



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

J Component report

Programme : B.Tech (CSE AI & ML)
Course Title : Machine Learning Essentials
Course Code : CSE1015
Slot : A1

Title: Financial Fraud Detection

Team Members:

Nischit KR | 20BAI1159

Rohan Alroy B | 20BAI1245

Shivaprakash S J | 20BAI1205

Faculty: Dr. R. Rajalakshmi

Sign:

Date: 29.04.2022

CSE-1015- Machine Learning Essentials

J Component Report

A project report titled

Financial Fraud Detection

By

Reg. No: 20BAI1205
Jayachamarajapura

Name: Shivaprakash Shivakumaraswamy

Reg. No: 20BAI1159

Name: Nischit KR

Reg. No: 20BAI1245

Name: Rohan Alroy.B

BACHELOR OF TECHNOLOGY

IN

**COMPUTER SCIENCE AND ENGINEERING with Specialization in
Artificial Intelligence and Machine Learning**

Submitted to

Dr. R. Rajalakshmi

School of Computer Science and Engineering



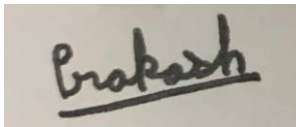
VIT[®]

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

April 2022

DECLARATION BY THE CANDIDATE

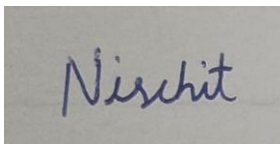
I hereby declare that the report titled “**Financial Fraud Detection**” submitted by me to VIT Chennai is a record of bona-fide work undertaken by me under the supervision of **Dr. R. Rajalakshmi, Associate Professor, SCOPE, Vellore Institute of Technology, Chennai.**

A handwritten signature in black ink that reads "Prakash".

Signature of the Candidate(20BAI1205, Shivaprakash Shivakumaraswamy Jayachamarajapura)

A handwritten signature in black ink that reads "Rohan".

Signature of the Candidate(20BAI1245, Rohan Alroy.B)

A handwritten signature in blue ink that reads "Nischit".

Signature of the Candidate(20BAI1159,Nischit KR)

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Dr. R. Rajalakshmi**, School of Computer Science and Engineering for her consistent encouragement and valuable guidance offered to us throughout the course of the project work.

We are extremely grateful to **Dr. R. Ganesan, Dean**, School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, Chennai, for extending the facilities of the School towards our project and for his unstinting support.

We express our thanks to our **Head of the Department** for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the courses.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

BONAFIDE CERTIFICATE

Certified that this project report entitled “**Financial Fraud Detection**” is a bonafide work of **Shivaprakash Shivakumaraswamy Jayachamarajapura (20BAI1205)**, **Nischit KR (20BAI1159)**, **Rohan Alroy.B (20BAI1245)** carried out the “J”-Project work under my supervision and guidance for **CSE-1015, Machine Learning Essentials**.

Dr. R. Rajalakshmi

SCOPE

TABLE OF CONTENTS

Ch. No	Chapter	Page Number
1	Abstract	6
2	Introduction	6
3	Dataset Description	7
4	Literature Survey	8
5	Proposed Methodology	16
6	Appendix	24
7	Results and Discussion	34
8	Conclusion	48
9	Reference	50

ABSTRACT:

Fraud in the financial realm is defined as deceit with the intention to illegally gain at the expense of others. Transactional fraud occurs when someone steals a credit card or data to perform a transaction. Payment card fraud losses around the world was \$28.65 billion in 2019. With the onset of the pandemic our online transactions have significantly increased and with that, there is an increase in fraudulent transactions as well. According to the Central Bank of India-RBI's annual report, Card/Internet frauds have increased to 195 crores in FY20 as compared to 71 crores in FY19. Fraudulent transactions not only affect individuals but also businesses. Estimates show that businesses can lose around 4 to 5% of their total revenue due to fraud. Detecting fraud in real time can help prevent fraudulent transactions or at least prevent subsequent fraudulent transactions from the same account. Established companies have the experience and algorithms to detect frauds, but this is not the case for small businesses. For our project we will be using a Multilayer Perceptron to detect fraudulent transactions. We intend for our project to assist small retail businesses by feeding their transactional data to our trained algorithm to detect fraudulent transactions in real time. Our endeavour will be to eliminate false negatives and achieve high accuracy. We trained the MLP with a Synthetic Financial Datasets for Fraud Detection available in Kaggle. The final MLP model achieved an ROC-AUC score of 0.955 and accuracy of 0.99.

INTRODUCTION:

Today, illegal activities regarding online financial transactions have become increasingly complex and borderless. In recent years, transaction fraud is becoming a major complication for banks as it has become very difficult for detecting fraud in the credit card and other payment systems. This unlawful activity results in substantial economic losses for both customers and organizations. To overcome this issue Machine learning plays an eminent role in detecting fraud in the transactions as it can help find hidden correlations in data to detect frauds that traditional methods of detecting frauds have overlooked.

Considering all factors, we have decided to use Neural networks to find patterns from the fraudulent transaction and thus detect the fraudulent transactions using this system. Eliminating false negatives will ensure that fraudulent transactions will not go through, however some of the real transactions might get classified as fraudulent transactions which need to be verified by the retailer.

DATASET DESCRIPTION:

This dataset is made from data on simulated mobile-money transactions. The creator of this dataset has used aggregated data from private datasets to generate a synthetic dataset to resemble the same. There are a total of 11 columns namely:

1.step - maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).

2.type - CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.

3.amount -amount of the transaction in local currency.

4.nameOrig - customer who started the transaction

5.oldbalanceOrg - initial balance before the transaction

6. newbalanceOrig - new balance after the transaction

7. nameDest - customer who is the recipient of the transaction

8. oldbalanceDest - initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).

9. newbalanceDest - new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).

10. isFraud - This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers accounts and try to empty the funds by transferring to another account and then cashing out of the system.

11. isFlaggedFraud - The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction.

A sample of the dataset is given below

Unnamed: 0	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	263314	15	PAYMENT	22726.39	C1444634854	0.00	0.00	M1897335016	0.00	0.00	0.0
1	871831	42	PAYMENT	11520.44	C1543928118	0.00	0.00	M1447434896	0.00	0.00	0.0
2	488155	19	PAYMENT	18387.10	C1637229477	0.00	0.00	M1263657510	0.00	0.00	0.0
3	231470	14	PAYMENT	7594.83	C2068036489	0.00	0.00	M485877510	0.00	0.00	0.0
4	1049672	95	CASH_IN	29754.17	C1149489060	5424630.06	5454384.23	C2089313881	695026.65	665272.48	0.0

Link to dataset: <https://www.kaggle.com/ealaxi/paysim1>

LITERATURE SURVEY:

Paper 1-Credit Card Fraud Detection with a Neural-Network

Description about the paper

This paper was written by Sushmito Ghosh and Douglas L. Reilly. The research done in this paper was based on data given by a credit card issuer. A neural network-based fraud detection system was trained on data of fraud due to stolen credit cards, lost credit cards, application fraud, counterfeit fraud, mail-order fraud, and non-received issue fraud. The result showed that the neural network detected significantly more frauds with fewer false positives when compared to the classic rule-based fraud detection procedures. The research was not only focused on detection accuracy but also the earliness of fraud detection. At the time of the publication of the paper, the fraud detection system was on an IBM 3090 at Mellon Bank and was in use for fraud detection on that bank's credit card portfolio.

Summary of the paper

Most banks rely on a rule-based statistics approach to detect fraud based on a historical analysis of fraud in the given portfolio. The paper views fraud detection as a pattern detection problem that can be solved with the use of neural networks. Values from 50 data fields in the given dataset were combined to get 20 features which would be used to train the model. A P-RCE neural network was used to build the model. The P-RCE is a three-layer, feed-forward network that is distinguished by its use of only two training passes through the data set. The P-RCE output layer was used to get a numeric value which acted as the fraud score. This fraud score, based on defined thresholds, was used to classify a transaction as fraud. Originally, a higher fraud score threshold was given for

marking a transaction from a card as fraud, but this resulted in late detection of fraud, causing a higher loss of money. To improve early detection of fraud, a lower threshold value was used.

The paper's results and my takeaways

The neural network-based detection system demonstrated that it is possible to achieve a reduction of 20% to 40% of the total fraud losses. This was also achieved with a reduced caseload of potential fraud flagged for review by humans. The system would give around 50 accounts to be reviewed per day, where nearly 40% of all fraudulent transactions were appearing among the accounts under review. This was compared to the traditional fraud detection system, which would give 750 accounts per day, resulting in only one detected fraudulent account per week. From this paper, I understand that a neural network-based approach to detecting fraud has great potential to classify transactions as fraud. The research also helped me understand that identifying fraud early is equally as important as having high accuracy in detecting fraud.

Paper 2-THE ROLE OF RANDOM FOREST IN CREDIT CARD FRAUD ANALYSIS

Description about the paper

This paper was written by Sudeep Dogga and aims to deal with the increase in fraud caused by the increasing use of credit cards as a form of payment. The paper analyses credit card fraud with the use of the Random Forest algorithm because of advantages like its higher dimensionality and high accuracy. The paper uses the Random Forest algorithm because it outperforms other algorithms in terms of comprehensiveness and accuracy in analysing huge amounts of data. The paper used a Kaggle dataset to train the model and worked on solving issues related with overfitting and class imbalance.

Summary of the paper

Credit card fraud is hard to detect as the fraudsters keep finding new ways of committing fraud. Fraudulent transactions represent only a small percentage of the total transactions. The paper suggests the use of machine learning algorithms and Random Forest in particular, to analyse and classify the large number of transactions. Random Forest uses multiple decision trees to come to a decision. Each tree gives a classification, and the majority output of all the decision trees is used to come to a final decision. The data had to be cleaned by removing columns with a high number of missing values. Some missing values were also imputed with the use of the mice algorithm. The data set was highly imbalanced, and undersampling, oversampling, and combining the two had to be done on the data. Random Forest was then applied to the dataset and the final results were recorded.

The paper's results and my takeaways

Due to Random Forest being able to handle imbalanced data, a final accuracy of 90% was achieved. Something I found very important is that the paper highlights issues related to imbalanced data. The paper suggested using techniques like undersampling and oversampling to overcome this problem. Random Forest Algorithm is a good approach to deal with detecting fraud.

Paper 3-Machine learning system for fraud detection. A methodological approach for a development platform.

Description about the paper

This paper was written by Jamal Malki, Salma El Hajjami, Mohammed Berrada, Harti Mostafa and Alain Bouju. This paper talks about fraud detection as part of the anomaly detection problem. This field is continuously looking for optimised solutions to detect anomalies. Due to the growing volume of data, the field requires techniques that are capable of analysing the data. This paper looks into building a fraud detection system on open-source Big Data technology. This paper suggests a machine learning based approach to detecting fraud. The models used were Random Forest and Multilayer Perceptron. The paper used a Kaggle credit card fraud dataset to build and test the model.

Summary of the paper

Due to the increase in e-commerce, there has been an increase in credit card transactions. The paper highlights the need to detect fraud in real time. The paper supports the use of machine learning to solve the anomaly detection problem. The data was transformed by using PCA transformation. The features were selected based on visualising how they impacted the classification. One Side Behavioral Noise Reduction was used as a sampling approach for the data. Random Forest and Multilayer Perceptron were used on the selected features. Due to the

small quantity of transactions being fraudulent, accuracy is not a good measure of performance. The model was evaluated based on the area under the curve of the ROC curve.

The paper's results and my takeaways

The Multilayer Perceptron gave an AUC metric of 0.955 while the Random Forest algorithm gave an AUC metric of 0.942. Something that the paper highlights is the misleading nature of accuracy in highly imbalanced datasets. It provides area under the curve of the ROC curve as an alternative metric to evaluate performance. This paper compared the Random Forest algorithm to the Multilayer Perceptron and helps us conclude that the Multilayer Perceptron is a better model to detect credit card fraud.

Paper 4: CREDIT CARD FRAUD DETECTION USING MACHINE LEARNING

I. INTRODUCTION

Due to the rapid growth of the E-Commerce industry, the use of credit cards for online purchases has increased dramatically. In recent years, credit card fraud is becoming a major complication for banks as it has become very difficult for detecting fraud in the credit card system. To overcome this hardship Machine learning plays an eminent role in detecting the credit card fraud in the transactions

In Machine learning the machine is trained at first to predict the output so, to predict the various bank transactions various machine learning algorithms are used.

A major challenge in applying Machine Learning to fraud detection is the presence of highly imbalanced dataset. In many publicly available databases, the vast majority of transactions are lawful, with only a small percentage of them being fraudulent.

Researchers face a big issue in designing an accurate fraud detection system with fewer fraud transactions compared to legal transactions, allowing them to detect fraudulent behavior successfully.

II. DATASET AND METHODOLOGY

In this work, Kaggle's Credit Card Fraud Detection dataset was employed. The transactions in this dataset were made by European cardholders over the period of two days in September 2013. The dataset has 31 numerical features. The "Time" feature shows the amount of time that has passed between the first and subsequent transactions in the dataset. The "Amount" function shows the total amount of credit card transactions. The "Class" feature displays the label and only allows two values: 1 for fraudulent transactions and 0 for all regular transactions. The dataset included 284,807 transactions, 492 of which were fraudulent and the rest were legitimate.

III. CONCLUSION AND RESULTS

The major purpose of this article was to look at a variety of machine learning algorithms for detecting fraudulent transactions.

Five machine learning methods were employed to detect fraud in the credit card system in this article. Data from 80% of the training dataset and 20% of the testing dataset were utilized to

evaluate the algorithms. As a consequence of the comparison, it was discovered that the accuracy score for KNN, Decision tree, Logistic Regression and Random forest is great but Xgboost algorithm produces the best results, i.e. best classifies whether transactions are fraudulent or not. This was determined using a variety of metrics, including recall score, accuracy, and precision, the f1 score, and the AUC-roc curve.

For this type of situation, having a high recall value is crucial. It has been established that feature selection and dataset balancing are critical in achieving significant results. Other machine learning techniques, such as evolutionary algorithms and various forms of stacked classifiers, as well as rigorous feature selection, should be studied further to improve results.

Paper 5: Finance Fraud Detection With Neural Network

I. INTRODUCTION

The most common fraud in daily life is credit card or debit card fraud. To address this problem, many researchers include data scientists and software engineers tried to develop a learning algorithm to find patterns from the fraud transaction and thus detect the potential frauds using this system. In past years, some machine learning algorithms is proposed to detect transaction frauds. Machine learning algorithms is a process to build a mathematical and learning model based on training data to make predictions or decisions without being explicitly programmed. Some classical machine learning algorithms like logistic regression, Naïve Bayes, and support vector machine (SVM) have been used in fraud detection task.

However, these kinds of algorithms are unable to extract higher level of information from data. In this task, we built a deep neural network model which contain multiple perception layers to extract fraud information from data.

II. DATASET

The dataset for the model is IEEE CIS [8] dataset, which is a famous dataset for transaction fraud detection. The transaction tables have 22 categorical features and 372 numeric features.

III. RESULTS

The Neural network-based model outperforms the other three models on both auc roc score and accuracy score, which means the proposed can detect fraud more accurately than other three models.

Paper 6: Credit Card Fraud Detection using Machine Learning Algorithms

I. PROPOSED METHOD

Card transactions are always unfamiliar when compared to previous transactions made the customer. This unfamiliarity is a very difficult problem in real-world when are called concept drift problems. Concept drift can be said as a variable which changes over time and in unforeseen ways. These variables cause a high imbalance in data. The main aim of our research is to overcome the problem of Concept drift to implement on real-world scenario.

II. Evaluation Method

Accuracy and precision are never good parameters for evaluating a model. But accuracy and precision are always considered as the base parameter to evaluate any model. The Matthews

Correlation Coefficient (MCC) is a machine learning measure which is used to check the balance of the binary (two-class) classifiers. It takes into account all the true and false values that is why it is generally regarded as a balanced measure which can be used even if there are different classes

III. CONCLUSION AND RESULTS

In this paper we developed a novel method for fraud detection, where customers are grouped based on their transactions and extract behavioral patterns to develop a profile for every cardholder. Then different classifiers are applied on three different groups later rating scores are generated for every type of classifier. This dynamic changes in parameters lead the system to adapt to new cardholder's transaction behaviors timely. Followed by a feedback mechanism to solve the problem of concept drift. We observed that the Matthews Correlation Coefficient was the better parameter to deal with imbalance dataset. MCC was not the only solution. By applying the SMOTE, we tried balancing the dataset, where we found that the classifiers were performing better than before. The other way of handling imbalance dataset is to use one-class classifiers like one-class SVM. We finally observed that Logistic regression, decision tree and random forest are the algorithms that gave better results.

Paper 7: AN ARTIFICIAL INTELLIGENCE APPROACH TO FINANCIAL FRAUD DETECTION UNDER IOT ENVIRONMENT: A SURVEY AND IMPLEMENTATION

KEYWORDS:

- Hidden Markov Model
- Genetic Programming
- Bayesian Belief Network
- Support Vector Machine
- Neural Networks
- Deep Learning
- Supervised and Unsupervised Learning

SUMMARY:

- Financial fraud under IoT environment is the fast-growing issue since the mobile channel can facilitate nearly any type of payments
- The online credit card fraud that does not require the presence of a credit card mainly occurs under IoT environment, since the payment under IoT environment does not require the presence of a physical payment tool; instead, it needs some information such as card number, expiration date, card verification code, and pin number to make the fraudulent payment
- We performed the validation based on the identical actual financial transaction data for machine learning method and artificial neural network
- We reviewed the latest financial fraud detection technique using machine learning and artificial neural networks and implemented the experiment based on the real financial data in Korea
- Experimental results show that machine learning based method has higher detection efficiency than neural networks at various ratios; the feature selection process must be performed according to input data

- The algorithms based on the unsupervised learning have achieved a maximum accuracy improvement of 11.5% and an average of about 11% after the feature selection process in open dataset
- Aim to Discover Hidden patterns using both unsupervised and supervised Learning
- Performed the validation based on the identical actual financial transaction data for machine learning method and artificial neural network.
- The maximum detection rate of the machine learning method was 1, the lowest detection rate was 0.736, and the average detection rate was 0.98618 when all of the algorithms were utilized.
- The maximum detection rate in all ratios of the artificial neural network was 0.914, the lowest detection rate was 0.651, and the average detection rate was 0.77228.
- 12 show the F-measure value of the artificial neural network for detecting financial fraud in various ratios.

Paper 8: INTELLIGENT FINANCIAL FRAUD DETECTION PRACTICES IN POST-PANDEMIC ERA

KEYWORDS:

- K-Nearest Neighbor
- Logistic Regression Probabilistic Neural Network Decision Trees
- Group Method of Data Handling

SUMMARY:

- With the rapid growth of information technology, the types of data used for financial fraud detection continue to expand, which can be roughly divided into three categories, i.e., basic quantitative structured data (a.k.a. tabular data), diverse semi-structured data, and complex unstructured data.
- Reviewing the history of data types, the data used in fraud detection practices have experienced the development from basic quantitative data to the current multi-source data.
- Analogous to the evolution of data types, methods for fraud detection experienced a rapid proliferation in the past decades. Thus, recently, researchers tend to incorporate and exploit information from as many aspects as possible for comprehensive monitoring.⁶² Following these trends, in this section, we survey existing financial fraud detection methods based on the technical development routes. We highlight the research proposed in the recent 2 years to demonstrate how researchers excavate related information from multiple perspectives in the post-pandemic era. For those antiquated techniques, we merely list representative cases to clarify the historical line.
- In the early stages, data used for fraud detection are usually highly structured, e.g., transaction logs or well-designed financial metrics, and the means for detecting fraud are undecorated. A number of rules and static thresholds can be used to filter out misbehavior. A straightforward case is that a system will alert if important indexes like liquidity or profitability are unusually high or low. Then, expert systems were designed to facilitate the work of human auditors. They generally use symbolic rules to encode knowledge created by human experts, which was an important part of artificial intelligence during the 1970s and 1980s. This encoded knowledge base is then queried to yield a result through reasoning.¹⁰⁵ For example, Quinlan et al. Nevertheless, these manual and rule-based approaches have become particularly costly and ineffective at present.¹⁰⁶ As fraudsters begin to employ trickier strategies to elude regulators, rich financial-related information is required to be analyzed, which undoubtedly exacerbates difficulties in extracting and summarizing effective rules.

- Considering the defects of rule-based approaches, growing numbers of machine learning-based methods have been developed. applied LR model to detect fraudulent insurance claims based on the Spanish market and estimated the error rate.⁸⁷
- Deep Learning (DL) is becoming a particular type of machine learning, as it achieves great success in various domains. At its heart, the most essential advantages of DL models are that they can extract features directly from raw data without hard-coding task-specific knowledge or tedious feature engineering. With the increasingly complex fraud in the financial scenario, researchers try their best to use these massive and various data to uncover these concealed miscreants. Thus, DL techniques for fraud detection have gained popularity over recent years, especially in the post-pandemic era where digital transformation has become the new normal. In this section, we discuss the surveyed approaches according to the different types of input data.
- Hence, for better excavating and utilizing sequential data, more complex and elaborate network structures are designed. Convolutional Neural Networks (CNNs), with the convolutional operations, are capable of capturing short-term contextual information and can be applied in financial fraud detection. Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) are typical architectures of RNNs. considered the fraud detection problem in e-commerce as a sequence classification task and employed LSTM networks to incorporate the historical behavior of the users for detecting fraud on new incoming transactions.⁹¹ Branco et al. proposed a Hierarchical Explainable Network (HEN) to model users' behavior sequences. In HEN, a field-level extractor encodes both first- and second-order information through Factorization Machines (FM). Then, an event-level extractor captures higher-order feature interactions for better sequence representation.
- Although data-driven artificial intelligent techniques have achieved excellent performance in the financial fraud detection domain, there are still key issues remaining unsolved, as financial fraud schemes are rapidly evolving to adapt to this new digital environment. In this section, we provide the major challenges and suggest directions for future work from task-oriented, data-oriented, and model- oriented perspectives.
- Financial fraud is harder to identify due to its increasing secretiveness and complexity
- The secretiveness of financial fraud leads to the natural error in samples
- The complexity of financial activities leads to massive information involved
- Data isolation is difficult to resolve
- Large-scale data processing brings great challenges to model training
- Financial fraud detection models need to be more flexible and interpretable
- Robustness needs to be strengthened
- Interpretability needs to be improved

Paper 9: PERFORMANCE OF MACHINE LEARNING TECHNIQUES IN THE DETECTION OF FINANCIAL FRAUDS

KEYWORDS:

- Effective Financial Fraud Detection Graph Neural Networks
- Graph
- Feature Vectors
- MultiVector Perceptron
- Deep Learning

SUMMARY:

- The aim of this study is to identify the techniques and methods that give the best results that have been perfected so far.

- Probabilistic neural network (PNN) is a feed-forward Neural Networks (NN) involving a one pass training algorithm used for classification and mapping of data
- Bivariate Probit Model (BP) is typically used where a dichotomous indicator is the outcome of interest and the determinants of the probable outcome includes qualitative information in the form of a dummy variable where, even after controlling for a set of covariates, the possibility that the dummy explanatory variable is endogenous cannot be ruled out a priori
- Halvaie developed a novel model for credit card fraud detection using Artificial Immune System (AIS) and introduced a new model called AIS-based Fraud Detection Model (AFDM), increase the accuracy up to 25%, reduce the cost up to 85%, and decrease system response time up to 40% compared to the base algorithm
- In 2007, Kirkos investigated the usefulness of Decision Trees, Neural Networks and Bayesian Belief Networks in the identification of fraudulent financial statements. Genetic algorithm approach was proposed by HOOGS the patterns are capable of identifying potentially fraudulent behavior despite occasional missing values, and provide low false positive rates
- Types of Financial Fraud:
 - Securities and commodities fraud,
 - Money Laundering
 - Financial statement fraud (corporate fraud)
 - Credit card fraud
 - Mortgage Fraud
- Detection Techniques:
 - Descriptive or Unsupervised Techniques:
 - Self-Organizing Maps
 - Group method of data handling
 - Outlier detection methods
 - Association rule analysis
 - Density based spatial clustering of applications with noise (DBSCAN)
 - Predictive Techniques:
 - Logistic Regression (LR)
 - Decision Trees (DT)
 - Classification and Regression Tree (CART)
 - Association rule analysis
 - Cost-sensitive decision tree (CSDT)
 - Neural Networks (NN)
 - Probabilistic neural network (PNN)
 - Support Vector Machines (SVM)
 - Naïve Bayes (NB)
 - Bayesian belief network (BBN)
 - Bayesian skewed logit model (BSL)
 - K-nearest neighbor (KNN)
 - Bivariate Probit Model (BP)
 - Artificial & Computational Intelligence Techniques:
 - Genetic Algorithm (GA)
 - Genetic programming (GP)
 - Scatter Search (SS)
 - Hidden Markov Model (HMM)
 - Iterative Dichotomiser 3 (ID3)
 - Artificial Immune System (AIS)
 - Artificial Immune Recognition System (AIRS)
 - Artificial neural network (ANN)
 - Multilayer Perception Algorithm (MPL)

- Parenclitic Network (PN)
- Multi-layer feed forward neural network (MLFF-NN)
- Other Concepts:
 - Fuzzy logic
 - Dempster Shafer Theory (DST)
 - Computational fraud detection model (CFDM)
 - Locally Weighted Learning (LWL)

PROPOSED METHODOLOGY:

Preprocessing

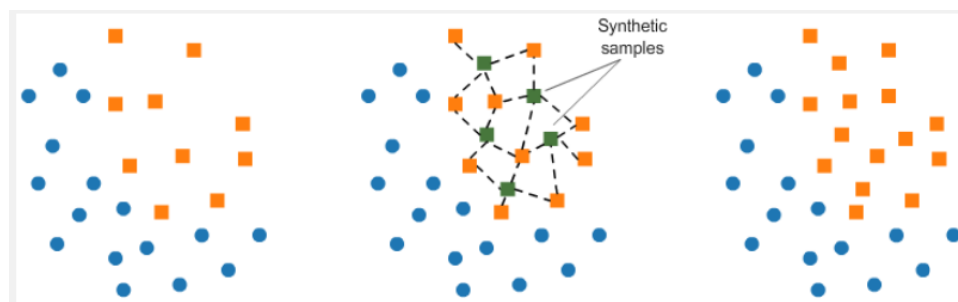
As the original dataset is quite large a random sample of 100,000 transactions were used in the project. Before building the model, the data has to be preprocessed. The preprocessing involved dropping the isFlaggedFraud feature to prevent data leakage, converting the Type column into dummy variables, hashing nameOrig and nameDest, and normalizing the data. The data was then split into training and testing sets where 70% of the data was used for training and 30% was used for testing.

Dealing with data imbalance

SMOTE and its variants were used to deal with the data imbalance present in the dataset. Given below is the explanation of each of the methods.

Synthetic Minority Oversampling Technique (SMOTE)

This technique generates synthetic data for the minority class. SMOTE works by randomly picking a point from the minority class and computing the k-nearest neighbors for this point. The synthetic points are added between the chosen point and its neighbors. The figure given below explains the workings of SMOTE on a dataset.



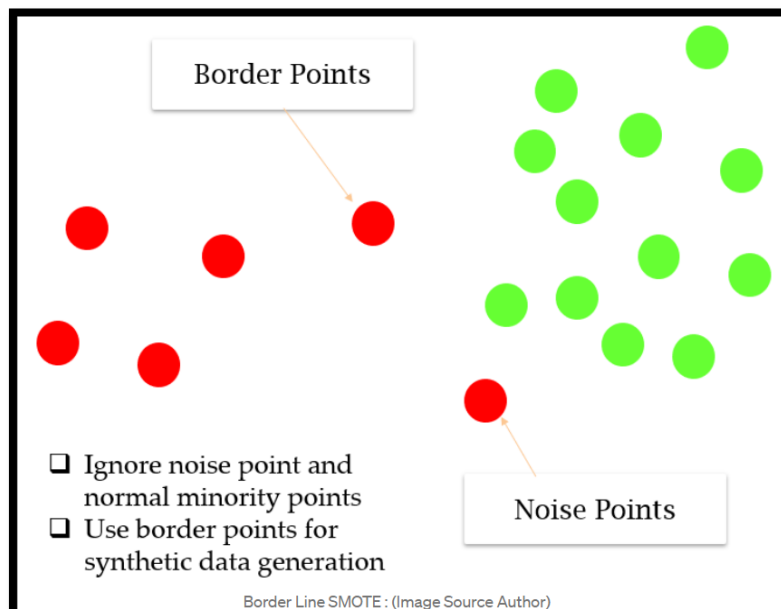
VARIANTS OF SMOTE:

An issue with SMOTE:



If there are observations in the minority class which are outlying and appears in the majority class, it causes a problem for SMOTE, by creating a line bridge with the majority class.

Borderline SMOTE: - This solves the above issue.



Borderline-SMOTE is a variation of the SMOTE. Just like the name implies, it has something to do with the border.

So, unlike with the SMOTE, where the synthetic data are created randomly between the two data, **Borderline-SMOTE only makes synthetic data along the decision boundary between the two classes.**

This algorithm starts by classifying the minority class observations. It classifies any minority observation as a noise point if all the neighbours are the majority class and such an observation is ignored while creating synthetic.

Further, it classifies a few points as border points that have both majority and minority class as neighbourhood and resample completely from these points (Extreme observations on which a support vector will typically pay attention to).

We can implement Borderline-SMOTE using the BorderlineSMOTE class from imbalanced-learn.

Instead of generating new synthetic examples for the minority class blindly, we would expect the Borderline-SMOTE method to only create synthetic examples along the decision boundary between the two classes.

Also, there are two kinds of Borderline-SMOTE; there are Borderline-SMOTE1 and Borderline-SMOTE2. The differences are simple; Borderline-SMOTE1 also oversampled the majority class where the majority data are causing misclassification in the decision boundary, while Borderline-SMOTE2 only oversampled the minority classes.

Issue: End up giving more attention to these extreme observations.

Borderline-SMOTE SVM

Another variation of Borderline-SMOTE is Borderline-SMOTE SVM, or we could just call it SVM-SMOTE.

The main differences between SVM-SMOTE and the other SMOTE are that instead of using K-nearest neighbours to identify the misclassification in the Borderline-SMOTE, the technique would incorporate the SVM algorithm.

In the SVM-SMOTE, the borderline area is approximated by the support vectors after training SVMs classifier on the original training set. Synthetic data will be randomly created along the lines joining each minority class support vector with a number of its nearest neighbours.

What special about Borderline-SMOTE SVM compared to the Borderline-SMOTE is that more data are synthesized away from the region of class overlap. It focuses more on where the data is separated.

ADASYN:

ADASYN is another variation from SMOTE. ADASYN takes a more different approach compared to the Borderline-SMOTE. While Borderline-SMOTE tries to synthesize the data near the data decision boundary, **ADASYN creates synthetic data according to the data density.**

The synthetic data generation would be inversely proportional to the density of the minority class. It means more synthetic data are created in regions of the feature space where the density of minority examples is low, and fewer or none where the density is high.

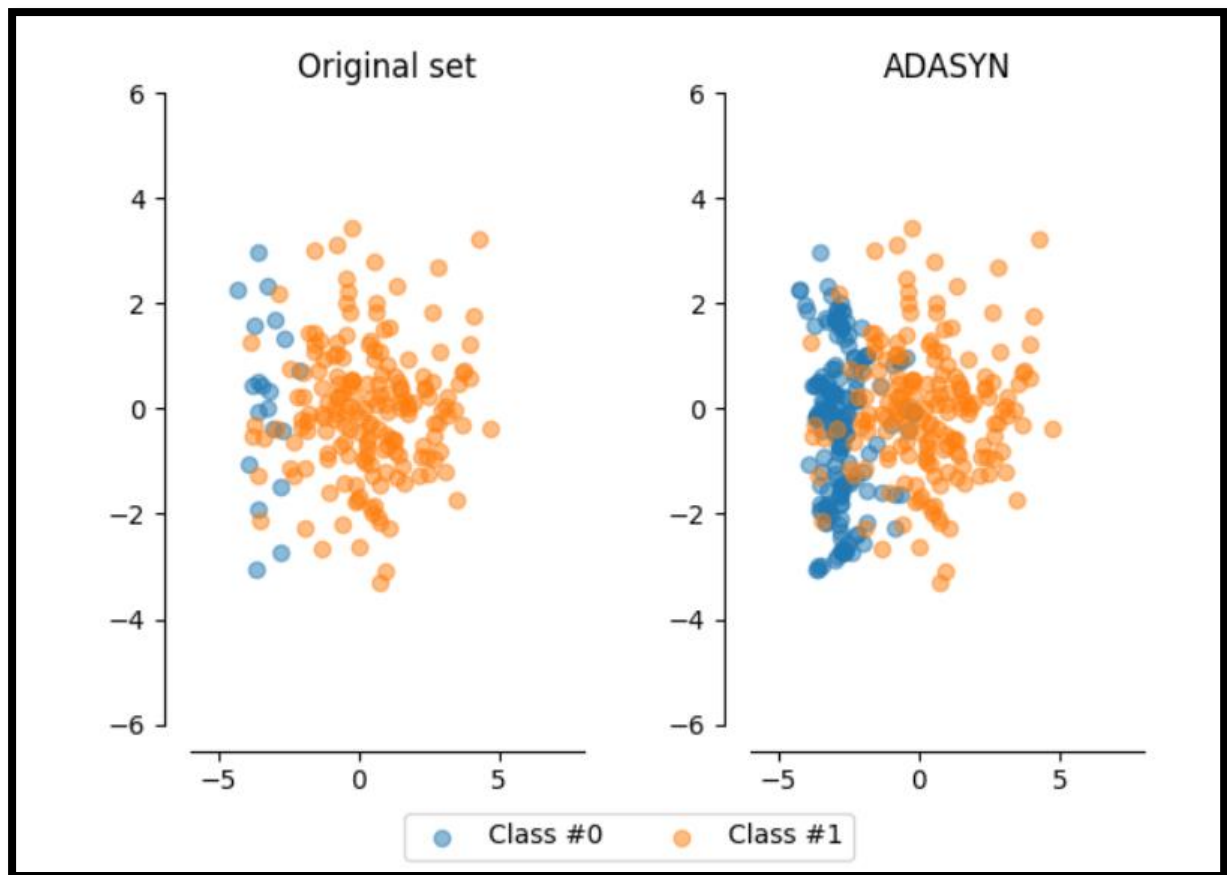
In simpler terms, in an area where the minority class is less dense, the synthetic data are created more. Otherwise, the synthetic data is not made so much

ADASYN is a more generic framework, for each of the minority observations it first finds the impurity of the neighbourhood, by taking the ratio of majority observations in the neighbourhood and k .

Minority Class	Minority Neighbours	Majority Neighbours	Impurity Ratio
Obs 1	3	2	.6
Obs 2	4	1	.4
Obs 3	1	4	.8
Obs 4	5	0	0

ADASYN Impurity Ratio

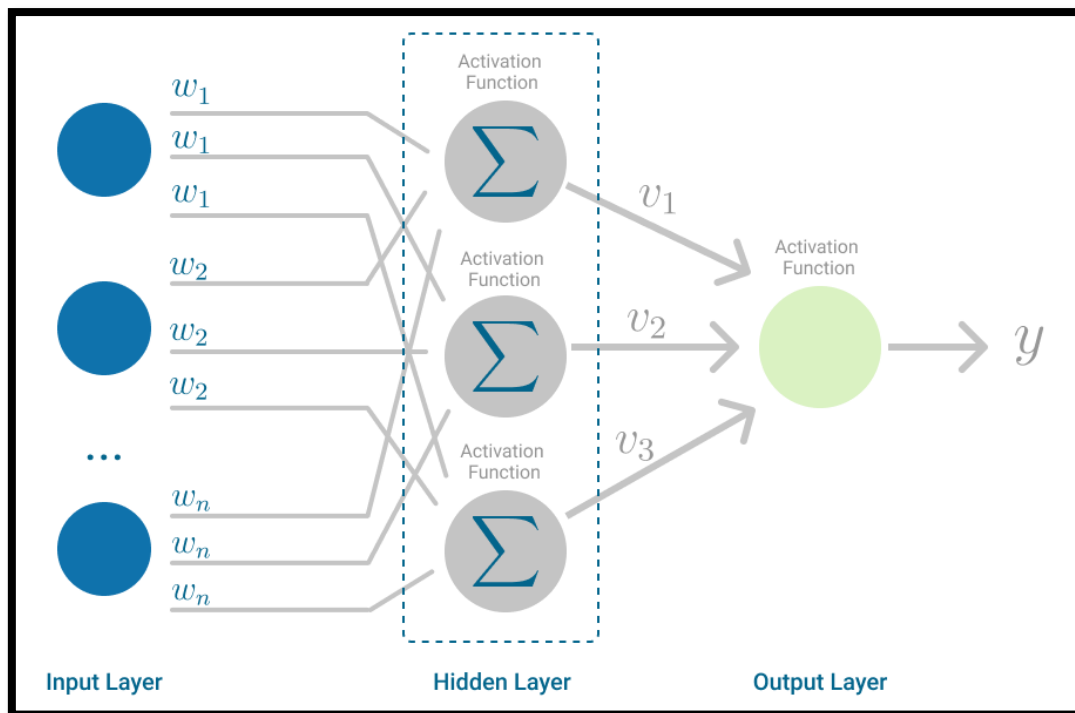
Now, first of all, this impurity ratio is converted into a probability distribution by making the sum as 1. Then higher the ratio more synthetic points are generated for that particular point. **Hence the number of synthetic observations to be created for Obs 3 is going to be double that of Obs 2.** So, it's not so extreme as Borderline SMOTE and the boundary between the noise point, border point, and regular minority points are much softer. (Not a hard boundary). Thus, the name adaptive.



MLP (Multilayer Perceptron)

The **Multilayer Perceptron** was developed to tackle this limitation. It is a neural network where the mapping between inputs and output is non-linear.

A Multilayer Perceptron has input and output layers, and one or more **hidden layers** with many neurons stacked together. And while in the Perceptron the neuron must have an activation function that imposes a threshold, like ReLU or sigmoid, neurons in a Multilayer Perceptron can use any arbitrary activation function.



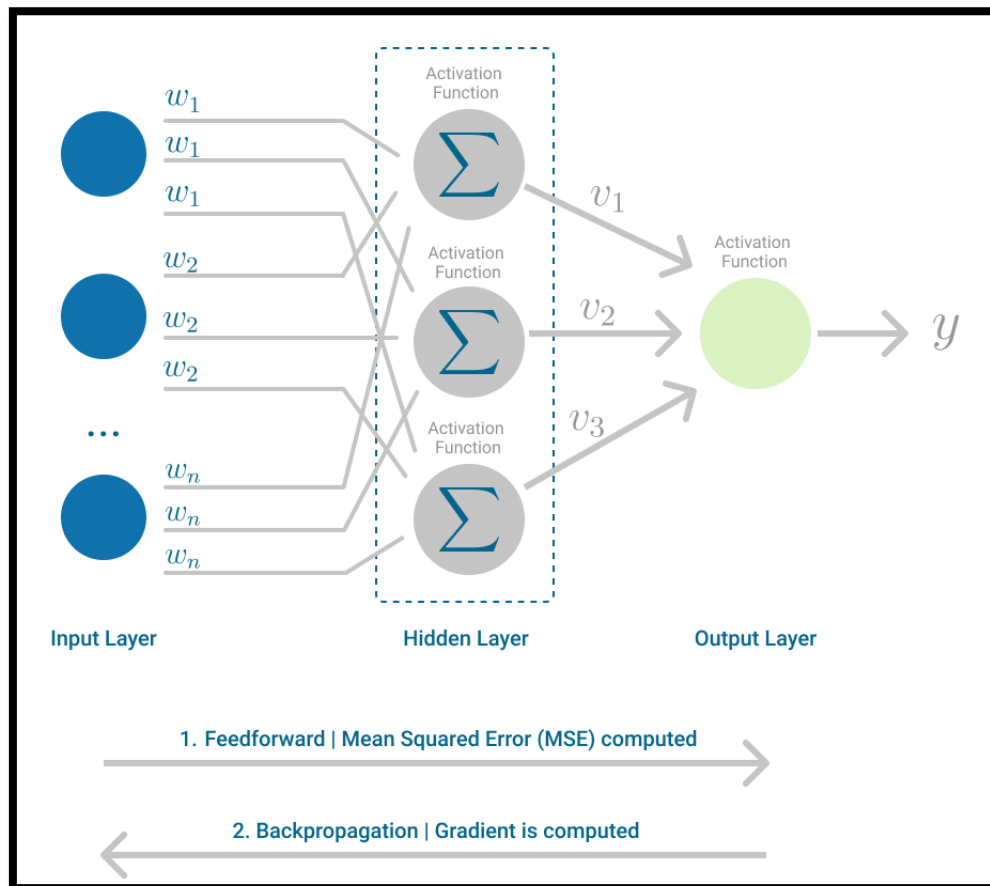
Multilayer Perceptron falls under the category of [feedforward algorithms](#), because inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like in the Perceptron. But the difference is that each linear combination is propagated to the next layer.

Each layer is *feeding* the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer.

If the algorithm only computed the weighted sums in each neuron, propagated results to the output layer, and stopped there, it wouldn't be able to *learn* the weights that minimize the cost function. If the algorithm only computed one iteration, there would be no actual learning.

This is where **Backpropagation** comes into play. Backpropagation is the learning mechanism that allows the Multilayer Perceptron to iteratively adjust the weights in the network, with the goal of minimizing the cost function.

There is one hard requirement for backpropagation to work properly. The function that combines inputs and weights in a neuron, for instance the weighted sum, and the threshold function, for instance ReLU, must be differentiable. These functions must have a **bounded derivative**, because [Gradient Descent](#) is typically the optimization function used in Multilayer Perceptron.



In each iteration, after the weighted sums are forwarded through all layers, the gradient of the **Mean Squared Error** is computed across all input and output pairs. Then, to propagate it back, the weights of the first hidden layer are updated with the value of the gradient. That's how the weights are propagated back to the starting point of the neural network!

$$\Delta_w(t) = -\varepsilon \frac{dE}{dw(t)} + \alpha \Delta_w(t-1)$$

Bias
Error
Learning Rate

Gradient
Current Iteration
Weight vector
Gradient
Previous Iteration

This process keeps going until gradient for each input-output pair has converged, meaning the newly computed gradient hasn't changed more than a specified *convergence threshold*, compared to the previous iteration.

APPENDIX:

Implementation and code:

```
# -*- coding: utf-8 -*-
"""ComparisonVersion_FinalReport.ipynb

Automatically generated by Colaboratory.

Original file is located at
https://colab.research.google.com/drive/1p42eaZ7TePZOQ\_ussQAR\_sBml32UME7a

# **ML Final Review**

## TEAM MEMBERS:

## NISCHIT K R - 20BAI1159

## SHIVAPRAKASH SHIVAKUMARASWAMY JAYACHAMARAJAPURA- 20BAI1205

## ROHAN ALROY.B - 20BAI1245

## Importing Libraries
"""

# Commented out IPython magic to ensure Python compatibility.
import numpy as np
import math
import pandas as pd
import datetime
import matplotlib.pyplot as plt
import matplotlib.cm as cm
# %matplotlib inline
from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR')

import seaborn as sns
sns.set_style("dark")
from matplotlib.pyplot import pie, axis, show
from sklearn import preprocessing
from scipy.stats import skew, boxcox
from sklearn import metrics
import warnings

"""## Reading the data

"""
```

```

data = pd.read_csv("sampled_fraud.csv")

data.shape

data.head()

data.isnull().sum()

data.info()

data['type'].value_counts()

data.describe()

#Total number of unique customers
print(f"Total number of unique customers are {data.nameOrig.nunique()}")
print(f"Total number of unique recipients are {data.nameDest.nunique()}")
print(f"Average no. of transactions per customer are {data.shape[0]/data.nameOrig.nunique()}")
print(f"Average no. of transactions per recipient are {data.shape[0]/data.nameDest.nunique()}")

"""# EXPLORATORY DATA ANALYSIS

# Through the exploratory data analysis, we can prove some hypothesis about
fraud attacks and get some visual interpretations from data.

# Plotting different types of transactions
"""

fig, ax = plt.subplots()
plt.rcParams['figure.figsize'] = (20, 16)
sns.set_theme(style="whitegrid")
tips = sns.load_dataset("tips")
sns.countplot(x='type', data=data,palette="tab20_r",edgecolor='black')
for p in ax.patches:
    ax.annotate(str(format(int(p.get_height()), 'd')), (p.get_x(),
p.get_height()*1.01))
plt.title('Total transactions',fontsize=25,color='#E43A36')
df_pie=data.groupby(['type']).size()
#axis('equal');
#pie(df_pie, labels=df_pie.index);
# Pie chart
plt.rcParams['figure.figsize'] = (8, 6)
labels = df_pie.index
sizes = df_pie
# only "explode" the 2nd slice (i.e. 'Hogs')

```

```

explode = (0.02, 0.02, 0.02, 0.02,0.02)
#add colors
colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99','#FAE959']
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%',
        shadow=True, startangle=90,wedgeprops={"edgecolor":"0",'linewidth':
0.5,'linestyle': 'solid', 'antialiased': True})
# Equal aspect ratio ensures that pie is drawn as a circle
ax1.axis('equal')
plt.tight_layout()
plt.title('% OF ALL TYPES OF TRANSACTIONS',fontsize=25,color='#E43A36')
plt.show()

pd.plotting.scatter_matrix(data)

data.corr()["isFraud"].sort_values(ascending = False)

print(data["isFraud"].value_counts())
sns.countplot(y="isFraud",data=data)
plt.show()

data['type'].value_counts()

data['type'] =
data['type'].map({'CASH_OUT':1,'PAYMENT':2,'CASH_IN':3,'TRANSFER':4,'DEBIT':5}
)

x = data[['type','amount','oldbalanceOrig','newbalanceOrig']]
x

y = data['isFraud']
y

y.count()

y_true = list(data['isFraud'])
y_pred = list(data['isFlaggedFraud'])
cf_matrix = metrics.confusion_matrix(y_true, y_pred)
sns.heatmap(cf_matrix, annot=True, cmap='YlGnBu')
plt.xlabel("isFlaggedFraud")
plt.ylabel("isFraud")
plt.title("Confusion matrix for simulator results")

"""The Fraud Check System in place has performed poorly, catching almost
negligible percentage of fraud transcation, 16 in total, whereas there are
more than 8000 fraud transactions in total, but it has not wrongly flagged
legit transactions

```

```

# Plotting types of transactions which fall under the Fraud category
"""

fig, ax = plt.subplots()
df_fraud=data[data['isFraud']==1]
sns.countplot(x='type',data=df_fraud,
ax=ax,palette="tab20_r",edgecolor='black',alpha=0.75)
def change_width(ax, new_value) :
    for patch in ax.patches :
        current_width = patch.get_width()
        diff = current_width - new_value

        # we change the bar width
        patch.set_width(new_value)

        # we recenter the bar
        patch.set_x(patch.get_x() + diff * .5)

change_width(ax, .5)
for p in ax.patches:
    ax.annotate(str(format(int(p.get_height()), 'd')), (p.get_x(),
p.get_height()*1.01))
plt.title('FRAUDULENT TRANSCATIONS',fontsize=25,color='#E43A36')
plt.show()

"""There are only two types of transctions which fall under fraud category as
shown in the above bar plot

There are two types of transactions based on Recipients, One is normal
customer and the other one is Merchant, with simple checking it was found that
all the transactions pertaining with the Merchants were legit and none of them
was found to be fraud, same is described in the below pie chart.
"""

df = data.copy()
df['percentage'] = np.where(df['amount']>df['oldbalanceOrg'], 100.0,
(df['amount']/df['oldbalanceOrg'])*100)
df['dest']=df['nameDest'].astype(str).str[0]

list1=list(df[df['isFraud']==1].groupby(['dest']).size())
list2=list(df[df['isFraud']==0].groupby(['dest']).size())
newlist=list1+list2
newlist
plt.rcParams['figure.figsize'] = (8, 6)
labels = ['Customers(Fraud)', 'Customers(not Fraud)', 'Merchants']
sizes = newlist
# only "explode" the 2nd slice (i.e. 'Hogs')

```

```

explode = (0.05,0.05, 0.05)
#add colors
colors = ['#ff9999','#ff9999','#66b3ff']
fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, colors=colors,
autopct='%1.1f%%',
        shadow=True, startangle=90,wedgeprops={"edgecolor":"0",'linewidth':
0.5,'linestyle': 'solid', 'antialiased': True})
# Equal aspect ratio ensures that pie is drawn as a circle
ax1.axis('equal')
plt.tight_layout()
plt.title('Total transactions distribution on types of
recipients',fontsize=25,color='#E43A36',)
plt.show()

"""There are in total about 0.1% transactions and all are in the Customer's
Category

# Plotting heatmap to check correlation between different parameters
"""

df_heatmap =
data[['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDe
st','isFraud','isFlaggedFraud']]
sns.heatmap(df_heatmap.corr(), cmap="YlGnBu", annot=True)
plt.title('CORRELATION HEATMAP OF ALL PARAMS',fontsize=25,color='#E43A36')

"""Fraud category has more dependence on Amount category, we will try to plot
it

# Scatter Plot between Amount and Balance, highlighting the fraud transaction
"""

newscatplot=df[df['isFraud']==1]
plt.figure(figsize=(8,8))
ax = plt.gca()
ax.set_ylim(0,2*1e7)
ax.set_xlim(0,2*1e7)
df.plot.scatter(x='oldbalanceOrg',y='amount',
ax=ax,edgecolors='black',s=100,alpha=0.1,label="Legit transaction")
newscatplot.plot.scatter(x='oldbalanceOrg',y='amount', color='#FCD735',
ax=ax,edgecolors='red',s=100,alpha=0.1,label="Fraud transcation")
plt.title('Amount vs Balance',fontsize=25,color='#E43A36')

"""There are various inferences from the above graph

Firstly,The fraud line is inclined equally from both axis which implies that
whenever the amount dealt was equal to the balance, it was most likely a Fraud

```

After a while the fraud line hits a constant limit, which implies that there is limit in some form to the maximum amount which can be CASH OUT or TRANSFER, hence there are no fraud amounts which surpass this limit

One popular trend is fraud attacks by big criminals happen in a short period of time. To check this, we can plot the fraud attacks with time. In our data, we have data every hour. The below graph shows the fraud attack every hour in the 102 hours. As expected there are peaks and troughs and also a very big peak. This suggests that frauds happen in short period of time.

```
"""
```

```
sns.lineplot(x=list(range(1,102)),y=data.groupby("step")["isFraud"].sum())
plt.xlabel("Hour of the month")
plt.ylabel("Number of transactions per hour")
plt.show()
```

```
"""Another analysis that can be interesting is at which hour of a day, the
fraud attacks generally happen. From the step variable, we can get the hour of
day. The below plot shows the frauds at different hours of day. It tells that
frauds happen during sleeping hours the most. Close to 20% of transactions
that happen during 4 AM and 5 AM are fraud transactions."""
```

```
data["hour"] = data.step % 24
frauds_hour =
pd.concat([data.groupby("hour")["isFraud"].sum(),data.groupby("hour")["isFraud
"].count()],axis=1)
frauds_hour.columns = ["Frauds","Transactions"]
frauds_hour["fraud_rate"] = frauds_hour.Frauds/frauds_hour.Transactions
sns.barplot(x=frauds_hour.index,y=frauds_hour.fraud_rate)
plt.show()
```

```
"""We can look at the transaction amount and customer's opening balance for
fraud and non-fraud cash-out and tranfer transactions seperately. Median for
these variables are compared because mean is biased because of outliers. From
the plots, we can see that these variables are abnormally high for fraud cash-
out transactions compared to non-fraud cash-out transactions
```

```
## Preprocessing
```

```
**To prevent data leakage we drop isFlaggedFraud**
"""
```

```
data = data.drop("isFlaggedFraud", axis = 1)
data.info()
```

```
"""**Converting Categorical variables to numerical ones**
```

```

Making dummy variables for type
"""

types = pd.get_dummies(data["type"],drop_first = True)
data.drop("type",axis = 1,inplace = True)

data = pd.concat([data, types], axis = 1)
data.info()

"""Hashing nameOrig and nameDest"""

data["nameOrig"] = pd.util.hash_array(data["nameOrig"].to_numpy())
data["nameDest"] = pd.util.hash_array(data["nameDest"].to_numpy())

"""**Normalizing the data**"""

data.head(20)

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled = scaler.fit_transform(data)
data = pd.DataFrame(scaled, columns = data.columns)

data.head()

"""**Splititng the data into training and testing data sets**"""

from sklearn.model_selection import train_test_split
X_train,X_test, Y_train, Y_test = train_test_split(data.drop("isFraud", axis =
1),data["isFraud"], test_size = 0.3, random_state = 69)

"""**As the data is highly imbalanced SMOTE is used an Oversampling technique
for the training data**"""

# Looking at the counts of the Y_train before using SMOTE
Y_train.value_counts()

from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state =0)
# making copies of X_train and Y_train to test an MLP model without SMOTE
X_train_without_SMOTE = X_train.copy()
Y_train_without_SMOTE =Y_train.copy()
# applying SMOTE on X_train and Y_train
X_train, Y_train = smote.fit_resample(X_train, Y_train)
Y_train.value_counts()

"""## Building the initial MLP model with the data which doesn't have SMOTE
applied to it"""

```

```

from sklearn.neural_network import MLPClassifier
initial_model = MLPClassifier(random_state=0)
initial_model.fit(X_train_without_SMOTE, Y_train_without_SMOTE)
predictions_initial_model = initial_model.predict(X_test)

"""**Getting the evaluation of the initial model**"""

from sklearn.metrics import classification_report
print(classification_report(Y_test, predictions_initial_model))

from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, predictions_initial_model)

from sklearn.metrics import roc_auc_score
prob_initial_model = initial_model.predict_proba(X_test)[::,1]
roc_auc_score(Y_test, prob_initial_model)

"""## Building the Multi Layer Perceptron model

Based on finetuning a MLP model with hidden layer sizes = (50,50,50) is built
"""

from sklearn.neural_network import MLPClassifier
model = MLPClassifier(hidden_layer_sizes=(50, 50, 50), random_state=0)
model.fit(X_train, Y_train)
predictions = model.predict(X_test)

"""## Evaluating the Model"""

from sklearn.metrics import classification_report
print(classification_report(Y_test, predictions))

from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, predictions)

from sklearn.metrics import roc_auc_score
prob = model.predict_proba(X_test)[::,1]
roc_auc_score(Y_test, prob)

"""A final metric of .955 for roc auc is achieved.

## Downloading the trained model
"""

from joblib import dump, load
dump(model, 'ML_MODEL.joblib')

```



```

"""## Loading the trained model"""

from joblib import load
#Loading the trained model
trained_model = load('ML_MODEL.joblib')

"""## Summarizing the results of all the models

"""

table = {
    'MLP Without SMOTE': [27,0,0.66,0.958],
    'SMOTE initial':[10,42,0.87,0.973],
    'ADASYN':[10,95,0.87,0.969],
    'BorderLine SMOTE':[14,40,0.82,0.982],
    'SVM SMOTE':[16,8,0.80,0.966],
    'SMOTE_FINE TUNED':[9,170,0.88,0.955]
}
columns = ['False Negatives', 'False Positives', 'Recall', 'ROC_AUC' ]
summary = pd.DataFrame.from_dict(table, orient = 'index', columns = columns)
summary
# False Negatives, False Positives, Recall, ROC_AUC,

"""## Plotting Graphs

### ROC_AUC_CURVE
"""

from sklearn.metrics import plot_roc_curve
plot_roc_curve(trained_model, X_test, Y_test)
plt.show()

"""## Comparing false negatives, recall and roc_auc_score"""

FN = sns.barplot( x = summary.index,y = summary["False Negatives"])
FN.set_xticklabels(FN.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()

recall = sns.barplot(x = summary.index, y = summary["Recall"])
recall.set_xticklabels(FN.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()

ROC_AUC = sns.barplot(x = summary.index, y = summary["ROC_AUC"])
ROC_AUC.set_xticklabels(FN.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()

```

```
plt.show()

"""The summary shows that SMOTE after fine tuning has the lowest False
Negatives and highest Recall. Even though the ROC_AUC score is slightly lesser
than the other models, due to the high Recall and low False Negatives, SMOTE
after finetuning is chosen as the best model

## Sources
https://machinelearningmastery.com/smote-oversampling-for-imbalanced-
classification/

https://medium.com/analytics-vidhya/credit-card-fraud-detection-in-depth-
study-evaluating-the-classification-model-
a3680a5a5897#:~:text=The%20most%20frequent%20metric%20used,situations%20like%2
0imbalanced%20class%20datasets.

https://scikit-learn.org/stable/modules/model\_persistence.html
"""
```

RESULTS AND DISCUSSION:

We have chosen MLP classifiers for training the model.

For the implementation with respect to pre-processing we have used MinMaxScaler() to scale the data for better accuracy.

WITHOUT SMOTE:

Building the model, without SMOTE:

Building the initial MLP model with the data which doesn't have SMOTE applied to it

```
In [133]: from sklearn.neural_network import MLPClassifier
initial_model = MLPClassifier(random_state=0)
initial_model.fit(X_train_without_SMOTE, Y_train_without_SMOTE)
predictions_initial_model = initial_model.predict(X_test)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,
```

Getting the evaluation of the initial model

```
In [134]: from sklearn.metrics import classification_report
print(classification_report(Y_test, predictions_initial_model))
```

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	29960
1.0	1.00	0.33	0.49	40
accuracy			1.00	30000
macro avg	1.00	0.66	0.75	30000
weighted avg	1.00	1.00	1.00	30000

```
In [135]: from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, predictions_initial_model)
```

```
Out[135]: array([[29960,  0],
 [ 27,  13]])
```

```
In [136]: from sklearn.metrics import roc_auc_score
prob_initial_model = initial_model.predict_proba(X_test)[:,1]
roc_auc_score(Y_test, prob_initial_model)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,
```

```
Out[136]: 0.9577920560747664
```

As can be seen, although the accuracy values are high this is only because the number of positive cases is very less, and as will be seen in the test-case scenario, the model performs very bad.

As the number of false negatives is 27/40, this causes a lot of loss, and the recall value is the lowest for this model.

As the number of fraudulent positive data is less, we have performed many over-sampling/under-sampling methods to work on the same.

They include:

- SMOTE()
- Random
- NearMiss

SMOTE:

SMOTE is an over-sampling technique, which we have used to over-sample the number of True Fraudulent datasets

```
In []: # Looking at the counts of the Y_train before using SMOTE
      Y_train.value_counts()
Out[]: 0.0    69905
      1.0     95
      Name: isFraud, dtype: int64

In []: from imblearn.over_sampling import SMOTE
      smote = SMOTE()
      X_sm, Y_sm = smote.fit_resample(X_train, Y_train)
      Y_sm.value_counts()
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,
```

Based on a test-MLP classifier model, we have found the following confusion matrix and classification report and roc-auc-score:

```
Building the Multi Layer Perceptron model

An initial multilayer perceptron is built

In []: from sklearn.neural_network import MLPClassifier
      model = MLPClassifier(random_state = 0)
      model.fit(X_sm, Y_sm)
      predictions = model.predict(X_test)
```

```
In []: from sklearn.metrics import classification_report
      print(classification_report(Y_test, predictions))
```

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	29960
1.0	0.42	0.75	0.54	40
accuracy			1.00	30000
macro avg	0.71	0.87	0.77	30000
weighted avg	1.00	1.00	1.00	30000

```

In [ ]: from sklearn.metrics import confusion_matrix
        confusion_matrix(Y_test, predictions)

Out[ ]: array([[29918,  42],
              [ 10,  30]])

In [ ]: from sklearn.metrics import roc_auc_score
        prob = model.predict_proba(X_test)[:,1]
        roc_auc_score(Y_test, prob)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,

Out[ ]: 0.97298564753004

```

Random Oversampling:

This is an over-sampling technique. Following a similar format for SMOTE, as mentioned before.

```

In [ ]: from imblearn.over_sampling import RandomOverSampler
        from collections import Counter
        rus = RandomOverSampler(random_state=42) # fit predictor and target variable
        x_rus, y_rus = rus.fit_resample(X_train, Y_train)

        print('original dataset shape:', Counter(Y_train))
        print('Resample dataset shape:', Counter(y_rus))

original dataset shape: Counter({0.0: 69905, 1.0: 95})
Resample dataset shape Counter({0.0: 69905, 1.0: 69905})

```

```

In [ ]: from sklearn.neural_network import MLPClassifier
        model = MLPClassifier(random_state = 0)
        model.fit(x_rus, y_rus)
        predictions_2 = model.predict(X_test)

```

```

In [ ]: from sklearn.metrics import classification_report
        print(classification_report(Y_test, predictions_2))

              precision    recall  f1-score   support

    0.0         1.00      0.99      1.00     29960
    1.0         0.17      0.80      0.28         40

 accuracy          0.99     30000
 macro avg       0.59      0.90      0.64     30000
 weighted avg     1.00      0.99      1.00     30000

In [ ]: from sklearn.metrics import confusion_matrix
        confusion_matrix(Y_test, predictions_2)

Out[ ]: array([[29806, 154],
              [  8,  32]])

In [ ]: from sklearn.metrics import roc_auc_score
        prob_2 = model.predict_proba(X_test)[:,1]
        roc_auc_score(Y_test, prob_2)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  FutureWarning,

Out[ ]: 0.9805949599465955

```

NearMiss;

This is an under-sampling technique. Following a similar structure to before.

```
In []: from imblearn.under_sampling import NearMiss

nm = NearMiss()

x_nm, y_nm = nm.fit_resample(X_train, Y_train)

print('Original dataset shape:', Counter(Y_train))
print('Resample dataset shape:', Counter(y_nm))

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,

Original dataset shape: Counter({0.0: 69905, 1.0: 95})
Resample dataset shape: Counter({0.0: 95, 1.0: 95})
```

```
In []: from sklearn.neural_network import MLPClassifier
model = MLPClassifier(random_state=0)
model.fit(x_nm, y_nm)
predictions_3 = model.predict(X_test)
```

```
In []: from sklearn.metrics import classification_report
print(classification_report(Y_test, predictions_3))
```

	precision	recall	f1-score	support
0.0	1.00	0.34	0.51	29960
1.0	0.00	0.93	0.00	40
accuracy			0.34	30000
macro avg	0.50	0.63	0.26	30000
weighted avg	1.00	0.34	0.51	30000

```
In []: from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, predictions_3)
```

```
Out[]: array([[10274, 19686],
               [  3,   37]])
```

```
In []: from sklearn.metrics import roc_auc_score
prob_3 = model.predict_proba(X_test)[:,1]
roc_auc_score(Y_test, prob_3)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,

Out[]: 0.728715787716956
```

As can be seen on comparison of the roc_auc_score, classification reports and the number of false positives and false negatives, SMOTE processing is the best and thus we will be going ahead with the same.

We have thus finalized to go with SMOTE() analysis and go in-depth with different variants of SMOTE to check for better accuracy.

We have performed-3 variants of SMOTE, namely:

- SVMSMOTE
- BorderlineSMOTE
- ADASYN Smote

For each of the SMOTE variants we have found done the respective pre-processing and then used gridsearchcv to find the best parameters and hidden layers and then use the same to generate the ML Model.

ADASYN:

Oversampling:

Choosing the SMOTE variant

As the data is highly imbalanced ADASYN SMOTE is used as an Oversampling technique for the training data

```
In [12]: # Looking at the counts of the Y_train before using SMOTE
Y_train.value_counts()
```

```
Out[12]: 0.0    69905
         1.0     95
         Name: isFraud, dtype: int64
```

```
In [13]: from imblearn.over_sampling import ADASYN
smote = ADASYN()
X_train, Y_train = smote.fit_resample(X_train, Y_train)
Y_train.value_counts()
```

```
Out[13]: 0.0    69905
         1.0    69890
         Name: isFraud, dtype: int64
```

Building the Multi Layer Perceptron model using GridSearchCV to get the best parameters for the hidden layer

```
In [14]: from sklearn.neural_network import MLPClassifier
         from sklearn.model_selection import GridSearchCV
         grid_params = {'hidden_layer_sizes': [(150, 100, 50), (120, 80, 40), (100, 50, 30)]}
         grid = GridSearchCV(MLPClassifier(random_state = 0), grid_params, scoring='recall')
         grid.fit(X_train, Y_train)
```

```
Out[14]: GridSearchCV(estimator=MLPClassifier(random_state=0),
                    param_grid={'hidden_layer_sizes': [(150, 100, 50), (120, 80, 40),
                    (100, 50, 30)]},
                    scoring='recall')
```

```
In [15]: grid.best_estimator_
```

```
Out[15]: MLPClassifier(hidden_layer_sizes=(150, 100, 50), random_state=0)
```

building the model with the best parameters

```
In [18]: model = MLPClassifier(hidden_layer_sizes=(150, 100, 50), random_state=0)
         model.fit(X_train, Y_train)
```

```
Out[18]: MLPClassifier(hidden_layer_sizes=(150, 100, 50), random_state=0)
```

```
In [19]: predictions = model.predict(X_test)
```

Evaluating the Initial Model

```
In [20]: from sklearn.metrics import classification_report
         print(classification_report(Y_test, predictions))
```

```
precision    recall  f1-score   support

0.0         1.00      1.00      29960
1.0         0.24      0.75      0.36      40

accuracy          1.00      30000
macro avg         0.62      0.87      0.68      30000
weighted avg      1.00      1.00      1.00      30000
```

```
In [21]: from sklearn.metrics import confusion_matrix
         confusion_matrix(Y_test, predictions)
```

```
Out[21]: array([[29865, 95],
               [10, 30]], dtype=int64)
```

```
In [22]: from sklearn.metrics import roc_auc_score
         prob = model.predict_proba(X_test)[::,1]
         roc_auc_score(Y_test, prob)
```

```
Out[22]: 0.9687633511348464
```

BorderlineSMOTE:

Following a similar structure as before:

As the data is highly imbalanced BorderlineSMOTE is used as an Oversampling technique for the training data

```
[ ] 1 # Looking at the counts of the Y_train before using BorderlineSMOTE
    2 Y_train.value_counts()

0.0    69905
1.0      95
Name: isFraud, dtype: int64
```

```
[ ] 1 # Using SMOTE variant-BorderlineSMOTE
    2 from imblearn.over_sampling import BorderlineSMOTE
    3 smote = BorderlineSMOTE()
    4 X_train, Y_train = smote.fit_resample(X_train, Y_train)
    5 Y_train.value_counts()

0.0    69905
1.0    69905
Name: isFraud, dtype: int64
```

Building the Multi Layer Perceptron model using GridSearchCV to get the best parameters for the hidden layer

```
[ ] 1 from sklearn.neural_network import MLPClassifier
    2 from sklearn.model_selection import GridSearchCV
    3 grid_params = {'hidden_layer_sizes': [(150,100,50), (120,80,40), (100,50,30)]}
    4 grid = GridSearchCV(MLPClassifier(random_state = 0),grid_params, scoring='recall')
    5 grid.fit(X_train, Y_train)

GridSearchCV(estimator=MLPClassifier(random_state=0),
              param_grid={'hidden_layer_sizes': [(150, 100, 50), (120, 80, 40),
              (100, 50, 30)]},
              scoring='recall')
```

```
[ ] 1 grid.best_estimator_

MLPClassifier(hidden_layer_sizes=(120, 80, 40), random_state=0)
```

building the model with the best parameters

```
[ ] 1 model = MLPClassifier(hidden_layer_sizes=(120, 80, 40), random_state=0)
    2 model.fit(X_train, Y_train)

MLPClassifier(hidden_layer_sizes=(120, 80, 40), random_state=0)

[ ] 1 predictions = model.predict(X_test)
```

building the model with the best parameters

```
[ ] 1 model = MLPClassifier(hidden_layer_sizes=(120, 80, 40), random_state=0)
    2 model.fit(X_train, Y_train)

MLPClassifier(hidden_layer_sizes=(120, 80, 40), random_state=0)

[ ] 1 predictions = model.predict(X_test)
```


▼ Evaluating the Best Model

```
1 from sklearn.metrics import classification_report
2 print(classification_report(Y_test,predictions))
```

```
              precision    recall  f1-score   support

    0.0         1.00      1.00      1.00     29960
    1.0         0.39      0.65      0.49         40

 accuracy          0.70      0.82      0.74     30000
 macro avg          1.00      1.00      1.00     30000
 weighted avg          1.00      1.00      1.00     30000
```

```
[ ] 1 from sklearn.metrics import confusion_matrix
     2 confusion_matrix(Y_test, predictions)
```

```
array([[29920,  40],
       [ 14,   26]])
```

```
[ ] 1 from sklearn.metrics import roc_auc_score
     2 prob = model.predict_proba(X_test)[::,1]
     3 roc_auc_score(Y_test,prob)
```

```
0.9823222630173565
```

SVMSMOTE:

Following a similar structure:

▼ Choosing the SMOTE variant

As the data is highly imbalanced SMOTE is used an Oversampling technique for the training data

```
1 # Looking at the counts of the Y_train before using SMOTE
2 Y_train.value_counts()
```

```
0.0    69905
1.0      95
Name: isFraud, dtype: int64
```

```
1 from imblearn.over_sampling import SVMSMOTE
2 smote = SVMSMOTE()
3 X_train, Y_train = smote.fit_resample(X_train, Y_train)
4 Y_train.value_counts()
```

```
0.0    69905
1.0   34203
Name: isFraud, dtype: int64
```

Building the Multi Layer Perceptron model using GridSearchCV to get the best parameters for the hidden layer

```
[ ] 1 from sklearn.neural_network import MLPClassifier
    2 from sklearn.model_selection import GridSearchCV
    3 grid_params = {'hidden_layer_sizes': [(150,100,50), (120,80,40), (100,50,30)]}
    4 grid = GridSearchCV(MLPClassifier(random_state = 0),grid_params, scoring='recall')
    5 grid.fit(X_train, Y_train)

GridSearchCV(estimator=MLPClassifier(random_state=0),
              param_grid={'hidden_layer_sizes': [(150, 100, 50), (120, 80, 40),
              (100, 50, 30)]},
              scoring='recall')
```

```
[ ] 1 grid.best_estimator_

MLPClassifier(hidden_layer_sizes=(120, 80, 40), random_state=0)
```

building the model with the best parameters

```
[ ] 1 model = MLPClassifier(hidden_layer_sizes=(120,80,40),random_state=0)
    2 model.fit(X_train, Y_train)

MLPClassifier(hidden_layer_sizes=(120, 80, 40), random_state=0)

[ ] 1 predictions = model.predict(X_test)
```

Evaluating the Initial Model

```
[ ] 1 from sklearn.metrics import classification_report
    2 print(classification_report(Y_test,predictions))

      precision    recall  f1-score   support

    0.0         1.00      1.00      1.00     29960
    1.0         0.75      0.60      0.67         40

 accuracy          1.00      30000
 macro avg         0.87      0.80      0.83      30000
 weighted avg         1.00      1.00      1.00      30000

[ ] 1 from sklearn.metrics import confusion_matrix
    2 confusion_matrix(Y_test, predictions)

array([[29952,    8],
       [  16,   24]])

[ ] 1 from sklearn.metrics import roc_auc_score
    2 prob = model.predict_proba(X_test)[:,1]
    3 roc_auc_score(Y_test,prob)

0.9662191255006676
```

Now the original SMOTE:

As the data is highly imbalanced SMOTE is used an Oversampling technique for the training data

```
In [128]: # Looking at the counts of the Y_train before using SMOTE
Y_train.value_counts()
```

```
Out[128]: 0.0    69905
          1.0      95
          Name: isFraud, dtype: int64
```

```
In [129]: from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=0)
X_train, Y_train = smote.fit_resample(X_train, Y_train)
Y_train.value_counts()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,
```

```
Out[129]: 0.0    69905
          1.0    69905
          Name: isFraud, dtype: int64
```

Building the Multi Layer Perceptron model

Based on finetuning a MLP model with hidden layer sizes = (50,50,50) is built

```
In [130]: from sklearn.neural_network import MLPClassifier
model = MLPClassifier(hidden_layer_sizes=(50, 50, 50), random_state=0)
model.fit(X_train, Y_train)
predictions = model.predict(X_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,
```

Evaluating the Model

```
In [131]: from sklearn.metrics import classification_report
print(classification_report(Y_test, predictions))
```

	precision	recall	f1-score	support	
	0.0	1.00	0.99	1.00	29960
	1.0	0.15	0.78	0.26	40
accuracy			0.99	30000	
macro avg	0.58	0.88	0.63	30000	
weighted avg	1.00	0.99	1.00	30000	

```
In [132]: from sklearn.metrics import confusion_matrix
confusion_matrix(Y_test, predictions)
```

```
Out[132]: array([[29790, 170],
                [ 9, 31]])
```

```
In [133]: from sklearn.metrics import roc_auc_score
prob = model.predict_proba(X_test)[::,1]
roc_auc_score(Y_test, prob)
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,

```
Out[133]: 0.9554981642189586
```

A final metric of .955 for roc auc is achieved.

We will go with this SMOTE model, as we get the minimum false negative and a low false positive, thus we have done our final model with this SMOTE processing.

COMPARIOSONS:

SUMMARIZATION TABLE:

Out[155]:

	False Negatives	False Positives	Recall	ROC_AUC
MLP Without SMOTE	27	0	0.66	0.958
SMOTE initial	10	42	0.87	0.973
ADASYN	10	95	0.87	0.969
BorderLine SMOTE	14	40	0.82	0.982
SVM SMOTE	16	8	0.80	0.966
SMOTE_FINE TUNED	9	170	0.88	0.955

GRAPHS:

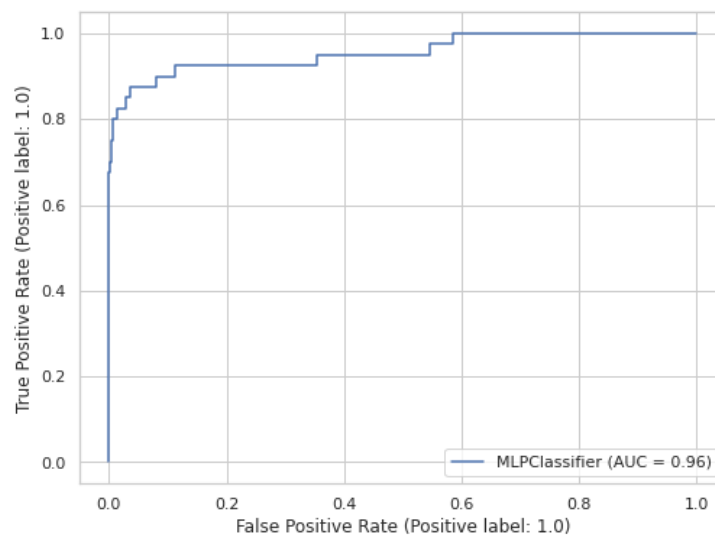
(for smote)

Plotting Graphs

ROC_AUC_CURVE

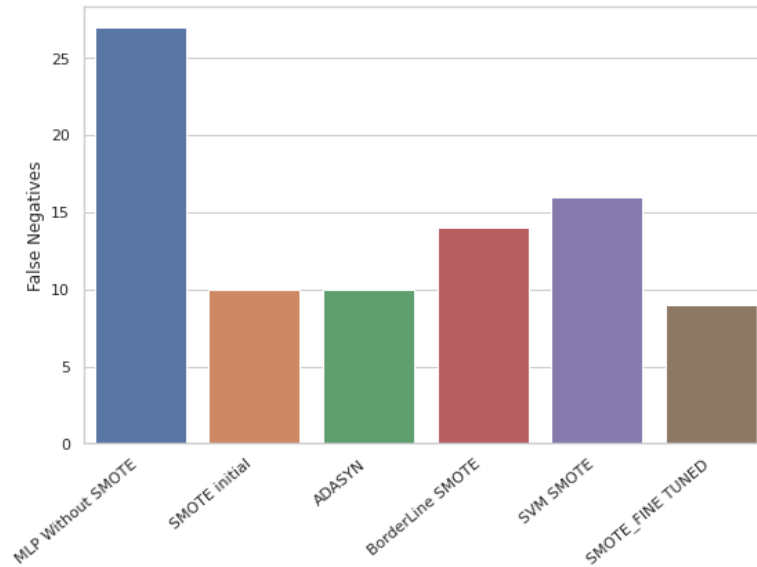
```
In [156]: from sklearn.metrics import plot_roc_curve  
plot_roc_curve(trained_model, X_test, Y_test)  
plt.show()
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_roc_curve is deprecated; Function :func:'plot_roc_curve' is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:'sklearn.metrics.RocCurveDisplay.from_predictions' or :meth:'sklearn.metrics.RocCurveDisplay.from_estimator'.
warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,

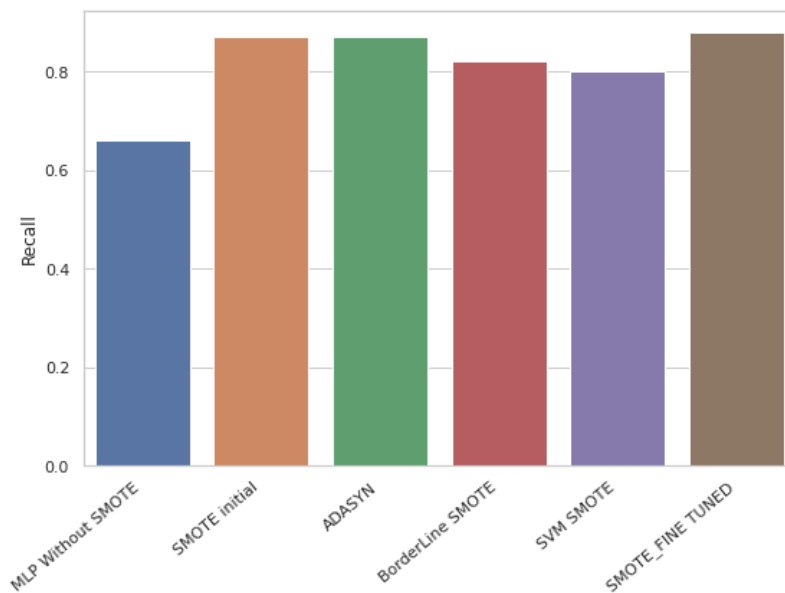


Comparing false negatives, recall and roc_auc_score

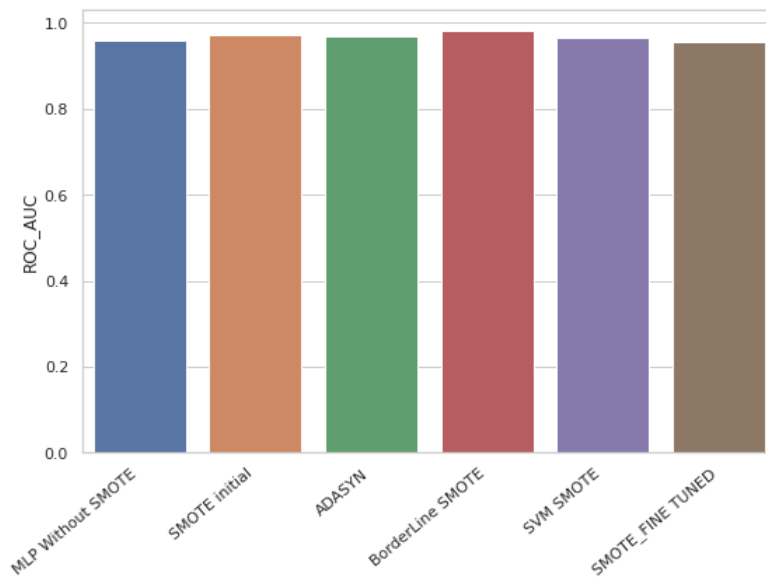
```
In [157]: FN = sns.barplot(x = summary.index, y = summary["False Negatives"])
FN.set_xticklabels(FN.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```



```
In [158]: recall = sns.barplot(x = summary.index, y = summary["Recall"])
recall.set_xticklabels(recall.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```



```
In [159]: ROC_AUC = sns.barplot(x = summary.index, y = summary["ROC_AUC"])
ROC_AUC.set_xticklabels(FN.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```



The summary shows that SMOTE after fine tuning has the lowest False Negatives and highest Recall. Even though the ROC_AUC score is slightly lesser than the other models, due to the high Recall and low False Negatives, SMOTE after finetuning is chosen as the best model

CONCLUSION:

The summary shows that SMOTE after fine tuning has the lowest False Negatives and highest Recall. Even though the ROC_AUC score is slightly lesser than the other models, due to the high Recall and low False Negatives, SMOTE after finetuning is chosen as the best model.

Based on the results and discussions made, we have concluded to use the SMOTE Model, with the following metrics

As the data is highly imbalanced SMOTE is used as an Oversampling technique for the training data

```
In [128]: # Looking at the counts of the Y_train before using SMOTE
Y_train.value_counts()
```

```
Out[128]: 0.0    69905
          1.0      95
          Name: isFraud, dtype: int64
```

```
In [129]: from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=0)
X_train, Y_train = smote.fit_resample(X_train, Y_train)
Y_train.value_counts()
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,
```

```
Out[129]: 0.0    69905
          1.0    69905
          Name: isFraud, dtype: int64
```

Building the Multi Layer Perceptron model

Based on finetuning a MLP model with hidden layer sizes = (50,50,50) is built

```
In [130]: from sklearn.neural_network import MLPClassifier
model = MLPClassifier(hidden_layer_sizes=(50, 50, 50), random_state=0)
model.fit(X_train, Y_train)
predictions = model.predict(X_test)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,
```

Evaluating the Model

```
In [131]: from sklearn.metrics import classification_report  
print(classification_report(Y_test, predictions))
```

	precision	recall	f1-score	support	
	0.0	1.00	0.99	1.00	29960
	1.0	0.15	0.78	0.26	40
accuracy			0.99	30000	
macro avg	0.58	0.88	0.63	30000	
weighted avg	1.00	0.99	1.00	30000	

```
In [132]: from sklearn.metrics import confusion_matrix  
confusion_matrix(Y_test, predictions)
```

```
Out[132]: array([[29790, 170],  
                [ 9, 31]])
```

```
In [133]: from sklearn.metrics import roc_auc_score  
prob = model.predict_proba(X_test)[:,1]  
roc_auc_score(Y_test, prob)
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
FutureWarning,

```
Out[133]: 0.9554981642189586
```

A final metric of .955 for roc auc is achieved.

REFERENCES:

Link to the dataset: <https://www.kaggle.com/datasets/ealaxi/paysim1>

<https://thecleverprogrammer.com/2020/08/04/fraud-detection-with-machine-learning/>

[https://spd.group/machine-learning/fraud-detection-with-machine-learning/#Credit Card Fraud Detection with Machine Learning](https://spd.group/machine-learning/fraud-detection-with-machine-learning/#Credit%20Card%20Fraud%20Detection%20with%20Machine%20Learning)

<https://www.investopedia.com/terms/n/neuralnetwork.asp>

<https://www.analyticsvidhya.com/blog/2021/03/basics-of-neural-network/>

<https://www.analyticsvidhya.com/blog/2020/12/an-overview-of-neural-approach-on-pattern-recognition/>

<https://support.sas.com/documentation/onlinedoc/stat/141/hpsplit.pdf>

<https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/>

<https://analyticsindiamag.com/a-beginners-guide-to-scikit-learns-mlpclassifier/>

<https://towardsdatascience.com/how-to-sample-a-dataframe-in-python-pandas-d18a3187139b>

<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1056.6194&rep=rep1&type=pdf>

[https://www.researchgate.net/publication/347441357 THE ROLE OF RANDOM FOREST IN CREDIT CARD FRAUD ANALYSIS](https://www.researchgate.net/publication/347441357_THE_ROLE_OF_RANDOM_FOREST_IN_CREDIT_CARD_FRAUD_ANALYSIS)

[https://www.researchgate.net/profile/Salma-El-Hajjami/publication/352834391 Machine Learning System for Fraud Detection A Methodological Approach for a Development Platform/links/619791e161f0987720b33c6c/Machine-Learning-System-for-Fraud-Detection-A-Methodological-Approach-for-a-Development-Platform.pdf](https://www.researchgate.net/profile/Salma-El-Hajjami/publication/352834391_Machine_Learning_System_for_Fraud_Detection_A_Methodological_Approach_for_a_Development_Platform/links/619791e161f0987720b33c6c/Machine-Learning-System-for-Fraud-Detection-A-Methodological-Approach-for-a-Development-Platform.pdf)

<https://asianssr.org/index.php/ajct/article/view/1151>

[https://www.researchgate.net/publication/347411268 Finance Fraud Detection With Neural Network](https://www.researchgate.net/publication/347411268_Finance_Fraud_Detection_With_Neural_Network)

<https://www.sciencedirect.com/science/article/pii/S187705092030065X>

<https://www.hindawi.com/journals/scn/2018/5483472/>

<https://www.sciencedirect.com/science/article/pii/S2666675821001016>

[https://www.researchgate.net/publication/331308198 Performance of machine learning techniques in the detection of financial frauds](https://www.researchgate.net/publication/331308198_Performance_of_machine_learning_techniques_in_the_detection_of_financial_frauds)

[https://medium.com/analytics-vidhya/balance-your-data-using-smote-98e4d79fcddb#:~:text=SMOTE%20proceeds%20by%20joining%20the,in%20the%20SMOTE\(\)%20function](https://medium.com/analytics-vidhya/balance-your-data-using-smote-98e4d79fcddb#:~:text=SMOTE%20proceeds%20by%20joining%20the,in%20the%20SMOTE()%20function)

<https://zhuanlan.zhihu.com/p/99618155>

References for MLP

1. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* 521, 436–444 (2015)
2. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. The MIT Press.
3. McCulloch, W.S., Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* 5, 115–133 (1943)
4. Frank Rosenblatt. The Perceptron, a Perceiving and Recognizing Automaton Project Para. *Cornell Aeronautical Laboratory* 85, 460–461 (1957)
5. Minsky M. L. and Papert S. A. 1969. *Perceptrons*. Cambridge, MA: MIT Press.
6. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. (2013). An introduction to statistical learning : with applications in R. New York :Springer
7. D. Rumelhart, G. Hinton, and R. Williams. Learning Representations by Back-propagating Errors. *Nature* 323 (6088): 533–536 (1986).

References for SMOTE:

1. Chawla NV, Bowyer KW, Hall LO, Kegelmeyer WP. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*. 2002 Jun 1;16:321–57.
2. Fernández A, García S, Herrera F, Chawla NV. SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary. *Journal of artificial intelligence research*. 2018 Apr 20;61:863–905.