Thesis for Master of Engineering

Credit Card Fraud Detection using Imbalanced Data Classification

December 2020

Graduate School
Kumoh National Institute of Technology

Department of IT Convergence Engineering

ABBAS QALAB E

Thesis for Master of Engineering

Credit Card Fraud Detection using Imbalanced Data Classification

December 2020

Graduate School
Kumoh National Institute of Technology

Department of IT Convergence Engineering

ABBAS QALAB E

Credit Card Fraud Detection using Imbalanced Data Classification

Supervisor: Jang Sung Bong

This Thesis Presented for the
Master of Engineering

December 2020

Graduate School
Kumoh National Institute of Technology

Department of IT Convergence Engineering

ABBAS QALAB E

Approval of the Thesis for the Master of Engineering Submitted by Qalab E Abbas

December 2020

Chair of Committee Wansu Lim (seal)

Committee Jae Min Lée (seal)

Committee Sung Bong Jang (seal)

Graduate School

Kumoh National Institute of Technology

Credit Card Fraud Detection using Imbalanced Data Classification

ABBAS QALAB E

Department of IT Convergence Engineering,
Graduate School
Kumoh National Institute of Technology

Abstract

The problem of fraudulent credit card transactions, in this era of online shopping and data overload is overwhelming. Today, it's not very easy to be sure of how safe your information and the information about your credit card is. There are a million ways that your credit card can be stolen and be used without your knowledge until it's too late. Everyday there are reports of credit card information being sold on the dark web. This is very difficult for the customers, but it can be very financially taxing for the credit card companies as well. The thesis proposes a fast and reliable solution based on Machine Learning to detect if a transaction is fraudulent or non-fraudulent so the transaction can be flagged. The problem when dealing with Credit Card transaction dataset is the highly skewed distribution of class labels. We deal with class imbalance and provide a framework that can be used for skewed datasets. Our dataset contains around 99% non-fraud cases

and 0.17% fraud cases. Dealing with highly imbalanced data is different so we first use different sampling techniques to balance out the class distribution without losing much information and then use the best performing machine learning algorithms to get best possible results. The proposed method generates great results but does it in much less time since time is very crucial in this case, we can say that the proposed approach is definitely a step in the right direction.

불균형 데이터 분류를 사용한 신용카드 사기 감지

ABBAS QALAB E

금오공과대학교 대학원 IT융복합공학과

요 약

온라인 쇼핑 및 데이터 시대의 사기 카드 거래 문제는 매우 큰 문제이다. 오늘날, 신용카드에 대한 정보가 얼마나 안전한가 믿기는 쉽지 않다. 너무 늦은 시간까지, 신용카드를 도난당 할 수 있는 방법은 수만가지이다. 매일 '다크 웹' 상에서 신용카드정보가 판매된다는 보고가 있다. 이것은 고객들에게는 좋지 못한 상황이고, 신용카드 회사에도 상당한 세금이 부과될 수 있다. 본 논문은 신용거래가 불법적인 거래인지 아닌지를 포착하여 표시할 수 있는 머신러닝에 기반한 빠르고 신뢰도가 높은 해결책을 제시한다. 신용카드 거래의 데이터 집합을 다룰 때 문제는 클래스 라벨의 높은 왜곡된 분포이다. 우리는 클래스 불균형을 다루며 왜곡된 데이터 집합에 사용될 수 있는 틀을 제공한다. 우리의 데이터 집합은 약 99%의 정상적인 경우와 0.17%의 사기 경우를 포함한다. 매우 불균형한 데이터를 다루는 것은 다르기

때문에 우리는 우선 서로 다른 샘플링 기술을 사용하여 클래스 분포의 많은 정보를 잃지 않고 균형있게 처리한 다음 가장 최상의 결과를 얻기위해 최고의 기계 학습 알고리즘을 사용한다. 제안된 방법은 좋은 결과를 발생시킨다. 하지만 이러한 경우에는 시간이 매우 중요하기 때문에 훨씬 짧은 시간 내에 제안된 접근방식이 올바른 방향으로의 확고한 단계라고 말할 수 있다.

Contents

[List of Figures]

[List of Tables]

[List of Abbreviations]

ML	Machine Learning
ANN	Artificial Neural Network
RF	Random Forest
LR	Logistic Regression
XBG	XGBoost
RUS	Random Under Sampling
NM	Near Miss Undersampling
CNN	Condensed Nearest Neighbor Rule
ENN	Edited Nearest Neighbors Rule
TL	Tomek Links Undersampling
NCR	Neighborhood Cleaning Rule
OSS	One-Sided Selection
SMOTE	Synthetic Minority Oversampling Technique
B-SMOTE	Borderline SMOTE
ADASYN	Adaptive Synthetic Sampling
SVM-SMOTE	Borderline Oversampling with SVM

[Acknowledgement]

First and foremost, I would like to thank The Almighty Allah for giving me strength and making me capable of achieving this milestone in life. Without His will, nothing would have been possible and verily, to Him we return.

Next, I would like to express my deep gratitude to my academic supervisor, Prof. Jang, for giving me this opportunity to work under his supervision and giving me guidance to carry out research. It was a privilege and honor to work under him and learn the axels of both research and life. I will surely miss every moment spent with him both professionally and as a mentor. I hope I have come upon your expectations, professor.

I would also like to thank all my brothers in arms who have been with me through thick and thin during these 2 years. I can never forget the 1am walks for ice cream that we have made, and I hope that someday, somewhere in life, we can repeat it again.

Last but not the least, I would like to thank my parents for everything that they have done for me. Their support has always been the key to my success. I owe it all to them and I wish that I can make them proud someday. Their happiness is everything to me.

Chapter 1

Introduction

With the advancement in Ecommerce, the number of online shoppers is growing exponentially and so is the need for a fast and reliable transaction authentication system. In this age of internet more and more businesses are setting up online shops to make it easier for their customers. Although there is no doubt that this has made the life of so many people easier but there are some exceptions as well. The use of Credit Cards online has also made it easier for identity thieves and other parties to use this as an opportunity to harm other people financially. Credit card scams and frauds are so common these days that it is making the life of customers and credit card companies very difficult. Despite the different security measures the credit card fraud is still on the rise.

Some of the already existing solutions which generate a higher fraud detection rate involve asking the user so many questions that the drop rate of customers at the checkout is higher than usual which is also affecting the business. We need a smart fraud detection system that can be updated with time and handle and predict the fraudulent transaction whenever it happens. This thesis attempts to take a step in the direction of smart fraud detection using Machine Learning.

Machine Learning (ML) provides the system an ability to learn and improve from the data provided. ML consists of multiple algorithms like Logistic Regression, XGBoost, Random Forest and Artificial Neural Networks to name a few. Selection of the algorithm mainly depends on the nature of data, but there is no set rule, the most common practice is to try multiple algorithms and then choose the best one depending on the results.

The Credit Card Transaction dataset is available on Kaggle, it consists of credit card transactions made in September 2013. The goal is to detect fraudulent transaction from normal or non-fraudulent transactions. The dataset contains 284,807 total transaction with only 492 fraudulent ones. The fraudulent transactions only make only 0.17% of the entire dataset. Such imbalanced data cannot be directly used for learning purpose. There are many approaches to deal with this issue which we will discuss in this paper.

After preparing the dataset, it is divided into training and testing datasets. Training data is fed to our ML algorithms which will learn from the data and then should be able to classify the unseen transaction as fraudulent and non-fraudulent with greater accuracy.

Chapter 2

Related Work

Fraud detection is a hot topic in machine learning and a lot of researchers have worked on credit card fraud detection using all sorts of methods from supervised learning to unsupervised learning, Algorithm based approach to sampling-based approaches and some authentication-based approaches as well.

User Authentication approaches include password base authentication, physiological authentication, behavioral authentication and combined authentications which include a combination of afore mentioned approaches. Authentication based approaches can hinder the transaction before it happens, they are the first defense[1]-[4].

Supervised Machine learning offers many approaches for fraud detection. Especially logistic regression has been used for a long time because of its simplicity. However, these methods are now outdated and with complex data it is becoming increasingly difficult to use these techniques. Logistic Regression combined with a transaction aggregation is used to detect fraud detection in credit card transaction dataset [5]. Another research compares Neural Networks to Logistic Regression for fraud detection, results show that ANN gives better output on the test data as compared to LR [6]. It also shows overfitting of model with Logistic Regression. Logistic Regression against deep learning and gradient boosted tree, for a huge dataset of 80 million transaction, show how deep learning outperforms the other models in term of fraud detection [7]. LR shows worst performance, probably due to its inability to comprehend complex relationships in data.

Artificial Neural Networks are able to handle complex data because their depth can be increased according to the complexity of the data. If a dataset

is more complex, it is highly likely that a Neural Network with more hidden layers would provide better results than one with a smaller number of hidden layers. A deeper Neural Network would need more computational power than a shallow one, they are also prone to overfitting and underfitting Nonetheless ANN show great promise in fraud detection [8].

Decision Tree classifier have also been used for fraud detection. They are easy to understand and implement with very little computational power as compared to other algorithms. Their cons include inability to deal with skewed data and instability. In a study where DTs are compared against ANN and LR, ANN come at the top followed by LR and DTs show poor performance [9].

Decision Trees might not be best suited for imbalanced classification on their own but a variation of them known as Random Forest (RF) perform very well. RF overcomes the limitation of DTs by using an ensemble of decision trees, which makes RF more resilient against overfitting. Many studies compare the performance of RF against ANN, SVM, LR and KNN on real life credit card datasets and in most of these case RF outperforms the competition [10]– [11].

Chapter 3

Imbalanced Classification

3.1 Imbalanced Classification

A dataset with fewer examples of one case and a greater number of examples for the other case or cases is called imbalanced data. When such imbalanced data is used for classification purpose, we call such classification as Imbalanced classification.

For example, a dataset consists of plant specifications and has 80% of the examples of one type of plant and 20% of the examples are for second type of plant, the distribution of classes in this dataset is highly imbalanced. This would be an example of Imbalanced Classification.

Another example would the credit card transaction dataset, normal transactions make up 99.83% of the data and fraudulent transactions are only 0.017%. Credit card transaction dataset can be called highly skewed due to the severe imbalance of fraudulent and non-fraudulent cases.

As mentioned in the examples above the imbalance can vary from problem to problem, it can be a slight imbalance like in the case of plant types. On the other hand the imbalance can be severe as is the case in credit card transaction dataset[12].

A dataset with slight imbalance is not that difficult to handle and can be tackled with cost-sensitive classification algorithms. The problem arises when we have to deal with severe imbalance where 90% or more of the data is from one class like in the case of credit card transaction dataset. Machine Learning models alone are not enough to handle severe data imbalance,

some data preparation techniques are used to better prepare imbalanced datasets for the classification models.

In an imbalanced dataset, majority class is the one with higher number of examples and minority class has fewer number of examples. A dataset can have more than one majority classes but usually there is only one minority class which is typically of most interest. It is more important for a model to predict the label of minority class rather than the majority class.

Predicting the correct label for minority class is hard due to the fewer number of examples in the data. As in ML, models use the data to learn and then predict so it is understandable how it might be difficult for a model to learn from minority class, also the majority class can bury the minority class due to its size. Most classification algorithms are designed keeping in mind a balanced distribution of classes of data, a severe imbalance can negatively affect the learning of the model and generate poor results [13].

Imbalanced classification is not resolved yet, there are different approaches to tackle such problem which varies on the type of data. Even if we have more data with skewed distribution, or we use large and advanced techniques like neural network models and XGBoost, dealing with imbalanced data is still a challenge. One approach that can be used for many imbalanced classification problems is sampling as it deals with the data imbalance itself.

3.2 Sampling Imbalanced Data

The main issue when it comes to imbalanced classification is the skewed distribution of class labels. One way to handle imbalanced classification is to remove or reduce the difference in class distributions. There are techniques which are specifically designed for this purpose, known as sampling techniques. Sampling is also one of the most common ways to handle the imbalanced classification tasks because it is easy to understand

and practice. Another major reason why sampling is the better way to deal with imbalanced data is that after sampling the distribution of classes is made normal and now this sampled data is ready to be used by numerous machine learning algorithms without changing the algorithms to fit the imbalanced data. Simply put, instead of using a modified model to handle the imbalance we modify the dataset in such a way that the class distribution becomes balanced, doing this removes the basic issue of imbalance which effects the model training [14].

When it comes to application of sampling, it is applied only on the training dataset, the data which is used for model's learning. And not on the validation or test dataset. It is because the purpose of sampling is to only help the model with learning and if sampling is done on the entire dataset before train and test split it can cause leakage of information, which can result in model overfitting. An overfitted model will show results which are too good but don't actually reflect the model performance, this happens mainly due the information leakage between tarin and test dataset due to sampling the entire dataset. This is the reason why we should separate training dataset from testing data and apply sampling only on training dataset and evaluate the model on test dataset later on.[15].

The main purpose of sampling to even the class distribution, there are different ways to go about it for example. deleting exampling from majority class, adding exampling to minority class or even mixing both of these ways, next we will talk about these sampling techniques in detail.

3.2.1 Under Sampling

When we decrease the number of samples in majority class either by deleting some examples or by keeping them, it is known as undersampling. Some of the most commonly used undersampling methods are mentioned here.

- Random Undersampling
- Condensed Nearest Neighbor Rule (CNN)
- Near Miss Undersampling
- Tomek Links Undersampling (TL)
- Edited Nearest Neighbors Rule (ENN)
- One-Sided Selection (OSS)
- Neighborhood Cleaning Rule (NCR)

First the basic form of undersampling, it involves removing data from the majority class randomly, known as Random Undersampling. Though it is very fast and can balance the class distribution but there is a very high chance of losing valuable information as we are just randomly deleting data with any logic.

Next undersampling technique is used to reduce dataset size for K-NN classification. It generates a dataset which consists of examples from the original dataset and is capable of correctly classifying the original data using k=1 in K-NN algorithm, it is known as Condensed Nearest Neighbors (CNN). Since resulting subset is much smaller in size than the original hence this method decrease the execution time and reduced the space complexity [16].

Near Miss undersampling also uses K-NN algorithm to select cases from the majority class. It can be further divided depending on how we select the examples from the data. If we want to select examples from the majority class with minimum average distance to three of the closest examples from the minority class, it would be Near Miss-1. Near Miss-2 consists of examples from majority class with minimum average distance the three of the furthest examples from minority class and Near Miss-3 selects examples from majority class for each example in the minority class which is closest.

Tomek Links and Edited Nearest Neighbors Rule are undersampling techniques which pick examples to delete rather than keeping them like was the case in Random undersampling, Condensed Nearest Neighbor and Near Miss undersampling. The selected examples are usually the ones that are difficult to classify and create uncertainty to the decision boundary. Tomek Links is one the broadly used deletion undersampling technique and was proposed as an extension to CNN. Tomek Links consists of a pair of examples from the training dataset, where both of them belong to different classes and have minimum distance between them.

The second deletion type of undersampling is called Edited Nearest Neighbors rule. ENN uses k=3 in K-NN to identify the misclassified examples in the dataset and then removes them from the data. This step can be repeated numerous times which results in a much-refined dataset, this is called Repeated Edited Nearest Neighbors (RENN).

The last set of undersampling techniques are One-Sided Selection (OSS) and the Neighborhood Cleaning Rule (NCR), these techniques use both ways of undersampling by keeping some examples and by deleting some. These are basically the combination of two undersampling techniques, OSS combines TL and CNN [17] and NCR combines CNN with ENN. These combination techniques generally result in a cleaner and dense dataset.

3.2.2 Over Sampling

Oversampling involves increasing the examples in minority class to balance the class distribution. It is done either by duplicating the examples of minority class or by synthesizing new examples for the minority class. Broadly used oversampling techniques include.

- Random Oversampling
- Synthetic Minority Oversampling Technique (SMOTE)
- Borderline-SMOTE

- Borderline Oversampling with SVM
- Adaptive Synthetic Sampling (ADASYN)

Random Oversampling, just like the name suggests balances the class distribution by randomly replicating examples from the minority class hence increasing the number of examples in it. Just like Random Undersampling, Random oversampling is also a hit and miss type of algorithm because there is no logic behind it which can cause the model to overfit.

Synthetic Minority Oversampling Technique, also known as SMOTE is the most used and well-known sampling method. SMOTE doesn't just replicates the example but as its name suggests it generate new example based on the minority class. This is the overfitting problem caused by random oversampling can be avoided. SMOTE works by randomly selecting an example in the minority class then using K-NN a random neighbor is selected and new example is synthesized between these points in the sample space [18].

Other than SMOTE there are no new oversampling methods rather there are different variation of SMOTE. Borderline-SMOTE, first identifies only those examples of minority class which are misclassified by K-NN and then only generating examples like those [19]. Borderline Oversampling with SVM is another variation of SMOTE which uses examples near the support vectors to generate new examples.

ADASYN is a variation of SMOTE which is a little bit generalized, it also works by synthesizing example in minority class but how it goes about it is a bit different from regular SMOTE. ADASYN takes into consideration the density distribution of feature space when generating examples. So places where the density of minority class is low, it generates more examples there and vice versa.

3.2.3 Hybrid Sampling

Hybrid sampling involves applying one oversampling technique with one undersampling technique back-to-back on the same dataset. This combination has proved be more effective as compared to using one or the other sampling method. The dataset generated this way shows an overall better performance in terms of results and execution time as well. There are many combinations that can used for Hybrid Sampling, mentioned here are some of the most used techniques.

- Random Undersampling and SMOTE
- Tomek Links and SMOTE
- Edited Nearest Neighbors Rule and SMOTE
- OneSidedSelection and SMOTE

The sampling techniques can used in any order, but I prefer using oversampling after undersampling. This way makes the sampled dataset to have a more compact size and a balanced distribution. This method is more favorable when dealing with a huge dataset and large imbalance. Otherwise, if the dataset is smaller, we can use oversampling first to generate new examples and then use an undersampling technique to remove the noise from both of the classes. SMOTE is a very powerful oversampling method and it is often paired with an undersampling technique[20]. Tomek Link and Edited Nearest Neighbors Rule are the most used undersampling methods along with SMOTE. OneSidedSelection along with SMOTE also shows the most time efficient results while keeping the same F-score.

3.2.4 Ensemble Sampling

Ensemble sampling is the term I used for when I am using more than two sampling techniques, after going through all the above sampling techniques I can separate the efficient ones. Here I will try to combine those for best performance.

Here are the following techniques I have tried

- OSS+ TL+ SMOTE
- OSSx2+ SMOTE
- OSSx3+ SMOTE
- OSSx4+ SMOTE
- OSSx5+ SMOTE

Using OSS first reduces the size of data which makes it easier to implement other more time-consuming algorithms. I have not seen this applied before, but I think it can be useful in reducing the execution time.

3.3 Performance Metrics

An important factor when finalizing the model for machine learning is the selection of performance metrics. We need an evolution method which correctly measures the model's performance. There are many performance metrics available and used based on the type of data and problem. For classification problems, accuracy is the most widely used performance metric. Though accuracy can evaluate a classification model when the class distribution is even, but it does not convey the same message when dealing with imbalanced data. The reason why accuracy does not perform well for imbalanced data is simply because of the imbalanced distribution of classes. For example, our credit card dataset consists of 99.83% examples from the majority class, we can get 99% accuracy just by simply classifying all the

examples to majority class. In that way we a high accuracy score but clearly our model did not learn anything at all.

For imbalanced data, precision, recall and F-score are the most widely used performance metrics which give an accurate measure of model's learning ability [21][22].

Before talking about precision and recall it is important that we understand what true positive, true negative, false positive and false negative mean. Let's take the example of credit card data, where if the transaction is non-fraudulent then it is negative otherwise its positive. Usually the important class, fraud in our case, is the positive class. A true positive would be when a fraudulent transaction is correctly classified as that and a true negative

would be when a non-fraudulent transaction is correctly classified as non-fraudulent. If a fraudulent transaction is classified as non-fraudulent then that would be a false positive, whereas a false negative would be when a non-fraudulent transaction is classified as a fraudulent one.

Precision tells us how many of true positives were actually classified correctly. It is a ratio of true positive divided by the total examples in the dataset as in equation (1)[15].

$$\text{Precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}} \qquad (1)$$

Precision can vary from 0.0 to 1.0 with1.0 being when all the true positives are correctly classified. In python, precision score can be found out by using scikit-learn library [23].

Recall tells us about how many of the true positives were correctly classified out of all the possible true positive classifications. Recall shows the misclassified true positive as well unlike precision which only focuses on true positives [15] .

$$\text{Recall} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalseNegative}} \quad (2)$$

Recall can vary from 0.0 to 1.0 with 1.0 being when there are no false negatives. In python, recall score can be found out by using scikit-learn library [23].

Between precision and recall, there is usually always an inverse relationship, if we focus on increasing one the other will be negatively affected. For a model to be evaluated effectively it is important that we focus on both of these factors and finalize a model which gives us a good precision as well as recall score. A way to achieve this homogeneity between precision and recall is to use F-score.

F-Score gives a single value which takes into account both precision and recall. Because on their own, precision and recall are not capable of describing the entire story. If we focus on getting a good F-score, we can be assured that both precision and recall would be pretty good. F-score is calculated as described in equation (3)..

$$\text{F-Score} = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (3)$$

F-score is possibly the most common performance metrics used for imbalanced classification as it focuses on precision as well as recall and tries to get the best of both [15].

F-score can also vary from 0.0 to 1.0 with 1.0 being when all the classifications are made perfectly. In python, recall score can be found out by using scikit-learn library and f1 score() function. [23].

3.4 Machine Learning Algorithms

There are hundreds of ML algorithms which can be used depending on the situation. But it is not always clear or easy to find the best algorithm that

will give us the best results. The best approach is to test few of the most commonly used techniques and then decide on the final one depending on the results. Here we have tested following four algorithms.

1) Logistic Regression
2) Random Forest
3) XGBoost
4) Neural Networks

Logistic regression, a Machine Learning model that comes from statistics and is names after the main function that is used in it. The function is called logistic function also known as sigmoid function. Mathematical form of logistic regression is very similar to linear regression, the key difference logistic regression gives out binary output (0 or 1) whereas linear regression gives out numerical value [24].

$$y = \frac{e^{(b0+b1*x)}}{(1+e^{(b0+b1*x)})} \quad (4)$$

Equation (4) shows the mathematical form of Logistic Regression. Here y is the output, x is the input and b0 is the bias and b1 is the coefficient of input.

Random forest is not a new algorithm in itself, but it is an ensemble of multiple decision trees. Each of these decision trees work on its own and generates an output and the final outcome would follow the basic rule of the wisdom of the crowds. Basically the most common output of decision trees will be the final output of Random Forest [25].

XGBoost is also a decision tree in its core, but instead of regular decision trees, it uses gradient boosted trees. XGBoost stands for extreme gradient boosting, which pushes the model to its limit to generate better results with much more speed [26].

Artificial neural networks (ANN) are designed on the principle of human brain and how every neuron is connected to every other neuron in the brain and they can communicate with each other. This communication of between the nodes allows the ANN to learn things which are complicated for simple Machine learning algorithms [27].

If we look at the structure of ANN, there are three types of layers. Input layer, responsible for data input to the neural network. There is only one input layer. Then comes the hidden layers, their number can vary depending on the complexity of the data or model and they are responsible for all the computation. Lastly the output layer, responsible for generating results. Number of output layers can vary as well.

The above mentioned four ML algorithms are used to handle different types of task and all perform very well in their position. In next chapter we will test all of them and then decide on our final ML model depending on the results.

Chapter 4

Credit Card Fraud Detection

4.1 Credit Card Transaction Dataset

To understand how to deal with a highly imbalanced dataset, we will be using an open-source dataset available at Kaggle. The data consists of credit card transactions made in September 2013[28]. As Kaggle is a major website for Machine Leaning competition so there are some famous solution for this problem as well by some great researchers [10], [29]-[35].

The goal is to detect fraudulent transaction from normal or non-fraudulent transactions. The dataset contains 284,807 total transaction with only 492 fraudulent ones. The fraudulent transactions only make only 0.17% of the entire dataset.

Due to privacy reason we don't know what each column of the dataset represents and all of the columns except 'time' and 'amount', have gone through a principal component analysis transformation [36]. Since for PCA, scaling is a pre-requisite so we can assume that features V1 to V28 have been scaled as well. Features 'time' and 'amount' have not gone through PCA, so they have not been scaled as well so all we have to do for data preparation is to scale 'time' and 'amount'. Amount.

4.2 Dataset Preparation for Sampling

4.2.1 Feature Scaling

The data does not have any missing value so there is no need for cleaning or replacement.

Since features V1, V2, …. V28 have gone through PCA, we can assume that they have been scaled as well because that's a prerequisite for PCA [36].

Features 'Time' and 'Amount' are scaled using Robust Scaler so that all the features are scaled. Other than that, there is no need to do anything else with the dataset.

4.2.2 Train Test Split

Dataset is split into train and test datasets and only the training dataset is sampled. Test dataset remains in its original form and does not contain any duplicated or synthetic examples. Sampled training data with balanced distribution help the with learning.

Another reason to split the data before sampling is to avoid the leakage of information, as sampling duplicates the data instances for example oversampling, if we spilt the data after sampling that some of the training data can be present in the testing data set which can corrupt our results.

4.3 Comparison of Sampling Methods

As discussed in the chapter 3, There are many sampling methods available depending on the situation and data. The only way to see which sampling technique to use depends on the problem for example in fraudulent transaction detection, I believe it is crucial for the process to be extremely fast even so much so that a little sacrifice on the accuracy can also be

tolerated.
So, the execution time is the most important factor for our problem.

There are two, time consuming process in our algorithm

1) Time taken by sampling method

2) Time taken by the Classifier

First, we go through the most popular Under Sampling method and see their effect on the dataset.

Table 4.1: Undersampling of training data

	Dimensions of Dataset (Training Data)				
Sampling Method	Before Sampling	After Sampling	Label '1' (%)	Label '0' (%)	Execution Time
No Sampling	(227845, 30)	(227845, 30)	0.17	99.83	0:00:00
Random Undersampling	(227845, 30)	(782,30)	50	50	0:00:01
NearMiss-1	(227845, 30)	(782,30)	50	50	0:00:04
NearMiss-2	(227845, 30)	(782,30)	50	50	0:00:27
NearMiss-3	(227845, 30)	(735,30)	53.2	46.8	0:00:04
TomeLinks	(227845, 30)	(227826,30)	0.17	99.82	0:08:52
ENN	(227845, 30)	(227705,30)	0.17	99.82	1:57:38
CNN	(227845, 30)	(1368,30)	28.6	71.42	24:27:57
NCR	(227845, 30)	(227598,30)	0.17	99.82	1:59:16
OSS	(227845, 30)	(25910,30)	1.5	98.5	0:01:35

Table 4.1 shows us the effect of different undersampling techniques on our dataset. ENN, CNN and NCR have a very high execution time and they did not affect the dataset much at all so we can eliminate the techniques as they do go along with our strategy. TomeLinks took 8 minutes which is still a lot, but we will eliminate it yet. OSS performed great but it did not affect the class distribution much but that can be handled by using SMOTE later on. Random Undersampling and NearMiss algorithms reduced the data set a lot which I am afraid is also not great as we might have lost a lot of info but

cannot be sure unless we try it on the classifiers. After Undersampling let's see how Oversampling deals with the dataset.

Table 4.2: Oversampling of training data

	Dimensions of Dataset				
Sampling Method	Before Sampling	After Sampling	Label '1' (%)	Label '0' (%)	Execution Time
SMOTE	(227845, 30)	(454908,30)	50	50	0:00:10
B-SMOTE	(227845, 30)	(454908,30)	50	50	0:00:30
ADASYN	(227845, 30)	(454925, 30)	50.00	49.99	0:00:29
SVM-SMOTE	(227845, 30)	(454908,30)	50	50	1:19:00

As we know that Oversampling duplicates the minority class to even out the distribution, so the dataset size is nearly doubled after the sampling. Here only SVM-SMOTE is eliminated because it took too long.

Next, we will look at Hybrid Sampling techniques and their effect on the dataset.

Table 4.3: Hybrid sampling of training data

	Dimensions of Dataset				
Sampling Method	Before Sampling	After Sampling	Label '1' (%)	Label '0' (%)	Execution Time
OSS+ SMOTE	(227845, 30)	(47734, 30)	50	50	0:01:02
TL+ SMOTE	(227845, 30)	(454870, 30)	50	50	3:24:04
ENN-SMOTE	(227845, 30)	(454628, 30)	50	50	2:28:33

Both TL+ SMOTE and ENN+ SMOTE are taking way to long for the sampling so the only viable option here is OSS+ SMOTE.

Ensemble sampling showed pretty exciting results as well.

Table 4.4: Ensemble sampling of training data

	Dimensions of Dataset				
Sampling Method	Before Sampling	After Sampling	Label '1' (%)	Label '0' (%)	Execution Time
OSS+ TL+ SMOTE	(227845, 30)	(45966, 30)	50	50	0:00:38
OSSx2+ SMOTE	(227845, 30)	(15840, 30)	50	50	0:00:22
OSSx3+ SMOTE	(227845, 30)	(3518, 30)	50	50	0:00:23
OSSx4+ SMOTE	(227845, 30)	(3194,30)	50	50	0:00:23
OSSx5+ SMOTE	(227845, 30)	(2078,30)	50	50	0:00:23

Execution time for all of these was very good, the multiple use of OSS decreased the size of dataset, in the results section we can see how it will affect the performance metrics.

Performance metrics used for imbalanced classification is F-Score. It gives a single value which takes into account both precision and recall. It has been discussed in detail in chapter 3.

4.4 Comparison of ML Classifiers

Once our data has a balanced Class distribution, we can use any of the available Machine Learning algorithms to proceed. I have chosen to test four algorithms, Logistic Regression (LR), XGBoost (XGB), Random Forest (RF) and Neural Networks (NNs). These are the standard algorithms with LR being one the simplest and widely used ones which can provide us with a base model score.

RF and XGB are great classifiers especially good for imbalanced classification. These algos can even produce great results without sampling but can a take up quite some time, RF in particular. Neural Networks works great as well but need a lot of tweaking and hyperparameter tunning. These classifiers are evaluated on the basis of F-Score and the execution time. First let's see how the perform on undersampled data.

Table 4.5: F-Score with Undersampled data

	F-Score of Each ML Algorithm			
Sampling Method	Logistic Regression	Random Forest	XGBoost	Neural Network
No Sampling	0.74	0.83	0.86	0.82
Random Undersampling	0.1	0.1	0.08	0.12
NearMiss-1	0.01	0	0	0.01
NearMiss-2	0	0	0	0
NearMiss-3	0.03	0.59	0.06	0.15
TomeLinks	0.75	0.85	0.86	0.83
OSS	0.77	0.85	0.85	0.84

Table 4.5 shows very interesting results, first without any sampling, RF, XGB and NNs generate pretty great results. Random Sampling and NearMiss

algorithms fell apart probably due to lose of too much info so these can be eliminated from final selection as well. TomeLinks shows the best results with XGBoost with F-score of 0.86. OSS also shows great result of 0.85 as well both with RF and XGB.

Table 4.6 shows the effect of oversampled data with the classifiers.

Table 4.6: F-Score with Oversampled data

	F-Score of Each ML Algorithm			
Sampling Method	Logistic Regression	Random Forest	XGBoost	Neural Network
SMOTE	0.12	0.86	0.79	0.53
B-SMOTE	0.2	0.85	0.84	0.79
ADASYN	0.04	0.85	0.77	0.64

RF performs the best with oversampled data by giving a F-score of 0.86.

Table 4.7 shows how Hybrid Sampling works with the classifiers.

Table 4.7: F-Score after Hybrid Sampling

	F-Score of Each ML Algorithm			
Sampling Method	Logistic Regression	Random Forest	XGBoost	Neural Network
OSS+ SMOTE	0.45	0.86	0.79	0.58

OSS+ SMOTE also gives the best result with Random Forest.

Ensemble Sampling use multiple sampling techniques to generate the data. Table 4.8 shows the F-Score of classifiers with Ensemble Sampling techniques.

Table 4.8: F-Score after Ensemble Sampling

	F-Score of Each ML Algorithm			
Sampling Method	Logistic Regression	Random Forest	XGBoost	Neural Network
OSS+ TL+ SMOTE	0.12	0.86	0.79	0.53
OSSx2+ SMOTE	0.40	0.85	0.76	0.26
OSSx3+ SMOTE	0.49	0.86	0.77	0.3
OSSx4+ SMOTE	0.43	0.87	0.75	0.51
OSSx5+ SMOTE	0.44	0.79	0.63	0.48

Random Forest shows the best score here as well until OSSx4+ SMOTE and it starts going down after that.

From all the results in Tables 4.5-4.8, it is clear that Random Forest is showing the best and most persistent results. RF will the final ML model for this task.

Chapter 5

Experiment Results

Now we are in a position to decide on the best possible approach after all the experiment in chapter 4. As I mentioned earlier, execution time is one of the most important factors. We have already eliminated most of the sampling techniques and decided on the best performing ML algorithms.

Let's see which combination of sampling technique and which ML algorithm shows best performance with least execution time and highest F-Score. We will look at sampling technique execution time, ML algo execution time and the F-score to finalize the model.

Table 5.1: F-Score and Execution Time after Under Sampling

	Execution Time of Each ML Algorithm			
Under Sampling Method	Random Forest		XGBoost	
	F-Score	Execution Time	F-Score	Execution Time
No Sampling	0.83	0:23:00	0.86	0:01:06
TomeLinks	0.85	0:21:06	0.86	0:01:12
OSS	0.85	0:26:20	0.85	0:01:02

Table 5.1 shows that TL gives the best F-score with XGBoost and the execution time is around 1 minutes. But we also know that TL itself takes around 8 minutes to complete the sampling process so in total if we go TL and XGBoost we get the F-score of 0.86 with 9 minutes of execution time. Which is a lot, so undersampling on its own cannot generate the best optimal results for us.

Table 5.2: F-Score and Execution Time after Over Sampling

	Execution Time of Each ML Algorithm			
Over Sampling Method	Random Forest		XGBoost	
	F-Score	Execution Time	F-Score	Execution Time
SMOTE	0.86	3:45:14	0.79	0:12:04
B-SMOTE	0.85	3:55:43	0.84	0:13:55
ADASYN	0.85	3:58:52	0.77	0:15:05

From Table 5.2 it is clear that almost all the oversampling techniques generate great results with random forest but the problem here is execution time, which is almost 4 hours. So oversampling is also not the best approach.

Table 5.3: F-Score and Execution Time after Hybrid Sampling

	Execution Time of Each ML Algorithm			
Hybrid Sampling Method	Random Forest		XGBoost	
	F-Score	Execution Time	F-Score	Execution Time
OSS+ SMOTE	0.86	0:05:40	0.79	0:00:52

Table 5.3 shows the hybrid OSS+ SMOTE model, which generates the highest F-score of 0.86 with Random Forest in 5 minutes and 40 seconds which is pretty good but still execution time is on the higher end. Let's see if we can decrease the execution time even further with keeping the same F-score.

Table 5.4: F-Score and Execution Time after Ensemble Sampling

	Execution Time of Each ML Algorithm			
Ensemble Sampling Method	Random Forest		XGBoost	
	F-Score	Execution Time	F-Score	Execution Time
OSS+ TL+ SMOTE	0.86	1:45:14	0.79	0:12:04
OSSx2+ SMOTE	0.85	0:00:32	0.76	0:00:04
OSSx3+ SMOTE	0.86	0:00:14	0.77	0:00:01
OSSx4+ SMOTE	0.87	0:00:08	0.75	0:00:01

The execution time is directly related to the training dataset size so if can reduce the size of dataset without losing too much info, it might be possible to reduce the execution time as well. Since from all the experiments it is clear that OSS is performing better than others so I tried using OSS multiple time and observed how it affected the results.

Table 5.4 shows that we can use a combination of OSSx4 followed by SMOTE to generate the highest possible F-score of 0.87 but in only 8 seconds.

Looking at the results of all the experiments we can see that in terms of F-score, XGBoost performed the best with unsampled data and Random Forest produced best results with our modified ensemble sampling method. Since the F-Score same in both cases the deciding factor will be the execution time, which is significantly less with our final model.

Table 5.5: F-Score and Execution Time Final Model vs Unsampled data

Sampling Method	Random Forest		XGBoost	
	F-Score	Time	F-Score	Time
No Sampling	0.83	0:23:00	0.86	66s
Final Model (OSSx4+ SMOTE)	0.87	23s+ 8s	0.75	23s+ 1s

Table 5.5 shows our final model performance against the best result which we got without sampling. The final model got the F-score of 0.87, which is a little better than the opponent, but the main achievement here is the reduction in execution time which is almost 50% reduction.

We can confirm if go into a little detail about this final comparison by looking at the confusion matrix for XGBoost for unsampled data and for Random Forest for final model.

Fig 5.1 is the confusion matrix of XGBoost algorithm on Unsampled dataset. We can see the precision here is very good, but recall is not so great. We are looking for a balance between precision and recall as both of these metrics are important. The confusion matrix shows us that out of 101 fraudulent cases, 82 were detected correctly. 7 of fraudulent cases were classified as non-fraudulent and 19 of the normal transaction were classified as fraudulent cases.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56861
1	0.92	0.81	0.86	101
accuracy			1.00	56962
macro avg	0.96	0.91	0.93	56962
weighted avg	1.00	1.00	1.00	56962

Runtime of XGBoost on Unsampled Data is 68.2778959274292

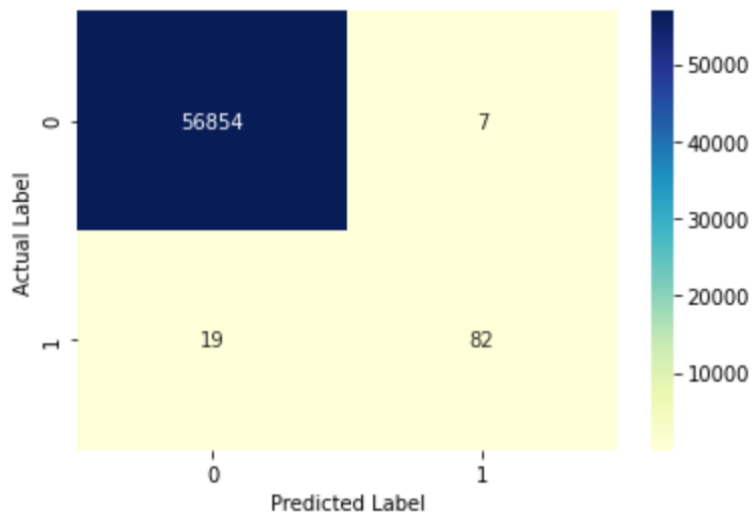

Figure 5.1: Confusion Matrix of XGBoost without Sampling

Now let's look at the confusion matrix of Random Forest with ensemble sampling in Fig 5.1, here the F-score is a little better at 0.87. But the precision here drops a little bit and recalls goes up by a little margin. As discusses in chapter 3, there is a tug of war situation between precision and recall so we look for a balance between these two. RF with ensemble sampling give us that. Another important thing here is the reduction in execution time as discussed earlier.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56861
1	0.89	0.85	0.87	101
accuracy			1.00	56962
macro avg	0.94	0.93	0.93	56962
weighted avg	1.00	1.00	1.00	56962

Runtime of Random Forest with ensemble sampling is 8.497461080551147

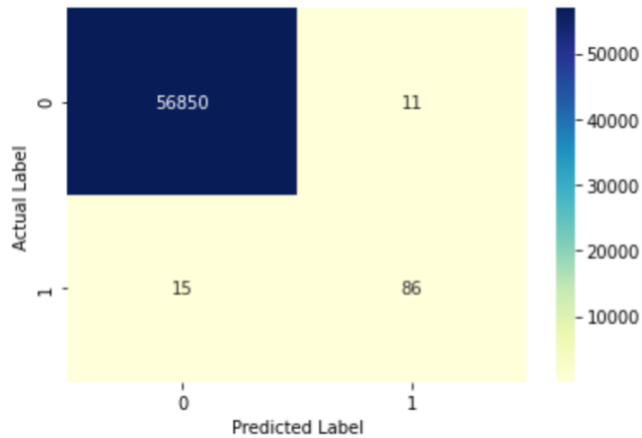

Figure 5.2: Confusion Matrix of Random Forest with Ensemble Sampling

Chapter 6

Conclusion

Resampling the imbalanced data is the most popular approach when dealing imbalanced classification, however there are different sampling techniques each with its pros and cons. The thesis discusses the effect of different sampling techniques on a skewed credit card transaction dataset. As shown in chapter 5, undersampling has shown poor results mainly due to the loss of information during resampling. SMOTE shows very good results but in turn uses a lot of computational power and time. The Hybrid approach where the data is first undersampled using OSS and then oversampled with SMOTE, has shown a huge reduction in execution time mainly because using undersampling before SMOTE decreases the size of the dataset. The final technique using OSS, 4 times then followed by SMOTE generated an F-score of 0.86 with an execution time of only 31 seconds.

From the ML models that we tested, Random Forest and XGBoost showed the best results but the final model was RF with OSSx4+ SMOTE sampling technique.

In the case of fraudulent transaction detection, speed of the algorithm is of utmost importance as there are hundreds if not thousands of transactions happening every minute, online and offline, if the model is very accurate but has a higher execution time it will be of no use as we might miss the fraud detection due to the delay. Our model shows considerably great detection rate in a fraction of the time.

Bibliography

[1] T. Van Nguyen, N. Sae-Bae, and N. Memon, "DRAW-A-PIN: Authentication using finger-drawn PIN on touch devices," *Comput. Secur.*, 2017, doi: 10.1016/j.cose.2017.01.008.

[2] M. Inoue and T. Ogawa, "TapOnce: a novel authentication method on smartphones," *Int. J. Pervasive Comput. Commun.*, 2018, doi: 10.1108/IJPCC-D-18-00006.

[3] M. H. Barkadehi, M. Nilashi, O. Ibrahim, A. Zakeri Fardi, and S. Samad, "Authentication systems: A literature review and classification," *Telematics and Informatics*. 2018, doi: 10.1016/j.tele.2018.03.018.

[4] D. Kunda and M. Chishimba, "A Survey of Android Mobile Phone Authentication Schemes," *Mob. Networks Appl.*, 2018, doi: 10.1007/s11036-018-1099-7.

[5] S. Jha, M. Guillen, and J. Christopher Westland, "Employing transaction aggregation strategy to detect credit card fraud," *Expert Syst. Appl.*, 2012, doi: 10.1016/j.eswa.2012.05.018.

[6] Y. Sahin and E. Duman, "Detecting credit card fraud by ANN and logistic regression," 2011, doi: 10.1109/INISTA.2011.5946108.

[7] G. Rushin, C. Stancil, M. Sun, S. Adams, and P. Beling, "Horse race analysis in credit card fraud - Deep learning, logistic regression, and Gradient Boosted Tree," 2017, doi: 10.1109/SIEDS.2017.7937700.

[8] K. Whitehair, "Libraries in an Artificially Intelligent World," *Public Libraries Online*, 2016.

[9] A. Shen, R. Tong, and Y. Deng, "Application of classification models on credit card fraud detection," in *2007 International conference on service systems and service management*, 2007, pp. 1-4.

[10] A. Dal Pozzolo, "Adaptive machine learning for credit card fraud detection," 2015.

[11] C. Whitrow, D. J. Hand, P. Juszczak, D. Weston, and N. M. Adams, “Transaction aggregation as a strategy for credit card fraud detection,” *Data Min. Knowl. Discov.*, vol. 18, no. 1, pp. 30–55, 2009.

[12] A. Ali, S. M. Shamsuddin, and A. L. Ralescu, “Classification with class imbalance problem: a review,” *Int. J. Adv. Soft Compu. Appl*, vol. 7, no. 3, pp. 176–204, 2015.

[13] A. Fernández, S. García, M. Galar, R. C. Prati, B. Krawczyk, and F. Herrera, *Learning from imbalanced data sets*. Springer, 2018.

[14] M. Kuhn and K. Johnson, *Applied predictive modeling*, vol. 26. Springer, 2013.

[15] H. He and Y. Ma, *Imbalanced learning: foundations, algorithms, and applications*. John Wiley & Sons, 2013.

[16] M. Palt and M. Palt, “ScienceDirect The Proposal of Undersampling Method for Learning from The Proposal of Undersampling Method for Learning from Imbalanced Datasets Imbalanced Datasets,” *Procedia Comput. Sci.*, vol. 159, pp. 125–134, 2019, doi: 10.1016/j.procs.2019.09.167.

[17] M. Kubat and S. Matwin, “Addressing the curse of imbalanced training sets: one-sided selection,” in *Icml*, 1997, vol. 97, pp. 179–186.

[18] H. Mansourifar and W. Shi, “Deep synthetic minority over-sampling technique,” *arXiv*, vol. 16, pp. 321–357, 2020.

[19] H. Han, W. Y. Wang, and B. H. Mao, “Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning,” in *Lecture Notes in Computer Science*, 2005, vol. 3644, no. PART I, pp. 878–887, doi: 10.1007/11538059_91.

[20] E. Rendón, R. Alejo, C. Castorena, F. J. Isidro-Ortega, and E. E. Granda-Gutiérrez, “Data sampling methods to dealwith the big data multi-class imbalance problem,” *Appl. Sci.*, vol. 10, no. 4, 2020, doi: 10.3390/app10041276.

[21] Google, “Machine Learning Crash Course,” *Machine Learning Crash Course*. https://developers.google.com/machine-learning/crash-course/classification/accuracy.

[22] C. G. Weng and J. Poon, “A new evaluation measure for imbalanced datasets,” *Conf. Res. Pract. Inf. Technol. Ser.*, vol. 87, no. 81373883, pp. 27–32, 2008.

[23] F. Pedregosa *et al.*, “Scikit-learn: Machine Learning in Python,” *J. Mach. Learn. Res.*, vol. 12, no. 85, pp. 2825–2830, 2011, [Online]. Available: http://jmlr.org/papers/v12/pedregosa11a.html.

[24] R. E. Wright, “Logistic regression.,” 1995.

[25] A. Liaw and M. Wiener, “Classification and regression by randomForest,” *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[26] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, vol. 13-17-Augu, pp. 785–794, 2016, doi: 10.1145/2939672.2939785.

[27] X. Jielai, J. Hongwei, and T. Qiyi, “Introduction to artificial neural networks,” *Adv. Med. Stat.*, no. 9624670, pp. 1431–1449, 2015, doi: 10.1142/9789814583312_0037.

[28] M. L. Group-ULB, “Kaggle.” https://www.kaggle.com/mlg-ulb/creditcardfraud/metadata.

[29] A. Dal Pozzolo, O. Caelen, Y.-A. Le Borgne, S. Waterschoot, and G. Bontempi, “Learned lessons in credit card fraud detection from a practitioner perspective,” *Expert Syst. Appl.*, vol. 41, no. 10, pp. 4915–4928, 2014.

[30] F. Carcillo, A. Dal Pozzolo, Y.-A. Le Borgne, O. Caelen, Y. Mazzer, and G. Bontempi, “Scarff: a scalable framework for streaming credit card fraud detection with spark,” *Inf. fusion*, vol. 41, pp. 182–194, 2018.

[31] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi,

"Credit card fraud detection: a realistic modeling and a novel learning strategy," *IEEE Trans. neural networks Learn. Syst.*, vol. 29, no. 8, pp. 3784–3797, 2017.

[32] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi, "Calibrating probability with undersampling for unbalanced classification," in *2015 IEEE Symposium Series on Computational Intelligence*, 2015, pp. 159–166.

[33] F. Carcillo, Y.-A. Le Borgne, O. Caelen, and G. Bontempi, "Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization," *Int. J. Data Sci. Anal.*, vol. 5, no. 4, pp. 285–300, 2018.

[34] B. Lebichot, Y.-A. Le Borgne, L. He-Guelton, F. Oblé, and G. Bontempi, "Deep-learning domain adaptation techniques for credit cards fraud detection," in *INNS Big Data and Deep Learning conference*, 2019, pp. 78–88.

[35] F. Carcillo, Y.-A. Le Borgne, O. Caelen, Y. Kessaci, F. Oblé, and G. Bontempi, "Combining unsupervised and supervised learning in credit card fraud detection," *Inf. Sci. (Ny).*, 2019.

[36] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemom. Intell. Lab. Syst.*, vol. 2, no. 1–3, pp. 37–52, 1987.

Credit Card Fraud Detection using Imbalanced Data Classification	Thesis for Master of Engineering	December 2020		ABBAS QALAB E

Appendix

Code for Machine Learning Algorithms (LR, RF, XGB, ANN) for Unsampled Data:

The experiment was performed on MacBook Pro (Late 2013) with 16 gb RAM and 512 GB SSD and no GPU was used.
Anaconda with Jupyter Notebook was used for the coding environment.

```
#importing all the Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split as holdout
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score,
accuracy_score, classification_report
from sklearn.metrics import average_precision_score,confusion_matrix,
precision_recall_curve
from sklearn.ensemble import RandomForestClassifier as rfc
import xgboost as xgb
from keras.models import Sequential
from keras.layers import Dense, Dropout
import time

#Importing Dataset
df = pd.read_csv('creditcard.csv')
df.head()

#Checking the class Distribution
```

```
sns.countplot(x='Class', data=df, palette='CMRmap')
print('Non-fraud transactions:
{}%'.format(round(df.Class.value_counts()[0]/len(df)*100.0,2)))
print('Fraud transactions:
{}%'.format(round(df.Class.value_counts()[1]/len(df)*100.0,2)))

#Scaling 'Time' and 'Amount' Features
rs = RobustScaler()
df['scaled_amount'] = rs.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rs.fit_transform(df['Time'].values.reshape(-1,1))
df.drop(['Time', 'Amount'], axis=1, inplace=True)
scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']
df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
df.insert(0, 'scaled_amount', scaled_amount)
df.insert(0, 'scaled_time', scaled_time)
df.head()

#Train test split

x = np.array(df.iloc[:, df.columns != 'Class'])
y = np.array(df.iloc[:, df.columns == 'Class'])
x_train, x_test, y_train, y_test = holdout(x, y, test_size=0.2,
random_state=0)

#Logistis Regression on Original data

st = time.time()
logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_test)
cnf_matrix = confusion_matrix(y_test, y_pred)

sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu",
fmt='g')
```

```
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')

labels = ['Non-fraud', 'Fraud']
print(classification_report(y_test, y_pred, target_names=labels))

et = time.time()
print(f"Runtime of LR on unsampled data is {et - st}")
```

#RF on Unsampled Data

```
st = time.time()
rand_f = rfc(n_estimators=1000, min_samples_split=10,
min_samples_leaf=1,
            max_features='auto', max_leaf_nodes=None,
            oob_score=True, n_jobs=-1, random_state=1)
rand_f.fit(x_train, y_train)
y_pred = rand_f.predict(x_test)

cnf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu",
fmt='g')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')

print(classification_report(y_test, y_pred))

et = time.time()
print(f"Runtime of RFC on Unsampled Data {et - st}")
```

#XGBoost on Unsampled Data

```
st = time.time()

xgbmodel = xgb.XGBClassifier()
```

```
xgbmodel.fit(x_train, y_train)
y_pred = xgbmodel.predict(x_test)

cnf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu",
fmt='g')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')

print(classification_report(y_test, y_pred))

et = time.time()
print(f"Runtime of XGBoost on Unsampled Data is {et - st}")
```

#Neural Network Model

```
nnmodel = Sequential([Dense(input_dim=30, units=16, activation='relu'),
                      Dense(units=24, activation='relu'),
                      Dropout(0.5),
                      Dense(units=20, activation='relu'),
                      Dense(units=24, activation='relu'),
                      Dense(units=1, activation='sigmoid')])
st = time.time()

nnmodel.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
nnmodel.fit(x_train, y_train, batch_size=15, epochs=15)
et = time.time()

print(f"Runtime of ANN on Unsampeld data is {et - st}")

score = nnmodel.evaluate(x_test, y_test)
print(score)
```

```
y_pred = nnmodel.predict_classes(x_test)

cnf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu",
fmt='g')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')

print(classification_report(y_test, y_pred))
```

Code for Final Model(RF with Ensemble Sampled Data):

```
#Import all the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import RobustScaler
from sklearn.model_selection import train_test_split as holdout
from sklearn.metrics import confusion_matrix, precision_recall_curve,
classification_report
from sklearn.metrics import precision_score, recall_score,
average_precision_score
from imblearn.under_sampling import OneSidedSelection
from imblearn.over_sampling import SMOTE
from sklearn.ensemble import RandomForestClassifier as rfc
import time

#Importing dataset
df = pd.read_csv('creditcard.csv')
df.head()
```

#Class distributation check

```
sns.countplot(x='Class', data=df, palette='CMRmap')
print('Non-fraud transactions:
{}%'.format(round(df.Class.value_counts()[0]/len(df)*100.0,2)))
print('Fraud transactions:
{}%'.format(round(df.Class.value_counts()[1]/len(df)*100.0,2)))
```

#Scaling 'Time' and 'Amount' Features

```
rs = RobustScaler()
df['scaled_amount'] = rs.fit_transform(df['Amount'].values.reshape(-1,1))
df['scaled_time'] = rs.fit_transform(df['Time'].values.reshape(-1,1))
df.drop(['Time', 'Amount'], axis=1, inplace=True)
scaled_amount = df['scaled_amount']
scaled_time = df['scaled_time']
df.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
df.insert(0, 'scaled_amount', scaled_amount)
df.insert(0, 'scaled_time', scaled_time)
df.head()
```

#Train, Test Split

```
x = np.array(df.iloc[:, df.columns != 'Class'])
y = np.array(df.iloc[:, df.columns == 'Class'])
x_train, x_test, y_train, y_test = holdout(x, y, test_size=0.2,
random_state=0)
```

#Sampling the data with OSSx4 and then SMOTE

```
sts= time.time()
print('-------------------------------------------------------
-------------')
print("Transaction Number x_train dataset: ", x_train.shape)
print("Transaction Number y_train dataset: ", y_train.shape)
```

```
print("Transaction Number x_test dataset: ", x_test.shape)
print("Transaction Number y_test dataset: ", y_test.shape)

print("Before Sampling, counts of label '1': {}".format(sum(y_train==1)))
print("Before Sampling, counts of label '0': {}
₩n".format(sum(y_train==0)))
print('----------------------------------------------------------
-------------')
undersample1 = OneSidedSelection(n_neighbors=1, n_seeds_S=200)
x_train_s, y_train_s = undersample1.fit_sample(x_train, y_train.ravel())

print('After OSSx1, the shape of train_x: {}'.format(x_train_s.shape))
print('After OSSx1, the shape of train_y: {} ₩n'.format(y_train_s.shape))

print("After OSSx1, counts of label '1': {}".format(sum(y_train_s==1)))
print("After OSSx1, counts of label '0': {} ₩n".format(sum(y_train_s==0)))


print("After OSSx1, counts of label '1', %:
{}".format(sum(y_train_s==1)/len(y_train_s)*100.0,2))
print("After OSSx1, counts of label '0', %:
{}".format(sum(y_train_s==0)/len(y_train_s)*100.0,2))
print('----------------------------------------------------------
-------------')
undersample2 = OneSidedSelection(n_neighbors=1, n_seeds_S=200)
x_train_s2, y_train_s2 = undersample2.fit_sample(x_train_s,
y_train_s.ravel())
print('After OSSx2, the shape of train_x: {}'.format(x_train_s2.shape))
print('After OSSx2, the shape of train_y: {} ₩n'.format(y_train_s2.shape))

print("After OSSx2, counts of label '1': {}".format(sum(y_train_s2==1)))
print("After OSSx2, counts of label '0': {}
₩n".format(sum(y_train_s2==0)))
```

```
print("After OSSx2, counts of label '1', %:
{}".format(sum(y_train_s2==1)/len(y_train_s2)*100.0,2))
print("After OSSx2, counts of label '0', %:
{}".format(sum(y_train_s2==0)/len(y_train_s2)*100.0,2))

print('-------------------------------------------------------
-------------')
undersample3 = OneSidedSelection(n_neighbors=1, n_seeds_S=200)
x_train_s3, y_train_s3 = undersample3.fit_sample(x_train_s2,
y_train_s2.ravel())
print('After OSSx3, the shape of train_x: {}'.format(x_train_s3.shape))
print('After OSSx3, the shape of train_y: {} ₩n'.format(y_train_s3.shape))

print("After OSSx3, counts of label '1': {}".format(sum(y_train_s3==1)))
print("After OSSx3, counts of label '0': {}
₩n".format(sum(y_train_s3==0)))


print("After OSSx3, counts of label '1', %:
{}".format(sum(y_train_s3==1)/len(y_train_s3)*100.0,2))
print("After OSSx3, counts of label '0', %:
{}".format(sum(y_train_s3==0)/len(y_train_s3)*100.0,2))

print('-------------------------------------------------------
-------------')
undersample4 = OneSidedSelection(n_neighbors=1, n_seeds_S=200)
x_train_s4, y_train_s4 = undersample4.fit_sample(x_train_s3,
y_train_s3.ravel())
print('After OSSx4, the shape of train_x: {}'.format(x_train_s4.shape))
print('After OSSx4, the shape of train_y: {} ₩n'.format(y_train_s4.shape))

print("After OSSx4, counts of label '1': {}".format(sum(y_train_s4==1)))
print("After OSSx4, counts of label '0': {}
₩n".format(sum(y_train_s4==0)))
```

```
print("After OSSx4, counts of label '1', %:
{}".format(sum(y_train_s4==1)/len(y_train_s4)*100.0,2))
print("After OSSx4, counts of label '0', %:
{}".format(sum(y_train_s4==0)/len(y_train_s4)*100.0,2))

print('-------------------------------------------------------
-------------')
sm = SMOTE()

x_train_ss, y_train_ss = sm.fit_sample(x_train_s4, y_train_s4.ravel())

print('After OSSx4+ SMOTE, the shape of train_x:
{}'.format(x_train_ss.shape))
print('After OSSx4+ SMOTE, the shape of train_y: {}
₩n'.format(y_train_ss.shape))

print("After OSSx4+ SMOTE, counts of label '1':
{}".format(sum(y_train_ss==1)))
print("After OSSx4+ SMOTE, counts of label '0': {}
₩n".format(sum(y_train_ss==0)))


print("After OSSx4+ SMOTE, counts of label '1', %:
{}".format(sum(y_train_ss==1)/len(y_train_ss)*100.0,2))
print("After OSSx4+ SMOTE, counts of label '0', %:
{}".format(sum(y_train_ss==0)/len(y_train_ss)*100.0,2))


print('-------------------------------------------------------
-------------')

sns.countplot(x=y_train_ss, data=df, palette='CMRmap')
```

```
print('-------------------------------------------------------
-------------')

ets=time.time()
print(f"Runtime for Sampling is {ets - sts}")
```

#Random Forest Classifier

```
strf = time.time()
rand_f = rfc(n_estimators=1000, min_samples_split=10,
min_samples_leaf=1,
                max_features='auto', max_leaf_nodes=None,
                oob_score=True, n_jobs=-1, random_state=1)
rand_f.fit(x_train_ss, y_train_ss)
y_pred = rand_f.predict(x_test)

cnf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu",
fmt='g')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')

print(classification_report(y_test, y_pred))

etrf= time.time()
print(f"Runtime of Random Forest with ensemble sampling is {etrf - strf}")
```