CREDIT CARD FRAUD DETECTION USING MACHINE LEARNING

A PROJECT REPORT

Submitted by

S ATCHAYA	**621518104005**
N PADMAVATHI	**621518104048**
M SONA	**621518104064**

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

MAHENDRA COLLEGE OF ENGINEERING

MINNAMPALLI, SALEM

ANNA UNIVERSITY : CHENNAI 600 025

JUNE 2022

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"CREDIT CARD FRAUD DETECTION USING MACHINE LEARNING''** is the bonafide work of

ATCHAYA S	**621518104005**
PADMAVATHI N	**621518104048**
SONA M	**621518104064**

who carried out the project under my supervision.

SIGNATURE
Dr.H.LILY BEAULAH,Ph.D.,
HEAD OF THE DEPARTMENT,
Computer Science and Engineering,
Mahendra college of Engineering,
Minnampalli,Salem.

SIGNATURE
Mr.M.JENOLIN REX,M.E.,
SUPERVISOR,
Assistant Professor,
Computer Science and Engineering,
Mahendra college of Engineering,
Minnampalli,Salem.

Submitted to Project and Viva Voce held on ……at……

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and an extremely fortunate to have got this all along the completion of our project work.

We request and thank **Thirumigu M. G. BHARATHKUMAR,** Founder & Chairman, **Shrimathi. VALLIYAMMAL BHARATHKUMAR,** Secretary for their valuable guidance and blessings, also we express our deepest gratitude to the Managing Directors **Er. Ba. MAHENDHIRAN, Er. B .MAHA AJAY PRASATH** who modeled as both technically and morally for achieving great success in life.

We are extremely grateful to **Dr . N . MOHANA SUNDARA RAJU ,** who provided for his constant encouragement, inspiration, presence and blessings throughout our course especially providing with an environment to complete our project successfully.

We also extend our sincere thanks to **Dr. H. LILLY BEAULAH,** Head of the Department of Computer Science and Engineering who provided her valuable suggestions and precious time in accomplishing our project report.

Our profound gratitude to our project guide who have helped us a lot **Mr. M. JENOLIN REX,** Assistant professor who took keen interest on our project work and providing all the necessary information for developing the project successful. We also thank all the technical and non-technical for their help in making this project a successful one.

Lastly, we would like to thank our parents for moral support and our friends with whom we shared our day-to-day experience and received lots of suggestion that improved our quality of work.

ABSTRACT

Credit Card Fraud detection is a challenging task for researchers as fraudsters are innovative, quick-moving individuals. The credit card fraud detection system is challenging as the dataset provided for fraud detection is very imbalanced. In today's economy, credit card (CC) plays a major role. It is an inevitable part of a household, business & global business. While using CCs can of er huge advantages if used cautiously and safely, significant credit & financial damage can be incurred by fraudulent activity. Several methods to deal with the rising credit card fraud (CCF) have been suggested.

In this paper, an ensemble learning-based an intelligent approach for detecting fraud in credit card transactions using XGboost classifier is used to detect credit card fraud, and it is a more regularized form of Gradient Boosting. XGBoost uses advanced regularization (L1&L2), which increases model simplification abilities. Furthermore, XGBoost has an inherent ability to handle missing values. When XGBoost encounters node at lost value, it tries to split left &right hands & learn all ways to the highest loss.

The experiments are conducted on the real time publicly available kaggle dataset with 284,807 credit card transactions included 8 and31 columns. The experimental results show that the proposed scheme provides better accuracy compared with the previous algorithms. Keywords: Credit card fraud detection, XGboost classifier, fraud detection, machine learning.

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

1.1 General

Credit card fraud is a huge pain and comes with huge fees for banks and card provider companies. Financial companies try to prevent account abuse by using individual security responses. The more complex the impact of fraudulent transactions on the transfer of services, rates and popularity of the organization. A range of techniques are used to detect fraud, each attempting to block the maximum penalty for the service while keeping false alarm fees to a minimum. Fraud is the price, and detecting it before a transaction is recorded will significantly lower that price, requiring a very accurate device with very few false alarms. Credit card fraud happens frequently in many different ways. Before the popular of machine learning or deep learning, it is hard for banks and companies to deploy an effective fraud detection system. With the progress in supervise learning, the problem is addressed by transform the fraud problem into a binary classification. The higher risk transaction corresponds to a higher score. In recent years, many researchers have collected financial data from network or banks to build up a fraud detection system. They fed the classical machine learning algorithms with the structural financial data. The goal of these algorithms is to build a supervise learning model based on the dataset without being explicitly programmed. In practical, some classical algorithms like Naïve Bayes , logistic regression and support vector machine are applied for the fraud detection system. This project aims to focus mainly on increasing the accuracy. The algorithm we use is XGBOOST Algorithm. XGBoost, which stands for Extreme Gradient Boosting, is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning library.

1.2 Credit Card Fraud

Credit card fraud, by definition, is the fraudulent use of a credit card done so through the theft of the cardholder's personal details. Thanks to the invention of the internet and the endless supply of eCommerce sites that came with it, credit card scammers now have an easier time than ever pinching your details.

In simpler times – before the internet exploded into society – you might've visualised credit card fraud as a man dressed in all black stealing your card from your wallet. But these days scammers have an arsenal of tricks to swipe from your credit card, and most don't even need your physical card to do it at all. Here are the five most common types of credit card fraud out there.

- **PoS Fraud**

In this type of fraud, small skimming devices are attached to normal Point-of-Sale (PoS) devices to hack your data. These devices scan and store the card information while the customer completes a swipe transaction. Usually, this involves a merchant or store employee who shares these details with malicious actors. Similar attachments may also be fastened on to ATM card slots to clone card information, while a camera is secretly placed over the keypad to capture your PIN.

- **Phishing and vishing**

These involve impersonating official communication from the bank which in turns acts as a bait for you to click on false links. This will usually take you to websites that look authentic. Once you enter your card details on these fake links, fraudsters can access the details and use them for their benefit. Another version is when fraudsters impersonate bank officials on phone calls, asking

you to share an OTP to ‘verify your card’ or ‘avail the reward points’ or ‘extend the validity of your reward points.’

❖ **Keystroke logging**

Today, since most financial transactions are online, hackers have started relying on keystroke logging through malicious software to grab credit card details. This usually begins after you have clicked on a suspicious link and unknowingly installed malware on your system. The software records every key pressed on the system, eventually stealing card details, PIN and more.

❖ **Application fraud**

This is a type of identity theft where fraudulent actors impersonate a genuine customer by using their stolen or counterfeited documents to obtain a credit card. While this might be detected after thorough background checks, if carried out, this will allow criminals to use a valid credit card with a false paper trail. A similar type of fraud involves taking over a valid credit card account by posing as the customer using a similar fake paper trail.

❖ **Theft or loss of card**

If your physical credit card gets stolen or misplaced, there are chances it could be misused. There are precautions you can take in advance as well as safeguards you can put in place even after it is stolen.

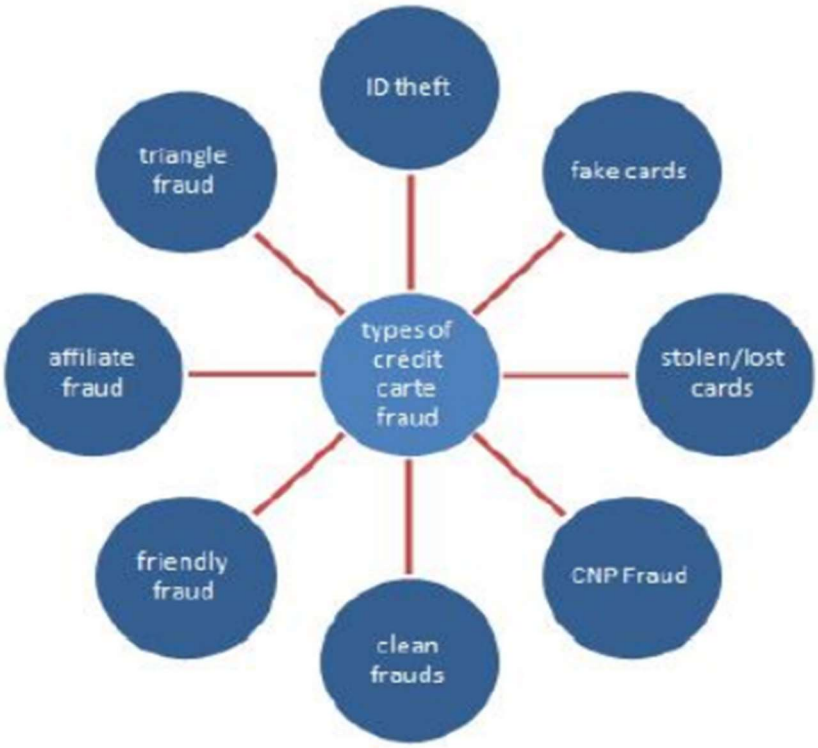


Fig 1.1 Types of credit card fraud

1.2.1 Fraud detection process

The transactions are first checked at the terminal point to be valid or not, which is shown in figure 1.2. At the terminal point, certain essential conditions such as sufficient balance, valid PIN (Personal Identification Number), etc. are validated and the transactions are filtered accordingly.

All the valid transactions are then scored by the predictive model, which then classifies the transactions as genuine or fraudulent. The investigators investigate each fraudulent alert and provide feedback to the predictive model to improve the model's performance. This thesis only deals with the predictive model.

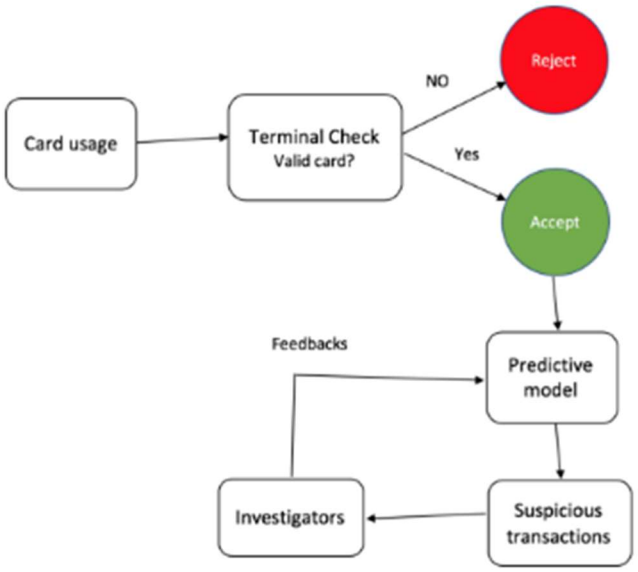


Figure 1.2: Fraud detection process

1.3 Domain Explanation

Before going further into the specific application domain, it is better to get familiar with some of the vital machine learning theories. This chapter will help the reader to understand the proposed fraud detection model using machine learning. More specifically, we will be focussing on the supervised learning, in which the model will be trained with previously labeled data.

1.3.1 Machine Learning

In general context, machine learning can be defined as a field in artificial intelligence that provides the system the capability to learn from the experience automatically without the human intervention and aims to predict the future outcomes as accurate as possible utilizing various algorithmic models.

Machine Learning is very different than the conventional computation approaches, where systems are explicitly programmed to calculate or solve a problem. Machine learning deals with the input data that are used to train a model where the model learns different patterns in the input data and uses that knowledge to predict unknown results. The application of machine learning is incredibly vast. It is used in various applications like the spam filter, weather prediction, stock market prediction, medical diagnosis, fraud detection, autopilot, house price prediction, face detection, and many more.

Typically, machine learning has three categories: supervised, unsupervised and reinforcement learning. This thesis deals with supervised learning and we will discuss it in the next section. For now, we can define supervised learning as the approach where the model is trained with both input and output labels. In contrast, unsupervised learning is where the dataset has input labels, (i.e., a model is trained with unlabeled data), from which it learns different patterns and structures. Usually, it is implemented in applications like visual recognition, robotics, speech recognition and so on. Reinforcement learning deals with learning how to obtain a complex goal by maximizing along a specific dimension step by step, (e.g., maximizing the points won in every round [sky]).

Principal Component Analysis

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process. So to sum up, the idea of PCA is simple — reduce the number of variables of a data set, while preserving as much information as possible.

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the **Principal Components**. It is one of the popular tools that is used for exploratory data analysis and predictive modeling. It is a technique to draw strong patterns from the given dataset by reducing the variances.

PCA generally tries to find the lower-dimensional surface to project the high-dimensional data.

PCA works by considering the variance of each attribute because the high attribute shows the good split between the classes, and hence it reduces the dimensionality. Some real-world applications of PCA are ***image processing, movie recommendation system, optimizing the power allocation in various communication channels.*** It is a feature extraction technique, so it contains the important variables and drops the least important variable.

The PCA algorithm is based on some mathematical concepts such as:

- Variance and Covariance

- Eigenvalues and Eigen factors

Some common terms used in PCA algorithm:

- **Dimensionality:** It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- **Correlation:** It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- **Orthogonal:** It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
- **Eigenvectors:** If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
- **Covariance Matrix:** A matrix containing the covariance between the pair of variables is called the Covariance Matrix.

1.3.2 Supervised Learning

Supervised learning can be defined as a machine learning approach in which both input and output labels are provided to the model to train. The supervised model uses the input and output labeled data for training, and it extracts the patterns from the input data. These extracted patterns are used to support future judgments. Supervised learning can be formally represented as follows:

$$Y = f(x)$$

where x represent the input variables, Y denotes an output variable and f(X) is a mapping function. The goal is to approximate mapping function such that when an unseen input is given to the mapping function, it can predict the output variable (Y) correctly. Furthermore, supervised learning has two sub-categories: classification and regression. In a classification problem, the output variable is a category, (e.g., fraud or genuine, rainy or sunny, etc.). In a regression problem, the output variable is a real value, (e.g., the price of a house, temperature, etc.). This thesis only deals with the classification problem.

1.3.3 Classification

Classification problem in machine learning can be defined as the task of predicting the class label of a given data point. For example, fraud detection can be identified as a classification problem. In this case, the goal is to predict if a given transaction is fraud or genuine. Generally, there are three types of classification: binary classification, where there are two output labels (e.g., classifying a transaction which may be fraud or genuine), multi-class classification, where there are more than two output labels (e.g., classifying a set of images of flowers which may be Rose or Lilly or Sunflower) and multi-label classification, where the data samples are not mutually exclusive and each data samples are assigned a set of target labels (e.g., classifying a crab on the basis of the sex and color in which the output labels can be male/female and red/black). This thesis deals with the binary classification problem where the output label is either normal or fraud.

1.3.4 Class Imbalance Problem

Most real-world applications possess unbalanced class distribution where the number of a class label heavily dominates the count of another class label. One of the best example to explain class imbalance problem is the fraud detection task, where the number of fraud class label is very low as compared to the normal class label. Most machine learning algorithms work poorly in the presence of unbalanced class distribution (i.e., the predictive model tends to classify the minority example as the majority example). So some questions may arise such as: (i) How to tackle the class imbalance problem? (ii) Which machine learning algorithms should be applied in the presence of unbalanced class distribution? (iii) What evaluation metrics should be used to assess the performance of a predictive model when the dataset is highly unbalanced? In the next sections, we will discuss the solutions to these questions.

Handling class imbalance problem

This section explains various approaches that can solve the class imbalance problem. We can roughly classify the approaches into three categories: resampling approach, ensemble-based approach, and cost-sensitive learning approach. This thesis only deals with resampling approach and ensemble-based approach which we will go through them in details in the next sections. Just to give a brief overview, cost-sensitive learning takes misclassification costs into consideration. For example, in medical diagnosis of cancer, the misclassification cost of missing a cancer is much higher than the cost of predicting that a healthy person has cancer. Hence, by considering the

misclassification cost of minority class more heavily than that of majority class, the true positive rate of the model can be improved.

Resampling Approach

Most of the predictive model works worst in the presence of unbalanced class distribution. Therefore, some data preprocessing task has to be performed before providing data as an input to the model. In the case of class imbalance problem, such data preprocessing is performed using a data level approach, which is called resampling approach. Basically, there are three resampling approaches: undersampling, oversampling, and hybrid.

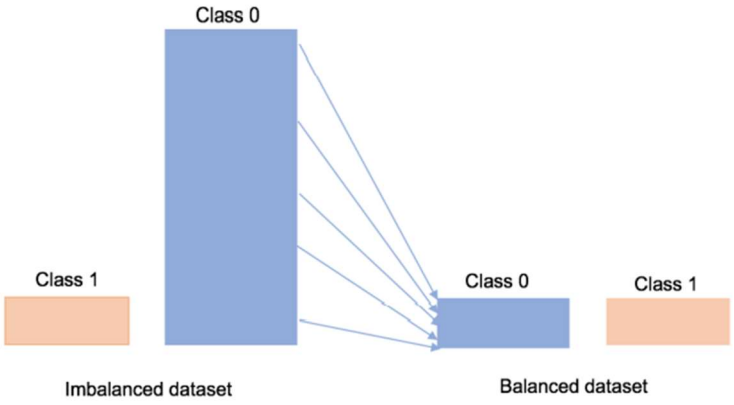


Fig 1.3 Undersampling approach

In the undersampling method, the majority class is reduced in order to make the dataset balanced, which is shown in figure1.3. It is best to implement when the size of the dataset is huge and reducing the majority samples can greatly boost the runtime and reduce the storage troubles.

An oversampling method is exactly opposite to the undersampling method. This method works with the minority class. It replicates the observations from minority class to balance the ratio between majority and

minority sample, which is shown in figure 2.2. At last, a hybrid method applies both undersampling and oversampling method for rebalancing purpose.

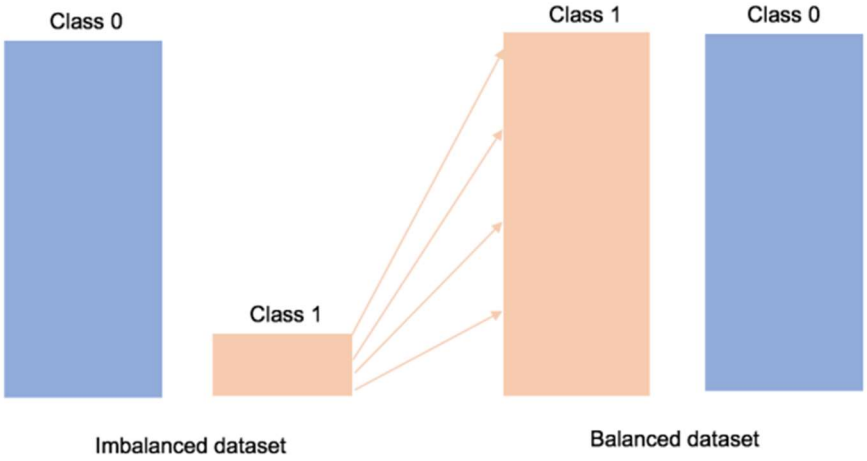
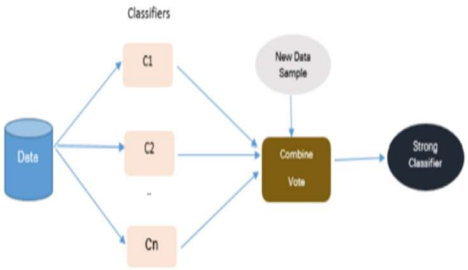


Fig 1.4 Oversampling approach

Ensemble approach

In the previous section, we discussed the data-level approach in which resampling techniques are used to balance the class distributions. In this section, we will discuss the algorithmic approach which is called the ensemble approach. Ensemble approach deals with modifying existing classification algorithms to tackle the unbalanced class distribution. In general, an ensemble approach is a learning algorithm that assembles a set of classifiers and applies them for classification by taking a vote of their predictions , which is also shown in figure. Typically, there are two types of the ensemble approach: bagging and boosting.



1.5 Ensemble approach

Bagging

Bagging which is an abbreviation form of Bootstrap Aggregation is a simple yet very powerful ensemble technique. This method involves bootstrapping that generates new training samples from the original training set with replacement. These new training samples are called bootstrap training samples. Each bootstrap sample is used for training the individual model separately, which are then used for prediction. Finally, the predictions from all the bootstrapped models are aggregated by averaging the output (for regression) or voting (for classification). Figure 2.6 gives a better picture of bagging. It helps to reduce overfitting and variance.

Boosting

Boosting is another very powerful ensemble technique. It involves combining weak learners, which are also called base learners, to create a strong learner that can give a better result as compared to the results generated by an individual learner. Unlike bagging in which each model runs parallelly and then the outputs are combined at the end, the boosting deals with training the weak learners sequentially, such that each learner tries to correct its predecessor by adding more weights to the samples that were previously misclassified. Therefore, the future weak learner will focus more on the misclassified cases.

CHAPTER 2: SYSTEM ANALYSIS

2.1 LITERATURE SURVEY

2.1.1 Difficulties of Credit Card Fraud Detection

Fraud detection systems are prune to several difficulties and challenges enumerated below. An effective fraud detection technique should have abilities to address these difficulties in order to achieve best performance.

● Imbalanced data: The credit card fraud detection data has an imbalanced nature. It means that very small percentages of all credit card transactions are fraudulent. This makes the detection of fraud transactions very difficult and imprecise.

● Different misclassification importance: in fraud detection tasks, different misclassification errors have different importance. Misclassification of a normal transaction as fraud is not as harmful as detecting a fraud transaction as normal. Because in the first case the mistake in classification will be identified in further investigations.

● Overlapping data: many transactions may be considered fraudulent, while actually they are normal (false positive) and reversely, a fraudulent transaction may also seem to be legitimate (false negative). Hence obtaining a low rate of false positives and false negatives is a key challenge of fraud detection systems.

● Lack of adaptability: classification algorithms are usually faced with the problem of detecting new types of normal or fraudulent patterns. The supervised and unsupervised fraud detection systems are inefficient in detecting new patterns of normal and fraud behaviours, respectively.

● Fraud detection cost: The system should take into account both the cost of fraudulent behaviour that is detected and the cost of preventing it. For example, no revenue is obtained by stopping a fraudulent transaction of a few dollars .

● Lack of standard metrics: there is no standard evaluation criterion for assessing and comparing the results of fraud detection systems.

2.1.2 Credit Card Fraud Detection Techniques

The credit card fraud detection techniques are classified in two general categories: fraud analysis (misuse detection) and user behaviour analysis (anomaly detection). The first group of techniques deals with supervised classification tasks at the transaction level. In these methods, transactions are labelled as fraudulent or normal based on previous historical data. This dataset is then used to create classification models which can predict the state (normal or fraud) of new records. There are numerous model creation methods for a typical two class classification task such as rule induction , decision trees and neural networks .This approach is proven to reliably detect most fraud tricks which have been observed before, also known as misuse detection. The second approach deals with unsupervised methodologies which are based on account behaviour. In this method a transaction is detected as fraudulent if it is in contrast with 19 the user's normal behaviour. This is because we don't expect fraudsters behave the same as the account owner or be aware of the behaviour model of the owner .To this aim, we need to extract the legitimate user behavioural model (e.. user profile)for each account and then detect fraudulent activities according to it.

Comparing New behaviours with this model, different enough activities are distinguished as frauds. The profiles may contain the activity information of the account; such as merchant types, amount, location and time of transactions, .This method is also known as anomaly detection. It is important to highlight the key differences between user behaviour analysis and fraud analysis approaches. The Fraud analysis method can detect known fraud tricks, with a low false positive rate.

These systems extract the signature and model of fraud tricks presented in oracle dataset and can then easily determine exactly which frauds, the system is currently experiencing. If the test data does not contain any fraud signatures, no alarm is raised. Thus, the false positive rate can be reduced extremely. However, since learning of a fraud analysis system (i.e. classifier) is based on limited and specific fraud records, It cannot detect novel frauds. As a result, the false negative rate may be extremely high depending on how ingenious the fraudsters. User behaviour analysis, on the other hand, greatly addresses the problem of detecting novel frauds. These Methods do not search for specific fraud patterns, but rather compare incoming activities with the constructed model of legitimate user behaviour. Any activity that is sufficiently different from the model will be considered as a possible fraud. Though user behaviour analysis approaches are powerful in detecting innovative frauds, they really suffer from high rates of false alarm. Moreover, if a fraud occurs during the training phase, this fraudulent behaviour will be entered in baseline mode and is assumed to be normal in further analysis.In this section we will briefly introduce some current fraud detection techniques which are applied to credit card fraud detection tasks.

Artificial Neural Network

An artificial neural network (ANN) is a set of interconnected nodes designed to imitate the functioning of the human brain . Each node has a weighted connection to several other nodes in adjacent layers. Individual nodes take the input received from connected nodes and use the weights 6 together with a simple function to compute output values. Neural networks come in many shapes and architectures. The Neural network architecture, including the number of hidden layers, the number of nodes within a specific hidden layer and their connectivity, must be specified by the user based on the complexity of the problem. ANNs can be configured by supervised, unsupervised or hybrid learning methods.

Supervised techniques

In supervised learning, samples of both fraudulent and non-fraudulent records, associated with their labels are used to create models. These techniques are often used in fraud analysis approach. One of the most popular supervised neural networks is back propagation network (BPN). It minimizes the objective function using a multi-stage dynamic optimization method that is a generalization of the delta rule. The back propagation method is often useful for feed-forward network with no feedback. The BPN algorithm is usually time-consuming and parameters like the number of hidden neurons and learning rate of delta rules require extensive tuning and training to achieve the best performance . In the domain of fraud detection, supervised neural networks like back-propagation are known as efficient tools that have numerous applications . Raghavendra Patidar, used a dataset to train a three layers backpropagation neural network in combination with genetic algorithms (GA)for credit card fraud detection. In this work, genetic algorithms were

responsible for making decisions about the network architecture, dealing with the network topology, number of hidden layers and number of nodes in each layer. Also, Aleskerovet developed a neural network based data mining system for credit card fraud detection.The proposed system (CARDWATCH) had three layers of autoassociative architectures.They used a set of synthetic data for training and testing the system. The reported results show very successful fraud detection rates. In a P-RCE neural network was applied for credit card fraud detection.P-RCE is a type of radial-basis function networks that usually applied for pattern recognition tasks. Kroenke Et al. proposed a model for real time fraud detection based on bidirectional neural networks. They used a large data set of cell phone transactions provided by a credit card company. It was claimed that the system outperforms the rule based algorithms in terms of false positive rate. Again in a parallel granular neural network (GNN) is proposed to speed up data mining and knowledge discovery process for credit card fraud detection. GEN is a kind of fuzzy neural network based on knowledge discovery (FNNKD). The underlying dataset was extracted from SQL server database containing sample Visa Card transactions and then pre-processed for applying in fraud detection. They obtained less average training errors in the presence of larger training dataset.

Unsupervised Techniques

The unsupervised techniques do not need the previous knowledge of fraudulent and normal records. These methods raise alarm for those transactions that are most dissimilar from the normal ones. These techniques are often used in user behaviour approach. ANNs can produce acceptable results for enough large transaction dataset. They need a long training dataset. Self Organizing map (SOM) is one of the most popular unsupervised neural

networks learning which was introduced. SOM provides a clustering method, which is appropriate for constructing and analysing customer profiles, in credit card fraud detection, as suggested. SOM operates in two phases: training and mapping. In the former phase, the map is built and weights of the neurons are 7 updated iteratively, based on input samples , in latter, test data is classified automatically into normal and fraudulent classes through the procedure of mapping. As stated , after training the SOM, new unseen transactions are compared to normal and fraud clusters, if it is similar to all normal records, it is classified as normal. New fraud transactions are also detected similarly.

One of the advantages of using unsupervised neural networks over similar techniques is that these methods can learn from data streams. The more data passed to a SOM model, the more adaptation and improvement on result is obtained. More specifically, the SOM adapts its model as time passes. Therefore it can be used and updated online in banks or other financial corporations. As a result, the fraudulent use of a card can be detected fast and effectively. However, neural networks have some drawbacks and difficulties which are mainly related to specifying suitable architecture on one hand and excessive training required for reaching the best performance on the other hand.

Hybrid supervised and unsupervised techniques

In addition to supervised and unsupervised learning models of neural networks, some researchers have applied hybrid models. John ZhongLeiet.Al proposed hybrid supervised (SICLN) and unsupervised (ICLN)learning networks for credit card fraud detection. They improved the reward only rule of SICLN model to ICLN in order to update weights according to both reward and penalty. This improvement appeared in terms of increasing stability and reducing the training time. Moreover, the number of final clusters of the ICLN

is independent from the number of initial network neurons. As a result the inoperable neurons can be omitted from the clusters by applying the penalty rule. The results indicated that both the ICLN and the SICLN have high performance, but the SICLN outperforms well-known unsupervised clustering algorithms.

2.2 Analysis Approach

2.2.1 Existing System

The existing system uses a deep neural network based algorithm for credit card detection. The deep neural network and deep learning has arose a revolution in many artificial intelligence areas including computer vision , natural language processing , speech recognition . There are many factors contribute to the success of deep neural network. The first factor is the rationality of the algorithm, that is simulation of brain. The second factor is the ever-increasing data of many areas generating from cameras, microphones, mobile applications and etc. The final factor is the rapid increasing processor like Gpu cards, Tpu Cards and FPGA.

The training process of the existing model are accelerated by Gpu card. In practical, Gpu card is necessary when training a deep neural network. The gpu card for training is Nvidia P100, 182 Authorized licensed use limited to: University of Brighton. Downloaded on October 11,2020 at 07:02:19 UTC from IEEE Xplore. Restrictions apply. which is a well- known graphic card. To help train model faster, they use Nadam optimizer, which is an adaptive optimizer so that the model will converge faster than SGD optimizer. The super parameters of model would influence the final result. The learning rate, epoch, batch size should be set carefully. The Neural network model outperforms the other models with accuracy of 95%.

2.2.2 Proposed System

First the credit card dataset is taken from the supply, and cleaning and approval is executed on the dataset which joins disposal of excess, filling void territories in sections, changing imperative variable into components or exercises then actualities is part into 2 sections, one is preparing dataset and another is check data set. The main idea of boosting is to combine a series of weak classifiers with low accuracy to build a strong classifier with better classification performance. If the weak learner for each step is based on the gradient direction of the loss function, it can be called the Gradient Boosting Machines. XGBoost is optimized under the Gradient Boosting framework and developed by Chen and Guestrin , which is designed to be highly efficient and portable. Our proposed system provides the accuracy of 99%.

2.2.3 Advantages Of Existing System

- Increase in accuracy than existing system
- Less feature engineering required (No need for scaling, normalizing data, can also handle missing values well)
- XGBoost has in-built L1 (Lasso Regression) and L2 (Ridge Regression) regularization which prevents the model from overfitting.

CHAPTER 3: SOFTWARE SPECIFICATIONS

3.1 Libraries

Libraries used:

- ❖ Pandas
- ❖ Seaborn
- ❖ Matplotlib
- ❖ Sklearn
- ❖ Numpy
- ❖ XGBoost

3.1.1 Pandas

Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays. As one of the most popular data wrangling packages, Pandas works well with many other data science modules inside the Python ecosystem, and is typically included in every Python distribution, from those that come with your operating system to commercial vendor distributions like ActiveState's ActivePython.

Pandas makes it simple to do many of the time consuming, repetitive tasks associated with working with data, including:

- Data cleansing
- Data fill
- Data normalization
- Merges and joins
- Data visualization
- Statistical analysis

- Data inspection
- Loading and saving data
- And much more

In fact, with Pandas, you can do everything that makes world-leading data scientists vote Pandas as the best data analysis and manipulation tool available.

How to install Pandas?

You can install Pandas by using the following commands:

```
#To install pandas in terminal or command line use one of the commands
pip install pandas
# or
conda install pandas
# To install pandas in jupyter notebook use this command
!pip install pandas
```

How to import Pandas?

```
import pandas as pd
```

By using the above command you can easily import pandas library.

Pandas Data Structures

Pandas deals with three types of data structures :

- Series

- DataFrame
- Panel

a)Series is a one-dimensional array-like structure with homogeneous data. The size of the series is immutable(cannot be changed) but its values are mutable.

b) DataFrame is a two-dimensional array-like structure with heterogeneous data. Data is aligned in a tabular manner(Rows &Columns form). The size and values of DataFrame are mutable

c) The panel is a three-dimensional data structure with heterogeneous data. It is hard to represent the panel in graphical representation. But it can be illustrated as a container of DataFrame. The size and values of a Panel are mutable.

Creating Series and Data Frames

Series :

pandas.Series(data=None , index=None , dtype=None,copy = false)

parameters:

data: data takes various forms like ndarray, list, constants

index: index values must be unique and hashable, the same length as data. Default np.arrange(n) if no index is passed.

dtype: dtype is for data type. If None, the data type will be inferred

copy: Copy input data. default false

3.1.2 Seaborn

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on the top of matplotlib library and also closely integrated to the data structures from pandas. Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs, so that we can switch between different visual representations for same variables for better understanding of dataset.

Different categories of plot in Seaborn

Plots are basically used for visualizing the relationship between variables. Those variables can be either be completely numerical or a category like a group, class or division. Seaborn divides plot into the below categories:

- **Relational plots:** This plot is used to understand the relation between two variables.
- **Categorical plots:** This plot deals with categorical variables and how they can be visualized.
- **Distribution plots:** This plot is used for examining univariate and bivariate distributions
- **Regression plots:** The regression plots in seaborn are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses.

- **Matrix plots:** A matrix plot is an array of scatterplots.
- **Multi-plot grids:** It is an useful approach is to draw multiple instances of the same plot on different subsets of the dataset.

Installation

For python environment :

pip install seaborn

For conda environment :

conda install seaborn

Dependencies

- Python 3.6+
- numpy (>= 1.13.3)
- scipy (>= 1.0.1)
- pandas (>= 0.22.0)
- matplotlib (>= 2.1.2)
- statsmodel (>= 0.8.0)

Some basic plots using seaborn

Dist plot : Seaborn dist plot is used to plot a histogram, with some other variations like kdeplot and rugplot.

Line plot : The line plot is one of the most basic plot in seaborn library. This plot is mainly used to visualize the data in form of some time series, i.e. in continuous manner.

Lmplot : The lmplot is another most basic plot. It shows a line representing a linear regression model along with data points on the 2D-space and x and y can be set as the horizontal and vertical labels respectively.

3.1.3 Matplotlib

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

Installation :

Windows, Linux and macOS distributions have matplotlib and most of its dependencies as wheel packages.

Run the following command to install matplotlib package :

python -mpip install -U matplotlib

Importing matplotlib :

from matplotlib import pyplot as plt

or

import matplotlib.pyplot as plt

Matplotlib is a python library used to create 2D graphs and plots by using python scripts. It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc. It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.

Now let's check different categories of plots that **Matplotlib** provides.

- Line plot
- Histogram
- Bar Chart
- Scatter plot
- Pie charts
- Boxplot

- **Line Plots**

A line plot is used to see the relationship between the x and y-axis.

The plot() function in the **Matplotlib** library's Pyplot module is used to create a 2D hexagonal plot of the coordinates x and y. plot() will take various arguments like **plot(x, y, scalex, scaley, data, **kwargs)**.

x, y are the coordinates of the horizontal and vertical axis where x values are optional and its default value is range(len(y)).

scalex, scaley parameters are used to auto scale the x-axis or y-axis, and its default value is true.

****kwargs** is used to specify the property like line label, linewidth, marker, color, etc.

- **Histogram-**

The most common graph for displaying frequency distributions is a histogram. To create a histogram the first step is to create a bin of ranges, then distribute the whole range of value into series of intervals, and count the value which will fall in the given interval. we can use **plt.hist()** function for plotting the histograms which will take various arguments like data, bins, color, etc.

x: x-coordinate or sequence of the array

bins: integer value for the number of bins wanted in the graph

range: the lower and upper range of bins

density: optional parameter that contains boolean values

histtype: optional parameter used to create different types of histograms like:- bar, bar stacked, step, step filled and the default is a bar

- **Bar Plot**

Mainly bar plot is used to show the relationship between the numeric and categoric values. In a bar chart, we have one axis representing a particular category of the columns and another axis representing the values or count of

the specific category. Bar charts are plotted both vertically and horizontally and are plotted using the following line of code:

plt.bar(x,height,width,bottom,align)

x: representing the coordinates of the x-axis

height: the height of the bars

width: width of the bars. Its default value is 0.8

bottom: It's optional. It is a y-coordinate of the bar its default value is None

align: center, edge its default value is center

- **Scatter Plot**

Scatter plots are used to show the relationships between the variables and use the dots for the plotting or it used to show the relationship between two numeric variables.

The **scatter()** method in the **Matplotlib** library is used for plotting.

- **Pie Chart-**

A pie chart (or circular chart) is used to show the percentage of the whole. Hence it is used when we want to compare the individual categories with the whole. **Pie()** will take the different parameters such as:

x: Sequence of an array

labels: List of strings which will be the name of each slice in the pie chart

Autopct: It is used to label the wedges with numeric values. The labels will be placed inside the wedges. Its format is %1.2f%.

- **Box Plot-**

A Box plot is used to show the summary of the whole dataset or all the numeric values in the dataset. The summary contains minimum, first quartile, median, third quartile, and maximum. Also, the median is present between the first and third quartile. Here x-axis contains the data values and y coordinates show the frequency distribution.

Parameters used in box plots are as follows:

data: NumPy array

vert: It will take boolean values i.e true or false for the vertical and horizontal plot default is True

width: This will take array and sets of the width of boxes, Optional parameters

Patch_artist: It is used to fill the boxes with color and its default value is false

labels: Array of strings which is used to set the labels of the dataset

- **Area Chart-**

Area chart or area plot is used to visualize the quantitative data graphically it is based on the line plot. **fill_between()** function is used to plot the area chart.

Parameter:

x,y represent the x and y coordinates of the plot. This will take an array of length n.

Interpolate is a boolean value and is optional. If true, interpolate between the two lines to find the precise point of intersection.

****kwargs**: alpha, color, facecolor, edgecolur, linewidth.

- **Word Cloud-**

Wordcloud is the visual representation of the text data. Words are usually single, and the importance of each word is shown with the font size or color. The **wordcloud()** function is used to create word cloud in python.

The **wordcloud()** will take various arguments like:

width: set the width of the canvas .default 400

height: set the height of the canvas .default 400

max_words: number of words allowed, its default value is 200.

background_color: background color for the word-cloud image.The default color is black.

Once the word cloud object is created, you can call the generate function to generate the word cloud and pass the text data.

3.1.4 Sklearn

Scikit-learn is a Python module for machine learning built on top of SciPy and is distributed under the 3-Clause BSD license.

The project was started in 2007 by David Cournapeau as a Google Summer of Code project, and since then many volunteers have contributed. Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib!

The functionality that scikit-learn provides include:

- **Regression**, including Linear and Logistic Regression
- **Classification**, including K-Nearest Neighbors
- **Clustering**, including K-Means and K-Means++
- **Model selection**
- **Preprocessing**, including Min-Max Normalization

Dependencies

scikit-learn requires:

- Python (>= 3.8)
- NumPy (>= 1.17.3)
- SciPy (>= 1.3.2)
- joblib (>= 1.0.0)
- threadpoolctl (>= 2.0.0)

Installation

If you already have a working installation of numpy and scipy, the easiest way to install scikit-learn is using `pip`:

- `pip install -U scikit-learn`

or `conda`:

- `conda install -c conda-forge scikit-learn`

3.1.5 Numpy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be

defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Installation:

- **Mac** and **Linux** users can install NumPy via pip command:

pip install numpy

- **Windows** does not have any package manager analogous to that in linux or mac.

1. Arrays in NumPy: NumPy's main object is the homogeneous multidimensional array.

- It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
- In NumPy dimensions are called *axes*. The number of axes is *rank*.
- NumPy's array class is called **ndarray**. It is also known by the alias **array**.

2. Array creation: There are various ways to create arrays in NumPy.

- For example, you can create an array from a regular Python **list** or **tuple** using the **array** function. The type of the resulting array is deduced from the type of the elements in the sequences.
- Often, the elements of an array are originally unknown, but its size is known. Hence, NumPy offers several functions to create arrays with **initial placeholder content**. These minimize the necessity of growing arrays, an expensive operation. **For example:** np.zeros, np.ones, np.full, np.empty, etc.

- To create sequences of numbers, NumPy provides a function analogous to range that returns arrays instead of lists.
- **arange:** returns evenly spaced values within a given interval. **step** size is specified.
- **linspace:** returns evenly spaced values within a given interval. **num** no. of elements are returned.
- **Reshaping array:** We can use **reshape** method to reshape an array. Consider an array with shape (a1, a2, a3, …, aN). We can reshape and convert it into another array with shape (b1, b2, b3, …, bM). The only required condition is: a1 x a2 x a3 … x aN = b1 x b2 x b3 … x bM . (i.e original size of array remains unchanged.)
- **Flatten array:** We can use **flatten** method to get a copy of array collapsed into **one dimension**. It accepts *order* argument. Default value is 'C' (for row-major order). Use 'F' for column major order.

Note: Type of array can be explicitly defined while creating array.

3. Array Indexing: Knowing the basics of array indexing is important for analysing and manipulating the array object. NumPy offers many ways to do array indexing.

- **Slicing:** Just like lists in python, NumPy arrays can be sliced. As arrays can be multidimensional, you need to specify a slice for each dimension of the array.
- **Integer array indexing:** In this method, lists are passed for indexing for each dimension. One to one mapping of corresponding elements is done to construct a new arbitrary array.
- **Boolean array indexing:** This method is used when we want to pick elements from array which satisfy some condition.

4. Basic operations: Plethora of built-in arithmetic functions are provided in NumPy.

- **Operations on single array:** We can use overloaded arithmetic operators to do element-wise operation on array to create a new array. In case of +=, -=, *= operators, the existing array is modified.
- **Unary operators:** Many unary operations are provided as a method of **ndarray** class. This includes sum, min, max, etc. These functions can also be applied row-wise or column-wise by setting an axis parameter.
- **Binary operators:** These operations apply on array elementwise and a new array is created. You can use all basic arithmetic operators like +, -, /, , *etc. In case of* +=, -=, = operators, the existing array is modified.
- **Universal functions (ufunc):** NumPy provides familiar mathematical functions such as sin, cos, exp, etc. These functions also operate elementwise on an array, producing an array as output.
- **4. Sorting array:** There is a simple **np.sort** method for sorting NumPy arrays.

Advanced methods available in NumPy

1.Stacking: Several arrays can be stacked together along different axes.

- **np.vstack:** To stack arrays along vertical axis.
- **np.hstack:** To stack arrays along horizontal axis.
- **np.column_stack:** To stack 1-D arrays as columns into 2-D arrays.
- **np.concatenate:** To stack arrays along specified axis (axis is passed as argument).

2.Splitting: For splitting, we have these functions:

- **np.hsplit:** Split array along horizontal axis.
- **np.vsplit:** Split array along vertical axis.
- **np.array_split:** Split array along specified axis.

3.Broadcasting: The term broadcasting describes how NumPy treats arrays with different shapes during arithmetic operations. Subject to certain constraints, the smaller array is "broadcast" across the larger array so that they have compatible shapes.

Broadcasting provides a means of vectorizing array operations so that looping occurs in C instead of Python. It does this without making needless copies of data and usually leads to efficient algorithm implementations. There are also cases where broadcasting is a bad idea because it leads to inefficient use of memory that slows computation.

NumPy operations are usually done element-by-element which requires two arrays to have exactly the same shape. Numpy's broadcasting rule relaxes this constraint when the arrays' shapes meet certain constraints.

The Broadcasting Rule: In order to broadcast, the size of the trailing axes for both arrays in an operation must either be the same size or one of them must be **one**.

4.Working with datetime: Numpy has core array data types which natively support datetime functionality. The data type is called "datetime64", so named because "datetime" is already taken by the datetime library included in Python.

5.Linear algebra in NumPy: The Linear Algebra module of NumPy offers various methods to apply linear algebra on any numpy array. You can find:

- rank, determinant, trace, etc. of an array.
- eigen values of matrices
- matrix and vector products (dot, inner, outer,etc. product), matrix exponentiation
- solve linear or tensor equations and much more!

Indexing using index arrays

Indexing can be done in numpy by using an array as an index. In case of slice, a view or shallow copy of the array is returned but in index array a copy of the original array is returned. Numpy arrays can be indexed with other arrays or any other sequence with the exception of tuples. The last element is indexed by -1 second last by -2 and so on.

NumPy is a widely used general purpose library which is at the core of many other computation libraries like scipy, scikit-learn, tensorflow, matplotlib, opencv, etc. Having a basic understanding of NumPy helps in dealing with other higher level libraries efficiently!

3.2 Algorithm

3.2.1 XGBoost

XGBoost is the abbreviation form of eXtreme Gradient Boosting. It is an advanced implementation of gradient boosting proposed by Chen and Guestrin . Since XGBoost improves on gradient boosting algorithm, we will briefly explain it. Gradient boosting is also a boosting algorithm, which tries to combine the weak learners to form a strong learner. It generates weak learners during the learning process. At each level of the process, the weak learner

predicts the values or class label and then calculates the loss, (i.e., the difference between real value and the predicted value). Depending upon the loss, it creates a new weak learner and then the weak learner trains on the remaining errors. This process continues until a certain threshold. This process is called gradient descent optimization problem and therefore this algorithm is called gradient boosting. It is another way for giving high preference to the misclassified samples whereas in boosting algorithm like Ada boost, the weights of the misclassified samples are given higher values, so that next weak learner works on it.

It uses decision trees as the weak learners and it has many advantages over standard gradient boosting algorithm. XGBoost has better regularization than gradient boosting. Therefore, it reduces overfitting. XGboost allows parallel processing, so it is much faster than standard gradient boosting. XGBoost has the inbuilt capability to handle missing data. Gradient boosting is a greedy algorithm since it stops splitting the node as soon as it encounters a negative loss in the split whereas xgboost splits up to the maximum depth specified. XGBoost has built-in cross-validation feature, so it is easier to determine the number of boosting rounds at each run. There are quite a few hyperparameters that need to be tuned in order to get the best result from xgboost algorithm.

XGBoost is a software library that you can download and install on your machine, then access from a variety of interfaces. Specifically, XGBoost supports the following main interfaces:

- Command Line Interface (CLI).
- C++ (the language in which the library is written).
- Python interface as well as a model in scikit-learn.
- R interface as well as a model in the caret package.
- Julia.

- Java and JVM languages like Scala and platforms like Hadoop.

3.2.2 XGBoost Features

The library is laser focused on computational speed and model performance, as such there are few frills. Nevertheless, it does offer a number of advanced features.

Model Features

The implementation of the model supports the features of the scikit-learn and R implementations, with new additions like regularization. Three main forms of gradient boosting are supported:

- **Gradient Boosting** algorithm also called gradient boosting machine including the learning rate.
- **Stochastic Gradient Boosting** with sub-sampling at the row, column and column per split levels.
- **Regularized Gradient Boosting** with both L1 and L2 regularization.

Gradient Boosting

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.
There is a technique called the Gradient Boosted Trees whose base learner is CART (Classification and Regression Trees).

Gradient Boosting is an *iterative functional gradient algorithm*, i.e an algorithm which minimizes a loss function by iteratively choosing a function that points towards the negative gradient; a weak hypothesis.

System Features

The library provides a system for use in a range of computing environments, not least:

- **Parallelization** of tree construction using all of your CPU cores during training.
- **Distributed Computing** for training very large models using a cluster of machines.
- **Out-of-Core Computing** for very large datasets that don't fit into memory.
- **Cache Optimization** of data structures and algorithm to make best use of hardware.

Algorithm Features

The implementation of the algorithm was engineered for efficiency of compute time and memory resources. A design goal was to make the best use of available resources to train the model. Some key algorithm implementation features include:

- **Sparse Aware** implementation with automatic handling of missing data values.
- **Block Structure** to support the parallelization of tree construction.
- **Continued Training** so that you can further boost an already fitted model on new data.

XGBoost is free open source software available for use under the permissive Apache-2 license.

3.2.3 XGBoost architecture

XGBoost stands for Extreme Gradient Boosting. It's a parallelized and carefully optimized version of the gradient boosting algorithm. Parallelizing the whole boosting process hugely improves the training time.

Instead of training the best possible model on the data (like in traditional methods), we train thousands of models on various subsets of the training dataset and then vote for the best-performing model.

For many cases, XGBoost is better than usual gradient boosting algorithms. The Python implementation gives access to a vast number of inner parameters to tweak for better precision and accuracy.

Some important features of XGBoost are:

- ***Parallelization:*** The model is implemented to train with multiple CPU cores.
- ***Regularization:*** XGBoost includes different regularization penalties to avoid overfitting. Penalty regularizations produce successful training so the model can generalize adequately.
- ***Non-linearity:*** XGBoost can detect and learn from non-linear data patterns.
- ***Cross-validation:*** Built-in and comes out-of-the-box.
- ***Scalability:*** XGBoost can run distributed thanks to distributed servers and clusters like Hadoop and Spark, so you can process enormous amounts of data. It's also available for many programming languages like C++, JAVA, Python, and Julia.

How does the XGBoost algorithm work?

- Consider a function or estimate . To start, we build a sequence derived from the function gradients. The equation below models a particular form of gradient descent. The represents the Loss function to minimize hence it gives the direction in which the function decreases. is the rate of change fitted to the loss function, it's equivalent to the learning rate in gradient descent. is expected to approximate the behaviour of the loss suitably.

$$F_{x_t+1} = F_{x_t} + \epsilon_{x_t} \frac{\partial F}{\partial x}(x_t)$$

- To iterate over the model and find the optimal definition we need to express the whole formula as a sequence and find an effective function that will converge to the minimum of the function. This function will serve as an error measure to help us decrease the loss and keep the performance over time. The sequence converges to the minimum of the function . This particular notation defines the error function that applies when evaluating a gradient boosting regressor.

$$f(x, \theta) = \sum l(F((X_i, \theta), y_i))$$

Other Gradient Boosting methods

Gradient Boosting Machine (GBM)

GBM combines predictions from multiple decision trees, and all the weak learners are decision trees. The key idea with this algorithm is that every node

of those trees takes a different subset of features to select the best split. As it's a Boosting algorithm, each new tree learns from the errors made in the previous ones.

Light Gradient Boosting Machine (LightGBM)

LightGBM can handle huge amounts of data. It's one of the fastest algorithms for both training and prediction. It generalizes well, meaning that it can be used to solve similar problems. It scales well to large numbers of cores and has an open-source code so you can use it in your projects for free.

Categorical Boosting (CatBoost)

This particular set of Gradient Boosting variants has specific abilities to handle categorical variables and data in general. The CatBoost object can handle categorical variables or numeric variables, as well as datasets with mixed types. That's not all. It can also use unlabelled examples and explore the effect of kernel size on speed during training.

CHAPTER 4: MODULE DESCRIPTION

Modules

- Importing necessary libraries
- Importing data
- Data processing and understanding
- Train and split
- Handling Class imbalance with SMOTE
- Model building

4.1 Importing Necessary Libraries

It is a good practice to import all the necessary libraries in one place — so that we can modify them quickly. For this credit card data, the features that we have in the dataset are the transformed version of PCA, so we will not need to perform the feature selection again. Python scikit-learn module , python pandas module , python matplotlib module are imported.

4.2 Importing Data

This dataset contains the real bank transactions made by European cardholders in the year 2013. As a security concern, the actual variables are not being shared but — they have been transformed versions of PCA. As a result, we can find 29 feature columns and 1 final class column. Importing the dataset is pretty much simple. You can use pandas module in python to import it. The dataset is IEEE CIS dataset which can be downloaded on Kaggle platform. This dataset is a well-known dataset for transaction fraud detection.

4.3 Data Processing And Understanding

The dataset contains an European cardholder transactions that occurred in two days with 284,807 transactions from September 2013. The dataset was obtained from Kaggle. It contains 28 attributes, which have been scaled and modified. However their description has not been given. The only attributes known to us are

1.Amount

2.Time

3.Output class[0 for a normal transaction and 1 for a fraudulent transaction]

The total dataset has 284807 rows and 31 columns. Out of these, the normal transactions were 284315 and fraudulent transactions were 492.

Class Wise Analysis

4.4 Train And Split

- Removal of the “Time” attribute since it has no contribution towards the prediction of the class.

- Division of train and test data in the existing dataset , with 80% training data and 20% testing data.

- We have to split the data into two different data set . One dataset (Train data)will be used for training our model and the data which is unseen will be used for testing.

4.5 Handling Class Imbalance With Smote

Most of the predictive model works worst in the presence of unbalanced class distribution. Therefore, some data preprocessing task has to be performed before providing data as an input to the model. In the case of class imbalance problem, we are using SMOTE- Synthetic Minority Oversampling Technique.

The method was proposed in a 2002 paper in the Journal of Artificial Intelligence Research. SMOTE is an improved method of dealing with imbalanced data in classification problems.

The component works by generating new instances from existing minority cases that you supply as input. This implementation of SMOTE does *not* change the number of majority cases.

The new instances are not just copies of existing minority cases. Instead, the algorithm takes samples of the *feature space* for each target class and its nearest neighbors. The algorithm then generates new examples that combine features of the target case with features of its neighbors. This approach increases the features available to each class and makes the samples more general.

SMOTE takes the entire dataset as an input, but it increases the percentage of only the minority cases.

4.6 Model Building

After training our data, We can make predictions using the fit model on the test dataset. To make predictions we use the scikit-learn function model.predict(). By default, the predictions made by XGBoost are probabilities. We can evaluate the predictions by finding accuracy.

There are a variety of measures for various algorithms and these measures have been developed to evaluate very different things. So there should be criteria for evaluation of various proposed methods. False Positive (FP), False Negative (FN), True Positive (TP), and True Negative (TN) and the relation between them are quantities which are usually adopted by credit card fraud detection researchers to compare the accuracy of different approaches. The definitions of mentioned parameters are presented below:

- FP: the false positive rate indicates the portion of the non-fraudulent transactions wrongly being classified as fraudulent transactions.

- FN: the false negative rate indicates the portion of the fraudulent transactions wrongly being classified as normal transactions.

● TP: the true positive rate represents the portion of the fraudulent transactions correctly being classified as fraudulent transactions.

● TN: the true negative rate represents the portion of the normal transactions correctly being classified as normal transactions.

Accuracy (ACC)/Detection rate

Accuracy is the percentage of correctly classified instances. It is one the most widely used classification performance metrics.

Formula: TN + TP/TP + FP + FN + TN

CHAPTER 5: SYSTEM DESIGN

5.1 Sequence Diagram

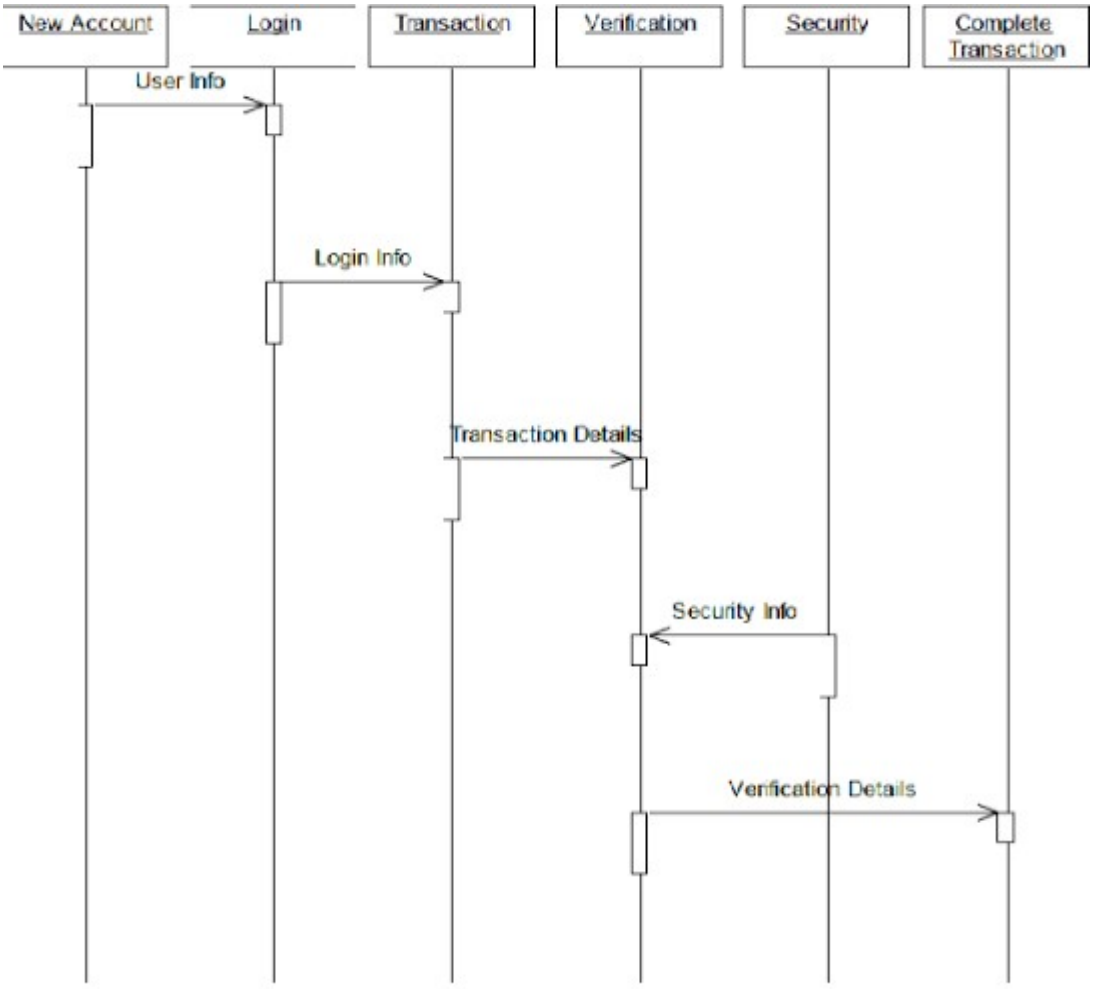


Fig 5.1 Sequence Diagram

5.2 Class Diagram

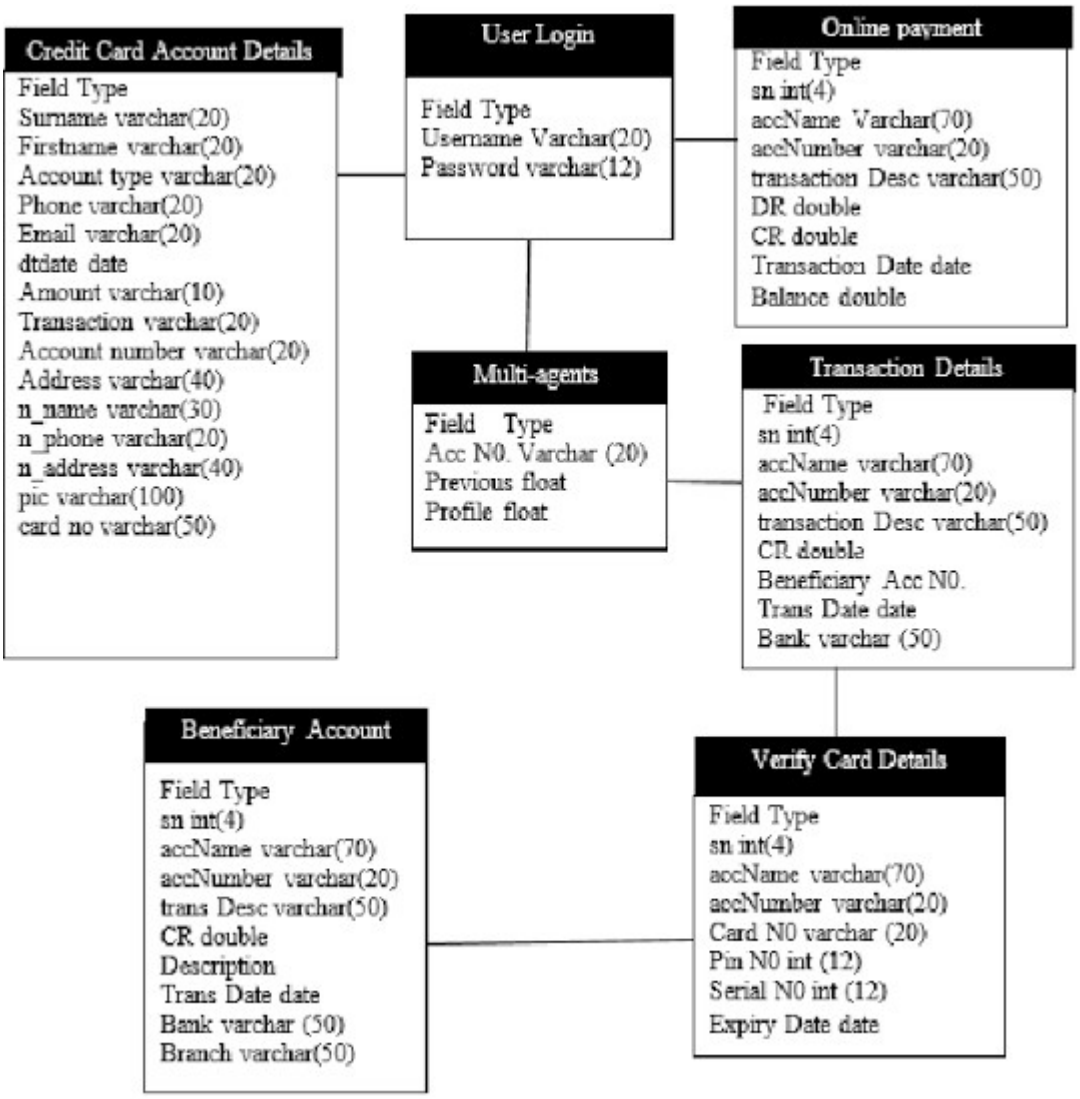


Fig 5.2 Class diagram

5.3 Flowchart

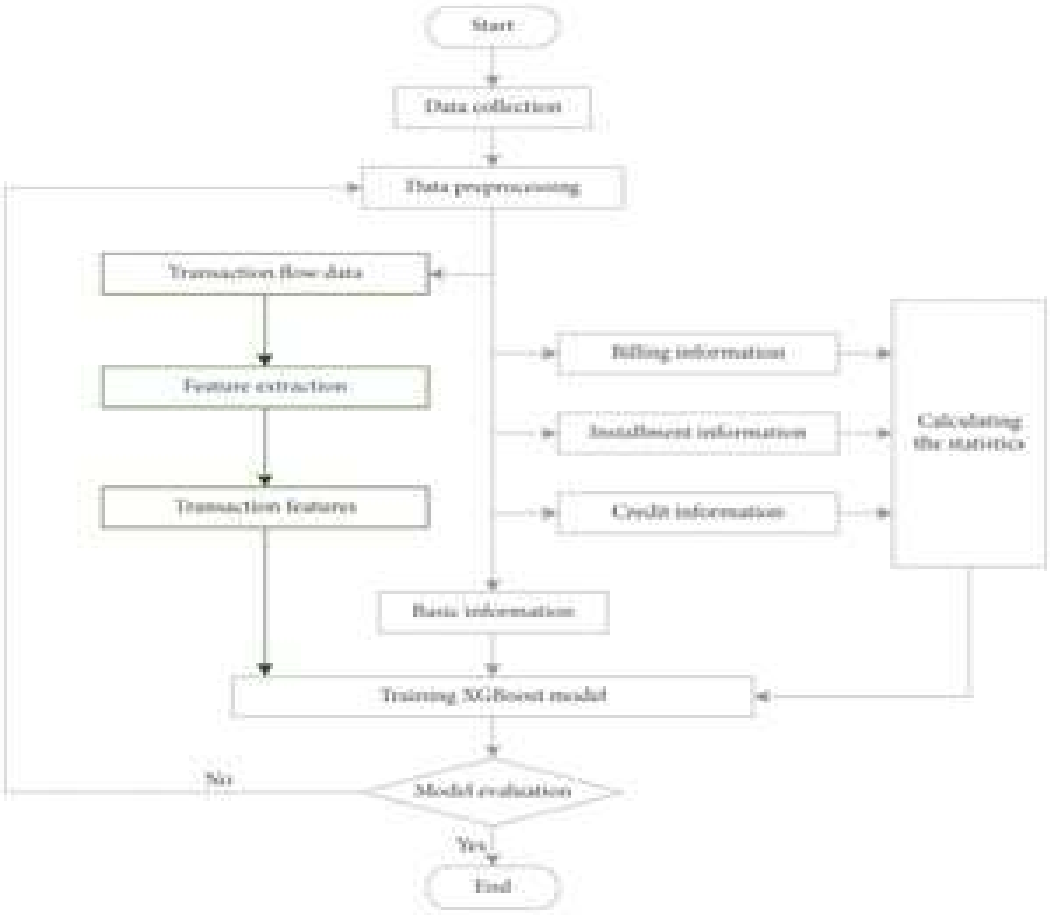


Fig 5.3 Flowchart

CHAPTER 6: CODING AND RESULTS

6.1 Coding

#Packages related to general operating system & warnings

```
import os
import warnings
warnings.filterwarnings('ignore')
```

#Packages related to data importing, manipulation, exploratory data analysis, data understanding

```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
from termcolor import colored as cl # text customization
#Packages related to data visualizaiton
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
#Setting plot sizes and type of plot
plt.rc("font", size=14)
plt.rcParams['axes.grid'] = True
plt.figure(figsize=(6,3))
```

```
plt.gray()
from matplotlib.backends.backend_pdf import PdfPages
from sklearn.model_selection import train_test_split, GridSearchCV
from xgboost import XGBClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
# understanding data
Total_transactions = len(data)
normal = len(data[data.Class == 0])
fraudulent = len(data[data.Class == 1])
fraud_percentage = round(fraudulent/normal*100, 2)
print(cl('Total number of Trnsactions are {}'.format(Total_transactions), attrs
= ['bold']))
print(cl('Number of Normal Transactions are {}'.format(normal), attrs =
['bold']))
print(cl('Number of fraudulent Transactions are {}'.format(fraudulent), attrs =
['bold']))
print(cl('Percentage of fraud Transactions is {}'.format(fraud_percentage),
attrs = ['bold']))
```

```
print(("Distribution of fraudulent
points:{:.2f}%".format(len(data[data['Class']==1])/len(data)*100)))
sns.countplot(data['Class'])
plt.title('Class Distribution')
plt.xticks(range(2),['Normal','Fraud'])
plt.show()
data.shape
```

#removing duplicates

```
data.drop_duplicates(inplace=True)
data.shape
#training data
X = data.drop('Class', axis = 1).values
y = data['Class'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 1)
#balancing using smote
from imblearn.over_sampling import SMOTE
smote=SMOTE()
X_train_smote,y_train_smote=smote.fit_resample(X_train.astype('float'),y_tr
ain)
from collections import Counter
```

```
print("before smote:" Counter(y_train))

print("after smote:" Counter(y_train_smote))

#xgboost model

xgb = XGBClassifier(max_depth = 4)

xgb.fit(X_train_smote, y_train_smote)

xgb_yhat = xgb.predict(X_test)

print('Accuracy score of the XGBoost model is
{}'.format(accuracy_score(y_test, xgb_yhat)))

pd.crosstab(y_test,xgb_yhat)
```

6.2 Output

Data:

```
Total number of Trnsactions are 284807
Number of Normal Transactions are 284315
Number of fraudulent Transactions are 492
Percentage of fraud Transactions is 0.17
```

Fig 6.1 Data of transactions

Graphical Representation Of Data:

Class Distribution

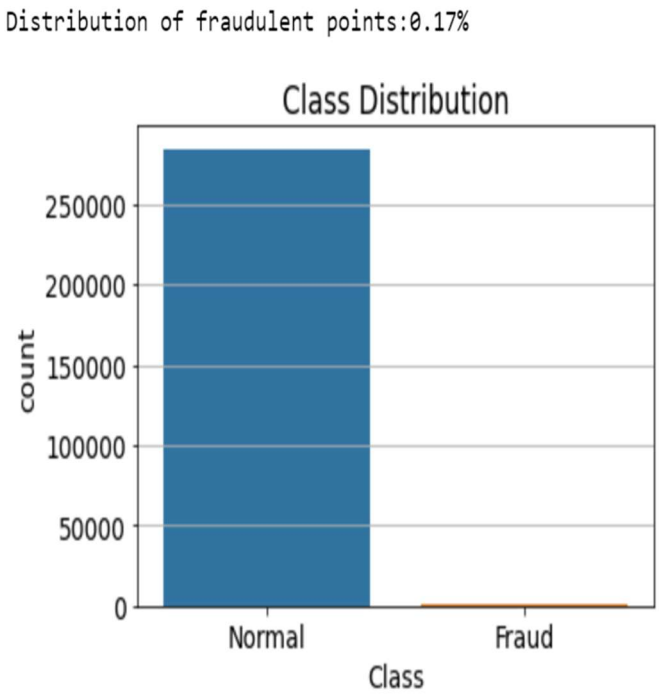


Fig 6.2 Class Distribution

Amount Per Transaction By Class

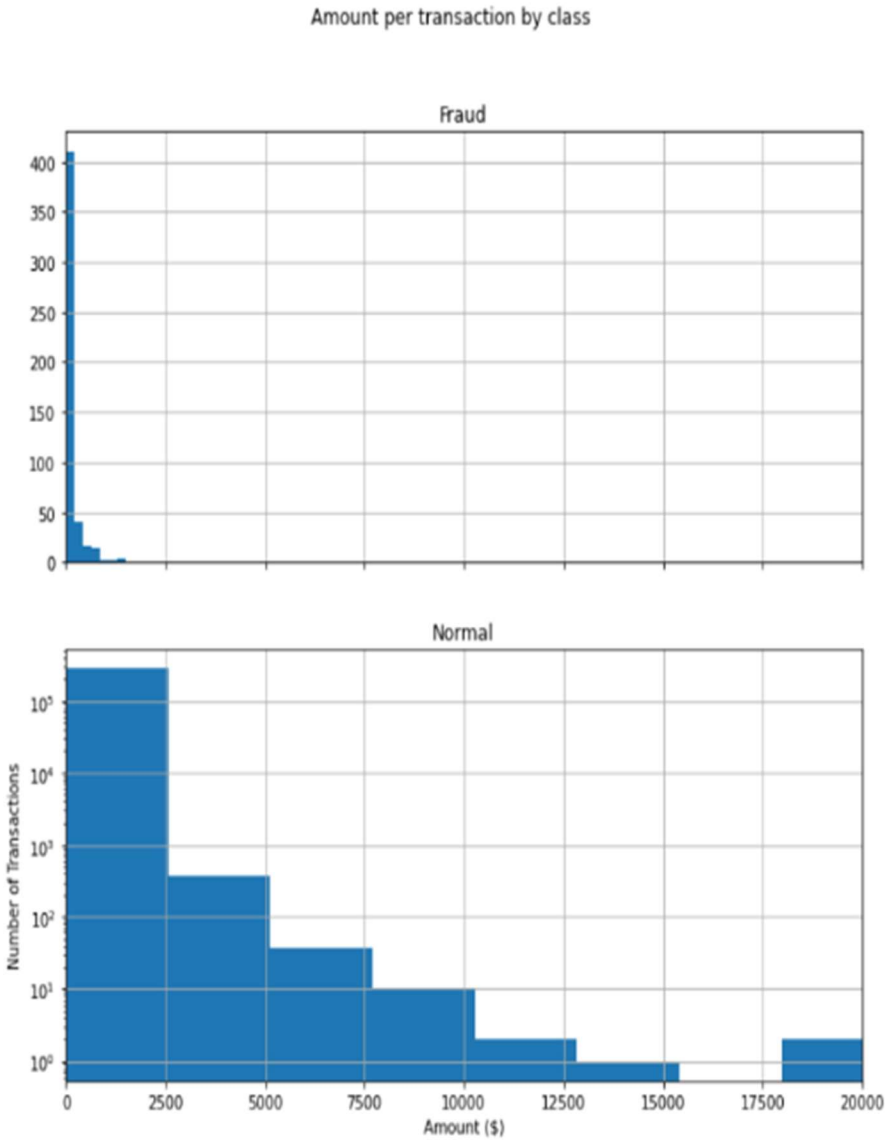


Fig 6.3 Amount per transaction by class

Time Of Transaction Vs Amount By Class

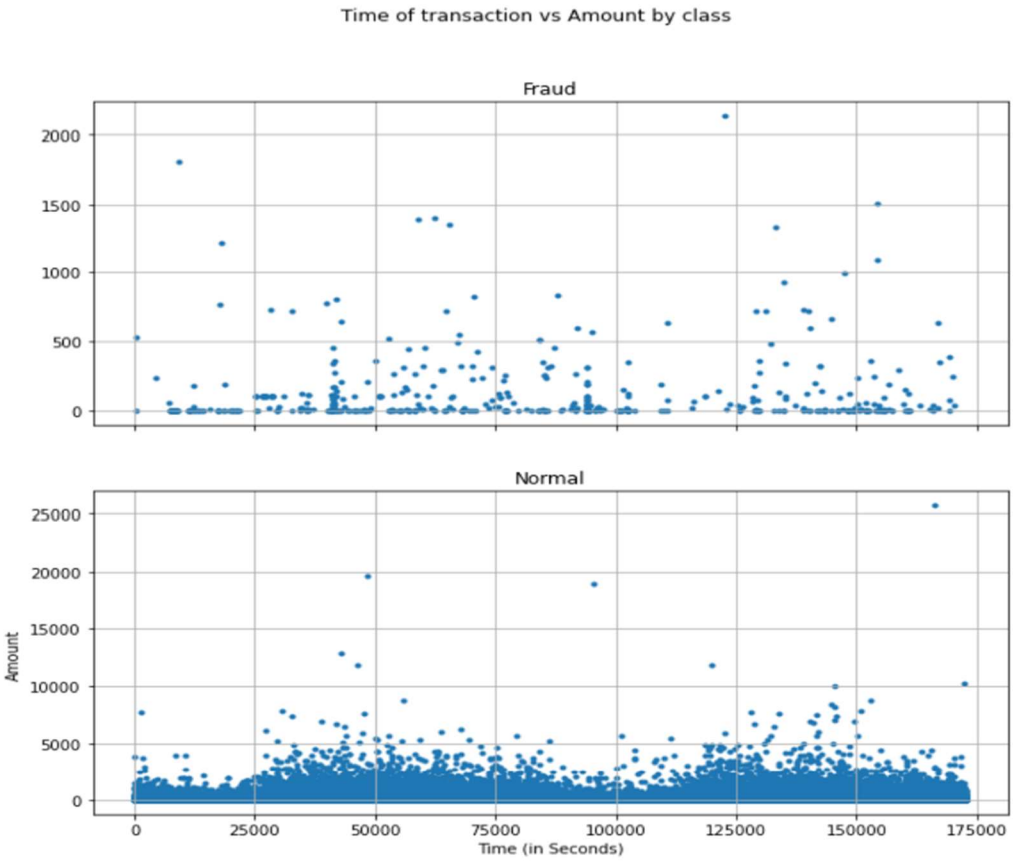


Fig 6.4 Time of transaction vs amount by class

After Removal Of Duplicates

```
In [5]: data.shape

Out[5]: (284807, 31)

In [6]: #removing duplicates
        data.drop_duplicates(inplace=True)
        data.shape

Out[6]: (283726, 31)
```

Fig 6.5 After removal of duplicates

Before And After SMOTE:

```
In [12]: from collections import Counter
         print("before smote:", Counter(y_train))
         print("after smote:", Counter(y_train_smote))

         before smote: Counter({0: 226594, 1: 386})
         after smote: Counter({0: 226594, 1: 226594})
```

Fig 6.6 Before and after SMOTE

0 represents normal transaction

1 represents fraud transactions

Accuracy And Analysis

```
Accuracy score of the XGBoost model is 0.9990131463010609
```

Out[13]:

col_0	0	1
row_0		
0	56618	41
1	15	72

Fig 6.7 Analysis and accuracy

6.3 Results

Using XGBoost algorithm with SMOTE we acquired accuracy of 99.9%.

Total transactions in data: 2,83,726(without duplicates)

We split the data in the ratio of 1:4 for testing and training respectively.

Training : 2,26,980

Testing: 56,746

In these 2,26,980 transactions, 2,26,594 are normal transactions and 386 are fraud transactions.

When 56,746 transactions are tested,

Predicted value

	Normal	Fraud
Normal	**56,618**	41
Fraud	15	**72**

56,618 normal transactions and 72 fraud transactions are identified correctly and 41 normal transactions are identified wrongly as fraud transactions and 15 fraud transactions are identified wrongly as normal transactions.

6.4 Conclusion

When providing input data of a highly unbalanced class distribution to the predictive model, the model tends to be biased towards the majority samples. As a result, it tends to misrepresent a fraudulent transaction as a genuine transaction. To tackle this problem, data-level approach, where different resampling methods can be used. Here we used SMOTE method to balance the data and used XGBoost algorithm as prediction model and acquired higher accuracy.

6.5 Future Enhancement

For future work, a cost-sensitive learning approach can be implemented by considering the misclassification costs. The cost for misclassifying a fraudulent class as a legitimate class (False Negative), which corresponds to the fraud amount (can be from few to thousands of dollars) is much higher than the cost for misclassifying a legitimate class as a fraudulent class (False Positive), which corresponds to the cost related to analyzing the transaction and contacting the cardholder. So, this type of learning deals with classifying an example into a class that has the minimum expected cost. Credit card fraud is related to the non-stationary nature of transaction distributions in which the fraudsters usually always comes with a new way to attempt the fraudulent activities. Therefore, it becomes essential to consider these changing behavior as well while developing a predictive model. Hence, a detailed study on dealing with non-stationary nature in credit card fraud detection can be performed. However, this study requires a huge amount of data.

CHAPTER 7: REQUIREMENTS

7.1 Hardware Requirements

- OS – Windows 7, 8 and 10 (32 and 64 bit)
- RAM – 4GB

7.2 Software Requirements

- Python (Pandas,NumPy,sci-kit-learn,xgboost)
- Jupyter notebook

7.3 References:

[1] X. Yu, X. Li, Y. Dong and R. Zheng, "A Deep Neural Network Algorithm for Detecting Credit Card Fraud," 2020 International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), 2020, pp. 181-183, doi: 10.1109/ICBAIE49996.2020.00045.

[2] Michael Edward Edge, Pedro R, Falcone Sampaio, “A survey of signature based methods for financial fraud detection”, journal of computers and security, Vol. 28, pp 3 8 1 – 3 9 4, 2019.

[3] Linda Delamaire, Hussein Abdou, John Pointon, “Credit card fraud and detection techniques: a review”, Banks and Bank Systems, Volume 4, Issue 2, 2019.

[4] Soltani, N., Akbari, M.K., SargolzaeiJavan, M., “A new user-based model for credit card fraud detection based on artificial immune system,” Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on., IEEE, pp. 029-033, 2012.

[5] MasoumehZareapoor, Seeja.K.R, M.Afshar.Alam, ”Analysis of Credit Card Fraud Detection Techniques: based on Certain Design Criteria”, International Journal of Computer Applications (0975 – 8887) Volume 52– No.3, 2015.

[6] RaghavendraPatidar, Lokesh Sharma, “Credit Card Fraud Detection Using Neural Network”, International Journal of Soft Computing and Engineering (IJSCE) ISSN: 2231-2307, Volume-1, 2017.

[7] A. Krenker, M. Volk, U. Sedlar, J. Bester, A. Kosh, "Bidirectional Artificial Neural Networks for Mobile-Phone Fraud Detection,” Journal of Artificial Neural Networks, Vol. 31, No. 1, pp. 92-98, 2019.