University of Essex

Department of Mathematical Sciences

MA981: DISSERTATION

Credit Card Fraud detection using machine Learning

ARSHAD ALI

Supervisor: **Tao Gao**

June 22, 2024

Colchester

Contents

List of Figures

List of Tables

Abstract

With the global growth of credit card transactions, cases of fraud have also risen. Detecting fraudulent transactions is crucial to limiting losses and maintaining trust in the credit system. This thesis explores supervised machine learning approaches for improved identification of fraudulent credit card transactions.

Credit card transactional data presents challenges such as class imbalance with far more genuine transactions than fraud, requiring sampling techniques. Features also have complex relationships and dependencies. This study compares five classifiers on the credit card fraud dataset that includes random forest, KNN, regression based logistic function, decision tree, and support vector machine.

Efficiency is measured using AUC-ROC, recall, precision, and F_1 score measures. K Nearest Neighbour and Random Forest deliver the best performance, with 99.73% and 99.72% respectively. AUC-ROC on the balanced dataset showed that SVM has a slightly higher TPR. Support Vector machine classification is the most consistent performer across metrics and sampling methods.

The study provides a methodology for applying machine learning to imbalanced credit card transaction data. The results demonstrate superior fraud coverage through enhanced predictive accuracy. Recommendations are made for the production deployment of the models to cut fraud losses and reduce account abuse. Future research ought to look on grouping classifiers.

CHAPTER 1

Introduction

1.1 Background

Fraud evolves with the passage of time as humanity evolves. It changes its form with the need and technology. Banking products like credit and debit cards are no exception. fraud also found ways to steal the amounts from banking accounts using various money steeling techniques. During the COVID pandemic, the use of online transactions escalated worldwide, and criminals took advantage of the situation and discovered new ways. Fraud through online transactions using credit cards is an alarming issue for individuals and financial companies, costing billions of dollars each year [1]. Rule-based Fraud detection systems cannot cope with temporal fraud techniques and can be easily bypassed by sophisticated criminals. As a result, massive amounts can be transferred in a short period of time with no sign of risk and without the owner's approval. Every fraudulent transaction can be legitimized by a fraudster's operation, making fraud extremely difficult to detect [2]. Hence, we are sufficiently motivated to train a pass-through machine learning classification technique to detect credit card fraud. On the other hand, Machine learning techniques pose a different tactic to fraud detection. They can learn from historical data by recognizing patterns to identify fraudulent transactions.

1.2 Aim and Scope

The purpose of the study is to build and train a model using machine learning approach that can precisely spot credit card fraud based on historical credit card transactions. Subsequently, the same will be evaluated on holdout transactions, and its performance will be measured using accuracy, precision, and recall.

1.3 Methodology Background

To begin, we must comprehend why modern detection systems as these detection systems to prevent fraud are so complicated. For most owners and clients, the contemporary systems for fraud detection is known as anti-fraud programs. However, the fraud related to credit cards has not yet been defined in a clear manner or we can say the goal of fraud is ambiguous. Anti-fraud appears to be a binary problem on a small scale. However, the study discovered that it is a multi-classification problem since each sort of fraud can be dealt with differently [3]. Furthermore, a single type of fraud does not exist, and the tools of fraud are always advancing from time to time. Most banks, insurance companies, and our clients are victims of these emerging fraud tools and techniques. They must always show eagerness to enhance their system and programs to predict fraud. Rather than relying on the same old methods. This is the main challenge that fraud detection is currently facing.

1.3.1 Credit card fraud detection

To ensure their customers are not charged for unauthorized purchases, credit card companies must differentiate between fraudulent and non-fraudulent transactions. This helps maintain the integrity of their customers' accounts [4]. It is intended to prevent fraudulent transactions made via credit card and to recover losses for businesses and customers as well as their credibility. Despite the fact that there are stronger financial processes, the fraudster is evolving his approaches constantly. It also creates difficulty for anti-credit card fraud approaches; the typical anti-credit card fraud methods currently available on the market.

The validation method through merchant trade is one of these where merchants require

a list of receipts in order to identify the user, and tokenisation techniques have been used to safeguard the information of credit cards by using the card number that is referenced then the number the card is currently in use. It can ensure that the user must provide extra information such as a card security code, zip code, or pin code. Users may also be asked to display cards in order to complete the transaction process securely, and merchants are using them to combat fraud in current situations [5].

Geolocation technology also presents an accurate geographic position via the computing device's IP address from where the order is made, which can indicate regions with a maximum risk for fraud. In order to protect themselves from credit card fraud many sellers are now enabling authentication with diverse applications for transactions that look like to be problematic cases. [6].

The detection method ensures that the country of the ordering device confirms the country of the address mentioned in the billing information. The program can detect the IP address and country of the customer placing the order by employing a fraud prevention service. In case the customer's delivery and billing addresses are not from that country [7], where the person is placing the transaction, a more detailed inspection is required, and at the same time anti-fraud prevention system must be activated immediately.

Many reputable consumers use free email accounts as they are easy and cost-effective. To remain anonymous, most fraudsters utilise free email addresses. One of the most essential approaches to improve fraud detection is to identify the registration of emails from various new domains [8]. Secondly, anonymous devices act as proxy servers to enable internet users to conceal their true IP addresses. Because the major goal of surfing through these servers is to prevent discovery or stay anonymous, people should maintain a database of various servers that are providing proxies in order to prevent credit card fraud by using a web service.

1.4 Dissertation Outline

The organization of the thesis is described in the preceding lines. Chapter 2 contains a related literature work while chapter 3 explains related methodologies used. Chapter 4

discusses the experimental results and provides the conclusion.

CHAPTER 2

Literature Review

Md Shufyan and Prashant Prashun [9] discussed the problem of the class imbalance issue of the credit card dataset used for fraud detection and suggested an approach based on optimized machine learning algorithms to solve the class imbalance problem. These machine-learning methods were trained on a dataset of fraudulent and non-fraudulent transactions and then tested on a separate dataset of transactions. The trained algorithm was able to score a 99.7% accuracy while correctly predicting the fraudulent transactions.

Chole et al. presented an approach based on machine learning algorithms for fraud detection of credit cards [10]. The algorithm uses a synthesis of statistical and decision tree classifiers to assess each transaction individually. The results showed that the algorithm was able to achieve a considerable accuracy of 91.3% in identifying fraudulent transactions.

Khan et al. [11] proposed a model based machine learning algorithm for detecting fraud in credit cards. The approach uses a combination of support vector machines, random forests, and decision trees, to assess each transaction individually. The results displayed that the model scored an accuracy of 94.2% in identifying fraudulent transactions.

Tanouz et al. [12] presented a model based on a machine-learning approach for credit card fraud identification. The model uses a blend of light GBM, Adaboost, and random forest classifier to assess each transaction individually. A remarkable accuracy of 95.8%

was achieved by the model for credit card fraud identification.

Mallidi and Zagabathuni [13] analyzed the results of different machine-learning models for detecting fraud related to credit card on both balanced and imbalanced datasets. The models they considered were decision trees, naive Bayes, support vector machines, logistic regression, and random forests. The results showed that a varied performance by the models depending on the training and testing sample. On the balanced sample, all of the models performed well, with accuracies ranging from 90% to 95%. However, on the imbalanced dataset, the performance of the models was more varied. The best-performing model was random forests, with an accuracy of 94%.

Gupta and his fellows focused on presenting an automated approach that employs multiple ML models to identify fraudulent occurrences that are tied to clients economically but more specifically in credit card-related transactions. Out of all the techniques used, Nave Bayes performed the best in identifying fraud-based transactions, with a score of 80.4% accuracy and a score of 96.3% under the curve [14].

On an imbalanced dataset of credit card transactions, Ganga and Safa [15] studied the performance of K-nearest neighbour (KNN), Logistic Regression, and Naive Bayes models that achieved accuracy of 97.69%, 54.86%, and 83% trained models respectively.

Kiran and his team [16] offer a new approach named NBKNN for Fraud Detection of Credit Cards using Gaussian Naive Bayes (NB) enhanced (KNN) K-Nearest Neighbour. The experiment's results show the differences in the processes of each classifier on the same dataset this approach outperformed KNN, with an accuracy of 95% versus 90%.

Benchaji and Ouahidi [17] proposed a fraud detection model for credit cards that used Long Short-Term Memory (LSTM) and recurrent neural networks. The model uses the sequential information between transactions to improve the detection accuracy. The results indicate that the model scored 94.5% accuracy in detecting fraudulent transactions.

Varmedja et al. [18] presented an approach that used machine learning for credit card fraud identification. The approach uses a combination of decision trees, random forests, and support vector machines to assess each transaction individually. The results indicate that the approach scored 93.4% accuracy in detecting fraudulent transactions.

The method used by Najdat et al. to identify fraud-based transactions is (BiLSTM) [19]. BiLSTM MaxPooling is bidirectional Long short-term memory and BiGRU MaxPooling (BiGRU) is a bidirectional Gated recurrent unit. Furthermore, the committee chose six ML classification algorithms: Adaboost, Voting, Decision Tree, Random Forest, Logistic Regression, and Nave Bayes. KKNN scored 99.13% of accuracy, logistic regression had a score of 96.27%, decision tree had a score of 96.40%, and Naive bayes had a score of 96.98%.

In this study by Adepoju and his team [20], Logistic Regression, (SVM) Support Vector Machine, Naive Bayes, and (KNN) K-Nearest Neighbour were used on distorted credit card fraud data. The accuracy of all models was 99.07% for Logistic Regression, 95.98% for Nave Bayes, 96.91% for K-nearest neighbour, and 97.53% for the last model (SVM) Support Vector Machine.

They used a dataset from the European trading market with 284807 deals in the study [21]. They employed a hybrid technique of under-sampling and oversampling, coded in Python, and trained with three classifiers. The accuracy of KNN and logistic regression were 97.69% and 54.86%, respectively. According to the findings of his experimental study, KNN outperforms all other connecting techniques. It can serve as a reference, explain why the logistic regression is so low, and suggest how to improve the KNN accuracy.

Maniraj et al. [4] presented Fraud Detection using machine learning in order to demonstrate the modeling of a data set on credit cards. They attempt to detect 100% fraud-based transactions while minimising inaccurate fraud classification. A PCA-based transformation was conducted on a dataset of Credit Card Transaction, they emphasised more on preparing and analysing the datasets as well as employing multiple error detection methods like algorithm of the Local Factor Isolation Forest. The results indicate that the algorithm has a precision of roughly 28% when the tenth portion of the data set is predicted. however, the precision rises to 33% when the whole dataset is posed to the system.

Mohari et al. [22] identified and performed a comparison of Logistic Regression, AdaBoost, Genetic Algorithm, Random Forest, Artificial Neural Network, Decision Tree, Isolation Forest, Hidden Markov Model (HMM), KNN Classifier, and Local Outlier

Factor. The result indicates that Local Outlier Factor fraud scored higher accuracy in comparison to all other algorithms.

Two main problems identified by Ayorinde et al. [23] on Fraud detection of credit cards. The first one is that the properties of fraudulent and conventional transactions are varying with time, and databases for credit card fraud are extremely imbalanced. On highly non-uniformed data of credit card fraud, they analyse the model and data performance further using k-nearest Neighbour, naive Bayes, and logistic regression. They used three approaches for both preprocessed and raw data. The sensitivity, specificity, precision, accuracy, flat classification rate, and Matthew's correlation coefficient are used to examine the result of the approaches. The results indicates the accuracy of 97.69%, 97.92%,and 54.86% by the k-nearest neighbour, naive Bayes, and logistic regression classifiers respectively.

KNN Classifier, Hidden Markov Model (HMM), Isolation Forest, Decision Tree, and Local Outlier Factor are a few of the algorithms used. The result of the ten techniques indicate that, Local Outlier Factor out smart the other algorithms in terms of accuracy. A system for detecting fraud in credit card was developed by Braun et al. [24] using semi-supervised graph-based methodology. It has been documented that fraudulent operations have cost the world billions of US dollars. Automated Fraud Detection Systems (FDS) can initially reject a transaction before it is approved in order to halt these heinous actions. They built their FDS on a graph called APATE, which spreads malicious influence over a network using a few valid fraudulent transactions. They afterwards finetuned APATE to make it ideal for the e-commerce sector. Since these improvements multiply accuracy at 100 by three for both transaction prediction and fraudulent credit card detection, they have a significant impact on achievement. Real-world testing of this unique approach was carried out for three months utilising data from e-commerce credit card transactions obtained from a major credit card provider. Although this case also offered feedback, the impact can still be increased by looking at more cards, thus there isn't much of an improvement.

A number of scholars have examined the detection of credit card fraud using the XGBoost model that include Meng et al. [25] and Parmar et al. [26]. According to Meng et al. [25], the XGBoost system based on CART, effectively applies the Gradient Boosting

and GB technique. Meng and his colleague used accurate online transaction data from an online financial institution to do research on credit card fraud detection procedures. Using SMOTE, undersampling, and XGBoost, they investigated the performance learning method on the original data set. According to the study, the combination of both SMOTE and XGBoost produced excellent results.

In a related study, Parmar et al. [26] examine a variety of methods to identify credit card fraud, including K-Nearest Neighbour, Support Vector Machine (SVM), Decision Trees, Logistic Regression, Random Forest, and XGBoost. We tested a sample of 2,84,808 credit card transactions from financial institutions in the EU. Despite of non uniformity of the dataset, it contains 17.2% of fraud instances from actual transactions. The methods are implemented in Python, and the results of the methods is categorised using the clarity, F1 rating, and confusion matrix. The findings show that any set of rules may be used to detect credit card fraud with high precision.

Shirgave et al. [27] covered machine learning-based credit card fraud detection in their review article. They use criteria like accuracy, precision, and specificity to evaluate different machine learning fraud detection techniques. They also propose an FDS based on the supervised Random Forest approach. Their proposed technique increases the accuracy of credit card fraud detection. Additionally, by ranking the alert and utilising the learning to rank methodology, the proposed solution effectively solves the problem of concept drift in fraud recognition.The prevalence of fraudulent activities needs to be taken into consideration by both financial institutions and people. To solve this problem and differentiate between valid and fraudulent transactions, Priya et al. [28] find a reliable fraud detection system where valid transactions exceed fraudulent ones.

The Random Forest fraud detection system was utilised by More et al. [29]. This approach has steadily improved the ability to identify fraud in credit card transactions and can assist in resolving fraudulent activity in the real world. According to the findings, fraud occurs in 26.2% of whole dataset. The non-uniform dataset was processed and then 80% of it was used to train the model and 20% for testing. The performance was assessed for accuracy, recall, and precision. The suggested technique had shown enhanced accuracy for a large quantity of training data, as indicated by the accuracy score of 97%.

The Hidden Markov Model Algorithm is used by Rahmawati et al. [30] to analyse the event logs of a bank's credit business process in order to detect fraud. As was previously discussed in this book, several fraudulent operations are carried out every day using a variety of strategies. As a result, they offer Hidden Markov Models to detect fraudulent behaviour by using activity log data. The automated system uses the event log to assess the likelihood and potential for fraud by spotting signals of fraudulent behaviour. The analysis involved 90 cases, and the results show that the HMM technique can be used to spot fraud because it has a 94 percent accuracy rate. Of the 90 transactions, the model indicated that 10 would be fraudulent and 80 would be real.

Rocha and de Sousa Junior [31] used the CRISP-DM algorithm as well as Decision Tree algorithm to identify bank frauds. In order to point out and stop bank related fraud, their team analyse particular transactions via CRISP-DM and decision trees. Their team concur that decision trees are a key concept in machine intelligence, as did bunch of scholars who worked before them. After examining data on bank activities, the inquiry found various fraudulent practises in transactions based on online banking .

To point out fraud while utilising credit card, Jisha and Vimal [32] employed an improved and condensed fuzzy deep belief network based on population. The study employs a technique of fuzzy theory to find the valuable aspects that significantly influence the identification process as opposed to the conventional theory employed for an vigilant strategy of fraudulent transaction identification. The deep fuzzy network's extensive knowledge base and layered limited Boltzmann machine enable it to handle credit card transactions' complicated nature with ease.

CHAPTER 3

Mathematical Methodologies

Python is the Language that has been utilised in the study for data analysis and model training. Google Colab is the environment that has been used with 3.10.6 as the Python language version.

3.1 Data Preprocessing

This section has been started by loading data from the CSV file available at https://www.kaggle.com/c ulb/creditcardfraud. The file was loaded into to a data frame using the csv file reading function of pandas. The loaded data frame contains 23769 transactions out of which only 88 fraud transactions. It contains 31 columns out of which 28 columns are scaled and 3 three are not scaled (Time, Amount, and Class). The data frame is then cleaned from null values. After that, the mean, median, minimum, and maximum values of 30 variables except the class variable are derived. Mean and median Values of each variable for fraudulent transactions are displayed in Figures 3.1, and 3.2. Similarly, Minimum and maximum Values of each variable for fraudulent transactions are displayed in Figures3.3, and 3.4 respectively. Moreover, the Figure 3.5 displays the standard deviation for each numeric variable based on transactions marked as fraudulent. These measures are very supportive to tell how the data spread in the feature. The mean and median are two measures are used to find the middle of the data spread.

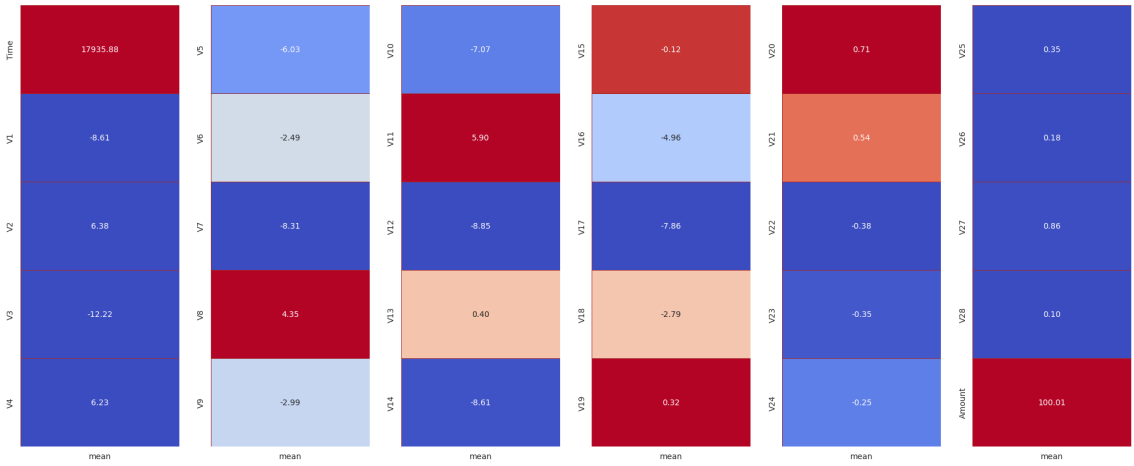

Figure 3.1: Mean Values of all numeric features of the dataset.

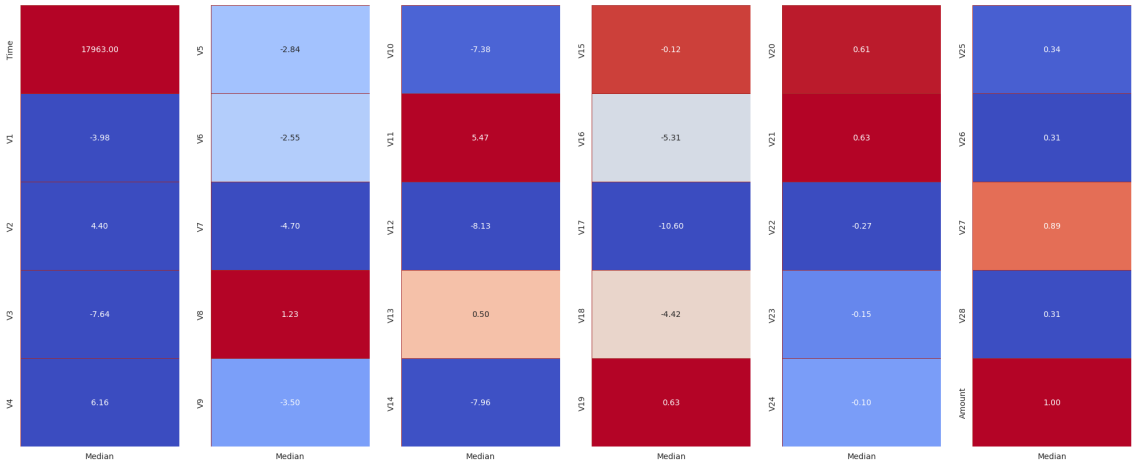

Figure 3.2: Median Values of all numeric features of the dataset.

3.1.1 Scaling and Distributing

Both amount and time are numeric variable but they are not scaled in the data like other features. So, the scaling of these variable is done. Now all of our features are now scaled except the target variable class. As we observed that our dataset is not balanced as 99% of the transactions belongs to clean case. The imbalnced data is So, we require to develop a balanced dataset sample that contains equal number of transaction for both cases represented by the target variable class.This indeed supported our approach understand the patterns in better way.Ultimately, these patterns help to identify fraudulent transactions from the clean ones.

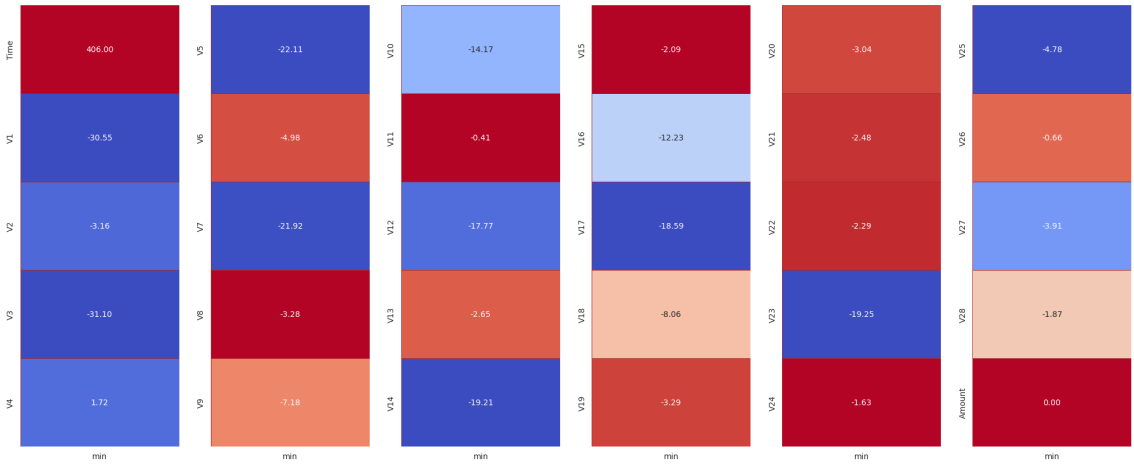

Figure 3.3: Minimum Values of all numeric features of the dataset.

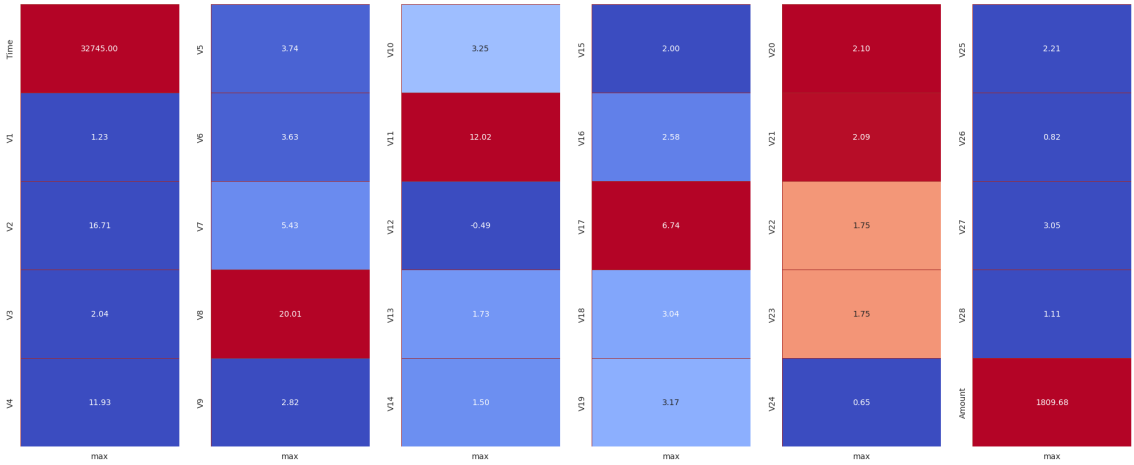

Figure 3.4: Maximum Values of all numeric features of the dataset.

3.1.2 Non-uniform Data

The data set is highly imbalanced with 99.69 % clean transactions represented by class 0 and 0.31 % fraud transactions represented by class 1 as shown in figure 3.6. If we utilise this data frame as the base for our classification approaches and exploratory data analysis, we may encounter over fitting and wrong correlation. In case of over fitting, Models will presume that there are no frauds in the majority of cases When a fraud happens. In our case, we want to avoid over fitting.

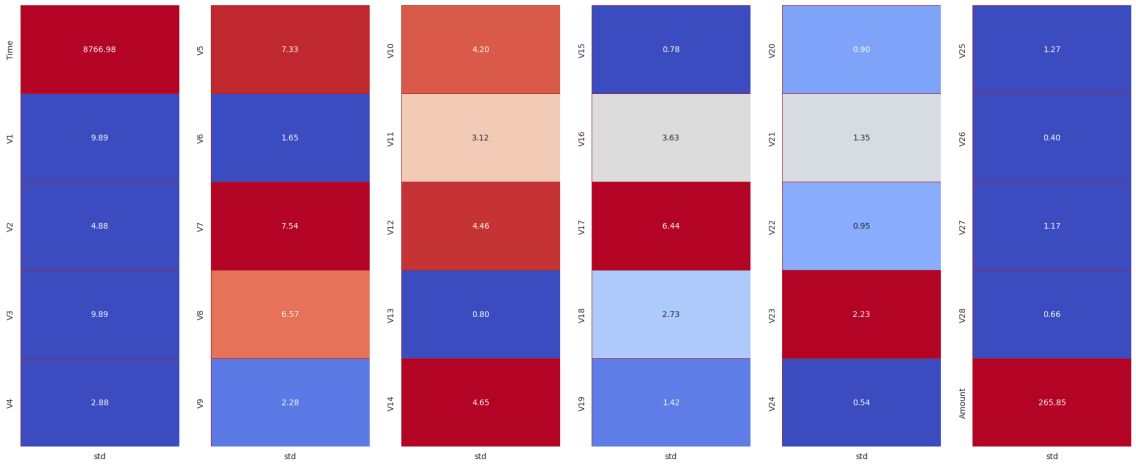

Figure 3.5: Standard deviation for each numeric features of the dataset.

Figure 3.6: Data distribution of target variable among two classes before sampling. Class 1 is for fraud transactions while class 0 is for non-fraud transactions.

So we have to develop a sample of that consists of same number of clean cases as of fraud cases, which will assist our classifier to grasps the sequences that guide us to identify the fraud cases. In order to get the best results the ratio between both transactions will be 50/50. There are two ways to solve the issue either we do under-sampling or over-sampling.

3.1.3 Under Sampling

The process of under sampling consists of selecting cases to exclude from class that have a high percentage of transactions to build a new sample [33]. So, we have to create a new dataset and delete clean transactions to balance data in such a way that both fraud data transactions and clean data transactions are equal in numbers in the newly created dataset. The question arises as to why we are creating a new dataset because we want the original dataset to be used for testing the model. Now, the ratio between both transactions is 50/50 in the new dataset as shown in figure 3.7

Figure 3.7: Data distribution of target variable among two classes after sampling. Class 1 is for fraud transactions while class 0 is for non-fraud transactions.

3.1.4 Over Sampling

It is the process that works on the phenomenon of add similar cases like exist in class that have less percentage of transaction in order to form a new sample that have equal no of cases [33]. This is solution other than the under sampling to the imbalanced dataset issue, it uses a Synthetic Minority Over-sampling Technique called SMOTE [34]. This technique generates fresh artificial points to make sure that the classes are equal. It calculates the distance among nearest neighbors of minor class and create artificial points in the calculated distance space. In this case, we do not delete any information, unlike under-sampling. So, it requires more time for training as the dataset increases due generation of new sample rows.

3.2 Data Analysis

In this section, we are going to find the relationships among both predictors and response variables.

3.2.1 Correlation

Correlation heat-maps are vital for comprehending our data. In our study, we want to find out are there any significant properties that have a meaning full impact while predicting a particular transaction is clean or not. Moreover, the correct sub-sample must be used to determine which attributes have a high positive or negative connection with fraudulent transactions. In our case, the data was imbalanced initially and there is no significant relationship between predictors and target class as shown in the first graph of Figure 3.8. On the other hand, the second graph shows the relationship among variables which was generated on sub-sampled data. It also shows that V21, V4, V11, and V2 are directly correlated with the target variable such that the higher values contribute more to detecting fraud transactions. In contrast, V12, V14, V10, and V3 are inversely correlated with the target variable such that the lower values contribute more to detecting fraud transactions.

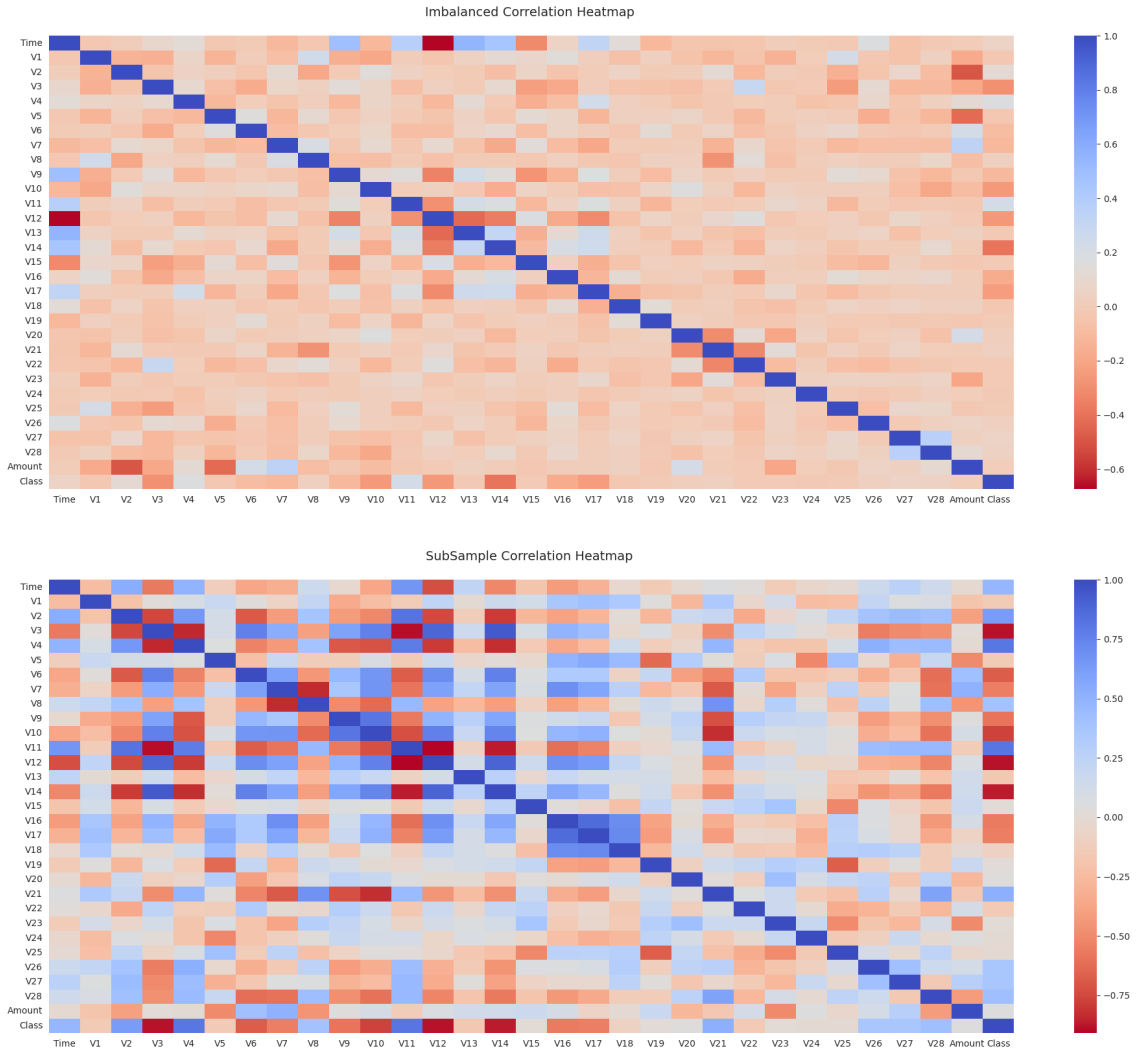

Figure 3.8: Correlation Matrices showing the relationship of each feature with others. The top panel is for the correlation heat map of imbalanced data. The bottom panel is for the correlation heat map of the sub-sample.

A density plot is another way to see data distribution over fraud and no fraud cases. It also confirms that the variables (V2, V4, V3, V14, V12, V10, V11, and V21) highlighted by the correlation matrix distinguish the class variable as shown in Figure 3.9. It also shows the distribution of data with respect to each variable based on clean and fraudulent transactions. The Blue curves are for clean transactions while Orange curves are for fraud transactions. It also shows that there are some features in which the pattern of both fraud and non fraud classes are the same just look at V24, V22, and V8 they are not the good class separators. On the other side, as we look at V12, V10, V11, and V14 they are clearly segregating both fraud and non fraud classes. So these feature are more

likely to distinguish classes from target.

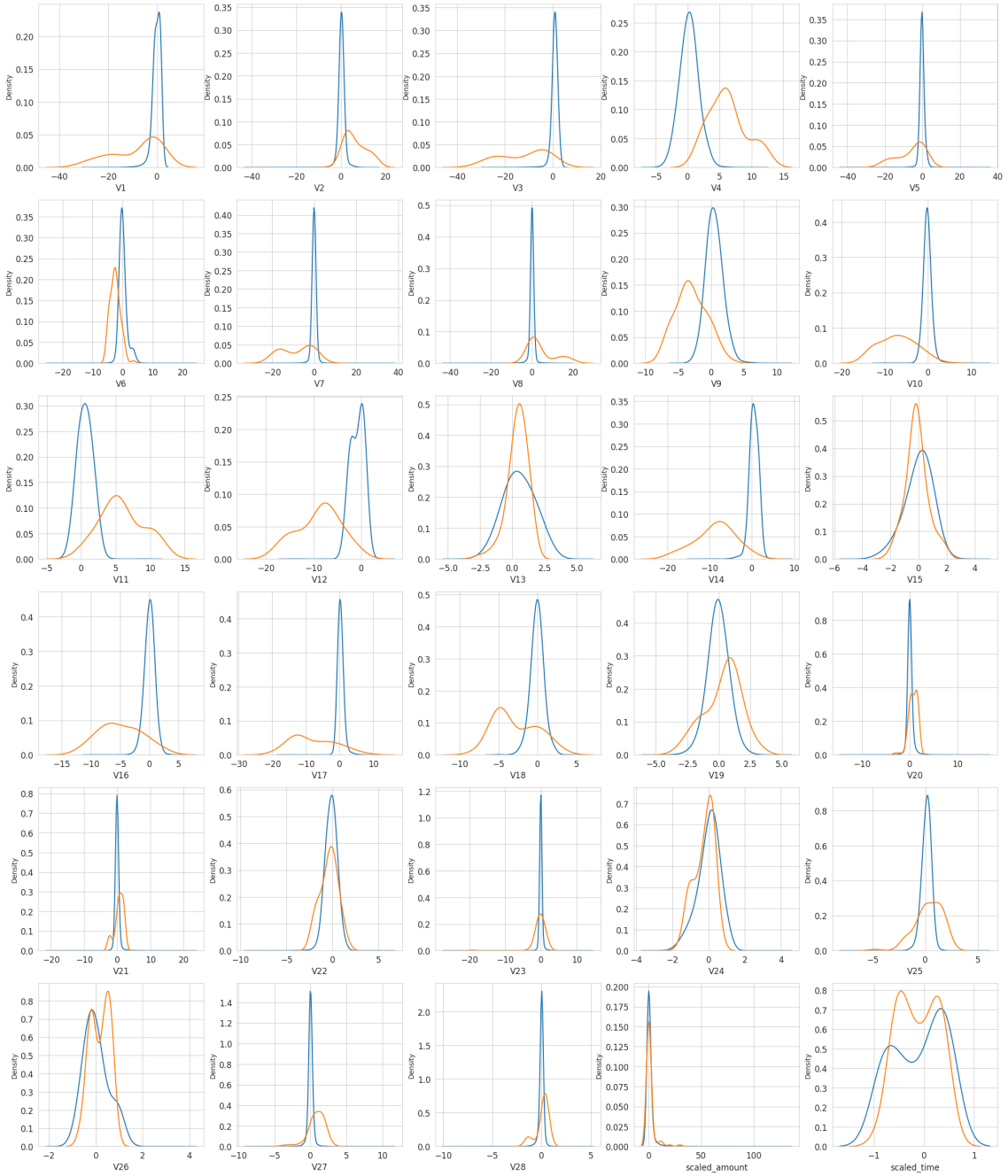

Figure 3.9: Density Plot for all features showing fraud and clean transaction. The Blue curves are for clean transactions while Orange curves are for fraud transactions.

3.3 Performance metrics

3.3.1 Confusion matrix

The evaluation metric that utilized most often in analysis for estimation and is also the simplest to learn. Confusion matrix utilised to calculate accuracy also the most often used metric along with recall. similarly, precision ca n also be calculated by using confusion matrix. It consists of values that are actually true and also predicted true known as true positive often represented as TP.Also, there are true negative that are actually false and also predicted false often abbreviated as TN. However, false positive are actually true and also predicted false often described as FP, and false negative are actually false and also predicted true often marked as FN. All of above discussed evaluation metrics can be calculated by using these four values of confusion matrix. The accuracy can be determined from confusion matrix as presented in equation (3.1)

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{3.1}$$

True Positive: present a case when actual and estimated values are true.

Type I Error: discuss the scenario when the estimated value is true on the contrary the actual value is false.

True Negative: highlight the case when both the values are false.

Type II error occur when false is the value desired and you got true.

3.3.2 Recall

It measures the percentage of real positives. Recall is the proportion of all true and positives that were discovered and recognized. The true positive cases as described in equation 3.2

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FNs}}. \tag{3.2}$$

We consider a football play ground with 22 players with blue and orange shorts, for instance. For example, a model recognises 8 player with blue shorts in the image, but only 5 players of those were genuinely in blue shorts such that TP, while the other 4 were in orange such that FP.So, the value of recall is 5/22.

3.3.3 Precision

This is the percentage of accurately classified positive cases to all positive instances, including both true positives and false positives as depicted in equation (3.3). Simply, precision is the percentage of cases that were discovered to be true positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{3.3}$$

The precision value in the aforementioned player's short color identification example is 5/8.

3.3.4 F_1 Score

The F_1 Score, some times pronounced as the F-measure or the F score, is the The harmonic average of recall and precision. The value of F_1 Score span between 0 and 1, from worst to best, and is computed in equation 3.4.

$$F_1 = \frac{2\,(\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}\,. \tag{3.4}$$

3.3.5 ROC curve

Among, the commonly utilized evaluation measures in estimation analysis describe us about the model, how well it works for various likelihood thresholds. In case of classification task, a likelihood threshold of 0.5 is utilised by default. A representation based on True Positive Rate abbreviated as TPR and some times called as sensitivity as mentioned in equation (3.5).The False Positive Rate denoted as FPR and the formula for FPR or specificity is shown in equation (3.6).

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{3.5}$$

$$\text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FPs}}. \tag{3.6}$$

Sensitivity and specificity have a negative relationship as the probability threshold is decreased, sensitivity grows while specificity drops and vice versa. The likelihood that a model will mark a spontaneously chosen positive instance more substantial than a spontaneously chosen negative instance can be measured as the area covered by the ROC curve.

3.3.6 Area Under Precision-Recall Curve

It is among the essential evaluation metrics for gauging the effectiveness of a model that makes prediction, particularly when the class distribution is imbalanced. The plot of PR curve represent precision scores and recall scores at different thresholds based on probability. PR curve covering the area may be utilized to gauge the efficiency of the model. You may evaluate the effectiveness of the depression filter using AUC-PR. We would first calculate the precision and recall of the depression filter at various thresholds. When determining if a person is depressed or not, a threshold is applied. For instance, we might decide on a 0.5 threshold, which designates a person as depressed if its likelihood of being depression surpasses 0.5. Then, for various levels, we would display precision versus recall.

The AUC-PR represents the region under this curve. A depression filter with a higher AUC-PR is likely to be more successful. The depression filter's AUC-PR in this instance is 0.9. This demonstrates how proficiently the depression filter can discriminate between depressed person and normal person.

Computing Methods

The area under the curve is roughly represented by a series of trapezoids using the trapezoidal rule. The total area under the curve is derived by calculating and summing the areas of each trapezium. The formula used compute is described in equation (3.7).

$$AUC_{\text{PR}} = \sum_{i=1}^{n-1} \frac{R_{i+1} + R_i}{2} \cdot (P_{i+1} - P_i). \tag{3.7}$$

This approach approximates the area under the curve more precisely than the trapezoidal rule, according to Simpson's rule. It separates the curve into Simpson's rule segments. The total area under the curve is computed by calculating and summing the

area of each Simpson's rule segment and equation denoted as (3.8).

$$AUC_{PR} \approx \frac{1}{n} \sum_{i=1}^{n/2} \left[P_{R_i} + 4P_{\frac{R_{i+1}+R_i}{2}} + P_{R_{i+1}} \right] (R_{i+1} - R_i). \tag{3.8}$$

Monte Carlo integration: This method estimates the area under the curve using random sampling. Random sampling is used to select many points from the curve, and the area under the curve is calculated using equation (3.9) by counting the number of points that fall above and below the curve.

$$AUC_{PR} \approx \frac{1}{m} \sum_{i=1}^{m} \frac{P(r_i)}{N(r_i)}. \tag{3.9}$$

The trapezoidal rule is the most straightforward way for calculating the AUC-PR, but it is also the least accurate. The Simpson's rule is more accurate than the trapezoidal rule, but it also takes longer to compute. The most precise method is the Monte Carlo integration method, but it is also the most computationally expensive.

The method utilised to compute the AUC-PR is determined by the application. If speed is critical, the trapezoidal rule may be an excellent option. If precision is critical, Simpson's rule or the Monte Carlo integration method may be preferable.

When evaluating models with different costs for both false positives and negatives, these statistics are useful. It is unaffected by skewed datasets. It is simple to understand. It is susceptible to outliers. Comparing AUC-PR results for models trained on various datasets might be problematic.

3.4 ML Classifiers

We used five classification approaches decision tree, regression based on logistic function, KNN, support vector machines and random forest to mark fraudulent transactions using credit card dataset. It is also widely used in categorization learning. Hence, comparison of the result obtained from these classifiers may reveal fascinating outcomes. In addition, the final results may include a detailed comparison with contemporary works in the research area.

3.4.1 Logistic regression

The algorithm was chosen for the idea where simplicity is preferred over complexity and a widely recognised approach based on statistics for determining the result of a binomial or polynomial. In order to categorise emails,spam detection, and tumours the model can be used. It calculates the likelihood of a given outcome, that can be binary or polynomial. Sigmoid is the main method that is utilised to specify data and their variables in such way that it tell how they are related to each other. Specially, the relation among predictors and target variables. In this study, to establish whether or not a transaction is fraudulent logistic regression may play a vital role. It is extremely effective despite the fact that of its over fitting for those data sets which have high dimensions . Unlike some other algorithms, it gives more precise and accurate results without making any assumptions about class spread in the area of features. The premise of linearity among target and predictors is a disadvantage.

When the model's outcome comes into one of the a plethora of selected beforehand classes, the categorization issue is interested with separating a variety of input factors and deciding its belonging class. It also play a vital role in resolving the binary classification issue. In which the predicate likelihood of binary speculation of the target is determined to agree a straight-line combo that works together of various input variables adjusted by the sigmoid method. It is often referred to as a logical model. The answer variable, which may be 0 or 1, should fall into one of the two possible classes, which the model should be able to predict. The traditional and ideal bicategorical algorithm for categorical problem resolution is logistic regression.

3.4.2 Decision Tree

A decision tree where each node stands in for a component called attribute, every link named as branch based on some decision known as rule, and every leaf for an outcome may be a categorical or a continuous value. The entire construction of the decision tree works on giving the data a tree-like shape. And the decision tree on each plate deals with individual outcomes. It is simple to assess the facts we provide and produce some wonderful insights because decision trees are made to mimic human horizontal thinking. Decision trees offer the reasoning behind how data is comprehended.

The major goal of decision tree lies in to create a training approach to forecast the target variable's classes. This methodology is techniques used to forecast transaction classification among various others. It is a collection of edges that connect a number of branches and nodes. An interior node of a tree performs an evaluation, and the result is displayed by the edges. The edge nodes reflect a label of category. The focus is to divide the input dataset into classes iterative manner utilizing the method known as Depth-First Breadth unless every value in a sample is assigned to a specific category.

The main advantages of this techniques is that it ignores feature scaling. It is more robust to outliers issue, and is good at managing information that is missing without any intervention. It is quite adept at tackling issues associated with regression and classification, and it takes less time during the training stage. One important challenge is the growing dataset size when it increases, the single tree complexity increases and it becomes more difficult to understand. This complexity of the tree some times result in example of an over fitting scenario.

A decision tree is a model for prediction that shows how an object's properties and values relate to one another. It is commonly utilized approach work on the basis of attributes, branches and decisions in order to create predictions for various classification problem mentioned by [35]. We selected it for the fraud detection system training for this reason.

3.4.3 Random Forest

Since ensemble learning is the fundamental technique of random forests, classification and regression issues are better suited to them. The efficiency of the approach as a group is improved by using ensemble forest, which creates numerous decision trees and mixes them [36]. When producing a forecast, preference is given to the class with the most votes. In order to produce a group of decision trees that would after all combine to form a forest, the approach uses a bagging strategy. This method has the advantage of not requiring feature selection, in order to enhance model execution, and skillfully managing the mistakes.

Its sensitivity to data that have a wide range of values is downside of the approach. It looks ordinary while identifying fraudulent transactions if data values have more

spread. Decision tree ensemble known as "forest" that this algorithm generates, and its training works on the basis of bagging technique that also integrated with bootstrap technique and create an highly efficient machine learning algorithm.

Bagging

The grouping approach called as bagging or in other terms we ca say it Bootstrap Aggregation. The technique is basic, easy to learn at the same time very impressive. The technique create new training samples called bootstrap by replacing data from original training set. These bootstrap are called training samples. Each bootstrap sample presented to various models for training intensively prior to the model utilized for actual predictions. The overall outcome calculation is basically depends on the issue type. In classification issue a majority vote is decisive while in case of regression average is used to produce the final result. Decision trees act as the basic role for bagging approach.

Boosting

Another ensemble method shown remarkable result is known as boosting. While producing a powerful learner that can perform so well and generate such outclass results to those produced by an individual learner. It work on grouping of base learners also referred to as weak learners. Boosting entails training the weak learners consecutively in an effort to have each learner try to correct the preceding one by giving the incorrectly classified samples more weight. This contrasts with bagging, which executes each approach simultaneously prior to merging the outputs. As a outcome, the weak predictor require more training on more labelled data to produce more accurate result rather that wrong ones.

Bagging along with random forests are two approaches that combine various techniques to produce a groups. Both algorithms displayed admirable result in a variety of forecasting problems. This is among the finest approaches for detecting banking frauds. The benefit of using ensemble approach is that problems with classification and regression can both be resolved. It is more important to select the best feature among all characteristics for modelling since the Random technique constantly adds randomness as it begins to create the tree. This is especially important during the splitting of the node.

The model's ability to predict outcomes was enhanced or sped up by the random forest hyper parameter. One of the anomalies with modelling based on machine learning is the over fitting. Despite this, a classifier using classification approach might be handy because it can produce several trees for the random forest without over fitting the model.

3.4.4 K-Nearest Neighbour

It was first presented by Cover and Hart during 1968 and is a very sound technique based on theory and the simplest one among other classification approaches for data mining [37]. The closest K neighbouring values that can be utilised to represent each sample are referred to as the K nearest neighbours. Each row in a data sample may be categorized implementing the KNN algorithm.

The majority-majority-voting rule is exercised to pick the nearest K have known examples from all of the available pieces. This is how the KNN algorithm is implemented for classification issues: Fresh points are classified by utilising other points of known classes as a pointers [38]. The distance (denoted as d) between the two points is calculated as per this equation (3.10).

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}. \tag{3.10}$$

3.4.5 SVM

It is a statistical classification and regression analysis procedure that utilises this supervised learning technique the most. Among generalised linear classifiers, support vector machines excel in maximising geometric edge regions and minimising empirical errors.

The major ideology that backs a support vector machine is to represent the values into a space that has higher dimension then the previous one and separate them using hyper plane. On both sides of the hyper plane that separate the data, two parallel hyper planes are constructed. Singh et al., [39], also claim that the split hyper planes optimise the spacing between the two parallel hyper planes. The overall error of the classifier is presumptive decreasing with increasing parallel hyperplane separation.

According to Bhattacharyya et al., SVM was utilised in this study to address the multi-class classification problem [40].

CHAPTER 4

Result and Discussion

4.1 Results

The credit card bank frauds data set is obtained from the website Kaggle.com. In addition, we pre-processed data and then scaled. We chose features and applied the both under sampling and oversampling techniques such that smote algorithm to deal with the imbalanced data set. Then we trained various classification models. First two algorithms are logistic regression classifier and KNN while the other include SVM classifier, decision tree classifiers, and random forest classifier. The approach can identify the fraud purchase cases among clean cases. After that, these models were evaluated on various efficiency measuring metrics like recall,F_1 score, precision, and accuracy as shown in Table 4.1.

Model	Accuracy	Precision	Recall	F_1 Score
Logistic Regression	0.9846	0.2937	0.7285	0.3639
Support Vector Machine	0.9876	0.2891	0.5	0.3177
Decision Tree	0.9818	0.2012	0.7285	0.3015
K Nearest Neighbour	0.9973	0.6324	0.7	0.6561
Random Forest	0.9972	0.5723	0.6571	0.5896

Table 4.1: A table of performance metrics score against each Classifier.

4.1.1 Accuracy Score

On under sample data logistic regression emerged to be best with a 100.00% accuracy as shown in Table 4.2 which may be the case of over fitting. However, the Random Forest achieved an accuracy score of 0.9966 using oversampling (SMOTE). Despite the fact that credit card spoofing detection is becoming more sophisticated, the majority of current research still relies on decision trees and logistic regression tests. However, five classifier were used to compare training results on both sampling techniques to cater class imbalance problem of the credit card data set. I also believe that substantial outcomes arose. Random Forest performed admirably, and we overcame the sample imbalance issue to achieve meaningful results.

Model	Under Sampling	SMOTE
Logistic Regression	1.000000	0.991406
Decision Tree	0.944444	0.969398
Random Forest	0.972222	0.996646
SVM	0.944444	0.989939
K Nearest Neighbour	0.972222	0.996437

Table 4.2: A table of accuracy score against each classifier for both under sampling and over sampling based on SMOTE.

4.1.2 ROC Curve

The ROC curve is a valuable tool for evaluating how well various classifiers work. The classifier scores as blue for logistics regression, orange for KNN, green for SVM, red for decision tree, and purple for random forest. However, the minimum ROC score of 50% is shown by the dashed curve. It can also be used to select a classifier with a desired level of performance. In our case, Decision Tree,KNN, Random Forest classifiers all three somehow have similar ROC curves, but the Support Vector Machine and logistic regression classifiers show similar pattern. Over all SVM has a slightly higher TPR and a slightly lower FPR as shown in Figure 4.1. SVM shows maximum area under the curve as it covers 99.41% of whole area under it curve while logistic regression a little behind as it covers 99.08% of the over all area under its cover. All the classifiers shows

very good result with score greater that 90%.

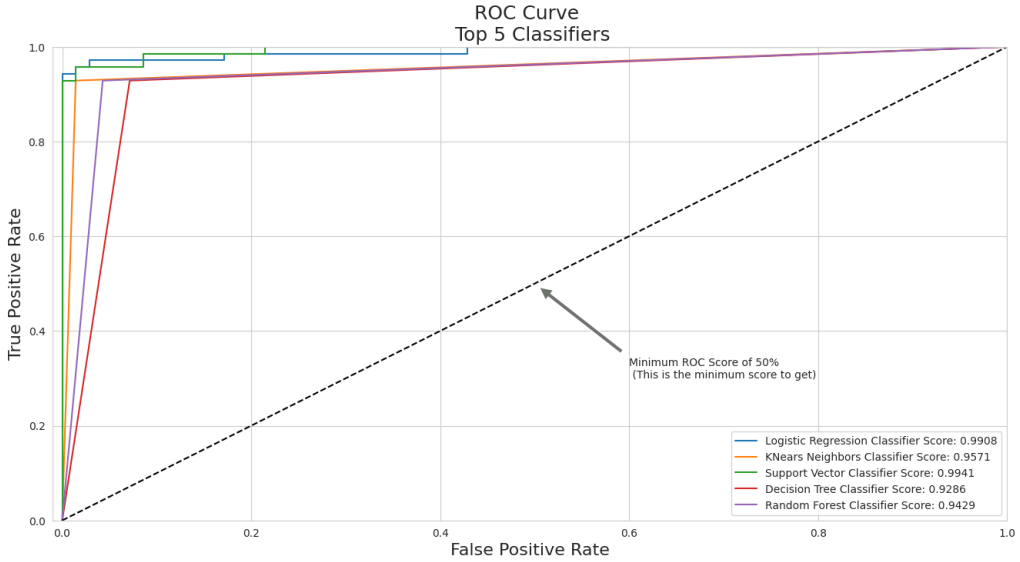

Figure 4.1: The solid curves are the classifier scores by using logistics regression (blue), KNN (orange), SVM (green), decision tree (red), and random forest (purple). The dashed curve is the minimum ROC score of 50%.

4.1.3 Model Learning Curves

A learning curve highlights the efficiency of a machine learning model on both training set and validation set with their increase. The x-axis of each graph is labeled "Training examples" and represents training examples numbers utilized for model trainig. The y-axis is labeled "Score" and represents the efficiency for both sets used by model.

The blue line in each graph represents the training score, training set efficiency shown by the model. The red line represents the testing score, validation set efficiency displayed by the model. In both Support Vector Machine shown in Figure 4.6 and Logistic Regression shown in Figure 4.2 models, the validation curve outperformed the training curve. However, in KNN shown in Figure 4.3 the results of both learning curves are very close to each other, with higher training scores. On the other hand, the training scores of both decision trees shown in Figure 4.4 and random forests are way higher than those of validation, as shown in Figure 4.5.

Figure 4.2: The Logistic Regression training scores are presented by the red curves, while the cross-validation scores are shown by the blue curves.

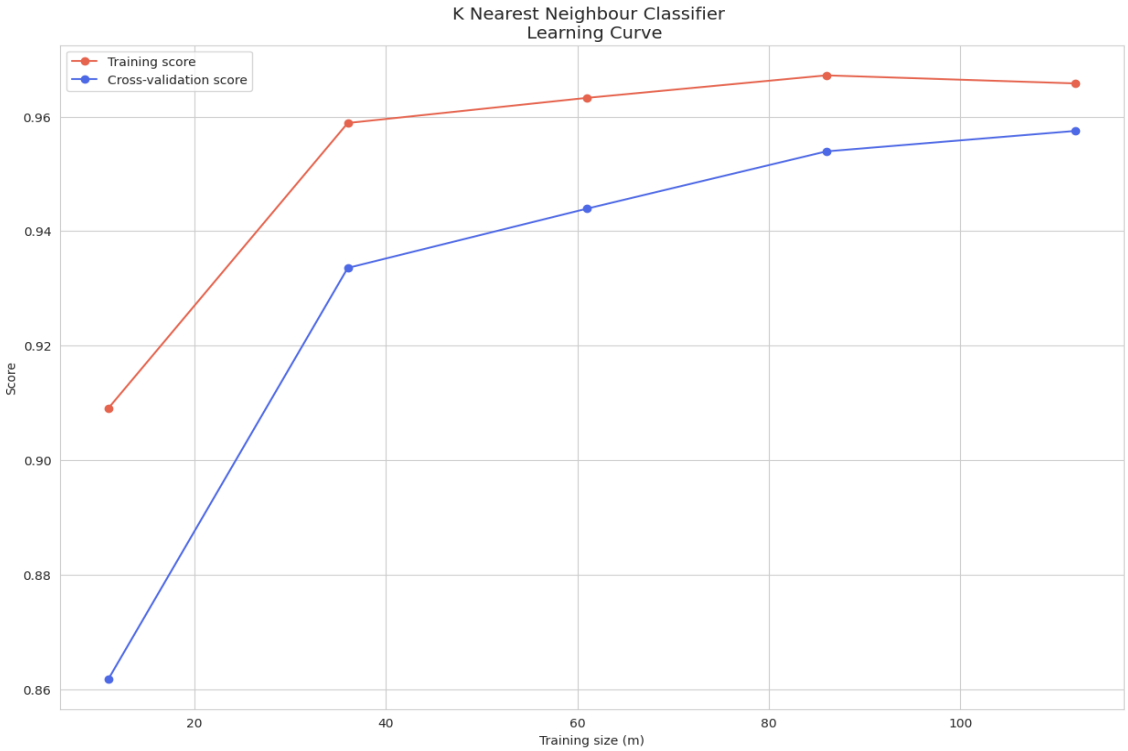

Figure 4.3: The K Nearest neighbour training scores are presented by the red curves, while the cross-validation scores are shown by the blue curves.

4.1.4 Confusion Matrices

A table called a error matrix **??** is employed to assess the effectiveness of a classifier. It compares the predicted values with the actual values and shows how many predictions were correct and how many were incorrect. The classifier Logistic Regression is the first of them and its confusion matrix is shown in the Figure 4.7. The confusion matrix of KNN is displayed in Figure 4.8, Similarly, the confusion matrices of Decision tree and SVM are sho in Figures 4.9 and 4.11 respectivly. The last of them is Random Forest Classifier and its confusion matrix is visualised in the Figure 4.10. All five models predicted 18 True Positive results, with predicting True Negative best figures shown by Logistic regression with 18 values. KNN and random forest predicted one False Positive and 17 True Negatives but both SVM and decision tree predicted two False Positives and 16 True Negatives as shown in Figure **??**. Logistics regression showed the best accuracy score by predicting 35 right values out of 36 among all models.

Figure 4.4: The Decision Tree training scores are presented by the red curves, while the cross-validation scores are shown by the blue curves.

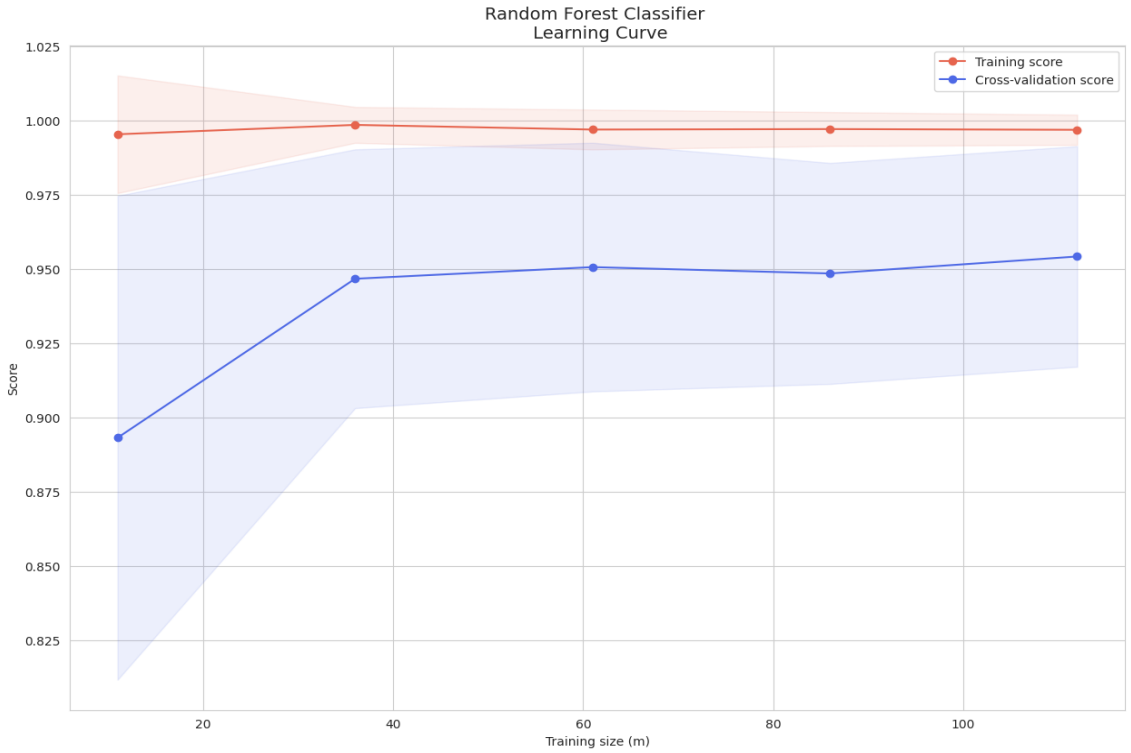

Figure 4.5: The Random Forest training scores are presented by the red curves, while the cross-validation scores are shown by the blue curves.

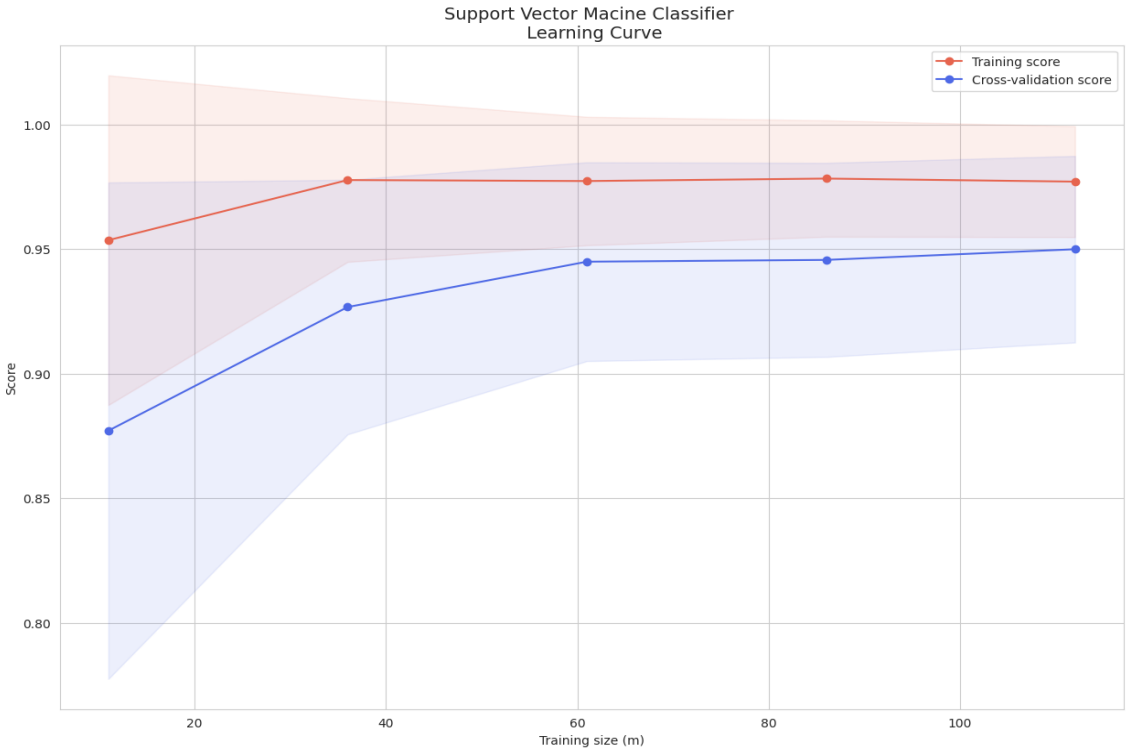

Figure 4.6: The Support Vector Machine training scores are presented by the red curves, while the cross-validation scores are shown by the blue curves.

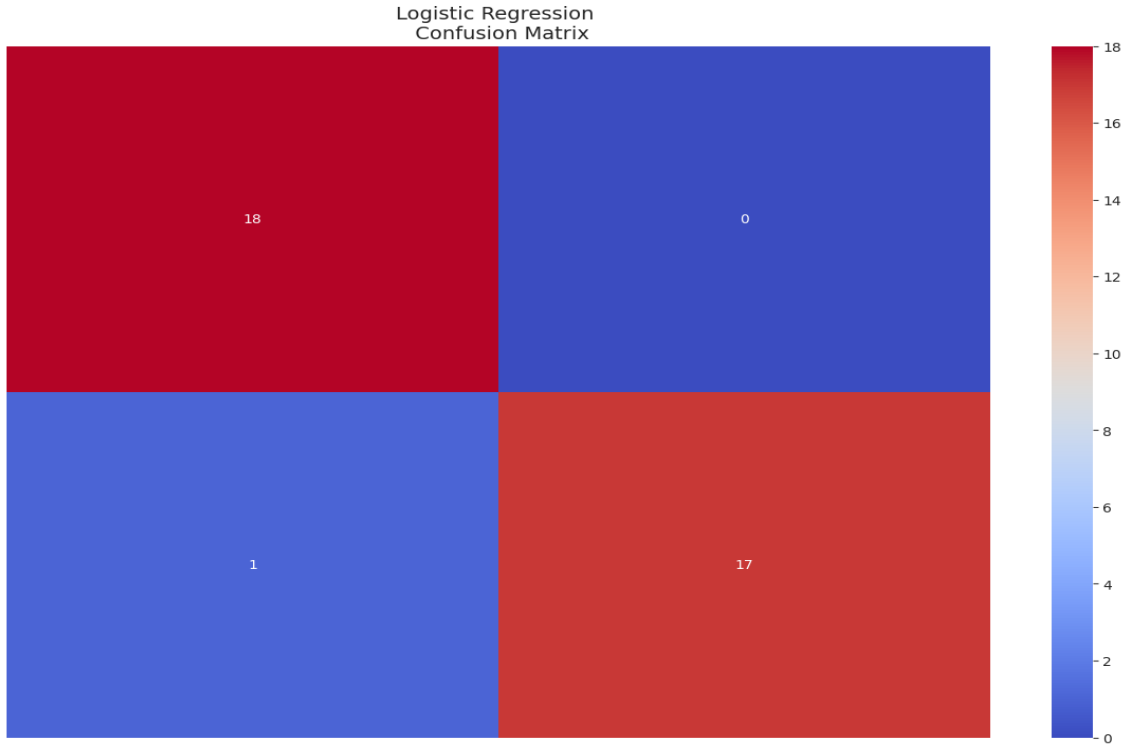

Figure 4.7: Error matrix based on the result of logistic regression classifier.

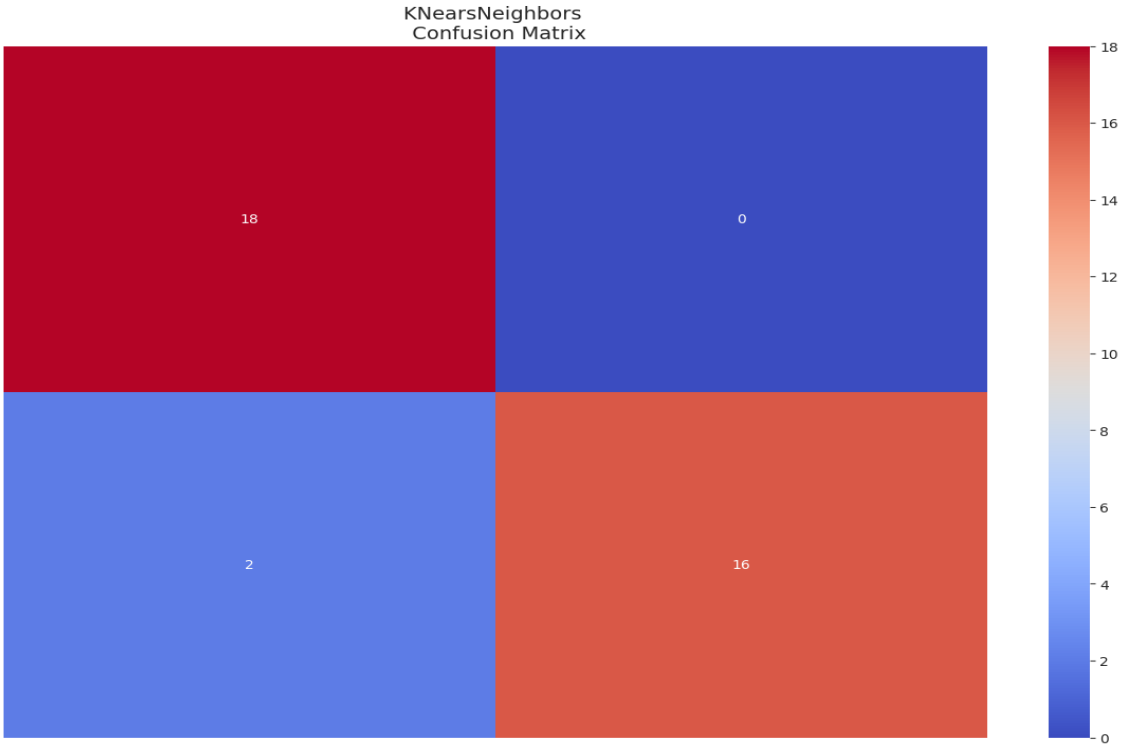

Figure 4.8: Error matrix based on the result of K Nearest Neighbour.

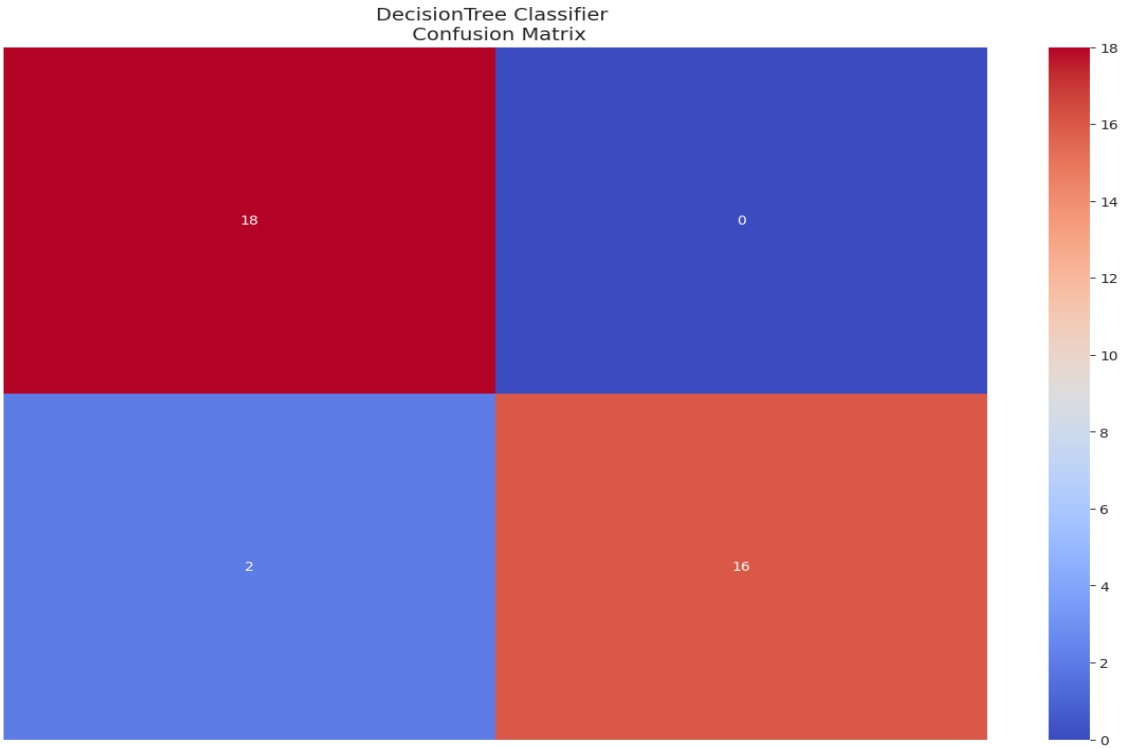

Figure 4.9: Error matrix based on the result of decision tree classifier.

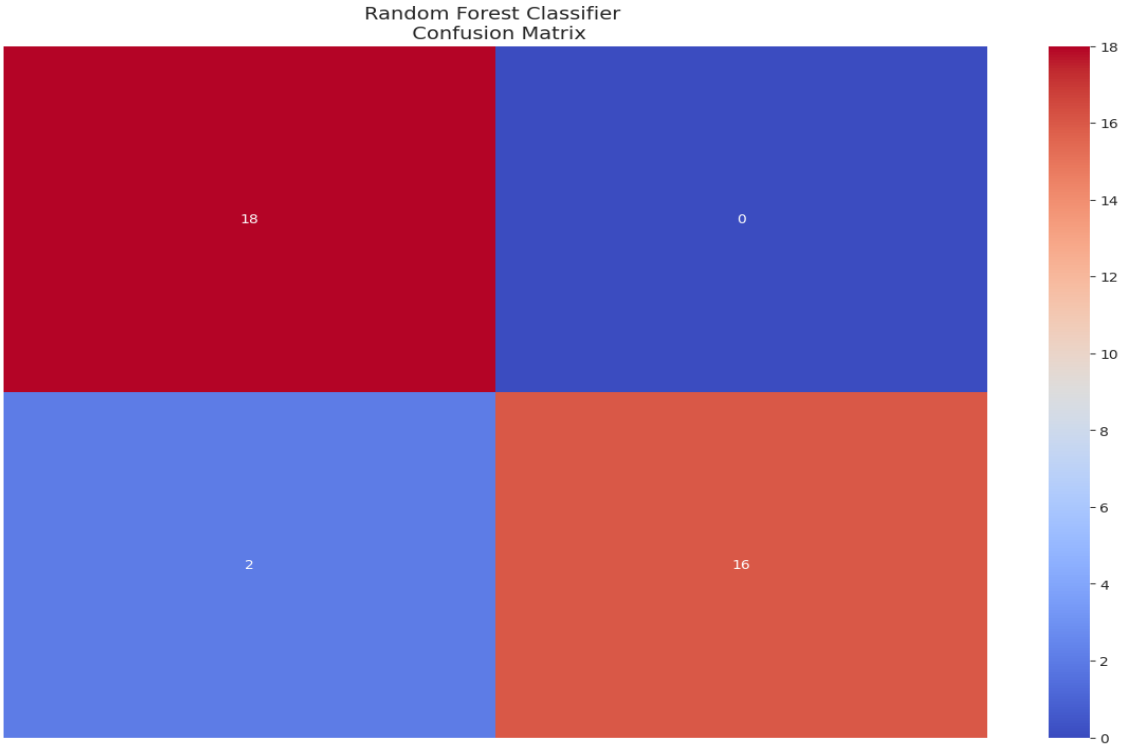

Figure 4.10: Error matrix based on the result of random forest.

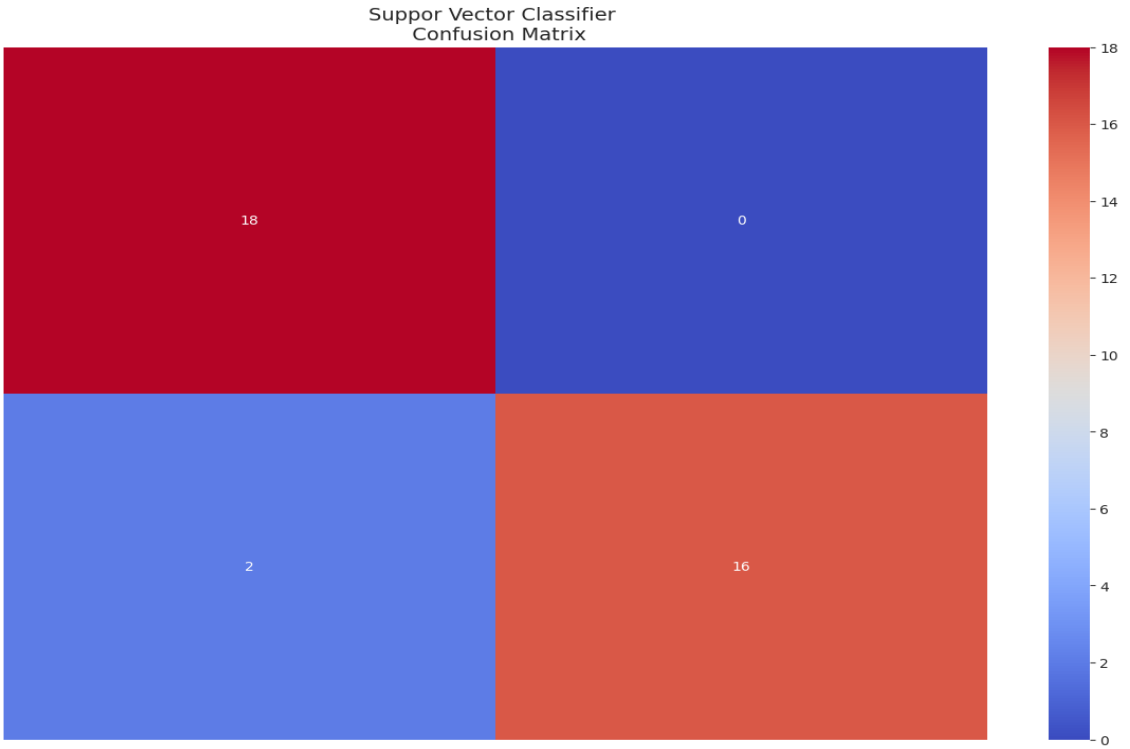

Figure 4.11: Error matrix based on the result of support vector machine.

CHAPTER 5

Conclusion

SMOTE helped us tackle the non-uniformity of our dataset, as we have more clean transactions than fraudulent ones. However, keep in mind that the outlier elimination was only done on the under sample dataset. Furthermore, our model is not worked efficiently in order to identify clean transactions. However, it predicted the clean cases as fraudulent transactions. Assume that consumers who were purchasing regularly and but unable to make purchases due to our model categorised these purchases as fraudulent; It can great disadvantage for retailers. As a result, the customer complaints and discontent will increase.

Credit card theft is influenced by the irregular form of transaction distributions since it drives criminals to always come up with new schemes to commit fraud. So, it is critical to consider these emerging behaviour while developing a predictive model. AUC-ROC, precision, recall, and F_1 score metrics are used to assess performance. K Nearest Neighbour and Random Forest perform the best, with 99.73% and 99.72% accuracy, respectively. On the balanced dataset, AUC-ROC revealed that SVM has a little higher TPR. The most consistent performance across metrics and sampling techniques is support vector machine classification. Future research should look at combination of classifiers into ensembles. However, a significant amount of data is required for this investigation. [41]

Bibliography

[1] E. Ileberi, Y. Sun, and Z. Wang, "A machine learning based credit card fraud detection using the ga algorithm for feature selection," *Journal of Big Data*, vol. 9, no. 1, pp. 1–17, 2022.

[2] V. N. Dornadula and S. Geetha, "Credit card fraud detection using machine learning algorithms," *Procedia computer science*, vol. 165, pp. 631–641, 2019.

[3] F. N. Ogwueleka, "Data mining application in credit card fraud detection system," *Journal of Engineering Science and Technology*, vol. 6, no. 3, pp. 311–322, 2011.

[4] S. Maniraj, A. Saini, S. Ahmed, and S. Sarkar, "Credit card fraud detection using machine learning and data science," *International Journal of Engineering Research*, vol. 8, no. 9, pp. 110–115, 2019.

[5] Jul 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Credit_card_fraud&oldid=970300096

[6] A. Hebert, A. Hernandez, R. Perkins, A. Puig, S. Levine, D. M., Garush, and M. Key, "Need help spotting, avoiding, and reporting scams? start with money matters," May 2023. [Online]. Available: https://consumer.ftc.gov/consumer-alerts/2022/12/need-help-spotting-avoiding-and-reporting-scams-start-money-matters

[7] E. Duman and I. Elikucuk, "Solving credit card fraud detection problem by the new metaheuristics migrating birds optimization," in *Advances in Computational Intelligence: 12th International Work-Conference on Artificial Neural Networks, IWANN 2013, Puerto de la Cruz, Tenerife, Spain, June 12-14, 2013, Proceedings, Part II 12*. Springer, 2013, pp. 62–71.

[8] T. P. Bhatla, V. Prabhu, and A. Dua, "Understanding credit card frauds," *Cards business review*, vol. 1, no. 6, pp. 1–15, 2003.

[9] M. Shufyan and P. Prashun, "An optimized machine learningalgorithms for solving class imbalance problem in credit card fraud detection," 2022.

[10] V. Chole, A. Mukherjee, K. Gaikwad, P. Gawai, P. Bagde, R. Mahule, and P. Pawar, "Revelation of credit card fraud using machine learning algorithm," 2022.

[11] S. Khan, A. Alourani, B. Mishra, A. Ali, and M. Kamal, "Developing a credit card fraud detection model using machine learning approaches," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 3, 2022.

[12] D. Tanouz, R. Raja Subramanian, D. Eswar, G. V. Parameswara Reddy, A. Ranjith kumar, and C. V. N. M. praneeth, "Credit card fraud detection using machine learning," in *2021 Fifth International Conference on ISMAC (IoT in Social Mobile Analytics and Cloud) (I-SMAC)*. IEEE, 2021, pp. 1–6.

[13] M. K. R. Mallidi and Y. Zagabathuni, "Analysis of credit card fraud detection using machine learning models on balanced and imbalanced datasets," *International Journal of Emerging Trends in Engineering Research*, vol. 8, no. 7, pp. 1389–1393, 2021.

[14] A. Gupta, M. Lohani, and M. Manchanda, "Financial fraud detection using naive bayes algorithm in highly imbalance data set," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 24, no. 5, pp. 1559–1572, 2021.

[15] M. U. Safa and R. Ganga, "Credit card fraud detection using machine learning," *International Journal of Research in Engineering, Science and Management*, vol. 2, no. 11, pp. 372–374, 2019.

[16] S. Kiran, J. Guru, R. Kumar, N. Kumar, D. Katariya, and M. Sharma, "Credit card fraud detection using naïve bayes model based and knn classifier," *International Journal of Advance Research, Ideas and Innovations in Technoloy*, vol. 4, no. 3, p. 44, 2018.

[17] S. D. Benchaji and B. E. Ouahidi, "Credit card fraud detection model based on lstm recurrent neural networks," *Journal of Advances in Information Technology*, vol. 12, no. 2, pp. 113–118, 2021.

[18] D. Varmedja, M. Karanovic, S. Sladojevic, M. Arsenovic, and A. Anderla, "Credit card fraud detection machine learning methods," in *Proceedings of the 18th International Symposium on INFOTEH-JAHORINA (INFOTEH)*. IEEE, 2019, pp. 1–5.

[19] H. Najadat, O. Altiti, A. A. Aqouleh, and M. Younes, "Credit card fraud detection based on machine and deep learning," in *Proceedings of the 11th International Conference on Information and Communication Systems (ICICS)*. IEEE, 2020, pp. 204–208.

[20] O. Adepoju, J. Wosowei, H. Jaiman *et al.*, "Comparative evaluation of credit card fraud detection using machine learning techniques," in *2019 Global Conference for Advancement in Technology (GCAT)*. IEEE, 2019, pp. 1–6.

[21] J. O. Awoyemi, A. O. Adetunmbi, and S. A. Oluwadare, "Credit card fraud detection using machine learning techniques: A comparative analysis," in *2017 international conference on computing networking and informatics (ICCNI)*. IEEE, 2017, pp. 1–9.

[22] A. Mohari, J. Dowerah, K. Das, F. Koucher, and D. J. Bora, "Credit card fraud detection techniques: A review," *Soft Computing for Intelligent Systems: Proceedings of ICSCIS 2020*, pp. 157–166, 2021.

[23] K. Ayorinde, *A Methodology for Detecting Credit Card Fraud*. Minnesota State University, Mankato, 2021.

[24] F. Braun, O. Caelen, E. N. Smirnov, S. Kelk, and B. Lebichot, "Improving card fraud detection through suspicious pattern discovery," in *Advances in Artificial Intelligence: From Theory to Practice: 30th International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2017, Arras, France, June 27-30, 2017, Proceedings, Part II 30*. Springer, 2017, pp. 181–190.

[25] C. Meng, L. Zhou, and B. Liu, "A case study in credit fraud detection with smote and xgboost," in *Journal of Physics: Conference Series*, vol. 1601, no. 5. IOP Publishing, 2020, p. 052016.

[26] J. Parmar, A. Patel, and M. Savsani, "Credit card fraud detection frameworkâa machine learning perspective," *International Journal of Scientific Research in Science and Technology*, vol. 7, no. 6, pp. 431–435, 2020.

[27] S. Shirgave, C. Awati, R. More, and S. Patil, "A review on credit card fraud detection using machine learning," *International Journal of Scientific & technology research*, vol. 8, no. 10, pp. 1217–1220, 2019.

[28] A. Priya, A. S. Narayanan, S. M. Bala, and B. D. Patel, "Optimal algorithm for credit card fraud detection," in *2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS)*. IEEE, 2022, pp. 1091–1098.

[29] R. More, C. Awati, S. Shirgave, R. Deshmukh, and S. Patil, "Credit card fraud detection using supervised learning approach," *International journal of scientific & technology research*, vol. 9, no. 10, pp. 216–219, 2021.

[30] D. Rahmawati, R. Sarno, C. Fatichah, and D. Sunaryono, "Fraud detection on event log of bank financial credit business process using hidden markov model algorithm," in *2017 3rd International Conference on Science in Information Technology (ICSITech)*. IEEE, 2017, pp. 35–40.

[31] B. C. da Rocha and R. T. de Sousa Junior, "Identifying bank frauds using crisp-dm and decision trees," *International Journal of Computer Science and Information Technology*, vol. 2, no. 5, pp. 162–169, 2010.

[32] M. Jisha and D. V. Kumar, "Population based optimized and condensed fuzzy deep belief network for credit card fraudulent detection," *International Journal of Advanced Computer Science and Applications*, vol. 11, no. 9, 2020.

[33] J. Brownlee, "Random oversampling and undersampling for imbalanced classification," Jan 2021. [Online]. Available: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

[34] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[35] Y. G. Şahin and E. Duman, "Detecting credit card fraud by decision trees and support vector machines," 2011.

[36] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[37] T. Cover, "Estimation by the nearest neighbor rule," *IEEE Transactions on Information Theory*, vol. 14, no. 1, pp. 50–55, 1968.

[38] A. C. Bahnsen, D. Aouada, and B. Ottersten, "Example-dependent cost-sensitive logistic regression for credit scoring," in *2014 13th International conference on machine learning and applications*. IEEE, 2014, pp. 263–269.

[39] S. K. Singh, P. K. Srivastava, M. Gupta, J. K. Thakur, and S. Mukherjee, "Appraisal of land use/land cover of mangrove forest ecosystem using support vector machine," *Environmental earth sciences*, vol. 71, no. 5, pp. 2245–2255, 2014.

[40] S. Bhattacharyya, S. Jha, K. Tharakunnel, and J. C. Westland, "Data mining for credit card fraud: A comparative study," *Decision support systems*, vol. 50, no. 3, pp. 602–613, 2011.

[41] A. V. Etten, D. Lindenbaum, and T. M. Bacastow, "Spacenet: A remote sensing dataset and challenge series," 2019.

APPENDIX

My Python Code

Listing A.1: Libraries

```
from sklearn.pipeline import make_pipeline as MP
import numpy as NPY
from imblearn.over_sampling import SMOTE as SMT
import matplotlib.pyplot as PLT
from imblearn.pipeline import make_pipeline as IMP
import time
from sklearn.ensemble import RandomForestClassifier as RFC
import matplotlib.patches as MP
import pandas as PD
from imblearn.under_sampling import NearMiss as NM
import warnings
from imblearn.metrics import classification_report_imbalanced
    ↪ as CRI
import collections
from sklearn.preprocessing import RobustScaler as ROSC
import seaborn as SB
from sklearn.metrics import  roc_auc_score as RAS, recall_score
    ↪  as RS, f1_score as F1,precision_score as PS,
```

```
    ↪ classification_report as CR,  accuracy_score as AS
from sklearn.metrics import roc_curve as ROC, confusion_matrix
    ↪ as CM
from sklearn.tree import DecisionTreeClassifier as DTC
from sklearn.manifold import TSNE
from sklearn.linear_model import LogisticRegression as LRC
from collections import Counter
from sklearn.svm import SVC as SVMC
from sklearn.model_selection import cross_val_score as CVS,
    ↪ StratifiedKFold as SKF, train_test_split as TTS, KFold as
    ↪  KF,cross_val_predict as CVP
from sklearn.model_selection import StratifiedShuffleSplit as
    ↪ SSS,RandomizedSearchCV as RSCV,GridSearchCV as GSCV,
    ↪ learning_curve as LC,ShuffleSplit as SS
from sklearn.neighbors import KNeighborsClassifier as KNC
warnings.filterwarnings("ignore")
# from sklearn.decomposition import  TruncatedSVD,PCA

cre_card_trans = PD.read_csv('/content/drive/MyDrive/Arshad-
    ↪ Dissertation/Dataset/creditcard.csv')
cre_card_trans.head()

cre_card_trans.shape

cre_card_trans.columns

cre_card_trans.isnull().sum()

cre_card_trans.dropna(inplace=True)

cre_card_trans.isnull().sum()
```

```
SB.set_style('whitegrid')
SB.color_palette("coolwarm", as_cmap=True)
SB.heatmap(cre_card_trans.isnull(),cmap="coolwarm")

cre_card_trans.describe().T

fraud_trans = cre_card_trans[cre_card_trans['Class'] == 1].
    ↪ describe().T
clean_trans = cre_card_trans[cre_card_trans['Class'] == 0].
    ↪ describe().T

fig,SP = PLT.subplots(nrows = 1,ncols = 6,figsize = (25,10))
SB.heatmap(fraud_trans[['mean']][:5],cmap="coolwarm",linewidths
    ↪  = 0.5,annot = True,cbar = False,fmt = '.2f',linecolor =
    ↪ 'brown',ax=SP[0])
SB.heatmap(fraud_trans[['mean']][5:10],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[1])
SB.heatmap(fraud_trans[['mean']][10:15],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[2])
SB.heatmap(fraud_trans[['mean']][15:20],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[3])
SB.heatmap(fraud_trans[['mean']][20:25],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[4])
SB.heatmap(fraud_trans[['mean']][25:30],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[5])

fig,SP = PLT.subplots(nrows = 1,ncols = 6,figsize = (25,10))
```

```
SB.heatmap(fraud_trans[['min']][:5],cmap="coolwarm",linewidths
    ↪ = 0.5,annot = True,cbar = False,fmt = '.2f',linecolor = '
    ↪ brown',ax=SP[0])
SB.heatmap(fraud_trans[['min']][5:10],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[1])
SB.heatmap(fraud_trans[['min']][10:15],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[2])
SB.heatmap(fraud_trans[['min']][15:20],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[3])
SB.heatmap(fraud_trans[['min']][20:25],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[4])
SB.heatmap(fraud_trans[['min']][25:30],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[5])

fig,SP = PLT.subplots(nrows = 1,ncols = 6,figsize = (25,10))
SB.heatmap(fraud_trans[['std']][:5],cmap="coolwarm",ax=SP[0],
    ↪ annot = True,linecolor = 'brown',linewidths = 0.5,fmt = '
    ↪ .2f',cbar = False)
SB.heatmap(fraud_trans[['std']][5:10],cmap="coolwarm",ax=SP[1],
    ↪ annot = True,linecolor = 'brown',linewidths = 0.5,fmt = '
    ↪ .2f',cbar = False)
SB.heatmap(fraud_trans[['std']][10:15],cmap="coolwarm",ax=SP[2]
    ↪ ,annot = True,linecolor = 'brown',linewidths = 0.5,fmt =
    ↪ '.2f',cbar = False)
SB.heatmap(fraud_trans[['std']][15:20],cmap="coolwarm",ax=SP[3]
    ↪ ,annot = True,linecolor = 'brown',linewidths = 0.5,fmt =
    ↪ '.2f',cbar = False)
```

```
SB.heatmap(fraud_trans[['std']][20:25],cmap="coolwarm",ax=SP[4]
    ↪ ,annot = True,linecolor = 'brown',linewidths = 0.5,fmt =
    ↪ '.2f',cbar = False)
SB.heatmap(fraud_trans[['std']][25:30],cmap="coolwarm",ax=SP[5]
    ↪ ,annot = True,linecolor = 'brown',linewidths = 0.5,fmt =
    ↪ '.2f',cbar = False)

fig,SP = PLT.subplots(nrows = 1,ncols = 6,figsize = (25,10))
SB.heatmap(fraud_trans[['max']][:5],cmap="coolwarm",linewidths
    ↪ = 0.5,annot = True,cbar = False,fmt = '.2f',linecolor = '
    ↪ brown',ax=SP[0])
SB.heatmap(fraud_trans[['max']][5:10],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[1])
SB.heatmap(fraud_trans[['max']][10:15],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[2])
SB.heatmap(fraud_trans[['max']][15:20],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[3])
SB.heatmap(fraud_trans[['max']][20:25],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[4])
SB.heatmap(fraud_trans[['max']][25:30],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[5])

fraud_trans.rename(columns={"50%": "Median"}, inplace = True)
fig,SP = PLT.subplots(nrows = 1,ncols = 6,figsize = (25,10))
SB.heatmap(fraud_trans[['Median']][:5],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[0])
```

```
SB.heatmap(fraud_trans[['Median']][5:10],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[1])
SB.heatmap(fraud_trans[['Median']][10:15],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[2])
SB.heatmap(fraud_trans[['Median']][15:20],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[3])
SB.heatmap(fraud_trans[['Median']][20:25],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[4])
SB.heatmap(fraud_trans[['Median']][25:30],cmap="coolwarm",
    ↪ linewidths = 0.5,annot = True,cbar = False,fmt = '.2f',
    ↪ linecolor = 'brown',ax=SP[5])

X_Vals = [0, 1]
Y_Vals = [cre_card_trans['Class'].value_counts()[1],
    ↪ cre_card_trans['Class'].value_counts()[0],]
fig, SP = PLT.subplots()
SP.set_yscale('log')
SP.bar(X_Vals, Y_Vals,color=[ '#E6644E','#4E69E6'], tick_label
    ↪ =[ 'Fraud','Clean',])
SP.set_xlabel('Class')
SP.set_ylabel('Count(log␣scale)')
SP.set_title('Imbalanced␣Classes')

RoSc = ROSC()
cre_card_trans['scaled_amount'] = RoSc.fit_transform(
    ↪ cre_card_trans['Amount'].values.reshape(-1,1))
cre_card_trans['scaled_time'] = RoSc.fit_transform(
    ↪ cre_card_trans['Time'].values.reshape(-1,1))
```

```
cre_card_trans['Amount']=cre_card_trans['scaled_amount']
cre_card_trans['Time']=cre_card_trans['scaled_time']
cre_card_trans.drop(['scaled_time','scaled_amount'], axis=1,
    ↪ inplace=True)
cre_card_trans.head()

print('Clean␣Transactions=', round(cre_card_trans['Class'].
    ↪ value_counts()[0]/len(cre_card_trans) * 100,2), '%')
print('Fraud␣Transactions=', round(cre_card_trans['Class'].
    ↪ value_counts()[1]/len(cre_card_trans) * 100,2), '%')

print('Total␣Fraud␣Transactions␣in␣the␣DataSet␣=',
    ↪ cre_card_trans['Class'].value_counts()[1])
print('Total␣Clean␣Transactions␣in␣the␣DataSet␣=',
    ↪ cre_card_trans['Class'].value_counts()[0])

X = cre_card_trans.drop('Class', axis=1)
y = cre_card_trans['Class']

skf = SKF(n_splits=5, shuffle=False, random_state=None)

for tr_idx, tst_idx in  skf.split(X, y):
    print("Train:", tr_idx, "Test:", tst_idx)
    org_Xtr, org_X_tst = X.iloc[tr_idx], X.iloc[tst_idx]
    org_ytr, org_y_tst = y.iloc[tr_idx], y.iloc[tst_idx]

org_Xtr = org_Xtr.values
org_X_tst = org_X_tst.values
org_ytr = org_ytr.values
org_y_tst = org_y_tst.values
```

```
train_unique_label, tr_cnt_lbl = NPY.unique(org_ytr,
    ↪ return_counts=True)
test_unique_label, tst_cnt_lbl = NPY.unique(org_y_tst,
    ↪ return_counts=True)

print('Label Distributions: \n')
print(tr_cnt_lbl/ len(org_ytr))
print(tst_cnt_lbl/ len(org_y_tst))

cre_card_trans = cre_card_trans.sample(frac=1)

total_fraud_trans=cre_card_trans['Class'].value_counts()[1]
fraud_df = cre_card_trans.loc[cre_card_trans['Class'] == 1]
clean_df = cre_card_trans.loc[cre_card_trans['Class'] == 0][:
    ↪ total_fraud_trans]

normal_distributed_df = PD.concat([fraud_df, clean_df])

nd_samp = normal_distributed_df.sample(frac=1, random_state=42)

nd_samp.head()

SB.countplot(x='Class',data=nd_samp,palette="coolwarm")
PLT.title('Equally Distributed Classes', fontsize=14)
PLT.show()

f, (SP1, SP2) = PLT.subplots(1, 2, figsize=(24,16))

# Entire DataFrame
corr = cre_card_trans.corr()
SB.heatmap(corr, cmap='coolwarm_r', annot_kws={'size':20}, ax=
    ↪ SP1)
```

```
SP1.set_title("Imbalanced␣Correlation␣Heatmap␣\n␣", fontsize
    ↪ =14)



sub_sample_corr = nd_samp.corr()
SB.heatmap(sub_sample_corr, cmap='coolwarm_r', annot_kws={'size
    ↪ ':20}, ax=SP2)
SP2.set_title('SubSample␣Correlation␣Heatmap␣\n␣', fontsize=14)
PLT.show()



col_vals = cre_card_trans.columns.values
col_vals=NPY.delete(col_vals, 30)
i = 0
C = cre_card_trans.loc[cre_card_trans['Class'] == 0]
F = cre_card_trans.loc[cre_card_trans['Class'] == 1]
PLT.figure()
fig, SP = PLT.subplots(6,5,figsize=(16,28))

for v in col_vals:
    i += 1
    PLT.subplot(6,5,i)

    SB.kdeplot(C[v],label="Class␣=␣0", bw_method=0.5)
    SB.kdeplot(F[v],label="Class␣=␣0", bw_method=0.5)
    PLT.xlabel(v, fontsize=14)
    locs, labels = PLT.xticks()
    PLT.tick_params( which='major', labelsize=10,axis='both')

PLT.show();
```

```
target = 'Class'
predictors = ['scaled_amount','scaled_time', 'V1',  'V3', 'V5',
    ↪  'V7', 'V9','V11', 'V13', 'V15', 'V17', 'V19','V21', 'V23
    ↪ ', 'V25', 'V27',\
          'V2', 'V4', 'V6', 'V8', 'V10', 'V12', 'V14', 'V16', '
             ↪ V18','V20', 'V22', 'V24', 'V26', 'V28'\
         ]

tr_trans, tst_trans = TTS(cre_card_trans, random_state=42,
    ↪ shuffle=True, test_size=0.20 )
tr_trans, val_trans = TTS(tr_trans, random_state=42, shuffle=
    ↪ True, test_size=0.20 )

X = nd_samp.drop('Class', axis=1)
y = nd_samp['Class']

tr_X, tst_X, tr_y, tst_y = TTS(X, y, test_size=0.2,
    ↪ random_state=42)

clfs = {
     "Logisitic␣Regression": LRC(),
     "Decision␣Tree": DTC(),
     "Random␣Forests": RFC(),
     "K␣Nearest␣Neighbour": KNC(),
     "Support␣Vector␣Machine": SVMC()
}

for key, clf in clfs.items():
     clf.fit(tr_X, tr_y)
     acc_sc = CVS(clf, tr_X, tr_y, cv=5)
     print("Machine␣Learning␣Classifiers:␣", key, "has␣a␣
        ↪ accuracy␣score␣of", round(acc_sc.mean(), 2) * 100, "%
```

```
        ↪ ")

Param_LR = { 'C': [1, 0.1, 10, 0.01, 100,.001, 1000],"penalty":
    ↪  ['l2', 'l1']}
gr_lrc = GSCV(LRC(), Param_LR)
gr_lrc.fit(tr_X, tr_y)

glr_be = gr_lrc.best_estimator_
print("Logistic Regression:",glr_be)
Param_KN = {"n_neighbors": list(range(2,5,1)), 'algorithm': ['
    ↪ auto', 'ball_tree', 'kd_tree', 'brute']}

gr_knc = GSCV(KNC(), Param_KN)
gr_knc.fit(tr_X, tr_y)

gkn_be = gr_knc.best_estimator_
print("KNN:",gkn_be)

Param_SVMC = {'C': [0.5, 0.7, 0.9, 1], 'kernel': ['rbf', 'poly'
    ↪ , 'sigmoid', 'linear']}
gr_svmc = GSCV(SVMC(), Param_SVMC)
gr_svmc.fit(tr_X, tr_y)



gsvmc_be = gr_svmc.best_estimator_
print("Support Vector Machine:",gsvmc_be)

Param_DT = {"criterion": ["gini", "entropy"], "max_depth": list
    ↪ (range(2,4,1)),
                "min_samples_leaf": list(range(5,7,1))}
gr_dtc = GSCV(DTC(), Param_DT)
gr_dtc.fit(tr_X, tr_y)
```

```
gdt_be = gr_dtc.best_estimator_
print("Decision Tree:",gdt_be)

Param_RFC = {"n_estimators"      : [10,20,30],
             "max_features"      : ["sqrt", "log2"],
             "min_samples_split" : [2,4,8],
             "bootstrap": [True, False]}
gr_rfc = GSCV(RFC(), Param_RFC, n_jobs=-1, cv=5)
gr_rfc.fit(tr_X, tr_y)

grf_be = gr_rfc.best_estimator_
print("Random Forest:",grf_be)

lrc_sc = CVS(gr_lrc, tr_X, tr_y, cv=5)
print('Logistic Regression Cross Validation Score ', round(
    ↪ lrc_sc.mean() * 100, 2), '%')
knc_sc = CVS(gr_knc, tr_X, tr_y, cv=5)
print('Knears Neighbors Cross Validation Score', round(knc_sc.
    ↪ mean() * 100, 2), '%')
svmc_sc = CVS(gr_svmc, tr_X, tr_y, cv=5)
print('Support Vector Classifier Cross Validation Score', round
    ↪ (svmc_sc.mean() * 100, 2), '%')
dtc_sc = CVS(gr_dtc, tr_X, tr_y, cv=5)
print('DecisionTree Classifier Cross Validation Score', round(
    ↪ dtc_sc.mean() * 100, 2), '%')
rfc_sc = CVS(gr_rfc, tr_X, tr_y, cv=5)
print('Random Forest Classifier Cross Validation Score', round(
    ↪ rfc_sc.mean() * 100, 2), '%')

X_us = cre_card_trans.drop('Class', axis=1)
y_us = cre_card_trans['Class']
```

```
for tr_idx, tst_idx in  skf.split(X=X_us,y= y_us):
    print("Train:", tr_idx, "Test:", tst_idx)
    tr_X_us, tst_X_us = X_us.iloc[tr_idx], X_us.iloc[tst_idx]
    tr_y_us, tst_y_us = y_us.iloc[tr_idx], y_us.iloc[tst_idx]

tr_X_us = tr_X_us.values
tst_X_us = tst_X_us.values
tr_y_us = tr_y_us.values
tst_y_us = tst_y_us.values

acc_us = []
pre_us = []
rec_us = []
f1_us = []
auc_us = []

nm_X, nm_y = NM().fit_resample(X_us.values, y_us.values)
print('NearMiss Label Distribution: {}'.format(Counter(nm_y)))

for train, test in skf.split(tr_X_us, tr_y_us):
    pl_us = IMP(NM(sampling_strategy='majority'), gr_lrc)
    mdl_us = pl_us.fit(tr_X_us[train], tr_y_us[train])
    pred_us = mdl_us.predict(tr_X_us[test])

    acc_us.append(pl_us.score(org_Xtr[test], org_ytr[test]))
    pre_us.append(PS(org_ytr[test], pred_us))
    rec_us.append(RS(org_ytr[test], pred_us))
    f1_us.append(F1(org_ytr[test], pred_us))
    auc_us.append(RAS(org_ytr[test], pred_us))

pre_gr_lrc = CVP(gr_lrc, tr_X, tr_y, cv=5,
```

```
                                    method="decision_function")
pre_gr_knc = CVP(gr_knc, tr_X, tr_y, cv=5)
pre_gr_svmc = CVP(gr_svmc, tr_X, tr_y, cv=5,
                                    method="decision_function")
pre_gr_dtc = CVP(gr_dtc, tr_X, tr_y, cv=5)
pre_gr_rfc = CVP(gr_rfc, tr_X, tr_y, cv=5)

print('Logistic Regression: ', RAS(tr_y, pre_gr_lrc))
print('KNears Neighbors: ', RAS(tr_y, pre_gr_knc))
print('Support Vector Classifier: ', RAS(tr_y, pre_gr_svmc))
print('Decision Tree Classifier: ', RAS(tr_y, pre_gr_dtc))
print('Random Forest Classifier: ', RAS(tr_y, pre_gr_rfc))

fpr_gr_lrc, tpr_gr_lrc, gr_lrc_th = ROC(tr_y, pre_gr_lrc)
fpr_gr_knc, tpr_gr_knc, gr_kncr_th = ROC(tr_y, pre_gr_knc)
fpr_gr_svmc, tpr_gr_svmc, gr_svmc_th = ROC(tr_y, pre_gr_svmc)
fpr_gr_dtc, tpr_gr_dtc, gr_dtc_th = ROC(tr_y, pre_gr_dtc)
fpr_gr_rfc, tpr_gr_rfc, gr_rfc_th = ROC(tr_y, pre_gr_rfc)


def multi_roc_plot(fpr_gr_lrc, tpr_gr_lrc, fpr_gr_knc,
   ↪ tpr_gr_knc, fpr_gr_svmc, tpr_gr_svmc, fpr_gr_dtc,
   ↪ tpr_gr_dtc, fpr_gr_rfc, tpr_gr_rfc):
    PLT.figure(figsize=(16,8))
    PLT.title('ROC Curve \n Top 5 Classifiers', fontsize=18)
    PLT.plot(fpr_gr_lrc, tpr_gr_lrc, label='Logistic Regression
       ↪ Classifier Score: {:.4f}'.format(RAS(tr_y,
       ↪ pre_gr_lrc)))
    PLT.plot(fpr_gr_knc, tpr_gr_knc, label='KNears Neighbors
       ↪ Classifier Score: {:.4f}'.format(RAS(tr_y, pre_gr_knc
       ↪ )))
```

```
    PLT.plot(fpr_gr_svmc, tpr_gr_svmc, label='Support Vector
        ↪ Classifier Score: {:.4f}'.format(RAS(tr_y,
        ↪ pre_gr_svmc)))
    PLT.plot(fpr_gr_dtc, tpr_gr_dtc, label='Decision Tree
        ↪ Classifier Score: {:.4f}'.format(RAS(tr_y, pre_gr_dtc
        ↪ )))
    PLT.plot(fpr_gr_rfc, tpr_gr_rfc, label='Random Forest
        ↪ Classifier Score: {:.4f}'.format(RAS(tr_y, pre_gr_rfc
        ↪ )))
    PLT.plot([0, 1], [0, 1], 'k--')
    PLT.axis([-0.01, 1, 0, 1])
    PLT.xlabel('False Positive Rate', fontsize=16)
    PLT.ylabel('True Positive Rate', fontsize=16)
    PLT.annotate('Minimum ROC Score of 50% \n (This is the
        ↪ minimum score to get)', xy=(0.5, 0.5), xytext=(0.6,
        ↪ 0.3),
                 arrowprops=dict(facecolor='#6E726D', shrink
                     ↪ =0.05),
                 )
    PLT.legend()

multi_roc_plot(fpr_gr_lrc, tpr_gr_lrc, fpr_gr_knc, tpr_gr_knc,
    ↪ fpr_gr_svmc, tpr_gr_svmc, fpr_gr_dtc, tpr_gr_dtc,
    ↪ fpr_gr_rfc, tpr_gr_rfc)
PLT.show()

acc_lst = []
prec_lst = []
rec_lst = []
f1_lst = []
auc_lst = []
log_reg_sm = LRC()
```

```
rand_log_reg = RSCV(LRC(), Param_LR, n_iter=4)
Param_LR = {"penalty": ['l1', 'l2'], 'C': [0.001, 0.01, 0.1, 1,
    ↪ 10, 100, 1000]}
for train, test in skf.split(org_Xtr, org_ytr):
    pl = IMP(SMT(sampling_strategy='minority'), rand_log_reg)
    mdl = pl.fit(org_Xtr[train], org_ytr[train])
    best_est = rand_log_reg.best_estimator_
    pr_val = best_est.predict(org_Xtr[test])

    acc_lst.append(pl.score(org_Xtr[test], org_ytr[test]))
    prec_lst.append(PS(org_ytr[test], pr_val))
    rec_lst.append(RS(org_ytr[test], pr_val))
    f1_lst.append(F1(org_ytr[test], pr_val))
    auc_lst.append(RAS(org_ytr[test], pr_val))

print('---' * 21)
print("accuracy: {}".format(NPY.mean(acc_lst)),"\nprecision: {}
    ↪ ".format(NPY.mean(prec_lst)),"\nrecall: {}".format(NPY.
    ↪ mean(rec_lst)),"\nf1: {}".format(NPY.mean(f1_lst)))
print('---' * 21)

labels = ['Clean', 'Fraud']
SMT_pr_val = best_est.predict(org_X_tst)
print(CR(org_y_tst, SMT_pr_val, target_names=labels))

sm = SMT(sampling_strategy='minority', random_state=42)
Xsm_train, ysm_train = sm.fit_resample(org_Xtr, org_ytr)

t0 = time.time()
log_reg_sm = gr_lrc.best_estimator_
log_reg_sm.fit(Xsm_train, ysm_train)
knears_neighbors=gr_knc.best_estimator_
```

```
gr_knc.fit(Xsm_train, ysm_train)
svc=gr_svmc.best_estimator_
svc.fit(Xsm_train, ysm_train)
tree_clf=gr_dtc.best_estimator_
tree_clf.fit(Xsm_train, ysm_train)
rf_clf=gr_rfc.best_estimator_
rf_clf.fit(Xsm_train, ysm_train)
t1 = time.time()
print("Fitting oversample data took :{} sec".format(t1 - t0))

y_pred_log_reg = log_reg_sm.predict(tst_X)
y_pred_knear = gr_knc.predict(tst_X)
y_pred_svc = svc.predict(tst_X)
y_pred_tree = tree_clf.predict(tst_X)
y_pred_rf = rf_clf.predict(tst_X)


log_reg_cf = CM(tst_y, y_pred_log_reg)

kneighbors_cf = CM(tst_y, y_pred_knear)
svc_cf = CM(tst_y, y_pred_svc)
tree_cf = CM(tst_y, y_pred_tree)
rf_cf = CM(tst_y, y_pred_rf)

f, (SP1,SP2,SP3,SP4,SP5) = PLT.subplots(5, 1,figsize=(15,30))


SB.heatmap(log_reg_cf, ax=SP1, annot=True, cmap="coolwarm")
SP1.set_title("Logistic Regression \n Confusion Matrix",
   ↪ fontsize=14)
SP1.set_xticklabels(['', ''], fontsize=14, rotation=90)
SP1.set_yticklabels(['', ''], fontsize=14, rotation=360)
```

```
SB.heatmap(kneighbors_cf, ax=SP2, annot=True, cmap="coolwarm")
SP2.set_title("KNearsNeighbors \n Confusion Matrix", fontsize
    ↪ =14)
SP2.set_xticklabels(['', ''], fontsize=14, rotation=90)
SP2.set_yticklabels(['', ''], fontsize=14, rotation=360)

SB.heatmap(svc_cf, ax=SP3, annot=True, cmap="coolwarm")
SP3.set_title("Suppor Vector Classifier \n Confusion Matrix",
    ↪ fontsize=14)
SP3.set_xticklabels(['', ''], fontsize=14, rotation=90)
SP3.set_yticklabels(['', ''], fontsize=14, rotation=360)

SB.heatmap(tree_cf, ax=SP4, annot=True, cmap="coolwarm")
SP4.set_title("DecisionTree Classifier \n Confusion Matrix",
    ↪ fontsize=14)
SP4.set_xticklabels(['', ''], fontsize=14, rotation=90)
SP4.set_yticklabels(['', ''], fontsize=14, rotation=360)

SB.heatmap(rf_cf, ax=SP5, annot=True, cmap="coolwarm")
SP5.set_title("Random Forest Classifier \n Confusion Matrix",
    ↪ fontsize=14)
SP5.set_xticklabels(['', ''], fontsize=14, rotation=90)
SP5.set_yticklabels(['', ''], fontsize=14, rotation=360)
PLT.show()

print('Logistic Regression:')
print(CR(tst_y, y_pred_log_reg))

print('KNears Neighbors:')
print(CR(tst_y, y_pred_knear))
```

```
print('Support␣Vector␣Classifier:')
print(CR(tst_y, y_pred_svc))

print('Decision␣Tree␣Classifier:')
print(CR(tst_y, y_pred_tree))

print('Random␣Forest␣Classifier:')
print(CR(tst_y, y_pred_rf))

acc_lst = []
prec_lst = []
rec_lst = []
f1_lst = []
auc_lst = []

dt_sm = DTC()
dt_sm = RSCV( DTC(), Param_DT, n_iter=4)
for train, test in skf.split(org_Xtr, org_ytr):
    pl = IMP(SMT(sampling_strategy='minority'), dt_sm)
    mdl = pl.fit(org_Xtr[train], org_ytr[train])
    best_est_tree = dt_sm.best_estimator_
    pr_val = best_est_tree.predict(org_Xtr[test])

    acc_lst.append(pl.score(org_Xtr[test], org_ytr[test]))
    prec_lst.append(PS(org_ytr[test], pr_val))
    rec_lst.append(RS(org_ytr[test], pr_val))
    f1_lst.append(F1(org_ytr[test], pr_val))
    auc_lst.append(RAS(org_ytr[test], pr_val))
print('---' * 21)
print("accuracy:␣{}".format(NPY.mean(acc_lst)),"\nprecision:␣{}
   ↪ ".format(NPY.mean(prec_lst)),"\nrecall:␣{}".format(NPY.
   ↪ mean(rec_lst)),"\nf1:␣{}".format(NPY.mean(f1_lst)))
```

```
print('---' * 21)

acc_lst = []
prec_lst = []
rec_lst = []
f1_lst = []
auc_lst = []
rf_sm = RFC()
rf_sm = RSCV( RFC(), Param_RFC, n_iter=4)
for train, test in skf.split(org_Xtr, org_ytr):
    pl = IMP(SMT(sampling_strategy='minority'), rf_sm)
    mdl = pl.fit(org_Xtr[train], org_ytr[train])
    best_est_rf = rf_sm.best_estimator_
    pr_val = best_est_rf.predict(org_Xtr[test])

    acc_lst.append(pl.score(org_Xtr[test], org_ytr[test]))
    prec_lst.append(PS(org_ytr[test], pr_val))
    rec_lst.append(RS(org_ytr[test], pr_val))
    f1_lst.append(F1(org_ytr[test], pr_val))
    auc_lst.append(RAS(org_ytr[test], pr_val))

print('---' * 21)
print("accuracy: {}".format(NPY.mean(acc_lst)),"\nprecision: {}
    ↪ ".format(NPY.mean(prec_lst)),"\nrecall: {}".format(NPY.
    ↪ mean(rec_lst)),"\nf1: {}".format(NPY.mean(f1_lst)))
print('---' * 21)

acc_lst = []
prec_lst = []
rec_lst = []
f1_lst = []
auc_lst = []
```

```
svc_sm = SVMC()
svc_sm = RSCV(SVMC(),  Param_SVMC, n_iter=4)
for train, test in skf.split(org_Xtr, org_ytr):
    pl = IMP(SMT(sampling_strategy='minority'), svc_sm)
    mdl = pl.fit(org_Xtr[train], org_ytr[train])
    best_est_svc = svc_sm.best_estimator_
    pr_val = best_est_svc.predict(org_Xtr[test])

    acc_lst.append(pl.score(org_Xtr[test], org_ytr[test]))
    prec_lst.append(PS(org_ytr[test], pr_val))
    rec_lst.append(RS(org_ytr[test], pr_val))
    f1_lst.append(F1(org_ytr[test], pr_val))
    auc_lst.append(RAS(org_ytr[test], pr_val))
print('---' * 21)
print("accuracy: {}".format(NPY.mean(acc_lst)),"\nprecision: {}
   ↪ ".format(NPY.mean(prec_lst)),"\nrecall: {}".format(NPY.
   ↪ mean(rec_lst)),"\nf1: {}".format(NPY.mean(f1_lst)))
print('---' * 21)

acc_lst = []
prec_lst = []
rec_lst = []
f1_lst = []
auc_lst = []
knn_sm = KNC()
knn_sm = RSCV(KNC(),  Param_KN, n_iter=4)
for train, test in skf.split(org_Xtr, org_ytr):
    pl = IMP(SMT(sampling_strategy='minority'), knn_sm)
    mdl = pl.fit(org_Xtr[train], org_ytr[train])
    best_est_knn = knn_sm.best_estimator_
    pr_val = best_est_knn.predict(org_Xtr[test])
```

```
    acc_lst.append(pl.score(org_Xtr[test], org_ytr[test]))
    prec_lst.append(PS(org_ytr[test], pr_val))
    rec_lst.append(RS(org_ytr[test], pr_val))
    f1_lst.append(F1(org_ytr[test], pr_val))
    auc_lst.append(RAS(org_ytr[test], pr_val))
print('---' * 21)
print("accuracy: {}".format(NPY.mean(acc_lst)),"\nprecision: {}
    ↪ ".format(NPY.mean(prec_lst)),"\nrecall: {}".format(NPY.
    ↪ mean(rec_lst)),"\nf1: {}".format(NPY.mean(f1_lst)))
print('---' * 21)

y_pred_log = gr_lrc.predict(tst_X)
us_log = AS(tst_y, y_pred_log)
y_pred_tree = gr_dtc.predict(tst_X)
us_tree = AS(tst_y, y_pred_tree)
y_pred_rf = gr_rfc.predict(tst_X)
us_rf = AS(tst_y, y_pred_rf)
y_pred_svc = svc.predict(tst_X)
us_svc = AS(tst_y, y_pred_svc)
y_pred_KNN = gr_knc.predict(tst_X)
us_knn = AS(tst_y, y_pred_KNN)

y_pred_log_sm = best_est.predict(org_X_tst)
os_log = AS(org_y_tst, y_pred_log_sm)
y_pred_tree_sm = best_est_tree.predict(org_X_tst)
os_tree = AS(org_y_tst, y_pred_tree_sm)
y_pred_rf_sm = best_est_rf.predict(org_X_tst)
os_rf = AS(org_y_tst, y_pred_rf_sm)
y_pred_svc_sm = best_est_svc.predict(org_X_tst)
os_svc = AS(org_y_tst, y_pred_svc_sm)
y_pred_knn_sm = best_est_knn.predict(org_X_tst)
os_knn = AS(org_y_tst, y_pred_knn_sm)
```

```
results = {'Technique': ['Random UnderSampling', 'Oversampling
    ↪ (SMOTE)'],
       'LR': [us_log, os_log],
       'DT': [us_tree, os_tree],
       'RF': [us_rf, os_rf],
       'SVM': [us_svc, os_svc],
       'KNN': [us_knn, os_knn],}
us_os_df = PD.DataFrame(data=results)

log = us_os_df['LR']
tree = us_os_df['DT']
rf = us_os_df['RF']
sv = us_os_df['SVM']
knn = us_os_df['KNN']
us_os_df.drop('LR', inplace=True, axis=1)
us_os_df.drop('DT', inplace=True, axis=1)
us_os_df.drop('RF', inplace=True, axis=1)
us_os_df.drop('SVM', inplace=True, axis=1)
us_os_df.drop('KNN', inplace=True, axis=1)
us_os_df.insert(1, 'Logistic Regression', log)
us_os_df.insert(2, 'Decision Tree', tree)
us_os_df.insert(3, 'Random Forest', rf)
us_os_df.insert(4, 'Support Vector Machine', sv)
us_os_df.insert(5, 'K Nearest Neighbour', knn)

us_os_df

def draw_lc(clf1, clf2, clf3, clf4,clf5, X, y, ylim=None, cv=
    ↪ None,n_jobs=1, tr_sizes=NPY.linspace(.1, 1.0, 5)):
     f, (SP1, SP2, SP3, SP4, SP5) = PLT.subplots(5,1, figsize
         ↪ =(15,15))
```

```
    if ylim is not None:
        PLT.ylim(*ylim)

    tr_szs, tr_sc, test_sc = LC(
        clf1, X, y, cv=cv, n_jobs=n_jobs, train_sizes=tr_sizes)
    tsm = NPY.mean(tr_sc, axis=1)
    tss = NPY.std(tr_sc, axis=1)
    testsm = NPY.mean(test_sc, axis=1)
    testss = NPY.std(test_sc, axis=1)
    SP1.fill_between(tr_szs, tsm - tss,
                     tsm + tss, alpha=0.1,
                     color="#E6644E")
    SP1.fill_between(tr_szs, testsm - testss,
                     testsm + testss, alpha=0.1, color="#4E69E6
                        ↪ ")
    SP1.plot(tr_szs, tsm, 'o-', color="#E6644E",
             label="Training␣score")
    SP1.plot(tr_szs, testsm, 'o-', color="#4E69E6",
             label="Cross-validation␣score")
    SP1.set_title("Logistic␣Regression␣Learning␣Curve",
       ↪ fontsize=14)
    SP1.set_xlabel('Training␣size␣(m)')
    SP1.set_ylabel('Score')
    SP1.grid(True)
    SP1.legend(loc="best")


    tr_szs, tr_sc, test_sc = LC(
        clf2, X, y, cv=cv, n_jobs=n_jobs, train_sizes=tr_sizes)
    tsm = NPY.mean(tr_sc, axis=1)
    tss = NPY.std(tr_sc, axis=1)
    testsm = NPY.mean(test_sc, axis=1)
```

```
testss = NPY.std(test_sc, axis=1)
SP2.fill_between(tr_szs, tsm - tss,
                 tsm + tss, alpha=0.1,
                 color="#E6644E")
SP2.fill_between(tr_szs, testsm - testss,
                 testsm + testss, alpha=0.1, color="#4E69E6
                    ↪ ")
SP2.plot(tr_szs, tsm, 'o-', color="#E6644E",
         label="Training␣score")
SP2.plot(tr_szs, testsm, 'o-', color="#4E69E6",
         label="Cross-validation␣score")
SP2.set_title("Decision␣Tree␣Learning␣Curve", fontsize=14)
SP2.set_xlabel('Training␣size␣(m)')
SP2.set_ylabel('Score')
SP2.grid(True)
SP2.legend(loc="best")


tr_szs, tr_sc, test_sc = LC(
    clf3, X, y, cv=cv, n_jobs=n_jobs, train_sizes=tr_sizes)
tsm = NPY.mean(tr_sc, axis=1)
tss = NPY.std(tr_sc, axis=1)
testsm = NPY.mean(test_sc, axis=1)
testss = NPY.std(test_sc, axis=1)
SP3.fill_between(tr_szs, tsm - tss,
                 tsm + tss, alpha=0.1,
                 color="#E6644E")
SP3.fill_between(tr_szs, testsm - testss,
                 testsm + testss, alpha=0.1, color="#4E69E6
                    ↪ ")
SP3.plot(tr_szs, tsm, 'o-', color="#E6644E",
         label="Training␣score")
```

```
SP3.plot(tr_szs, testsm, 'o-', color="#4E69E6",
         label="Cross-validation score")
SP3.set_title("Random Forest Classifier \n Learning Curve",
   ↪  fontsize=14)
SP3.set_xlabel('Training size (m)')
SP3.set_ylabel('Score')
SP3.grid(True)
SP3.legend(loc="best")



tr_szs, tr_sc, test_sc = LC(
    clf4, X, y, cv=cv, n_jobs=n_jobs, train_sizes=tr_sizes)
tsm = NPY.mean(tr_sc, axis=1)
tss = NPY.std(tr_sc, axis=1)
testsm = NPY.mean(test_sc, axis=1)
testss = NPY.std(test_sc, axis=1)
SP4.fill_between(tr_szs, tsm - tss,
                 tsm + tss, alpha=0.1,
                 color="#E6644E")
SP4.fill_between(tr_szs, testsm - testss,
                 testsm + testss, alpha=0.1, color="#4E69E6
                    ↪ ")
SP4.plot(tr_szs, tsm, 'o-', color="#E6644E",
         label="Training score")
SP4.plot(tr_szs, testsm, 'o-', color="#4E69E6",
         label="Cross-validation score")
SP4.set_title("Support Vector Macine Classifier \n Learning
   ↪  Curve", fontsize=14)
SP4.set_xlabel('Training size (m)')
SP4.set_ylabel('Score')
SP4.grid(True)
SP4.legend(loc="best")
```

```
    tr_szs, tr_sc, test_sc = LC(
        clf5, X, y, cv=cv, n_jobs=n_jobs, train_sizes=tr_sizes)
    tsm = NPY.mean(tr_sc, axis=1)
    tss = NPY.std(tr_sc, axis=1)
    testsm = NPY.mean(test_sc, axis=1)
    testss = NPY.std(test_sc, axis=1)
    # SP5.fill_between(tr_szs, tsm - tss,
    #                  tsm + tss, alpha=0.1,
    #                  color="#E6644E")
    # SP5.fill_between(tr_szs, testsm - testss,
    #                  testsm + testss, alpha=0.1, color="#4
       ↪ E69E6")
    SP5.plot(tr_szs, tsm, 'o-', color="#E6644E",
             label="Training␣score")
    SP5.plot(tr_szs, testsm, 'o-', color="#4E69E6",
             label="Cross-validation␣score")
    SP5.set_title("K␣Nearest␣Neighbour␣Classifier␣\n␣Learning␣
       ↪ Curve", fontsize=14)
    SP5.set_xlabel('Training␣size␣(m)')
    SP5.set_ylabel('Score')
    SP5.legend(loc="best")
    return PLT

cv = SS(n_splits=100, test_size=0.2, random_state=42)
draw_lc(gr_lrc, gr_dtc,gr_rfc,gr_svmc, gr_knc, tr_X, tr_y,
   ↪ (0.87, 1.01), cv=cv, n_jobs=4)
us_os_df
```