Data Balancing for Credit Card Fraud Detection using Complementary Neural Networks and SMOTE Algorithm

by

Vrushal Shah

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science (MSc.) in Computational Sciences

The Faculty of Graduate Studies
Laurentian University
Sudbury, Ontario, Canada

THESIS DEFENCE COMMITTEE/COMITÉ DE SOUTENANCE DE THÈSE
Laurentian Université/Université Laurentienne
Faculty of Graduate Studies/Faculté des études supérieures

Title of Thesis Titre de la thèse	Data Balancing for Credit Card Fraud Detection using Complementary Neural Networks and SMOTE Algorithm
Name of Candidate Nom du candidat	Shah, Vrushal
Degree Diplôme	Master Science
Department/Program Département/Programme	Computational Sciences
Date of Defence Date de la soutenance	August 26, 2020

APPROVED/APPROUVÉ

Thesis Examiners/Examinateurs de thèse:

Dr. Kalpdrum Passi
(Supervisor/Directeur(trice) de thèse)

Dr. Ratvinder Grewal
(Committee member/Membre du comité)

Dr. Oumar Gueye
(Committee member/Membre du comité)

Approved for the Faculty of Graduate Studies
Approuvé pour la Faculté des études supérieures
Dr. David Lesbarrères
Monsieur David Lesbarrères

Dr. Nirbhay Chaubey Dean, Faculty of Graduate Studies
(External Examiner/Examinateur externe)

Doyen, Faculté des études supérieures

ACCESSIBILITY CLAUSE AND PERMISSION TO USE

Abstract

This Research presents an innovative approach towards detecting fraudulent credit card transactions. A commonly prevailing yet dominant problem faced in detection of fraudulent credit card transactions is the scarce occurrence of such fraudulent transactions with respect to legitimate (authorized) transactions. Therefore, any data that is recorded will always have a stark imbalance in the number of minority (fraudulent) and majority (legitimate) class samples. This imbalanced distribution of the training data among classes makes it hard for any learning algorithm to learn the features of the minority class. In this thesis work, we analyze the impact of applying class-balancing techniques on the training data namely oversampling (using SMOTE algorithm) for minority class and under sampling (using CMTNN) for majority class. The usage of most popular classification algorithms such as Artificial Neural Network (ANN), Support Vector Machine (SVM), Extreme Gradient Boosting (XGB), Logistic Regression (LR), Random Forest (RF) are processed on balanced data and which results to quantify the performance improvement provided by our approach. The experiments show that the hybrid approach which integrates Complementary Neural Network and Synthetic Minority Oversampling Technique gives a Quantitative performance in terms of Accuracy of 99% and 99.7% of AUC with Random Forest Classification Algorithm compared to simple undersampling and oversampling.

Key–Words: Fraud Detection, Complementary Neural Network, SMOTE, oversampling, under sampling, Class Imbalance

Acknowledgments

I would like to thank my supervisor, Dr. Kalpdrum Passi, who helped and guided me step by step and imparted deep knowledge and was compassionate and kind with me throughout the completion of this project. I would also like to thank my family and friends who were always there to support me. I would not have been able to complete my work without their immense support.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

Digital payment has experienced revolutionary changes over the last decade and is expected to keep up that trend for many years to come. Payment systems (however primitive) have existed for a very long time in the past. Starting from the ancient barter systems for exchanging goods to the modern one-click payment systems, we have come a long way. But with significant disruptions arises the need for more significant regulations, especially when it is related to financial transactions. With the technological advancement leading in smarter payment systems, there is an imminent need for a fast-paced evolution of the security systems that regulate these payments. The need for safer and more secure transactions has been globally accepted and is an active area of research. For modern regulatory security frameworks, it is equally important that they be robust (to adapt to different payment processes) and scalable (to afford large quantities of transactions).

The need for such security systems is alleviated due to rising cases of online thefts related to fraudulent transactions, causing people to lose millions every year. The different types of online payment frauds include (but are not limited to):

1. Friendly Fraud
2. Triangulation
3. Clean fraud
4. Identity theft

These are discussed in the next section.

1.2 Online Payment Frauds

For the scope of this work, online payments can be broadly classified into "Card-present" and "Card-not-present" transactions. In the former type of payments, the user (or the authorized cardholder) has to present the card physically at the time of the purchase (such as in-store purchases) to complete their transactions. In contrast, in the latter type of payments, the authorized user only verifies the security credentials and can perform a successful transaction without physically presenting the card. The Most conventional ways that fraudsters use to deceive merchants and customers are:

1. Friendly Fraud

   In this type of fraud, a customer-first makes a digital purchase with their credit card and then communicates their credit card issuer to argue for the charges by behaving like they never made the payment or the payment was made, but they did not receive the product/service. Obviously, not all chargebacks are fake – ordinarily; these cases may be legitimate. But, with very little evidence to verify the story, it has become a popular method for fraudulent activities in the last few years. It makes direct misfortune vendors as well as gets them penalized by the card issuer.

2. Triangulation

   The Triangulation fraud method involves an unsuspecting customer, a fake online store, and a mechanism to steal data. In this scenario, once the customer has made a transaction of its purchase, then the fraudulent merchant immediately takes his card details and cancels the transaction. Since the transaction was never successful, most of such activities go unnoticed by automated systems. The fake merchant now has access to confidential card details of the consumer and may use it later.

3. Clean fraud

   As discussed earlier, Friendly Fraud takes cover behind fake identities, or stolen data, hackers or fraudsters that go for an open theft usually have a great source of knowledge about the cardholders and their credit card information, and they use this customer

information to fool the systems. In this type of fraud, the criminal can be able to steal all-important real data and information and uses it to make a purchase that looks legitimate.

4. Identity theft

   Another kind of online payment fraud that can be very basic is identity theft. In this type of fraud, the pretender secures the key details of the customer-holding credit card and its personally identifiable information and then uses it for fraudulent purchases on the Internet. What sets this type of fraud apart is that the fraudster does not even need the card credentials, but only personally identifiable information related to the consumer.

Now, while it can be challenging to tackle all the forms of fraudulent transactions with one single approach, we have made an effort to detect "Triangulation", "Friendly Fraud" and "Identity Theft" using the transaction data analysis.

We have focused on the transaction data of credit cards, which is readily available for our analysis. Credit Card frauds are among the most natural and most rapidly rising scams in the modern world. While they provide a seamless service of executing monetary transactions, it also makes the task of detecting a fraudulent transaction equally harder. E-commerce and other similar online transactions account for a vast portion (roughly 80 %) of such fraudulent transactions.

With the growing number of credit card frauds (Figure 1.1), researchers have shown increasing interest related to fraud detection using classical machine learning approaches [1], [5], [7] as well as newer AI-based algorithms [6].

Figure 1.1 Credit Card Fraud Reports in the United States [8]

But one of the biggest bottlenecks in solving the classification problem (fraudulent transaction vs. legitimate transaction) is the imbalance in the transaction data. Since it is evident that the number of genuine transactions is far more than the number of fraudulent transactions, it becomes tough for any learning algorithm to be trained on the features of the minority class samples (fraudulent credit card transactions).

Several other challenges associated with credit card detection are namely fraudulent behavior profile are dynamic, that is fraudulent transactions tend to look like legitimate ones; credit card transaction datasets are rarely available and highly imbalanced (or skewed); optimal feature (variables) selection for the models; suitable metric to evaluate the performance of techniques on skewed credit card fraud data.

1.3 Imbalance Problem

The dataset used in this study is sourced from the ULB (Université Libre de Bruxelles) Machine Learning Group, and a description is found in [31]. The dataset holds credit card transactions made by European cardholders in September 2013. This dataset presents transactions that occurred in two days, consisting of 284,807 transactions. The positive class (fraud cases) make up 0.172% of the transaction data. The dataset is highly unbalanced and skewed towards the positive class

It is essential to understand that since over 99% of the transactions are non-fraudulent, an algorithm that always predicts that the transaction is non-fraudulent would achieve an accuracy higher than 99%. Thus, the accuracy metric in itself can be highly misleading in such cases of an extremely skewed dataset. Thus, our focus throughout this study has been on improving higher-order metrics like "AUC (Area under ROC curve)," "AUPRC: Area under Precision-Recall Curve," "F1 score," etc.

1.4 Objective of the Study & Outline of the thesis

The main objective of this study is to present an information-theory centric approach for handling Imbalanced Data Distribution using an Intelligent Sampling Network (or Data Balancing Network) based on CMTNN (complementary neural networks) and SMOTE (Synthetic Minority Over-sampling Technique). Different versions of the Data Balancing Network were tried and analyzed to achieve optimum performance. Firstly, different versions of the Data Balancing Network were compared for "the extent of Data Compression" and "the improvement in the accuracy metrics." Secondly these results were used to select the best version of the Data Balancing Network, which was later fixed in the pipeline and experimented with different classification algorithms to understand its impact on the predictions.

Imbalanced Binary Classification is a problem domain that has a lot of extremely well-written literature and yet is an unsolved problem. Through this study, we intend to register our contribution as well for this domain. This active research area is also closely linked to a lot of

practical scenarios that have significant implications in our lives. For the scope of this study, we have chosen to focus on the problem of "Credit Card Fraud."

⇒ **Research Question (that our work aims to answer)**

1. How do we handle and make use of imbalanced data to the tune of 0.172% minority class samples?
2. Can we develop a domain-independent technique to reduce the extent of the imbalance in the dataset?
3. How well do standard performance metrics like “Accuracy” and “Loss” fare when evaluating the Imbalanced Binary Classification problem?
4. What existing metrics or any new metric is best suited for performance evaluation in Imbalanced Binary Classification?

It is extremely important to understand at this stage that our research is not aimed at introducing a novel FDS (Fraud Detection System) for credit card fraud detection. In fact, the primary aim of our work is not aimed at proposing a solution to the problem of "Credit Card Fraud Detection". The focus of this work is to present a novel "Data Balancing Network" architecture, which can easily be plugged into any Imbalanced Binary Classification problem. The said network will use Deep Learning (CMTNN) and Statistical Machine Learning (SMOTE) to balance out the number of samples for each of the classes without introducing redundant samples in the minority class or removing any crucially important samples from the majority class.

This thesis is organized in the following sequence:

- **Chapter 1**: An introduction of the problem that we aim to solve has been discussed, so that the reader is mindful of both the need and the benefits of our research work.

- **Chapter 2**: A literature review comparing existing state-of-the-art techniques is presented. The importance of our work in the context of the existing literature and their limitations is explained.

- **Chapter 3**: The dataset used in this study is explained and analyzed. And also data preprocessing is explained in this chapter

- **Chapter 4**: A detailed description and analysis of the Methods and Classifiers used are explained

- **Chapter 5**: This chapter includes compilation and interpretation of the results.

- **Chapter 6**: In the final chapter, we conclude our study along with suggestions for future work.

Chapter 2

Literature Review

In this chapter, we provide an account of the background literature related to the existing work in the domain of "Credit Card Fraud Detection," projecting a major focus on the handling of imbalanced transaction data.

In Chapter 1 we discussed a few reasons explaining why detecting fraud is such a difficult task. In a broad sense, the single biggest drawback (positive drawback) in fraud detection is the fact that the number of genuine cases always highly outbalance the number of fraud cases. This leaves very little scope for analyzing and extracting patterns from fraudulent transactions.

A few more fundamental challenges in detecting fraudulent transactions have been very well-drafted in [15] and [16]:

- Any authentic, real-world data will always have an extremely large number of genuine transaction samples while containing only a handful of fraudulent cases. This is probably because fraudulent transactions, although they have increased rapidly over time but are a case of an anomaly in the regular genuine transactions. To make it worse, not all of the transactional data can be made available publicly due to the user-sensitive nature of such data. This makes it extremely difficult to create and validate any kind of detection system. To tackle this issue, people have tried to use statistical techniques such as oversampling, under sampling, and other such data manipulative methods. While these techniques do prove to be useful in prediction for some Machine Learning algorithms, there is an inherent risk of losing the authenticity of the dataset after applying these techniques.

- Another important issue to highlight is the practical implementation side. In the domain of banking, fraudulent transactions must be detected in real-time such that it can be blocked before it has been executed. Thus, implying the need for a robust and accurate FDS (Fraudulent Detection systems) which is also capable of providing real-time

classification results (classifying between genuine and fraud). It also needs to be easily scalable in order for it to not become obsolete sooner than later.

Based on these characteristic challenges it can be concluded that modern Statistical Machine Learning approaches like Support Vector Machine [5], Logistics Regression and Artificial Neural Network [6] are the best options to provide a coherent and wholesome solution.

For a very long time, Banks and almost all financial institutions have addressed fraud detection by modeling it through rule-based systems. The primitive rule-based Fraud Detection System is not of much use today as it is neither scalable nor does it settle well with the imbalance observed in the dataset. The underlying flaw with such modeling is perhaps more evident now than ever. The banking institutions had not anticipated the violent growth and technological disruption that was about to come. With technological advancements in the banking industry, the nature of the transactions diversified and the payment services grew at an unprecedented level. The rule-based systems were not scalable and thus had to be replaced with newer machine-learning-based systems.

One of the most popular architectures for the Fraud Detection System was suggested by Bolten et al. [16]. Bolton defined an outlier as an instance (or a pattern) in a dataset that does not conform to the expected behavior. Outlier detection is also used by intrusion detection systems in the domain of network intrusion, where an intrusion detection system often uses signature/pattern-based or anomaly detection techniques to detect intrusions. While the idea was simple yet powerful, it still suffered an important drawback of being neither scalable nor robust to advancements and disruptions in the technology. Since then, there have been numerous un-supervised and supervised approaches [2], [9], [10], [11], [14] that have been able to solve the problem to some extent but many of them still lacked the balance of scalability and robustness.

Among un-supervised approaches, clustering [8] by Vaishali et al. was one of the earliest to have gained popularity but failed to keep up with the growing amount of transaction data as it became computation heavy and results deteriorated significantly with the growing number of transactions. The earliest supervised approaches as shown by Dhankhad in [5] include Support Vector Machine, Random Forest, Extreme Gradient Boosting (XGB), etc. have almost all shown considerable improvement over the pre-existing clustering approaches. Dhankhad in [5] also experimented with under sampling the majority class to show improvement in performance. It is

from this reference that we garner our motivation to first attack the class imbalance itself and analyze the extent of performance improvement.

For some context, if we look back to several years of development, we can easily notice that there have been strong shifts, both technological and behavioral, in banking technology and day-to-day banking transactions. With each wave of technological disruption, we observe newer and more enhanced payment structures, which effectively alter the characteristic properties of a transaction itself. Thus, in a sense, all newer transactions become an outlier to the existing corpus of genuine transactions. Thus, any method (algorithm, model) employed for fraud detection must also be resilient to the changing technological outlook.

It is also interesting to note that consumer behavioral patterns themselves also contain useful information that could be incorporated into the FDS for better and more robust detections. Thus, more advanced techniques like Hidden Markov Model, Artificial Neural Network [6], Support Vector Machine [5], etc. are better suited for handling such fraud detection tasks. Also, because there is often a lack of "fraud" labeled data available, generative methods are expected to be more robust for anomaly detection than discriminative methods. Discriminative methods try to optimize a decision rule that classifies data into categories. Such methods do not model the relationships between the data and the underlying system process. Generative methods, on the other hand, try to learn a model that describes the system process. Due to the class imbalance in the dataset, discriminative models fail to capture enough discriminating rules to be able to successfully classify the transaction samples.

To tackle this imbalance problem at the data layer itself so that it does not affect the subsequent learning model, researchers have tried a lot of data up-sampling and down-sampling approaches. A random sampling approach is also used in [9, 10] and reports experimental results indicating that 50:50 artificially distribution of fraud/non-fraud training data generate classifiers with the highest true positive rate and low false-positive rate also hinting that reducing the class imbalance can result in much better performance.

Furthermore, a single transaction often provides not enough information to detect fraud, and thus often more sophisticated aggregate measures are employed in a Fraud Detection System. In Figure 2.1, examples of two aggregated features for transaction T with different time windows are illustrated. For instance, these features could represent the number of performed transactions

for the bank account within the past day or two days, respectively, for features x and y. Then x is two because only transactions T4 and T5 are within the time window of feature x. Similarly, for y, which contains five transactions within the time window of feature y.

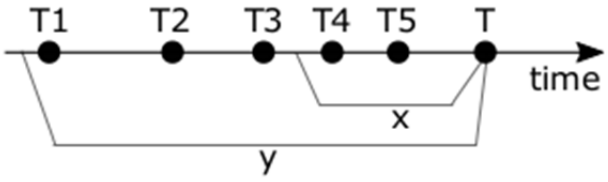

Figure 2.1 Temporal feature extraction from transaction data

Important features are often selected by the help of experts. These features are then used to define rules which can then be used by rule-based detection systems, as discussed in [17]. The downside of such a system (as already discussed above) is that it requires many interactions with experts of domain knowledge to maintain the rules. Also, with the growing scale of the number of transactions, it soon becomes extremely hard to be able to appropriately adapt a rule-based system. Nevertheless, people have used popular rule-based techniques like association-rule mining to perform detections [18]. Another rule-based technique that has often been used is Random Forests [7].

To handle the problem of class imbalance along with the exploding amount of transaction data, we have to tackle it via data processing before feeding the data to a learning model. We need an efficient way to under-sample the majority class and up-sample the minority class to establish sufficient class-balance while also keeping the authenticity of the data intact.

In [11], the authors use stratified sampling to under-sample the legitimate records to a meaningful number. It experiments on 50:50, 10:90 and 1:99 distributions of fraud to legitimate cases and reports that the 10:90 distribution has the best performance, perhaps owing to its authenticity to the real-world data distribution.

Thus, we acknowledge two challenges in an undersampling approach. First, simply applying a random sampling approach does not solve the problem, and secondly, the resulting balance among the majority and the minority class will also impact the resultant performance of the classifier. To solve these challenges, we apply a comprehensive solution. We choose

Complementary Neural Network [3] for under sampling because it follows a learning-based approach i.e., it learns the features of the majority class (legitimate transactions) and then removes only those samples from the training data that have relatively low information content and would essentially be redundant in training the classifier. This automatically takes care of the second challenge as the resulting balance among the classes would be an optimum balance i.e., one containing maximum information. This learning-based approach also makes sure that the authenticity of the data is not affected. It retains those majority class samples that contain most information and introduces non-redundant newer minority class samples. This compressed data is then used for training multiple classifiers to observe the performance improvement provided by this approach. In this thesis, we also experiment with different degrees of under sampling, the results of which will be analyzed later on in this thesis.

Chapter 3

Data Processing

In this chapter, our discussion will be centered on the dataset and its processing that we have used for our research work. Please note that the research work that has been presented and the proposed "Data Balancing network" are both data-independent. Thus, they can be easily plugged into any other Binary Classification problem. The choice of our data is completely driven by choice of our use case. To present our proposed pipeline and analyze its effectiveness in solving the imbalanced class distribution problem, we had opted for the "Credit Card Fraud" detection problem. It is for this reason that we have chosen the popular credit card transaction dataset curated and managed by the ULB Machine Learning Group.

Going ahead in this chapter, we will look further into the details of the dataset and the underlying imbalanced class distribution problem. We will also discuss more the existing solutions to this imbalance problem and contrast and compare them to our proposed "Data Balancing Network."

3.1 Dataset Description

The dataset used in our study is sourced from the ULB (Université Libre de Bruxelles) Machine Learning Group, and a description is found in [14]. The dataset contains credit card transactions made by European cardholders in September 2013. This dataset presents transactions that occurred in two days, consisting of 284,807 transactions. The positive class (fraud cases) make up 0.172% of the transaction data. The dataset is highly unbalanced and skewed towards the positive class (as shown in Figure 4.1). It contains only numerical (continuous) input variables, which are as a result of a Principal Component Analysis (PCA) feature selection transformation resulting in 28 principal components. Thus, a total of 30 input features are utilized in this study. The details and background information of the features could not be presented due to confidentiality issues. The time feature contains the seconds elapsed between each transaction and the first transaction in the dataset. The 'amount' feature is the transaction amount. Feature

'class' is the target class for the binary classification, and it takes value 1 for positive case (fraud) and 0 for negative case (non-fraud).

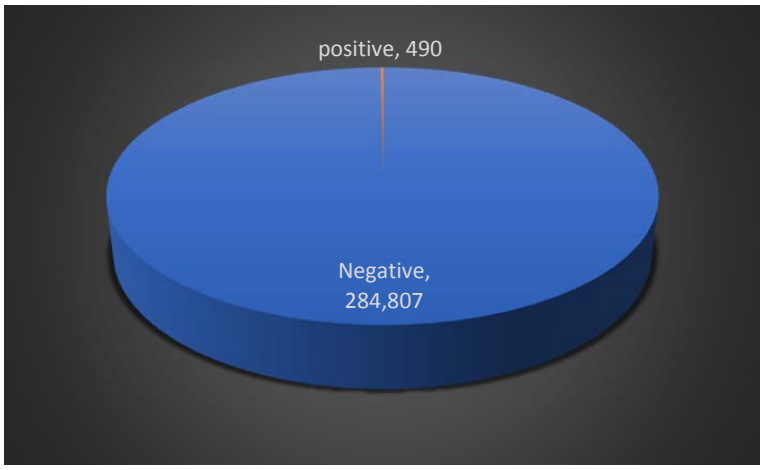

Figure 3.1 Class Imbalance (Showing Skewness)

It is important to understand that since over 99% of the transactions are non-fraudulent, an algorithm that always predicts that the transaction is non-fraudulent would achieve an accuracy higher than 99%. Thus, the accuracy metric in itself can be highly misleading in this case of the extremely skewed dataset. Thus, our focus throughout this study has been on improving higher-order metrics like "AUC: Area under ROC curve," "AUPRC: Area under Precision-Recall Curve", "F1 score" etc. While all of these metrics were employed during the analysis of Complementary Neural Network and Synthetic Minority Oversampling Technique performance, for visualization purposes and for clarity in the presentation of results, we will be considering only "AUC: Area under ROC curve" and "accuracy" metrics.

3.2 Class Imbalance problem

One of the toughest bottlenecks in credit card fraud detection arises due to the nature of the data. While there are millions of samples of fraudulent transactions, there are also billions of genuine transactions. Thus, imbalance of the class distribution is inherently present due to the nature of the domain of the problem. The dataset that we have considered contains only 0.172% of samples belonging to the minority class, as shown in Figure 3.1

It is also important to note that a lot of credit cards fraudulent transactions go unreported, and thus, it results in minority class samples being incorrectly labeled as majority class samples, which further increases the problem in classification. So, we are left with a dataset which, although huge in size, contains very few samples of the minority class (fraudulent transactions) while our main focus is directed at detecting the minority class samples correctly. Hence, any algorithm that we design must have a high "True Positive" fraction and low "False Negative" fraction.

This also perfectly resonates with the practical implications of the detection/classification problem at hand. It is safe to assume that any bank would be more interested in detecting the fraud transactions more accurately while compromising (classifying some "Genuine" transactions as "Fraud" transactions) on the correct classification of few genuine transactions.

3.3 Solutions to the imbalance problem

Learning from unbalanced datasets is a difficult task since most learning algorithms are not designed to cope with a large difference between the numbers of cases belonging to different classes [19]. There are several methods that deal with this problem, and we can easily distinguish between them based on the level that they operate on. Most of these techniques operate either on data level or on an algorithmic level. The authors in [6], [7] have presented a valuable resource on distinguishing and analyzing the data level techniques and algorithmic level techniques.

At the **data level**, the unbalanced strategies are used as a pre-processing step to rebalance the dataset or to remove the noise between the two classes, before any algorithm is applied. The aim of such methods is to analyze the underlying (imbalanced) data distribution and modify/process it to a desirable (balanced) data distribution. These methods are primarily focused on obtaining a reasonably well-balanced data distribution and then apply the classification algorithms as usual.

At the **algorithmic level**, algorithms are themselves adjusted to deal with the minority class detection. In this category, the data distribution is not the center of the focus, and thus no effects

are driven by the need to improve the data distribution itself. Instead, the focus is always to tune and modify the classification algorithm to obtain better classification scores. The classification algorithms are manipulated in a manner such that they learn to handle the stark skewness in the dataset as a part of their classification process.

Data level methods can be grouped into five main categories:

- Sampling,
- Ensemble,
- Cost-based,
- Distance-based
- Hybrid.

Within algorithmic methods instead, we can distinguish between:

- Classifiers that are specifically designed to deal with an unbalanced distribution
- Classifiers that minimize overall classification cost.

The latter is known in the literature as cost-sensitive classifiers [20]. Both data and algorithm level methods that are cost-sensitive target the unbalanced problem by using different misclassification costs for the minority and majority classes. At the data level, cost-based methods sample the data to reproduce the different costs associated with each class. Cost-sensitive classifiers instead directly minimize the costs by using cost specific loss function.

3.3.1 Our proposed solution

Standard data processing techniques like random under-sampling and replicative oversampling have existed among the research community for a long time and have also been successful in several instances. But in our particular case of fraud detection, the aim is not primarily to ease computation (this is usually where normally random sampling proves effective), and replicative oversampling can easily result in overfitting of our classifier. Therefore, in our case, we need a data-centric or information-theory centric approach to under-sampling and oversampling.

The proposed solution uses CMTNN for under-sampling the majority class while preserving training samples that provide crucial information for feature learning. For the oversampling part,

it uses the Synthetic Minority Oversampling Technique (SMOTE) to generate relevant samples for the minority class.

In the subsequent chapters, we will present the details of the research experiments that were carried out and also discuss in detail the underlying theory and motivation behind using the frameworks that were employed. We will look into the details of the architecture of the "Data Balancing Network" and also its building blocks "CMTNN: Complimentary Neural Networks" and "SMOTE: Synthetic Minority Over-Sampling Technique". We will also present a detailed account of the implementation of the above-mentioned two algorithms.

Chapter 4

Methods

4.1 Proposed Method

While we do not propose a novelty in the classification algorithm itself, we bring together different techniques of data analytics (Complementary Neural Network and Synthetic Minority Oversampling Technique) and Machine Learning (Support Vector Machine, Random Forest, Extreme Gradient Boosting, Logistics Regression, and Artificial Neural Network) in a systematic manner to tackle both data-level challenges and learning level challenges. As discussed in our experiments' credit card transaction data is highly skewed towards legitimate transactions as compared to fraudulent transactions.

The Flowchart of the overall work is described in Figure 4.1 which shows the flow of the data preprocessing followed by the balancing of the data by undersampling and oversampling using a hybrid approach of CMTNN and SMOTE which is followed by the classification process and to get the final output.

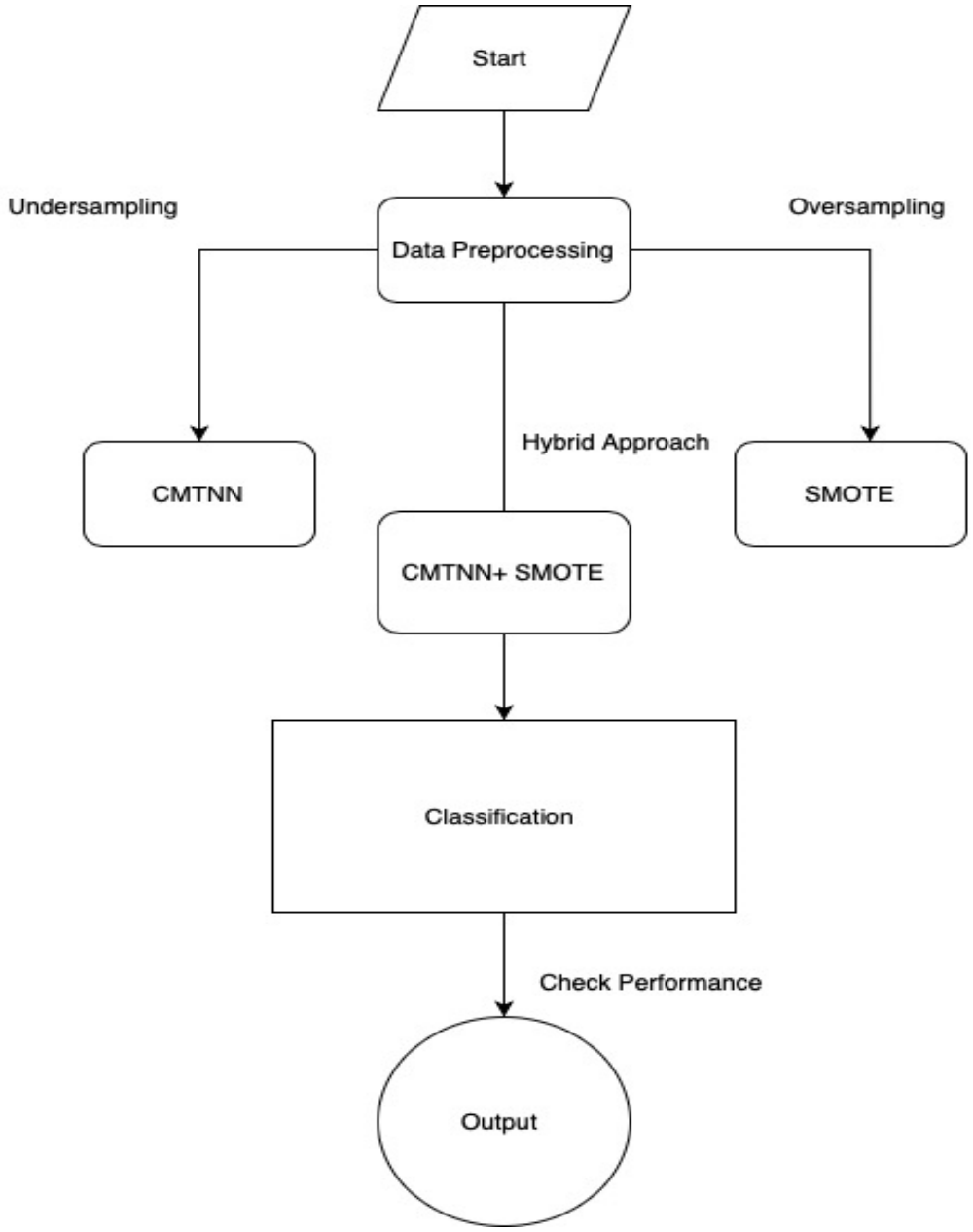

Figure 4.1: System Flowchart

In our proposed method, we have designed a two-block pipeline (Figure 4.2). The first stage deals with Data Balancing Network (balancing the skewness using under sampling and oversampling techniques), and the second stage deals with classification algorithms. The second stage is further divided into two sub-stages. The first sub-stage deals with classical machine learning algorithms, and the other sub-stage deals with advanced deep learning algorithms.

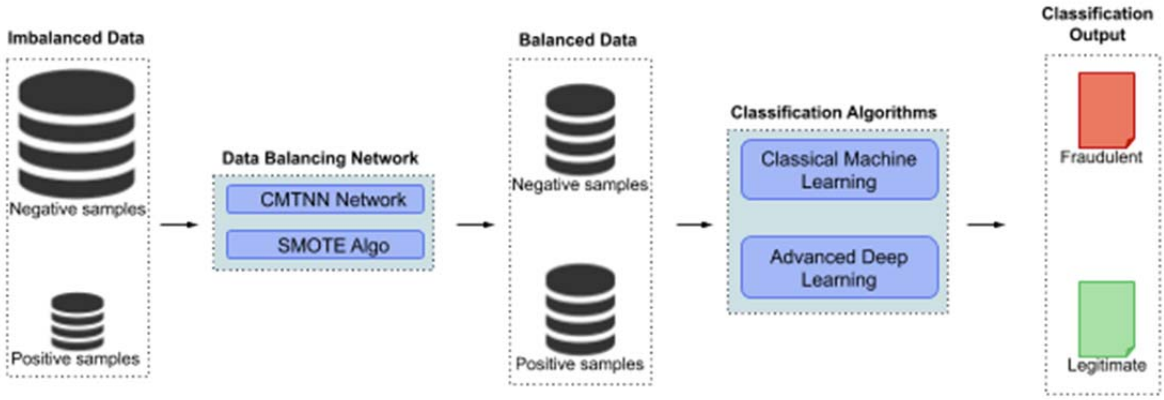

Figure 4.2: Proposed Pipeline

As shown in Figure 4.2, our primary focus has been on the first block of our pipeline, "Data Balancing Network." It is in this section that we perform the majority of the computations. This section although computationally intensive, the two processes i.e., under-sampling (CMTNN Network) and oversampling (SMOTE Algorithm) are independent of each other in the sequential sense and therefore can be parallelized (this will be part of our future work and thus will not be seen implemented as part of this thesis).

We discuss in brief how the “Data Balancing Network” was conceptualized and then explain the improvisations that were incorporated on the way. As explained, the focus of our research was to build a domain-independent "Data Balancing Network" to handle any kind of imbalanced binary classification dataset. The said network should also be simple enough to plug and play into any binary classification algorithm. Thus, considering all such requirements, we have structured our experimental procedures in the following manner:

1. **Step 1**: Decide upon a particular Binary Classification problem. For the scope of this research study, we have chosen “Credit Card Fraud” detection problem.
2. **Step 2**: Choose an authentic real-life dataset related to the chosen problem. (For our chosen problem, we decided to use “Credit Card Fraud” dataset provided by the ULB Machine Learning Group on the platform of Kaggle)

3. **Step 3**: Perform different variations of the proposed “Data Balancing Network” to get the final optimum network architecture and settings.
4. **Step 4**: After having decided upon the optimal version of the "Data Balancing Network," plug it with a few chosen binary classification techniques and analyze the improvement provided by the proposed network. For the scope of our research, we have chosen artificial neural network (ANN), support vector machine (SVM), Random Forest, XGBoost and Logistic Regression.

The experimental procedures enumerated above were diligently followed to establish a standard research framework. This helps to keep the research work easily replicable. Figure 4.1 gives a visual representation of the different stages of our experimental setup. We chose the "Credit Card Detection Problem" because it is one of the most highly imbalanced cases of binary classification, and also one of the most researched domains owing to the immense importance it holds in the real-life scenarios. We chose our dataset from the ULB Machine Learning Group repository(Kaggle). The said research group had curated an authentic version of the credit card transaction dataset containing only 0.172% samples for the class in the minority. We started experimenting with the most basic version of our proposed "Data balancing network" and kept on improving its performance in terms of reduction in the class imbalance. Once having attained a sufficiently good enough balancing of the dataset, we froze that model of the "Data Balancing Network". This best performing version of the network was then plugged together with different binary classification algorithms and used to perform binary classification on our dataset. The binary classification outputs were compared for the set of classification algorithms with and without the "Data Balancing Network" to analyze the improvement in performance provided by our "Data Balancing Network."

For our research, we have considered the following classification algorithms to evaluate the effect of our “Data Balancing Network”:

1. ANN (Artificial Neural Network)
2. SVM (Support Vector Machines)
3. Random Forest
4. XGBoost
5. Logistic Regression

The performance evaluation part holds great importance in our study. Due to the imbalanced nature of the dataset, standard performance metrics like "accuracy" and "Loss (performance metrics to analyse how bad the model's prediction is)" fail to present a true picture of the performance of any algorithm. For example, the dataset that we have used contains only 0.017% of fraudulent transactions. Thus, any classifier can achieve an accuracy score of 99.983% by classifying all the transactions as "Genuine." Therefore, for the scope of this study, and also based on the use case (Credit Card fraud detection), we have decided to keep our focus more on the "AUC (Area under ROC curve)" metric.

In the next section, we present our reasoning of using the "AUC (Area under the ROC curve)" metric for determining the classification performance score. For each of the classification algorithms that were used in our study, we have recorded the "accuracy" and the "AUC" scores using three different versions of our "Data Balancing Network." The "accuracy" metric was recorded only for observation purposes and was not utilized to analyze the performance improvement provided by our "Data Balancing Network."

4.2 Evaluation Framework

In this section, we will focus on the performance evaluation metrics employed for our experiments. Primarily we will discuss the need for not using the standard "accuracy" and "loss" values for scoring our system and we present our reasoning for using "AUC" as a good performance metric which reflects a true picture of the classification performance for Imbalanced Binary Classification problem.

"Accuracy" scores in Classification have been long used for evaluating the strength of a classifier. The reason for its popularity is 3-fold:

- It is easy to Calculate
- It is easy to interpret.
- It is easy to present an analysis based on a single number that sums up the performance strength of the complete system.

But in the case of Imbalanced Binary Classification, when the class distribution in the samples is severely skewed i.e., any one of the classes becomes extremely dominant owing to discriminatingly large numbers of samples of that class, "accuracy" scores become an unreliable metric of performance. The reason for this unreliability stems from the probability theory. Since one of the classes has an extremely large number of samples in the dataset, sometimes to the tune of 99.9%, probabilistically speaking, the chance of occurrence of that particular class has become as high as 99.9%. Thus, any naive algorithm can simply predict that class for all of the samples in the dataset and inaccurately achieve an accuracy score of 99.9%. This score, although looks impressive at first glance, is in no way close to the true performance measure of the algorithm, which is blindly predicting the occurrence-dominant class for all samples.

It is interesting to note that the "accuracy" metric is not at fault here. In fact, it still delivers the correct score. The "accuracy" score is still a true value of classification accuracy. But the predominant perception in the mind of a data-science or machine-learning researcher will lead him to believe that 99.9% is a great score, while actually, 99.9% is achievable without any thought and thus is probably not as great as it may look. Hence, perhaps in the actual case of imbalance classification, 99.999% might be a considerable score. But again, analyzing accuracy scores in only a range of 0.1% is not practically feasible and will introduce problems of its own.
Thus, the need arises for an evaluation metric:

- which presents a true picture of the performance strength of any given algorithm.
- which is simple enough to be used for critical comparison of relative performance strength of different classification algorithms
- which survives the test of Imbalanced Class problem and presents a reasonable score value focusing more on "False Negatives" and "True Negatives." (Assuming that the minority class is considered as the positive class)

Keeping all the above considerations in mind, we suggest using the "AUC" scores for model performance comparison. AUC is the area under the ROC curve. Since the ROC curve is framed inside a unit area square, the resultant AUC score is a real number ranging between 0.0 and 1.0.
One important reason for "AUC" scores to be better able to reflect true model performance strength for Classification when compared to "accuracy" scores is the way the two scores have been defined:

- **Accuracy**: For a given threshold on the prediction value of the model, it calculates the fraction of samples that have been correctly assigned to the class of their belonging, regardless of which class they belong to.

- **AUC**: it measures the probability that considering two random samples, one from the majority (negative) and one from the minority (positive) class, the classification algorithm will output a higher value for the sample from the positive class as compared to the sample from the negative class.

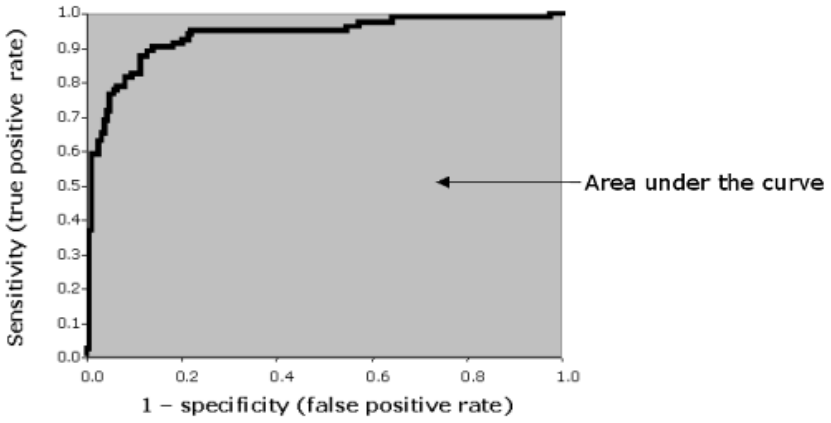

Figure 4.3 Area under the ROC curve

The accuracy depends on the threshold chosen, whereas the AUC considers all possible thresholds. Because of this, the AUC is better suited to present a "broader" view of the classification performance of any given algorithm.

Having established a strong understanding of the Research Framework, we next discuss the Data Balancing Network which targets our major focus presented in section 4.3

4.3 Data Balancing Network

In this section, we discuss the top-level overview of our "Data Balancing Network" (Figure 4.4). We also explain our motivation for using this architecture and contrast it with other existing solutions to handling imbalance class distribution in a dataset. This section will set up the base for subsequent articles, which will further discuss each of the parts (CMTNN Network and SMOTE algorithm) of the "Data Balancing Network" and explain the implementation procedure that we have followed in building up this network.

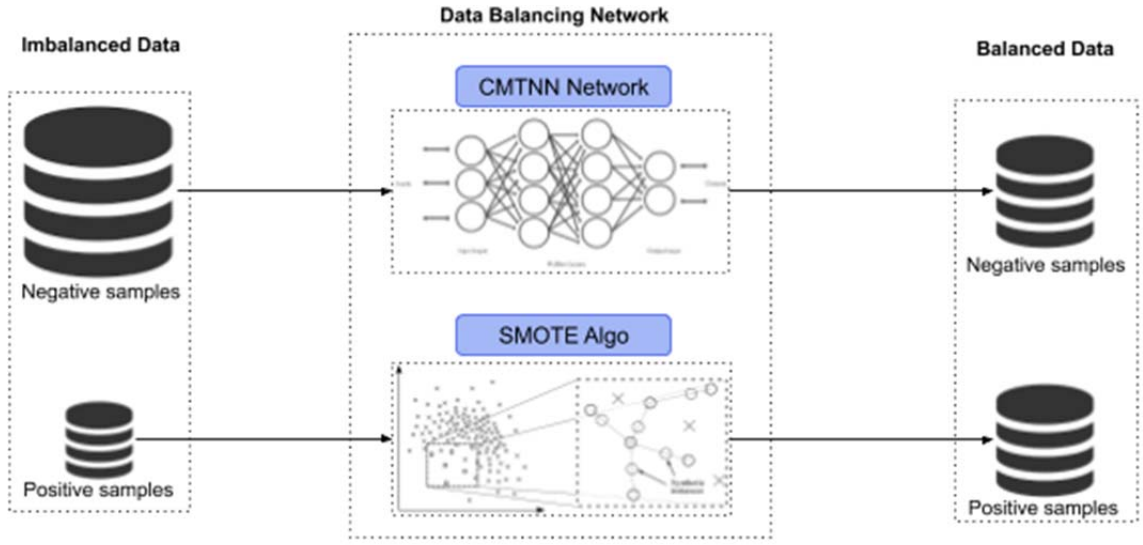

Figure 4.4 Data Balancing Network

Standard data processing techniques like random under-sampling and replicative oversampling have existed among the research community for a long time and have also been successful in several instances. But in our particular case of fraud detection, the aim is not primarily to ease computation (this is usually where typically random sampling proves useful), and replicative oversampling can easily result in overfitting of our classifier. Therefore, in our case, we need a data-centric or information-theory centric approach to under-sampling and oversampling.

Thus, according to Figure 4.4 we have used CMTNN Network for under-sampling while preserving training samples that provide crucial information for feature learning. For the oversampling part, we have used the Synthetic Minority Oversampling Technique (SMOTE) to generate relevant samples for the minority class.

4.4 Complementary Neural Network (CMTNN)

In recent years, Complementary Neural Networks (CMTNN) based on feedforward neural networks have been proposed to deal with both binary and multiclass classification problems [12], [13]. Instead of considering only the correct information, CMTNN found both truth and falsity information in order to enhance the classification results. CMTNN consists of truth neural network and Falsity neural network created based on truth and falsity information, respectively. It was found that CMTNN provides better performance compared to the traditional feedforward neural networks [12], [13]. Figure 4.5 shows the Complementary Neural Network design.

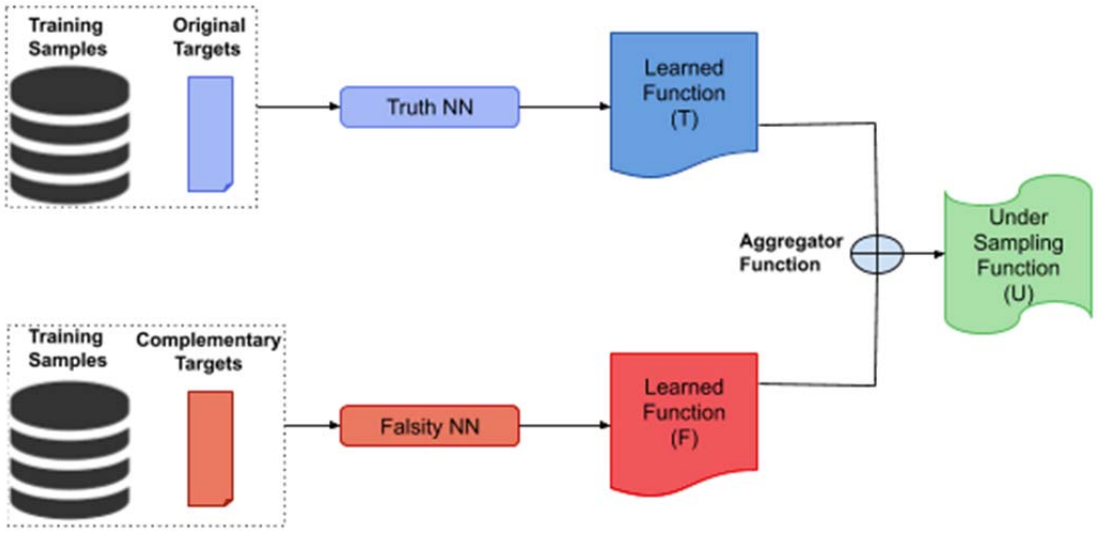

Figure 4.5 Complementary Neural Network Design

The Complementary Neural Network Design is designed in an efficient manner where the training samples are given to both the networks but the targets are complementary to each other. Then the training samples are passed through learned function of each neural network and then combined with an aggregator function where the unnecessary samples are removed

In our design of the CMTNN (Figure 4.6), we train two Artificial Neural Networks “Truth NN” and “Falsity NN”. Both networks are structurally identical in design shown in Figure 4.6 and follows the same ANN architecture inherently. The only difference is that Truth NN uses the

original targets/labels while training, whereas the Falsity NN uses complementary targets while training.

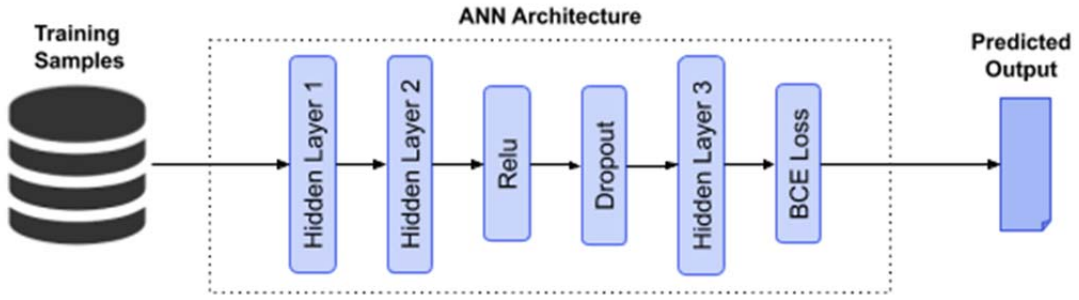

Figure 4.6 Complementary Neural Networks ANN Architecture

In our Architecture of CMTNN (Figure 4.6) describes the architecture of single neural network such as “Truth NN” and “Falsity NN”. Here we use three hidden layers and some functions such as Relu, Dropout and BCE loss which are explained below

Relu: Relu is one of the common activation function of Artificial Neural Network it is also known as rectified linear unit. The mathematical function for Relu is y=max(0,x) where x is a input to the neuron

Dropout: It is a function which is used as a regularization technique in Artificial Neural Network to prevent overfitting of values

BCE loss: The BCE loss is used when we have binary classification which works in the following way when we have one output node and need to classify into two different classes

4.4.1 Implementation of CMTNN network

To apply CMTNN for the under-sampling problem, Truth NN and Falsity NN are employed to detect misclassification patterns i.e. all the patterns classified incorrectly. The intersection of these incorrectly typed samples is then removed from our training data.

a. Truth NN is trained on the training data keeping the target values as it is to learn a classification function (T). Falsity NN is trained on the training data with target values complementary to that provided for Truth NN to learn a classification function (F).
b. Both the networks after reaching convergence are used to classify the majority class of the training data individually.
c. The misclassification patterns i.e., patterns for which the network output is different from the actual output (for Falsity NN actual output is complementary to the actual output of the Truth NN), are stored in Mtruth and Mfalse.
d. Then we find the intersection of Mtruth and Mfalse. These intersection samples are stored in Mfinal. The patterns in Mfinal, in a sense are output of the aggregated function (U). This aggregated function is just a combination of intersection over the classification functions (T) and (F).
e. In the final step, we remove the patterns of Mfinal from our training dataset, thus effectively under-sampling our majority class.

4.4.2 Code Snippets for CMTNN

Truth Neural Network

```
# Creating Torch dataset for training Truth NN
train_ds_tnn =
torch.utils.data.TensorDataset(torch.tensor(xtrain).float(),
torch.tensor(ytrain).float())
valid_ds_tnn = torch.utils.data.TensorDataset(torch.tensor(xtest).float(),
torch.tensor(ytest).float())

# Creating Torch dataloader for training Truth NN
bs =128 # Batch Size
train_dl_tnn = torch.utils.data.DataLoader(train_ds_tnn, batch_size=bs)
valid_dl_tnn = torch.utils.data.DataLoader(valid_ds_tnn, batch_size=bs)

# Static Model Parameters
n_input = xtrain.shape[1]
n_output = 1
n_hidden = 20

# Initializing the Truth NN model
```

```
TNN =
Classifier(n_input=n_input,n_hidden=n_hidden,n_output=n_output,drop_prob=0
.2)

# Static Training Parameters
lr = 0.005   # Learning Rate
pos_weight = torch.tensor([5])  # weight adjustment for class imbalance
opt = torch.optim.SGD(TNN.parameters(), lr=lr, momentum=0.9) #Optimizer
loss_func = torch.nn.BCEWithLogitsLoss(pos_weight=pos_weight)  #Loss
function
#loss_func = torch.nn.BCELoss()
n_epoch = 50  # No. of epochs to run training

# Start Training
train_loss_tnn, val_loss_tnn =
train(n_epoch,TNN,loss_func,opt,train_dl_tnn,valid_dl_tnn)
```

In the first step of CMTNN we create Truth NN with original training samples and with a batch size of 128 which directly describes that number of samples in training in one iteration After that the initialization of TNN is done and static training parameters are given such as Learning Rate, Positive weight, Optimizer and Loss function. Also one of the important parameters epochs which describes the number of passes of training data iteration through the neural network is given.

Falsity Neural Network

```
# Preparing complement targets for "Falsity NN"
ytrain_fnn = 1-ytrain
ytest_fnn = 1-ytest

# Creating Torch dataset for training Falsity NN
train_ds_fnn =
torch.utils.data.TensorDataset(torch.tensor(xtrain).float(),
torch.tensor(ytrain_fnn).float())
valid_ds_fnn = torch.utils.data.TensorDataset(torch.tensor(xtest).float(),
torch.tensor(ytest_fnn).float())

# Creating Torch dataloader for training Falsity NN
bs =128 # Batch Size
train_dl_fnn = torch.utils.data.DataLoader(train_ds_fnn, batch_size=bs)
valid_dl_fnn = torch.utils.data.DataLoader(valid_ds_fnn, batch_size=bs)
```

```
# Static Model Parameters
n_input = xtrain.shape[1]
n_output = 1
n_hidden = 20

# Initializing the "Falsity NN" model
FNN =
Classifier(n_input=n_input,n_hidden=n_hidden,n_output=n_output,drop_prob=0
.2)
# Start Training
train_loss_fnn, val_loss_fnn =
train(n_epoch,FNN,loss_func,opt,train_dl_fnn,valid_dl_fnn)
```

The second step is similar to Truth Neural Network but here the complements of training targets are given as an input.

Combining Truth Neural Network and Falsity Neural Network

```
M_intersection = list(set(M_fnn).intersection(M_tnn))
M_union = list(set(M_fnn).union(M_tnn))


xtrain_cm = np.delete(xtrain, M_union, axis=0)
ytrain_cm = np.delete(ytrain, M_union, axis=0)
```

Finally, both the neural networks are combined where intersection and union function is used where the union samples are deleted/removed/ undersampled, only the samples with intersection remain.

4.5 SMOTE Algorithm

One approach to addressing imbalanced datasets is to oversample the minority class. The most straightforward approach involves duplicating examples in the minority class, although these examples do not add any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of data augmentation for the minority class and is referred to as the Synthetic Minority Oversampling Technique, or SMOTE for short.

This technique increases the number of minority class instances by interpolation methods. The minority class instances that lie together are identified before they are employed to form new minority class instances. This technique is able to generate synthetic instances rather than replicate minority class instances; therefore, it can avoid the over-fitting problem.

4.5.1 Implementation of SMOTE algorithm

Synthetic Minority Over-sampling Technique (SMOTE) algorithm applies the KNN approach where it selects K-nearest neighbors, joins them, and creates the synthetic samples in the space (Figure 4.7). The algorithm takes the feature vectors and its nearest neighbors and computes the distance between these vectors. The difference is multiplied by a random number between (0, 1), and it is added back to feature. We have applied SMOTE only on our minority class samples to increase the number of minority class samples and make it equal to the remaining samples in the majority class (remaining after applying CMTNN).

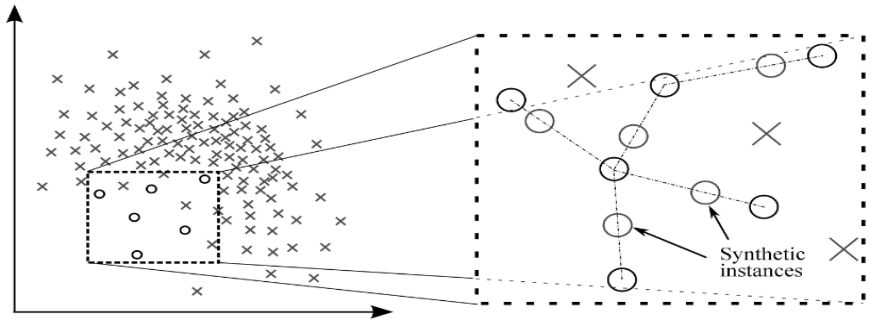

Figure 4.7 Synthetic Minority Over-sampling Technique [23]

It is important to note that in our workflow, we start with a huge bias in the number of samples where the minority class constitutes only 0.172% of the total transaction samples. We then under sample the majority class by applying CMTNN and then oversample the minority class to match the number of samples in the reduced majority class.

To analyze the effect of SMOTE alone, first, we take a fraction of the raw data (we had used 60% of the information for this part). Then, we first generate a scatter plot of this fraction of data before applying SMOTE techniques. This will also form a variation of our "Data Balancing Network".

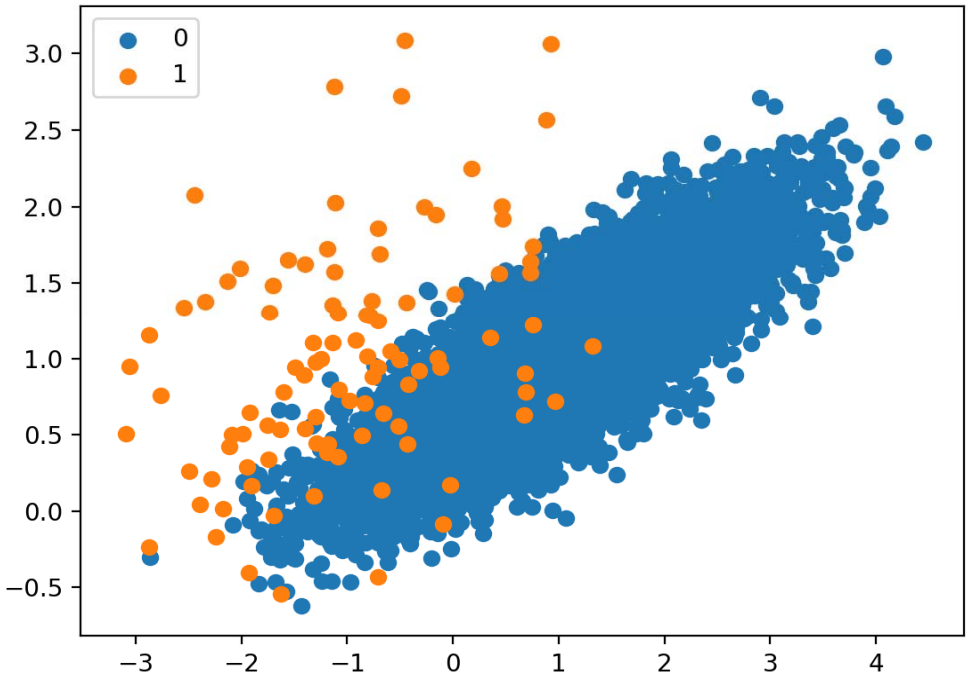

Figure 4.8 Scatter plot of raw data (before applying SMOTE)

Then we take this fraction of the raw data and run the SMOTE algorithm over it to obtain a balanced class distribution in the dataset's samples. A scatter plot is then generated again to analyse the class distribution of the data graphically.

Figure 4.9 shows a scatter plot for the class distribution of the dataset after restoring class balance on applying SMOTE.

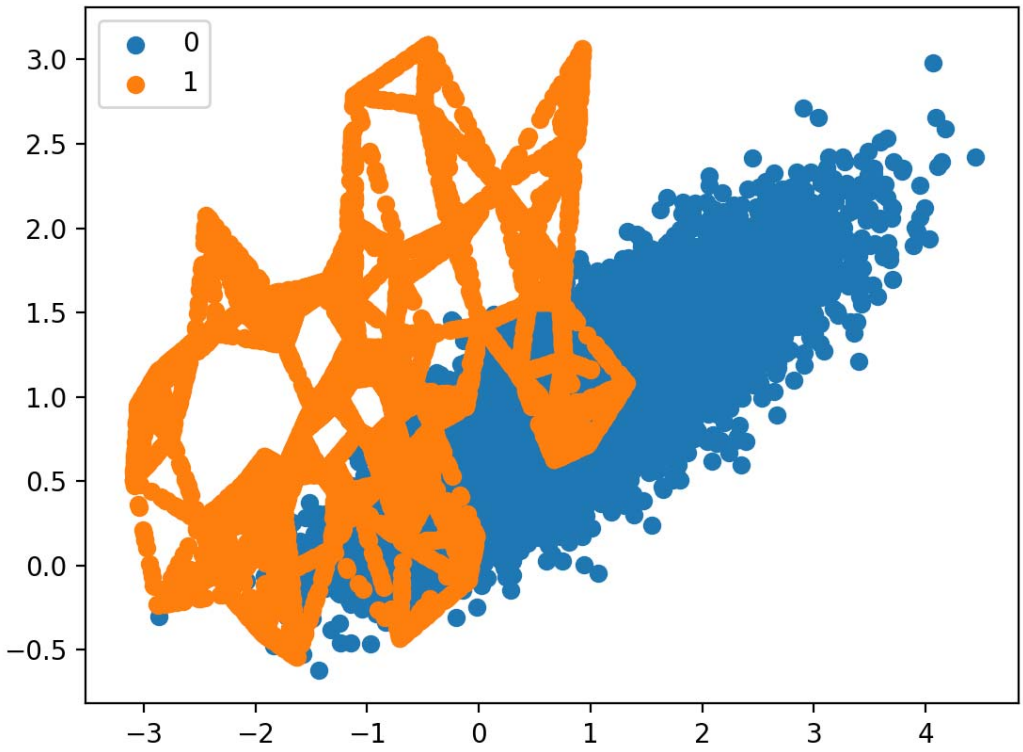

Figure 4.9 Scatter plot of processed data (after applying SMOTE)

From the two scatter plots, it is clearly observable that SMOTE performs reasonably well in generating new samples to increase the number of samples in the minority class.

After our Major focus is done using CMTNN and SMOTE the balanced data is then applied to the classification Algorithms which will be discussed in Section 4.6.

4.6 Classification Algorithms used for final analysis

The next block in our pipeline (after the Data Balancing Network) is the classification algorithms block (Figure 4.10). For our experiments, we had narrowed down our list to five classifiers (SVM, Random Forest [7], XGBoost [10], Logistic Regression, ANN [6]). The data, after being

processed from the Data Balancing Network is then passed to the classification algorithms block. Figure 4.10 shows the classification algorithms used in the final stage.

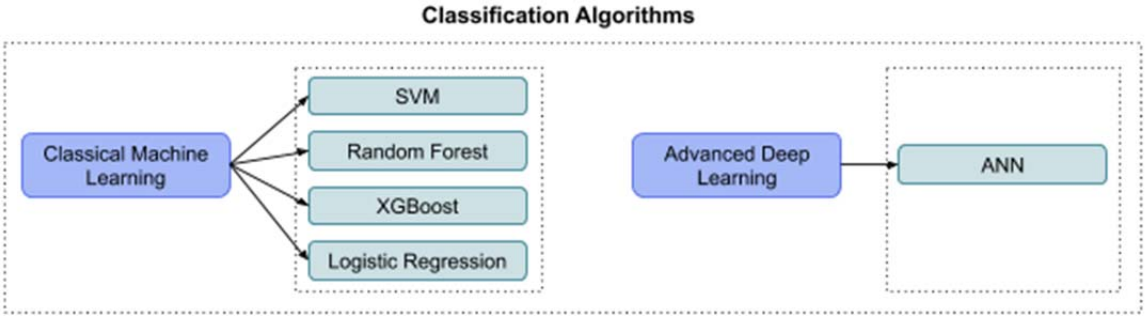

Figure 4.10 Classification Algorithms (used for final analysis)

The choice of Classifiers was made based on existing papers that have employed these classifiers for credit card fraud detection or similar tasks, and also on their frequent use in the research community. The ANN classifier follows a simple network architecture of “3 Hidden Layers”, “1 ReLU block”, and “1 Dropout Layer”. The final output of the “Dropout Layer” was passed through a “Sigmoid Layer" and then fed to “Binary Cross-Entropy Loss Layer." Structurally, the architecture is identical to that of CMTNN Network.

We will keep our focus at presenting our results obtained from this study and extracting meaningful insights. In particular, we will focus on analyzing the improvement in AUC score provided by our “Data Balancing Network” for each of the classification algorithms.

Earlier, we had discussed how we could frame the "fraud detection" problem as a variant of "Outlier detection". In this section, we will see a more generic and more popular method of framing the "fraud detection" problem as a classification problem. Wherein, it is assumed that you have two classes - "Genuine" and "Fraudulent" and you train a classifier on your dataset to learn how to assign a class to any given transaction event. Supervised learning approaches are best suited for this kind of problem.

In this thesis, we would be utilizing several supervised learning classifiers to demonstrate the effect of our "Data Balancing Network." As discussed above, the supervised learning approach is

a machine learning approach that utilizes training data with the target classes in the data set to produce an inferred function that pairs an input to the desired output value. Popular choices of supervised learning-based classifiers include:

- Support Vector Machine
- Logistic Regression
- Artificial Neural Networks
- Extreme Gradient Boosting
- Random Forest

Support Vector Machine and Neural Networks are one of the popular trainable classifiers for multi-dimensions and continuous features. They are particularly impressive in their ability to extract higher-order hidden patterns and features in the dataset. Neural network models and Support Vector Machine require huge training dataset sizes to achieve their maximum prediction accuracy, which again becomes a bottleneck as most of the data related to fraud detection suffers from the "Imbalance Problem."

4.6.1 Support Vector Machines (SVM)

In 1995, Cortes and Vapnik first introduced Support Vector Machines (SVM). This Machine Learning algorithm discovers a special kind of linear model, the maximum margin hyperplane. The training instances are correctly classified by the separation of classes into a search space using hyperplane. The maximum margin hyperplane (MMH) is the one that gives the most significant separation between the classes – it comes no closer to any of the classes than it has to. The instances that are closest to the maximum margin hyperplane – the ones with minimum distance to it – are called support vectors. There is always minimum one support vector for each class, and often there are more. The optimal hyperplane is found by maximizing the width of the margin. As shown in Figure 4.11 below, the margin is the distance between the separating hyperplane and the closest positive class and negative class

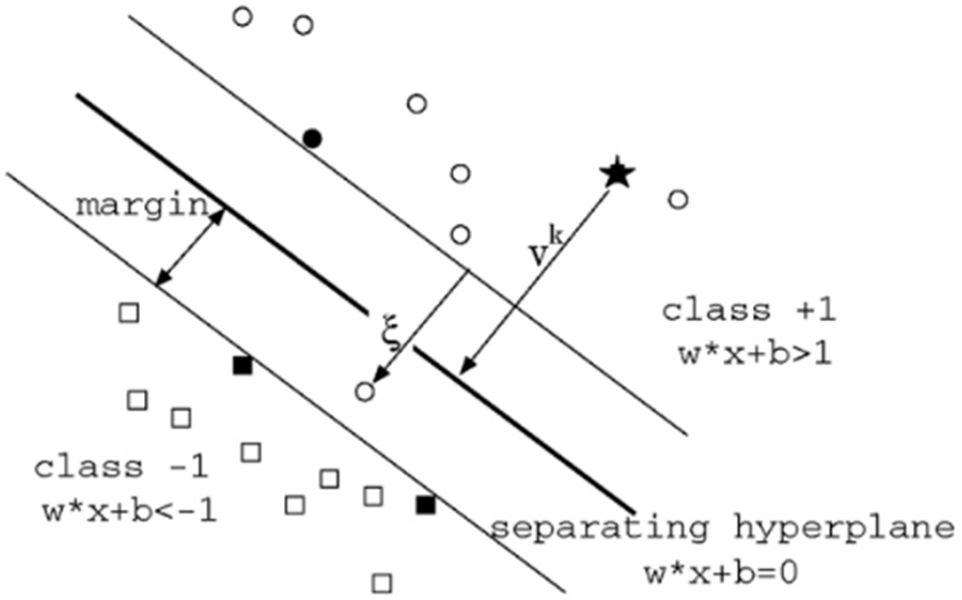

Figure 4.11 Maximum Margin Hyperplanes (MMH) separating two classes in SVM [24]

In situations that the classes are not entirely separable, the SVM algorithm finds the hyperplane that maximizes the margin while minimizing the misclassified instances.

There is some possibility where a nonlinear region can separate the classes more effectively. Instead of fitting nonlinear curves to the data, SVM determines a dividing line with the help of a kernel function to map the data into a different space where a hyperplane can be used to do a linear separation. Kernel mapping function is compelling because it allows SVM models to perform separations even with very complex boundaries. An infinite number of kernel mapping functions can be used. Still, the Radial Basis Function (RBF) has been found to work well for a wide variety of applications, including credit card fraud [27].The transformation to a high-dimensional space is done by replacing every dot product in the SVM algorithm with the Gaussian radial basis (GRB) function kernel as follows:

$$K(\chi_i, \chi_j) = \exp(-\gamma \parallel \chi_i - \chi_j \parallel^2), \gamma \geq 0$$

$$K(\chi_i, \chi_j) = \varphi(\chi_i) \cdot \varphi(\chi_j)$$

where $K(\chi_i, \chi_j)$ is the kernel function, and φ(x) $\varphi(\chi_i)$ is the transformation function [28].

4.6.2 Logistic Regression

When the dependent variable takes only two values, and the independent variables are continuous, categorical, or both that's when Logistic regression is used. The logistic regression method is ideal when classifying outcomes that only have two values because the logistic curve is limited to values between 0 and 1.

In credit card fraud detection, the dependent variable would take on a value of 0 (legitimate transaction) or 1 (fraudulent transaction). Unlike ordinary linear regression, however, logistic regression does not assume a linear relationship between the independent variables and the dependent variable. Nor does it imply that the dependent variable or the error terms are distributed normally.

We can call a Logistic Regression a Linear Regression model, but the Logistic Regression uses a more complex cost function, this cost function can be defined as the 'Sigmoid function' or also known as the 'logistic function' instead of a linear function. To map predicted values to probabilities, we use the Sigmoid function. The feature maps any real value into another value between 0 and 1. For our particular case, Logistic Regression would output probability values indicating the probability of a transaction belonging to the "Genuine" class or "Fraudulent" class.

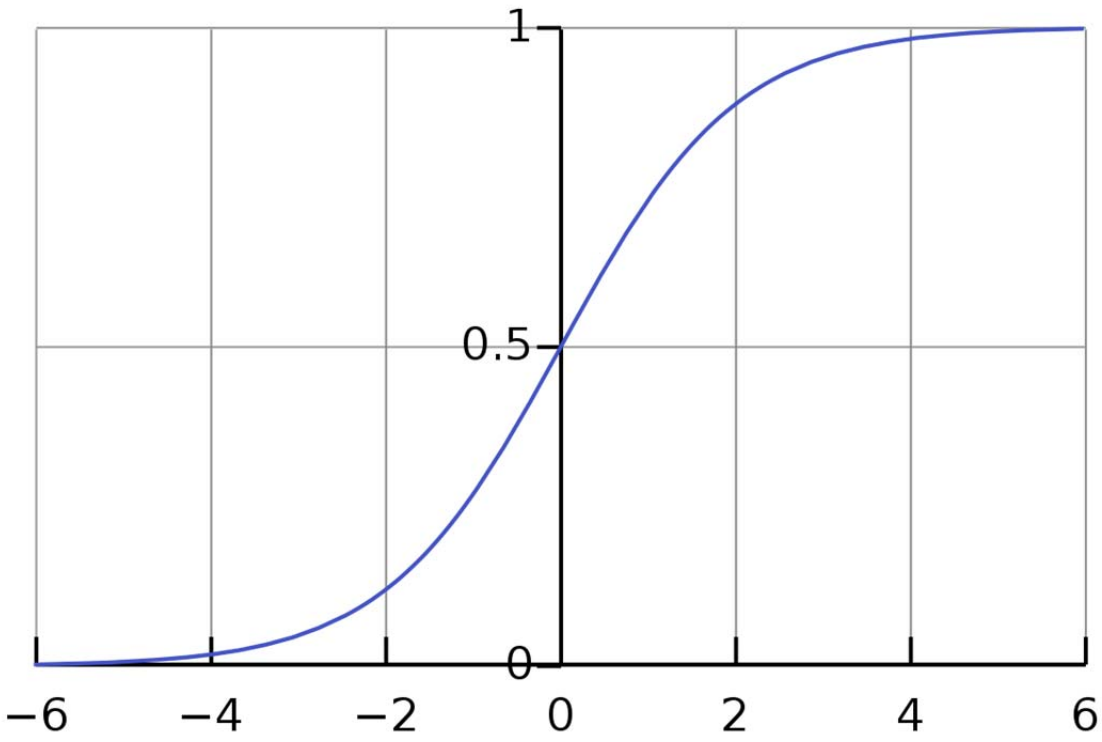

Figure 4.12 Logistic or Sigmoid function [25]

Figure 4.12 shows a graphical representation of the "Sigmoid" function. Thus, the output of Logistic Regression is mapped to a value lying between 0 and 1. They are thus resulting in a probability of belonging to a particular class.

4.6.3 Neural Networks

Artificial Neural Networks (ANN) are computational models that try to mimic our bodies' biological neural networks and quickly adapt to change. This mathematical model consists of interconnected artificial neurons (nodes) that can receive one or more inputs and sums them to produce a prediction (output). A neuron has two modes of operation: training mode, and usage mode. In training mode, the neuron can be taught to associate a particular prediction with an input pattern. While in usage mode, if the neuron detects a taught input pattern its associated prediction is outputted.

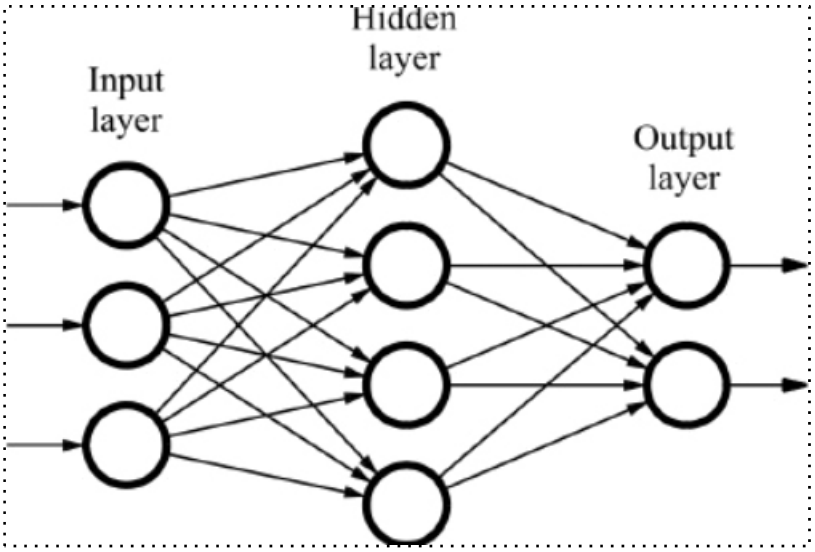

Figure 4.13 Neural Network Architecture with 1-Hidden Layer [26]

A neural network consists of three essential layers:

Input Layer: As the name suggests, this layer accepts all the inputs provided by the programmer.
Hidden Layer: Layer between the input and the output layer is a set of layers known as Hidden layers. In this layer, computations are performed, which result in the output.
Output Layer: The inputs go through a series of transformations via the hidden layer which finally results in the output generation at this layer.

The effect of each input's contribution to the final prediction is dependent on the weight of the particular input. To determine a neural network that is an accurate predictor, appropriate weights for the connections must be determined. The most famous method used to determine the optimal connection weights is called backpropagation. This method was introduced by Rumelhart et al. [32], and through their work artificial neural network research gained recognition in machine learning.

With the evolution of the technological landscape, computation has become more affordable and viable over the years, which has given a significant boost to the machine learning industry. This has also led to a multitude of innovations in the Neural Network space itself. Deeper and more

powerful Neural Networks are being deployed and used in all walks of our life. The banking industry too, has utilized Neural Networks for various tasks including fraud detection. Neural Networks are considered a very powerful tool in providing smarter and better services with "Advanced Recommendations," "Consumer Risk Modelling" etc.

4.6.4 Random Forest

As its name suggests, Random Forest consists of a large number of individual decision-making trees that function as a collective. Every tree in the random forest spits out the class prediction, and the class with the most votes becomes our model prediction. The fundamental concept behind the random forest is simple but powerful — the wisdom of the crowds. In data science, the reason that the random forest model works so well is a large number of relatively uncoordinated models (trees) operating as a committee will outperform any of the individual constituent models.

This idea of having an ensemble of a large number of uncoordinated models brings about the desired robustness and decreased error rate in the Random Forest algorithm. The explanation for this wonderful impact is that trees shield each other from their errors (as long as they don't all fail in the same direction). When individual trees are incorrect, several other trees would be correct, so that a community of trees will migrate in the right direction.

The Decision Trees form the building blocks of the Random Forest. The decision trees are made up of nodes and branches. Typically, the starting node is referred to as the root node. -- node is labeled with a name for a feature, and each branch out of it is labeled with one or more possible values for that feature. Every node has only one incoming branch, except the root, which is designated as the starting point. Growing the internal node in the tree is a check of the value of one of the features. The node divisions are labeled with the possible values of the test. Leaves are labeled with the values of the classification features and determine the value to be returned when that leaf is reached. By taking the set of features and their corresponding values as input, the decision tree is able to define the case by moving through the decision tree. Depending on

whether the check result is true or false, the tree branches to one node or another. The label of the instance corresponding to the tree root label is compared to the values of the outgoing branches of the root, and the matching branch is selected. This node label matching and branch selection process continue until a terminal node, referred to as leaf, is reached, at which point the case is classified according to the label of the leaf and a decision is made on the class assignment of the case [19].

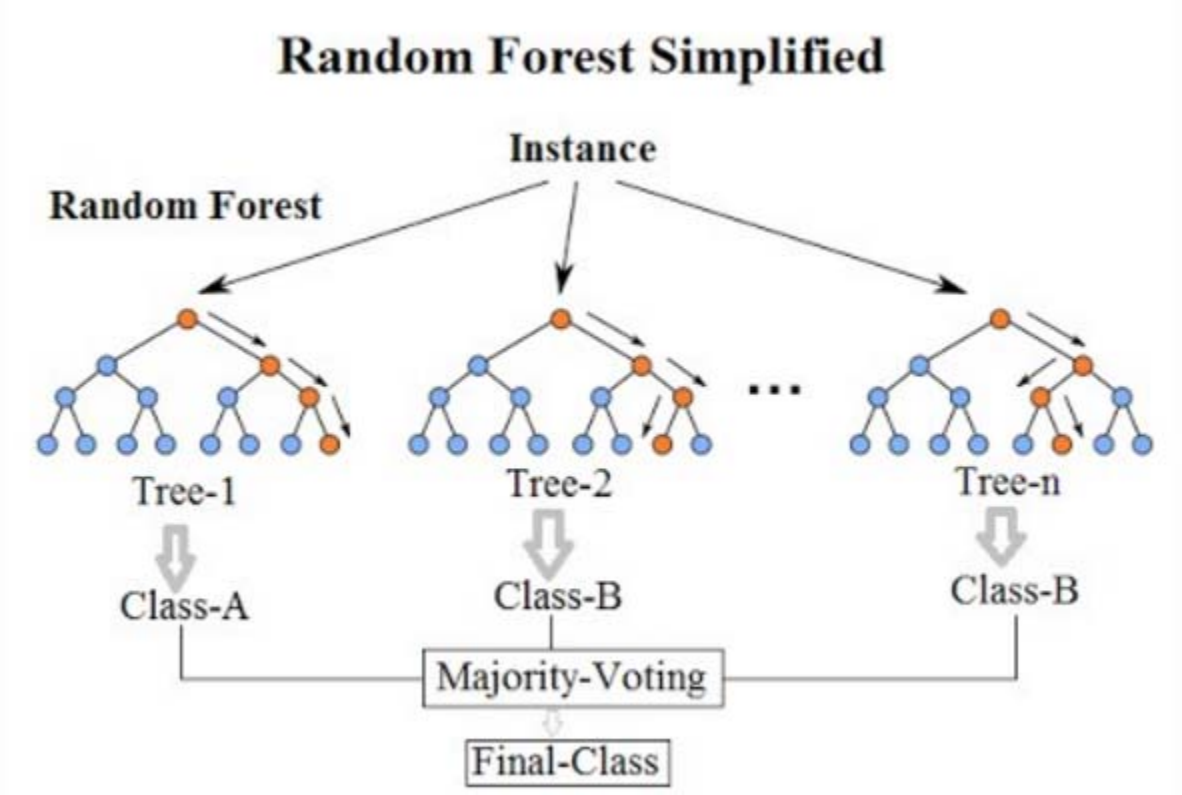

Figure 4.14 Random Forest example with constituent Decision Trees [27]

The first algorithm for Random Forests was created by Tin Kam Ho [20] using the random subspace method [21] which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg [22].

4.6.5 XGBOOST [Gradient Boosted Decision Trees]

XGBoost stands for 'Extreme Gradient Boosting,' where the word 'Gradient Boosting' originates from Friedman et al [22]. Boosting means combining a learning algorithm in series to

achieve a strong learner from many sequentially connected weak learners. The weak learners are decision trees, in case of gradient boosted decision trees algorithm,

XGBOOST algorithm is similar to the Random Forest algorithm in the sense that they are both a sort of wrapper that is implemented over the Decision Tree classifier. In simple terms, both Random Forest and XGBOOST are ways of combining data from multiple decision trees to obtain improved robustness to error and high accuracy performance. Every tree aims to minimize the errors in the previous tree. Trees in boosting are weak learners, but adding a lot of trees in series and each focusing on the mistakes of the previous one will boost a highly efficient and accurate model. In contrast to bagging, boosting does not involve bootstrap sampling. Whenever a new tree is added, it matches the updated version of the initial dataset. Since trees are added sequentially, boosting algorithms learn slowly.

Gradient boosting algorithm sequentially combines weak learners in such a way that each new learner matches the residuals of the previous stage in order to strengthen the model. The final model integrates the effects of each move and a good learner is achieved. Gradient-boosted decision trees algorithm uses decision trees as weak learners. The loss function is used to identify residues. For example, mean squared error (MSE) can be used for regression tasks, and log loss can be used for classification tasks. It should be noted that existing trees in the model do not change when a new tree is added. The added decision tree fits the residues of the current model. In XGBOOST and Random Forest, the performance measure is directly dependent on the performance of the underlying decision trees to a great extent. Therefore, it has become important to make sure that the performance of the underlying decision trees itself does not become a bottleneck for the final prediction performance.

All the supervised learning approaches discussed above have their own weaknesses and strengths. But more importantly almost all of them have shown exemplary performance with respect to the general metric of "Accuracy". Although, it is equally important to understand that in cases of imbalanced class datasets, "Accuracy" metric does not deliver a clear measure of the performance. For example, the dataset that we have used, contains only of fraudulent transactions. Thus, any classifier can achieve an accuracy score of by classifying all the

transactions as "Genuine". Therefore, for the scope of this study, and also based on the use case (Credit Card fraud detection), we have decided to keep our focus more on the "AUC (Area Under ROC curve)" metric. The reasoning behind using "AUC" and the benefits expected and observed out of it will be discussed in chapter 5.

Chapter 5

Results and Discussion

5.1 Complementary Neural Network (CMTNN) Experiments

To solve the class imbalance problem described in Chapter 4, we present our innovative approach of utilizing "CMTNN [3]" and "SMOTE [4]" in a synergic manner to implement under sampling on majority class and oversampling on the minority class. The crucial part for achieving the best possible effect of the “Data Balancing Network” is hinged on the performance of its underlying constituent parts: CMTNN Network and SMOTE Network.

Now, out of the two constituent blocks of the "Data Balancing Network," the CMTNN network has a lot of scope for tuning to optimization due to its neural network like architecture. Thus, we perform detailed experiments to figure out the optimum parameters for the CMTNN network to achieve the best possible under sampling, i.e., it should not remove samples containing crucial information. It should also provide a significant reduction in the number of samples to reduce the imbalance. Our experiments on CMTNN network aim to find the perfect balance to get the desired imbalance reduction without compromising on crucial information.

Since the CMTNN network has an underlying neural network architecture, it is essential to note that differently trained CMTNNs might have a different extent of under sampling (i.e., the final compression achieved over the majority class data). Therefore, we perform an incremental analysis over the training epochs to decide how much (to how many epochs) training would provide optimum results. This analytical exploration is aimed at only finding the optimum extent of training of the "CMTNN Network," and thus, we have used only one classifier at this stage (i.e., ANN). We compare the performance of "CMTNN + ANN" architecture w.r.t. a baseline of vanilla ANN at six different levels of training of "CMTNN Network," i.e. epoch 10, epoch 20, epoch 50, epoch 100, epoch 150.

Table 5.1: Performance of Incremental training of CMTNN

Metrics	**ANN (Classifier) (No CMTNN)**	**CMTNN (Balancing Network) + ANN(Classifier) (by varying extent of Training of CMTNN Network)**				
	(No CMTNN)	*Epoch 10*	*Epoch 20*	*Epoch 50*	*Epoch 100*	*Epoch 150*
AUC	0.898	0.854	0.659	0.688	0.741	0.670
Accuracy	0.837	0.739	0.340	0.397	0.522	0.371
Compression (%)	0.0	0.02	44.78	49.72	69.83	88.24

Data set which is used is highly Imbalanced with 99.83% and 0.17% of Non -Fraud Transactions (Class :0) and Fraud Transactions (Class :1) respectively. Dataset should be balanced in order to perform training part. To tackle this problem CMTNN comes into the picture. CMTNN is an Under sampling Approach which reduces the Number of Samples from the Majority Class which is Non -Fraud Transactions (Class :0). CMTNN have been used for reducing Number of Samples from Majority Class.

It can be seen from the Table 5.1 that if one doesn't use CMTNN then ANN classifier gives good AUC and Accuracy. Reason behind this is data set is not balanced yet and having most of the samples of majority classes which makes ANN to assume that Majority of Samples are of Non-Fraud transaction class. That's why ANN ends up with the Non-Fraud transaction class as a result which is true in most of the cases because our dataset made from the majority of non-fraud transaction class. Due to this reason it gives better accuracy but the model and the performance is not efficient because the model was trained with the dataset which are having less information about the fraud transaction class. To overcome with this problem CMTNN is used.

After applying CMTNN to the majority class, we will see the reduction in Number of Samples of majority class. The reduction of samples is shown by compression (%). Different extent of training the CMTNN network also resulted in different levels of reduction (due to under-sampling) achieved in the number of samples of majority class. The amount of reduction

(mentioned as “Compression” in percentage form in Table 5.1) achieved by varying the number of epochs of training the “CMTNN Network” is presented in the Table 5.1 and also represented in the Figure 5.2. The compression in the number of samples of the majority class is calculated as percentage of the number of total samples of the majority class before any under sampling. After reducing the number of samples in the majority class, we use that reduced data set to perform prediction by passing it through normal ANN classifier.

.

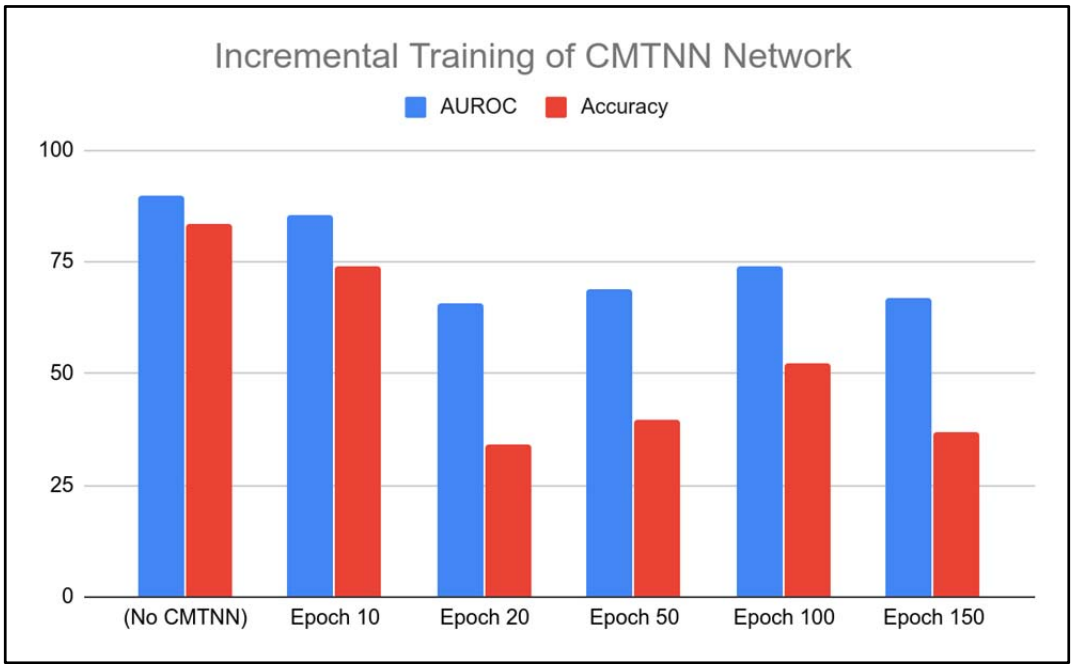

Figure 5.1 Analysis of incremental training of CMTNN

The detailed results are present in the Table 5.1 and Figure 5.1 above.

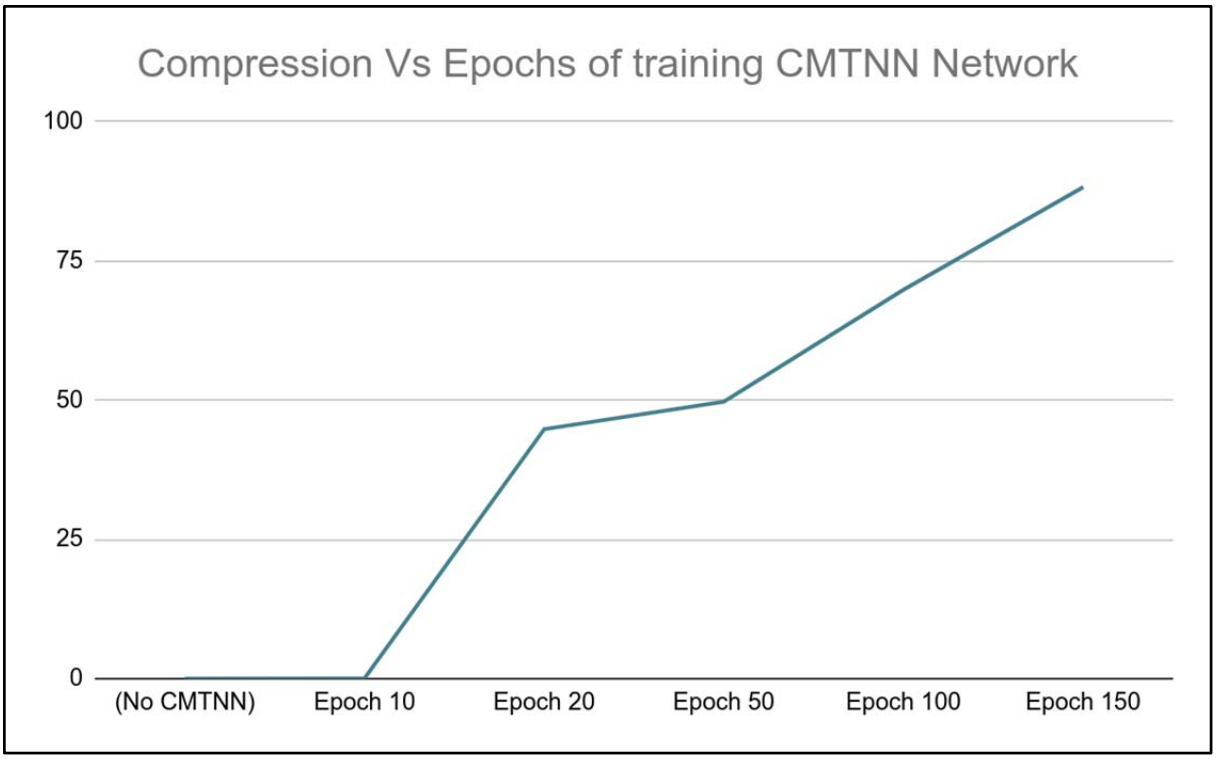

Figure 5.2 Analysis of incremental training of CMTNN

From the results obtained in our exploratory analysis we have observed that any significant compression only starts after the CMTNN Network has been trained for at least 20 epochs. One can also notice that the “Accuracy” and “AUC" metrics are on a rising curve from epoch 20 to epoch 100 but then there is a noticeable dip from epoch 100 to epoch 150. Thus, we can conclude that based on our dataset and the underlying neural network architecture of our CMTNN network, we achieve optimum performance (in terms of Accuracy and AUC) of CMTNN when it is trained for 100 epochs. A compression of 70% in total no. of samples of majority class is achieved when the CMTNN Network is trained at its optimum level of performance (epoch 100).

5.2 Binary Classification Results

For the larger part of our experimental analysis, we have employed five classifiers (Support Vector Machine, Random Forest, XGBoost, Logistic Regression and Artificial Neural Network), one based on advanced deep learning architecture and the remaining four based on traditional machine learning methodologies. Our choice of classifiers for this part of experiment was based

on our literature review of existing works on credit card fraud detection in the research community. It is important for the reader to note that the focus of our study is not to explore the best classifier but to explore the effect of our “Data Balancing Network” on the performance of the classifiers. Thus, for each of the classifiers we have analyzed them in three different forms i.e. “CMTNN”, “SMOTE”, “CMTNN + SMOTE”.

In the “CMTNN” form, The proposed CMTNN based under sampling on the majority class and left the minority class as-it-is. In the “SMOTE” form, the SMOTE based oversampling on the minority class to increase the number of samples to match that of the majority class. In the “CMTNN + SMOTE” form, firstly the applied CMTNN based under sampling on the majority class and then used SMOTE based oversampling on the minority class to increase the number of samples to match that of the modified majority class.

In all of the cases, which are mentioned above 70:30 training to testing ratio, is used. After applying CMTNN, SMOTE and CMTNN+SMOTE in each case new dataset is generated from which 70% of the data has been used to train the model and remaining 30% is used to test the model.

Thus, there are two baselines to compare the performance of our proposed network and all of the classifiers in all of the forms were trained till convergence. The results of this study are presented in Table 5.2

The Study also presented Precision, Recall and F-score for showing contribution in performance of the experiments. The terms are explained below.

Precision: Precision is a term which describes the number of correct predictions made by the classifier. It is calculated as

Precision= True Positive / (True Positive + False Positive)

Recall: Recall is also known as the "true positive" rate. It calculates the proportion of actual positives that are correctly identified as such. Generally, it is calculated as

Recall = True Positive / (True Positive + False Negative)

F-Score: F-Score is an integration of the term precision and recall into a single score which is F-measure It is calculated as

F-score = (2* Precision* Recall)/ (Precision + Recall)

Table 5.2: Performance results of different models

ANN					
	AUC	**Accuracy**	**Precision**	**Recall**	**F-score**
CMTNN	0.741	0.522	**1.00**	0.93	0.97
SMOTE	0.796	**0.999**	**1.00**	**1.00**	**1.00**
CMTNN + SMOTE	**0.950**	0.880	**1.00**	0.97	0.98
SVM					
	AUC	**Accuracy**	**Precision**	**Recall**	**F-score**
CMTNN	0.924	**0.998**	**1.00**	**1.00**	**1.00**
SMOTE	0.899	0.988	**1.00**	0.99	**1.00**
CMTNN + SMOTE	**0.953**	0.926	**1.00**	0.93	0.97
Random Forest					
	AUC	**Accuracy**	**Precision**	**Recall**	**F-score**
CMTNN	0.956	0.998	**1.00**	**1.00**	**1.00**
SMOTE	0.880	**0.999**	**1.00**	**1.00**	**1.00**
CMTNN + SMOTE	**0.997**	0.993	**1.00**	0.99	**1.00**
XGBoost					
	AUC	**Accuracy**	**Precision**	**Recall**	**F-score**
CMTNN	0.925	**0.993**	**1.00**	**0.99**	**1.00**
SMOTE	0.937	0.993	**1.00**	0.99	**1.00**
CMTNN + SMOTE	**0.967**	0.965	**1.00**	0.96	0.98
Logistic Regression					
	AUC	**Accuracy**	**Precision**	**Recall**	**F-score**
CMTNN	0.973	**0.999**	**0.99**	**0.93**	**0.97**
SMOTE	0.914	0.972	**1.00**	0.99	**1.00**
CMTNN + SMOTE	**0.932**	0.912	**1.00**	0.99	**1.00**

As evident from the above results, our proposed approach "CMTNN + SMOTE" provides greater "AUC: Area under the ROC curve" than any of the three baselines for every classifier while compromising for a small dip in accuracy.

To understand the significance of the results obtained in this study, one must also understand why "AUC" score is more reflective of the classifier's ability to detect fraudulent transactions for our case of highly skewed dataset. "Accuracy" measures the percentage of points correctly classified, regardless of which class they belong to. Whereas "AUC" is a measure of the likelihood that given two random points — one from the positive and one from the negative class — the classifier will rank the point from the positive class higher than the one from the negative one. Thus, in a sense "AUC" gives a measure of how clearly (with greater likelihood) the classifier can differentiate a sample of the positive class from a sample of the negative class, which is undoubtedly the most important part in detection of fraudulent credit card transactions.

5.3 Comparative Analysis of Performance Results by different research frameworks

In the previous section, we discussed our results with different performance metrics. In this section, a detailed comparative analysis is done which describes the performance of various existing research frameworks and highlights the benefits and improvements that our framework provides when tested on identical dataset (mentioned in Chapter 3). Table 5.3 provides a comparative analysis. Through this table, we aim to present the superior performance of our proposed fraud detection system as well as a reference to better understand the significance of the relative improvement in performance scores that our "Data Balancing Network" provides.

Table 5.3: Accuracy and AUC scores for research frameworks discussed in this thesis work

Reference Paper	Algorithms Used	Best Classifier	Accuracy	AUC
Sahayasakila et al., 2019 [28]	SMOTE (Oversampling) Whale Optimization Algorithm	Logistic Regression	0.94	0.91
Niu, Wang, & Yang, 2019 [29]	HMM (Hidden Markov Model)	Extreme Gradient Boosting	0.98	0.96
Puh and Brkić, 2019 [30]	SMOTE PCA for Dimensionality Reduction	Random Forest	0.99	0.96
Hordri et al., 2018 [31]	RUC ROS SMOTE	Random Forest	0.99	0.96
Proposed method	**CMTNN with SMOTE**	**Random Forest**	**0.99**	**0.99**

In Table 5.3, we have tabulated the AUC and Accuracy scores from recent research works published in renowned IEEE publication and compared them with the AUC and Accuracy scores of our proposed method. Column-2 presents the underlying algorithms, and different sampling and optimization techniques used. Column-3 mentions the best classifier that corresponds to the highest performance scores. As it can be easily observed, the works of Puh and Brkić [30] and Hordri et al. [31] and our proposed method, all three have shown greater performance score on the accuracy metric. All three methods have successfully achieved an accuracy score of 99%. While on the AUC front, we notice that our proposed method has successfully shown an improvement by 3%. All four trailing methods (Sahayasakila et al. [28], Niu, Wang, & Yang [29], Puh and Brkić [30] and Hordri et al. [31]) achieved an AUC score of 0.96 while our proposed method achieved an AUC score of 0.99.

From comparison of the above scores, it is clear that our proposed method while achieving a stellar accuracy score of 99%, also does not compromise on the AUC score. While AUC scores in general are definitely important, they become increasingly crucial in case of fraud detection systems on highly skewed (imbalanced binary data) datasets. “Accuracy” measures the percentage of points correctly classified, regardless of which class they belong to. Whereas “AUC” is a measure of the likelihood that given two random points — one from the positive and one from the negative class the classifier will rank the point from the positive class higher than the one from the negative one. Thus, the superior performance of our proposed method in terms of AUC score also warrants the superior ability of our system to correctly differentiate among genuine and fraudulent transaction events and thus result in lower false negatives. This is extremely important in the credit card fraud detection domain, because the number of false positives as a percentage of the total transactions is as low as 0.0128% and thus even a miniscule change in the false negative score could result into an unacceptable level of fraudulent transactions to be masked as genuine transactions. Table 5.2 also shows the precision, recall and F-scores while none of the compared papers include this analysis.

Chapter 6

Conclusions and Future Work

6.1 Conclusions from the Research

In this thesis, we have applied a “Data Balancing Network" (based on CMTNN and SMOTE) that reduces class imbalance in a skewed dataset. The architecture of the “Data Balancing Network” is independent of the dataset size or type and thus this technique can be extended to any classification task where the dataset suffers from class imbalance problem. Using five different classification algorithms (Support Vector Machine, Logistic Regression, Random Forest, XGBoost, Artificial Neural Network), we have shown that our proposed “Data Balancing Network” boosts the AUC score for all of the five algorithms. Since AUC score tends to be a better reflection of the classification performance for tasks like fraudulent transaction detection, we have established the performance enhancement provided by our proposed network for such tasks irrespective of the classification algorithm employed.

We have also established the superior performance of our proposed method in comparison to the various recently published research papers. We have also shown that this improved performance in terms of AUC (Area under the ROC curve) is achieved without any compromise in the “accuracy” performance, which our proposed method still managed to keep at 99%. Thus, in this research thesis, we have presented a unique “Data Balancing Network” which boosts performance of any existing classifiers by improving the skewness in the dataset. On top of that, we have also shown how a vanilla “Random Forest” algorithm outperforms the existing advanced solutions when coupled with our proposed “Data Balancing Network”, both in terms of accuracy and AUC scores.

6.2 Future Work

In future, we would implement and analyze the impact of our network on datasets with lower skewness and for a greater variety of classification algorithms (to include higher order Deep Learning algorithms as well). Another possible scope for future development lies in experimenting with the underlying architecture of the CMTNN network. Since the CMTNN network follows a deep learning architecture. This opens up new possibilities of improvement that might be possible by further optimizing the architecture and hyper-parameters of the CMTNN network.

Bibliography

[1] Awoyemi, John O., Adebayo, O. A., Oluwadare S., “Credit Card Fraud Detection Using Machine Learning Techniques: A Comparative Analysis.” In Proceedings of the International Conference on Computing Networking and Informatics (ICCNI), Lagos, October 2017, doi:10.1109/iccni.2017.8123782.

[2] K. Randhawa, C. K. Loo, M. Seera, C. P. Lim and A. K. Nandi, "Credit Card Fraud Detection Using AdaBoost and Majority Voting," in IEEE Access, vol. 6, pp. 14277-14284, 2018, doi: 10.1109/ACCESS.2018.2806420.

[3] Jeatrakul, P., Wong, K.W. and Fung, C.C. “Classification of imbalanced data by combining the complementary neural network and SMOTE algorithm”, In Proceedings of the 17th International Conference on Neural Information Processing, ICONIP 2010, 22 - 25 November, Sydney.

[4] Promrak, J., Kraipeerapun, P. and Amornsamankul, S. “Combining Complementary Neural Network and Error-Correcting Output Codes for Multiclass Classification Problems” Proceedings of the 10th WSEAS International Conference on Applied Computer and Applied Computational Science.

[5] Mohammed, Emad, and Behrouz Far. “Supervised Machine Learning Algorithms for Credit Card Fraudulent Transaction Detection: A Comparative Study.” IEEE Annals of the History of Computing, IEEE, 1 July 2018, doi.ieeecomputersociety.org/10.1109/IRI.2018.00025.

[6] Roy, Abhimanyu, et al. “Deep Learning Detecting Fraud in Credit Card Transactions.” Systems and Information Engineering Design Symposium (SIEDS), 2018, doi:10.1109/sieds.2018.8374722.

[7] Xuan, Shiyang, et al. “Random Forest for Credit Card Fraud Detection.” In Proceedings of the 15th International Conference on Networking, Sensing and Control (ICNSC), IEEE, 2018, doi:10.1109/icnsc.2018.8361343.

[8] Vaishali et al. “Fraud Detection in Credit Card by Clustering Approach”. 2014 International Journal of Computer Applications 98(3):29-32

[9] Stolfo, S., Fan, D. W., Lee, W., Prodromidis, A., & Chan, P. “Credit card fraud detection using meta-learning: Issues and initial results”. In AAAI-97 Workshop on Fraud Detection and Risk Management, 1997.

[10] Pun, J. K. F. “Improving Credit Card Fraud Detection using a Meta-Learning Strategy”, (Doctoral dissertation, University of Toronto), 2011.

[11] Duman, E., Buyukkaya, A., & Elikucuk, I. “A novel and successful credit card fraud detection system implemented in a turkish bank”. In Data Mining Workshops (ICDMW), 2013 IEEE 13th International Conference on (pp. 162-171).

[12] P. Kraipeerapun and C.C. Fung. “Binary classification using ensemble neural networks and interval neutrosophic sets. Neurocomputing”, 72:2845–2856, 2009.

[13] P. Kraipeerapun, C.C. Fung, and K.W. Wong. “Uncertainty assessment using neural networks and interval neutrosophic sets for multiclass classification problems”. WSEAS Transactions on Computers, 6:463–470, 2007.

[14] Pozzolo, A. D., Caelen, O., Johnson, R. A., and Bontempi, G. “Calibrating Probability with Under sampling for Unbalanced Classification.”, In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE.

[15] Wei, W., Li, J., Cao, L. et al. “Effective detection of sophisticated online banking fraud on extremely imbalanced data.” World Wide Web 16, 449–475 (2013). https://doi.org/10.1007/s11280-012-0178-0

[16] Richard J Bolton and David J Hand, “Unsupervised profiling methods for fraud detection. Credit Scoring and Credit Control VII”, pages 235–255, 2001

[17] Aisha Abdallah, Mohd Aizaini Maarof, and Anazida Zainal. “Fraud detection system: A survey. Journal of Network and Computer Applications”, 68:90–113, 2016.

[18] Daniel Sánchez, MA Vila, L Cerda, and José-María Serrano. “Association rules applied to credit card fraud detection. Expert systems with applications”, 36(2):3630–3640, 2009.

[19] Quinlan, J. Ross. C4.5: Programs for Machine Learning. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[20] Ho, Tin Kam "Random Decision Forests", Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Archived from the original on 17 April 2016. Retrieved 5 June 2016.

[21] Ho TK (1998). "The Random Subspace Method for Constructing Decision Forests". IEEE Transactions on Pattern Analysis and Machine Intelligence. 20 (8): 832–844. doi:10.1109/34.709601

[22] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2008). The Elements of Statistical Learning (2nd ed.). Springer. ISBN 0-387-95284-5.

[23] Analytics Vidhya article published on March 17, 2017
https://www.analyticsvidhya.com/blog/2017/03/imbalanced-data-classification/

[24] Shubhendu Trivedi, Onionesque Reality article published on March-22
https://onionesquereality.wordpress.com/2009/03/22/why-are-support-vectors-machines-called-so/

[25] Nabil M Abbas, medium article published on Feb 2016
https://medium.com/swlh/what-is-logistic-regression-62807de62efa

[26] Henok Tilaye and Haylemicheal Berihun, medium article published on Apr-22, 2019
https://medium.com/analytics-vidhya/image-classification-with-convolutional-neural-networks-ac14a978f0fa

[27] V. Hanagandi, A. Dhar and K. Buescher, "Density-based clustering and radial basis function modeling to generate credit card fraud scores," IEEE/IAFE 1996 Conference on Computational Intelligence for Financial Engineering (CIFEr), New York City, NY, USA, 1996, pp. 247-251, doi: 10.1109/CIFER.1996.501848.

[28] Sahayasakila.V, D. Kavya Monisha, Aishwarya, Sikhakolli Venkatavisalakshiseshsai Yasaswi. "Credit Card Fraud Detection System using Smote Technique and Whale Optimization

Algorithm", in Proceedings in the International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249-8958, Volume-8 Issue-5, June 2019

[29] Xuetong Niu, Li Wang, Xulei Yang, "A Comparison Study of Credit Card Fraud Detection: Supervised versus Unsupervised", arXiv:1904.10604v1 [cs.LG] 24 Apr 2019

[30] Maja Puh and Ljiljana Brkić., "Detecting Credit Card Fraud Using Selected Machine Learning Algorithms", 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics, 2019(MIPRO)

[31] Hordri, N., Sophiayati, S., Firdaus, N., & Mariyam, S. "Handling Class Imbalance in Credit Card Fraud using Resampling Methods. International Journal of Advanced Computer Science and Applications", 2018, 9. 10.14569/IJACSA.2018.091155.

[32] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986). https://doi.org/10.1038/323533a0

Appendix A

Code in Python

```
path_to_file = 'data/creditcard.csv'

# Libraries to be imported
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn import metrics
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

from sklearn.preprocessing import MinMaxScaler
from collections import Counter

import torch

# to make sure all the plots remain in the ipython notebook
get_ipython().run_line_magic('matplotlib', 'inline')

Data Preprocessing

# To read data and checkout first five rows
data = pd.read_csv('/Users/vrushalshah/Desktop/Thesis/Data Set/creditcard.csv')
data.head()

# Plot to visualize imbalance of the class
plt.hist(data['Class'], color='red')
plt.xlabel('Class')
plt.ylabel('Transaction')
plt.title('Class Imbalance', fontsize=15)

# To calculate percentage of fraud transactions
pc_fraud = len(data.loc[data['Class'] == 1].values)/len(data.loc[data['Class'] == 0].va
lues)
pc_fraud

```

```
# To visualize the withdrawal pattern in fraud transactions
isFraudAmt = data.loc[data['Class']==1]['Amount']

plt.hist(isFraudAmt, color='green')
plt.xlabel('Amount')
plt.ylabel('No. of Transaction')
plt.title('Fraudulent Transaction')

# To contrast wrt to the above pattern chart
notFraudAmt = data.loc[data['Class']==0]['Amount']

plt.hist(notFraudAmt, color='yellow')
plt.hist(isFraudAmt, color='green')
plt.xlabel('Amount')
plt.ylabel('No. of Transaction')
plt.title('Non-Fraudulent Transaction')

# To verify if the amount of withdrawal represents any bias towards any class
#(Found no significant bias)
s = sns.boxplot(x="Class", y="Amount", hue="Class",data=data,
                palette="Set1",showfliers=False).set_title("Transaction Amount")

# Used for Data Normalization to rectify any skewed values
from sklearn.preprocessing import RobustScaler
rob_scaler = RobustScaler()

data['Norm_Amount'] = rob_scaler.fit_transform(data['Amount'].values.reshape(-1,1))
data['Norm_Time'] = rob_scaler.fit_transform(data['Time'].values.reshape(-1,1))

data.drop(['Time','Amount'], axis=1, inplace=True)
data = data.sample(frac=1)

# Function to plot Receiver Operating Characteristics
def plot_roc_curve(fpr, tpr):
    plt.plot(fpr, tpr, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

```
77.
78. # Extracting feature set (x-values) and target(y-values) from the data
79. x = data.drop('Class', axis=1)
80. y = data['Class']
81.
82. # Applying primary component analysis to reduce feature the dimension from 30 --> 2
83. from sklearn.decomposition import PCA
84. pca = PCA(n_components=2)
85. pca.fit(x)
86. x_pca = pca.transform(x)
87.
88. # Plotting the 2D data after applying PCA
89. plt.figure(figsize=(5, 5))
90. plt.scatter(x_pca[:,0], x_pca[:,1], c=y, edgecolor='none',
91.         cmap=plt.cm.get_cmap('Spectral', 10))
92. plt.xlabel('component 1')
93. plt.ylabel('component 2')
94. plt.title('PCA on Original Data')
95. plt.colorbar()
96.
97.
98. # Diffrentiating between fraudulent and non-fraudulent classes
99. # Under Sampling the majority class
100.       fraud_data = data.loc[data['Class']==1]
101.       not_fraud_data = data.loc[data['Class']==0][:492]
102.       under_data = pd.concat([fraud_data, not_fraud_data])
103.       under_data = under_data.sample(frac=1, random_state=0)
104.
105.       # To visualize the regained class balance
106.       plt.hist(under_data['Class'])
107.       plt.xlabel('Class')
108.       plt.ylabel('Frequency')
109.       plt.title('Under Sampled Data')
110.
111.       # Confusion Matrix for the original data shows how the data is skewed towards on
    e class only
112.       plt.figure(figsize=(24,20))
113.       original_corr = data.corr()
114.       sns.heatmap(original_corr, cmap='Blues')
115.       plt.figure('Confusion Matrix for Orginal Data')
116.
```

```
117.        # Confusion Matrix for the under sampled data shows how the data is balanced tow
    ards both the classes
118.        plt.figure(figsize=(24,20))
119.        under_corr = under_data.corr()
120.        sns.heatmap(under_corr, cmap='Blues')
121.        plt.title('Confusion Matrix for Undersampled Data')
122.        plt.show()
123.
124.        # Extracting the "feature set" and the  "targets" for model training
125.        x = under_data.drop('Class', axis=1)
126.        y = under_data['Class']
127.
128.        # Applying "primary component analysis" to reduce feature the dimension from 30
    --> 2
129.        from sklearn.decomposition import PCA
130.        pca = PCA(n_components=2)
131.        pca.fit(x)
132.        x_pca = pca.transform(x)
133.
134.        # Plotting the 2D data after applying PCA
135.        plt.figure(figsize=(5, 5))
136.        plt.scatter(x_pca[:,0], x_pca[:,1], c=y, edgecolor='none',
137.                cmap=plt.cm.get_cmap('Spectral', 10))
138.        plt.xlabel('component 1')
139.        plt.ylabel('component 2')
140.        plt.title('PCA on Undersampled Data')
141.        plt.colorbar()
142.
143.        # Applying TSNE [T-Distributed Stochastic Neighbouring Entities]
144.        # Better suited to visualize high-dimensional data
145.        # Follows a probabilistic approach
146.        from sklearn.manifold import TSNE
147.        x_tsne = TSNE(n_components=2, random_state=0).fit_transform(x)
148.
149.        # Visualization of data using t-SNE approach
150.        plt.figure(figsize=(5, 5))
151.        plt.scatter(x_tsne[:,0], x_tsne[:,1], c=y, edgecolor='none',
152.                cmap=plt.cm.get_cmap('Spectral', 10))
153.        plt.xlabel('component 1')
154.        plt.ylabel('component 2')
155.        plt.title('t-SNE on Undersampled Data')
```

```
        plt.colorbar()

        Differentiating between fraudulent and non-fraudulent classes


        not_fraud = data.loc[data['Class']==0]
        is_fraud = data.loc[data['Class']==1]
        not_fraud = not_fraud[:(len(not_fraud)//2)]  # Taking floor values
        is_fraud = is_fraud[:(len(is_fraud)//2)]  # Taking floor values

        new_data = pd.concat([not_fraud, is_fraud])
        new_data = new_data.sample(frac=1, random_state=0)

        # To calculate the percentage of minority class (fraudulent transactions)
        (len(new_data.loc[new_data['Class']==1])/len(new_data.loc[new_data['Class']==0])
    )*100

        x = new_data.drop('Class', axis=1)
        y = new_data['Class']

        # Train - Test split for training the models
        xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.3, random_stat
    e=0)

        Applying SMOTE for oversampling of the minority class

        sm = SMOTE(random_state = 0)
        xtrain_over, ytrain_over = sm.fit_sample(xtrain, ytrain.ravel())

        # Applying Primary Component Analysis to reduce the dimension of the feature set
     from 30 --> 2
        from sklearn.decomposition import PCA
        pca = PCA(n_components=2)
        pca.fit(xtrain_over)
        x_pca = pca.transform(xtrain_over)

        # 2D plot representing the data after applying PCA
        plt.figure(figsize=(5, 5))
        plt.scatter(x_pca[:,0], x_pca[:,1], c=ytrain_over, edgecolor='none',
                    cmap=plt.cm.get_cmap('Spectral', 10))
        plt.xlabel('component 1')
```

```
194.        plt.ylabel('component 2')
195.        plt.title('PCA on Oversampled Data')
196.        plt.colorbar()
197.
198.        # Converting the extracted "feature set" and "target values" to pandas dataframe
    for processing
199.        x_df = pd.DataFrame(xtrain_over)
200.        y_df = pd.DataFrame(ytrain_over)
201.
202.        x_df['Class'] = y_df # Adding the Class value to the dataframe
203.        over_data = x_df
204.
205.        # Histogram plot to show the regained class balance achieved after applying SMOT
    E
206.        plt.hist(over_data['Class'])
207.        plt.xlabel('Class')
208.        plt.ylabel('Frequency')
209.        plt.title('Oversampled Data')
210.
211.        # Confusion Matrix of the Oversampled data to show how SMOTE results in balanced
    classes
212.        plt.figure(figsize=(24,20))
213.        over_corr = over_data.corr()
214.        sns.heatmap(over_corr, cmap='Blues')
215.        plt.title('Confusion Matrix for Oversampled Data')
216.        plt.show()
217.        /* After Undersampling the classification is done seperatly
218.
219.        Applying CMTNN to Minority Class
220.        # Creating Torch dataset for training Truth NN
221.        train_ds_tnn = torch.utils.data.TensorDataset(torch.tensor(xtrain).float(), torc
    h.tensor(ytrain).float())
222.        valid_ds_tnn = torch.utils.data.TensorDataset(torch.tensor(xtest).float(), torch
    .tensor(ytest).float())
223.
224.        # Creating Torch dataloader for training Truth NN
225.        bs =128 # Batch Size
226.        train_dl_tnn = torch.utils.data.DataLoader(train_ds_tnn, batch_size=bs)
227.        valid_dl_tnn = torch.utils.data.DataLoader(valid_ds_tnn, batch_size=bs)
228.
229.        # Static Model Parameters
```

```
        n_input = xtrain.shape[1]
        n_output = 1
        n_hidden = 20

        # Initializing the Truth NN model
        TNN = Classifier(n_input=n_input,n_hidden=n_hidden,n_output=n_output,drop_prob=0.2)

        # Static Training Parameters
        lr = 0.005   # Learning Rate
        pos_weight = torch.tensor([5])  # weight adjustment for class imbalance
        opt = torch.optim.SGD(TNN.parameters(), lr=lr, momentum=0.9) #Optimizer
        loss_func = torch.nn.BCEWithLogitsLoss(pos_weight=pos_weight)  #Loss function
        #loss_func = torch.nn.BCELoss()
        n_epoch = 50  # No. of epochs to run training

        # Start Training
        train_loss_tnn, val_loss_tnn = train(n_epoch,TNN,loss_func,opt,train_dl_tnn,valid_dl_tnn)


        # Preparing complement targets for "Falsity NN"
        ytrain_fnn = 1-ytrain
        ytest_fnn = 1-ytest

        # Creating Torch dataset for training Falsity NN
        train_ds_fnn = torch.utils.data.TensorDataset(torch.tensor(xtrain).float(), torch.tensor(ytrain_fnn).float())
        valid_ds_fnn = torch.utils.data.TensorDataset(torch.tensor(xtest).float(), torch.tensor(ytest_fnn).float())

        # Creating Torch dataloader for training Falsity NN
        bs =128 # Batch Size
        train_dl_fnn = torch.utils.data.DataLoader(train_ds_fnn, batch_size=bs)
        valid_dl_fnn = torch.utils.data.DataLoader(valid_ds_fnn, batch_size=bs)

        # Static Model Parameters
        n_input = xtrain.shape[1]
        n_output = 1
        n_hidden = 20

```

```
        # Initializing the "Falsity NN" model
        FNN = Classifier(n_input=n_input,n_hidden=n_hidden,n_output=n_output,drop_prob=0.2)
        # Start Training
        train_loss_fnn, val_loss_fnn = train(n_epoch,FNN,loss_func,opt,train_dl_fnn,valid_dl_fnn)


        Combining Truth Neural Network and Falsity Neural Network

        M_intersection = list(set(M_fnn).intersection(M_tnn))
        M_union = list(set(M_fnn).union(M_tnn))


        xtrain_cm = np.delete(xtrain, M_union, axis=0)
        ytrain_cm = np.delete(ytrain, M_union, axis=0)









        CMTNN + SMOTE Approach

        # Creating Torch dataset for training Truth NN
        train_ds_tnn = torch.utils.data.TensorDataset(torch.tensor(xtrain).float(), torch.tensor(ytrain).float())
        valid_ds_tnn = torch.utils.data.TensorDataset(torch.tensor(xtest).float(), torch.tensor(ytest).float())

        # Creating Torch dataloader for training Truth NN
        bs =128 # Batch Size
        train_dl_tnn = torch.utils.data.DataLoader(train_ds_tnn, batch_size=bs)
        valid_dl_tnn = torch.utils.data.DataLoader(valid_ds_tnn, batch_size=bs)

        # Static Model Parameters
        n_input = xtrain.shape[1]
        n_output = 1
        n_hidden = 20
```

```
304.
305.       # Initializing the Truth NN model
306.       TNN = Classifier(n_input=n_input,n_hidden=n_hidden,n_output=n_output,drop_prob=0
   .2)
307.
308.       # Static Training Parameters
309.       lr = 0.005   # Learning Rate
310.       pos_weight = torch.tensor([5])  # weight adjustment for class imbalance
311.       opt = torch.optim.SGD(TNN.parameters(), lr=lr, momentum=0.9) #Optimizer
312.       loss_func = torch.nn.BCEWithLogitsLoss(pos_weight=pos_weight)  #Loss function
313.       #loss_func = torch.nn.BCELoss()
314.       n_epoch = 50  # No. of epochs to run training
315.
316.       # Start Training
317.       train_loss_tnn, val_loss_tnn = train(n_epoch,TNN,loss_func,opt,train_dl_tnn,vali
   d_dl_tnn)
318.
319.       # Plot to visualize training loss
320.       plt.plot(train_loss_tnn)
321.       plt.xlabel('Epochs')
322.       plt.ylabel('Training Loss')
323.       plt.title('Training Loss Characteristics (Truth NN)')
324.       plt.show()
325.
326.       # Plot to visualize validation loss
327.       plt.plot(val_loss_tnn)
328.       plt.xlabel('Epochs')
329.       plt.ylabel('Validation Loss')
330.       plt.title('Validation Loss Characteristics (Truth NN)')
331.       plt.show()
332.
333.       # Using Truth NN to get classification on training set
334.       ytrain_pred_tnn = TNN(torch.tensor(xtrain).float()).detach().numpy()
335.       ytrain_pred_tnn = mat_normalize(ytrain_pred_tnn)
336.
337.       plt.hist(ytrain_pred_tnn, bins=20)
338.
339.       # Adjusting predictions to get binary output
340.       ytrain_pred_tnn [ytrain_pred_tnn>=0.5] =1.0
341.       ytrain_pred_tnn [ytrain_pred_tnn<0.5] =0.0
342.
```

```
        # Confusion matrix to visualize "TP", "FP", "TN", "FN"
        #
        # [TN FP]
        # [FN TP]
        print('Confusion matrix: {}'. format(metrics.confusion_matrix(ytrain, ytrain_pred_tnn, labels=[0, 1])))

        # M_tnn = Misclassified samples using Truth NN
        comb_tnn = ytrain_pred_tnn - ytrain
        M_tnn = np.where(comb_tnn == 1)
        M_tnn = list(M_tnn[0])
        print(len(M_tnn))

        # Preparing complement targets for "Falsity NN"
        ytrain_fnn = 1-ytrain
        ytest_fnn = 1-ytest

        # Creating Torch dataset for training Falsity NN
        train_ds_fnn = torch.utils.data.TensorDataset(torch.tensor(xtrain).float(), torch.tensor(ytrain_fnn).float())
        valid_ds_fnn = torch.utils.data.TensorDataset(torch.tensor(xtest).float(), torch.tensor(ytest_fnn).float())

        # Creating Torch dataloader for training Falsity NN
        bs =128 # Batch Size
        train_dl_fnn = torch.utils.data.DataLoader(train_ds_fnn, batch_size=bs)
        valid_dl_fnn = torch.utils.data.DataLoader(valid_ds_fnn, batch_size=bs)

        # Static Model Parameters
        n_input = xtrain.shape[1]
        n_output = 1
        n_hidden = 20

        # Initializing the "Falsity NN" model
        FNN = Classifier(n_input=n_input,n_hidden=n_hidden,n_output=n_output,drop_prob=0.2)

        # Static Training Parameters
        lr = 0.005   # Learning Rate
        pos_weight = torch.tensor([1/5])  # weight adjustment for class imbalance
        opt = torch.optim.SGD(FNN.parameters(), lr=lr, momentum=0.9) #Optimizer
```

```
380.        loss_func = torch.nn.BCEWithLogitsLoss(pos_weight=pos_weight)  #Loss function
381.        #loss_func = torch.nn.BCEWithLogitsLoss()
382.        #loss_func = torch.nn.BCELoss()
383.        n_epoch = 50  # No. of epochs to run training
384.
385.        # Start Training
386.        train_loss_fnn, val_loss_fnn = train(n_epoch,FNN,loss_func,opt,train_dl_fnn,vali
    d_dl_fnn)
387.
388.        # Plot to visualize training loss
389.        plt.plot(train_loss_fnn)
390.        plt.xlabel('Epochs')
391.        plt.ylabel('Training Loss')
392.        plt.title('Training Loss Characteristics (Falsity NN)')
393.        plt.show()
394.
395.        # Plot to visualize validation loss
396.        plt.plot(val_loss_fnn)
397.        plt.xlabel('Epochs')
398.        plt.ylabel('Validation Loss')
399.        plt.title('Validation Loss Characteristics (Falsity NN)')
400.        plt.show()
401.
402.        # Using the Falsity NN to get misclassified samples
403.        ytrain_pred_fnn = FNN(torch.tensor(xtrain).float()).detach().numpy()
404.        ytrain_pred_fnn = mat_normalize(ytrain_pred_fnn)
405.
406.        plt.hist(ytrain_pred_fnn, bins=20)
407.
408.        # Adjusting the predictions to output binary predictions
409.        ytrain_pred_fnn [ytrain_pred_fnn>=0.5] =1.0
410.        ytrain_pred_fnn [ytrain_pred_fnn<0.5] =0.0
411.
412.        # Confusion matrix to visualize "TP", "FP", "TN", "FN"
413.        #
414.        # [TN FP]
415.        # [FN TP]
416.        print('Confusion matrix: {}'. format(metrics.confusion_matrix(ytrain_fnn, ytrain
    _pred_fnn, labels=[0, 1])))
417.
418.
```

```
        #M_fnn = List of samples misclassified by Falsity NN
        comb_fnn = ytrain_fnn - ytrain_pred_fnn
        M_fnn = np.where(comb_fnn == 1)
        M_fnn = list(M_fnn[0])
        print(len(M_fnn))

        Combining Truth NN and Falsity NN

        M_intersection = list(set(M_fnn).intersection(M_tnn))
        M_union = list(set(M_fnn).union(M_tnn))

        print(len(M_union))
        print(len(M_intersection))

        # Finally removing the union of misclassified samples under-
    sample majority class
        # Please note that the minority class samples are left untouched
        xtrain_cm = np.delete(xtrain, M_union, axis=0)
        ytrain_cm = np.delete(ytrain, M_union, axis=0)
        print('Train dataset shape after CMTNN %s' % Counter(ytrain_cm))

        print('Original train dataset shape %s' % Counter(ytrain))
        print('CMTNN train dataset shape %s' % Counter(ytrain_cm))
        print('Original test dataset shape %s' % Counter(ytest))

        Applying SMOTE

        # Using SMOTE(Applied on the training set only) to over-
    sample the minority class
        sm = SMOTE(random_state = 1)
        xtrain_cm_sm, ytrain_cm_sm = sm.fit_sample(xtrain_cm, ytrain_cm.ravel())

        print('Original train dataset shape %s' % Counter(ytrain))
        print('Resampled train dataset shape %s' % Counter(ytrain_cm_sm))
        print('Original validation dataset shape %s' % Counter(ytest))

        # Applying Primary Component Analysis to reduce the dimension of the feature set
     from 30 --> 2
        from sklearn.decomposition import PCA
        pca = PCA(n_components=2)
        pca.fit(xtrain_cm_sm)
```

```
457.        x_pca = pca.transform(xtrain_cm_sm)
458.
459.        # 2D plot representing the data after applying PCA
460.        plt.figure(figsize=(5, 5))
461.        plt.scatter(x_pca[:,0], x_pca[:,1], c=ytrain_cm_sm, edgecolor='none',
462.                cmap=plt.cm.get_cmap('Spectral', 10))
463.        plt.xlabel('component 1')
464.        plt.ylabel('component 2')
465.        plt.title('PCA on Oversampled Data')
466.        plt.colorbar()
467.
468.        # Converting the extracted "feature set" and "target values" to pandas dataframe
    for processing
469.        x_df = pd.DataFrame(xtrain_cm_sm)
470.        y_df = pd.DataFrame(ytrain_cm_sm)
471.
472.        x_df['Class'] = y_df # Adding the Class value to the dataframe
473.        over_data = x_df
474.
475.        # Histogram plot to show the regained class balance achieved after applying SMOT
   E
476.        plt.hist(over_data['Class'])
477.        plt.xlabel('Class')
478.        plt.ylabel('Frequency')
479.        plt.title('Oversampled Data')
480.
481.        # Confusion Matrix of the Oversampled data to show how SMOTE results in balanced
    classes
482.        plt.figure(figsize=(24,20))
483.        over_corr = over_data.corr()
484.        sns.heatmap(over_corr, cmap='Blues')
485.        plt.title('Confusion Matrix for Oversampled Data')
486.        plt.show()
487.
488.        Classification with CMTNN and SMOTE
489.
490.        SVM
491.
492.        from sklearn.svm import SVC
493.        svc = SVC()
494.
```

```
svc.fit(xtrain_cm_sm, ytrain_cm_sm)
ypred = svc.predict(xtest)

print('AUPRC score: {}'. format(metrics.average_precision_score(ytest, ypred)))

print('AUROC score: {}'.format(metrics.roc_auc_score(ytest, ypred)))
print('Accuracy score: {}'.format(metrics.accuracy_score(ytest, ypred)))
print(metrics.classification_report(ytest, ypred))

yscore = svc.decision_function(xtest)
fpr, tpr, thresholds = metrics.roc_curve(ytest, yscore)
plot_roc_curve(fpr, tpr)

Random Forest

from sklearn.ensemble import RandomForestClassifier
random_clf = RandomForestClassifier(random_state=0)

random_clf.fit(xtrain_cm_sm, ytrain_cm_sm)
ypred = random_clf.predict(xtest)

print('AUPRC score: {}'. format(metrics.average_precision_score(ytest, ypred)))

print('AUROC score: {}'.format(metrics.roc_auc_score(ytest, ypred)))
print('Accuracy score: {}'.format(metrics.accuracy_score(ytest, ypred)))
print(metrics.classification_report(ytest, ypred))

yscore = random_clf.predict_proba(xtest)
fpr, tpr, thresholds = metrics.roc_curve(ytest, yscore[:,1])
plot_roc_curve(fpr, tpr)

XGBoost

from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(xtrain_cm_sm, ytrain_cm_sm)

xtest_df = pd.DataFrame(xtest)
xtest_df.columns = [i for i in range(0, xtest_df.shape[1])]
xtest = xtest_df.values
ypred = xgb.predict(xtest)
```

```
534.
535.       print('AUPRC score: {}'. format(metrics.average_precision_score(ytest, ypred)))
536.       print('AUROC score: {}'.format(metrics.roc_auc_score(ytest, ypred)))
537.       print('Accuracy score: {}'.format(metrics.accuracy_score(ytest, ypred)))
538.       print(metrics.classification_report(ytest, ypred))
539.
540.       yscore = xgb.predict_proba(xtest)
541.       fpr, tpr, thresholds = metrics.roc_curve(ytest, yscore[:,1])
542.       plot_roc_curve(fpr, tpr)
543.
544.       ANN with CMTNN and SMOTE
545.
546.       class Classifier(torch.nn.Module):
547.           def __init__(self, n_input=10, n_hidden = 20, n_output = 1,drop_prob=0.5):
548.               super().__init__()
549.               self.extractor1 = torch.nn.Linear(n_input, n_hidden)
550.               self.extractor2 = torch.nn.Linear(n_hidden, n_hidden)
551.               self.relu = torch.nn.ReLU()
552.               self.drop_out = torch.nn.Dropout(drop_prob)
553.               self.classifier = torch.nn.Linear(n_hidden, n_output)
554.
555.
556.           def forward(self, xb):
557.               x = self.relu(self.extractor1(xb))
558.               x = self.relu(self.extractor2(x))
559.               x = self.drop_out(x)
560.               return self.classifier(x).squeeze()
561.
562.       def loss_batch(model, loss_func, xb, yb, opt=None):
563.           loss = loss_func(model(xb), yb)
564.
565.           if opt is not None:
566.               loss.backward()
567.               opt.step()
568.               opt.zero_grad()
569.
570.           return loss.item(), len(xb)
571.
572.       def train(epochs, model, loss_func, opt, train_dl, valid_dl):
573.           train_loss = []
```

```
574.           validation_loss = []
575.           for epoch in range(epochs):
576.               model.train()
577.               count = 0
578.               loss = 0
579.               for xb, yb in train_dl:
580.                 losses, nums = loss_batch(model, loss_func, xb, yb, opt)
581.                 loss += losses*nums
582.                 count+=nums
583.               train_loss.append(loss/count)
584.         model.eval()
585.               with torch.no_grad():
586.                   losses, nums = zip(
587.                       *[loss_batch(model, loss_func, xb, yb) for xb, yb in valid_dl]
588.                   )
589.               val_loss = np.sum(np.multiply(losses, nums)) / np.sum(nums)
590.               validation_loss.append(val_loss)
591.
592.               print(epoch, val_loss)
593.
594.           return train_loss, validation_loss
595.
596.       #Normalizing function [Rescales values of an array to fit in [0,1]]
597.       def mat_normalize(arr):
598.         arr = arr - arr.min()
599.         arr = arr / arr.max()
600.         return arr
601.
602.       # Creating Torch dataset for training
603.       train_ds = torch.utils.data.TensorDataset(torch.tensor(xtrain).float(), torch.te
    nsor(ytrain).float())
604.       valid_ds = torch.utils.data.TensorDataset(torch.tensor(xtest).float(), torch.ten
    sor(ytest).float())
605.
606.       # Creating Torch dataloader for training
607.       bs =128 # Batch Size
608.       train_dl = torch.utils.data.DataLoader(train_ds, batch_size=bs)
609.       valid_dl = torch.utils.data.DataLoader(valid_ds, batch_size=bs)
610.
611.       # Static Model Parameters
612.       n_input = xtrain.shape[1]
```

```
613.        n_output = 1
614.        n_hidden = 20
615.
616.        # Initializing the model
617.        model = Classifier(n_input=n_input,n_hidden=n_hidden,n_output=n_output,drop_prob
    =0.2)
618.
619.        # Static Training Parameters
620.        lr = 0.005   # Learning Rate
621.        pos_weight = torch.tensor([5])  # weight adjustment for class imbalance
622.        opt = torch.optim.SGD(model.parameters(), lr=lr, momentum=0.9) #Optimizer
623.        loss_func = torch.nn.BCEWithLogitsLoss(pos_weight=pos_weight)  #Loss function
624.        n_epoch = 50  # No. of epochs to run training
625.
626.        # Start Training
627.        train_loss, val_loss = train(n_epoch,model,loss_func,opt,train_dl,valid_dl)
628.
629.        # Plot to visualize training loss
630.        plt.plot(train_loss)
631.        plt.xlabel('Epochs')
632.        plt.ylabel('Training Loss')
633.        plt.title('Training Loss Characteristics')
634.        plt.show()
635.
636.        # Plot to visualize validation loss
637.        plt.plot(val_loss)
638.        plt.xlabel('Epochs')
639.        plt.ylabel('Validation Loss')
640.        plt.title('Validation Loss Characteristics')
641.        plt.show()
642.
643.        # Using the trained model for prediction
644.        ypred = model(torch.tensor(xtest).float()).detach().numpy()
645.        ypred = mat_normalize(ypred)
646.        ypred [ypred>=0.5] =1.0
647.        ypred [ypred<0.5] =0.0
648.
649.        # Confusion matrix to visualize "TP", "FP", "TN", "FN"
650.        #
651.        # [TN FP]
652.        # [FN TP]
```

```
653.        print(pos_weight)
654.        print('Confusion matrix: {}'. format(metrics.confusion_matrix(ytest, ypred, labe
    ls=[0, 1])))
655.
656.        # Area Under "Precision-Recall Curve"
657.        print('AUPRC score: {}'. format(metrics.average_precision_score(ytest, ypred)))


658.
659.        # Area Under "ROC Curve" [ROC: Reciever-OPerating Characteristics (TP vs FP)]
660.        print('AUROC score: {}'.format(metrics.roc_auc_score(ytest, ypred)))
661.
662.        # Accuracy of the model
663.        print('Accuracy score: {}'.format(metrics.accuracy_score(ytest, ypred)))
664.
665.        print(metrics.classification_report(ytest, ypred))
```