turn a Binary classification problem.
Two types of learners in classification are lazy learners and eager learners.

- Lazy learners - These learners store the training data in a data structure to learn patterns in it and classify the patterns in new data (test data) based on the patterns similar to or matching those in training data. They have a small learning time but a high prediction time as they try to match patterns in new data to those identified in old data. Some popular examples of Lazy learners are K - Nearest Neighbour, Case - Based Reasoning.
- Eager learners - Eager learners perform the classification task based on the given training data before receiving data for classification. These learners start the learning task as soon as they receive the training data which results in a low prediction time but a high learning time. Some popular Eager learning algorithms are Decision Tree, Naive Bayes, Artificial Neural Networks.

7.3.2 Regression Predictive Modelling

Regression Predictive modeling is a mathematical problem for predicting a **continuous** output variable y from input variables X by approximating a mapping function f.

A Random Variable is said to be continuous if its Cumulative Density Function can be differentiated.

$$f(t) = F'(t) \tag{3}$$

, where $t \geq 0, f(t)$ is the PDF and $F'(t)$ is the derivative of $F(t)$

7.4 Class Imbalance Problem

When a dataset is dominated by data belonging to one class in comparison to other classes, the dataset is said to be imbalanced and this problem is known as a class imbalance problem. Most real-world applications (datasets) have such a class distribution where the count of one class label heavily dominates the count of another class label. Fraud detection is one of the best examples of class imbalance, where the number of fraud class label is very low compared to the normal class label. Most machine learning algorithms have a very poor performance for an unbalanced dataset. In the next sections, we will discuss how this problem can be tackled, which algorithms can be used, and the evaluation metrics that can be used for performance assessment.

7.5 Solutions to the Class imbalance problem

The popular approaches for solving the class imbalance problem can generally be classified into the following categories: sampling/resampling approach, ensemble learning approach, and cost-sensitive learning approach. This Thesis focuses on resampling and ensemble approach which are discussed in the following sections.

7.6 Resampling

As previously mentioned, a lot of Machine Learning algorithms do not perform well for an unbalanced dataset. Therefore, this unbalanced data needs to be processed before feeding it to the Machine Learning algorithms. Resampling is of, basically, three types: undersampling, oversampling, and hybrid.

- Undersampling: The number of the majority class data is reduced in order to balance the dataset. This sampling technique is considered to be beneficial when the dataset size is huge and reducing the majority class data (samples) improves the runtime.
- Oversampling: It is the opposite of Undersampling. Instead of the majority class, Oversampling replicates the minority class data in order to balance the majority and minority class data numbers.
- Hybrid: A hybrid approach, as the name suggests, implements both over and undersampling approaches in order to rebelance the data.

7.7 Random Undersampling

As the name suggests, this method randomly removes the majority class data to achieve a balanced dataset. It is preferred to be used when the training dataset is huge. Some advantages of this method include an improved runtime whereas an important disadvantage is that there is no way to track which majority class data will be deleted, therefore the loss of valuable information can reduce the effectiveness of this method.

7.8 Tomek Link removals

As mentioned by (Batista et al., 2004), a Tomek Link is a pair of examples belonging to different classes which are each other's nearest neighbours. If D1, D2 are two samples belonging to two different classes in a dataset, then the pair (D1, D2) is a Tomek Link if there is no sample D3 such that the distance between D3 and D1 or D3 and D2 is less than the distance between D1 and D2. This approach can be considered as an undersampling approach.

7.9 Random Oversampling

This method is the opposite of random undersampling. It randomly removes the minority class data to achieve a balanced dataset. It does not lead to information loss like random undersampling does but there is a high chance of overfitting the data as it copied the minority class data over and over in order to achieve a balanced dataset.

7.10 Synthetic Minority Oversampling Technique (SMOTE)

SMOTE is a very popular technique used for rebalancing. This technique, which was developed by Chawla el al. (Bowyer et al., 2011), creates new minority class data (examples) by interpolating between several nearest minority example data, rather than replacing as done by Random oversampling which reduces the problem of overfitting the training dataset. SMOTE selects these nearest minority class examples randomly depending on the amount of oversampling required.

7.11 SMOTE and Tomek Link removal together

As per Chawla el al. (Bowyer et al., 2011) SMOTE, whilst creating new synthetic minority examples tends to overfit the data. This needs to be mitigated and an efficient technique to mitigate this overfitting is to us SMOTE along with removal of Tomek Links. Over sampling is first carried out using SMOTE followed by removal of Tomek Links in order to achieve a balanced dataset.

7.12 Ensemble Learning

The process of aggregating answers for a question from different sources is the basic idea behind the term *ensemble*.
In a Machine Learning context, an ensemble is a group of predictors. The technique of learning from a dataset to make predictions is known as *ensemble learning* and this method of learning is known as an *ensemble method*.
According to the paper by Peter Bühlmann (Bühlmann, 2012b)
Assume a training set of data

$$D = (x_1, y_1), ..., (x_n, y_n) \tag{4}$$

drawn from a probability distribution $(x_n, y_n) \sim (X, Y)$
Given an fixed ensemble of classifiers

$$h = h_1(x), ..., h_K(x) \text{for } K \geq 1 \tag{5}$$

Consider A to be an outcome for a classifier $h_k(x)$, we define

$$\hat{P}(n) = \text{empirical probability of A} \tag{6}$$

The empirical margin function is given by

$$\hat{m}(x,y) = \hat{P}_k(h_k(x) = y) - \max_{j \neq y} \hat{P}_k(h_k(x) = y) \tag{7}$$

$$= \text{average margin of the ensemble of classifiers} \tag{8}$$

$$h_k(x) = h(x|\Theta_k) \text{where} k \in K, \Theta_k = (\theta_{k_1}, ..., \theta_{k_n}) \tag{9}$$

The *generalisation error* of the classifier ensemble is

$$e = P_{x,y}(\hat{m}(x,y) < 0) \tag{10}$$

As $K \to \infty$, where K is the number of trees,

$$e \to_{K \to \infty} P_{x,y}\left[P_\Theta(h(x,\Theta) = y) - \max_{j \neq y} \hat{P}_\Theta(h_k(x,\Theta) = j) < 0\right]$$

There are a lot of ensemble methods available viz. *bagging, boosting, stacking, etc.* For our research we tried out the following popular methods

- Random Forests (Bagging)
- Boosting

7.12.1 Voting Classifiers

A very simple way to create an even better classifier is to aggregate the predictions of each classifier and predict the class that gets the most votes. This is known as a *majority-vote classifier*. Suppose for a classification task we have three classifiers, $h_1(X), h_2(X), h_3(X)$. In order to produce a better classifier we must combine these classifiers. A very banal way to combine these rules is to calculate the *mode*.

$$h(X) = \text{mode}\{h_1(X), h_2(X), h_3(X)\} \tag{11}$$

This procedure can be easily extended for any number of classifiers and even for weighted classifiers (See Equation (12))

$$h(X) = \text{argmax}_i \sum_{j=1}^{N} \theta_j I(h_j(X) = i) \tag{12}$$

, where

- $\theta_1, ..., \theta_N$ are weights that sum to 1
- $I(.)$ is the indicator vector

In terms of empirical probability Equation (12) can be rewritten as:

$$h(X) = \text{argmax}_i \sum_{j=1}^{N} \theta_j \hat{p}_j^i \tag{13}$$

, where

- $\theta_1, ..., \theta_N$ are weights that sum to 1
- $\hat{p}_j^i$ is the probability estimate from the j^{th} classification to i^{th} classification.

Equation (11) is referred as Majority Vote Learner (MaVL) and equation (12) is referred as Semi Majority Vote Learner (Semi MaVL)

7.12.2 Bagging

Bootstrap aggregation, also known as *Bagging*, is an ensemble method for improving estimation or classification schemes. According to Leo Breiman (Breiman, 1996), Bagging is a variation reduction technique for a given base procedure.
Bühlmann, Peter and Yu, Bin (Bühlmann and Yu, 2002) showed that Bagging is a smoothing operation used to effectively improve the predictive performance of regression and classification trees.
Consider a regression or classification setting,

$$(X^i, Y^i) \text{ for } i = 1, ..., n \tag{14}$$

where

- $X^i \in \mathbb{R} \text{and} X^i$ is a vector (predictor variable).
- $Y^i \in 0, 1, ..., J-1$ is the classification with J classes.
- $\mathbb{E}[Y|X = x]$ is the target function for regression.
- $\mathbb{P}[Y = j|X = x]$ for $j = 1, ..., n$ is the target function for classification.

$h = \hat{g}(.) = h_n\left((X^1, Y^1), ..., (X^n, Y^n)\right)(.) : \mathbb{R}^d \to \mathbb{R}$ is the Predictor or Estimator function

The algorithm constructs a bootstrap sample $((X^{1^*}, Y^{1^*}), ..., (X^{n^*}, Y^{n^*}))$ by randomly picking n times with replacement from $\left((X^1, Y^1), ..., (X^n, Y^n)\right)$
It then computes the bootstrap estimator $\hat{g}^*(.) = h_n\left((X^{1^*}, Y^{1^*}), ..., (X^{n^*}, Y^{n^*})\right)(.)$
The algorithm then repeats these two steps M number of times yielding $\hat{g}^{*k}(.)$ for $j = 1, ..., M$.
The bagged estimator here is defined as $\hat{g}_{Bag}(.) = M^{-1} \sum_{k=1}^{M} \hat{g}^{*k}(.)$.
In theory,

$$\hat{g}_{Bag}(.) = \mathbb{E}^*[\hat{g}^*(.)] \text{for} M = \infty \tag{15}$$

According to Peter Bühlmann (Bühlmann, 2012a) the finite number M in practice governs the accuracy of the Monte Carlo approximation but otherwise, it should not be viewed as a tuning parameter for bagging.
For Classification, the estimator function is defined as:

$$h = \hat{g}_j^{*k}(.) = \hat{\mathbb{P}^*}[Y^{*k} = j|X^{*k} = .] \text{for} j = 0, 1, ..., J-1 \tag{16}$$

which yields an estimator $\mathbb{P}[Y = j|X = .]$

7.12.3 Out-of-bag Error Estimation

Consider a loss $\rho(Y, \hat{g}(X)) = |Y - \hat{g}(X)|^2$ which is the measure of the noise(discrepancy) between the estimated function, $\hat{g}$, evaluated at X, and the corresponding Y.
The generalization error in this case is:

$$err = \mathbb{E}\left[\rho(Y, \hat{g}(X))\right] \tag{17}$$

where

- the expectation $\mathbb{E}$ is calculated over the independent identically distributed training data $(X^1, Y^1), ..., (X^n, Y^n)$.
- $\hat{g}(.)$ is a function of the training data
- (X, Y) is a new test observation.

According to Peter Bühlmann (Bühlmann, 2012a, p. 10), approximately $\exp(-1) \approx 37\%$ of the original observations in a bagged sample are left out.
These left-out observations, according to Leo Breiman (Breiman, 1996), are known as "out-of-bag" observations denoted by $Boot^k$.
$Boot^k$ are basically the original sample indices that were resampled in the k^{th} bootstrap sample.
The out-of-bag error estimate function is defined as:

$$\hat{err}_{OB} = n^{-1} \sum_{i=1}^{n} N_M^{-1} \sum_{k=1}^{M} I_{[(X^i, Y^i) \notin Boot^k] \rho(Y^i, \hat{g}^k(X^i))}$$

$$N_M = \sum_{k=1}^{M} I_{[(X^i, Y^i) \notin Boot^k]}$$

7.12.4 Boosting

Boosting (also known as *hypothesis boosting*), as proposed by Schapire (Schapire, 1990) and (Schapire, 2002) and also mentioned by Freund (Freund, 1995) (Freund and Schapire, 1996), is an ensemble method which combines several weak learners to produce a strong learner.
It trains the predictors sequentially with each predictor trying to correct its predecessor.
These are sequential ensemble methods where the weights θ_k in Equation (18) depend on $\hat{g}_1, ..., \hat{g}_{k-1}$

$$\hat{g}_{ens}(.) = \sum_{k=1}^{M} \theta_k \hat{g}_k(.) \tag{18}$$

A review of boosting is presented by Buehlmann and Hothorn in (Buehlmann and Hothorn, 2007) where concepts and algorithms have been very elaborately illustrated.
As pointed out by L. Breiman (Breiman, 1998) (Breiman, 1999), boosting can be viewed as a non-parametric optimization algorithm in functional thesis.
The Gradient Boosting algorithm (XGBoost) was used in this Thesis to predict fraudulent transactions.

7.12.5 Boosting as a gradient descent

L. Breiman (Breiman, 1998) (Breiman, 1999) introduced boosting as a gradient descent method. The goal is to predict a function (See equation (19)) which minimizes loss (See equation (20))

$$g : \mathbb{R}^d \to \mathbb{R} \tag{19}$$

$$\mathbb{E}\left[\rho(Y, g(X))\right] \tag{20}$$

$$\rho(.,.) : \mathbb{R} \text{ x } \mathbb{R} \to \mathbb{R}^+ \tag{21}$$

based on (X^i, Y^i) for $i = 1, ..., n$. The loss function, ρ, is convex (See equation (21)).
The classification case, where the response Y is a discrete Random Variable, has been taken into consideration.
As mentioned by L. Breiman (Breiman, 1998) (Breiman, 1999), boosting algorithms have a low empirical risk (See equation (22)).

$$n^{-1} \sum_{i=1}^{n} \rho(Y_i, g(Y_i)) \tag{22}$$

where $g(.) = \sum_k \theta_k g_k(.)$

7.12.6 Margin for classification

The margin is denoted by $\bar{y}g$ where $\bar{y} = 2y - 1 \in \{-1, 1\}$ Therefore, loss is a function of $\bar{y}g$ only (See equation (23)).

$$(\bar{y} - g)^2 = \bar{y}^2 - 2\bar{y}g + g^2 \tag{23}$$

$$= 1 - 2\bar{y}g + (\bar{y}g)^2 \tag{24}$$

using $y^2 = 1$ The misclassification loss $\mathbb{I}_{[\bar{y}g<0]}$ is also a function of $\bar{y}g$

7.13 Selected models

The this section we discuss the models selected for predictive analysis. We chose very popular predictive models like Logistic Regression, Ensemble Learning methods like Random Forest and XGBoost. The main motivation behind choosing these methods was based to the paper by Kostiantis et al. (Kotsiantis and Pintelas, 2005) where the authors quoted that

> ensembles are often much more accurate than the individual classifiers that make them up. The main reason is that many learning algorithms apply local optimization techniques, which may get stuck in local optima. For instance, decision trees employ a greedy local optimization approach, and neural networks apply gradient descent techniques to minimize an error function over the training data. As a consequence even if the learning algorithm can in principle find the best hypothesis, we actually may not be able to find it. Building an ensemble may achieve a better approximation, even if no assurance of this is given. Boosting algorithms are considered stronger than bagging on noise-free data, however, bagging is much more robust than boosting in noisy settings

7.13.1 Logistic Regression

Logistic Regression, also known as *logit*, is a regression algorithm commonly used for classification predictive modelling computes the probability of a set of values belonging to a class. It works on the principle of a binary classifier. The goal of Logistic Regression is to compute a parameter vector and uses maximum likelihood method for parameter estimation.

$$odds = \frac{P(event)}{1 - P(event)} \tag{25}$$

$$\text{Let } P(y = 1|X) = p(X) \text{ such that } X \in \mathbb{R} \text{ and } p(X) \in [0, 1] \tag{26}$$

The *Sigmoid function*, as per King (King and Zeng, 2001), is written as:

$$\sigma(X) = \frac{1}{1 + \exp(-t)} \tag{27}$$

This $\sigma(X)$ is the $p(X)$. The equation (27) can be written in terms of log odds as:

$$\log \frac{p(X)}{1 - p(X)} = t = \theta^T.X \tag{28}$$

The goal of Logistic Regression is to is to estimate parameter vector $\hat{\theta}$. It uses *maximum likelihood* method for parameter estimation.
The θ^T is the transpose (row vector instead of column vector) of θ - the model's parameter vector. The θ^T has been written as θ in the derivations further.
Consider N samples with labels 0 or 1. For samples labelled 1, we need to estimate $\hat{\theta}$ such that $\widehat{p(X)}$ is as close to 1 as possible and for samples labelled 0, we need to estimate $\hat{\theta}$ such that $1 - \widehat{p(X)}$ is as close to 1 as possible.
This can be shows as the product of all samples labelled as 1 (See equation (29)) and all samples labelled as 0 (See equation (30)).

$$\prod_{s \in y^i = 1} p(x^i) \tag{29}$$

$$\prod_{s \in y^i = 0} (1 - p(x^i)) \tag{30}$$

, where s is the sample.
Combining these equations (29) and (30) in the likelihood function, we obtain:

$$L(\theta) = \prod_{s \in y^i = 1} p(x^i) \prod_{s \in y^i = 0} (1 - p(x^i)) \tag{31}$$

$$L(\theta) = \prod p(x^i)^{(y^i)} \prod (1 - p(x^i))^{(1 - y^i)} \tag{32}$$

$$\tag{33}$$

Converting this to log-likelihood, we get:

$$l(\theta) = \sum_{i=1}^{n} y^i log(p(x^i)) + \sum_{i=1}^{n} (1 - y^i) log(1 - p(x^i)) \tag{34}$$

Substituting $p(x^i)$ with its exponential form as seen in equation (28), we get:

$$l(\theta) = \sum_{i=1}^{n} y^i y^i \theta x^i - log(1 + \exp^{\theta x^i}) \tag{35}$$

Such equations are also known as *transindental equations*
The goal is: $\theta =$ argmax $_\theta l(\theta)$.
For this the *Newton Raphson* method is used as per the paper by Akram et al. (Akram and ul Ann, 2015)

$$\nabla_\theta l(\theta) = \nabla_\theta l(\theta^*) + (\theta - \theta^*) \nabla_{\theta\theta} l(\theta^*) \tag{36}$$

where θ^* is the global minimum and ∇_θ is the *gradient* with respect to θ

$$\implies \nabla_\theta l(\theta^*) + (\theta - \theta^*) \nabla_{\theta\theta} l(\theta^*) = 0 \tag{37}$$

$$\implies \theta = \theta^* - \frac{\nabla_\theta l(\theta^*)}{\nabla_{\theta\theta} l(\theta^*)} \tag{38}$$

and

$$\theta^{(t+1)} = \theta^t - \frac{\nabla_\theta l(\theta^t)}{\nabla_{\theta\theta} l(\theta^t)} \tag{39}$$

where $\nabla_{\theta\theta}$ is the *Hessian Matrix* which is the gradient of ∇_θ.
The equation :

$$\nabla_\theta l = \nabla_\theta \left[\sum_{i=1} n y^i \theta x^i - \log(1 + \exp(\theta x^i)) \right] \tag{40}$$

can be further written as

$$\sum_{i=1} n \left[y^i - p(x^i) \right] x^i \tag{41}$$

and therefore, the equation :

$$\nabla_{\theta\theta} l = \nabla_\theta \sum_{i=1} n \left[y^i - p(x^i) \right] x^i \tag{42}$$

is the gradient of

$$\nabla_\theta l \tag{43}$$

and can be further written as

$$\sum_{i=1} n p(x^i) \left[y^i - p(x^i) \right] {(x^i)}^T x^i \tag{44}$$

where $(x^i)^T$ is the transpose of x^i.

This can further be written in matrix form as:

$$\nabla_\theta l = X^T(Y - \hat{Y}) \tag{45}$$

$$\nabla_{\theta\theta} l = -X^T P(1-P)X = -X^T W X \tag{46}$$

where $P(1-P)$ is the diagonal matrix W

$$\implies \theta^{t+1} = \theta^t + (X^T W^t X)^{-1}(Y - \hat{Y}) \tag{47}$$

By choosing a threshold value, the probability can now be estimated.

$$p(X) = \frac{1}{1 + \exp(-\theta X)} = \hat{p} \tag{48}$$

After estimating the probability $\hat{p} = h_\theta(x)$ of data (instance) belonging to the positive class, it is easily able to make the prediction $\hat{y}$ (See equation (49))

$$\hat{y} = \begin{cases} 0, & \text{if } \hat{p} < 0.5 \\ 1, & \text{if } \hat{p} \geq 0.5 \end{cases} \tag{49}$$

The cost function of the Logistic Regression is just the average cost of all training instances for the whole training set. It is popularly known as the *log loss*, show in equation (50)

$$h_\theta = -\frac{1}{m}\sum_{i=1}^{m}\left[y^i \log\left(\widehat{p^i}\right) + \left(1 - y^i\right)\log\left(1 - \widehat{p^i}\right)\right] \tag{50}$$

This cost function is a convex function, therefore any optimization algorithm is going to find the global minimum (According to King et al. (King and Zeng, 2001)).
The partial derivatives (See equation (51)) with j^{th} θ_j basically computes the average after computing the prediction error and multiplying by the j^{th} feature value. This gradient vector containing all partial derivatives can now be used in the Batch Gradient Descent.

$$\frac{\partial}{\partial\theta_j} h_\theta = -\frac{1}{m}\sum_{i=1}^{m}\left[\sigma\left(\theta^T.x^i\right) - y^i\right] x^i_{(j)} \tag{51}$$

7.13.2 Decision Trees

Decision Trees are very versatile Machine Learning algorithms which are capable of performing both classification and regression tasks.
The Basics - Assume that there are n data samples and feature vectors $\{x^i\}_{i=1}^n$ with classes y^i. A decision tree (predictive model) is used to go from observations about an item (represented in the branches) to conclusions about the item's value (represented in the leaves). It has a *root node* (depth zero, at the top of the tree). If a condition is satisfied, the observer moves down in either the left direction or right, depending on what it wants to conclude.
Every node has a *samples* attribute, which counts how many instances it is related to, and a *value* attribute, which tells how many training instances of each class it applies to, as mentioned in the publication by Klusowski et al. (Klusowski, 2020). **Gini Impurity** - A node also has a *gini* attribute (See equation (52)) which measures its *impurity*. According to Jason Klusowski

(Klusowski, 2020), a node is said to be "pure" if all training instances it applies to, belong to the same class. In that case, it is said to have *gini = 0*.

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2 \tag{52}$$

, where

- G_i is the gini score
- $p_{i,k}$ is the ratio of class k instances among the training instances in the i^{th} node.

Computational Complexity - A prediction cycle in a Decision Tree involves traversing it from the root node to the leaf node. This computation usually has a complexity of $O(\log_2(m))$ and since each node, in one traversal cycle, is checked only for one feature, the overall prediction complexity is a mere $O(\log_2(m))$.
The training algorithm, however, compares all features and therefore it accumulates a training complexity of $O(n \text{ x } m\log(m))$ as shown in the section on Decision Trees in the paper by Klusowski et al. (Klusowski, 2020).
Another measure of impurity - Another measure of impurity for a Decision Tree is known as *Entropy* impurity measure (See equation (53)). A set's entropy has a value zero when it contains instances of only one class. The following equation (53) shows the entropy of the i^{th} node.

$$H_i = -\sum_{k=1_{p_{i,k}\neq 0}}^{n} p_{i,k}\log(p_{i,k}) \tag{53}$$

Gini impurity is slightly faster to compute as compared to entropy but it tends to isolate the most frequently occurring class in its own branch of the tree, while entropy, on the other hand, tends to produce slightly more balanced trees as mentioned in the paper by Laura E. Raileanu et al. (Raileanu and Stoffel, 2000).
Regularisation - Compared to linear models that assume that the data provided to them is linear, the Decision Trees make very few assumptions about the data, this basically means that they can work on any type of data that is provided to them, which is obviously based on very loose details.
As mentioned previously, the tree structure adapts itself to the data provided, thus increasing the possibility of overfitting it. It produces a model, known as a *non-parametric model*, which does not have the parameters determined prior to the training task.
To avoid this overfitting, regularisation is needed which is subjective to the algorithm but is usually controlled by setting the *max_depth* parameter as mentioned earlier.
Instability The main issue with Decision Trees is their sensitivity. This means that they are highly sensitive to small variations in the training data.
Random Forests can limit this instability by averaging predictions over many trees, which is discussed in the next section 7.13.3.

7.13.3 Random Forests

An ensemble of Decision Trees, according to Ho et al. (Tin Kam Ho, 1995), is a Random Forest.

A random forest searches for the best feature among a random subset of features and adds extra randomness during the growing phase of the tree instead of searching the best feature for a node during splitting, according to Ho et al.(Tin Kam Ho, 1995).
This randomness results in a much diverse tree structure, as compared to a Decision Tree, supported by a lower variance.
Mathematically speaking,
Assume a training set of data

$$D = (x^1, y^1), ..., (x^n, y^n) \tag{54}$$

drawn randomly from a probability distribution $(x^n, y^n) \sim (X, Y)$
Given an ensemble of classifiers

$$h = h_1(x), ..., h_K(x) \text{for K} \geq 1 \tag{55}$$

If each $h_k(x)$ is a decision tree, then this ensemble is a random forest (See equation (56)).

$$h_k(x) = h(x|\Theta_k) \text{where} k \in K, \Theta_k = (\theta_{k_1}, ..., \theta_{k_n}) \tag{56}$$

The classification $f(x)$ is done on a voting scheme.

7.13.4 XGBoost

Extreme Gradient Boosting popularly known as XGBoost is one of the most efficient boosting algorithms till date. XGBoost is based on the boosting technique ,explained previously, usually used with decision tree as a weak learner. We have discussed batch learning earlier and we explain a new approach to Machine Learning known as *stream learning*.
Stream learning is a machine learning technique that involves learning from data streams.
For our Credit Card Fraud Detection dataset, run-time consideration is a critical measure. For fraud detection in transaction data waiting for a long time before prediction can lead to a lot of fraudulent transactions being missed out.
In stream learning, the learning models have access to data only once and thus need to process it in real time. This limited access may lead to change in relationship between features and targets as the data keeps on changing. This change in data, as mentioned by the authors Jacob Montiel et al. (Montiel et al., 2020) is known as *concept drift*. The relationship between features and targets is known as a "concept".
In case of a concept drift, batch methods usually fail as they're trained on a different concept whereas stream learning models are constantly updated to accommodate this concept drift.
In stream learning there are two main types of algorithms which are distinguished from each other based on the schema used to train the model.

- Instance - these are incremental algorithms where a single data sample is used at a time.
- Batch incremental - these are incremental algorithms where batches of data samples are used one at a time. Once a given (parameter) number of data samples are stored in a batch, the learning model is trained.

The *Accuracy weighted ensemble*, as mentioned in the paper by Monteil et al. (Montiel et al., 2020), is a framework that uses weighted classifiers for mining streams with a concept drift. Ensemble predictors are discarded if their prediction is below a threshold value based on an instance-based pruning strategy.

The goal of the XGBoost algorithm is to predict

$$Y = y_i : i \in 1, 2, ..., n \tag{57}$$

corresponding to a set of feature vectors $X = x_i : i \in 1, 2, ..., n$, as mentioned by Chen et al. (Chen and Guestrin, 2016).
Ensemble methods yield $\hat{y}_i$ corresponding to a given input x_i by combining predictors of all members of the ensemble learning, as per Chen et al. (Chen and Guestrin, 2016).

$$h_K = \sum_{k=1}^{K} h(Y, \hat{Y}^{k-1} + f_k(X)) + \omega(f_k) \tag{58}$$

, where

- $f_k \in$ F is the space of the base functions
- $\hat{Y}$ is the predicton
- f_k is the base function
- $\omega(f_k)$ is the regularisation parameter (penalty function)

The ensemble is created using a technique known as *forward additive modelling*, as mentioned in the paper by Monteil et al. (Montiel et al., 2020), where new trees are added one step at a time. At step k, the existing predictors evaluate the training data and the corresponding prediction scores Y^k, obtained after this evaluation, are used to create new predictors.

$$\hat{Y}_k = \sum_{k=1}^{K} f_k(X) = \hat{Y}_{k-1} + f_k(X) \tag{59}$$

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i) \tag{60}$$

, where $\hat{y}_i$ is the final prediction obtained by adding the predictions for each tree f_k.
According to Jacob Montiel et al. (Montiel et al., 2020), in a batch setting

$$\hat{Y}_k = \sum_{k=1}^{K} f_k(\theta_k) = \hat{Y}_{k-1} + f_k(\theta_k) \tag{61}$$

There are actually two strategies (Montiel et al., 2020) to update the ensemble.

- Push strategy - the ensemble in this strategy resembles a queue that works on a "First in first out" strategy (FIFO)
- Replacement strategy - in this strategy, the older predictors are replaced by the newer ones.

$$W_i = \min\left(W_{min}.2^i, W_{max}\right) \tag{62}$$

which is knows as the *dynamic window size* (Montiel et al., 2020) and W is the window (buffer) size.
In both the strategies mentioned above, the algorithm waits for K number of iterations for a new ensemble model.
As discussed earlier, the learning model in stream learning requires new predictions at anytime due to the concept drift, which may or may not occur, and thus a window (buffer) is needed to make predictions to accommodate for the new prediction time. This is known as the *window*.

$$i = \left[\log_2 \frac{W_{max}}{W_{min}}\right] \tag{63}$$

, where

$$\sum_{i=0}^{K-1} W_{min}.2^i << K.W_{max} \tag{64}$$

According to Jacob Montiel et al. (Montiel et al., 2020), to handle the concept drift, the XGBoost algorithm uses the method ADWIN to track changes in the performance of XGBoost taking the concept drift into consideration measured by a metric such as *Classification Accuracy* (Montiel et al., 2020).

7.14 Evaluation Metrics

Different performance metrics can be used for evaluating different Machine Learning algorithms. In this study the focused has been on *Confusion Matrix, Precision, Recall, F1 Score, Area under the Precision Recall Curve (AUC-PR) and the Area under the Receiver Operating Characteristic Curve (AUC-ROC).*

7.14.1 Confusion Matrix

A metric for measuring the accuracy and correctness of the predictions by a model is the confusion matrix. It is used for Classification problems where the output can belong to two or more types of classes. The confusion matrix, is an `N x N` matrix where `N` is the number of class labels that need to be classified, describes the overall performance of a predictive model when used on some dataset.
This metric is generally a measure of the count of the number of errors or false predictions when feature A (class A) is classified as feature B or vice versa.
Each column in a confusion matrix represents a predicted class whereas each row represents the actual class.
This matrix consists of the following statistical measures:

- True Positive (TP): a value is called a TP when the actual and the predicted values both are positive.
- False Positive (FP): a value is called an FP when the actual value is negative and the predicted value is positive.

- True Negative (TN): a value is called a TN when the actual value is negative and the predicted value is negative.
- False Negative (FN): a value is called a TN when the actual value is positive and the predicted value is negative.

A perfect classifier has only true positives and true negatives, so its confusion matrix would have non-zero values on its main diagonal (top left to bottom right)

7.14.2 Precision

We obtain a lot of information from the confusion matrix, but it is usually easy to interpret a concise metric.
Precision is defined as the accuracy of the positive predictions (See equation (65)).

$$precision = \frac{TP}{TP + FP} \tag{65}$$

, where

- TP is the number of True Positives
- FP is the number of False Positives

7.14.3 Recall

One can compute one single correct positive prediction (prediction $= \frac{1}{1} = 100\%$). This is, of course, absurd as the classifier would ignore everything except one positive instance.
Therefore, precision is usually used with another metric, *recall* or *True positive rate (TPR)* which is a measure of the correctly detected positive instances (See equation (66)).

$$recall = \frac{TP}{TP + FN} \tag{66}$$

, where

- TP is the number of True Positives
- FN is the number of False Negatives

7.14.4 Precision and Recall - The F1 Score

Precision and Recall are often combined into a single metric known as the F1 score (See equation (67)).
The F1 score is basically the *harmonic mean* (See equation (67)) of the precision and recall.
The harmonic mean gives much more weight to low values as it is largely insensitive to outliers that have much larger values than the other data (Dodge, 2008, p. 240). The classifier gets a higher F1 score if both recall and precision are high.
The harmonic mean of n observations is defined as n divided by the sum of the inverses of all of the observations.

According to Yadolah Dodge (Dodge, 2008, p. 240)

$$H = \frac{n}{\sum_{i=1} n \frac{1}{x^i}} \tag{67}$$

, where

- x^i are n non-zero quantities for $i = 1, ..., n$ of a quantitative variable X

$$F1 = \frac{TP}{TP + \frac{FN+FP}{2}} \tag{68}$$

, where

- TP is the number of True Positives
- FN is the number of False Negatives
- FP is the number of False Positives

7.14.5 Precision/Recall Trade-off

Precision increases as the recall decreases and vice versa. This relationship between precision and recall is commonly known as the *Precision/Recall Trade-off*.

7.14.6 Area under the Precision Recall Curve

The Area under the Precision Recall Curve (AUC-PR) is a commonly used tool to measure the performance of a predictive model in case of unbalanced data and is based on the Precision and Recall values shown at different probability thresholds. The area under this curve can be used to measure the performance of a model. Figure 1 shows an example of a Precision Recall Curve (PR Curve).

7.14.7 The ROC Curve

The *Receiver Operating Characteristic* is another commonly used tool with binary classifiers. The Receiver Operating Characteristic curve plots the recall (also known as the *True Positive Rate* or *sensitivity*) against the *False Positive Rate* unlike the precision/recall curve, which plots the precision versus recall.
The ratio of negative instances that are incorrectly classified as positive is known as the False Positive Rate.

$$FPR = 1 - TNR \tag{69}$$

, where

- TNR is the number of True Positive Rate
- FPR is the number of False Positive Rate

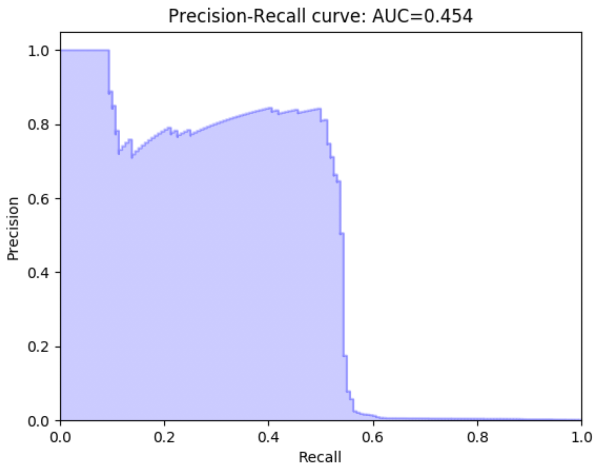

Figure 1: Precision Recall Curve

The ratio of correctly classified negative instances is the True Negtive Rate, also known as *specificity*.
The ROC curve plots the graph of *sensitivity* versus 1− *specificity*.
Just like the Precision/Recall trade-off discussed earlier, there exists a trade-off here as well: higher the recall (TPR), the more false positives (FPR) the classifier produces.
One way to compare classifiers is by computing the Area under the curve.

- A perfect classifier will have a value of AUC-ROC equal to 1
- A purely random classifier will have a value of AUC-ROC equal to 0.5

This fact, according to Serrano-López et al. (Serrano-López et al., 2010), can be used as a suitable measurement of global accuracy of classification. An interesting property that the AUC-ROC posesses is that the Area under the curve obtained without approximating the parameters corresponds with the Wilcoxon-Mann-Whitney statistic (Mann and Whitney, 1947). This enables testing the hypothesis accurately and establishing a result. Figure 2 shows an example of a Confusion Matrix.

8 Data and the method

8.1 Data description

The dataset used in this thesis was originally used for a research project (Pozzolo et al., 2015) carried out by Worldline and the Machine Learning Group of ULB (Universit Libre de Bruxelles), and it was also released on Kaggle, a community of data scientists and machine learners. The Credit Card Fraud Detection dataset (ULB, 2018) contains credit card transactions that occurred in September 2013 by many European cardholders.

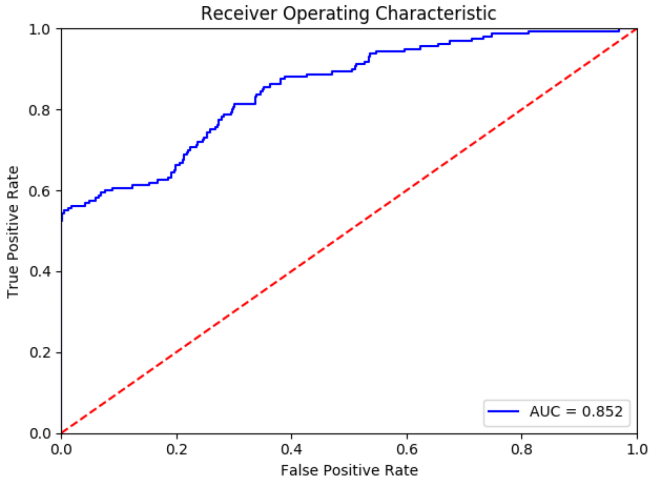

Figure 2: Confusion Matrix

Variable	Data Type	Description
V1-V28	Double	Principal components
Amount	Double	Amount of the transaction
Class	Integer	Class of the transaction (1 - Fraud or 0 - normal)
Time	Integer	Time between each transaction and the first transaction

Table 1: Description of the dataset

The time-span of this transactional data (ULB, 2018) is two days, where there are 492 identified frauds out of 284,807 transactions. This is a highly unbalanced dataset (ULB, 2018), in which the positive class (frauds) are 0.172 percent of all transactions, which can be seen in the Figure 3 Principle Component Analysis (PCA) transformation has been applied to the dataset (ULB, 2018), due to confidentiality issues, to not disclose the original features. There are total 30 features out of which 28 features have been generated by PCA.
PCA is a dimensionality-reduction technique in which a large number of original variables are reduced into a smaller subset of feature variables. The only features that have not been transformed into principal components are 'Amount' and 'Time'. The Table 1 shows the description of the data.

- Features V1, V2, ... V28 are the principal components obtained with PCA.
- the features Time and Amount have not been transformed using PCA. The feature Time contains time elapsed between each transaction and the first transaction in seconds where as the feature Amount contains the amount of the transaction.
- The feature Class is the features that is used to differentiate between the fraud cases, value 1, from the non fraud cases, with value 0.

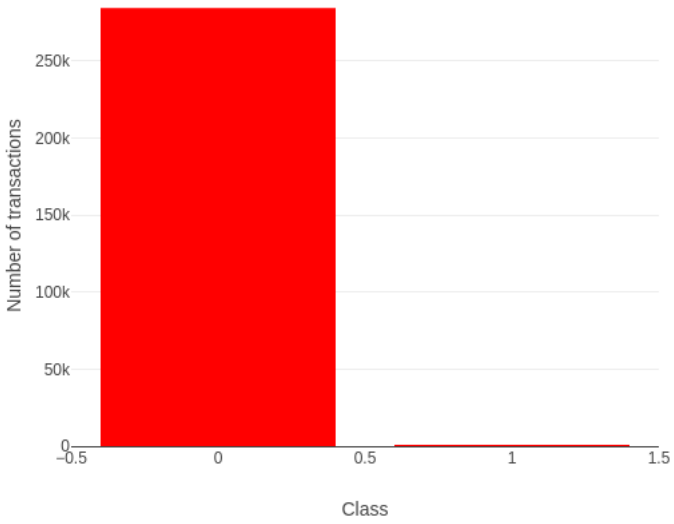

Figure 3: A comparison of the number of Fraudulent transactions versus the number of normal transactions

In predictive analysis, it is better to visualize the data before implementing the model. The data was visualised in terms of the correlation between variables. Two statistics that are used to measure the correlation between the datasets are **covariance** and **correlation coefficient**.
Covariance is mathematically defined as:

$$\mathrm{Cov}(X,Y) = \frac{\mathrm{E}(X-\mu_X)(Y-\mu_Y)}{\sigma_X \sigma_Y} \tag{70}$$

where

- Cov is the covariance
- μ_X is the mean of the random variable X
- μ_Y is the mean of the random variable Y
- σ_X is the standard deviation of the random variable X
- σ_Y is the standard deviation of the random variable Y

A very common method of computing the correlation is known as the **Pearson's Correlation Coefficient** or **Pearson's r** (Chee, 2015).
Pearson's r is a measure of the linear relationship between two interval or ratio variables, and can have a value between -1 and 1. It is the same measure as the point-biserial correlation; a measure of the relationship between a dichotomous (yes or no, male or female) and an interval/ratio variable according to Cramer et al. (Cramer, 1998)
According to Cramer (Cramer, 1998)

> Pearson's correlation is the ratio of the variance shared by two variables.

Given a pair of random variables *(X, Y)*:

$$r = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y} \tag{71}$$

where

- Cov is the covariance
- σ_X is the standard deviation of the random variable X
- σ_Y is the standard deviation of the random variable Y

In terms of the expectation of the random variables *(X, Y)*, the covariance is mathematically defined as:

$$\mathrm{Cov}(X, Y) = \frac{\mathrm{E}(X - \mu_X)(Y - \mu_Y)}{\sigma_X \sigma_Y} \tag{72}$$

We know,

$$\mu_X = \mathrm{E}[X]$$
$$\mu_Y = \mathrm{E}[Y]$$
$$\sigma_X^2 = \mathrm{E}[X]^2 - [\mathrm{E}[X]]^2$$
$$\sigma_Y^2 = \mathrm{E}[Y]^2 - [\mathrm{E}[Y]]^2$$

$$\mathrm{E}(X - \mu_X)(Y - \mu_Y) = \mathrm{E}(X - \mathrm{E}[X])(Y - \mathrm{E}[Y]) = \mathrm{E}[XY] - \mathrm{E}[X]\mathrm{E}[Y] \tag{73}$$

Now, substituting (73) into (70) and (71), we get

$$r_{X,Y} = \frac{\mathrm{E}[XY] - \mathrm{E}[X]\mathrm{E}[Y]}{\sqrt{\mathrm{E}[X]^2 - [\mathrm{E}[X]]^2}\sqrt{\mathrm{E}[Y]^2 - [\mathrm{E}[Y]]^2}} \tag{74}$$

The given correlation matrix (Figure 4) 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'.
The correlation coefficient mentioned in equation (71) ranges from -1 to 1. A closer value to 1 implies that it has a strong positive correlation as mentioned in the paper by Marilyn K. Simon (Marilyn K. Simon, 2011); for example, as the 'Amount' increases, the feature 'V7' also increases. When the correlation is closer to -1, then there is a strong negative correlation, as mentioned in the paper by Marilyn K. Simon (Marilyn K. Simon, 2011), which basically means that the values of the features tends to go down as the 'Amount' increases. In the end, there are values close to 0, which imply that there is no *linear correlation* as mentioned in the paper by Senthilnathan Samithamby (Senthilnathan, 2019).
As it can be seen in the correlation matrix (See Figure 4), the attributes V1 to V28 are almost uncorrelated except certain correlations between some of these attributes with Time. For example, 'Time' is inversely correlated with the attribute 'V3' and 'Amount' is directly correlated with 'V7' and 'V20' along with being inversely correlated with attributes 'V2' and 'V5'.
Generally, feature variables with higher correlation have a more significant impact during the training phase, as mentioned in the study by Failing et al. (Failing and Theeuwes, 2014).

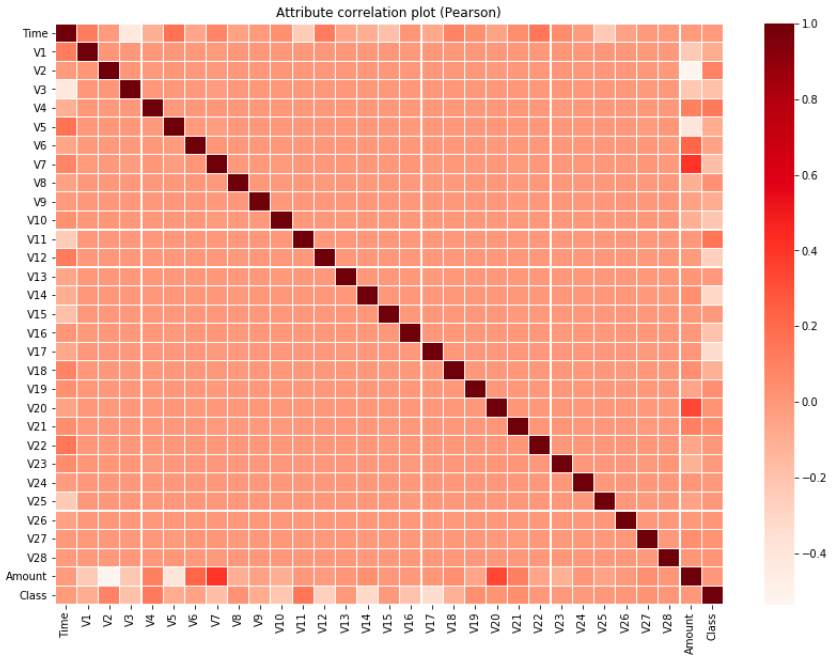

Figure 4: Heatmap for Correlation

8.2 Standardization of the data

A very common requirement for most Machine Learning algorithms is that the data be standardized. Data Standardisation refers to rescaling the features, so that they have a mean of 0 and a standard deviation of 1, so that they are like the standard normal distribution.

8.3 Splitting the data

A splitting ratio of 70:30 was chosen to split the data into 70% training and 30% test sets. The training set was used training the model, hyperparameter tuning and resampling whereas the test set was used to test the performance of the training model.

8.4 Data resampling

The dataset was resampled using using various resampling techniques like random under and oversampling along with SMOTE and removal of Tomek Links, which have been discussed in the previous sections in detail.

8.5 10 fold cross validation

K-fold cross-validation means splitting the training set into K-folds (in this case, ten), then making predictions and evaluating them on each fold using a model trained on the remaining folds.

As discussed earlier, "hyperparameters" are input parameters provided to a machine learning algorithm which control the performance on new data; hyperparameters are usually parameters that control the required amount of regularization, learning rates, and the learning process of an algorithm. The efficiency of a Machine Learning model depends on how these hyperparameters are configured or tuned and for tuning these hyperparameters, a popular technique used is cross-validation.

8.6 Training and Testing

After tuning the hyperparameters, the hyperparameters were set for each model and the resamples dataset was passed to each model for training. As a result, the models learned different patterns in the resampled training data. This trained model was then tested on the test set to eventually evaluate their performances.

8.7 Performance evaluation

In this thesis, the focus has been on Recall, Precision, F1 Score, AUC-PR and AUC-ROC as a measure of prediction efficiency. The accuracy of the model has not been used as Accuracy usually tends to give a misleading conclusion in case of an unbalanced dataset. As an example, consider a dataset with 100 transactions out of which 5 are fraudulent transactions. A model predicting all of the transactions passed to it as legitimate achieves an accuracy of 95%, which is very high but the model never predicted any fraudulent transaction.
The consequences of missing fraudulent transactions for any financial institution or an individual can be catastrophic. For the models to not miss fraudulent transactions in the dataset, the recall score had to be high. The precision should not be neglected in this case as it was equally important to not predict a transaction as fraudulent when it was not. The harmonic mean of both these measures, the F1 Score, was also considered as a metric for evaluating the prediction efficiency.
As mentioned by Saito et al. (Saito and Rehmsmeier, 2015) ROC is a popular and strong measure to evaluate the performance of binary classifiers. However, it requires special caution when used with imbalanced datasets. The Precision Recall Curve (PR Curve) changes with the ratio of positives and negatives.
In this thesis, the performance of the models was evaluated based on both these curves.

9 Results

The results of all the models have been compared with each other using the comparison metrics discussed previously viz. Precision, Recall, F1 score, AUC-ROC score, and AUC-PR score for all the previously discussed solutions to the class imbalance problem. Hyperparameters (as explained earlier in previous sections) were selected for each model using the GridSerachCV technique.

9.1 Logistic Regression

9.1.1 No resampling

The best value of the hyperparameter "C" for the Logistic Regression, with a value of 0.1, was found using the GridSerachCV technique. As discussed earlier, in case of an imbalanced dataset,

the logistic regression or any other model will generally perform very well for predicting the normal or non-fraud transactions. The Logistic Regression, in this case, performed very well without resampling with precision, recall and f1 scores of 0.9991, 0.9995 and 0.9993 respectively as shown in Table 2. On the contrary, logistic regression performed really bad when dealing with fraudulent transactions, where the precision and recall were 0.6774 and 0.5250. In addition, Precision Recall area under the curve (AUC-PR) and AUC-ROC curve values, which are shown in Figure 7 and Figure 6 respectively, also didn't turn out to be very satisfactory. Figure 5 shows the confusion matrix of the logistic regression model when no resampling was performed on the dataset.

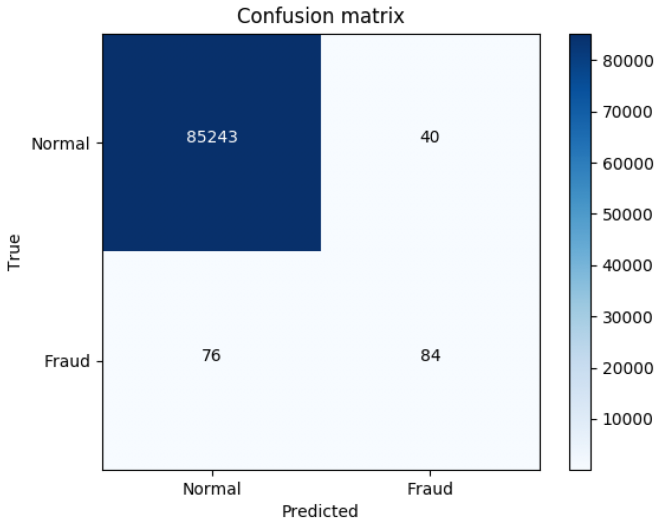

Figure 5: Logistic Regression Confusion Matrix - No Resampling

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0,9991	0,9995	0,9993
1	160	0,6774	0,5250	0,5915
avg/total	85443	0.9985	0.9986	0.9986

Table 2: Logistic Regression results for various threshold values for No Resampling

9.1.2 Random Undersampling

The best value of the hyperparameter "C" for the Logistic Regression, with a value of 0.1, was found using the GridSerachCV technique. Logistic Regression, in this case, performed very well on random undersampling of data with precision, recall and f1 scores of 0.9998, 0.9819 and 0.9908 respectively as shown in Table 3. The performance was satisfactory when dealing with fraudulent transactions, where the precision and recall were 0.0857 and 0.9062. The Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 10 and Figure 9

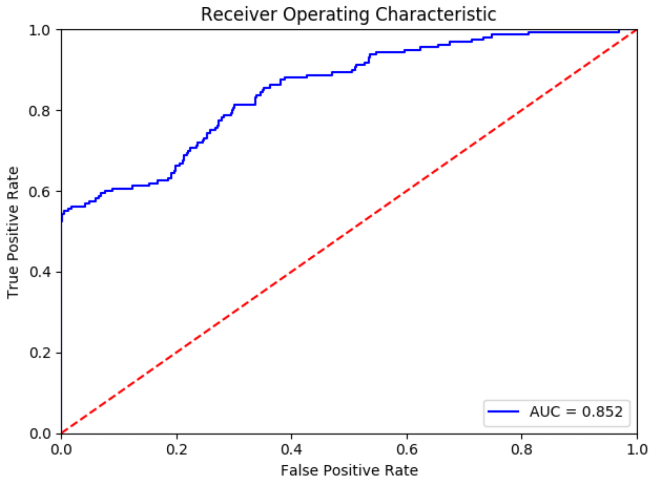

Figure 6: Logistic Regression AUC-ROC - No Resampling

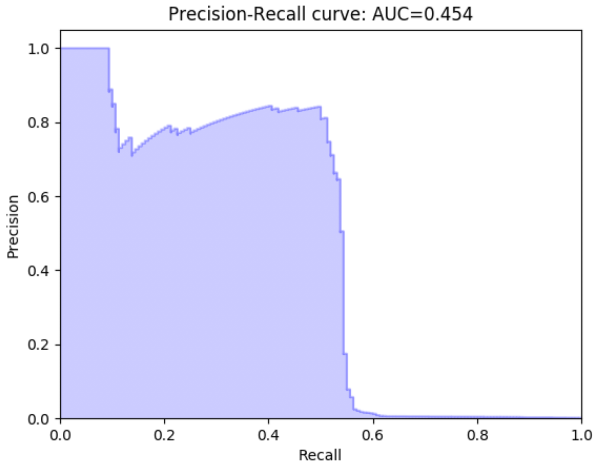

Figure 7: Logistic Regression AUC-PR Curve - No Resampling

respectively. Figure 8 shows the confusion matrix of the logistic regression model when random undersampling was performed on the dataset.

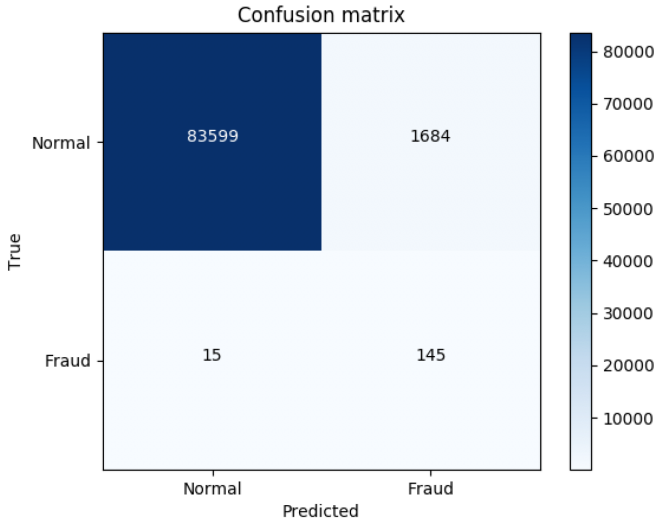

Figure 8: Logistic Regression Confusion Matrix - Random Undersampling

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9998	0.9819	0.9908
1	160	0.0857	0.9062	0.1566
avg/total	85443	0.9981	0.9817	0.9892

Table 3: Logistic Regression results for various threshold values for Random Undersampling

9.1.3 Tomek Links Removal

The best value of the hyperparameter "C" for the Logistic Regression, with a value of 0.1, was found using the GridSerachCV technique. Logistic Regression, in this case, performed very well on removing Tomek Links with precision, recall and f1 scores of 0.9991, 0.9995 and 0.9993 respectively as shown in Table 4. On the contrary, logistic regression performed really bad when dealing with fraudulent transactions, where the precision and recall were 0.6746 and 0.5312. In addition, Precision Recall area under the curve AUC-PR and AUC-ROC curve values, which are shown in Figure 13 and Figure 12 respectively, also didn't turn out to be very satisfactory. Figure 11 shows the confusion matrix of the logistic regression model when tomek links were removed from the dataset.

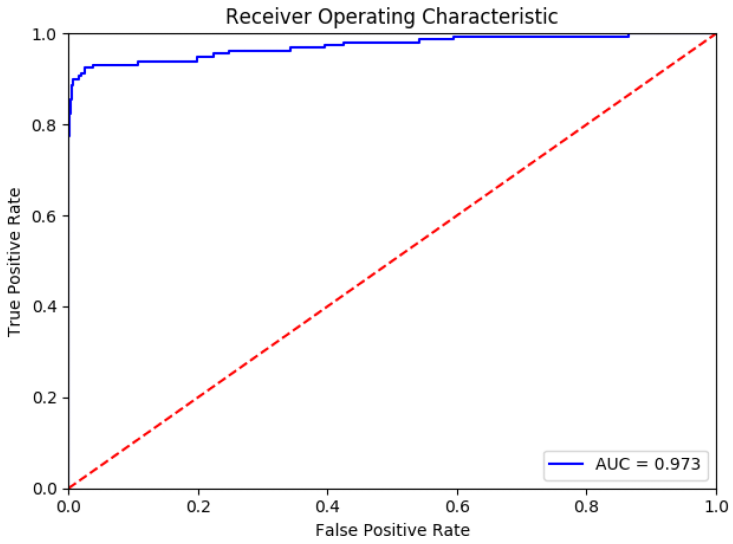

Figure 9: Logistic Regression AUC-ROC - Random Undersampling

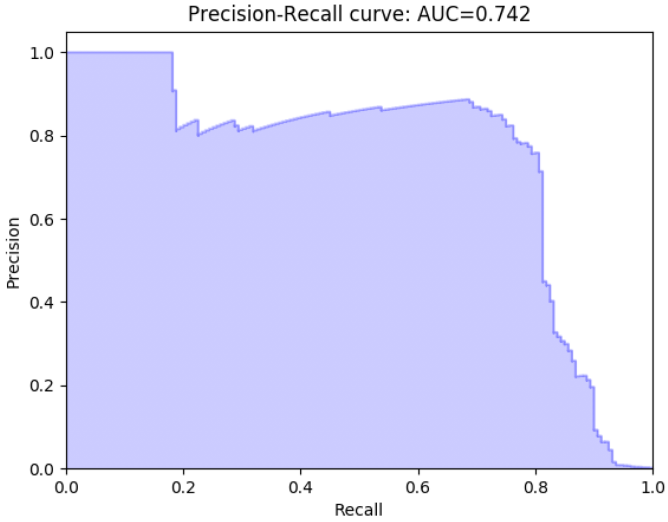

Figure 10: Logistic Regression AUC-PR Curve - Random Undersampling

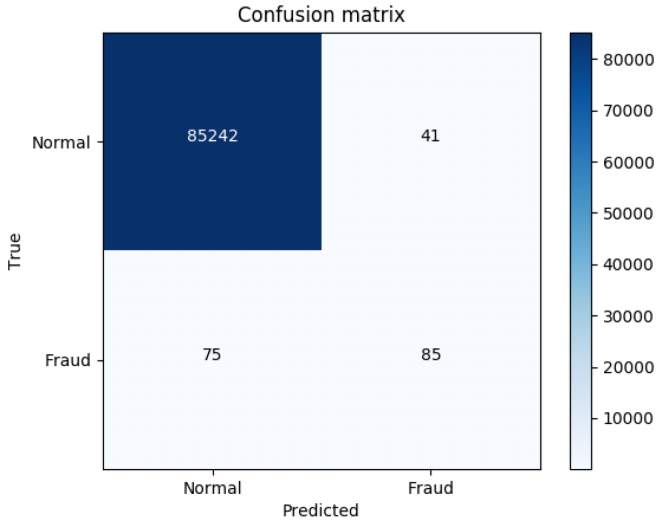

Figure 11: Logistic Regression Confusion Matrix - Tomek Links removal

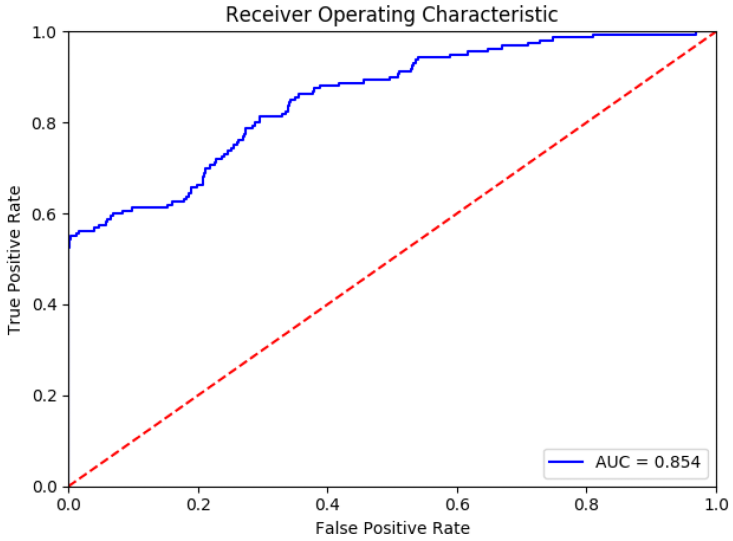

Figure 12: Logistic Regression AUC-ROC - Tomek Links removal

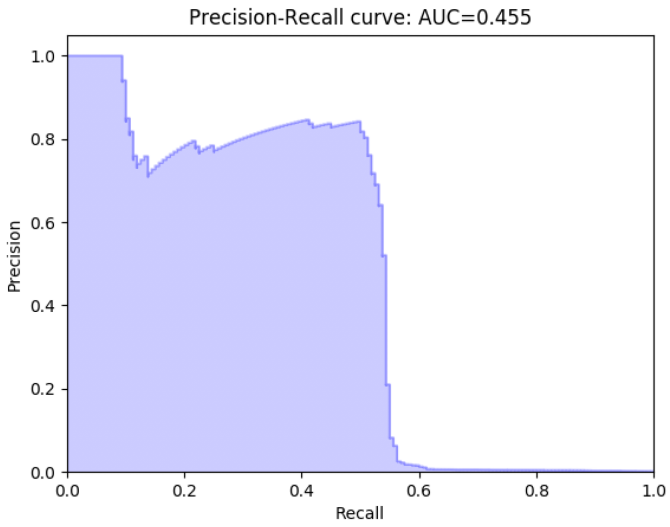

Figure 13: Logistic Regression AUC-PR Curve - Tomek Links removal

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9991	0.9995	0.9993
1	160	0.6746	0.5312	0.5944
avg/total	85443	0.9985	0.9986	0.9986

Table 4: Logistic Regression results for various threshold values for Tomek Links removal

9.1.4 Random Oversampling

The best value of the hyperparameter "C" for the Logistic Regression, with a value of 0.001, was found using the GridSerachCV technique. Logistic Regression, in this case, performed very well on random oversampling of data with precision, recall and f1 scores of 0.9998, 0.9874 and 0.9890 respectively as shown in Table 5. The performance was satisfactory when dealing with fraudulent transactions, where the precision and recall were 0.0724 and 0.9000. The Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 16 and Figure 15 respectively. Figure 14 shows the confusion matrix of the logistic regression model when random oversampling was performed on the dataset.

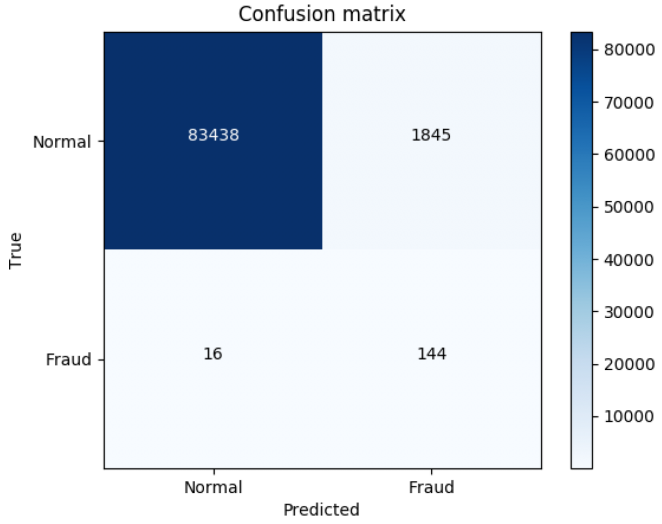

Figure 14: Logistic Regression Confusion Matrix - Random Oversampling

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9998	0.9874	0.9890
1	160	0.0724	0.9000	0.1340
avg/total	85443	0.9981	0.9872	0.9874

Table 5: Logistic Regression results for various threshold values for Random Oversampling

9.1.5 SMOTE

The best value of the hyperparameter "C" for the Logistic Regression, with a value of 0.01, was found using the GridSerachCV technique. Logistic Regression, in this case, performed very well on random oversampling of data with precision, recall and f1 scores of 0.9998, 0.9840 and 0.9918 respectively as shown in Table 6. The performance was satisfactory when dealing with fraudulent transactions, where the precision and recall were 0.0938 and 0.8812. The Precision Recall area

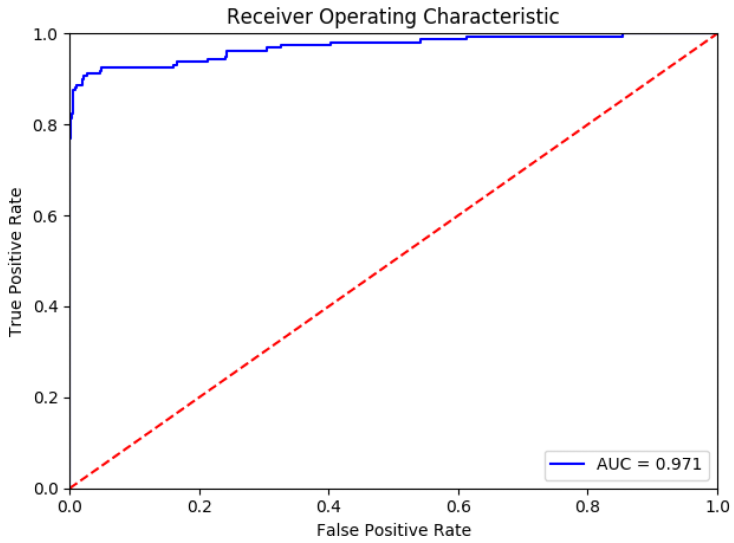

Figure 15: Logistic Regression AUC-ROC - Random Oversampling

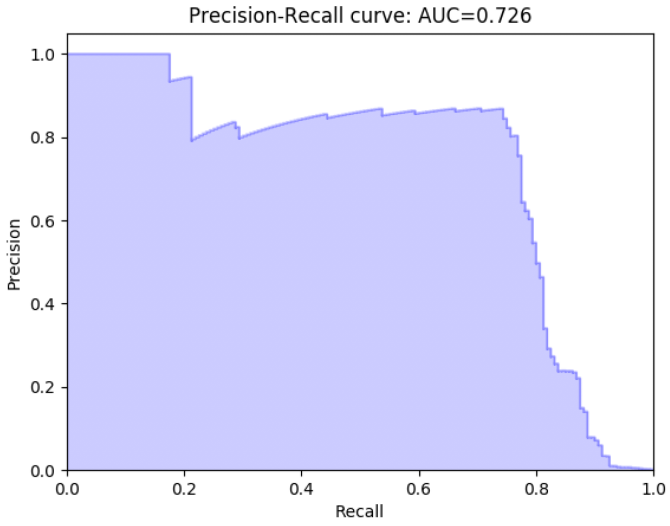

Figure 16: Logistic Regression AUC-PR Curve - Random Oversampling

under the curve AUC-PR and AUC-ROC curve values are shown in Figure 19 and Figure 18 respectively. Figure 17 shows the confusion matrix of the logistic regression model when SMOTE was performed on the dataset.

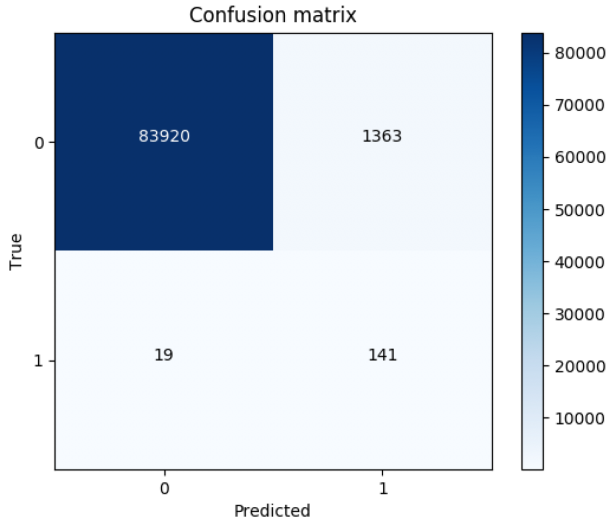

Figure 17: Logistic Regression Confusion Matrix - SMOTE

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9998	0.9840	0.9918
1	160	0.0938	0.8812	0.1695
avg/total	85443	0.9981	0.9838	0.9903

Table 6: Logistic Regression results for various threshold values for SMOTE

9.1.6 SMOTE & Tomek Links removal

The best value of the hyperparameter "C" for the Logistic Regression, with a value of 0.01, was found using the GridSerachCV technique. Logistic Regression, in this case, performed very well on random oversampling of data with precision, recall and f1 scores of 0.9998, 0.9803 and 0.9899 respectively as shown in Table7. The performance was satisfactory when dealing with fraudulent transactions, where the precision and recall were 0.0793 and 0.9062. The Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 22 and Figure 21 respectively. Figure 20 shows the confusion matrix of the logistic regression model when SMOTE was performed along with removing Tomek Links on the dataset.

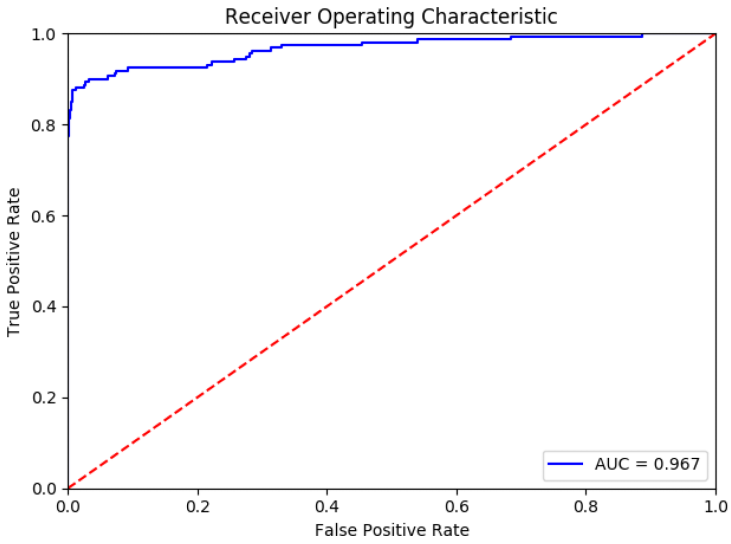

Figure 18: Logistic Regression AUC-ROC - SMOTE

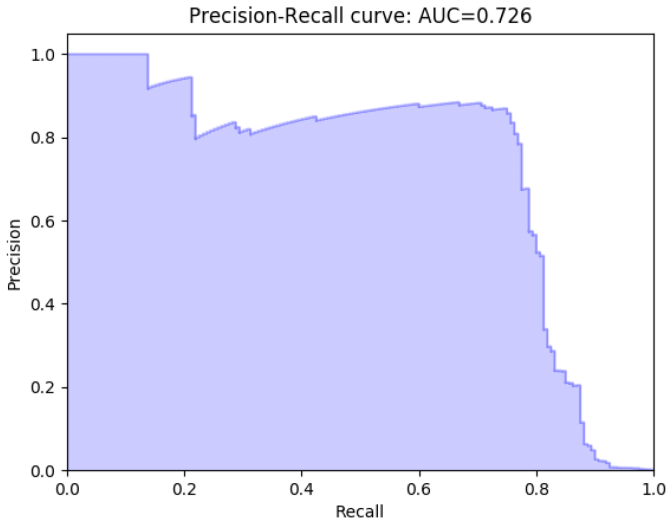

Figure 19: Logistic Regression AUC-PR Curve - SMOTE

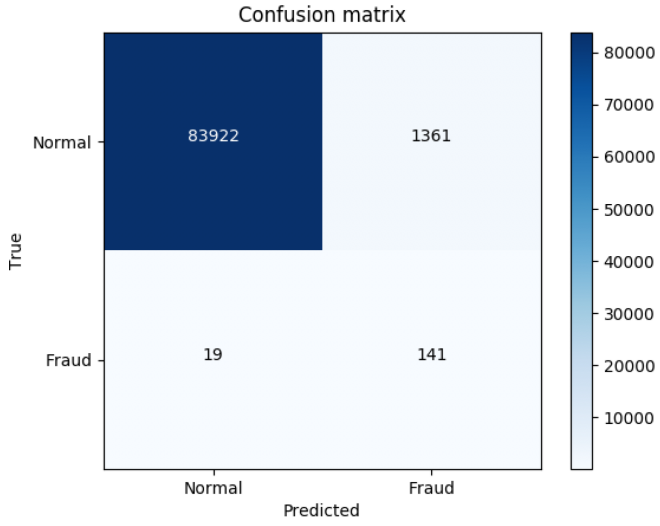

Figure 20: Logistic Regression Confusion Matrix - SMOTE and Tomek Links removal

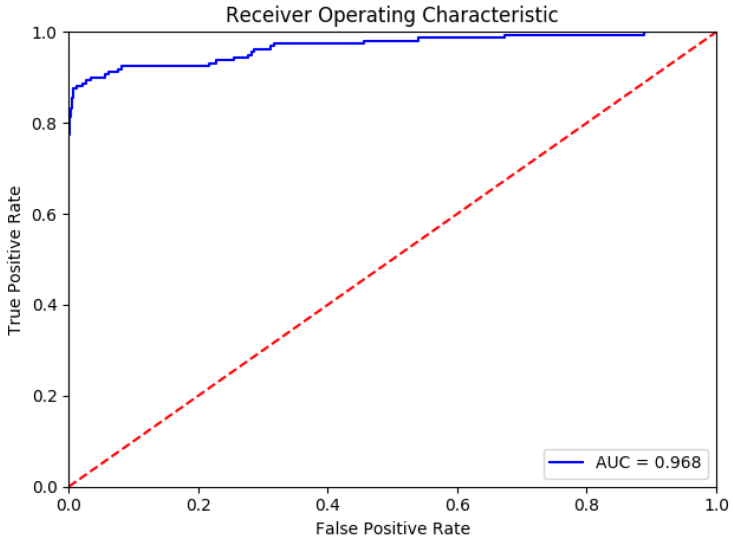

Figure 21: Logistic Regression AUC-ROC - SMOTE and Tomek Links removal

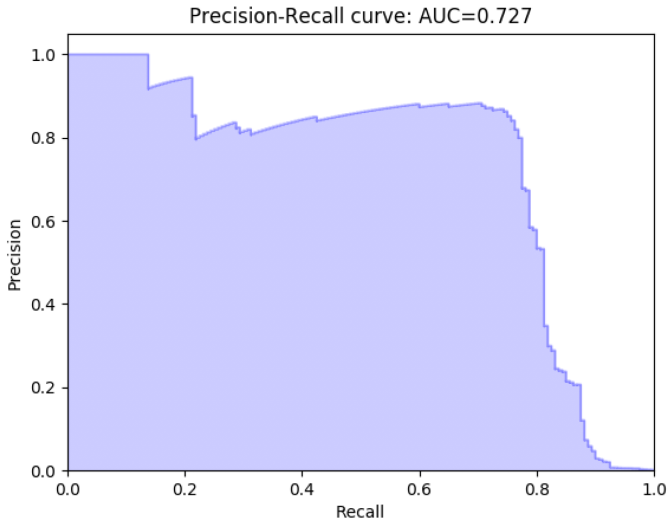

Figure 22: Logistic Regression AUC-PR Curve - SMOTE and Tomek Links removal

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9998	0.9803	0.9899
1	160	0.0793	0.9062	0.1458
avg/total	85443	0.9981	0.9801	0.9884

Table 7: Logistic Regression results for various threshold values for SMOTE and Tomek Links removal

9.2 Random Forest

9.2.1 No resampling

The best hyperparameters for the Random Forest were "total trees: 400, max features: auto" , which were found using the GridSerachCV technique. As discussed earlier, in case of an imbalanced dataset, the Random Forest or any other model will generally perform very well for predicting the normal or non-fraud transactions. The Random Forest performed very well for both normal and fraudulent classes with an overall F1 Score of 0.9991, 0.9995 and 0.9993 respectively as shown in Table 8.
On the contrary, Random Forest performed really bad when dealing with fraudulent transactions, where the precision and recall were 0.6774 and 0.5250. The Precision Recall area under the curve (AUC-PR) and (AUC-ROC) curve values are shown in Figure 25 and Figure 24 respectively. Figure 23 shows the confusion matrix of the Random Forest model when no resampling was performed on the dataset.

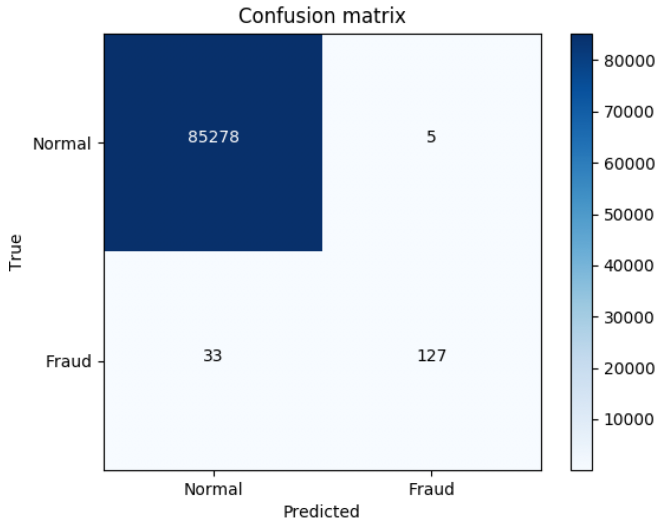

Figure 23: Random Forest Confusion Matrix - No Resampling

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0,9991	0,9995	0,9993
1	160	0,6774	0,5250	0,5915
avg/total	85443	0.9995	0.9996	0.9997

Table 8: Random Forest results for various threshold values for No Resampling

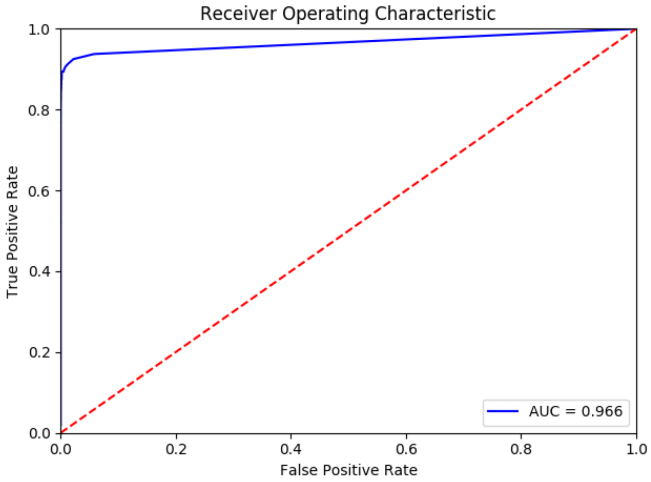

Figure 24: Random Forest AUC-ROC - No Resampling

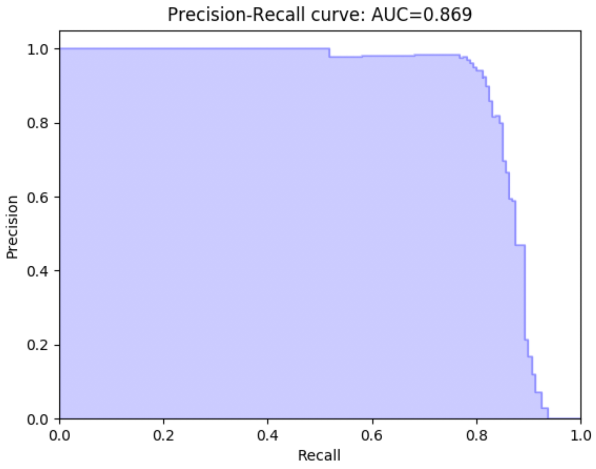

Figure 25: Random Forest AUC-PR Curve - No Resampling

9.2.2 Random Undersampling

The best hyperparameters for the Random Forest were "total trees: 400, max features: auto" , which were found using the GridSerachCV technique. Random Forest, in this case, performed very well on random undersampling of data with precision, recall and f1 scores of 0.9998, 0.9637 and 0.9814 respectively as shown in Table 9. On the contrary, Random Forest performed really bad when dealing with fraudulent transactions, where the precision and recall were 0.0453 and 0.9187. The Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 28 and Figure 27 respectively. Figure 26 shows the confusion matrix of the Random Forest model when random undersampling was performed on the dataset.

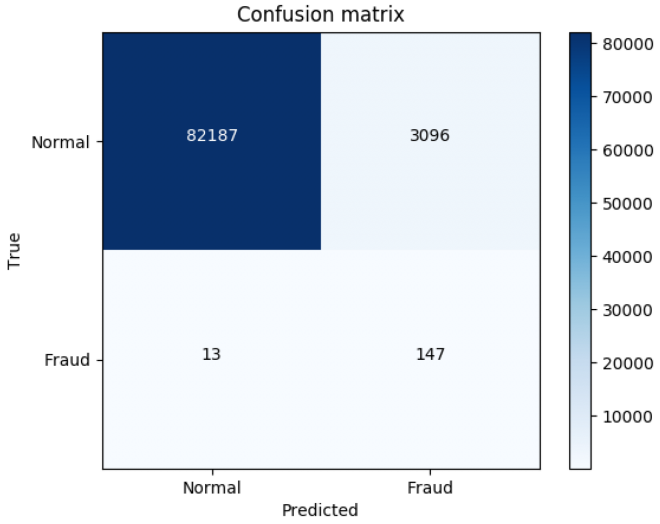

Figure 26: Random Forest Confusion Matrix - Random Undersampling

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9998	0.9637	0.9814
1	160	0.0453	0.9187	0.0864
avg/total	85443	0.9981	0.9636	0.9798

Table 9: Random Forest results for various threshold values for Random Undersampling

9.2.3 Tomek Links Removal

The best hyperparameters for the Random Forest were "total trees: 400, max features: auto" , which were found using the GridSerachCV technique. The Random Forest performed very well for both normal and fraudulent classes with an overall F1 Score of 0.84 as shown in Table 10. The Precision Recall Curve and the ROC-AUC Curve are shown in Figure 31 and Figure 30 re-

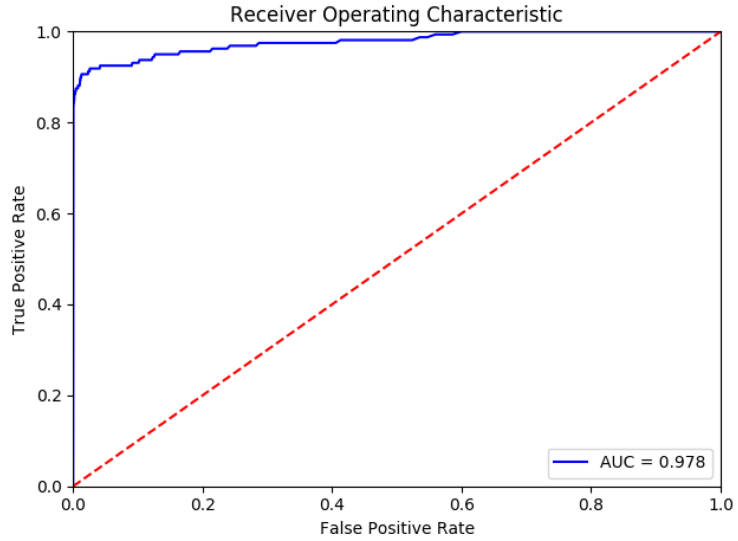

Figure 27: Random Forest AUC-ROC - Random Undersampling

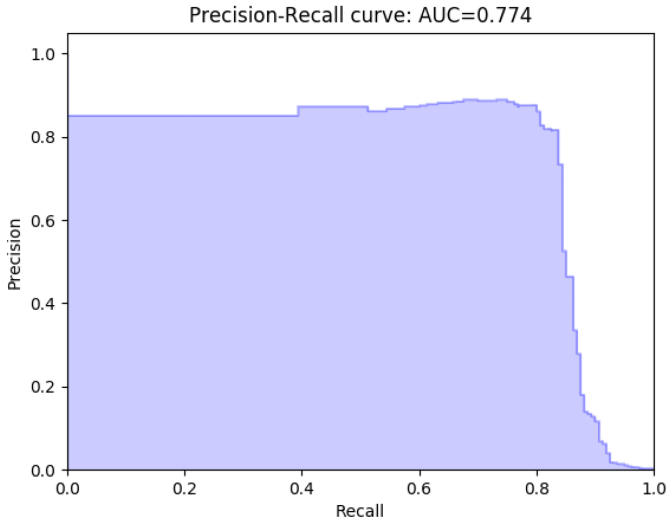

Figure 28: Random Forest AUC-PR Curve - Random Undersampling

spectively. Figure 29 shows the confusion matrix of the Random Forest model when tomek links were removed from the dataset.

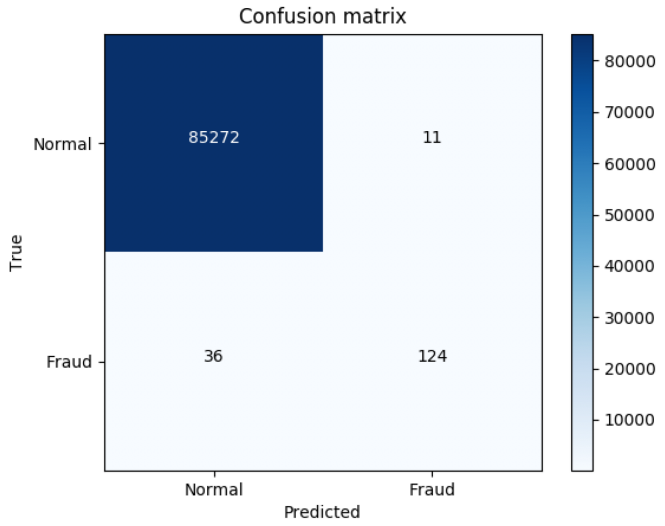

Figure 29: Random Forest Confusion Matrix - Tomek Links removal

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9996	0.9999	0.9997
1	160	0.9815	0.7750	0.8407
avg/total	85443	0.9994	0.9994	0.9994

Table 10: Random Forest results for various threshold values for Tomek Links removal

9.2.4 Random Oversampling

The best hyperparameters for the Random Forest were "Number of trees: 800, max features: auto" , which were found using the GridSerachCV technique. The Random Forest performed very well for both normal and fraudulent classes with an overall F1 Score of 0.85 as shown in Table 11. Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 34 and Figure 33 respectively. Figure 32 shows the confusion matrix of the Random Forest model when random oversampling was performed on the dataset.

9.2.5 SMOTE

The best hyperparameters for the Random Forest were "total trees: 600, max features: auto" , which were found using the GridSerachCV technique. This time the overall F1 Score was 0.76 along with the Precision score decreasing to 0.68 and the Recall of 0.87 which raised doubts about selecting this result. These results are shown in Table 12. Precision Recall area under the curve

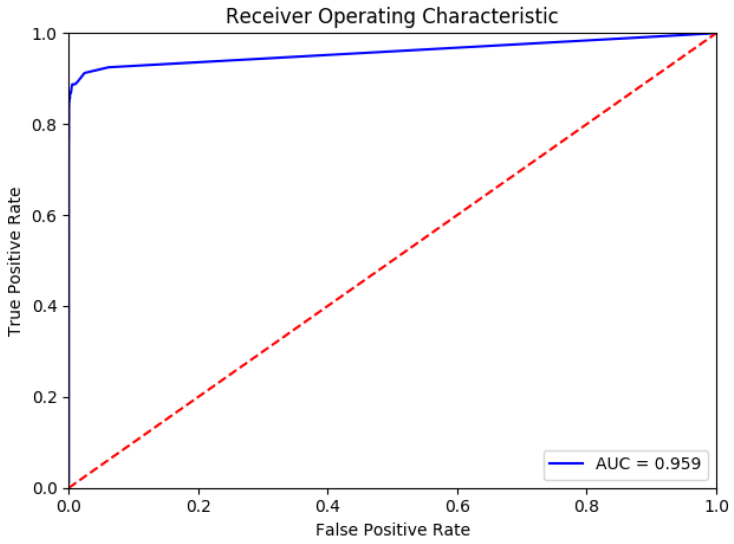

Figure 30: Random Forest AUC-ROC - Tomek Links removal

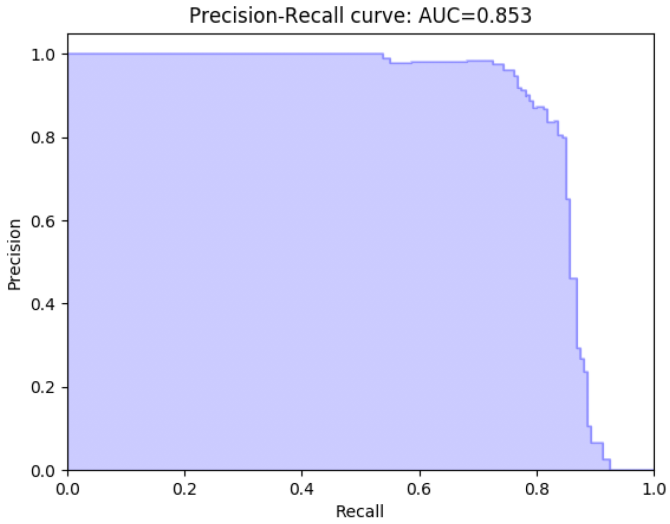

Figure 31: Random Forest AUC-PR Curve - Tomek Links removal

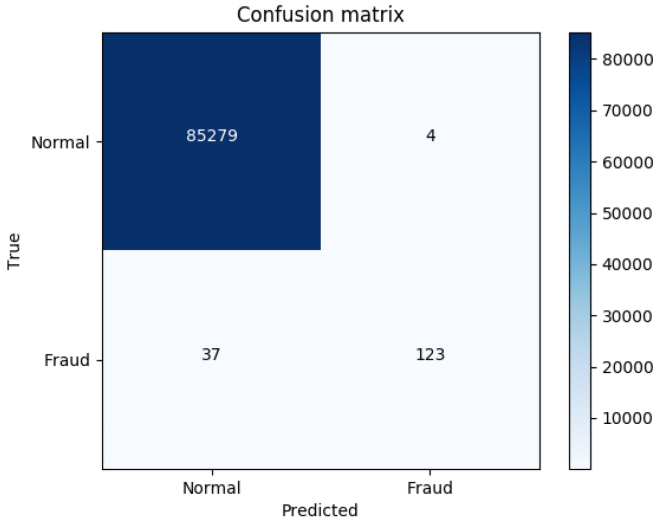

Figure 32: Random Forest Confusion Matrix - Random Oversampling

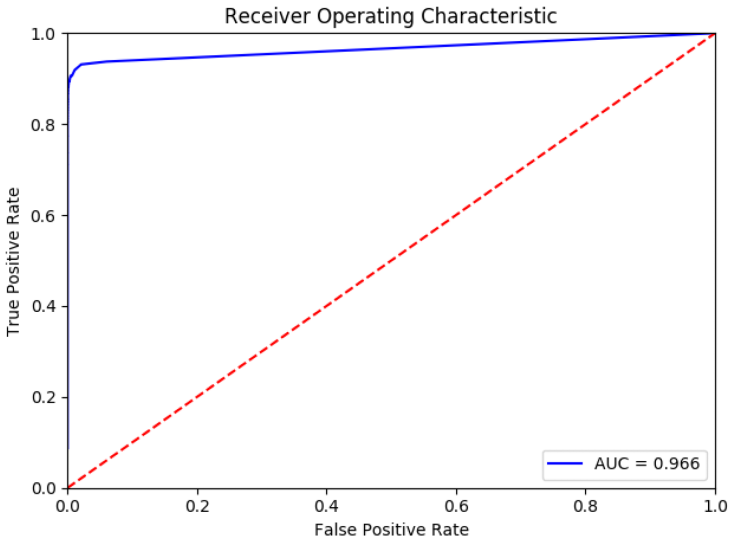

Figure 33: Random Forest AUC-ROC - Random Oversampling

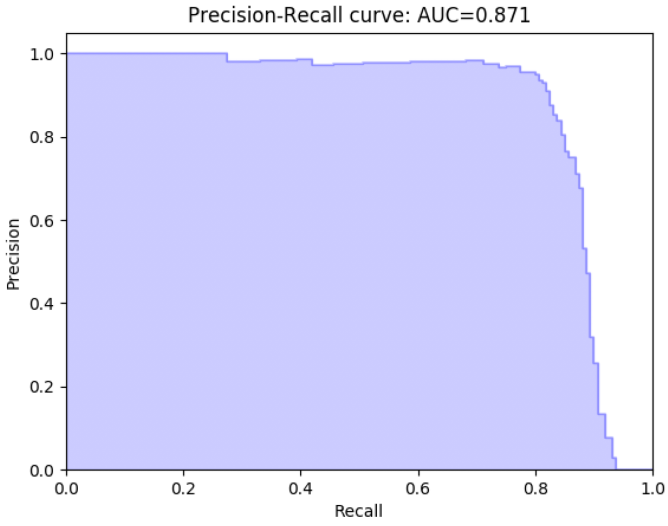

Figure 34: Random Forest AUC-PR Curve - Random Oversampling

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9996	1.000	0.9998
1	160	0.9685	0.7688	0.8571
avg/total	85443	0.9995	0.9995	0.9995

Table 11: Random Forest results for various threshold values for Random Oversampling

AUC-PR and AUC-ROC curve values are shown in Figure 37 and Figure 36 respectively. Figure 35 shows the confusion matrix of the Random Forest model when SMOTE was performed on the dataset.

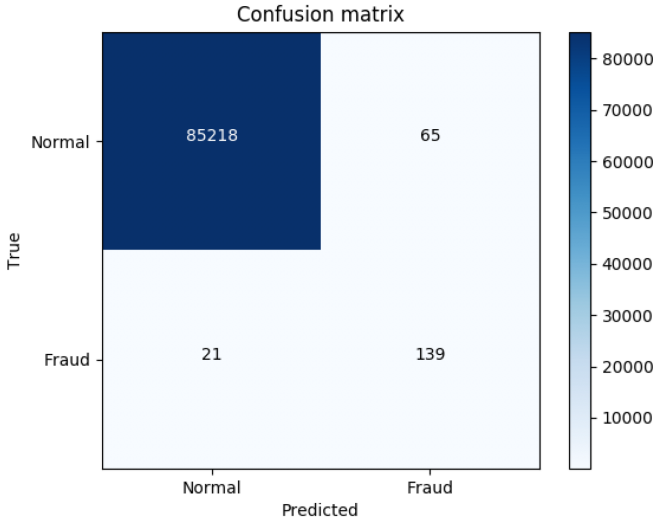

Figure 35: Random Forest Confusion Matrix - SMOTE

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9998	0.9840	0.9918
1	160	0.0938	0.8812	0.1695
avg/total	85443	0.9992	0.9990	0.9991

Table 12: Random Forest results for various threshold values for SMOTE

9.2.6 SMOTE & Tomek Links removal

The best hyperparameters for the Random Forest were "total trees: 400, max features: auto" , which were found using the GridSerachCV technique. This time the overall F1 Score was 0.84 along with a good balance of Precision score (0.84) and the Recall of 0.84 as shown in Table 13. The Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 40 and Figure 39 respectively. Figure 38 shows the confusion matrix of the Random Forest model when SMOTE was performed along with removing Tomek Links on the dataset.

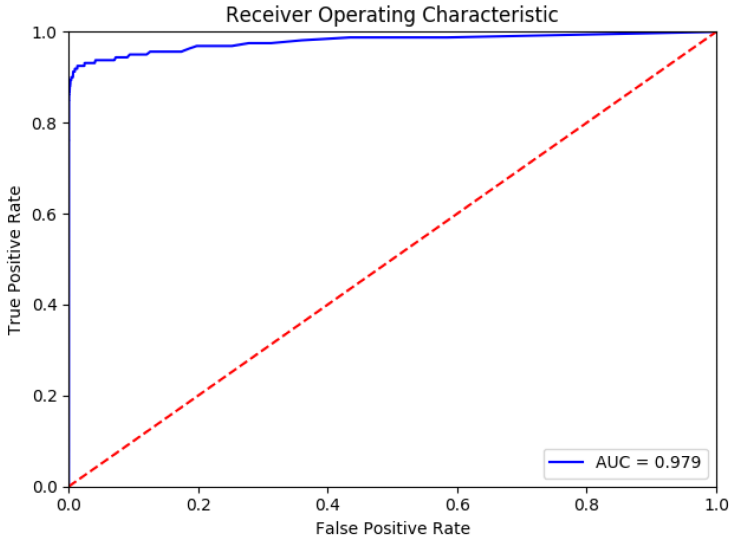

Figure 36: Random Forest AUC-ROC - SMOTE

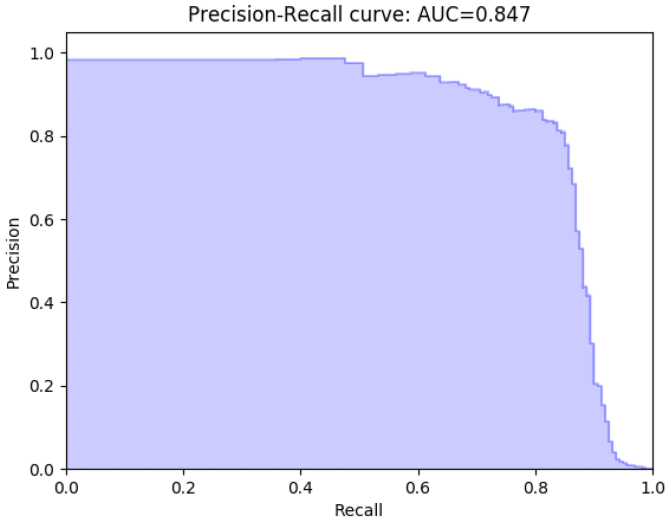

Figure 37: Random Forest AUC-PR Curve - SMOTE

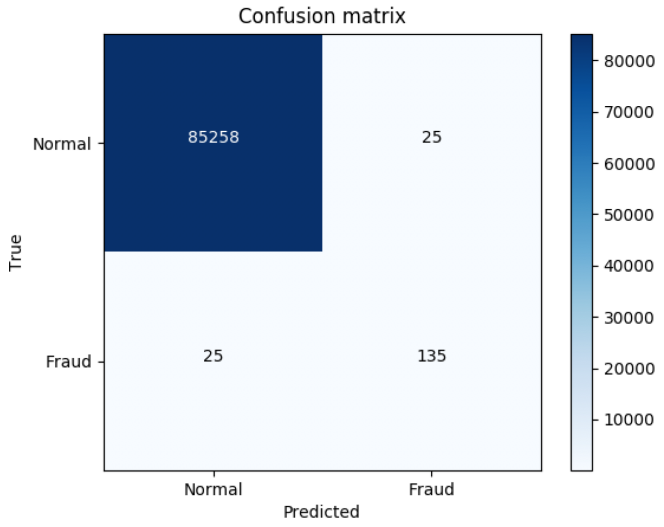

Figure 38: Random Forest Confusion Matrix - SMOTE and Tomek Links removal

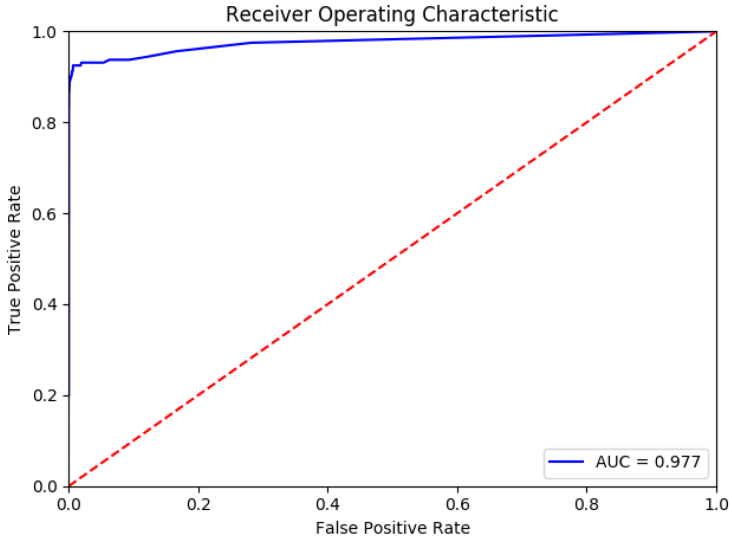

Figure 39: Random Forest AUC-ROC - SMOTE and Tomek Links removal

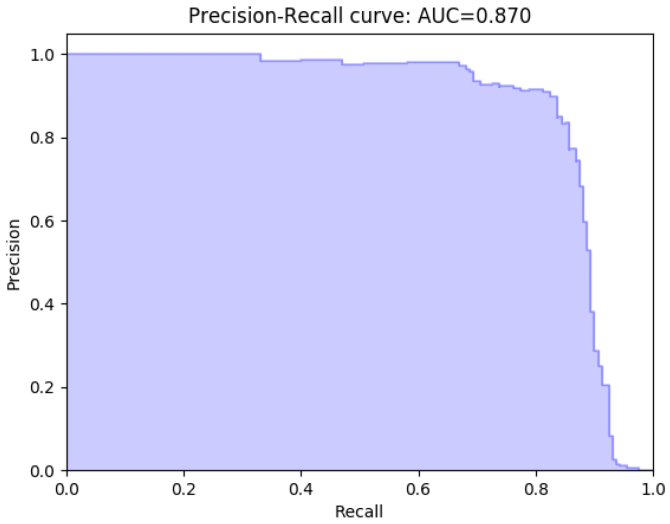

Figure 40: Random Forest AUC-PR Curve - SMOTE and Tomek Links removal

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9998	0.9803	0.9899
1	160	0.0793	0.9062	0.1458
avg/total	85443	0.9994	0.9994	0.9994

Table 13: Random Forest results for various threshold values for SMOTE and Tomek Links removal

9.3 XGBoost

9.3.1 No resampling

The best hyperparameters for the XGBoost were "Learning rate:0.1, total estimators:1000, maximum depth:6, minimum child weight:6, Sub-sample:0.85" , which were found using the GridSerachCV technique. XGBoost performed really well without resampling in classifying the normal transactions and obtained a Recall score of 0.73 which was later improved. The evaluation metrics of the XGBoost are shown in the Table 14.
Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 43 and Figure 42 respectively. Figure 41 shows the confusion matrix of the XGBoost model when no resampling was performed on the dataset.

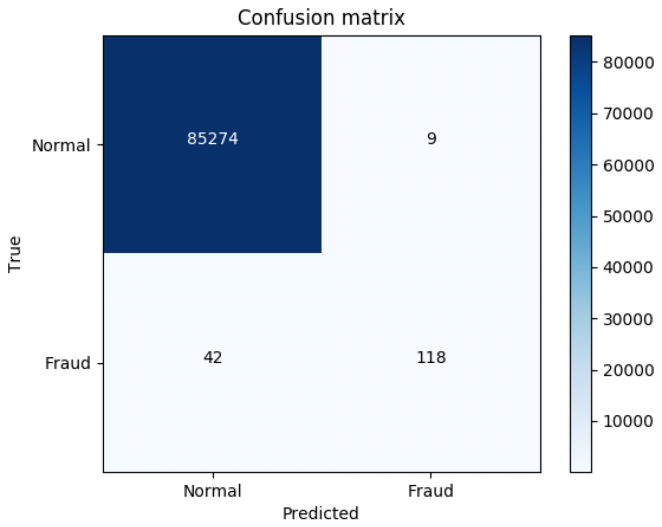

Figure 41: XGBoost Confusion Matrix - No Resampling

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9995	0.9999	0.9993
1	160	0.9291	0.7375	0.8223
avg/total	85443	0.9994	0.9994	0.9994

Table 14: XGBoost results for various threshold values for No Resampling

9.3.2 Random Undersampling

The best hyperparameters for the XGBoost were "Learning rate: 0.1, number of estimators:50, maximum depth:5, minimum child weight:4, subsample:0.85" , which were found using the GridSerachCV technique. XGBoost performed really bad with random undersampling in classifying

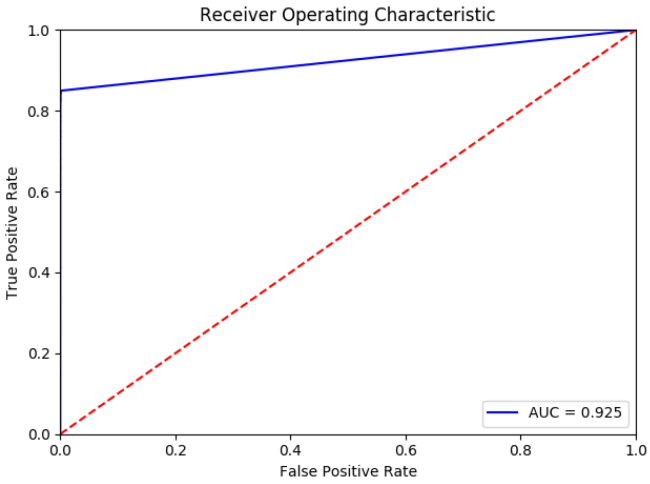

Figure 42: XGBoost AUC-ROC - No Resampling

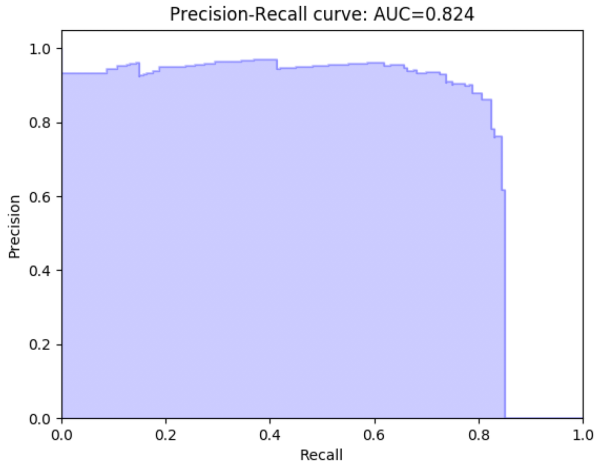

Figure 43: XGBoost AUC-PR Curve - No Resampling

the fraudulent transactions and obtained a Precision score of 0.037 which was later improved. The evaluation metrics of the XGBoost are shown in the Table 15.
Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 46 and Figure 45 respectively. Figure 44 shows the confusion matrix of the XGBoost model when random undersampling was performed on the dataset.

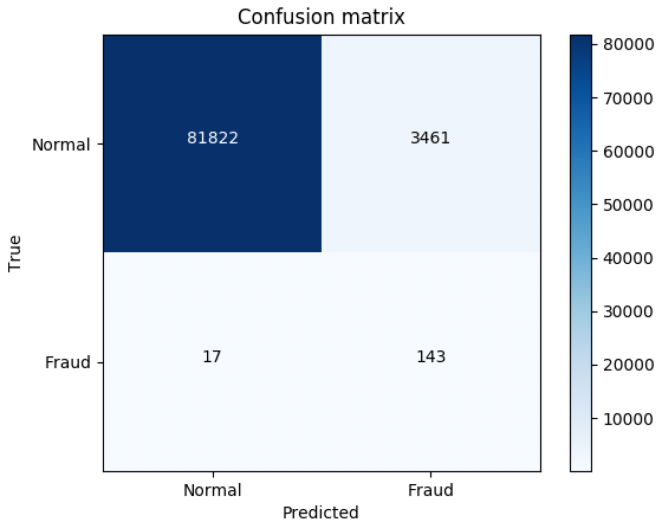

Figure 44: XGBoost Confusion Matrix - Random Undersampling

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9998	0.9594	0.9792
1	160	0.0397	0.8938	0.0760
avg/total	85443	0.9980	0.9593	0.9775

Table 15: XGBoost results for various threshold values for Random Undersampling

9.3.3 Tomek Links Removal

The best hyperparameters for the XGBoost were "Learning rate: 0.1, number of estimators:115, maximum depth:4, minimum child weight:5, subsample:0.65" , which were found using the GridSerachCV technique. XGBoost performed really bad with removal of Tomek Links in classifying the fraudulent transactions and obtained a Precision score of 0.87 which was a major improvement over random undersampling. The evaluation metrics of the XGBoost are shown in the Table 16.
Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 49 and Figure 48 respectively. Figure 47 shows the confusion matrix of the XGBoost model when Tomek links were removed from the dataset.

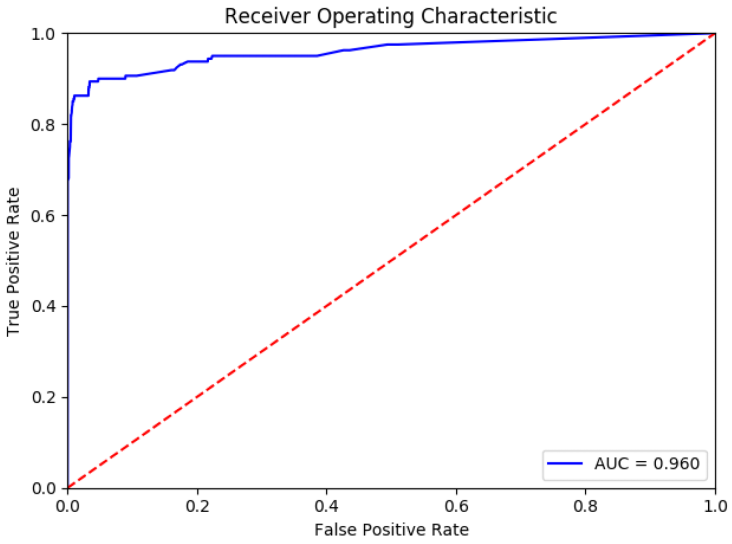

Figure 45: XGBoost AUC-ROC - Random Undersampling

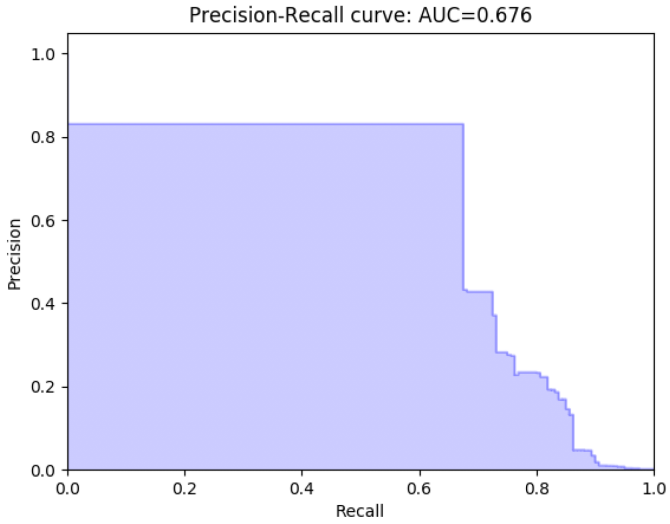

Figure 46: XGBoost AUC-PR Curve - Random Undersampling

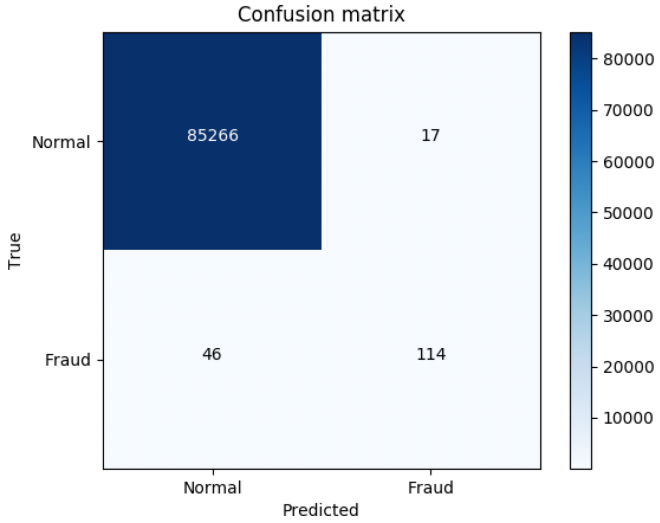

Figure 47: XGBoost Confusion Matrix - Tomek Links removal

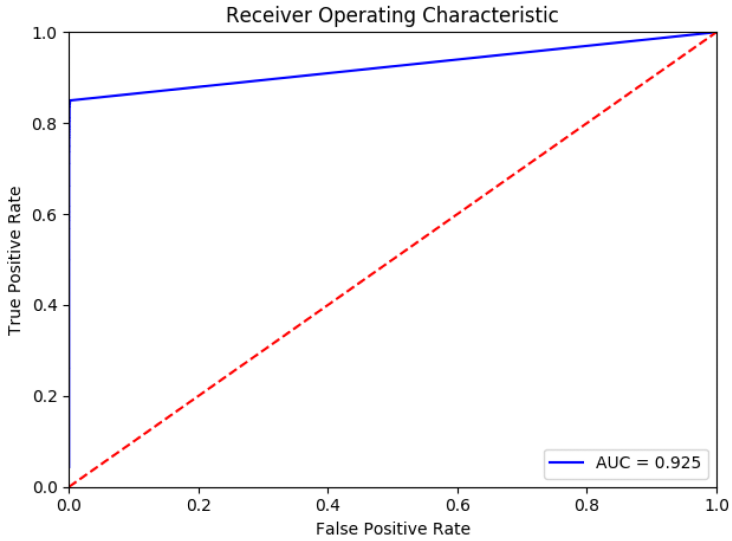

Figure 48: XGBoost AUC-ROC - Tomek Links removal

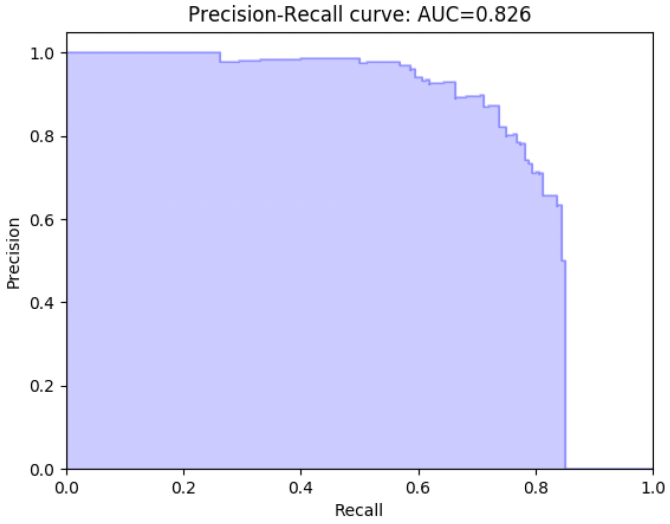

Figure 49: XGBoost AUC-PR Curve - Tomek Links removal

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9995	0.9998	0.9996
1	160	0.8702	0.7125	0.7835
avg/total	85443	0.9992	0.9993	0.9992

Table 16: XGBoost results for various threshold values for Tomek Links removal

9.3.4 Random Oversampling

The best hyperparameters for the XGBoost were "Learning rate: 0.1, number of estimators: 572, maximum depth: 5, minimum child weight: 6, Subsample: 0.75" , which were found using the GridSerachCV technique. XGBoost performed really well with random oversampling in classifying the fraudulent transactions and obtained a Precision score of 0.8 and Recall score of 0.83. The evaluation metrics of the XGBoost are shown in the Table 17.
Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 52 and Figure 51 respectively. Figure 50 shows the confusion matrix of the XGBoost model when random oversampling was performed on the dataset.

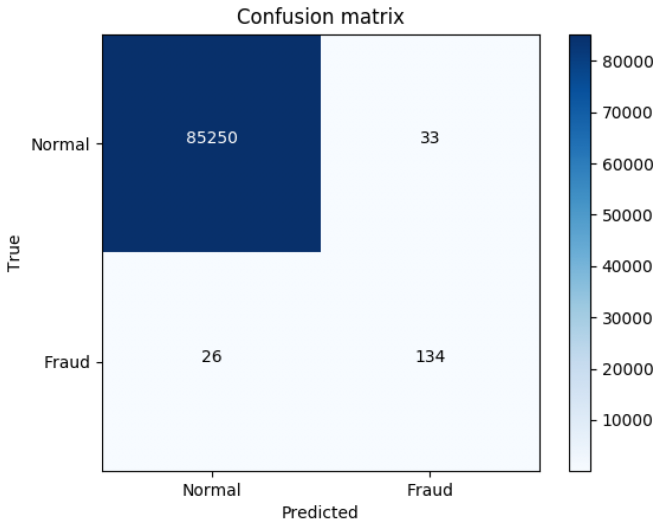

Figure 50: XGBoost Confusion Matrix - Random Oversampling

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9997	0.9996	0.9997
1	160	0.8024	0.8375	0.8196
avg/total	85443	0.9993	0.9993	0.9993

Table 17: XGBoost results for various threshold values for Random Oversampling

9.3.5 SMOTE

The best hyperparameters for the XGBoost were "Learning rate: 0.1, total estimators:870, maximum depth:10, minimum child weight:6, subsample:0.7" , which were found using the GridSerachCV technique. XGBoost performed really bad with SMOTE in classifying the fraudulent transactions and obtained a Precision score of 0.53 and Recall score of 0.87. The evaluation metrics of the XGBoost are shown in the Table 18.

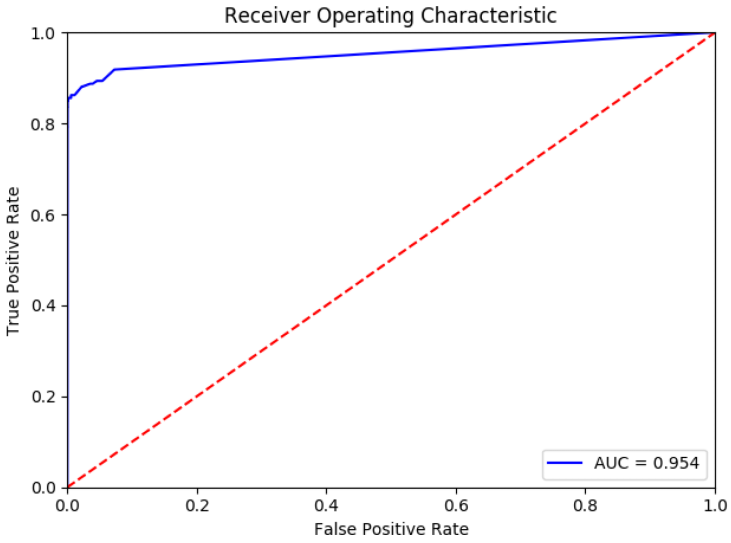

Figure 51: XGBoost AUC-ROC - Random Oversampling

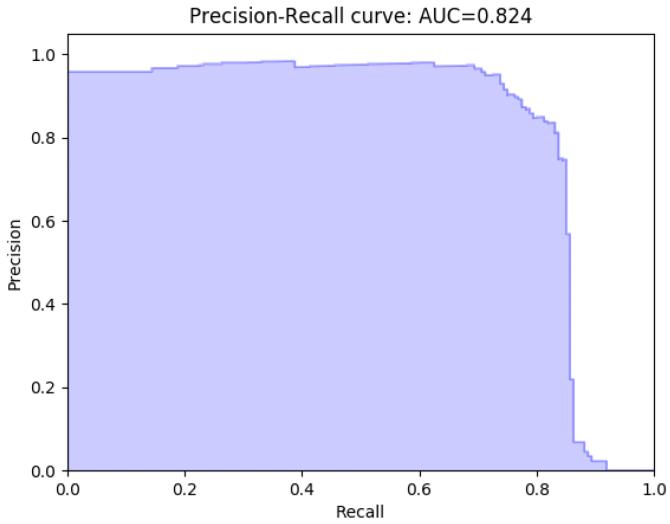

Figure 52: XGBoost AUC-PR Curve - Random Oversampling

Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 55 and Figure 54 respectively. Figure 53 shows the confusion matrix of the XGBoost model when SMOTE was performed on the dataset.

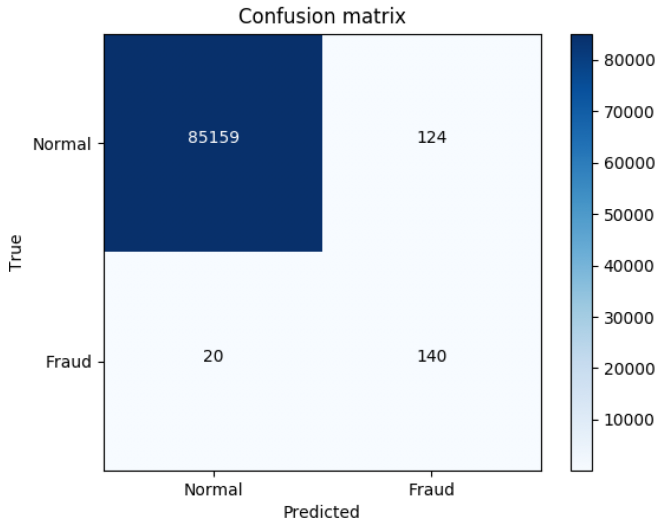

Figure 53: XGBoost Confusion Matrix - SMOTE

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9998	0.9985	0.9992
1	160	0.5303	0.8750	0.6604
avg/total	85443	0.9989	0.9983	0.9985

Table 18: XGBoost results for various threshold values for SMOTE

9.3.6 SMOTE & Tomek Links removal

The best hyperparameters for the XGBoost were "Learning rate: 0.1, total estimators:600, maximum depth:7, minimum child weight:6, subsample:0.7" , which were found using the GridSearchCV technique. XGBoost, with SMOTE and removal of Tomek Links for classifying the fraudulent transactions, obtained a Precision score of 0.20 and Recall score of 0.88. The evaluation metrics of the XGBoost are shown in the Table 19.
Precision Recall area under the curve AUC-PR and AUC-ROC curve values are shown in Figure 58 and Figure 57 respectively. Figure 56 shows the confusion matrix of the XGBoost model when SMOTE was performed on the dataset.

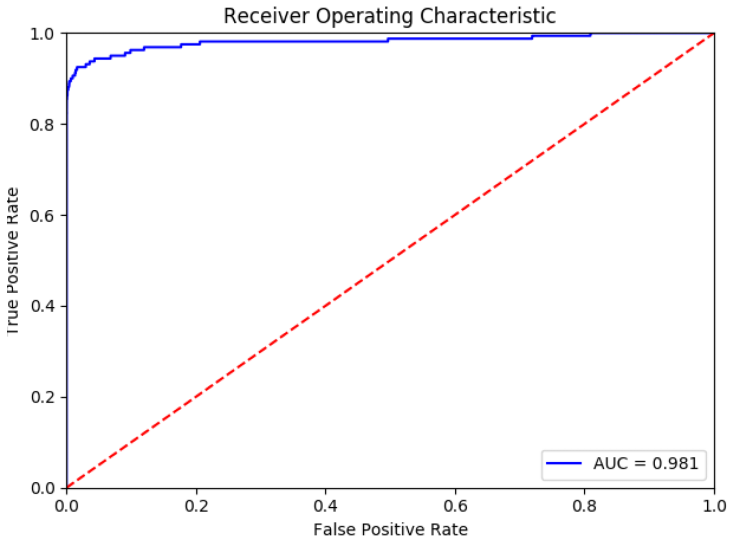

Figure 54: XGBoost AUC-ROC - SMOTE

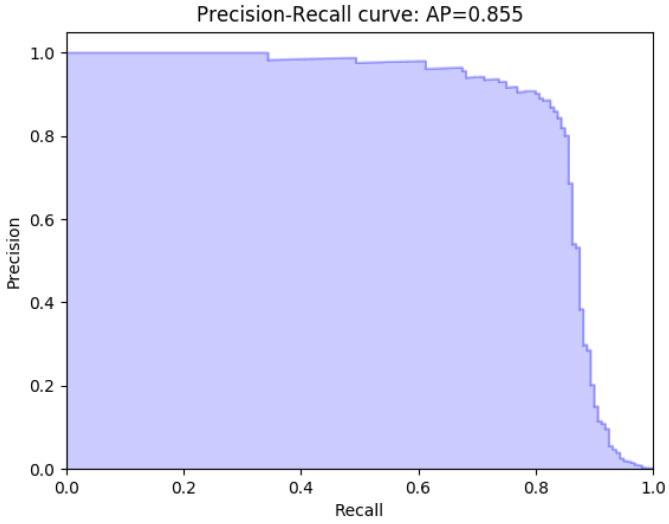

Figure 55: XGBoost AUC-PR Curve - SMOTE

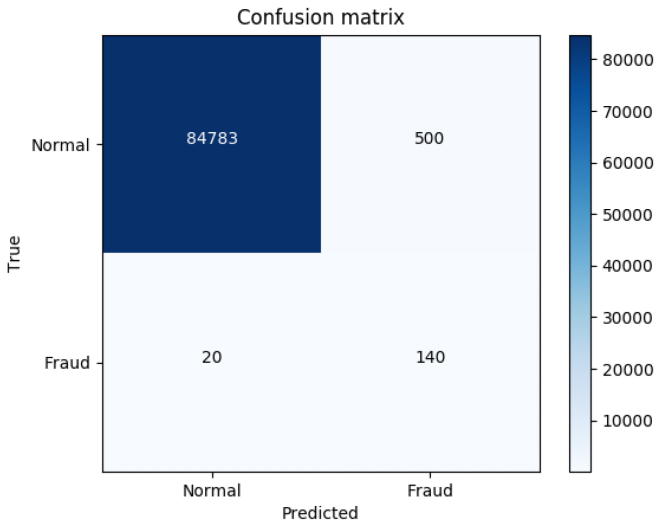

Figure 56: XGBoost Confusion Matrix - SMOTE and Tomek Links removal

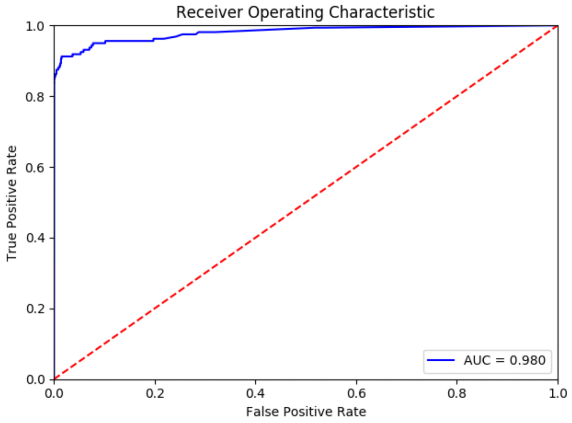

Figure 57: XGBoost AUC-ROC - SMOTE and Tomek Links removal

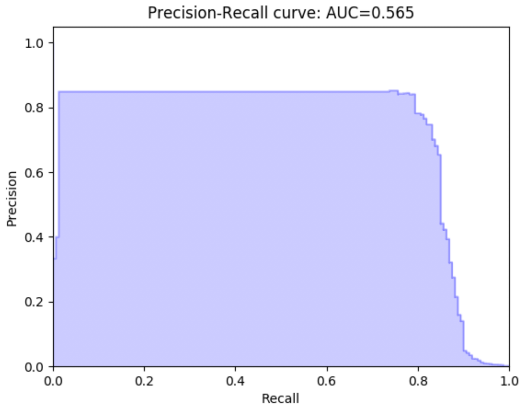

Figure 58: XGBoost AUC-PR Curve - SMOTE and Tomek Links removal

Class	No. of Samples	Precision	Recall	F1 Score
0	85283	0.9998	0.9934	0.9966
1	160	0.2023	0.8875	0.3295
avg/total	85443	0.9983	0.9932	0.9954

Table 19: XGBoost results for various threshold values for SMOTE and Tomek Links removal

9.4 Summary of results

The performance discussed above has been summarised in the following Table (See Table 20). The results Table 20 shows that the Logistic Regression performs the worst out of all the three

Classifier	Resampling Technique	Precision	Recall	F1 Score	PR AUC	ROC AUC
Logistic Regression	No Resampling	0.68	0.53	0.59	0.45	0.85
	Random Undersampling	0.09	0.91	0.16	0.72	0.97
	Tomek Links removal	0.67	0.53	0.59	0.45	0.85
	Random Oversampling	0.07	0.90	0.13	0.72	0.97
	SMOTE	0.09	0.88	0.17	0.72	0.96
	SMOTE & Tomek Links removal	0.08	0.91	0.15	0.72	0.97
Random Forest	No Resampling	0.96	0.79	0.87	0.86	0.96
	Random Undersampling	0.05	0.92	0.09	0.77	0.97
	Tomek Links removal	0.92	0.78	0.84	0.85	0.95
	Random Oversampling	0.97	0.77	0.86	0.87	0.96
	SMOTE	0.68	0.87	0.76	0.84	0.97
	SMOTE & Tomek Links removal	0.84	0.84	0.84	0.87	0.97
XGBoost	No Resampling	0.93	0.74	0.82	0.82	0.92
	Random Undersampling	0.04	0.89	0.08	0.67	0.96
	Tomek Links removal	0.87	0.71	0.78	0.82	0.95
	Random Oversampling	0.80	0.84	0.82	0.82	0.95
	SMOTE	0.53	0.88	0.66	0.85	0.98
	SMOTE & Tomek Links removal	0.22	0.88	0.35	0.56	0.98

Table 20: Summary of the results

models taking their F1 scores into account. This shows that ensemble techniques perform better even when the dataset is highly unbalanced. When comparing all the models without resampling, Random Forest and XGBoost performed much better than Logistic Regression keeping in mind their overall F1 scores. In case of random undersampling, all the models had good Recall scores but the Precision scores were really bad. Now, considering only the emsemble models, the Precision and Recall scores without resampling and on removing Tomek Links remained almost the same. In case of random oversampling, both the ensemble methods performed quite well considering their F1 Scores. For SMOTE, both the ensemble methods had poor Precision scores but the Recall scores were good. In the hybrid setting of SMOTE and removal of Tomek links, the Precision and Recall scores for the Random Forest was quite balanced in comparison to that of XGBoost. Additionally, there is was a lot of difference observed between the AUC-ROC Curves for both the models.

10 Conclusion

The Credit Card Fraud Detection dataset used in this study, was an extremely unbalanced dataset with $284,804$ normal transactions and 492 fraudulent transactions, was a publicly available dataset provided by the Machine Learning group of Universit Libre de Bruxelles containing records of credit card transactions made by European cardholders which occurred between two days of September 2013.

Predictive models usually tend to be biased towards the majority class samples for an unbalanced dataset which results in misclassification of data. This problem was tackled in this thesis by using resampling the dataset using techniques like random undersampling, Tomek links removal, random oversampling, SMOTE and a hybrid approach using SMOTE and Tomek links removal together and providing this dataset to bagging and boosting approaches like RandomForest and XGBoost respectively. The Logistic Regression model was used to compare the results of these ensemble methods. The comparison results showed that the random forest performs the best for resampled data using SMOTE and Tomek links removal technique.
This study can be extended by using a learning approach that takes the costs of misclassification into account. For many financial institutions this cost can be in the form of a few thousand to billions of dollars. The other costs that could be considered are cost of contacting the cardholders and transaction analysis. A study of models that consider such costs can be a way to work futher and extend this thesis. The only hindrance could the the amount of data as such studies require huge amounts of data to be available for research purposes.

Bibliography

Saba Akram and Qurrat ul Ann. Newton raphson method. *International Journal of Scientific and Engineering Research*, page 5, 07 2015.

E. Aleskerov, B. Freisleben, and B. Rao. Cardwatch: a neural network based database mining system for credit card fraud detection. In *Proceedings of the IEEE/IAFE 1997 Computational Intelligence for Financial Engineering (CIFEr)*, pages 220–226, 1997. doi: 10.1109/CIFER.1997.618940.

Haider Allamy. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices*, 12 2014.

Fernando Pereira Alon Halevy, Peter Norvig. The unreasonable effectiveness of data, 2009. URL `http://static.googleusercontent.com/media/research.google.com/fr//pubs/archive/35179.pdf`.

John Awoyemi, Adebayo Adetunmbi, and Samuel Oluwadare. Credit card fraud detection using machine learning techniques: A comparative analysis. pages 1–9, 10 2017. doi: 10.1109/ICCNI.2017.8123782.

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*, 31:1 – 16, 2018. ISSN 1467-0895. doi: https://doi.org/10.1016/j.accinf.2018.03.004. URL `http://www.sciencedirect.com/science/article/pii/S146708951630077X`.

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 Explor. Newsl.*, 6(1):20–29, June 2004. ISSN 1931-0145. doi: 10.1145/1007730.1007735. URL `https://doi.org/10.1145/1007730.1007735`.

Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011. URL `http://arxiv.org/abs/1106.1813`.

L Breiman. Arcing classifiers. *Statistics Department*, pages 801–824, 1998.

L Breiman. Prediction games and arcing algorithms. *Neural Computation*, pages 1493–1517, 1999.

Leo Breiman. Bagging predictors. *Machine Learning*, pages 123–140, 1996.

P. Buehlmann and T. Hothorn. Boosting algorithms: regularization, prediction and model fitting (with discussion). *Statistical Science*, pages 477–505, 2007.

Peter Bühlmann. Bagging, boosting and ensemble methods. *Handbook of Computational Statistics*, 01 2012a. doi: 10.1007/978-3-642-21551-3_33.

Peter Bühlmann. Bagging, boosting and ensemble methods. *Handbook of Computational Statistics*, 01 2012b. doi: 10.1007/978-3-642-21551-3_33.

Peter Bühlmann and Bin Yu. Analyzing bagging. *Ann. Statist.*, 30(4):927–961, 08 2002. doi: 10.1214/aos/1031689014. URL https://doi.org/10.1214/aos/1031689014.

Fabrizio Carcillo, Andrea Dal Pozzolo, Yann - A ë l Le Borgne, Olivier Caelen, Yannis Mazzer, and Gianluca Bontempi. SCARFF: a scalable framework for streaming credit card fraud detection with spark. *CoRR*, abs/1709.08920, 2017. URL http://arxiv.org/abs/1709.08920.

Jennifer Chee. Pearson's product moment correlation: Sample analysis. *University of Hawaii at Mānoa School of Nursing*, 05 2015.

Tianqi Chen and Carlos Guestrin. Xgboost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Aug 2016. doi: 10.1145/2939672.2939785. URL http://dx.doi.org/10.1145/2939672.2939785.

Duncan Cramer. *Fundamental Statistics for Social Research.* Routledge, Routledge, London, 1998.

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 Systems with Applications*, 41:4915–4928, 08 2014. doi: 10.1016/j.eswa.2014.02.026.

Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection and concept-drift adaptation with delayed supervised information. 07 2015. doi: 10.1109/IJCNN.2015.7280527.

Yadolah Dodge. *The Concise Encyclopedia of Statistics.* Springer Science + Business Media, LLC, Heidelberger Platz 3, Berlin, BE 14197, 2008.

P. Domingos. Metacost: a general method for making classifiers cost-sensitive. In *KDD '99*, 1999.

Michel Failing and Jan Theeuwes. Exogenous visual orienting by reward. *Journal of vision*, 14, 05 2014. doi: 10.1167/14.5.6.

Y Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, pages 256–285, 1995.

Y Freund and R.E. Schapire. Experiments with a new boosting algorithm. *Machine Learning: Proc. Thirteenth International Conference*, pages 148–156, 1996.

FIS Global. The best fraud prevention strategies for ecommerce, 2019a. URL https://www.fisglobal.com/en/insights/merchant-solutions-worldpay/article/the-best-fraud-prevention-strategies-for-ecommerce.

FIS Global. 6 things to look for in a credit card fraud detection solution, 2019b. URL https://www.fisglobal.com/en/insights/merchant-solutions-worldpay/article/6-things-to-look-for-in-a-credit-card-fraud-detection-solution.

Robert C. Holte, Liane E. Acker, and Bruce W. Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'89, page 813–818, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

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

Gary King and Langche Zeng. Logistic regression in rare events data. *Society for Political Methodology*, pages 1–27, 02 2001.

Jason Klusowski. Sparse learning with cart, 06 2020.

Sotiris Kotsiantis and P. Pintelas. Combining bagging and boosting. *International Journal of Computational Intelligence*, 1:324–333, 01 2005.

Bertrand Lebichot, Fabian Braun, Olivier Caelen, and Marco Saerens. A graph-based, semi-supervised, credit card fraud detection system. volume 693, pages 721–733, 11 2017. ISBN 978-3-319-50900-6. doi: 10.1007/978-3-319-50901-3_57.

LexisNexis. 2018 true cost of fraudsm study for the retail sector, 2018. URL `https://risk.lexisnexis.com/insights-resources/research/2018-true-cost-of-fraud-study-for-the-retail-sector`.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018. URL `https://arxiv.org/pdf/1603.06560.pdf`.

Marcos López de Prado. Overfitting: Causes and solutions, 2020. URL `https://ssrn.com/abstract=3544431`.

H. B. Mann and D. R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Statist.*, 18(1):50–60, 03 1947. doi: 10.1214/aoms/1177730491. URL `https://doi.org/10.1214/aoms/1177730491`.

Jim Goes Marilyn K. Simon. Correlational research. *Dissertation and Scholarly Research: Recipes for Success*, 04 2011.

Eric Brill Michele Banko. Scaling to very very large corpora for natural language disambiguation, 2001. URL `http://disi.unitn.it/$~$bernardi/Courses/CompLing/Papers/banko-brill-acl2001.pdf`.

Jacob Montiel, Rory Mitchell, Eibe Frank, Bernhard Pfahringer, Talel Abdessalem, and Albert Bifet. Adaptive xgboost for evolving data streams, 2020.

A. D. Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166, 2015. doi: 10.1109/SSCI.2015.33.

Andrea Dal Pozzolo and G. Bontempi. Adaptive machine learning for credit card fraud detection. 2015.

Laura E. Raileanu and Kilian Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41:77–93, 2000.

Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets. *PLOS ONE*, 10(3):e0118432, March 2015. doi: 10.1371/journal.pone.0118432. URL `https://doi.org/10.1371/journal.pone.0118432`.

R.E Schapire. The strength of weak learnability. *Machine Learning*, pages 197–227, 1990.

R.E Schapire. *The boosting approach to machine learning: an overview.* Springer, Springer, 2002.

Samithamby Senthilnathan. Usefulness of correlation analysis. *SSRN Electronic Journal*, 07 2019. doi: 10.2139/ssrn.3416918.

Antonio Serrano-López, Emilio Olivas, José Martín-Guerrero, Rafael Magdalena, and Juan Gómez-Sanchís. Feature selection using roc curves on classification problems. *Proceedings of the International Joint Conference on Neural Networks*, pages 1–6, 07 2010. doi: 10.1109/IJCNN.2010.5596692.

Abhinav Srivastava, Amlan Kundu, Shamik Sural, and Arun Majumdar. Credit card fraud detection using hidden markov model. *Dependable and Secure Computing, IEEE Transactions on*, 5: 37 – 48, 02 2008. doi: 10.1109/TDSC.2007.70228.

Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, 1995.

Machine Learning Group ULB. Credit card fraud detection - anonymized credit card transactions labeled as fraudulent or genuine, 2018. URL `https://www.kaggle.com/mlg-ulb/creditcardfraud`.

Venelin Valkov. *Hands-On Machine Learning from Scratch - Develop a Deeper Understanding of Machine Learning Models by Implementing Them from Scratch in Python.* Leanpub, Victoria, British Columbia, Canada, 2019.

Richard Wheeler and Stuart Aitken. Multiple algorithms for fraud detection. *Knowledge-Based Systems*, 13:93–99, 04 2000. doi: 10.1016/S0950-7051(00)00050-2.