ENSEMBLE BASED CLASSIFICATION FOR CLASS IMBALANCED CREDIT CARD FRAUDULENT DATA

A PROJECT REPORT

Submitted by

PANKAJ THAKUR [Reg No: RA1611003011072]
SIDDHARTH AGARWAL [Reg No: RA1611003011132]

Under the guidance of

Mrs. S. Priya

(Assistant Professor(Sr.G), Department of Computer Science & Engineering)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Kancheepuram District

MAY 2020

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this project report titled " **ENSEMBLE BASED CLASSIFICATION FOR CLASS IMBALANCED CREDIT CARD FRAUDULENT DATA**" is the bonafide work of " **PANKAJ THAKUR [Reg No: RA1611003011072], SIDDHARTH AGARWAL [Reg No: RA1611003011132]**", who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Mrs. S. Priya
GUIDE
Assistant Professor(Sr.G)
Dept. of Computer Science & Engineering

SIGNATURE

Dr. B. Amutha
HEAD OF THE DEPARTMENT
Dept. of Computer Science and Engineering

Signature of the Internal Examiner

Signature of the External Examiner

Own Work Declaration

Department of Computer Science and Engineering

SRM Institute of Science & Technology

Own Work* Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

Degree/ Course	:	**B.Tech/CSE**
Student Name	:	**Pankaj Thakur, Siddharth Agarwal**
Registration Number	:	**RA1611003011072, RA1611003011132**
Title of Work	:	**Ensemble Classification for Class imbalanced credit card fraud data**

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism**, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly references / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalised in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

Siddharth Agarwal

Pankaj Thakur

ABSTRACT

Knowledge extraction from imbalanced data has been receiving increasing interest in recent years. Most of the real world problems including credit card transaction frauds, disease prediction and so on have huge data instances but the count of positive of instances that is disease or fraud in those data sets is far lesser than the number of negative instances like healthy or non fraud. Suppose 99 % of the data tuples come from the majority class but only 1% of these instances belong from the positive or the minority class, and our classifier classifies each data tuple as majority instance, then the accuracy of our model will of course be 99% but this classifier is of no use as it does not detect any minority instance which is the main purpose of the classifier. This is why our imbalanced data needs to be balanced first for training our classifiers. The balanced dataset can now be used to train our classification model. There are so many classification algorithms present, and each of them has their own pros and cons. So, in order to achieve higher accuracy and better results various individual classification algorithms including Naive Bayes, Decision Trees and Random Forests are used. Ensemble models have been applied that includes Voting Classifier, XGBoost and ADABoost, It is seen that the efficiency of ensemble classifiers in data classification is much higher than that of individual algorithms.

ACKNOWLEDGEMENTS

We express our humble gratitude to Dr. Sandeep Sancheti, Vice Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to Dr. C. Muthamizhchelvan, Director, Faculty of Engineering and Technology, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank Dr. B. Amutha, Professor & Head, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for her valuable suggestions and encouragement throughout the period of the project work.

We are extremely grateful to our Academic Advisor Dr. A. Jeyasekar, Associate Professor,and Dr.R.Annie Uthra , Associate Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for their great support at all the stages of project work.

We would like to convey our thanks to our Panel Head, Dr. A. Jeyasekar, Associate Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for his inputs during the project reviews.

We register our immeasurable thanks to our Faculty Advisor, M. Uma Devi, Associate Professor, Department of Computer Science and Engineering, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to my guide, S. Priya, Assistant Professor(Sr.G), Department of Computer Science and Engineering, SRM Institute of Science and Technology, for providing me an opportunity to pursue my project under her mentorship. She provided me the freedom and support to explore the research topics of my interest. Her passion for solving the real problems and making a difference in the world has always been inspiring.

We sincerely thank staff and students of the Computer Science and Engineering Department, SRM Institute of Science and Technology, for their help during my research. Finally, we would like to thank my parents, our family members and our friends for their unconditional love, constant support and encouragement.

Pankaj Thakur

Siddharth Agarwal

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

SMOTE	Synthetic Minority Over-sampling TEchnique
PCA	Principal Component Analysis
MSMOTE	Modified Synthetic Minority Oversampling Technique
XGBoost	eXtreme Gradient Boosting
ADABoost	ADAptive Boosting
ADASYN	ADAptive SYNthetic Minority Over-sampling Technique

CHAPTER 1

INTRODUCTION

1.1 The Class Imbalance Problem

Many of the real world problems have their datasets that exhibit class imbalance. Imbalanced datasets are those datasets in which the count of one or more classes is much lesser than others. Imbalanced datasets can be intrinsic i.e. the domain of the dataset might be biased towards one class and very few cases belong to the other classes. Also, the data collection method might be inefficient that leads to imbalanced datasets. The class with the highest number of examples is known as the majority class and the class with very few examples in the dataset is known as the minority class. In imbalanced datasets, rare events are very hard to predict. Some of the examples of events that form imbalanced datasets are credit card frauds, telecom frauds, medical diagnosis, network intrusion detections. Rare events are very hard to predict because these events occur only once or twice in about a thousand records. Most of the standard classifiers including SVM (Support Vector Machine), decision trees work well on balanced dataset. While facing imbalanced datasets, the classification results are fine for the majority class but not so good in case of the minority classes. In a decision tree classifier, it's nodes represent the attributes of the data instances, values of those attributes are represented by the edges, and the leaf nodes are the class labels. Due to the imbalanced class problem, the decision tree classifier needs to create more tests in order to distinguish the minority classes from majority classes. This may lead to overfitting of the data hence reduces the accuracy of the classifier. For problems with binary classification, the imbalance ratio which is equal to the ratio of number of data instances in the minority class to the number of data instances in the majority class is taken into consideration. In some cases, the ratio can be very drastic like 0.001 or even worse. We will be using the credit card transactions data set, has an imbalanced ratio of 0.0017. To being overcome the class imbalanced anomaly, we can do by adding new samples or remove some of the existing samples. The process of adding new samples in the set of existing samples is known as Oversampling. The process of removing samples from the existing samples is known as Undersampling techniques. Through various experiments conducted by various scientists and researchers, it has been found that Oversampling is generally better in giving better classification results than other undersampling techniques. Undersampling may lead to loss of those data instances which might be quite important to the training of our classifier. Currently there are various Oversampling and Undersampling methods present that have proved to be useful in removing the class imbalance problem from our dataset. The major Oversampling algorithms include Synthetic Minority Over-sampling TEchnique (SMOTE), Borderline SMOTE, ADAptive SYNthetic Minority Over-sampling Technique (ADASYN), Modified Synthetic Minority Oversampling Technique (MSMOTE) (Modified Simple Minority Oversampling Technique) and so on. Oversampling means generating synthetic data for the minority or the negative class while Under-sampling means removing negative data from the majority or the negative class. The major Under-sampling techniques include Near Miss algorithm, Random Undersampling.

1.2 Ensemble Models

For the classification task, rather than relying on only one algorithm, we can use the ensemble model to get better predictive accuracy. The main idea behind the ensemble method is weighing several individual classification algorithms, and combining them in order to design a classification model that outperforms the individual classifier. The prediction error decreases when the ensemble model is used in place of relying on any single model. The Ensemble model aggregates more than one model of same or different algorithms that are trained on the same dataset or set of different datasets. There are various ensemble methods that include bagging, boosting etc. Examples of Ensemble models include Voting Classifier, ADAptive Boosting (ADABoost), eXtreme Gradient Boosting (XGBoost), Gradient tree Boosting etc.

CHAPTER 2

LITERATURE SURVEY

2.1 Balancing the dataset

To being overcome of issue of class imbalancement, various Oversampling and Undersampling methodologies are present. SMOTE is one of the most widely used oversampling techniques. SMOTE is an oversampling technique that is used to create synthetic minority datasets. It has been observed that for high dimensional data undersampling outperforms SMOTE, but for low dimensional data SMOTE works just fine. SMOTE works by selecting two of the minority instance points, drawing a straight line between them, and then selecting a random value between 0 and 1 and multiply the distance between those two points by the selected random number. This gives us a new point on that line. The selected point is the new data instance of the minority class. Borderline-SMOTE is an extended advanced version of the SMOTE method that only over-samples the data points that have majority of their neighboring instances from the majority/negative class. These data points considered to be DANGER points. For detecting minority instances, Borderline-SMOTE tends to have better F1 scores than SMOTE and other oversampling techniques. ADASYN is an improved version of SMOTE. It works similarly as SMOTE but instead of making the synthetic data points being linearly correlated to the older data points, it adds variance to these, i.e. the synthetic data points are much more scattered than before. Random Undersampling is an Undersampling technique that randomly removes data points from the bigger class to make the count of instances in both the class same. Near Miss is an undersampling technique that removes only those data points/instances from the bigger class that have the least average distance from the minority class. In other words, those instances of the majority/bigger which are closest to the minority/smaller class are removed.

2.2 Classification Algorithms

XGBoost stands for eXtreme Gradient Boosting. It is based on ensemble classifiers' boosting technique. The XGBoost implements a lot of features to give a rapid and expandable classifier. After testing XGBoost, ADABoost and GBM ensemble classifiers on different datasets, XGBoost has proved to be the best ensemble classifier. ADABoost and Voting Classifiers are also able to provide good results in handling imbalanced data. Another important and widely used classifier is the Logistic Regression Classifier. The Logistic Regression Classifier uses sigmoid function to calculate the class values. The output of the classifier is discreet unlike the output of the linear regression models whose output is continuous. The Gaussian Naive Bayes Classifiers is a classification model that works on the basis of bayes' theorems in probability. The classifier works best in the case of independent features. Decision Tree Classifier, on combining with oversampling technique, SMOTE is proved to be highly efficient in detecting the class labels of the imbalanced data sets. Random Forest classifiers are collections of decision tree which has the number of classifiers and its numbers of features can be defined.

CHAPTER 3

PROPOSED WORK

In this paper, we have used various classification algorithms, oversampling techniques and ensemble models and tested them with different parameters. The techniques, algorithms and models that we have used are the following. The SMOTE algorithm generates synthetic data points of the minority/positive class in order to make the count of minority/positive data instances i.e. the rare events equal to the count of majority/negative data instances. Either this or the ratio can be specified in which the minority and majority instances are required. SMOTE takes the feature vector of two minority class instances, then it takes a random value between 0 and 1, multiplies the difference of feature vectors of the minority instance with the random value and adds it to the feature vector of the instance with lower value instance. By doing this, we get a new instance whose feature vector lies between the previous two minority instances. The Borderline SMOTE algorithm also generates synthetic data tuples which belong to the minority class so as to increase numbers of minority data points. This method is extended version or an implementation of SMOTE and only takes the borderline data tuples of the smaller class. The borderline data tuples are the ones that have more number of majority instances as their neighbours than the minority instances.

m=no. of neighbours of the minority instance

m'=no. of neighbours of the minority instance that come from the majority class.

If m' = m, all neighbours of the given minority instance belong to the majority class, the point is treated to be as noise and will not be used in further training step.

If m/2 < m' < m, most of its neighbours of the smaller class instances belong to the majority classes, the data tuple considered to be a borderline point as it can easily be wrongly classified and put into DANGER sets.

If 0 < m' < m/2, i.e. most neighbours of the smaller instance class belong in the minority classes, this point is considered as secure/safe and need not be included for further step.

After getting the danger set, we apply the same steps as in SMOTE to get synthetic minority datasets.

Naive Bayes theorem is considered as a very powerful technique for Predictive Modelling in machine learning. Naive Bayes classifier is a collection of machine learning models which are based on the Bayes' Theorem in Probability theories. Although, it is just not a single technique but a family of methods where all of these algorithms are sharing one single working goal as a whole and one and every pairs of features that are being classified are fully independent of one another. The training of a Naive Bayes Model is speedy because only some probability values are calculated. The probability of occurrences of every class and also that of these class given different set of input variables as in the input data set need to be calculated. The input Data set is divided into following two partitions which are the X's (also known as the feature matrix or the feature vectors) and the Y's(also known as the response vector), in which the X's consists of the rows of data set in which each row consists of dependent feature and Y's contain the outputs of each feature vector(each row). Each value in a row makes an unbiased and independent contributions into the output class label. The Bayes' theorem is stated in mathematics as:

$$P(q1|q2) = (P(q2|q1) * P(q1))/P(q2) \tag{3.1}$$

A Naive Bayes classifier is an effective machine learning technique based on probability that is used for classification tasks. Logistic Regression can be used for binary classification of datasets. The logistic function (or the sigmoid function) is used by this classifier. The function takes input values and gives output as a number that is greater than or equal to 0 and less than or equal to 1. The sigmoid/logistic function can be defined as:

$$Sigmoid(x) = \frac{1}{1 - e^x} \tag{3.2}$$

The output of the function gives an S-shaped curve whose values lie between 0 and 1. We have also made use of decision tree classifier to classify our data tuples after training our decision tree classifier model with train dataset. The decision trees forms structure that look like trees with each of its node representing a parameter of our given dataset. The decision tree is constructed in a top down manner. The most important feature is chosen as the root node of the tree and the subsequent nodes are of features that best splits the set of items. Random Forest is a simple Supervised Machine Learning method and it constructs more than one number of distinct decision trees. The final decision of the output class label is basis on the outcomes of the majority of the outcomes of decision trees where a data set is classified as row sampling and feature sampling and it is going to be train in the multiple decision tree and may be some feature sampling and row sampling is common to the other decision tree and all the samples of the dataset are trained with the model and it gives some results in binary and continuous results of regression problems. Whenever we pass some test data in a binary way it gives some result by checking with all the decision tree.According to Bagging Finally have to aggregate and select the answer with majority votes. Random forest includes flexibility and reduces the high variance to the low variance by combining all decision trees with respect to majority votes. High variance is reduced to the low variance and even in the change of dataset it does not impact more on our result. XGBoost is also known as sequential ensemble technique. The main feature of this algorithm is scalability that leads to fast learning through parallel computing and performs efficient memory usage. It creates a sequential decision tree and the first decision tree takes a sample of records and classify as true or false and those records and classified results are known as 1st weak classifier and records classifies as false their weight of that sample is updated and updated records and similar weight records are passed to the next sequential decision tree and this is going to be continued. All the weak classifiers are going to be combined and get the final classifier. It checks the model as classified with 0 or 1 and by seeing the model with maximum number of output are going to be considered by model. In XGBoost it gives high bias and low variance and by using the decision tree up to some depth and it combine multiple weak classifier and it reduces the high bias into low bias. In the Adaboost model, the output of our other classification models is combined into a sum after being weighted. This sum represents the final output of the classification model. In AdaBoost , Decision Tree is created with only one depth and for every one feature we create one stumps. On seeing entropy and GiniIndex stumps with lesser entropy is going to be selected. Check the wrongly classified sample and calculate the total error(Classified error / Sum of all weight) and performance of the stump where

$$Performance = \frac{1}{2} * log(\frac{1 - totalerror}{totalerror}) \tag{3.3}$$

Then update the weight and our new sample weight.

$$NewSampleWeight = weight * e^{performance} \tag{3.4}$$

$$CorrectlyClassifiedSampleWeight = weight * e^{-performance} \tag{3.5}$$

After this we get Normalized value and updated weight, calculate mean of updated value and by dividing the value with the mean we will get normalized Value. Now , on working with normalized Value, create a new dataset and it selects wrong data and trains the model and divides it into buckets of normalized value and majority votes between all the stumps. We are combining weak learners to make it stronger. The voting classifier is a type of bagging ensemble model that combines different machine learning algorithms and tests the test data on all the models, then performs voting on the results of all these models and the final output is given as the class that achieves the majority of these votes. The weights of votes of each model are kept the same. The proposed architecture is shown in Figure 3.1.

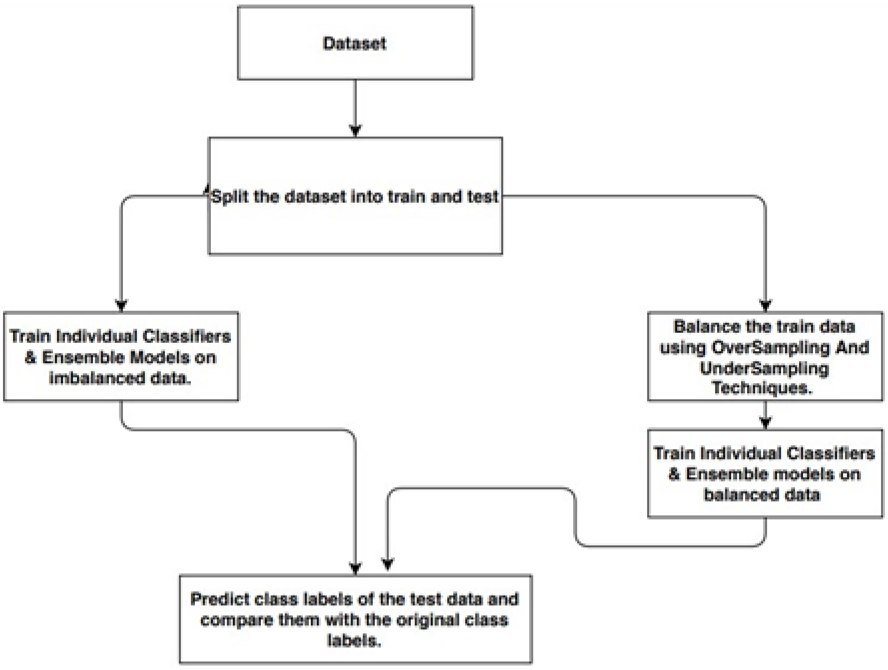

Figure 3.1: Proposed Architecture

CHAPTER 4

IMPLEMENTATION

4.1 Data set

We have made use of the imbalanced credit card fraudulent data set as it is highly class imbalanced. Our dataset consists of 2,84,807 instances of data, each representing one credit card transaction. The dataset has been taken from kaggle. The dataset is already normalized and the dimensionality reduction has already been carried out on the dataset using Principal Component Analysis (PCA)(Principal Component Analysis). The data set is extremely class imbalanced with 0.17% of instances belonging to the smaller/positive class i.e. fraud and rest of the 99.83% of the instances belongs from the negative class i.e. genuine transactions as shown in figure 2. The imbalance ratio of our dataset is 0.0017. The dataset is split in order to get the training set and test set. The dataset is divided into two parts in the ratio 4:1. The train dataset is then used to train all our models and then use the test dataset to predict class labels of the test dataset. The performance metrics Accuracy, F1 score, True Positive Rate and ROC-AUC score are used to check the efficiency of our classifiers on imbalanced data. The classifiers that are used are Naive Bayes, Decision Trees, Logistic Regression Binary Classifier and Random Forest Classifiers. The train dataset is then oversampled using SMOTE in order to make the number of minority class instances to be equal to that of the bigger class instances. After balancing the distribution of our dataset, we use our classifiers for prediction of the class labels of our testing dataset. Following this, Borderline SMOTE and ADASYN are used for oversampling the minority instances and their results are recorded. Undersampling techniques, NearMiss and Random Undersampling are also used to balance the train dataset. Undersampling removes data tuples from the majority class and makes the count of majority classes' instances to be equal to that of minority/smaller class. After applying undersampling on training dataset, we train our classifiers finally predict the class labels of our test dataset.

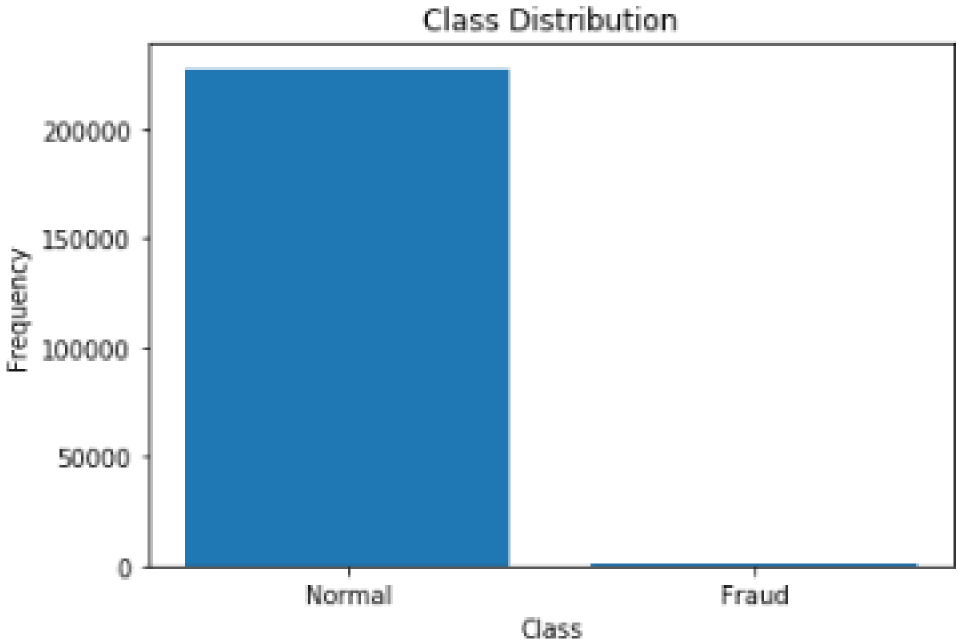

Figure 4.1: No. of instances in both the classes before oversampling.

Figure 4.2: No. of instances in both the classes after oversampling.

4.2 Model Training

The naive bayes algorithm is used to make classification of the test data into positive and negative classes after training the models with original data and oversampled data. We train two Naive Bayes models, one with original train data and the other with the oversampled train data.After training both the models, we get the accuracy of both the models by predicting the class labels of the Xtest dataset. After getting the predicted output labels from our models, we get the accuracy of both the models using confusion matrix. Similarly, we train Logistic Regression classifier, Decision Tree Classifier and Random Forest Classifier on the original train data and the oversampled train data and then predict the class labels of Xtest data using these models. Along with these we will keep using confusion matrix to get the accuracy of all these models. After testing the efficiency of the above algorithms individually, we applied ensemble models on our dataset to find out its efficiency in classification. We have applied various ensemble models including XGBoost, ADABoost, Voting Classifier. These ensemble models tend to give higher accuracy than all the individual models. We performed tests on all the given classification models with different parameters. The ratio of train and test data is set at 4:1. After applying oversampling and undersampling separately, the count of minority class tuples and the count of the majority class tuples are made equal.

4.3 Performance Metrics

The performance metrics of the classification models are generally the prediction accuracy. But, in case of imbalanced dataset, the model is biased in favour of the majority class, and the model classifies every data tuple as an instance that belonged to the negative class, the accuracy of our model will surely be very high but still, it is of no use. The rare events in our imbalanced dataset may even be treated as noise (outliers) or the noise might be misclassified as a rare event of a minority class. This is why, Accuracy cannot be treated as a good performance metric. This is why F1 score, True Positive Rate and ROC-AUC score are used as performance metrics[16]. F1 score is calculated using the precision and recall values of the predicted results from the confusion matrix.

Confusion matrix has four values, namely True Positives, True Negatives, False Positives and False Negatives. Precision value of a classification model is calculated as:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{4.1}$$

It can also be written as:

$$Precision = \frac{TruePositives}{TotalPredictedasPositives} \tag{4.2}$$

Similarly, the Recall value can be calculated as:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{4.3}$$

It can also be written as:

$$Recall = \frac{TruePositives}{TotalActualPositives} \tag{4.4}$$

The F1 score is a balance between both Precision and Recall and is very effective in case of imbalanced datasets. The F1 score of a classifier given TP,TN,FP,FN values can be calculated as:

$$F1score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.5}$$

The True Positive Rate is also known as Recall and is also known as Sensitivity. The TPR is an important performance metric in terms of imbalanced classes. ROC-AUC(Receiver Operator Characteristics - Area Under Curve) score is calculated by plotting the ROC curve by using Sensitivity and Specificity. Specificity can be calculated as:

$$Specificity = \frac{TrueNegatives}{TrueNegatives + FalsePositives} \tag{4.6}$$

The ROC curve is formed by plotting the Sensitivity value on the y-axis and (1-Specificity) values on the x-axis.

CHAPTER 5

RESULTS AND DISCUSSIONS

The Classifiers such as Naive Bayes' classifier, Logistic Regression binary classifier, Decision Trees and Random Forest classifiers are trained on the imbalanced dataset and the highest AUC Score is 0.87 and is obtained by the Decision Tree Classifier. After balancing the dataset by using SMOTE, ADASYN, Borderline SMOTE, Near Miss and Random under sampling separately, and training the aforementioned classifiers, the highest AUC Score is 0.95 which is obtained by Random Forest Classifier when trained using the dataset which is balanced using SMOTE. The ROCAUC score values of the individual classifiers are given in Table (5.1) and depicted as graph in Figure 5.9 The ROCAUC curves of the individual classifiers before and after oversampling are shown in Figures5.1 to 5.8

Table 5.1: ROCAUC score values of individual classifiers on imbalance and balance data

ROCAUC	Naive Bayes'	Logistic Regresion	Decision Trees	Random Forests
IMBALANCED	0.81	0.86	0.87	0.78
BALANCED BY OVERSAMPLING AND UNDER SAMPLING ALGORITHM				
SMOTE	0.88	0.91	0.92	0.95
ADASYN	0.86	0.87	0.88	0.91
BORDERLINE	0.84	0.9	0.9	0.93
NEARMISS	0.77	0.67	0.8	0.87
RANDOM	0.85	0.88	0.92	0.93

The aforementioned classifiers have the highest F1 score of 99.94% which is obtained by Random Forest when trained on imbalanced dataset. After balancing the dataset by using the aforementioned oversampling and under sampling techniques, the Random Forest Classifier obtains the highest F1 score of 99.95% when the dataset is balanced using SMOTE. The F1 scores of the individual classifiers is shown in Table (5.2) and Figure 5.10.

The aforementioned classifiers have the highest True Positive Rate of 72.16% which is obtained by Random Forest when trained on imbalanced dataset. After balancing

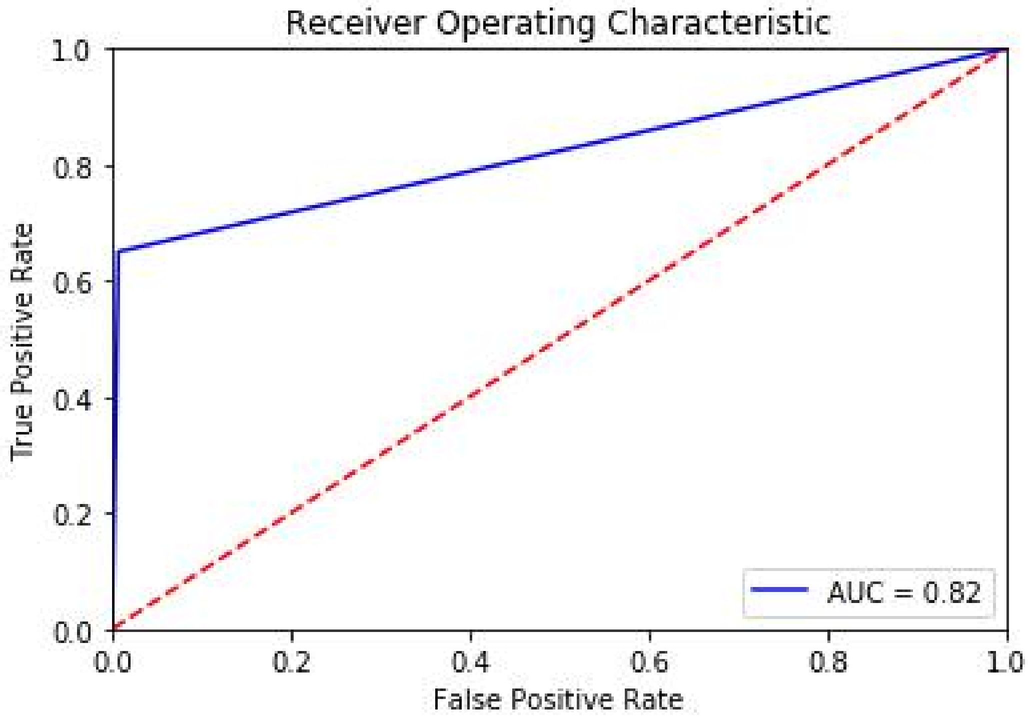

Figure 5.1: ROCAUC curve of the Naive Bayes' classifier on imbalance data

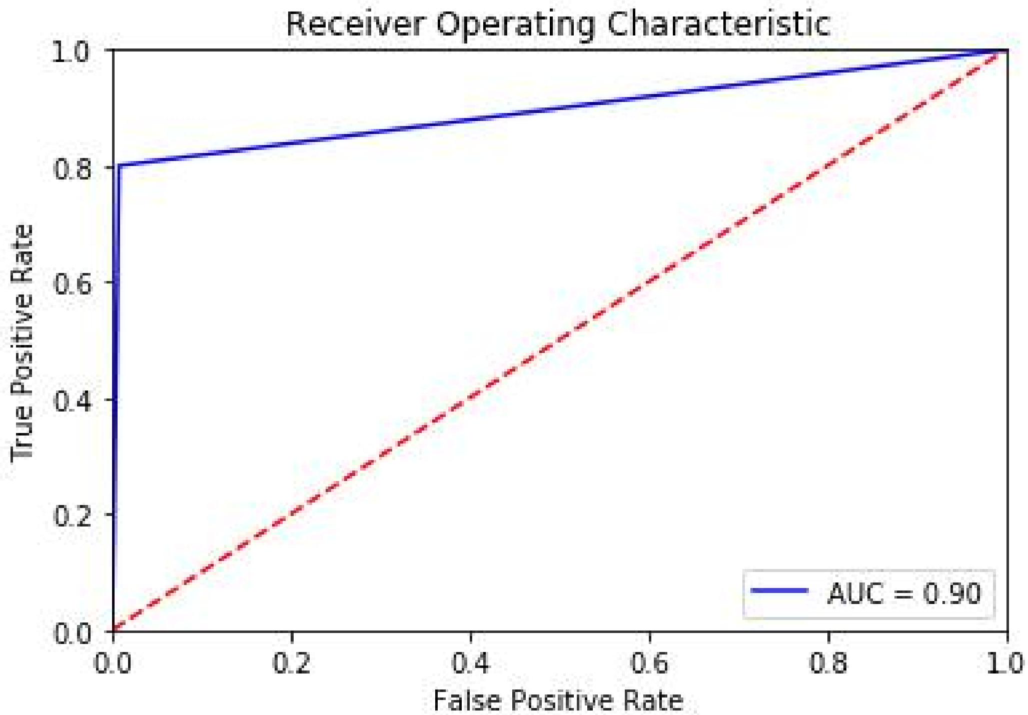

Figure 5.2: ROCAUC curve of Naive Bayes' classifier on balanced data

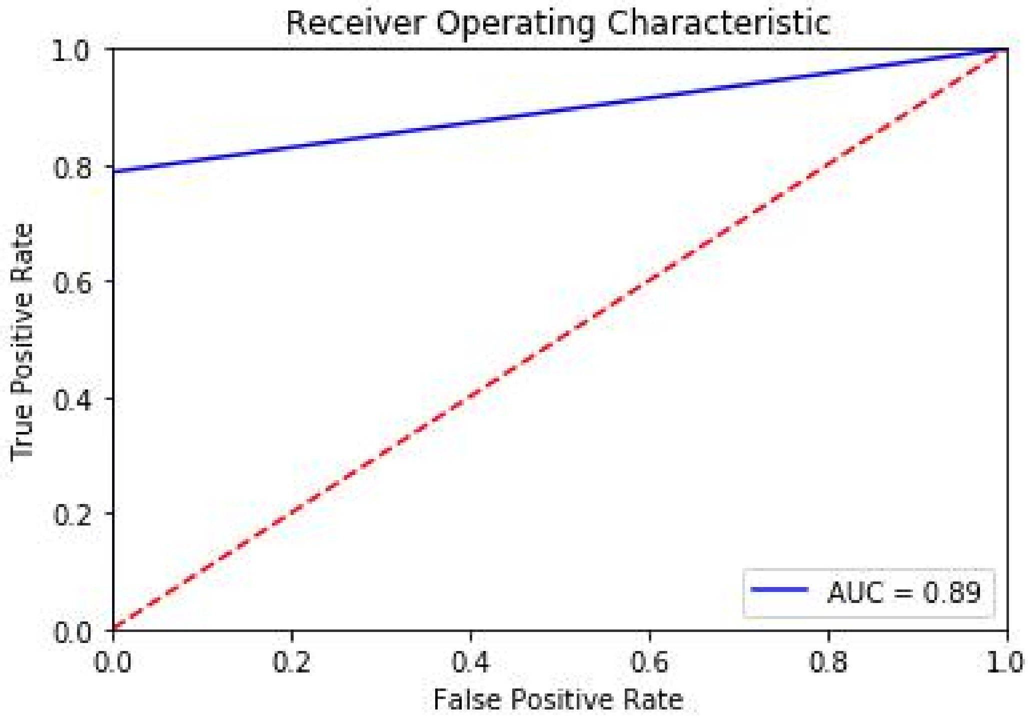

Figure 5.3: ROCAUC curve of the Decision Tree on imbalanced data

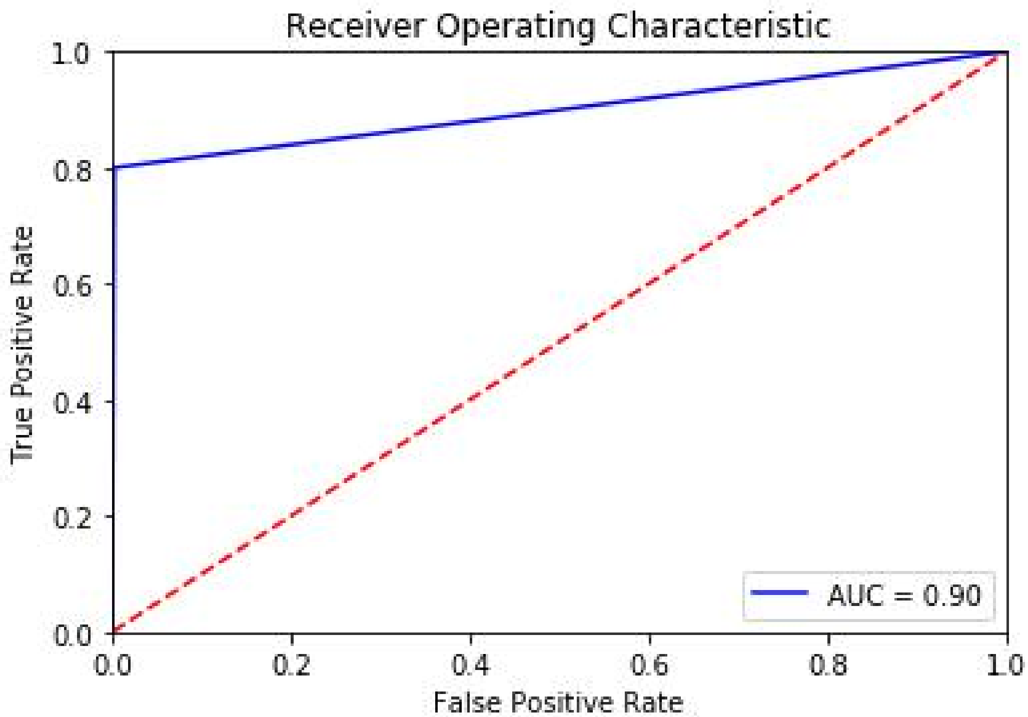

Figure 5.4: ROCAUC curve of the Decision Tree on balanced data

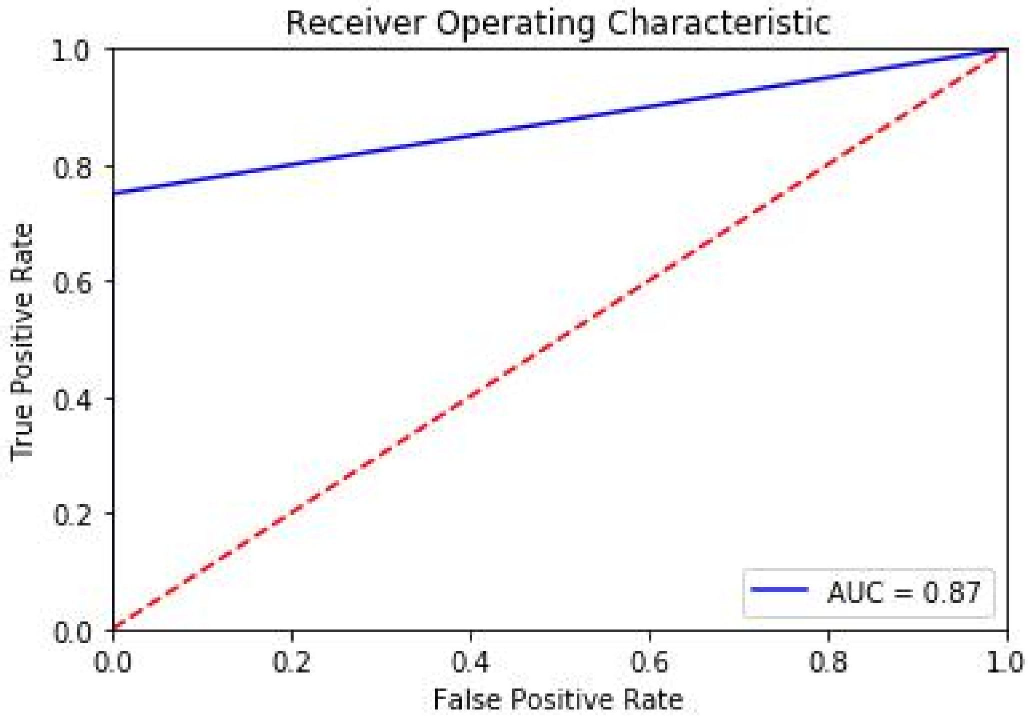

Figure 5.5: ROCAUC curve of the Random Forest on imbalance data

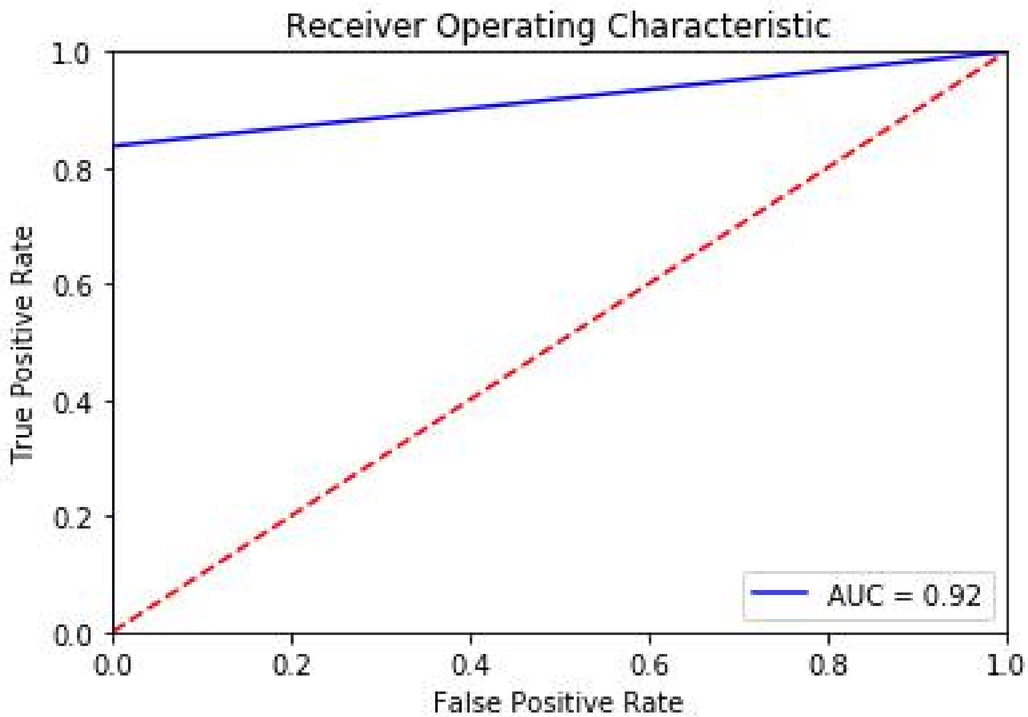

Figure 5.6: ROCAUC curve of the Random Forest on balanced data

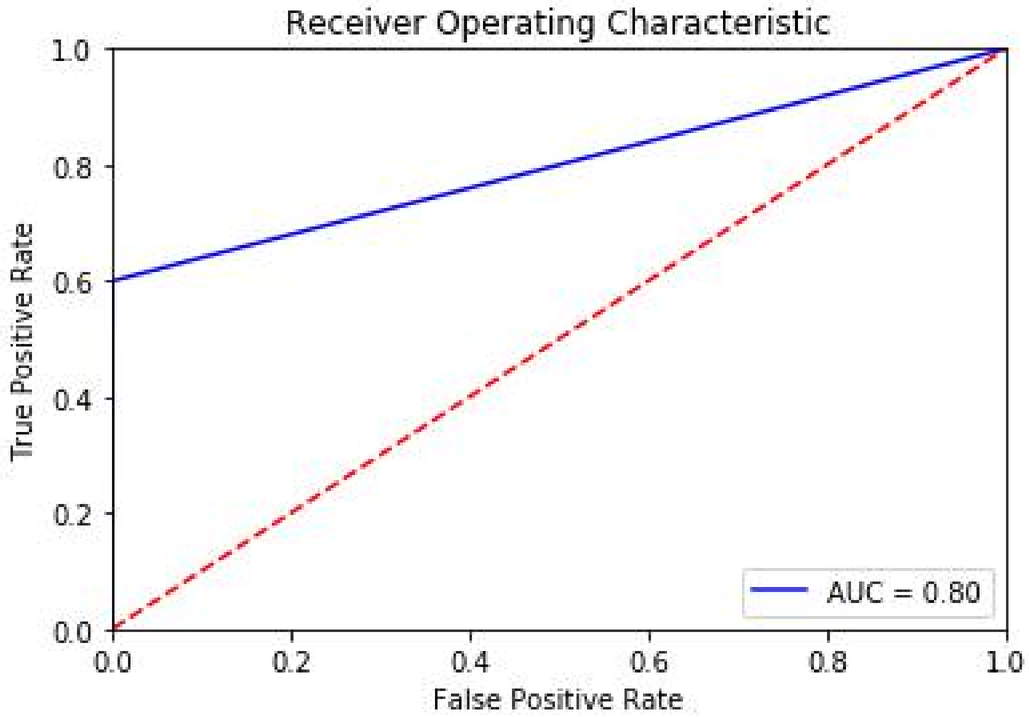

Figure 5.7: ROCAUC curve of the Logistic Regression on imbalance data

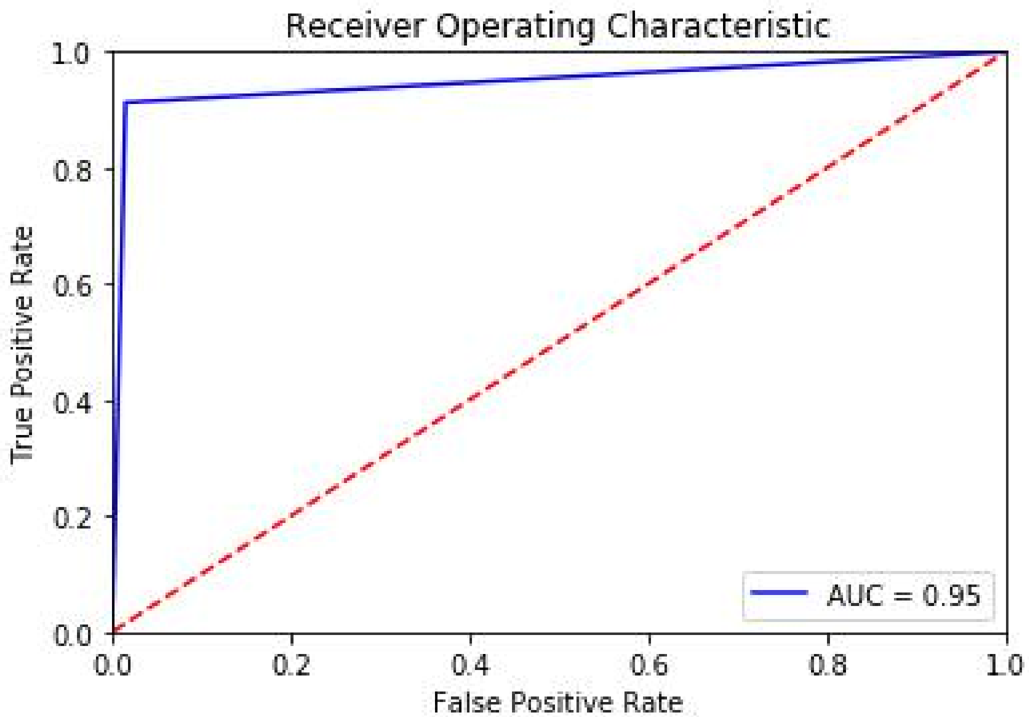

Figure 5.8: ROCAUC curve of the Logistic Regression on balanced data

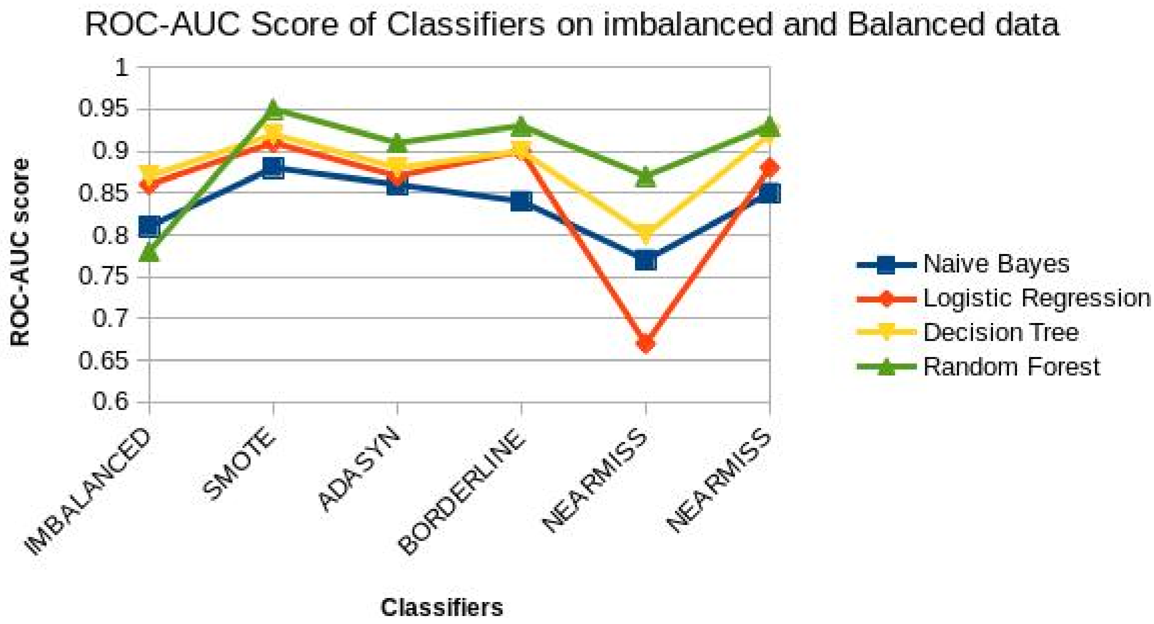

Figure 5.9: ROCAUC Score values of models on imbalance and Balance data

Table 5.2: F1 score values of individual models on imbalance and balance data

F1	Naive Bayes’	Logistic Regresion	Decision Trees	Random Forests
IMBALANCED	99.51	99.92	99.9	99.94
BALANCED BY OVERSAMPLING AND UNDER SAMPLING ALGORITHM				
SMOTE	99.51	97.41	99.79	99.95
ADASYN	99.46	97.65	99.95	92.41
BORDERLINE	99.6	98.7	99.9	99.95
NEARMISS	90.51	76.98	81.72	96.82
RANDOM	99.45	93.98	95.08	98

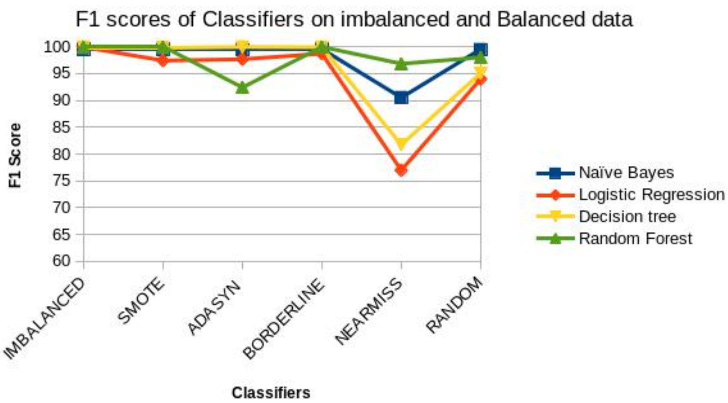

Figure 5.10: F1 score of models on imbalance and Balance data

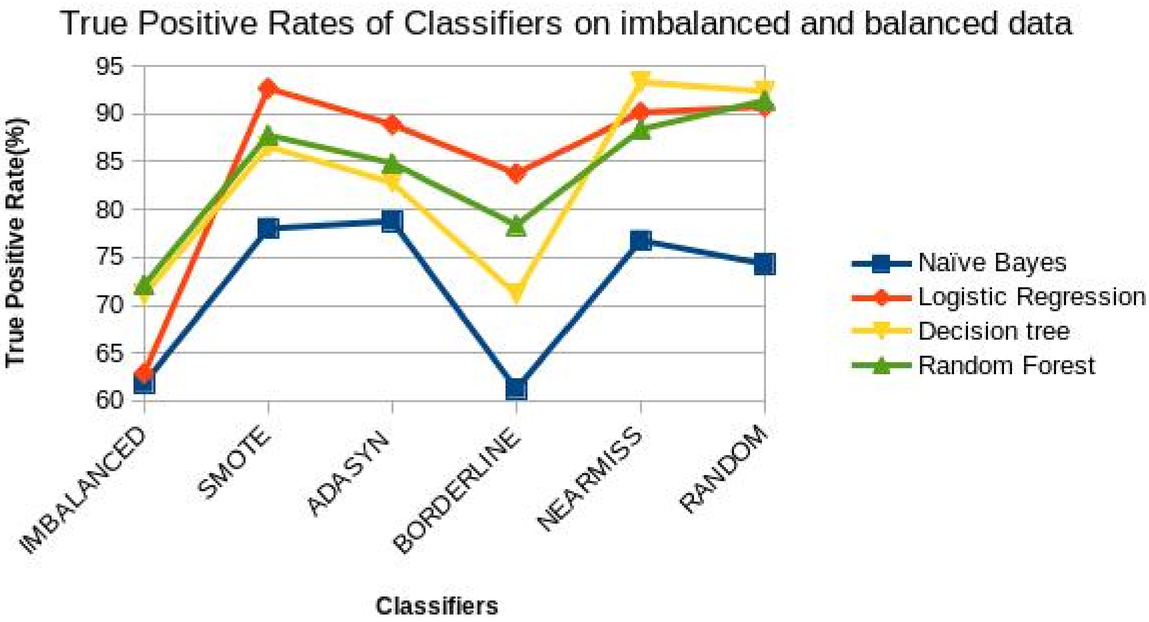

Figure 5.11: True Positive Rates of Classifier on imbalance and Balance data

the dataset by using the aforementioned oversampling and under sampling techniques, the Logistic Regression Classifier obtains the highest True Positive Rate of 92.68% when the dataset is balanced using SMOTE. The True Positive Rates of the individual classifiers is given in Table(5.3) and depicted as graph in Figure 5.11.

Table 5.3: True Positive Rates of classifiers on imbalanced and balanced data

TPR	Naive Bayes'	Logistic Regresion	Decision Trees	Random Forests
IMBALANCED	61.86	62.89	71.13	72.16
BALANCED BY OVERSAMPLING AND UNDER SAMPLING ALGORITHM				
SMOTE	78.05	92.68	86.59	87.8
ADASYN	78.79	88.89	82.83	84.85
BORDERLINE	61.26	83.78	71.17	78.38
NEARMISS	76.79	90.18	93.33	88.39
RANDOM	74.29	90.75	92.38	91.27

The Ensemble models like XGBoost, ADABoost, and Voting Classifier are trained on imbalanced data set and the highest ROCAUC score is 0.90 and is obtained by the XGBoost Ensemble Model. After balancing the dataset, the highest AUC score is 0.95, obtained by ADABoost when the dataset is balanced using ADASYN. The ROC-AUC scores of the ensemble classifiers is shown in Table (5.4) and Figure 5.18. The ROC-AUC curves of the ensemble models are shown in Figures 5.12 to 5.17. The ensemble classifiers have the highest F1 score of 99.95% which is obtained by XG-

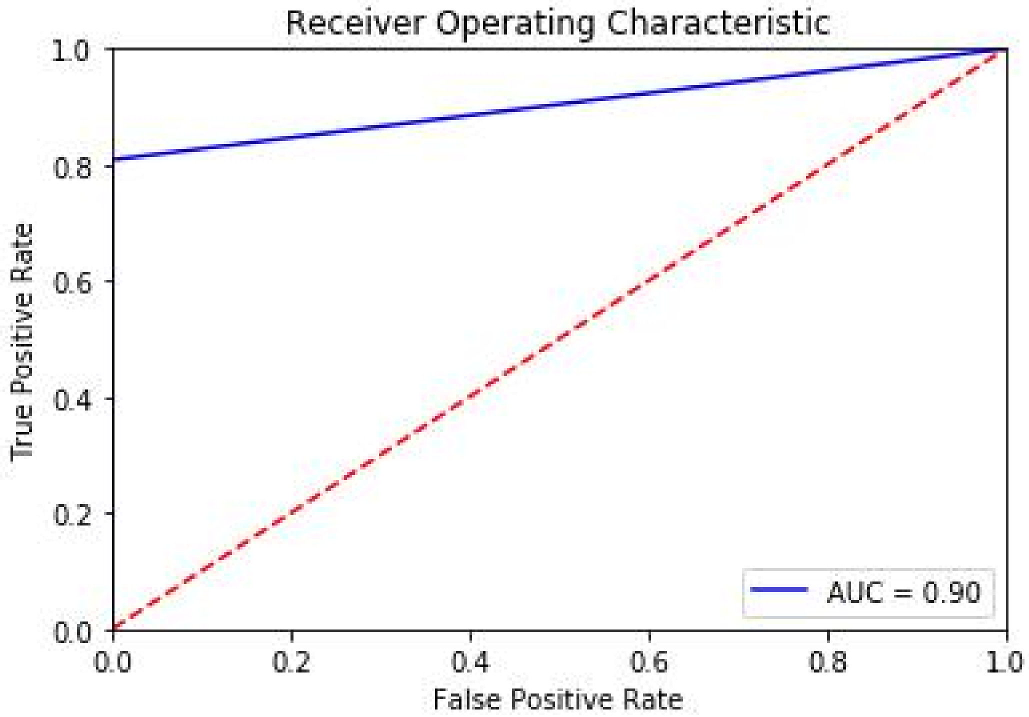

Figure 5.12: ROCAUC curve of the XGBoost classifier on imbalanced data

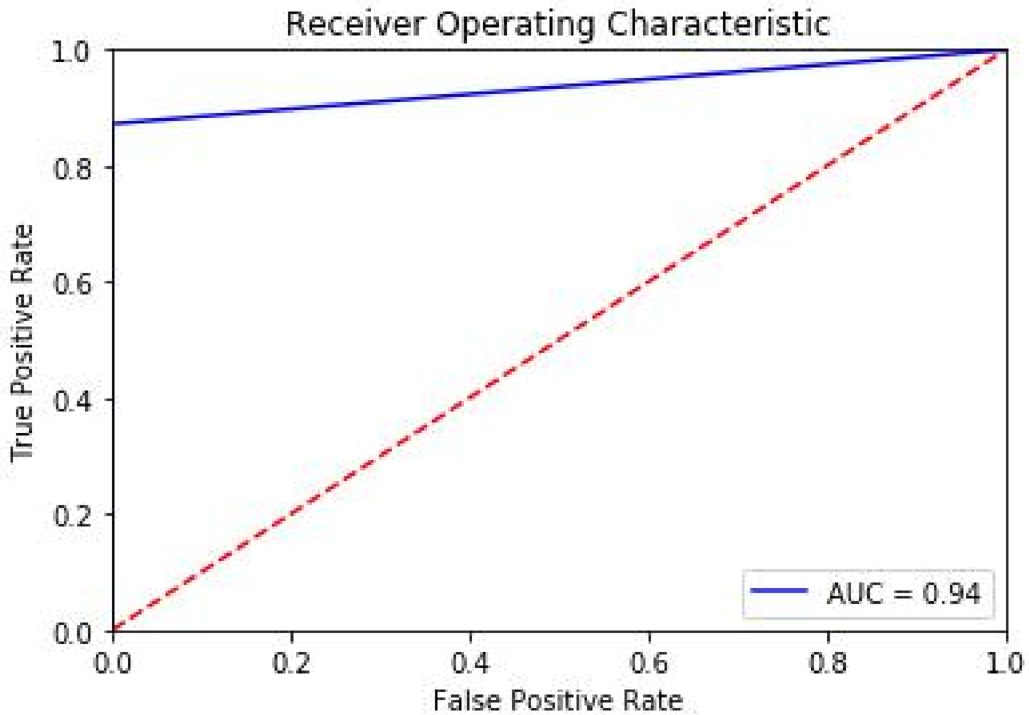

Figure 5.13: ROCAUC curve of XGBoost classifier on balanced data

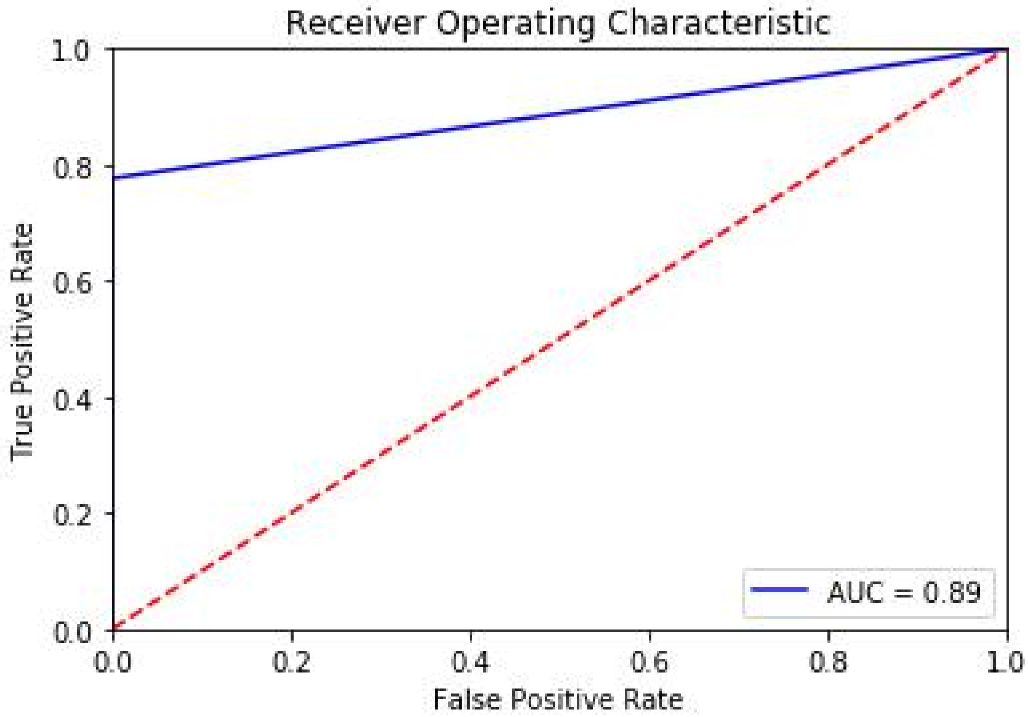

Figure 5.14: ROCAUC curve of ADABoost classifier on imbalanced data

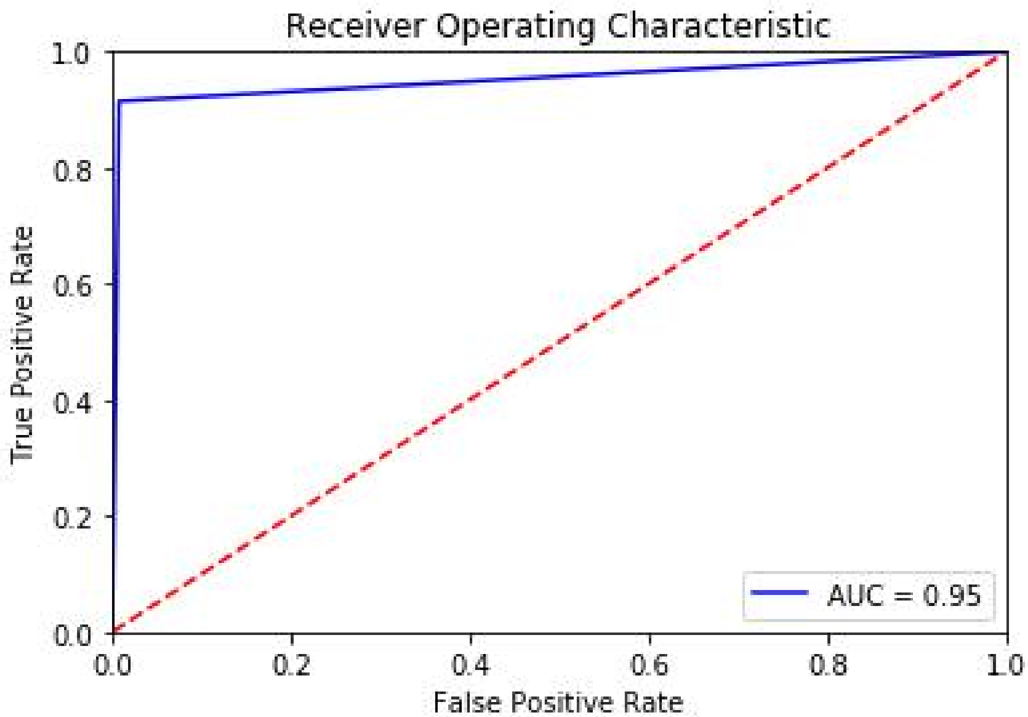

Figure 5.15: ROCAUC curve of ADABoost classifier on balanced data

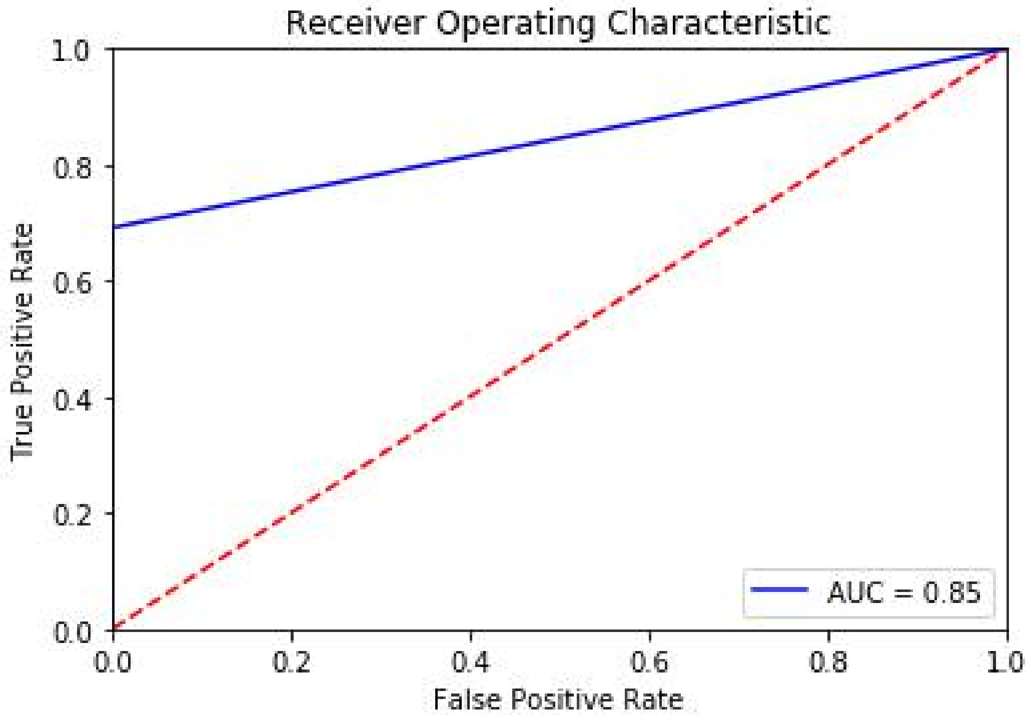

Figure 5.16: ROCAUC curve of Voting Classifier model on imbalanced data

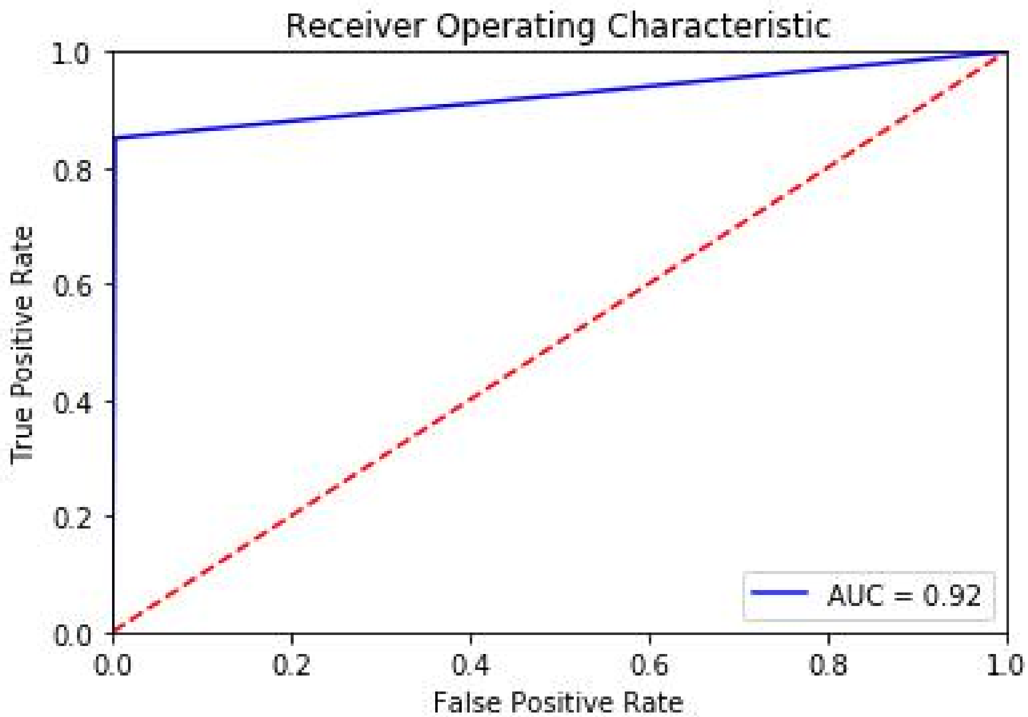

Figure 5.17: ROCAUC curve of Voting Classifier model on balanced data

Table 5.4: ROCAUC score values of Ensemble classifiers on imbalanced and balanced data

ROCAUC	XGBoost	ADABoost	Voting Classifier
IMBALANCED	0.9	0.87	0.87
BALANCED BY OVERSAMPLING AND UNDER SAMPLING ALGORITHM			
SMOTE	0.91	0.93	0.92
ADASYN	0.92	0.95	0.92
BORDERLINE	0.92	0.93	0.89
NEARMISS	0.88	0.78	0.85
RANDOM	0.93	0.93	0.93

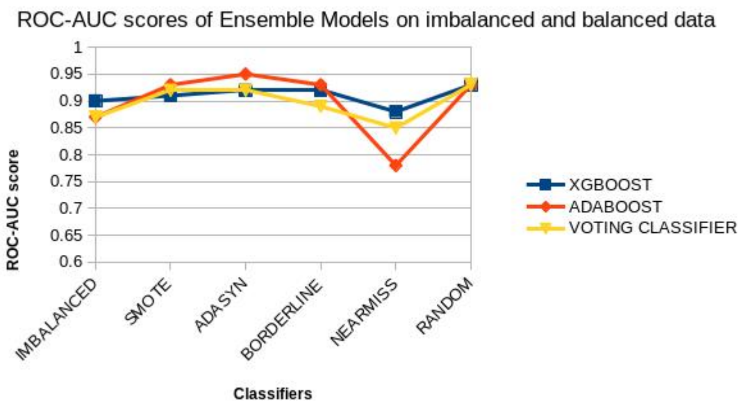

Figure 5.18: ROCAUC score values of Ensemble classifiers on imbalance and balance data

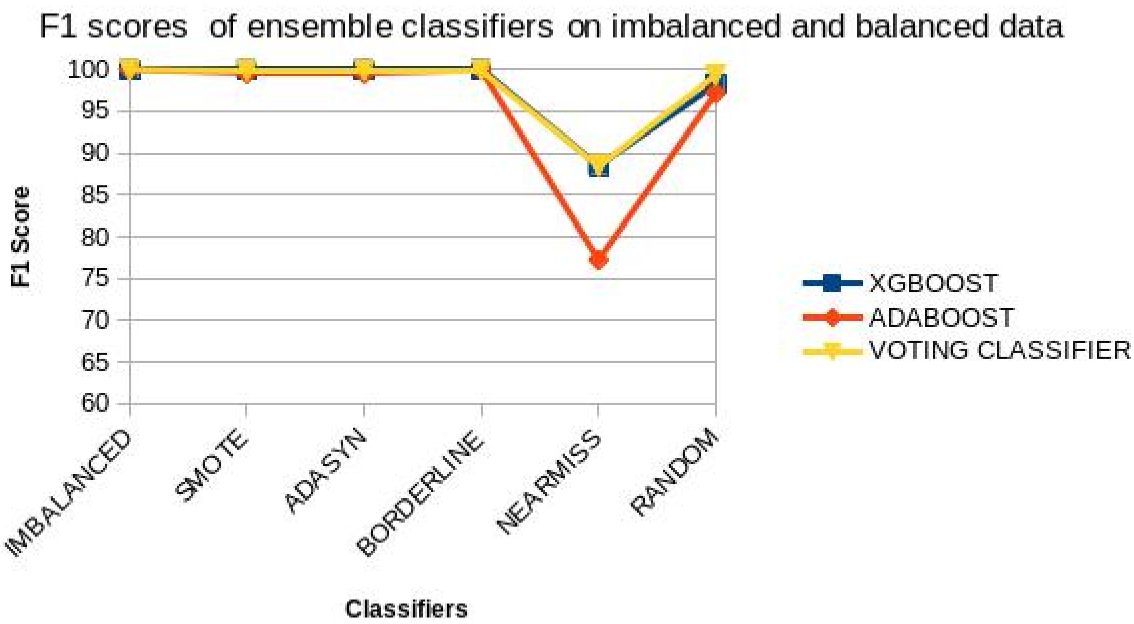

Figure 5.19: F1 score values of Ensemble classifiers on imbalance and balance data

Boost when trained on imbalanced dataset. After balancing the dataset by using the aforementioned oversampling and under sampling techniques, the ADASYN ensemble Classifier obtains the highest F1 score of 99.95% when the dataset is balanced using ADASYN. The F1 scores of the ensemble classifiers is shown in Table (5.5) and Figure 5.19. The aforementioned ensemble classifiers have the highest True Positive Rate of

Table 5.5: F1 scores of Ensemble classifiers on imbalance and balance data

F1	XGBoost	ADABoost	Voting Classifier
IMBALANCED	99.95	99.93	99.92
BALANCED BY OVERSAMPLING AND UNDER SAMPLING ALGORITHM			
SMOTE	99.94	99.49	99.78
ADASYN	99.95	99.49	99.86
BORDERLINE	99.97	99.86	99.90
NEARMISS	88.47	77.22	88.40
RANDOM	98.22	97.16	99.31

79.46% which is obtained by XGBoost when trained on the imbalanced dataset. After balancing the dataset by using the aforementioned oversampling and under sampling techniques, the XGBoost Classifier obtains the highest True Positive Rate of 95.56% when the dataset is balanced using Near Miss. The True Positive Rates of the ensemble classifiers is shown in Table (5.6) and Figure 5.20.

Table 5.6: True Positive Rates of Ensemble classifiers on imbalance and balance data

TPR	XGBoost	ADABoost	Voting Classifier
IMBALANCED	79.46	74.11	74.11
BALANCED BY OVERSAMPLING AND UNDER SAMPLING ALGORITHM			
SMOTE	82.14	86.61	84.82
ADASYN	84.31	81.18	84.31
BORDERLINE	83.33	86.67	78.89
NEARMISS	95.56	93.33	92.22
RANDOM	88.46	90.38	86.54

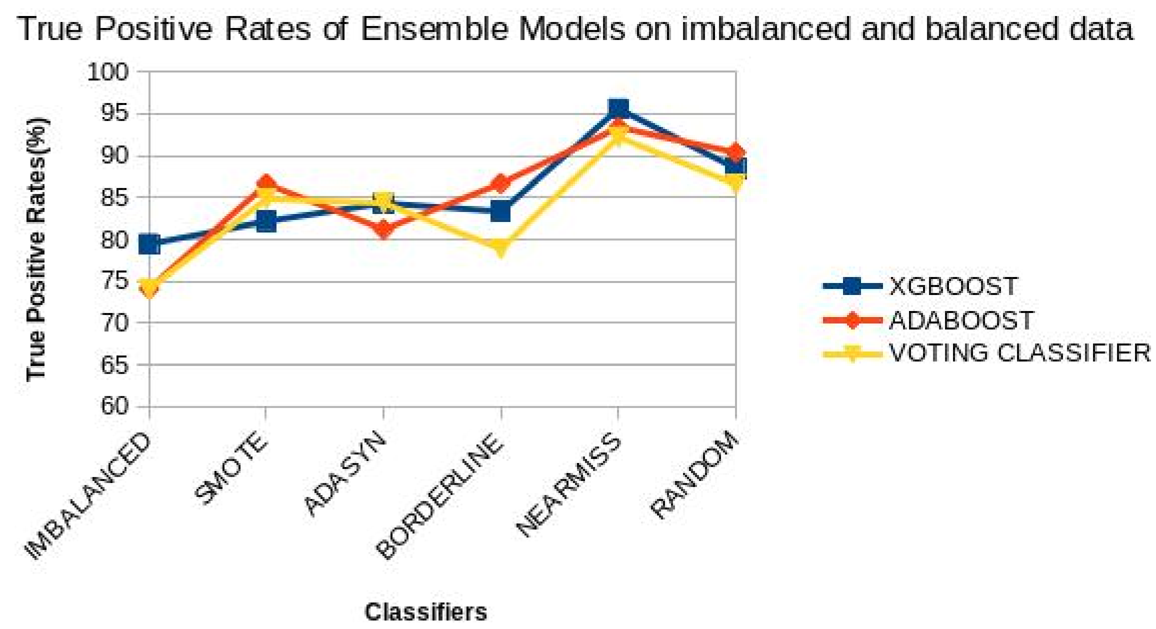

Figure 5.20: True Positive Rates of Ensemble classifiers on imbalance and balance data

CHAPTER 6

CONCLUSION

By looking at the results of all the algorithms that are used, it is seen that by removing the class imbalance and making the dataset balanced, the accuracy of our models have increased significantly. The True Positive rate of Naive Bayes classifier goes up by around 17%. The logistic regression classifier has shown an increase of about 28% in True Positive Rate. In the Decision Tree classifier model, the True Positive Rate increases by 17%. The Random Forest Classifier performs very well in classifying as it is also based on the ensemble model. The True Positive Rate of the Random Forest Classifier goes up by 20% and remains high at 91.43%. The ensemble classifiers XGBoost, ADABoost and Voting Classifiers show the best performance results in terms of ROC-AUC Score, F1 score and True Positive Rates. It has been seen that balancing the data using oversampling techniques helps the classifiers achieve better classification results than under sampling techniques. Also, the Ensemble models have outperformed the individual classifiers in most cases by providing better classification results.

REFERENCES

[1] Haixiang, Guo, et al. "Learning from class-imbalanced data: Review of methods and applications." Expert Systems with Applications 73 (2017): 220-239.
[2] Blagus, R., Lusa, L. SMOTE for high-dimensional class-imbalanced data. BMC Bioinformatics 14, 106 (2013).
[3] Han H., Wang WY., Mao BH. (2005) Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning. In: Huang DS., Zhang XP., Huang GB. (eds) Advances in Intelligent Computing. ICIC 2005. Lecture Notes in Computer Science, vol 3644. Springer, Berlin, Heidelberg.
[4] Mishra, Satwik, "Handling Imbalanced data: SMOTE vs. random undersampling." International Research of Engineering and Technology(IRJET)(2017).
[5] Yen, Show-Jane, and Yue-Shi Lee. "Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset." Intelligent Control and Automation. Springer, Berlin, Heidelberg, 2006. 731-740.
[6] T. Chen, T. He, M. Benesty, et al., Xgboost: extreme gradient boosting, R package version 0.4-2 (2015).
[7] Gómez-Ríos, Anabel, Julián Luengo, and Francisco Herrera. "A study on the noise label influence in boosting algorithms: AdaBoost, GBM and XGBoost." International Conference on Hybrid Artificial Intelligence Systems. Springer, Cham, 2017.
[8] Dreiseitl, Stephan, and Lucila Ohno-Machado. "Logistic regression and artificial neural network classification models: a methodology review." Journal of biomedical informatics 35.5-6 (2002): 352-359.
[9] Rish, Irina. "An empirical study of the naive Bayes classifier." IJCAI 2001 workshop on empirical methods in artificial intelligence. Vol. 3. No. 22. 2001.
[10] Safavian, S. Rasoul, and David Landgrebe. "A survey of decision tree classifier methodology." IEEE transactions on systems, man, and cybernetics 21.3 (1991): 660-674.
[11] Khoshgoftaar, Taghi M., Moiz Golawala, and Jason Van Hulse. "An empirical study of learning from imbalanced data using random forest." 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007). Vol. 2. IEEE, 2007.
[12] Ganganwar, Vaishali. "An overview of classification algorithms for imbalanced datasets." International Journal of Emerging Technology and Advanced Engineering 2.4 (2012): 42-47.
[13] Schapire, Robert E. "Explaining adaboost." Empirical inference. Springer, Berlin, Heidelberg, 2013. 37-52.
[14] Bauer, Eric, and Ron Kohavi. "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants." Machine learning 36.1-2 (1999): 105-139.
[15] Wold, Svante, Kim Esbensen, and Paul Geladi. "Principal component analysis." Chemometrics and intelligent laboratory systems 2.1-3 (1987): 37-52.
[16] Powers, David Martin. "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation." (2011).

APPENDIX A

CODE

In [1]:
```
import numpy as np
import pandas as pad
from sklearn.cluster import KMeans
from  sklearn.preprocessing  import  LabelEncoder
from sklearn.preprocessing
import MinMaxScaler import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import BorderlineSMOTE
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import train_test_split from
sklearn.datasets import make_blobs
from sklearn.linear_model import LogisticRegression
```

In [3]:
```
df=pad.read_csv("/home/sid/Documents/credit1.csv");
```

In [4]:
```
X=df.drop(['Class'],1)
Y=df['Class']
```

In [5]:
```
train_X,test_X,train_Y,test_Y=train_test_split(X,Y,test_size=0.2)
```

In [6]:
```
from imblearn.over_sampling import SMOTE
sm1 = SMOTE(random_state = 2,ratio=1)
Xoversampled,Yoversampled=sm1.fit_sample(train_X,train_Y)
```

In [7]:
```
from sklearn.naive_bayes import GaussianNB
```

In [8]:
```
modelgnb=GaussianNB()
```

In [9]:
```
modelgnb.fit(train_X,train_Y)
```

Out[9]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [10]: result_nb1=modelgnb.predict(test_X)

In [11]: confusion_matrix(result_nb1,test_Y)

Out[11]: array([[56497,   28],
                [385,   52]])

In [12]: accuracy_score(result_nb1,test_Y)

Out[12]: 0.9927495523331343

In [13]: roc_auc_score(result_nb1,test_Y)

Out[13]: 0.559248889487145

In [14]: f1_score(test_Y,result_nb1,average='weighted')

Out[14]: 0.9952414379913098

In [15]:
```
import sklearn.metrics as metr
preds = result_nb1
fp, tp, threshold = metr.roc_curve(test_Y, preds)
roc_auc = metrics.auc(fp, tp)

# method I: plt
import matplotlib.pyplot as p
p.title('ROC-AUC')
p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
p.legend(loc = 'lower right')
p.ylabel('Specificity')
p.xlabel('Sensitivity)


#toplot
p.show()
```

In [16]: model2=GaussianNB()

In [17]: model2.fit(Xoversampled,Yoversampled)

Out[17]: GaussianNB(prffriors=Noone, varsmoothing=1e-9)

In [18]: result_nb2=model2.predict(testX)

In [19]: accuracy_score(result_nb2,testY)

Out[19]: 0.9922228854323936

```
In [20]: confusion_matrix(result_nb2,testY)

Out[20]: array([[56455,      16],
               [  427,      64]])

In [21]: roc_auc_score(result_nb2,testY)

Out[21]: 0.565031450455925

In [22]: f1_score(testY,result_nb2,average='weighted')

Out[22]: 0.9950077301748161

In [23]: import sklearn.metrics as metr
         preds = result_nb2
         fp, tp, threshold = metr.roc_curve(test_Y, preds)
         roc_auc = metrics.auc(fp, tp)

         # method I: plt
         import matplotlib.pyplot as p
         p.title('ROC-AUC')
         p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
         p.legend(loc = 'lower right')
         p.ylabel('Specificity')
         p.xlabel('Sensitivity)


         #toplot
         p.show()


`        In [24]: from sklearn.tree import DecisonTreeClassifier

In [25]: dtree1=DecisionTreeClassifier()

In [26]: dtree1.fit(train_X,train_Y)

Out[26]: DecisionTreeClassifier(min_impurity_decrease=0.0,
                                max_depth=None,class_weight=None,
                                , min_impurity_split=None,
                                random_state=None, splitter='best',min_samples_leaf=1,
                                min_samples_split=2, presort=False,
                                min_weight_fraction_leaf=0.0, criterion='gini',
                                max_features=None, max_leaf_nodes=None
                                )
In [27]: result_dt1=dtree1.predict(test_X)

In [28]: print(accuracy_score(result_dt1,test_Y))

0.9991573329588147
```

In [29]: confusion_matrix(result_dt1,test_Y)

Out[29]: array([[56851,   17],
                [31,   63]])

In [30]: f1_score(test_Y,result_dt1,average='weighted')

Out[30]: 0.99919118150059

In [31]: roc_auc_score(result_dt1,test_Y)

Out[31]: 0.834956914033095

In [32]:
```
import sklearn.metrics as metr
preds = result_dt1
fp, tp, threshold = metr.roc_curve(test_Y, preds)
roc_auc = metrics.auc(fp, tp)

# method I: plt
import matplotlib.pyplot as p
p.title('ROC-AUC')
p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
p.legend(loc = 'lower right')
p.ylabel('Specificity')
p.xlabel('Sensitivity)

#toplot
p.show()
```

In [33]: dtree2=DecisionTreeClassifier()

In [34]: dtree2.fit(Xoversampled,Yoversampled)

Out[34 DecisionTreeClassifier(min_impurity_decrease=0.0,
max_depth=None,class_weight=None,
, min_impurity_split=None,
random_state=None, splitter='best',min_samples_leaf=1,
min_samples_split=2, presort=False,
min_weight_fraction_leaf=0.0, criterion='gini',
max_features=None, max_leaf_nodes=None
)

In [35]: result_dt2=dtree2.predict(test_X)

In [36]: confusion_matrix(result_dt2,test_Y)

```
Out[36]: array([[56767,      16],
                [  115,      64]])
```

```
In [37]: print(accuracy_score(result_dt2,test_Y))
```

```
0.9977002212000983
```

```
In [38]: roc_auc_score(result_dt2,test_Y)
```

```
Out[38]: 0.678630062483293
```

```
In [39]: f1_score(test_Y, result_dt2, average='weighted')
```

```
Out[39]: 0.9981387526827198
```

```
In [40]: import sklearn.metrics as metr
    preds = result_dt1
    fp, tp, threshold = metr.roc_curve(test_Y, preds)
    roc_auc = metrics.auc(fp, tp)

    # method I: plt
    import matplotlib.pyplot as p
    p.title('ROC-AUC')
    p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
    p.legend(loc = 'lower right')
    p.ylabel('Specificity')
    p.xlabel('Sensitivity)

    #toplot
    p.show()
```

```
In [40]: from sklearn.ensemble import RandomForestClassifier
```

```
In [42]: rfc1=RandomForestClassifier()
```

```
In [43]: rfc1.fit(train_X,train_Y)
```

```
Out[43]: RandomForestClassifier( class_weight=None, bootstrap=True, max_depth=None,
            criterion='gini' , max_leaf_nodes=None, max_features='auto',
            min_impurity_split=None,min_impurity_decrease=0.0,
            min_samples_split=2, min_samples_leaf=1,
            n_estimators=10,
            min_weight_fraction_leaf=0.0,
            oob_score=False, n_jobs=None,
            verbose=0,random_state=None,
            warm_start=False)
```

In [44]: result_rf1=rfc1.predict(test_X)

In [45]: print(accuracy_score(result_rf1,test_Y))

0.9994557775359011

In [46]: f1_score(test_Y, result_rf1, average='weighted')

Out[46]: 0.9994395805109743

In [47]: roc_auc_score(result_rf1,test_Y)

Out[47]: 0.9223594365404959

In [48]: confusion_matrix(result_rf1,test_Y)

Out[48]: array([[56871, 20],
[11, 60]])

In [49]:
```
import sklearn.metrics as metr
preds = result_rf1
fp, tp, threshold = metr.roc_curve(test_Y, preds)
roc_auc = metrics.auc(fp, tp)

# method I: plt
import matplotlib.pyplot as p
p.title('ROC-AUC')
p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
p.legend(loc = 'lower right')
p.ylabel('Specificity')
p.xlabel('Sensitivity)

#toplot
p.show()
```

In [50]: rfc2=RandomForestClassifier()

In [51]: rfc2.fit(Xoversampled,Yoversampled)

Out[51]: RandomForestClassifier(class_weight=None, bootstrap=True, max_depth=None,
criterion='gini' , max_leaf_nodes=None, max_features='auto',
min_impurity_split=None,min_impurity_decrease=0.0,
min_samples_split=2, min_samples_leaf=1,
n_estimators=10,
min_weight_fraction_leaf=0.0,
oob_score=False, n_jobs=None,
verbose=0,random_state=None,
warm_start=False)

In [52]: result_rf2=rfc2.predict(test_X)

In [53]: print(accuracy_score(result_rf2,test_Y))

0.9995259997893332

In [54]: f1_score(testY, result_rf2, average='weighted')

Out[54]: 0.9995274697563976

In [55]: confusion_matrix(result_rf2,test_Y)

```
Out[55]: array([[56868,     13],
                [    14,     67]])
```

In [56]: roc_auc_score(result_rf2,test_Y)

Out[56]: 0.9134659732545378

In [57]:
```
import sklearn.metrics as metr
preds = result_rf2
fp, tp, threshold = metr.roc_curve(test_Y, preds)
roc_auc = metrics.auc(fp, tp)

# method I: plt
import matplotlib.pyplot as p
p.title('ROC-AUC')
p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
p.legend(loc = 'lower right')
p.ylabel('Specificity')
p.xlabel('Sensitivity)

#toplot
p.show()
```

In [58]: from sklearn.linear_model import LogisticRegression

In [59]: clf = LogisticRegression(random_state=0).fit(train_X, train_Y)

In [60]: clf2 = LogisticRegression(random_state=0).fit(Xoversampled,Yoversampled)

In [61]: result_lr1=clf.predict(test_X)

In [62]: result_lr2=clf2.predict(test_X)

In [63]: accuracy_score(result_lr1,test_Y)

Out[63]: 0.9987886661282961

```
In [64]: accuracy_score(result_lr2,test_Y)

Out[64]: 0.985025104455602

In [65]: confusion_matrix(result_lr1,test_Y)

Out[65]: array([[56845,    32],
                [   37,    48]])

In [66]: confusion_matrix(result_lr2,test_Y)

Out[66]: array([[56036,       7],
                [  846,      73]])

In [67]: f1_score(result_lr1,test_Y,average='weighted')

Out[67]: 0.9987703392054017

In [68]: f1_score(result_lr2,test_Y,average='weighted')

Out[68]: 0.9787924770832608

In [69]: testY=np.array(test_Y)

In [70]: roc_auc_score(result_lr1,test_Y)

Out[70]: 0.7820716323873291

In [71]: roc_auc_score(result_lr2,test_Y)

Out[71]: 0.5396546317409741

In [72]: import sklearn.metrics as metr
         preds = result_lr1
         fp, tp, threshold = metr.roc_curve(test_Y, preds)
         roc_auc = metrics.auc(fp, tp)

         # method I: plt
         import matplotlib.pyplot as p
         p.title('ROC-AUC')
         p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
         p.legend(loc = 'lower right')
         p.ylabel('Specificity')
         p.xlabel('Sensitivity)

         #toplot
         p.show()
```

```
In [73]: import sklearn.metrics as metr
         preds = result_lr2
         fp, tp, threshold = metr.roc_curve(test_Y, preds)
         roc_auc = metrics.auc(fp, tp)

         # method I: plt
         import matplotlib.pyplot as p
         p.title('ROC-AUC')
         p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
         p.legend(loc = 'lower right')
         p.ylabel('Specificity')
         p.xlabel('Sensitivity)

         #toplot
         p.show()


In [1]:  import pandas as pad
         import numpy as np
         from sklearn.cluster import KMeans
         from sklearn.preprocessing import LabelEncoder
         from sklearn.preprocessing import MinMaxScaler
         import seaborn as sns
         import matplotlib.pyplot as plt
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import accuracy_score
         from imblearn.over_sampling import
         BorderlineSMOTE from sklearn.metrics
         import f1_score
         from sklearn.metrics import roc_auc_score
         import numpy as np
         import matplotlib.pyplot as plt
         import sklearn
         from sklearn.model_selection import
         train_test_split
         import pandas as pd
         from keras.optimizers import SGD
         import tensorflow as tf


In [2]: df=pad.read_csv("/home/sid/Documents/credit1.csv");

In [3]: X=df.drop(['Class'],1)
        y=df['Class']

In [4]: train_X,test_X,train_Y,test_Y=train_test_split(X,y,test_size=0.2)

In [5]: train_X=np.array(train_X)
```

```
        Test_X=np.array(test_X)
        Train_Y=np.array(train_Y)
        Test_Y=np.array(test_Y)
```

```
In [6]: from imblearn.over_sampling import SMOTE
        Sm1 = SMOTE(random_state = 2)
        Xoversampled,Yoversampled=Sm1.fit_sample(trainX,trainY)
```

```
In [7]: from xgboost import
        XGBClassifier
        my_model = XGBClassifier(
        colsaple_bytree=0.7, subample=0.7, max_dpth=23, n_estimators=150, n_jobs=28, ev
        my_model.fit(trainX, trainY)
```

```
Out[7]: XGBClassifier(boster='gbtree', base_scre=0.5, colsample_bynde=1,
                colsample_byevel=1,colsmple_bytree=0.7,
                gamma=0,eval_metric=['errr', 'auc'],
                max_delta_step=0, learnig_rate=0.1, in_hild_weight=1, ax_depth=23,
                n_estimators=150,missing=Nne,
                n_job=28,
                objecive='binary:logistic', random_stte=0, nhread=None,
                 regalpha=0,
                reg_lamda=1, calpos_weight=1, eed=None, slent=None,
                subsaple=0.7, verbsity=1)
```

```
In [8]: result_xgb1 = my_model.predict(test_X)
```

```
In [9]: accuracy_score(test_Y,result_xgb1)
```

```
Out[9]: 0.9996313331694814
```

```
In [10]: roc_auc_score(result_xgb1,test_Y)
```

```
Out[10]: 0.9808544387246573
```

```
In [11]: f1_score(test_Y,result_xgb1,average='weighted')
```

```
Out[11]: 0.9996153748109079
```

```
In [12]: confusion_matrix(result_xgb1,test_Y)
```

```
Out[12]: array([[56865,     18],
               [    3,     76]])
```

```
In [13]: import sklearn.metrics as metr
         preds = result_xgb1
         fp, tp, threshold = metr.roc_curve(test_Y, preds)
         roc_auc = metrics.auc(fp, tp)

         # method I: plt
         import matplotlib.pyplot as p
         p.title('ROC-AUC')
         p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
```

```
p.legend(loc = 'lower right')
p.ylabel('Specificity')
p.xlabel('Sensitivity)


#toplot
p.show()
```

In [14]:
```
my_model2 = XGBClassifier(
            colsmple_bytree=0.7, subsaple=0.7, ma_depth=23, n_stimators=150,
n_jos=28)
 my_model2.fit(Xoversampled, Yoversampled)
```

Out[14]:
```
XGBClassifier(boster='gbtree', base_scre=0.5, colsample_bynde=1,
              colsample_byevel=1,colsmple_bytree=0.7,
              gamma=0,eval_metric=['errr', 'auc'],
              max_delta_step=0, learnig_rate=0.1, in_hild_weight=1, ax_depth=23,
              n_estimators=150,missing=Nne,
              n_job=28,
              objecive='binary:logistic', random_stte=0, nhread=None,
               regalpha=0,
              reg_lamda=1, calpos_weight=1, eed=None, slent=None,
              subsaple=0.7, verbsity=1)
```

In [15]: `result_xgb2 = my_model2.predict(testX)`

In [16]: `accuracy_score(testY,result_xgb2)`

Out[16]: 0.9995435553526912

In [17]: `roc_auc_score(result_xgb2,testY)`

Out[17]: 0.9269778221315607

In [18]: `f1_score(testY,result_xgb2,average='weighted')`

Out[18]: 0.9995459536796755

In [19]: `confusion_matrix(testY,result_xgb2)`

Out[19]:
```
array([[56854,    14],
       [   12,    82]])
```

In [21]: `from sklearn.ensemble import AdaBoostClassifier`

In [22]: `clf = AdaBoostClassifier(n_estimators=100, random_state=0)`

```
In [23]: clf.fit(train_X,train_Y)

Out[23]: AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                            n_estimators=100, random_state=0)

In [24]: result_ada1=clf.predict(test_X)

In [25]: accuracy_score(result_ada1,test_Y)

Out[25]: 0.9994206664091185

In [26]: roc_auc_score(result_ada1,test_Y)

Out[26]: 0.9292271558130083

In [27]: f1_score(test_Y,result_ada1,average='weighted')

Out[27]: 0.9994061250766009

In [28]: confusion_matrix(test_Y,result_ada1)

Out[28]: array([[56856,    12],
                [   21,    73]])

In [29]: import sklearn.metrics as metr
         preds = result_xgb2
         fp, tp, threshold = metr.roc_curve(test_Y, preds)
         roc_auc = metrics.auc(fp, tp)

         # method I: plt
         import matplotlib.pyplot as p
         p.title('ROC-AUC')
         p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
         p.legend(loc = 'lower right')
         p.ylabel('Specificity')
         p.xlabel('Sensitivity)


         #toplot
         p.show()

In [30]: clf2 = AdaBoostClassifier(n_estimators=100, random_state=0)

In [31]: clf2.fit(Xoversampled,Yoversampled)

Out[31]: AdaBoostClassifier(algoritm='SAMME.R', bae_estimator=None, larning_rate=1.0,
                            n_etimators=100, radom_state=0)
```

In [32]: result_ada2=clf2.predict(test_X)

In [33]: accuracy_score(result_ada2,test_Y)

Out[33]: 0.9922931076858257

In [34]: roc_auc_score(result_ada2,test_Y)

Out[34]: 0.5831012815576

In [35]: f1_score(test_Y,result_ada2,average='weighted')

Out[35]: 0.994946493100334

In [36]: confusion_matrix(test_Y,result_ada2)

Out[36]:
```
array([[56437,    431],
       [    8,     86]])
```

In [37]:
```
import sklearn.metrics as metr
preds = result_ada2
fp, tp, threshold = metr.roc_curve(test_Y, preds)
roc_auc = metrics.auc(fp, tp)

# method I: plt
import matplotlib.pyplot as p
p.title('ROC-AUC')
p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
p.legend(loc = 'lower right')
p.ylabel('Specificity')
p.xlabel('Sensitivity)


#toplot
p.show()
```

In [38]:
```
from sklern.linear_model import LogisticRegression
from sklarn.naive_bayes import GaussianNB
from sklarn.ensemble import RandomForestClassifier
from sklearn.enemble import VotingClassifier
```

In [39]:
```
vclf1 = Logisticegression(random_stte=1)
vclf2 = RandoForestClassifier(n_estmators=50, radom_state=1)
vclf3 = GaussinNB()
c=[vclf1,vclf2,vclf3]
```

In [41]:
```
for i in c:
    i.fit(trainX,trainY)
```

In [42]:
```
x=len(c)
y=len(test_Y)
z=np.zeros((x,y))
```

In [43]:
```
for i in range(0, length(c)):
 z[i]=c[i].predict(test_X)
```

In [44]:
```
result_vot1=np.zeros(y)
for i in range(0,y):
        count1=0
        for j in range(0,x):
                if(z[j][i]==1):
                        count1=count1+1
        if(count1>1):
                result_vot1[i]=1
        else:
                result_vot1[i]=0
```

In [45]:
```
accuracy_score(result_vot1,testY)
```

Out[45]: 0.9992802219023208

In [46]:
```
roc_auc_score(result_vot1,testY)
```

Out[46]: 0.9218230218406012

In [47]:
```
f1_score(testY,result_vot1,average='weighted')
```

Out[47]: 0.9992444972430666

In [48]:
```
confusion_matrix(result_vot1,testY)
```

Out[48]:
```
array([[56856,    29],
       [   12,    65]])
```

In [49]
```
import sklearn.metrics as metr
preds = result_vot1
fp, tp, threshold = metr.roc_curve(test_Y, preds)
roc_auc = metrics.auc(fp, tp)

# method I: plt
import matplotlib.pyplot as p
p.title('ROC-AUC')
p.plot(fpr, tpr, 'b', label = 'AUC = %0.3f' % roc_auc)
p.legend(loc = 'lower right')
p.ylabel('Specificity')
p.xlabel('Sensitivity)
```

```
#toplot
p.show()
```

In [50]:
```
for i in c:
    i.fit(Xoversampled,Yoversampled)
```

In [51]:
```
x=len(c)
y=len(testY)
z=np.zeros((x,y))
```

In [52]:
```
for i in range(0,len(c)):
    z[i]=c[i].predict(testX)
```

In [54]:
```
accuracy_score(result_vot2,testY)
```

Out[54]: 0.996875109722271

In [55]:
```
roc_auc_score(result_vot2,testY)
```

Out[55]: 0.6638110086195783

In [56]:
```
f1_score(testY,result_vot2,average='weighted')
```

Out[56]: 0.9975664391933199

Format - I

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University u/s 3 of UGC Act, 1956)

Office of Controller of Examinations

REPORT FOR PLAGIARISM CHECK ON THE DISSERTATION/PROJECT REPORTS FOR UG/PG PROGRAMMES

(To be attached in the dissertation/ project report)

1	Name of the Candidate **(IN BLOCK LETTERS)**	**PANKAJ THAKUR, SIDDHARTH AGARWAL**
2	Address of the Candidate	Ward 06, bakhtiyarpur, Pusa, Samastipur,Bihar Nh 33, beside Bank of India, Dhalbhumgarh, Jharkhand **Mobile Number :** 9771186585, 7549116805
3	Registration Number	RA1611003011072, RA1611003011132
4	Date of Birth	05-01-1999, 08-01-1999
5	Department	Computer Science and Engineering
6	Faculty	Engineering and Technology
7	Title of the Dissertation/Project	Ensemble Classification for Class Imbalanced Credit card Fraudulent data
8	Whether the above project/dissertation is done by	Individual or group : group (Strike whichever is not applicable) a) If the project/ dissertation is done in group, then how many students together completed the project : 2 b) Mention the Name & Register number of other candidates : Siddharth Agarwal(RA1611003011132) Pankaj Thakur(RA1611003011072)
9	Name and address of the Supervisor / Guide	Mrs. S.Priya, Asst. Professor(Sr.G.), Dept. of Computer Science and Engineering, TP-8th floor SRM Institute of Science and Technology, Chennai **Mail ID :** priyas3@srmist.edu.in **Mobile Number :** 9965930862
10	Name and address of the Co-Supervisor / Co- Guide (if any)	**Mail ID : Mobile Number :**

11	Software Used			
12	Date of Verification			
13	**Plagiarism Details: (to attach the final report from the software)**			
Chapter	**Title of the Chapter**	**Percentage of similarity index (including self citation)**	**Percentage of similarity index (Excluding self citation)**	**% of plagiarism after excluding Quotes, Bibliography, etc.,**
1	Introduction	4%	2%	2%
2	Literature Survey	0%	0%	0%
3	Proposed Work	1%	1%	0%
4	Implementation	4%	2%	2%
5	Results and Discussions	0%	0%	0%
6	Conclusion	0%	0%	0%
7				
8				
9				
10				
Appendices				

I / We declare that the above information have been verified and found true to the best of my / our knowledge.

Pankaj Thakur Siddharth Agarwal **Signature of the Candidate**	**Name & Signature of the Staff (Who uses the plagiarism check software)**
Name & Signature of the Supervisor/Guide	**Name & Signature of the Co-Supervisor/Co-Guide**

Name & Signature of the HOD

Sidd_Pank

ORIGINALITY REPORT

4%	1%	2%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to University of Warwick Student Paper	1%
2	Submitted to Universiti Sains Malaysia Student Paper	1%
3	"Advanced Computing Technologies and Applications", Springer Science and Business Media LLC, 2020 Publication	<1%
4	Matignon. "Model Nodes", Data Mining Using SAS® Enterprise Miner™, 06/06/2007 Publication	<1%
5	Submitted to Universiti Teknologi Malaysia Student Paper	<1%
6	Submitted to University of Ulster Student Paper	<1%
7	Christopher J. Hoekstra, David A. Deppeler, Richard A. Rutt. "Criterion validity, reliability and clinical responsiveness of the CareConnections Functional Index", Physiotherapy Theory and	<1%

Practice, 2014
Publication

8	creativecommons.org Internet Source	<1%
9	Zhan Shi. "Improving k-Nearest Neighbors Algorithm for Imbalanced Data Classification", IOP Conference Series: Materials Science and Engineering, 2020 Publication	<1%
10	global.yamaha-motor.com Internet Source	<1%
11	estudogeral.sib.uc.pt Internet Source	<1%
12	www.utupub.fi Internet Source	<1%

Exclude quotes	On	Exclude matches	< 10 words
Exclude bibliography	On		

PAPER PUBLICATION

Priya.S, Pankaj Thakur, Siddharth Agarwal, Annie Uthra, "Ensemble Based Classification for Class Imbalanced Credit Card Fraudulent Data", International Journal of Advanced Science and Technology, Vol. 29(6),pp. 2129-2141, 2020.

International Journal of Advanced Science and Technology
Vol. 29, No. 6, (2020), pp. 2129 - 2141

Ensemble Based Classification for Class Imbalanced Credit Card Fraudulent Data

S.Priya*, Siddharth Agarwal, Pankaj Thakur, Annie Uthra

Department of Computer Science and Engineering,
SRM Institute of Science and Technology, Kattankulathur
**priyas3@srmist.edu.in,annieu@gmail.com,pankajpream524@gmail.com,*
sid7549@hotmail.com

Abstract

Knowledge extraction from imbalanced data has been receiving increasing interest in recent years. Most of the real world problems including credit card transaction frauds, disease prediction and so on have huge data instances but the number of positive instances in those datasets is far lesser than the number of negative instances. Suppose 99% of the data tuples come from the majority class but only 1% of the instances are from the minority class, and our classifier classifies each data tuple as majority instance, then the accuracy of our model will of course be 99% but this classifier is of no use as it doesn't detect any minority instance which is the main purpose of the classifier. This is why the data needs to be balanced first in order to train our classifiers. The balanced data can now be used to train our classification model. There are so many classification algorithms present, and each of them has their own pros and cons. So, in order to achieve higher accuracy and better results various individual classification algorithms including Naive Bayes, Decision Trees and Random Forests are used. After getting the results of the above mentioned classifiers, ensemble models have been applied that includes Voting Classifier, XGBoost and ADABoost. It is seen that the efficiency of ensemble classifiers in data classification is much higher than that of individual algorithms.

Keywords: *Class Imbalance, Ensemble Classifier, Sampling, Machine Learning.*