CECS 632 Data Mining

Project 2 : Credit Card Fraud Detection

Aliaa 10/20/19 Fall 19

1. Abstract:

The credit card has become the crucial mode of payment. Therefore, with the rise in the credit card transactions, the credit card frauds have also become frequent nowadays. Invention of credit cards has made online transactions seamless, easier, comfortable and convenient. However, it has also provided new fraud opportunities for criminals, and in turn, increased fraud rate. Credit card fraud is when someone uses another person's credit card or account information to make unauthorized purchases or access funds through cash advances. Credit card fraud doesn’t just happen online; it happens in brick-and-mortar stores, too. As a business owner, you can avoid serious headaches – and unwanted publicity – by recognizing potentially fraudulent use of credit cards in your payment environment.

The global impact of credit card fraud is alarming, millions of US dollars have been lost by many companies and individuals. Fraud detection is a challenging problem. The fact is that fraudulent transactions are rare; they represent a very small fraction of activity within an organization. The challenge is that a small percentage of activity can quickly turn into big dollar losses without the right tools and systems in place. Criminals are crafty. As traditional fraud schemes fail to pay off, fraudsters have learned to change their tactics. Thus, an improved fraud detection system has become essential to maintain the reliability of the payment system. The criterion is to assure secured transactions credit card owners so that they can make electronic payment safely for the services and goods which are provided on internet. Therefore, banks and financial institutions offer credit card fraud detection applications much value and demand. Fraudulent transactions can occur in various ways and can be put into different categories. The combination of machine learning and data mining techniques were able to identify fraud and non-fraud transactions by learning the patterns of the data.

The dataset provided is for predicting whether the credit transaction history is fraud or not. It has 434 features. The dataset contains 2 classes under the isFraud column: 0 as no fraud and 1 as fraud. The aim is to build a classification model to predict the isFraud column. The data provided is separated into training and testing. A main problem of data that data is are highly skewed data (where many more of data is legitimate, and a few of them is fraudulent)

In this project, I have used three different models for classification task along with a lot of preprocessing steps to prepare the data for mining processes. The first model is based on a random forest classifier which results were good enough with f- score 0.59. The second model is based on Adaboost classifier with /without base estimator which gives a f- score near 0.5 but has so long execution time. The last model I proposed in the project is a bagging classifier based on KNN as a base estimator which results were the lowest.

The hyper parameter tuning of all these models was based on random search algorithm to find out the optimum parameters for the models proposed here

2. Credit Card Fraud Detection:

2.1 Problem description

Credit card companies have a vested interest in identifying financial transactions that are illegitimate and criminal in nature. The stakes are high. According to the *Federal Reserve Payments Study*, Americans used credit cards to pay for 26.2 billion purchases in 2012. The estimated loss due to unauthorized transactions that year was $6.1 billion. The *federal Fair Credit Billing Act* limits the maximum liability of a credit card owner to $50 for unauthorized transactions, leaving credit card companies on the hook for the balance. Obviously fraudulent payments can have a big effect on the companies' bottom lines. The industry requires any vendors that process credit cards to go through security audits every year. But that doesn't stop all fraud.

Traditionally, detecting fraud relied on data analysis techniques that required significant human involvement. An algorithm would flag suspicious cases to be closely reviewed ultimately by human investigators who may even have called the affected cardholders to ask if they'd made the charges. Nowadays the companies are dealing with a constant deluge of so many transactions that they need to rely on big data analytics for help. Emerging technologies such as machine learning and cloud computing are stepping up the detection game

A machine learning algorithm for fraud detection needs to be trained first by being fed the normal transaction data of lots and lots of cardholders. Transaction sequences are an example of this kind of training data. A person may typically pump gas one time a week, go grocery shopping every two weeks, and so on. The algorithm learns that this is a normal transaction sequence. After this fine-tuning process, credit card transactions are run through the algorithm, ideally in real time. It then produces a probability number indicating the possibility of a transaction being fraudulent. If the fraud detection system is configured to block any transactions whose score is above, this assessment could immediately trigger a card rejection at the point of sale. The algorithm considers many factors to qualify a transaction as fraudulent: trustworthiness of the vendor, a cardholder's purchasing behavior including time and location, IP addresses, etc. The more data points there are, the more accurate the decision becomes.

Data sets are only growing larger, and as the volumes increase, so does the challenge of detecting fraud. In fact, data is key when it comes to building machine learning systems. The adage that more data equals better models is true when it comes to fraud detection. Practitioners need their machine learning platform to scale as data and complexity increase. While academic tools often work well with thousands of records and a few megabytes of data, real-world problems are measured in gigabytes or even terabytes of data. There is no single machine learning algorithm or method that works. Success comes from the ability to try lots of different machine learning-based methods, trying variations on them and testing them with a variety of data sets.

The data scientist needs a toolkit with a variety of supervised and unsupervised methods – as well as a variety of feature engineering techniques. Finally, there is a creative aspect or "art" to machine learning for fraud detection. It's the application of machine learning in new and novel ways, like combining a variety of supervised and unsupervised methods in one system to be more effective than any single method alone.

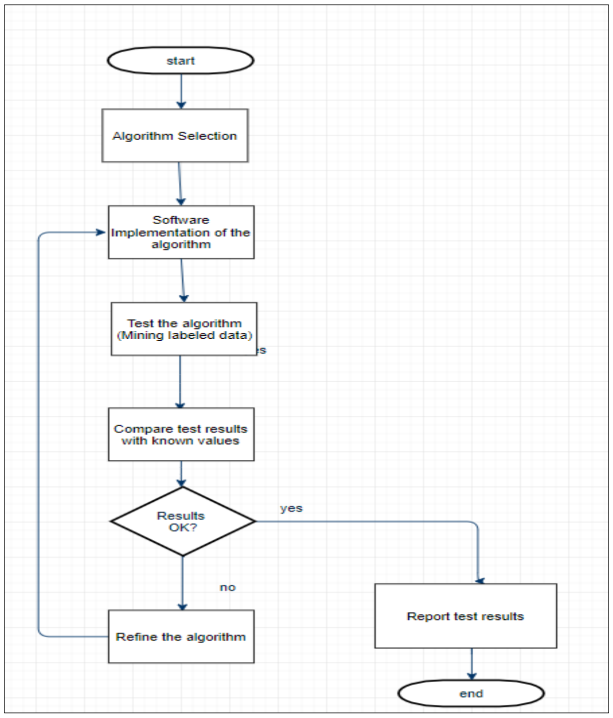

start
Algorithm Selection
Software Implementation of the algorithm
Test the algorithm (Mining labeled data)
Compare test results with known values
Results OK?
yes
no
Refine the algorithm
Report test results
end

Figure 1 Flow Chart of machine learning approach

Furthermore, the problem of credit card fraud detection has many constraints. There are various challenges in credit card fraud detection are: *Non availability of a real data set*, *Unbalanced Data Set, Size of data set ,Determining the appropriate evaluation parameters and Dynamic behavior of fraudster.*

2.2 Review of publications:

2.2.1 Application of Credit Card Fraud Detection: Based on Bagging Ensemble Classifier [1]

The objective of this paper is to examine the evaluation performance of three advanced data mining techniques: Support Vector machines (SVM), Naïve Bayes (NB), KNN and Bagging ensemble classifier.
Bagging classifier is an ensemble technique which was proposed by Leo Breiman in 1994. It can handle classification and regression methods. It is designed to improve the stability and accuracy of machine learning algorithms used in classification and regression. It works by combining classifications of randomly generated training sets to form a final prediction. Such techniques can typically be used as a variance reduction technique by randomization into its construction procedure and then creating an ensemble out of it. Bagging classifier has attracted much attention, due to its simple implementation and the improving accuracy. Thus, we can call bagging as a "smoothing operation" that has advantage when intending to improve the predictive performance of regression or classification trees. The basic principle behind of this ensemble method is that a group of "weak learners" can come together to form a "strong learner". Bagging grows many decision trees. Here each individual decision tree is a "weak learner", while all the decision trees taken together are a "strong learner". When a new instance has to be
classified, it is done repeatedly to each of the trees in the ensemble.

In this paper they used bagging classifier, with the decision tree algorithm J48 based on the C4.5 model as the single classifier to construct the ensemble. The reason for selecting decision three as a single classifier for our ensemble is that, the dataset is highly unbalanced, so decision three algorithm presents a very good behavior by weighting the results of the trees and reducing the variance of the dataset and the overfitting.

The objective of this paper is to examine the performance's evaluation of three advanced data mining techniques, with the well-known and proposed bagging ensemble classifier, for credit card fraud detection technique. In this work the paper used 10-fold cross validation techniques.

The obtained dataset includes of 100,000 records of credit card transactions. Each record has 20 fields. The data given to us was already labeled by bank, as legitimate and fraudulent. The ratio of legitimate transactions to fraudulent transactions approximately is 100:3. In this paper they didn't consider some of common metrics like accuracy and error rate, since they are known to be bias metrics in the case of unbalanced dataset. For fraud detection domain, the "fraud catching rate" and "false alarm rate" are the criteria metrics. We are evaluated the performance of the various techniques in terms of four classification metrics relevant to credit card fraud detection Fraud Catching Rate, False Alarm Rate, Balanced Classification Rate and Matthews Correlation Coefficient.

The paper's results of the evaluation of performance of the model developed from the dataset is a comparison between the three classifiers mentioned above with the bagging ensemble classifier. Bagging classifier based on decision three, very well in detecting the fraudulent transactions, the fraud catching rate is high and the false alarm rate very low. While other methods have problem to detect the fraudulent transactions by increasing the number of false alarm rate.

In this paper, they found that, the bagging classifier based on decision three works well with this kind of data since it is independent of attribute values. The second feature of this technique in credit card fraud detection is its ability to handle class unbalance. This is incorporated in the model by creating four subsets of the dataset (Df1, Df2, Df3, DF4) which the fraud rate in each of them were 20%, 15%, 10%, 3% respectively. Bagging classifier-based decision three algorithm performance is found to be stable gradually during the evaluation. Moreover, the bagging ensemble method takes very less time, which is also an important parameter of this real time application, because in fraud detection domain time is known one of the important parameters

2.2.2 Application of Hidden Markov Model in Credit Card Fraud Detection [2]

Hidden Marcov Field (HMM) is one of the most popular credit card fraudulent Detection techniques
It is one of the best engineering practices tool for credit card fraud system. Hidden markov model generate, observation symbols for online transaction. Observation probabilistic in an HMM based system initially studies spending profile of the cardholder and checking an incoming transaction, against spending behavior of the cardholder. we can show clustering model is used to classify the legal and fraudulent transaction using data conglomeration of regions of parameter. This paper has proposed application of hidden markov model in credit card fraud detection. Markov process is showing directly initial state and transaction state, but in HMM does not show directly transaction states it provides the observation state of initial state according its observation state transaction is succeeded. The Hidden Markov Model is a finite set of states, each of which is associated with a probability distribution. Transitions among the states are governed by a set of probabilities called transition probabilities. In a state an outcome or observation can be generated, according to the associated probability distribution. It is only the outcome, not the state visible to an external observer and therefore states are "hidden" to the outside; hence the name Hidden Markov Model.

The important benefit of the HMM-based approach is an extreme decrease in the number of False Positives transactions recognized as malicious by a fraud detection system even though they are genuine.
In this fraud detection system, the paper creates through original deciding the inspection symbols in representation. It considers buying values x into M price ranges V_1, V_2 . . .V_M, form the study symbols by the side of the issuing bank. The genuine price variety for each symbol is configurable based on the expenditure routine of personal cardholders. HMM determine these prices rang dynamically by using clustering algorithms (like K clustering algorithm) on the price values of every card holder transactions. It uses cluster V_k for clustering algorithm
as k = 1, 2 . . . .M, which represented both observations on price value symbols as well as on price value range In that prediction process ,it considers three different spending profiles of the card holder, which is depending upon price range, named high (h), medium (m) and low (l). In this set of symbols, we define V = {l, m, h} which makes M =3. The price range of proposed symbols has taken as low (0, $100], medium ($100, $500] and high ($500, up to credit card limit]. After finalizing the state and symbol representations, the next step is to determine different components of the HMM. The initial choice of parameters affects the performance of this algorithm and, hence, it is necessary to choose all these parameters carefully. we calculate probability of each spending profile (h, l and m). The most important thing is to estimate HMM parameters for each card holder. The forward- backward algorithm [13] starts with initial HMM parameters and converges to the nearest likelihood values.

In this paper, it has been shown that HMM initially checks the upcoming transaction is fraudulent or not. It also takes decision to add new upcoming transaction to existing sequence or not which will be dependent on percentage change in probabilities of old and new sequence. It will decide whether this transaction is genuine or fraudulent depending on threshold values. We have categorized different types of items and services such as restaurant, bill payment etc. These different categories have been considered as three different states of the Hidden Markov Model. In each category, they have further divided into three different groups, high, medium and low based on different ranges of transaction amount. These groups were considered as observation symbols. This technique helps to find the spending behavioral habit of cardholders and purchasing of different items. The most important application of this technique is to decide initial value of observation symbols, probability of transition states and initial estimation of the model parameters. In that proposed model, it was found out more than 88% transactions are genuine and very low false alarm which is about 8 % of total number of transactions.

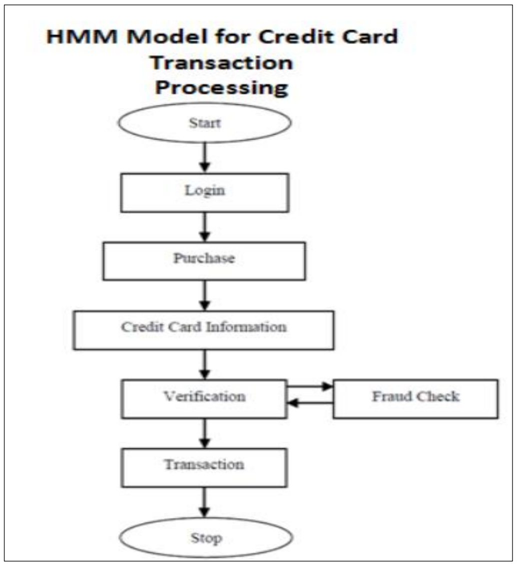


Figure 2 Flowchart of HMM module for credit card fraudulent detection

2.2.3 Credit card fraud detection by dynamic incremental semi-supervised fuzzy clustering [3]

In this paper, they propose a new approach for credit card fraud detection using a semi-supervised classification task on a data stream. The DISSFCM algorithm is applied, which is based on Dynamic Incremental Semi-Supervised Fuzzy C-Means that processes data grouped in small-size chunks.
Due to the dynamic nature of credit card transaction data, incremental learning, rather than static learning, is suitable for fraud detection. Incremental learning considers data as a continuous stream and processes each new instance on arrival. In this context it is important to preserve the previously acquired knowledge as well as to properly update it as new observations arrive. When the data distribution changes, the classifier should be able to learn from a new fraud distribution and forget outdated knowledge. Unsupervised methods do not require the prior knowledge on class transactions and are oriented to detect new fraudulent behaviors or unusual transactions an advantage of using unsupervised methods over supervised methods is that previously undiscovered types of fraud may be detected with no need of labeled data on past transactions. Unsupervised learning is suitable when the behavior of fraudsters is evolving, and we need to identify new patterns or anomalies (outliers).

The goal is to create a data stream classifier by exploiting partial supervision when available. To this aim, we leverage a fuzzy clustering process that considers the partial availability of class labels among samples. The method was successively refined by enabling the dynamic determination of the number of clusters through a splitting procedure, leading to the DISSFCM (Dynamic Incremental Semi-Supervised FCM) algorithm.

DISSFCM assumes that data belonging to C different classes are continuously available during time and processed as chunks. Namely, a chunk of N_1 data is available at time t_1, a chunk of $_{N2}$ data is available at t_2 and so on 3. We denote by X_t the data chunk available at time t. No assumption is made on the dimension of chunks that may vary from one chunk to another. One key feature of DISSFCM is the possibility to exploit partial supervision when available. Namely, when some pre-labeled data are available in a chunk, their labels can be used to drive the clustering process. The presence of pre-labeled data is not mandatory, but it should be assured in the first chunk in order to initialize properly the cluster prototypes.

The DISSFCM algorithm was applied on a publicly available dataset of real credit card transactions. Transaction records were gathered over a period of about 48 hours. Every transaction has a timestamp, a monetary amount, and 28 other real-valued, anonymized features. The dataset contains 284,807 transactions, of which 492 were fraudulent. Hence it is highly unbalanced, with the positive class (frauds) accounting for 0.172% of all transactions.

The results show that DISSFCM can cope with some challenges characterizing fraud detection: 1) handling the class imbalance, since legitimate transactions far outnumber the fraudulent ones and 2) operating with a small number of recent transactions, that may be partially labeled in the form of investigators feedback. DISSFCM gives comparable results to other incremental methods, and it adds the advantage of dealing with partially labeled data without a significant decay of performance. Differently from boosted methods, which are more accurate but less interpretable, DISSFCM creates the classification model as a simple collection of cluster prototypes that are easy to read and understand. This is an added value because interpretability plays an important role in the context of credit card fault detection.

2.3 Software tools:

For this project Python is used in both preprocessing and data mining process. Here is a list of packages that I used in the project with a brief description.

- **Pandas Library**

is a Python package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. The two primary data structures of pandas, Series (1-dimensional) and DataFrame (2-dimensional), handle many typical use cases in finance, statistics, social science, and many areas of engineering.

- **Matplotlib Library**

is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

- **Seaborn Library**

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures

- **Scikit-learn**

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, Classification, Clustering, Model selection and Preprocessing.

- **Pickle**

Python's pickle module is for: it serializes objects so they can be saved to a file and loaded in a program again later. It is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network. Later, this character stream can then be retrieved and de-serialized back to a Python object

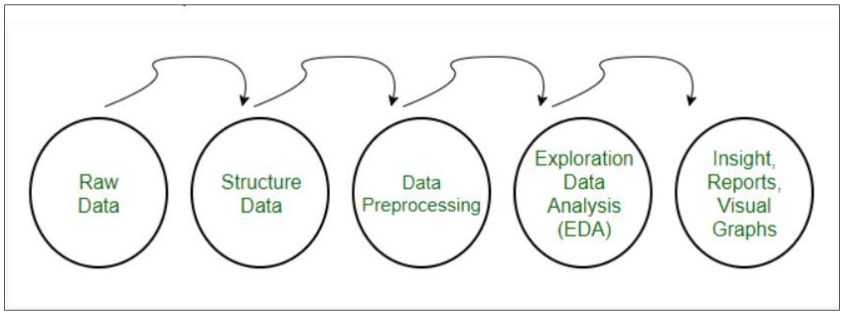
- **Imblearn**

imbalanced-learn is a python package offering several re-sampling techniques commonly used in datasets showing strong between-class imbalance. It is compatible with scikit-learn projects.

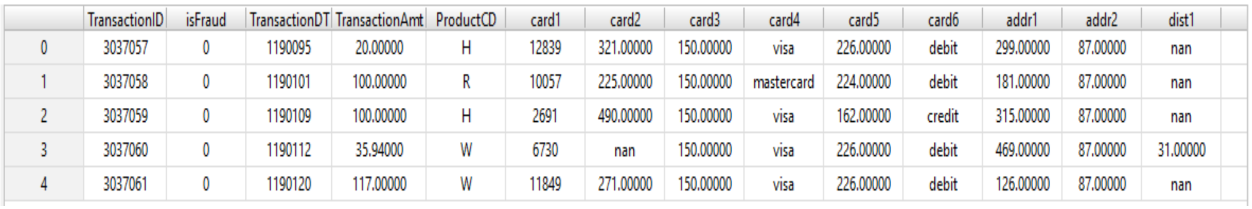
3.4 Preprocessing:

For this project, the first step is to identify the data, or the sources of information, and from that we will be able to determine what information should be studying to retrieve data from. Previously we mentioned that our target in this project to predict whether the transaction is fraud or not based on some classification models. Now, we investigate the training and test datasets how they are look like and try to identify descriptions of some features and explore the feature to make brain storming about observation from the data. The aim is to answer some questions like

- what type of data we have on our dataset?
- How many features, samples, percentage of missing data?
- What is the target distribution?


Raw Data
Structure Data
Data Preprocessing
Exploration Data Analysis (EDA)
Insight, Reports, Visual Graphs

The first csv file fraud.csv contains 540483 sample with 434 feature and the second csv file is fraud_test.csv has 50056 rows (samples). The training data as we mentioned earlier is labeled whether this credit card transaction is fraud or not.



	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	card4	card5	card6	addr1	addr2	dist1
0	3037057	0	1190095	20.00000	H	12839	321.00000	150.00000	visa	226.00000	debit	299.00000	87.00000	nan
1	3037058	0	1190101	100.00000	R	10057	225.00000	150.00000	mastercard	224.00000	debit	181.00000	87.00000	nan
2	3037059	0	1190109	100.00000	H	2691	490.00000	150.00000	visa	162.00000	credit	315.00000	87.00000	nan
3	3037060	0	1190112	35.94000	W	6730	nan	150.00000	visa	226.00000	debit	469.00000	87.00000	31.00000
4	3037061	0	1190120	117.00000	W	11849	271.00000	150.00000	visa	226.00000	debit	126.00000	87.00000	nan

Figure 3 Sample of training data

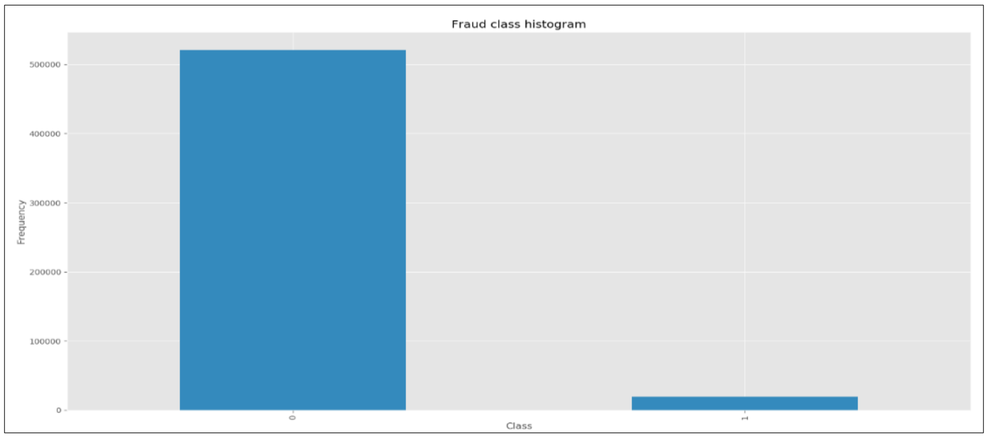


Figure 4 Fraud Class histogram

Understanding the target distribution:

The first question arises here how many fraudulent transactions in this large-scale dataset, we figured out that out of 540483 transaction, there are 19306 fraudulent transaction i.e. 3.5 % of the data is fraud. Therefore, the data is highly imbalanced. **Imbalance** means that the number of data points available for different the classes is different Most machine learning algorithms work best when the number of samples in each class are about equal. This is because most algorithms are designed to maximize accuracy and reduce error. So, dealing with the data like this with any dummy classifier we will have accuracy almost 99 % which is totally wrong. After data exploration we discuss how we are going to deal with unbalanced data.

Next, we need to find the **data description**:

- **TransactionDT:** timedelta from a given reference datetime (not an actual timestamp)
- **TransactionAMT**: transaction payment amount in USD
- **ProductCD**: product code, the product for each transaction
- **card1** - card6: payment card information, such as card type, card category, issue bank, country, etc.
- **addr**: address
- **dist**: distance
- **P_ and (R__) emaildomain**: purchaser and recipient email domain
- **C1-C14**: counting, such as how many addresses are found to be associated with the payment card, etc. The actual meaning is masked.
- **D1-D15**: timedelta, such as days between previous transaction, etc.
- **M1-M9**: match, such as names on card and address, etc.
- **Vxxx**: Vesta engineered rich features, including ranking, counting, and other entity relations.
- **Idxx**: masked identity information
- **<u>Categorical Features</u>**:
    - ProductCD
    - card1 - card6
    - addr1, addr2
    - P_emaildomain R_emaildomain
    - M1 - M9
    - DeviceType
    - DeviceInfo
    - id12 - id38

Now, we have listed all the feature in the provided dataset for this project, there are common issues we need to watch out before we build a model.

a) Attribute formatting:

This is provided with the dataset in a separated csv file, the datatype for each feature in our dataset.

b) Missing data to be dropped

Handling missing values is the important step while building your model. It will impact the result if not handled well. The missing values occur in data due to many reasons, such as problems occurred during extraction or data collection process. So, there are many ways you can treat missing values appropriately.

Deletion: List wise and Pair wise, we can delete the missing values using these two methods. In list wise method you need to erase complete row which contains missing value and pair wise method will do analysis by neglecting the missing value and considering only the available values.

Imputation: we can also choose imputation method to handle missing data in such cases that I want to replace them. I truly believe that there is no best way to deal with missing, especially when having to deal with partial information. Knowing which columns could be imputed or dropped may alter the result of the final predictions by a non-trivial amount

There are 414 columns in train dataset with missing values. Most of columns have missing data, which is normal in real world. Also there are columns with one unique value (or all missing).For my design I will compare first the filling factor of all features in both training set and testing dataset, If the percentage of missing values in each feature in both are almost the same (± 5%) then I will drop that feature. This test is performed in python by using the function that count the null values in panadas data frame.

From training and testing data sets, the columns with more that **90%** missing are :
id_07,id_08,id_21,id_22,id_23,id,24,id_25,id_26,id_27,D7,dist2,D6,D14 I will select to drop the columns that has missing value more than 90% .I believe this high amount if missing values in both train and test are not useful in building a model especially that my models don't handle the null values.

Next step is to find columns with most repeated values in training and testing datasets ,in other words ,most of the samples has the same value for these features and in case of categorical columns they belong to the same category which make these features are not useful in building a model ,moreover they may be a cause of building a wrong model. These features are:
'C3', 'V98', 'V101', 'V102', 'V103', 'V104', 'V105', 'V106', 'V107', 'V108', 'V109', 'V110', 'V111', 'V112', 'V113', 'V114', 'V115', 'V116', 'V117', 'V118', 'V119', 'V120', 'V121', 'V122', 'V123', 'V124', 'V125', 'V129', 'V132', 'V133', 'V134', 'V135', 'V136', 'V137', 'V281', 'V284', 'V286', 'V290', 'V293', 'V295', 'V296', 'V297', 'V298', 'V299', 'V300', 'V301', 'V305', 'V309', 'V311', 'V316', 'V318', 'V319', 'V320', 'V321' . These previous features don't have any variability.

Also, I have made a check to find any feature that always have a unique value in training or testing datasets, It shows that there are no columns in the train dataset with one unique value, On the other hand, there are 2 columns in the test dataset with one unique value. These two columns are: **M1 and V107**
useful. I got the list of such columns for train and test separately.
Moreover, we know that the column **TransactionID** is unrelated to predicting process.
These are totally 76 features will be dropped due to the reasons mentioned above.

Next, before we deal with other missing data that are less than 90% , We need to explore some features, I will start exploration on categorical features and transaction amounts my initial hypothesis is that there will be a significance difference between transaction amount in fraud samples and non-fraud ones. The question I am trying to answer are:

- What is the distribution of the transactions' values of fraud and non-fraud transactions?
- For Product Category in **ProductCD** feature Is there exist a dominated fraudulent product category?
- What features of target show any patterns?

Starting with the **TransactionAmt :**
The amount of credit card transaction. I have used the log transform in these plots to better show the distribution- otherwise the few, very large transactions skew the distribution. Because of the log transform, any values between 0 and 1 will appear to be negative.

We can observe that the average of fraud transactions seems to have a higher average transaction amount.
Mean **TransactionAmt** for fraud transactions is 149.48 and for non-fraud is 135.15.

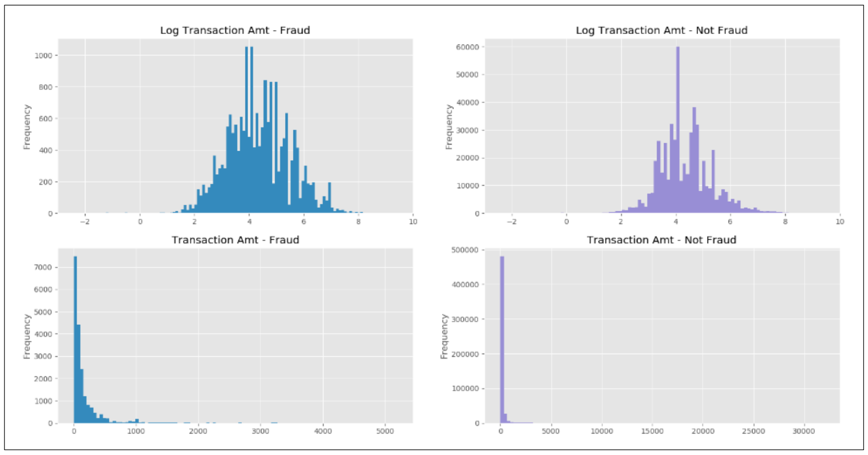


Figure 5 TransactionAmt feature among target

However, it is possible that the fraudulent charges with less amount are not discovered, since the people who have been frauded is less willing to record it. So, <u>this observation is not good enough.</u>

Next, I will explore some categorical features in the data set to figure out any important pattern or observation in the dataset. Start with **ProductCD** feature:
First, this column does not have any missing values in the data in both training and testing datasets, which it seems to be an important feature. Moreover, this column has only five distinct categories (**W, S, R, H, C**) we don't have any interpretation for what these letters stand for. From figure 5 we notice that the largest frequency is for products in **W** category, also, we observe that most of the fraudulent transaction are in category **C** and **W.**
Product W takes up 60% of fraud cases for transactions that have identity. But product **C** have highest rate of fraud more than any other category, its fraud rate is 11.8 %. We have no more information about the meaning of **C** and **W**hen trying to make other observation I compared the transaction amount **TransactionAmt** of **C** with the other categories, I find out that these products are with lower amounts than others.

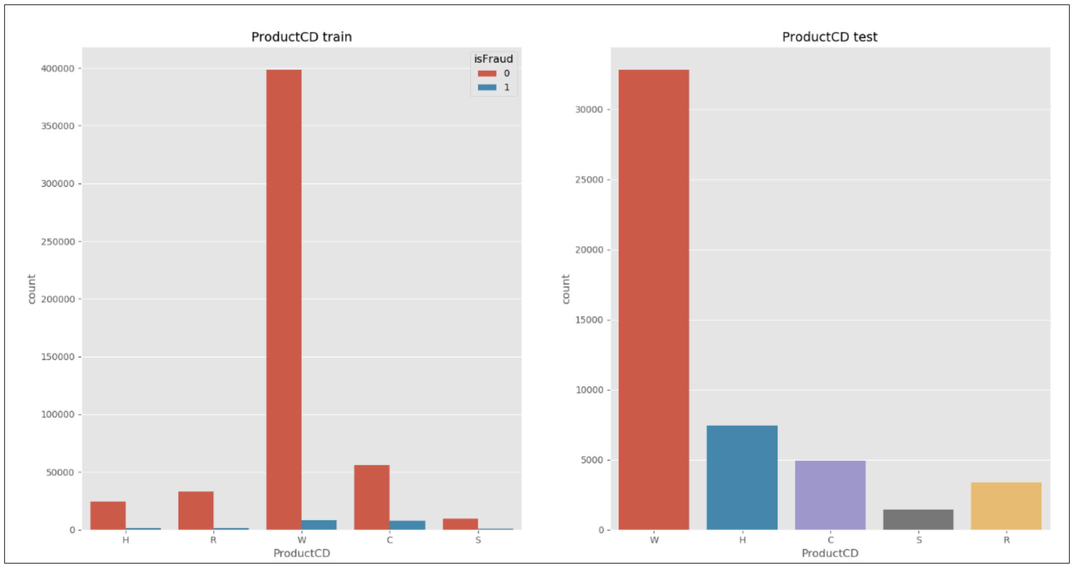


Figure 6 Count of observations by ProductCD

Next, for columns **card4** and **card6** in training dataset, these are 2 categorical features,We observe that there are four categories in **card4,** the most category has most frequency in both fraud and non-fraud is *Visa* and the second one is also the same *MasterCard,* So there are no significance difference between the distribution of categories among target.

Similarly, for the **card6** feature there are almost two categories in the dataset *debit* and *credit*, In the figure below, the most frequency is for these two features

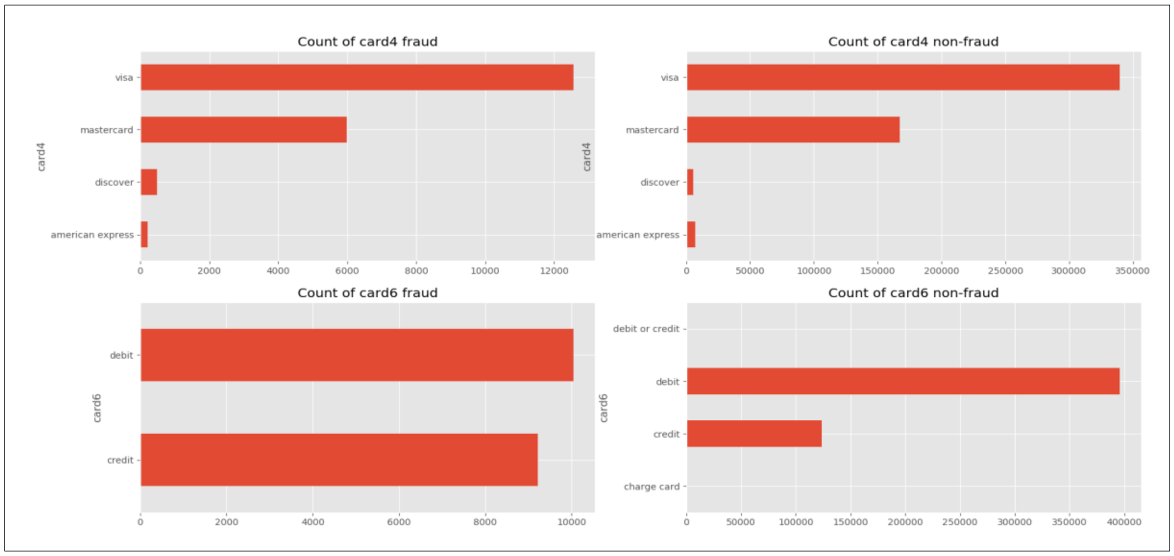


Figure 7 Distribution of Fraud and Non-fraud among card4 and card6

Visa accounts for 64% of all fraud transactions. However, when normalized by total number of each type, Visa have fraud rate of only 7%, lower than *Mastercard* and same as *Discovercard*. Only American Express have significantly lower fraud rate compare to others. Also, the number of card type are similar, and so does the fraud cases. In addition, not much difference in fraud rate between *credit card* and *debit card.*

Another important features, in my opinion, are **P_emaildomain** and **R_emaildomain,** I did some observation about the email domain in these two columns and it was noticed that a lot of domains came from the same distributors such as hotmail.com, hotmail.fr, yahoo.com, yahoo.fr, yahoo.de, etc. We can group these domains together under the parent distributors, that helps me for more clear visualization to examine the fraudulent rate in the dataset. *protonmail.com* has the highest fraud rate among all other domains, also, I observed that all the corresponding **R_emaildomain** even missing or *protonmail.com.*

The categorical feature **DeviceType ,** When we explore the data values in this column we have figured out that the column has a null values but less than 90% so according to my design this column will not be dropped and there are two categories for the device type : desktop and mobile and the distribution of them is in the following figure :

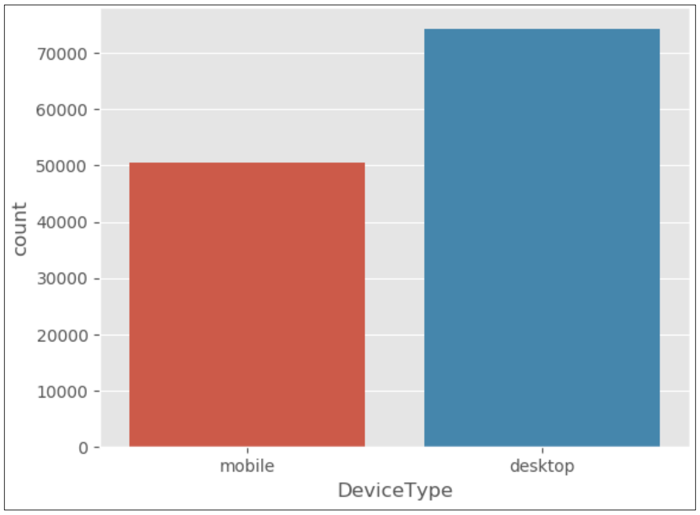


Figure 8 Device type distribution in the training dataset

When calculating fraud rate, I figured out that fraud rate is higher for mobile device compared to desktop.

c) Imputing missing Value:

At the beginning I dropped some features that have missing percentage above 90 % of the rows as I mentioned earlier, now I need to handle the remaining missing values so that we can built our model.
We will be using libraries in Python such as Numpy, Pandas and SciKit Learn to handle these values.

- Replacing with Mean/ Mode

This strategy can be applied on a feature which has numeric (mean) or categorical data (mode) like the **DeviceType** of transaction. We can calculate the mean, mode of the feature and replace it with the missing values. This is an approximation which can add variance to the data set. But the loss of the data can be negated by this method which yields better results compared to removal of rows and columns. Replacing with the above two approximations are a statistical approach of handling the missing values.
May be one of the disadvantages of this method is Imputing the approximations add variance and bias.
The *scikit-learn* library provides the *Imputer() preprocessing class* that can be used to replace missing values.
The function called *SimpleImputer* which is a flexibale function that allows you to specify the value to replace (in our case is *NAN*) and the the technique used to replace it (such as mean, median, or mode). The Imputer class operates directly on the NumPy array instead of the DataFrame.

d) Categorical Features Encoding:

Since most of the machine learning algorithms can not handle categorical features unless they are converted to numerical values. In general, this is mostly a constraint of the efficient implementation of machine learning algorithms rather than hard limitations on the algorithms themselves. Since there are a lot of categorical features in the dataset that have many values so using manual replacement is impossible task. The techniques I used is

- Label Encoding

In label encoding, we map each category to a number or a label. The labels chosen for the categories have no relationship. Numerical labels are always between 0 and n_categories – 1.
Converting the labels into numeric form to convert it into the machine-readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important pre-processing step for the structured dataset in supervised learning.
The function called *LabelEncoder* which is provided by *sklearn.preprocessing class* and it works in the categorical features in the pandas dataframe and then *fit function* to apply the fitted encoder into the dataframe column.

e) Split train and validation

Generally, when we are working on a model and want to train it, then is obvious that there is a dataset. However, after training, we must test the model on some test dataset. In such cases, an obvious solution is to split the dataset you have into two sets, one for training and the other for testing; and you do this before you start training your model.

What is the benefit to splitting a dataset into some ratio of training and validation subsets for a learning algorithm?

One benefit of splitting a dataset into some ratio of training and validation subsets is that it prevents 'overfitting', a problem where the model has become too finely tuned to the data it's been given so that it is unable to create accurate predictions on data it hasn't been trained on. It allows us to test our model on an independent dataset so we can estimate its performance.

There is a function called *train_test_split* which is provided by *sklearn.modelselection* class.

There are a few parameters that we need to understand before we use the class:

- ***test_size*** — This parameter decides the size of the data that must be split as the test dataset. This is given as a fraction. For example, if you pass 0.5 as the value, the dataset will be split 50% as the test dataset. If you're specifying this parameter, you can ignore the next parameter.
- ***train_size*** — You have to specify this parameter only if you're not specifying the test_size. This is the same as test_size, but instead you tell the class what percent of the dataset you want to split as the training set.
- ***random_state*** — Here you pass an integer, which will act as the seed for the random number generator during the split. Or, you can also pass an instance of the RandomState class, which will become the number generator. If you don't pass anything, the RandomState instance used by np.random will be used instead.
-

For my design choice, I have split the dataset in a 80–20 ratio, which is a common practice in data science, and I assigned the value of *random_state* with an integer to make sure that I have the same order each time. Since, *train_test_split* splits arrays or matrices into random train and test subsets. That means that every time you run it without specifying *random_state*, you will get a different result, this is expected behavior

f) Define a metric of the model

Different performance metrics are used to evaluate different machine learning algorithms for classification problems. We can use classification performance metrics such as Log-Loss, Accuracy, AUC (Area under Curve) etc. Another example of metric for evaluation of machine learning algorithms is precision, recall, which can be used for sorting algorithms primarily used by search engines.

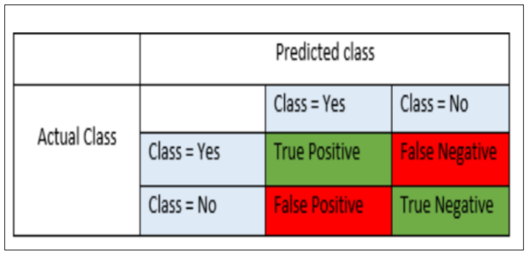
- **Confusion Matrix:**

The Confusion matrix is one of the most intuitive and easiest metrics used for finding the correctness and accuracy of the model. It is used for Classification problem where the output can be of two or more types of classes.

Let's give a label of to our target variable:

1*: When a transaction is fraud* ***0:*** *When a transaction is not fraud.*

Now that we have identified the problem, the confusion matrix, is a table with two dimensions ("Actual" and "Predicted") and sets of "classes" in both dimensions. Our actual classifications are columns and Predicted ones are rows.



		Predicted class	
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

Figure 9 Confusion matrix

<u>Terms associated with Confusion matrix:</u>

True Positives (TP): True positives are the cases when the actual class of the data point was 1(True) and the predicted is also 1(True)

Ex: The case where a transaction is actually fraud and the model classifying this transaction is fraud so it comes under True positive.

True Negatives (TN): True negatives are the cases when the actual class of the data point was 0(False) and the predicted is also 0 (False)

Ex: The transaction is NOT fraud and the model classifying it as NOT fraud ,it comes under True Negatives.

False Positives (FP): False positives are the cases when the actual class of the data point was 0(False) and the predicted is 1(True). False is because the model has predicted incorrectly and positive because the class predicted was a positive one. (1)
Ex: A trasnaction is NOT fraud and the model classifying it as fraud ,it comes under False Positives.
False Negatives (FN): False negatives are the cases when the actual class of the data point was 1(True) and the predicted is 0(False). False is because the model has predicted incorrectly and negative because the class predicted was a negative one. (0)
Ex: A transaction is fraud and the model classifying it as NOT fraud, it comes under False Negatives.

The ideal scenario that we all want is that the model should give 0 False Positives and 0 False Negatives. But that's not the case in real life as any model will NOT be 100% accurate most of the times.

- **Accuracy**

Accuracy in classification problems is the number of correct predictions made by the model over all kinds predictions made. Accuracy should NEVER be used as a measure when the target variable classes in the data are a majority of one class just like our case.

- **Precision/Recall**

These two performance metrics are often use in conjunction. Precision Formula TP / (TP + FP)
With precision, we are evaluating our data by its performance of 'positive' predictions.
Recall (also called ensitivity) formula: TP / (TP + FN)
With recall, we are evaluating our data by its performance of the ground truths for positive outcomes. Which means , we are judging how well predicted positive when the result was *Positive*.

- We cannot have the best of both worlds. In our fraud detection example, we had high precision and recalled scores. However, there's a trade-off. The denominator of precision is TP + FP. It does not consider FN. Thus, if we only made one positive prediction, and were correct, then our precision score would be 1. Even though, we might have missed a lot of actual positive observations.
- On the other hand, Recall's denominator is TP + FN. It does not look at all the confident predictions I made. We can predict that all transactions are fraud. Thus, we will have a Recall score of 1 since will are anticipating every example to be positive and our FN will be 0 (we are not prediction non-spam).

- **F1 Score:**

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an **uneven** class distribution. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

F1 Score = 2*(Recall * Precision) / (Recall + Precision)

In our case, as mentioned in project requirements, we must **calculate F1 score** for our predictive model,
So, there is a function called *f1_score* which is provided by the *sklearn.metrics* class.

All the metrics mentioned above have functions provided by this class.

g) Imblanced classes

Imbalanced classes are a common problem in machine learning classification where there are a disproportionate ratio of observations in each class. Most machine learning algorithms work best when the number of samples in each class are about equal. This is because most algorithms are designed to maximize accuracy and reduce error.

Dealing with imbalanced datasets entails strategies such as improving classification algorithms or balancing classes in the training data (data preprocessing) before providing the data as input to the machine learning algorithm. The later technique is preferred as it has wider application.

The main objective of balancing classes is to either increasing the frequency of the minority class or decreasing the frequency of the majority class. This is done in order to obtain approximately the same number of instances for both the classes.

Re-sampling techniques are divided in four categories:

1. *Random Under-sampling the majority class.*

Generally, under-sampling can be defined as removing some observations of the majority class. Under-sampling can be a good choice when you have a ton of data -think millions of rows. But a drawback is that we are removing information that may be valuable. This could lead to underfitting and poor generalization to the test set.

Advantages

It can help improve run time and storage problems by reducing the number of training data samples when the training data set is huge.

Disadvantages

It can discard potentially useful information which could be important for building rule classifiers.
The sample chosen by random under sampling may be a biased sample. And it will not be an accurate representative of the population. Thereby, resulting in inaccurate results with the actual test data set.

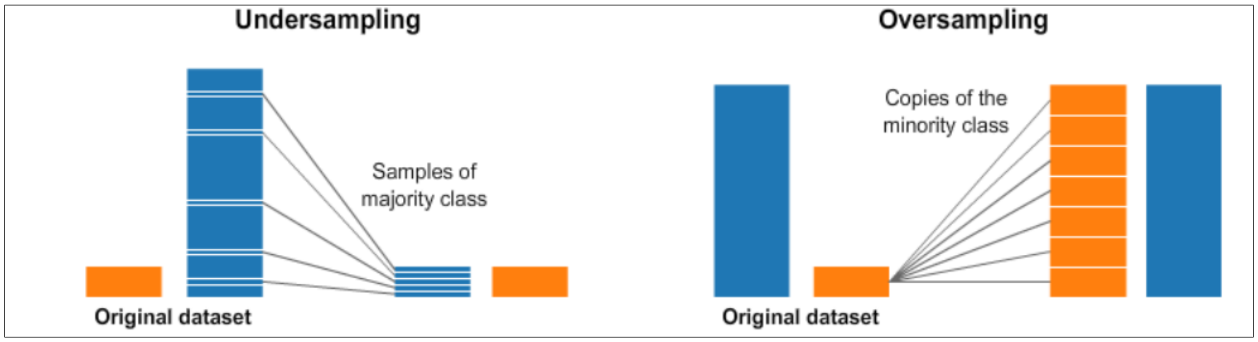


Figure 10 Under-sampling and Over-sampling

2. *Random Over-sampling the minority class.*

Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample. Oversampling can be a good choice when you don't have a ton of data to work with.

Advantages

Unlike under sampling this method leads to no information loss.
Outperforms under sampling

Disadvantages

It increases the likelihood of overfitting since it replicates the minority class events.

3. *Cluster-Based Over Sampling*

In this case, the K-means clustering algorithm is independently applied to minority and majority class instances. This is to identify clusters in the dataset. Subsequently, each cluster is oversampled such that all clusters of the same class have an equal number of instances and all classes have the same size.

Advantages

This clustering technique helps overcome the challenge between class imbalance. Where the number of examples representing positive class differs from the number of examples representing a negative class.
Also, overcome challenges within class imbalance, where a class is composed of different sub clusters. And each sub cluster does not contain the same number of examples.

Disadvantages

The main drawback of this algorithm, like most oversampling techniques is the possibility of over-fitting the training data

4. ***Informed Over Sampling: Synthetic Minority Over-sampling Technique (SMOTE)***

This technique is followed to avoid overfitting which occurs when exact replicas of minority instances are added to the main dataset. A subset of data is taken from the minority class as an example and then new synthetic similar instances are created. These synthetic instances are then added to the original dataset. The new dataset is used as a sample to train the classification models. What smote does is simple. First it finds the n-nearest neighbors in the minority class for each of the samples in the class . Then it draws a line between the the neighbors an generates random points on the lines.

In the figure ,the blue encircled dot is the current observation, the blue non-encircled dot is its nearest neighbor, and the green dot is the synthetic one.

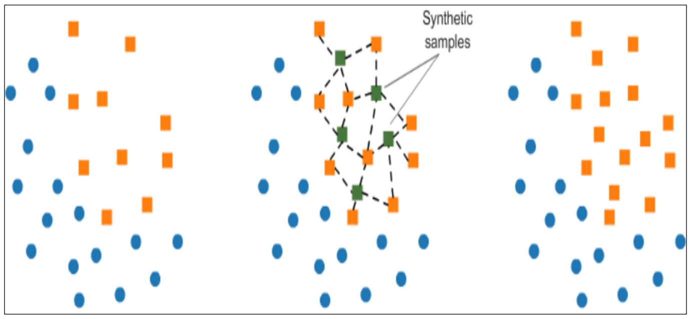
Advantages

Mitigates the problem of overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances

No loss of useful information

Disadvantages

While generating synthetic examples SMOTE does not take into consideration neighboring examples from other classes. This can result in increase in overlapping of classes and can introduce additional noise



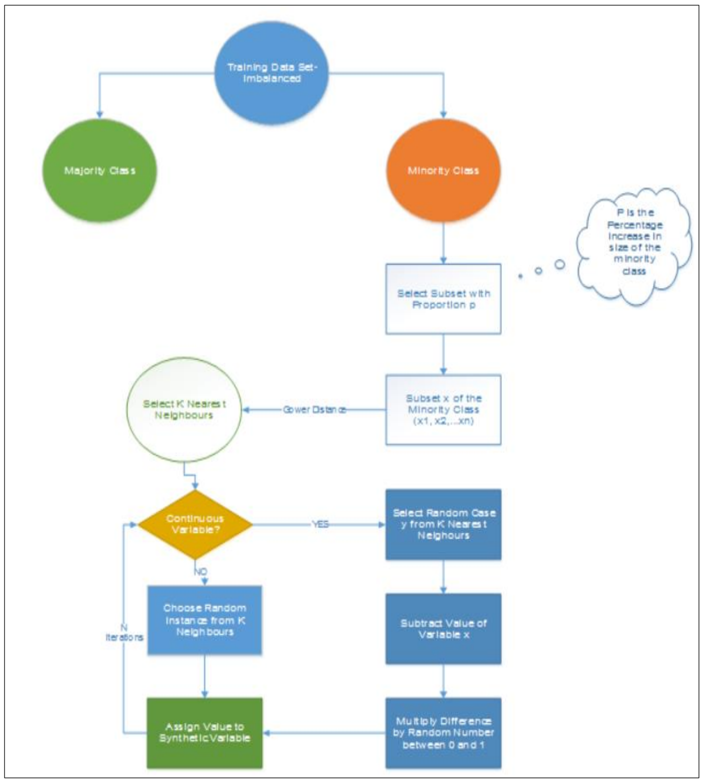

Training Data Set- Imbalanced
Majority Class
Minority Class
P is the Percentage Increase in size of the minority class
Select Subset with Proportion p
Subset x of the Minority Class (x1, x2,...xn)
Gower Distance
Select K Nearest Neighbours
Continuous Variable?
YES
NO
Select Random Case y from K Nearest Neighbours
Choose Random Instance from K Neighbours
N Iterations
Subtract Value of Variable x
Multiply Difference by Random Number between 0 and 1
Assign Value to Synthetic Variable

Figure 11 SMOTE technique illustration and how it works

For each observation that belongs to the under-represented class, the algorithm gets its K-nearest-neighbors and synthesizes a new instance of the minority label at a random location in the line between the current observation and its nearest neighbor.

For me in this project, is not to use Under sampling. The reason is that we are reducing the data set thus giving the model less data to feed on. I will give you an example we have a data set of 10000 data and there are only 100 data points for 1 while others are 0. Now after performing under sampling we are basically reducing the data set to 1100 samples where 1000 are 0 and 100 are 1. So, we are getting rid of almost 9000 samples and feeding it to the model. So, the model is more prone to error. In addition, Random over sampling has high possibility of over-fitting problem. Therefore, I choose to work with **SMOTE** to solve the imbalanced problem in our dataset.

In python, there is a library is called *imblearn.oversampling* ,the function is ***SMOTE.*** This function has *sampling_strategy='auto'* equivalent to 'not majority' and *k_neighbors=5* (the nearest five neighbors)

After applying ***SMOTE*** function in python, we apply the function *fit_resample* which resamples the train and test data again.

h) Feature Scaling

The provided dataset contains features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Euclidean distance between two data points in their computations, this is a problem. If left alone, these algorithms only take in the magnitude of features neglecting the units. The results would vary greatly between different units. The features with high magnitudes will weigh in a lot more in the distance calculations than features with low magnitudes. To suppress this effect, we need to bring all features to the same level of magnitudes. This can be achieved by scaling. So, the question now is when to scale the data. Some examples of algorithms where feature scaling matters are:
k-nearest neighbors with a Euclidean distance measure is sensitive to magnitudes and hence should be scaled for all features to weigh in equally.
Scaling is critical, while performing Principal Component Analysis (PCA). PCA tries to get the features with maximum variance and the variance is high for high magnitude features. This skews the PCA towards high magnitude features.
We can speed up gradient descent by scaling. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.
Tree based models are not distance based models and can handle varying ranges of features. Hence, Scaling is not required while modelling trees.
Algorithms like Linear Discriminant Analysis (LDA), Naive Bayes are by design equipped to handle this and gives weights to the features accordingly. Performing a feature scaling in these algorithms may not have much effect.

In python, I used during my experiments using **PCA** and **KNeighborhood** algorithms used the function *StandardScalar()* provided by *sklearn.preprocessing* then *fit_transform* the training and test data.

i) Feature Reduction

In machine learning, to catch useful indicators and obtain a more accurate result, we tend to add as many features as possible at first. However, after a certain point, the performance of the model will decrease with the increasing number of elements. This phenomenon is often referred to as "*The Curse of Dimensionality.*
The curse of dimensionality occurs because the sample density decreases exponentially with the increase of the dimensionality. When we keep adding features without increasing the number of training samples as well, the dimensionality of the feature space grows and becomes sparser and sparser. Due to this sparsity, it becomes much easier to find a "perfect" solution for the machine learning model which highly likely leads to overfitting.
Dimensionality reduction is the process of reducing the dimensionality of the feature space with consideration by obtaining a set of principal features. Dimensionality reduction can be further broken into feature selection and feature extraction.

Principal Component Analysis (PCA) is a linear dimensionality reduction technique that can be utilized for extracting information from a high-dimensional space by projecting it into a lower-dimensional sub-space. It tries to preserve the essential parts that have more variation of the data and remove the non-essential parts with fewer variation. PCA is an unsupervised dimensionality reduction technique, you can cluster the similar data points based on the feature correlation between them without any labels. PCA is based on when the data is projected into a lower dimension from a higher space, these dimensions are the Principal Components that captures most of the variance (information) of your data. principal components have both direction and magnitude. The direction represents across which principal axes the data is mostly spread out or has most variance and the magnitude signify the amount of variance that Principal Component captures of the data when projected onto that axis. The principal components are a straight line, and the first principal component holds the most variance in the data. Each subsequent principal component is orthogonal to the last and has

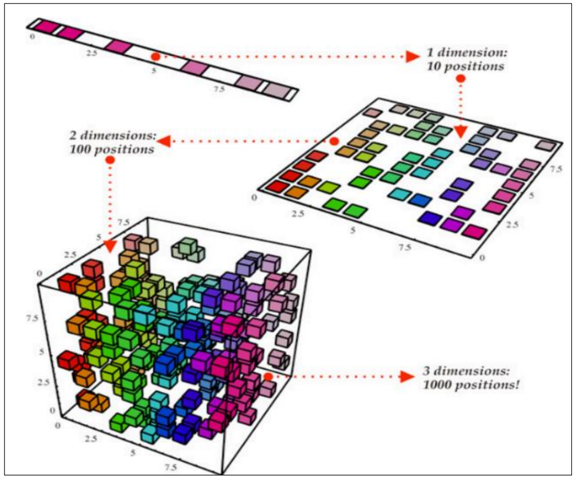

1 dimension: 10 positions
2 dimensions: 100 positions
3 dimensions: 1000 positions!

Figure 12 Principal Component Analysis (PCA) illustration

a lesser variance. In this way, given a set of x correlated variables over y samples you achieve a set of u uncorrelated principal components over the same y samples.

The reason you achieve uncorrelated principal components from the original features is that the correlated features contribute to the same principal component, thereby reducing the original data features into uncorrelated principal components; each representing a different set of correlated features with different amounts of variation.
Each principal component represents a percentage of total variation captured from the data.

In the project, I used PCA algorithm during preprocessing of the data. There is a function provided by *sklearn.decomposition* library with two alternatives to keep 120 or 100 components to fed this to the classifier. The function called *PCA*. I kept all its attributes by default values except the *n_compoenents* attribute

j) Select Features

It is mentioned earlier that I have eliminated the features that has the following:

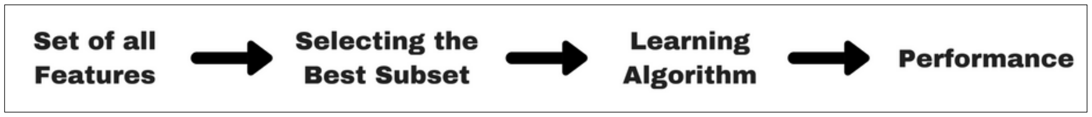
- Features with only 1 unique value
- Features with more than 90% missing values
- Features with the top value appears more than 90% of the time

Now, we are going to start the process where we automatically select those features in the dataset that contribute most to the prediction variable or output in which we are interested. Since having too many irrelevant features in the provided data can decrease the accuracy of the models. Three benefits of performing feature selection before modeling your data are:

- Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.
- Improves Accuracy: Less misleading data means modeling accuracy improves.
- Reduces Training Time: Less data means that algorithms train faster.
- It reduces the complexity of a model and makes it easier to interpret.

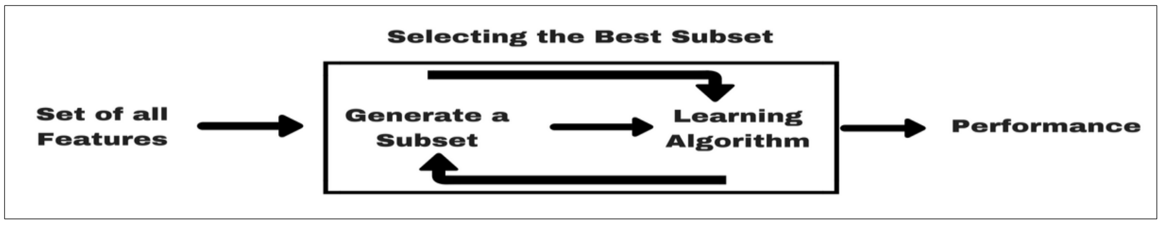
There are different types of general feature selection methods - Filter methods, Wrapper methods, and Embedded methods.

- **Filter methods:**



Filter method relies on the general uniqueness of the data to be evaluated and pick feature subset, not including any mining algorithm. Filter method uses the exact assessment criterion which includes distance, information, dependency, and consistency. The filter method uses the principal criteria of ranking technique and uses the rank ordering method for variable selection. The reason for using the ranking method is simplicity, produce excellent and relevant features. The ranking method will filter out irrelevant features before classification process starts. Features give rank based on statistical scores which tend to determine the features' correlation with the outcome variable. Correlation is a heavily contextual term, and it varies from work to work.
Some examples of some filter methods include the *Chi-squared test, information gain, and correlation coefficient scores.*

- **Wrapper methods:**



The wrapper method needs one machine learning algorithm and uses its performance as evaluation criteria. This method searches for a feature which is best suited for the machine learning algorithm and aims to improve the mining performance. To evaluate the features, the predictive accuracy used for classification tasks and goodness of cluster is evaluated using clustering.
Some typical examples of wrapper methods are forward feature selection, backward feature elimination, recursive feature elimination, etc.

- **Embedded methods:**

Embedded methods are iterative in a sense that takes care of each iteration of the model training process and carefully extract those features which contribute the most to the training for a iteration. Regularization methods are the most commonly used embedded methods which penalize a feature given a coefficient threshold

Difference between filter and wrapper methods
Well, it might get confusing at times to differentiate between filter methods and wrapper methods in terms of their functionalities.

- Filter methods do not incorporate a machine learning model in order to determine if a feature is good or bad whereas wrapper methods use a machine learning model and train it the feature to decide if it is essential or not.
- Filter methods are much faster compared to wrapper methods as they do not involve training the models. On the other hand, wrapper methods are computationally costly, and in the case of massive datasets, wrapper methods are not the most effective feature selection method to consider.
- Filter methods may fail to find the best subset of features in situations when there is not enough data to model the statistical correlation of the features, but wrapper methods can always provide the best subset of features because of their exhaustive nature.

In feature selection for this project, I used three techniques to include features during learning phase or to drop it
First, I used *Chi-Square test* which is one of the filter methods dealing with finding correlation between two categorical variables
The chi-square test is one of the most common ways to examine relationships between two or more categorical variables. It involves calculating a number, called the chi-square statistic - χ2 which follows a chi-square distribution. The chi-square test relies on the difference between observed and expected values.
Our hypotheses will be:

H0: There is no relationship between 2 categorical variables(features)
i.e. both are independent of each other
HA: There is a relationship between 2 categorical variables(features)
i.e. both are dependent of each other

I have implemented a class in python which find out categorical features in the pandas dataframe, then for each feature it calculates the chi-square statistic between the **isFraud** variable and other categorical variables.

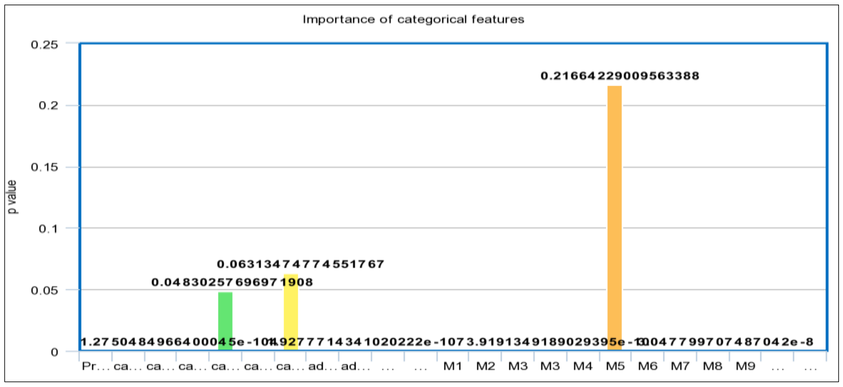


Figure 13 importance of categorical features according to chi-square

The test shows that features **card6** and **M5** are not important in classification.

Second, I used *Linear Discrimination algorithm* which also one of the filter methods dealing with finding correlation between continuous response features and categorical features. The idea of LDA is simple. In the training samples, LDA tries to project the sample onto a straight line so that the projection points of interclass samples are as close as possible and the projection points of the intraclass samples are as far apart as possible. When classifying a new sample, we projected it onto the same line. The classification of this sample is determined based on the position of the projected point. Unlike PCA, which tries to maintain data information as much as possible, LDA is to make the data points more distinguishable after dimension reduction. LDA takes labeling into account. The aim is the distance between data points of different categories after projection more significant and the distance between the data points of the same class more compact.
I used a function called *LinearDiscriminantAnalysis* which is provided in the class from sklearn. *discriminant_analysis* and set the *n_comepent =120, solver ='svd'*. After that we have to *fit_transaform* the classifier to the training data

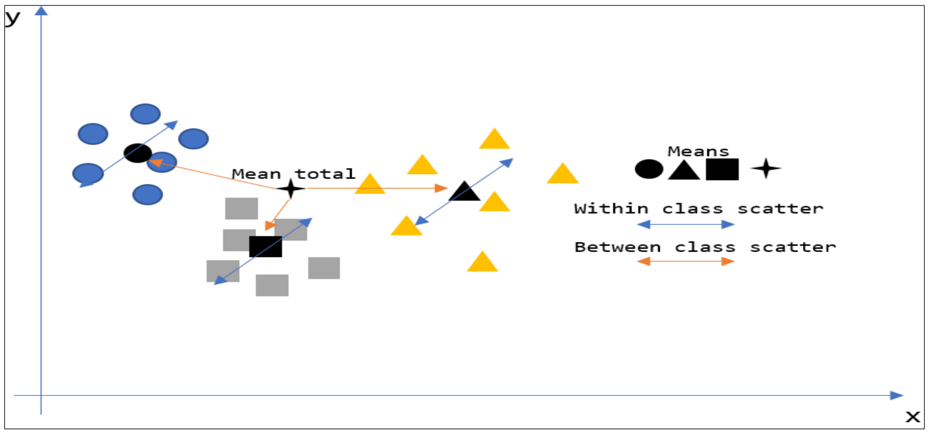


Figure 14 Linear Discrimination Analysis

One of the wrapper methods that I used is the **Recursive Feature Elimination (RFE)**. It works by recursively removing attributes and building a model on those attributes that remain. It uses the model performance metric to identify which attributes (and combination of attributes) contribute the most to predicting the target attribute. Moreover, feature selection methods can give us useful information on the relative importance or relevance of features for a given problem. We can use this information to create filtered versions of your dataset and increase the accuracy of the models.

1. Recursive feature elimination algorithm:

```
1.1  Tune/train the model on the training set using all predictors
1.2  Calculate model performance.
1.3  Calculate variable importance or rankings.
1.4  For Each subset size $S_i$ i=1…S do
1.5      Keep the $S_i$ most important variables
1.6      Preprocess the data [Optional]
1.7      Tune/train model on the traiing set using $S_i$ predictors
1.8      Caculate model eperformance.
1.9      Recalculate the ranking of each predictor
1.10 End
1.11 Calculate the performance profile over the
1.12 Determine the appropriate number of predictors $S_i$
1.13 Use the model coreesponding to the optimal $S_i$
```

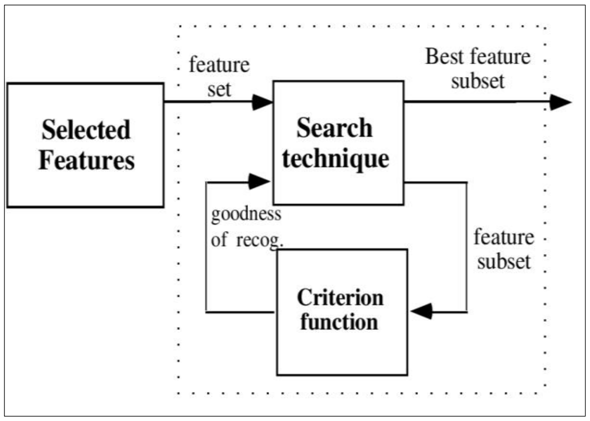


Figure 15 flowchart of RFE

3.5 Data mining Process

k) Selecting a classifier

Now I will explain the basic idea of the classifiers which I used in the classification process and the functions name of this classifiers in python. Later, I will explain each model and how I tune the hyper parameters of these different models. The classifiers in my different trials are:

1. Decision Tree:

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. It is a flowchart-like structure in which each internal node represents a “test” on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represent classification rules. Tree based learning algorithms are one of the best and mostly used supervised learning methods. Tree based methods empower predictive models with high accuracy, stability and ease of interpretation. Unlike linear models, they map non-linear relationships quite well. They are adaptable at solving any kind of problem at hand (classification or regression).

In python, It is provided a function called *DecisionTreeClassifier* that is provided by *sklearn* library using the *tree* class.

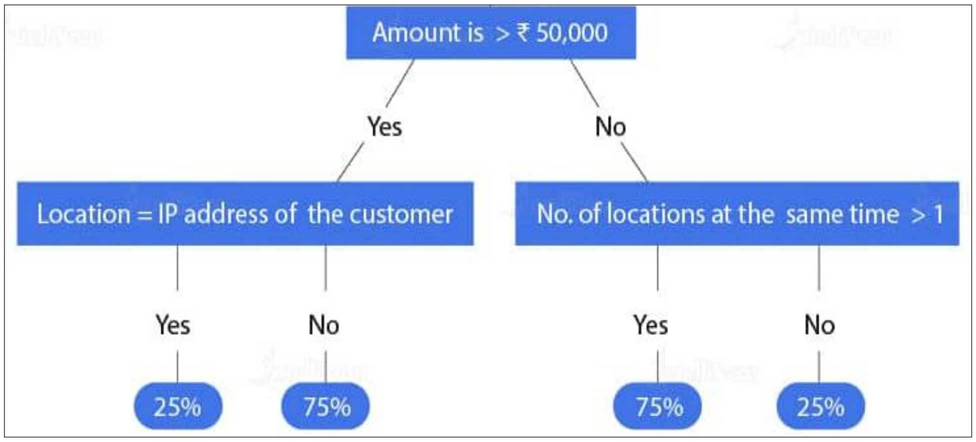

Amount is > ₹ 50,000
Yes
No
Location = IP address of the customer
No. of locations at the same time > 1
Yes
No
Yes
No
25%
75%
75%
25%

Figure 16 Simple illustration of Decision Tree

Ensemble Classifiers :

There are three main terms describing the ensemble (combination) of various models into one more effective model: Bagging to decrease the model’s variance; Boosting to decreasing the model’s bias, and;
Stacking to increasing the predictive force of the classifier.

2. Random Forest Classifier

It is a supervised classification algorithm. This algorithm creates the forest with several trees. Generally, the more trees in the forest the more robust the forest looks like. In the same way in the random forest classifier, the higher the number of trees in the forest gives the high accuracy results. we are creating a greater number of decision trees and how can we create a greater number of decision trees. As all the calculation of nodes selection will be same for the same dataset. To model a greater number of decision trees to create the forest you are not going to use the same apache of constructing the decision with information gain or gini index approach.
A random forest is an **ensemble** of decision tree classifiers. The trees are trained with some modifications which lead to a better overall classifier. Each tree in the forest is trained with a bootstrapped version of the original training data. Bootstrapping means to uniformly sample with replacement from the original data. At each node in the tree, only a random subset of features is used to split on.

Random forests (and decision trees in general) use a brute force approach when searching for the best split at a node. The “best” split is defined as the split which results in the maximum gain in node purity. It is common to use

Gini impurity to quantitatively define a node's purity. Remember that the goal in training a decision tree is to split up the data such that the leaf nodes have high purity – they should contain a single class or at least a majority of one class. To calculate the Gini impurity for a given node with C classes and where p_i is the fraction of data points in the node belonging to class ii, use the following formula:

$$I_{gini} = 1 - \sum_{j=1}^{C} p_j^2$$

The pseudocode for random forest algorithm can split into two stages.

3. Random forest creation pseudocode.

4. Pseudocode to perform prediction from the created random forest classifier.

First, let's begin with random forest creation pseudocode

1. Randomly select "k" features from total "m" features. Where k << m
2. Among the "k" features, calculate the node "d" using the best split point.
3. Split the node into daughter nodes using the best split.
4. Repeat 1 to 3 steps until "l" number of nodes has been reached.
5. Build forest by repeating steps 1 to 4 for "n" number times to create "n" number of trees.

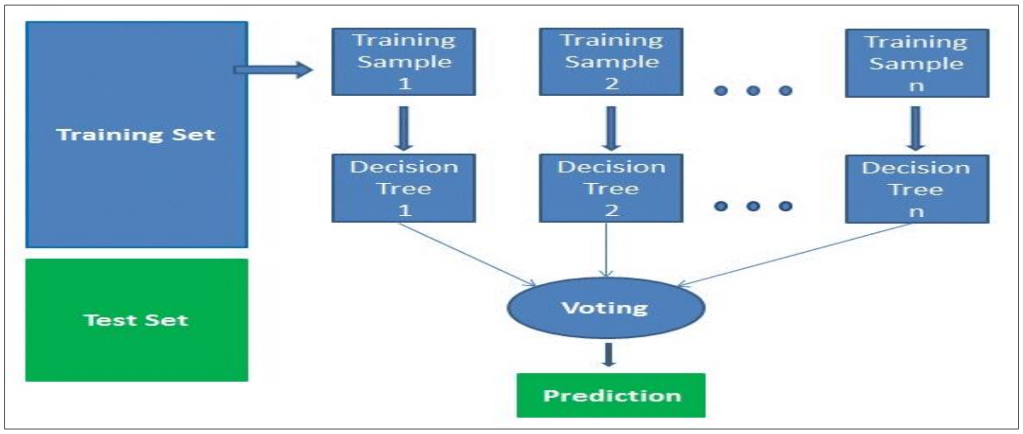


Figure 17 An illustration of how a random forest model is composed of multiple decision trees, each trained on a random subset of data.

The beginning of random forest algorithm starts with randomly selecting "k" features out of total "m" features. In the image, you can observe that we are randomly taking features and observations. In the next stage, we are using the randomly selected "k" features to find the root node by using the best split approach.

how a split is decided:

1. Calculate the impurity of the current node
2. Loop over possible splits
    2.1. Calculate impurity of the two resulting child nodes
    2.2. Calculate purity gain by subtracting the parent node's impurity and subtracting a weighted average of the child nodes impurities

3. Select the split which resulted in the highest purity gain.

The next stage, we will be calculating the daughter nodes using the same best split approach. Will the first 3 stages until we form the tree with a root node and having the target as the leaf node.

Finally, we repeat 1 to 4 stages to create "n" randomly created trees. This randomly created trees forms the random forest.

Random forest prediction pseudocode:

To perform prediction using the trained random forest algorithm uses the below pseudocode.

1. Takes the test features and use the rules of each randomly created decision tree to predict the outcome and stores the predicted outcome (target)
2. Calculate the votes for each predicted target.
3. Consider the high voted predicted target as the final prediction from the random forest algorithm.

To perform the prediction using the trained random forest algorithm we need to pass the test features through the rules of each randomly created trees. Suppose let's say we formed 100 random decision trees to from the random forest.
Each random forest will predict different target (outcome) for the same test feature. Then by considering each predicted target votes will be calculated. Suppose the 100 random decision trees are prediction some 3 unique targets x, y, z then the votes of x are nothing but out of 100 random decision trees how many trees prediction is x. Likewise, for other 2 targets (y, z). If x is getting high votes. Let's say out of 100 random decision tree 60 trees are predicting the target will be x. Then the final random forest returns the x as the predicted target.
This concept of voting is known as majority voting.
Advantages:
Random forests are considered as a highly accurate and robust method because of the number of decision trees participating in the process.
It does not suffer from the overfitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.
The algorithm can be used in both classification and regression problems.
Random forests can also handle missing values. There are two ways to handle these: using median values to replace continuous variables and computing the proximity-weighted average of missing values.
You can get the relative feature importance, which helps in selecting the most contributing features for the classifier.
Disadvantages:
Random forests are slow in generating predictions because it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest must make a prediction for the same given input and then perform voting on it. This whole process is time-consuming.
The model is difficult to interpret compared to a decision tree, where you can easily decide by following the path in the tree.

Random Forests vs Decision Trees

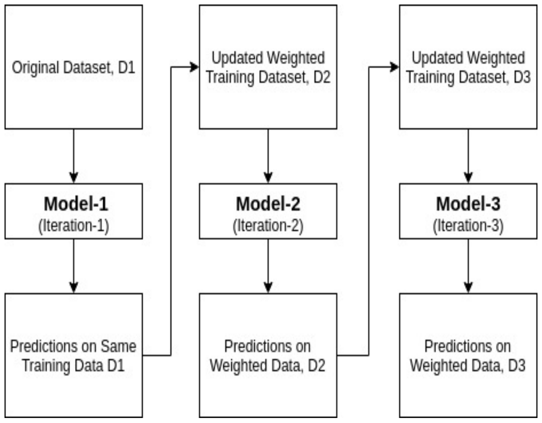
Random forests are a set of multiple decision trees.
Deep decision trees may suffer from overfitting, but random forests prevent overfitting by creating trees on random subsets.
Decision trees are computationally faster.
Random forests are difficult to interpret, while a decision tree is easily interpretable and can be converted to rules

The function *RandomForsetClassifier* is provided in *sklearn* library through *ensemble* class, this function has different attributes *criterion* two supported criterion of separation *gini* and *gain , (default="gini"),n_estimators* : The number of trees in the forest, I will specify the number of estimator later for each model I mention . *max_features*: the number of features to consider when looking for the best split. I left it auto.

- **AdaBoost Classifier**

Ada-boost or Adaptive Boosting is one of ensemble boosting. It combines multiple classifiers to increase the accuracy of classifiers. AdaBoost is an iterative ensemble method. AdaBoost classifier builds a strong classifier by combining multiple poorly performing classifiers so that you will get high accuracy strong classifier. The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures the accurate predictions of unusual observations. Any machine learning algorithm can be used as base classifier if it accepts weights on the training set. Adaboost should meet two conditions:

i. The classifier should be trained interactively on various weighed training examples.
ii. In each iteration, it tries to provide an excellent fit for these examples by minimizing training error.


Original Dataset, D1
Updated Weighted Training Dataset, D2
Updated Weighted Training Dataset, D3
Model-1 (Iteration-1)
Model-2 (Iteration-2)
Model-3 (Iteration-3)
Predictions on Same Training Data D1
Predictions on Weighted Data, D2
Predictions on Weighted Data, D3
Figure 18 Adaboost Classifier illustration

It works in the following steps:

1. Initially, Adaboost selects a training subset randomly.
2. It iteratively trains the AdaBoost machine learning model by selecting the training set based on the accurate prediction of the last training.
3. It assigns the higher weight to wrong classified observations so that in the next iteration these observations will get the high probability for classification.
4. Also, It assigns the weight to the trained classifier in each iteration according to the accuracy of the classifier. The more accurate classifier will get high weight.
5. This process iterates until the complete training data fits without any error or until reached to the specified maximum number of estimators.
6. To classify, perform a "vote" across all the learning algorithms you built

To build Adaboost model in python, there is a function provided called *AdaBoostClassifier* in *sklearn.ensamble* class
The most important parameters are *base_estimator, n_estimators, and learning_rate."*
base_estimator: It is a weak learner used to train the model. It uses *DecisionTreeClassifier* as default weak learner for training purpose. we can also specify different machine learning algorithms.
n_estimators: Number of weak learners to train iteratively.
learning_rate: It contributes to the weights of weak learners. It uses 1 as a default value.
Also, this ensemble classifier can be used with different base estimators, I used it with SVC algorithm but I added that SVC classifier works with probability to be able to work inside AdaBoost classifier.

Pros

AdaBoost is easy to implement. It iteratively corrects the mistakes of the weak classifier and improves accuracy by combining weak learners. You can use many base classifiers with AdaBoost. AdaBoost is not prone to overfitting. This can be found out via experiment results, but there is no concrete reason available.

Cons

AdaBoost is sensitive to noise data. It is highly affected by outliers because it tries to fit each point perfectly.

- **Bagging Classifier**

It is shorthand for the combination of bootstrapping and aggregating. Bootstrapping is a method to help decrease the variance of the classifier and reduce overfitting, by resampling data from the training set with the same cardinality as the original set. The model created should be less overfitted than a single individual model.
A high variance for a model is not good, suggesting its performance is sensitive to the training data provided. So, even if more the training data is provided, the model may still perform poorly. And, may not even reduce the variance of our model.
Bagging is an effective method when you have limited data, and by using samples you're able to get an estimate by aggregating the scores over many samples. The simplest approach with bagging is to use a couple of small subsamples and bag them, if the ensemble accuracy is much higher than the base models, it's working; if not, use larger subsamples. In bagging there is a tradeoff between base model accuracy and the gain you get through bagging. The aggregation from bagging may improve the ensemble greatly when you have an unstable model, yet when your base models are more stable.

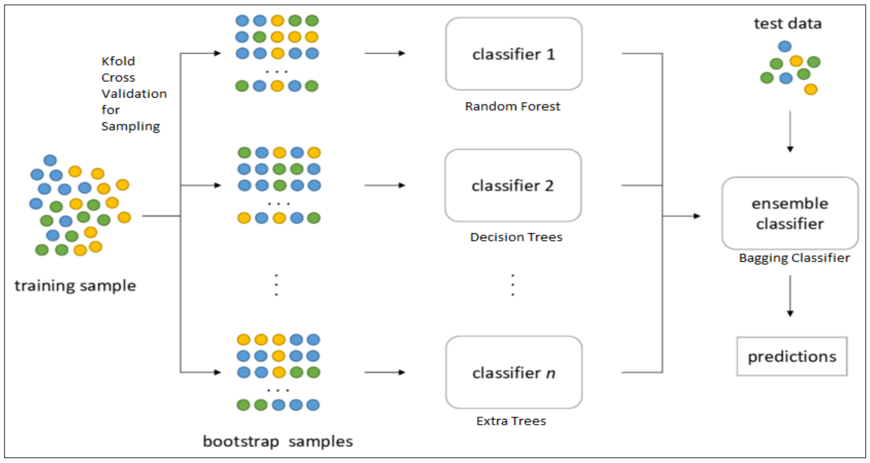


Figure 19 Bagging Classifier Process flow

To build Adaboost model in python, there is a function provided called *BaggingClassifier* in *sklearn.ensamble* class. The most important parameters are base_estimator, n_estimators, and *bootstrap=True*
base_estimator: It is a weak learner used to train the model. It uses DecisionTreeClassifier as default weak learner for training purpose. we can also specify different machine learning algorithms.
n_estimators: Number of weak learners to train iteratively.
I used it with a base estimator KNeighborsClassifier which is also provided in python through *sklearn* library in *neighbors* class, I set the number of neighbors randomly to best select the best one. Also, in this model I have scaled the data first because this classifier is sensitive to the value of features.

I) Hyper-parameter Tuning and parameter optimization

I have explained the basic idea of most classifiers that I used in my experiments to find out the best model for me.

I want to explain briefly how I build an algorithm to select the best model and tune the hyper parameters. This classifier alone is called a baseline model now I have to consider a specific way to select the best with high performance and tune the hyper parameters of that model.

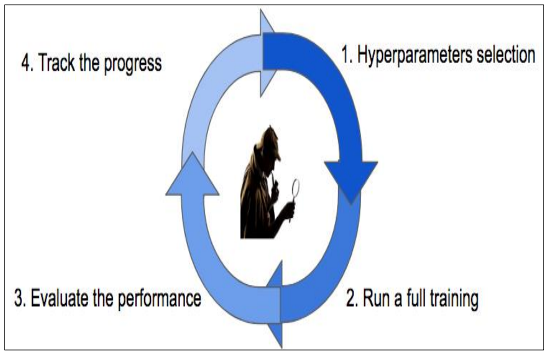


Figure 20 The hyperparameters search cycle

Hyper-parameters are all the training variables set manually with a pre-determined value before starting the training.

Hyper-parameter Tuning is nothing but searching for the right set of hyperparameter to achieve high precision and accuracy. Optimizing hyperparameters constitute one of the trickiest parts in building the machine learning models. The primary aim of hyperparameter tuning is to find the sweet spot for the model's parameters so that a better performance is obtained.

There are several parameter tuning techniques, but I will explain briefly two of them

<u>Grid Search</u>

In Grid Search, we try every combination of a preset list of values of the hyper-parameters and evaluate the model for each combination. The pattern followed here is like the grid, where all the values are placed in the form of a matrix. Each set of parameters is taken into consideration and the accuracy is noted. Once all the combinations are evaluated, the model with the set of parameters which give the top accuracy is the best.
One of the major drawbacks of grid search is that when it comes to dimensionality, it suffers when the number of hyperparameters grows exponentially. With as few as four parameters this problem can become impractical, because the number of evaluations required for this strategy increases exponentially with each additional parameter, due to the curse of dimensionality.

<u>Random Search</u>

Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. It tries random combinations of a range of values. To optimize with random search, the function is evaluated at some number of random configurations in the parameter space.
The chances of finding the optimal parameter are comparatively higher in random search because of the random search pattern where the model might end up being trained on the optimized parameters without any aliasing. Random search works best for higher dimensional data since the time taken to find the right set is less with a smaller number of iterations. Random search is the best parameter search technique when there are a larger number of dimensions

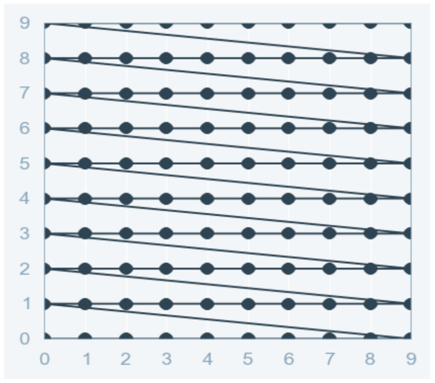


Figure 21 Visual Representation of grid search

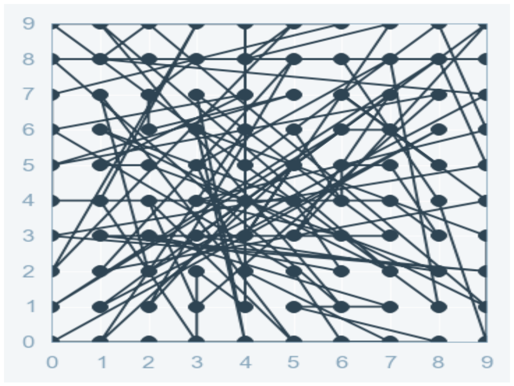


Figure 22 Visual Representation of Random Search

In this project, due to high dimensionality I used the second algorithm, random search gives best values for hyper parameters. But I did not use the function provided by python. I have create a iteration code that has some random values for the hyper parameters that the model used then I evaluate the f-score at the end of iteration Then, compare the models in each iteration and check the f-score each iteration if it is better then I store this model in a pickle file with all its features and the values of different hyper parameters of the classifier used.

j) Learning Process

In this part, I will discuss the different experiments I made on the provided dataset to achieve the best possible *f-score* among all trails and experiments using different models and feature selection techniques.
I built my techniques based on three different feature selection algorithms or, I can say different feature selection and dimensionality reduction algorithms. Moreover, I would like to mention that I have worked during my trials on subsets of the dataset not the 500K directly, but when I decided which model, I would choose I run this model on the whole dataset. First in all techniques, I perform data processing mentioned earlier in the report (such as replacing nans and label encode the categorical features) in addition to splitting the provided subset into training and testing as mentioned earlier in this report.

1. ***Model (1): Using Recursive feature elimination with cross validation using based on Random forest classier and followed by Random forest classifier in classification.***

1.1 Define a feature Recursive Elimination with Cross Validation classifier which I selected to be a random forest classifier. The attributes of the *REFCV* are randomly selected through my random search algorithm , this function has attribute for the base estimator which is a random first estimator in my technique, moreover I set the *step* attribute by 0.2 so that the REFCV which corresponds to the percentage of features to remove in such iteration. I left *cv as its* default value *cv* attribute determines the cross-validation splitting strategy. Since, the performance metric required is *f1 Score.*, so I set the *scoring* attribute is ‘f1’
1.2 Fit the RFE model and automatically tune the number of selected features.
1.3 Define a Random Forest Classifier to be used in learning process.

1.4 Set the number of estimators and maximum depth of these two classifiers are randomly selected in each iteration, In other words the number of estimators and maximum depth are the hyper parameters in that model ,so I am working on tuning them along each iteration till I got the best f score for the model.
1.5 We don’t have to using up sampling technique here ,since, this type of classifier has attribute called ***class_weight*** which I set with *balanced* value (The “*balanced*” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data)
1.6 We use the second classifier to predict the values in the test subset.
1.7 Calculate f score of the current model, save the model with f score higher than previous one (initially f sscore set to 0)
1.8 Go back to step 1.1 until you reached the best f score.

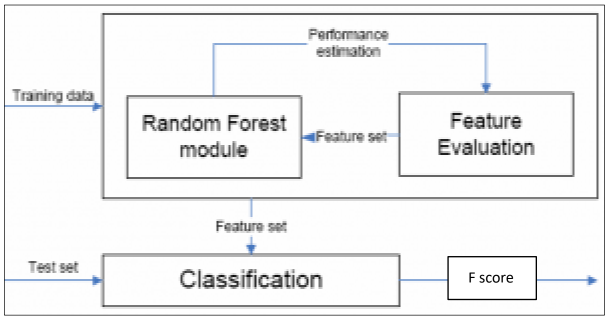


Figure 23 Model 1 Flow Chart

I have performed this technique over 3000 ,9000, 16000, 25000,50000.

Sample Size	Best f score
3000	0.250
9000	0.292
16000	0.304
20000	0.53
50000	0.593

Table 1: F scores of model 1 with different sample size

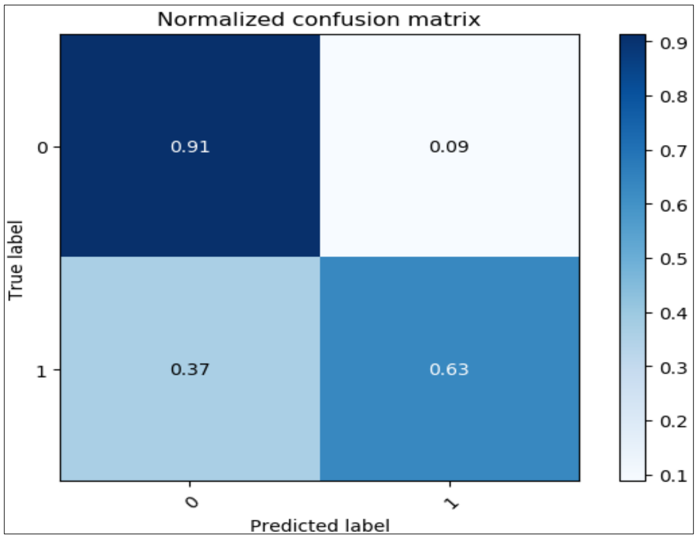


*Table 2 Normalized Confusion matrix for model 1 (**3000** samples)*

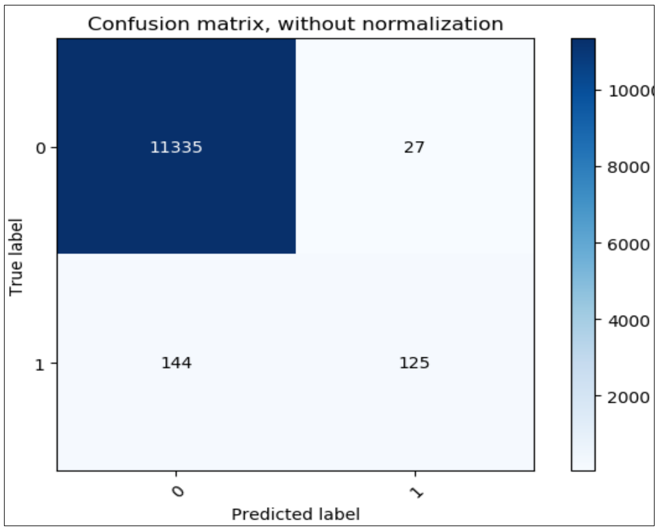


*Table3 Confusion matrix for model 1 (**50,000** samples)*

Feature Importance of the features selected by RFECV:

This is a list of selected features by RFECV with their rank:
['TransactionDT', 1], ['TransactionAmt', 1], ['ProductCD', 1], ['card1', 1], ['card2', 1], ['card3', 1], ['card5', 2], ['card6', 1], ['addr1', 1], ['dist1', 1], ['P_emaildomain', 2], ['R_emaildomain', 1], ['C1', 1], ['C2', 1], ['C4', 1], ['C5', 1], ['C6', 1], ['C8', 1], ['C9', 1], ['C10', 2], ['C11', 1], ['C12', 1], ['C13', 1], ['C14', 1], ['D1', 1], ['D2', 1], ['D3', 1], ['D4', 1], ['D5', 1], ['D8', 1], ['D9', 1], ['D10', 1], ['D11', 1], ['D12', 1], ['D15', 1], ['M5', 1], ['V30', 1], ['V49', 3], ['V69', 1], ['V70', 5], ['V87', 4], ['V90', 2], ['V91', 2], ['V94', 1], ['V97', 2], ['V127', 5], ['V130', 3], ['V149', 4], ['V158', 5], ['V198', 3], ['V201', 3], ['V243', 3], ['V244', 2], ['V246', 4], ['V257', 3], ['V264', 4], ['V268', 3], ['V280', 3], ['V283', 3], ['V285', 2], ['V294', 2], ['V307', 4], ['V308', 3], ['V310', 4], ['V312', 5], ['V314', 5], ['V317', 3], ['id_01', 2], ['id_02', 5], ['id_05', 5], ['id_06', 4], ['id_19', 3], ['id_20', 4], ['id_31', 3], ['id_33', 5], ['DeviceInfo', 5]

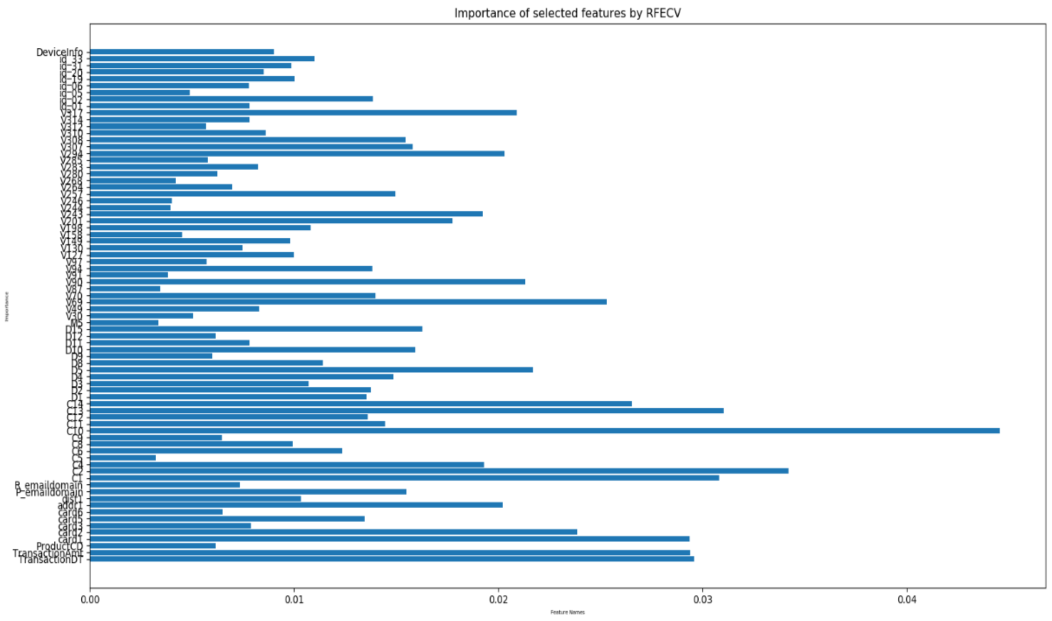


Figure 24 Importance of features according to model 1

There is a tool in python to help user to visualize a basic estimator of Random Forest classifier

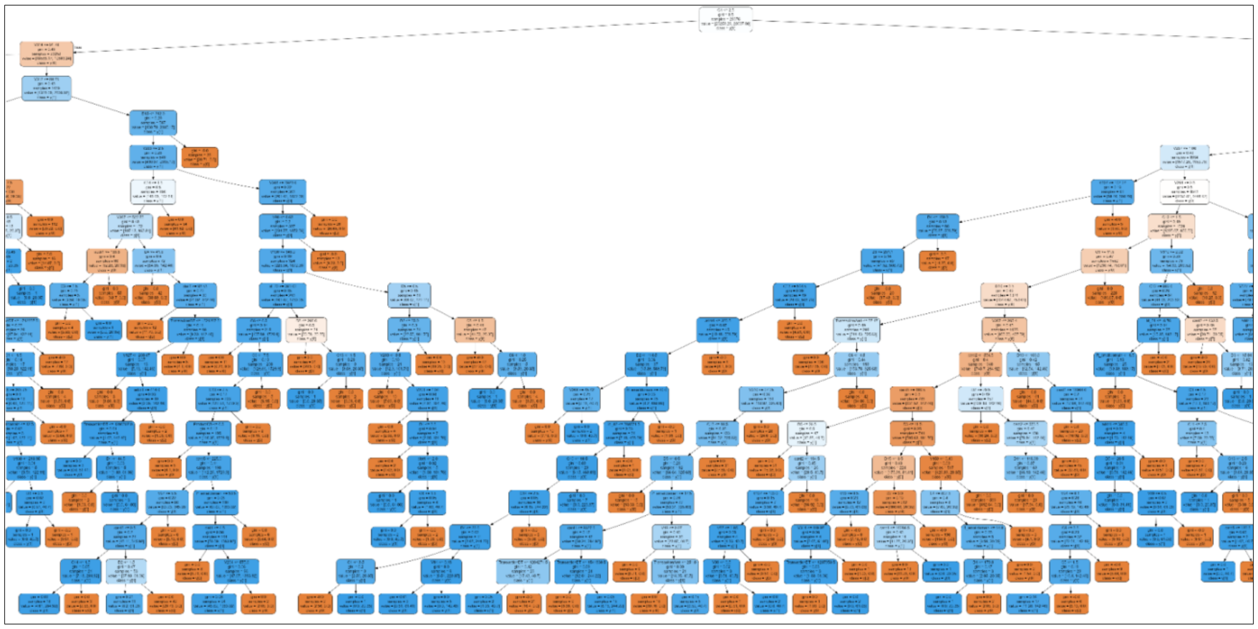


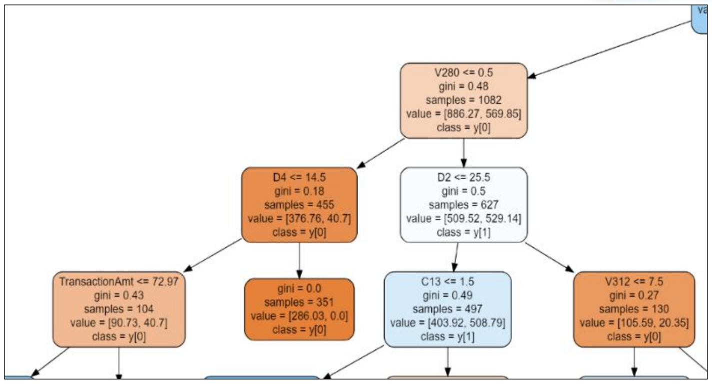
Figure 25 Part of Visualization of basic estomar(descion trww) ofRandomforest Classifier

Method to visualize descion tree of a random forest classifier:

Export Tree as .dot File: This makes use of the *export_graphviz* function in Scikit-Learn through *tree* class. There are many parameters here that control the look and information displayed.

Convert .dot to .png using a system command

A deeper look inside one node in this tree,
We can see that the decision tree (which is the base) of my random forest classifier, we can see that as I mentioned above it splits according to *gini* index of the feature and we can see how it separates according to a logical question like ,
Is **TransactionAmt** <=72.97 or the value of **D4** <=14.5, then it classifies the sample to class y[0] or class of y[1].



One last detail about the function REFCV that I have assigned the attribute ***n_jobs*** to be -1 so that I make the execution faster, since this option make the execution on all the processors.

As , we mentioned earlier, the data set is unbalanced ,so we don't take the accuracy score is a metric in our design because the accuracy will be always near 100 % , which is clear in the next chart which describes the different accuracy scores I got for different sample sizes , Clearly, the accuracy always near 100 % .

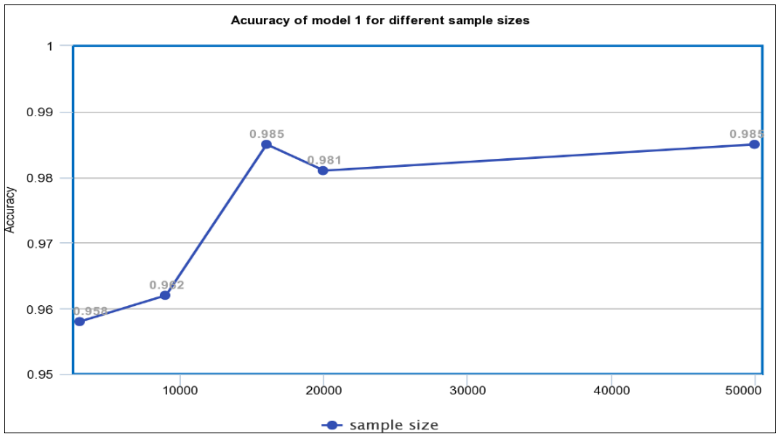
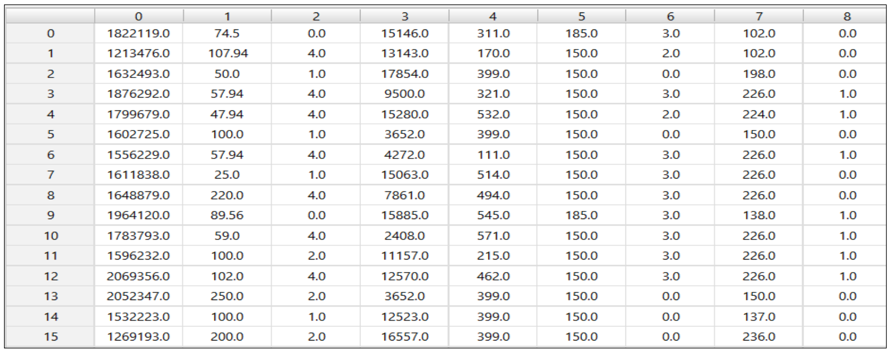


Figure 26 Accuracy of model 1

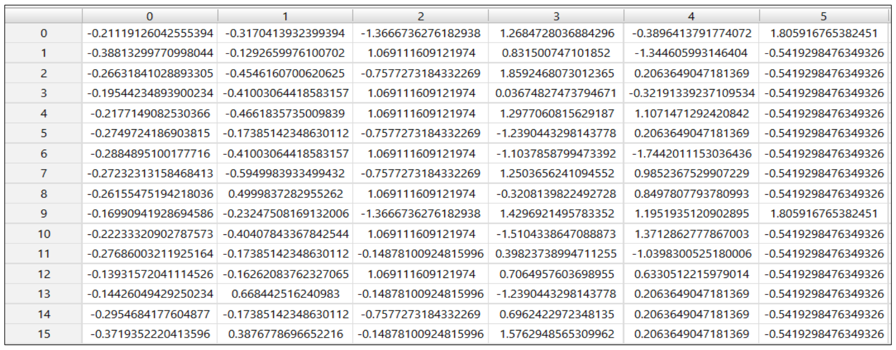
2. Model (2) Using Principle Component Analysis for dimensionality reduction and feature selection, followed by an AdaBoost Classifier

In this model, I have applied the same pre-processing steps on the features mentioned earlier, after, that I have split the data to training and test subset with the same ratio as model 1. There are two differences here in the pre-processing, the first one is that I used the **SMOTE** sampling algorithm to balance the data after splitting, moreover, I have to standardize the features first. Since PCA is a variance maximizing exercise. It projects your original data onto directions which maximize the variance. The first plot below shows the amount of total variance explained in the different principal components where we have not normalized the data. As you can see, it seems like component one explains most of the variance in the data. In the following figures, I am showing samples of the data set after imputing in the places of missing values and splitting to train and test. This data will be standardized before applying PCA algorithm on it (Figure 26). Finally, I have applied the PCA algorithm on this feature (before this step I can track the value of each feature provided in the dataset. But after applying the PCA algorithm we have a new component that their number is defined by me that will be used in learning process but what is the exact interpretation of this new features it is very hard to know that.



	0	1	2	3	4	5	6	7	8
0	1822119.0	74.5	0.0	15146.0	311.0	185.0	3.0	102.0	0.0
1	1213476.0	107.94	4.0	13143.0	170.0	150.0	2.0	102.0	0.0
2	1632493.0	50.0	1.0	17854.0	399.0	150.0	0.0	198.0	0.0
3	1876292.0	57.94	4.0	9500.0	321.0	150.0	3.0	226.0	1.0
4	1799679.0	47.94	4.0	15280.0	532.0	150.0	2.0	224.0	1.0
5	1602725.0	100.0	1.0	3652.0	399.0	150.0	0.0	150.0	0.0
6	1556229.0	57.94	4.0	4272.0	111.0	150.0	3.0	226.0	1.0
7	1611838.0	25.0	1.0	15063.0	514.0	150.0	3.0	226.0	0.0
8	1648879.0	220.0	4.0	7861.0	494.0	150.0	3.0	226.0	0.0
9	1964120.0	89.56	0.0	15885.0	545.0	185.0	3.0	138.0	1.0
10	1783793.0	59.0	4.0	2408.0	571.0	150.0	3.0	226.0	1.0
11	1596232.0	100.0	2.0	11157.0	215.0	150.0	3.0	226.0	1.0
12	2069356.0	102.0	4.0	12570.0	462.0	150.0	3.0	226.0	1.0
13	2052347.0	250.0	2.0	3652.0	399.0	150.0	0.0	150.0	0.0
14	1532223.0	100.0	1.0	12523.0	399.0	150.0	0.0	137.0	0.0
15	1269193.0	200.0	2.0	16557.0	399.0	150.0	0.0	236.0	0.0

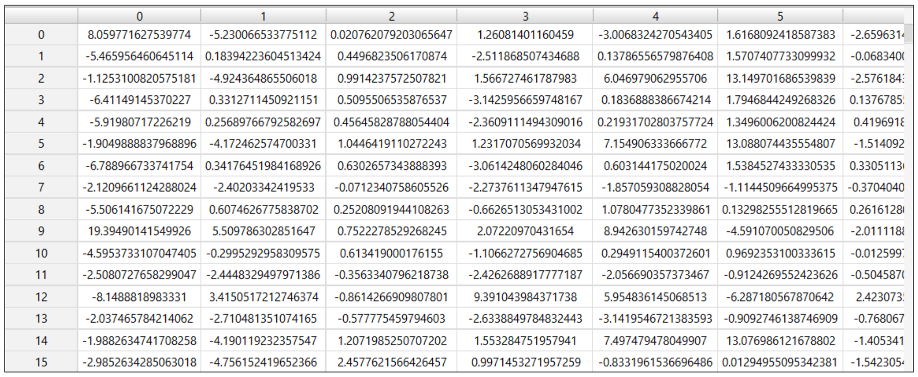
Figure 27 Features after imputing missing values



	0	1	2	3	4	5
0	-0.21119126042555394	-0.3170413932399394	-1.3666736276182938	1.2684728036884296	-0.3896413791774072	1.805916765382451
1	-0.38813299770998044	-0.1292659976100702	1.069111609121974	0.831500747101852	-1.344605993146404	-0.5419298476349326
2	-0.26631841028893305	-0.4546160700620625	-0.7577273184332269	1.8592468073012365	0.2063649047181369	-0.5419298476349326
3	-0.19544234893900234	-0.41003064418583157	1.069111609121974	0.03674827473794671	-0.32191339237109534	-0.5419298476349326
4	-0.2177149082530366	-0.4661835735009839	1.069111609121974	1.2977060815629187	1.1071471292420842	-0.5419298476349326
5	-0.2749724186903815	-0.17385142348630112	-0.7577273184332269	-1.2390443298143778	0.2063649047181369	-0.5419298476349326
6	-0.2884895100177716	-0.41003064418583157	1.069111609121974	-1.1037858799473392	-1.7442011153036436	-0.5419298476349326
7	-0.27232313158468413	-0.5949983933499432	-0.7577273184332269	1.2503656241094552	0.9852367529907229	-0.5419298476349326
8	-0.26155475194218036	0.4999837282955262	1.069111609121974	-0.3208139822492728	0.8497807793780993	-0.5419298476349326
9	-0.16990941928694586	-0.23247508169132006	-1.3666736276182938	1.4296921495783352	1.1951935120902895	1.805916765382451
10	-0.22233320902787573	-0.40407843367842544	1.069111609121974	-1.5104338647088873	1.3712862777867003	-0.5419298476349326
11	-0.27686003211925164	-0.17385142348630112	-0.14878100924815996	0.39823738994711255	-1.0398300525180006	-0.5419298476349326
12	-0.13931572041114526	-0.16262083762327065	1.069111609121974	0.7064957603698955	0.6330512215979014	-0.5419298476349326
13	-0.14426049429250234	0.668442516240983	-0.14878100924815996	-1.2390443298143778	0.2063649047181369	-0.5419298476349326
14	-0.2954684177604877	-0.17385142348630112	-0.7577273184332269	0.6962422972348135	0.2063649047181369	-0.5419298476349326
15	-0.3719352220413596	0.3876778696652216	-0.14878100924815996	1.5762948565309962	0.2063649047181369	-0.5419298476349326

Figure 28 Scaling the features using StandardScaler function

And the output after applying PCA analysis on the training data is the like below, we can see that the features' values have been changed and now the total number of features are 120 features.



	0	1	2	3	4	5	
0	8.059771627539774	-5.230066533775112	0.020762079203065647	1.26081401160459	-3.0068324270543405	1.6168092418587383	-2.659631
1	-5.465956460645114	0.18394223604513424	0.4496823506170874	-2.511868507434688	0.13786556579876408	1.5707407733099932	-0.068340
2	-1.1253100820575181	-4.924364865506018	0.9914237572507821	1.566727461787983	6.046979062955706	13.149701686539839	-2.576184
3	-6.41149145370227	0.3312711450921151	0.5095506535876537	-3.1425956659748167	0.1836888386674214	1.7946844249268326	0.137678
4	-5.91980717226219	0.25689766792582697	0.45645828788054404	-2.3609111494309016	0.21931702803757724	1.3496006200824424	0.419691
5	-1.9049888837968896	-4.172462574700331	1.0446419110272243	1.2317070569932034	7.154906333666772	13.088074435554807	-1.514092
6	-6.788966733741754	0.34176451984168926	0.6302657343888393	-3.0614248060284046	0.603144175020024	1.5384527433330535	0.330511
7	-2.1209661124288024	-2.40203342419533	-0.0712340758605526	-2.2737611347947615	-1.857059308828054	-1.1144509664995375	-0.370404
8	-5.506141675072229	0.6074626775838702	0.25208091944108263	-0.6626513053431002	1.0780477352339861	0.13298255512819665	0.261612
9	19.39490141549926	5.509786302851647	0.7522278529268245	2.07220970431654	8.942630159742748	-4.591070050829506	-2.011118
10	-4.5953733107047405	-0.2995292958309575	0.613419000176155	-1.1066272756904685	0.2949115400372601	0.9692353100333615	-0.012599
11	-2.5080727658299047	-2.4448329497971386	-0.3563340796218738	-2.4262688917777187	-2.056690357373467	-0.9124269552423626	-0.504587
12	-8.1488818983331	3.4150517212746374	-0.8614266909807801	9.391043984371738	5.954836145068513	-6.287180567870642	2.423073
13	-2.037465784214062	-2.710481351074165	-0.577775459794603	-2.6338849784832443	-3.1419546721383593	-0.9092746138746909	-0.768067
14	-1.9882634741708258	-4.190119232357547	1.2071985250707202	1.553284751957941	7.497479478049907	13.076986121678802	-1.405341
15	-2.9852634285063018	-4.756152419652366	2.4577621566426457	0.9971453271957259	-0.8331961536696486	0.01294955095342381	-1.542305

Figure 29 Training data after applying PCA

2.1. Define the Adaboost classifier using the provided function in python that I mentioned earlier the most important attribute for Adaboost classifier is the number of estimators that I set it a random number that is changed in each iteration. The number of estimators is the hyper parameters in that model, so I am working on tuning them along each iteration till I got the best f- score for the model.

2.2. I used this classifier in the learning processing, then calculating the f -score and the end of each iteration I calculate the f- score of the model and save the model (initially f -score =0), so I save models that are better than the previous one in pickle file

2.3 Go back to 2.1 till the model has the best f -score in my design

I did this process among different number of samples; I want to mention that also in this case I have as expected a very high accuracy for the same reason I have mentioned earlier.

Sample Size	Best f -score
3000	0.250
9000	0.27
16000	0.56
20000	0.42
50000	0.54

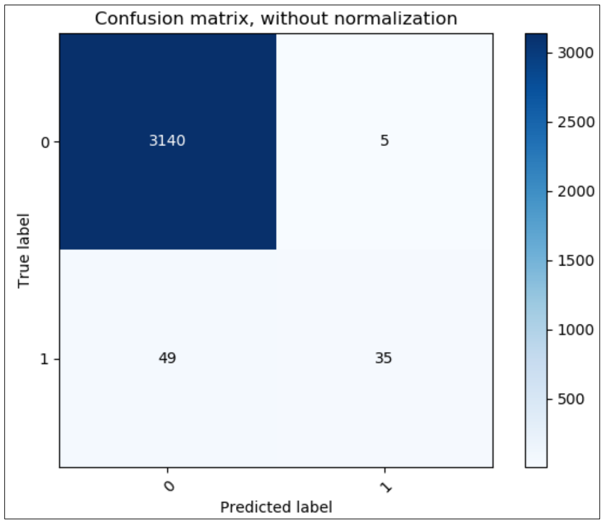


Figure 31 Confusion matrix for model 2 (16000 samples)

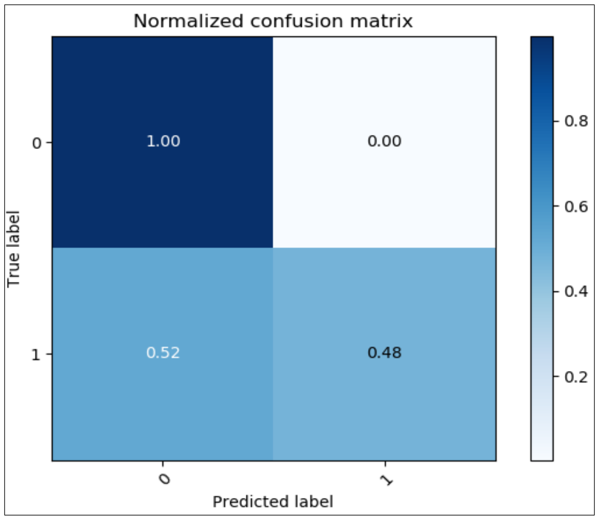


Figure 30 Normalized confusion matrix (50000)

In this model and I don't assign any base classifier for the Adaboost classifier, then it used a default base classifier which is a decision tree with randomly selected depth that it is optimized internally
And the number of estimators used in the Adaboost classifier in this model was 94 estimators (this was a random number to be set during random search), when I add a SVC as a base estimator which its kernel = '*rbf*' , the algorithm has so long time to run and the end the f -score has not exceed 0.25 in all sample sizes
Changing the **PCA** in the preprocessing to **LDA** don't make any significance change of the f- scores in case of 50,000 sample.

The next graph shows how the staged_score of the adaboost classifier changes versus the number of estimators. The staged score describes the ensemble score after each iteration of boosting and therefore allows monitoring, such as to determine the score on a test set after each boost.

In *figure 32* we can see that the Testing score starts at very low at the beginning then it increases with fluctuations in the testing phase till it be almost the same after 60 estimators for the Adaboost classifier.

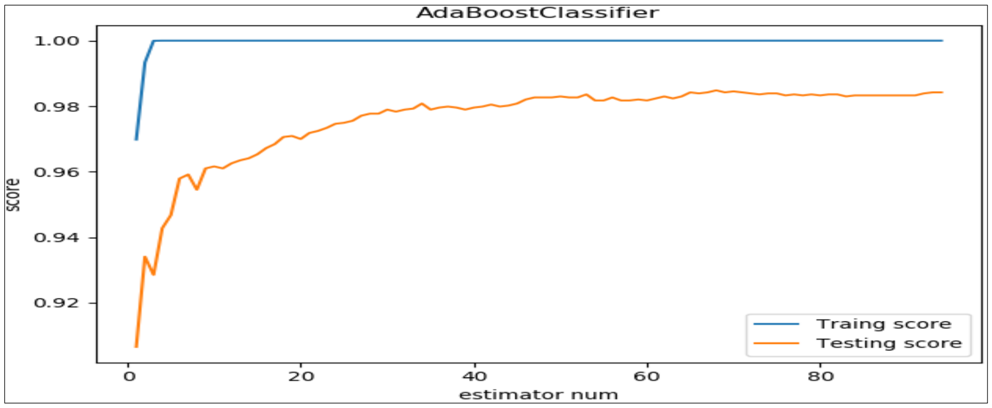


Figure 32 Staged_Score versus n_estimators for the Adaboost classifier in model 2

3. *Model (3) Using LDA for feature selection, followed by Bagging Classifier that has a base estimator of KNN*

In this model, the same exactly pre-processing steps like model 2 but instead of using PCA I used the LDA for feature selection and the end the number of features are 100 feature.

The model is built according to the following steps:
2.1. Define the Bagging classifier using the provided function in python that I mentioned earlier, the most important attribute for Bagging classifier is the number of estimators that I set it a random number that is changed in each iteration. The number of estimators is the hyper parameters in that model, so I am working on tuning them along each iteration till I got the best f -score for the model
For base classifier of bagging is the KNN classifier with n_neighborhoods = 20

2.2 I used this classifier in the learning processing, then calculating the f score and the end of each iteration
I calculate the f- score of the model and save the model (initially f -score =0), so I save models that are better than the previous one in pickle file

2.3 Go back to 2.1 till the model has the best f score in my design

The following table has the best f- score I got for different number of samples.

Sample Size	Best f- score
3000	0.08
9000	0.281
16000	0.283
20000	0.29
50000	0.311

We can see that this model has the least f -score among the previous two models
The number of estimators specified for this classifier is 150 estimators.
We can see that this estimator has the lowest f- score in all three models because K-NN is a slow algorithm: K-NN might be very easy to implement but as dataset grows efficiency or speed of algorithm declines very fast.
Moreover, due to curse of Dimensionality: KNN works well with small number of input variables but as the numbers of variables grow K-NN algorithm struggles to predict the output of new data point.
K-NN needs homogeneous features also may the optimal number of neighbors: One of the biggest issues with K-NN is to choose the optimal number of neighbors to be consider while classifying the new data entry.

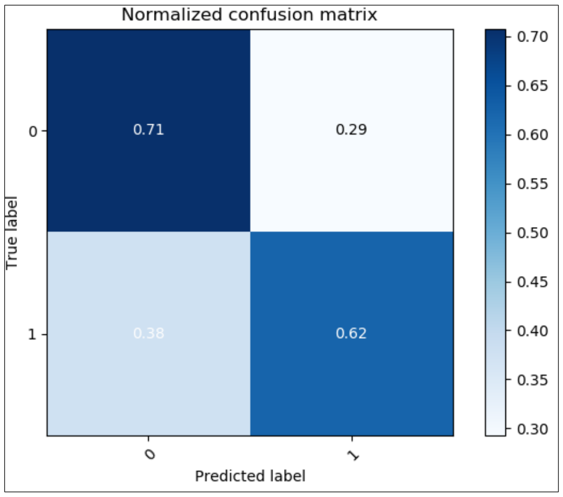


Figure 33 Normalized confusion matrix for model 3 (3000 sample)

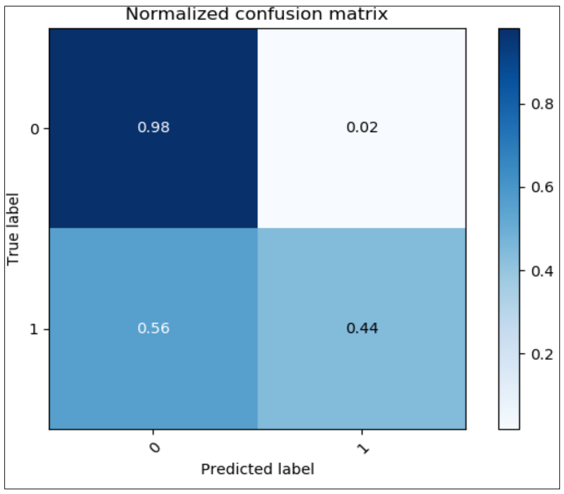


Figure 34 Normalized confusion matrix for model 3 (50,000 sample)

3.6 Results

In this project, I have worked mainly with ensemble classifiers with a random search technique to tune the hyper parameters of this classifiers and using three different feature elimination techniques , I can summarize the results that all models has a very high accuracy , which was expected from the beginning due to the quality of the dataset as it is imbalance dataset that is highly skewed and the percentage of fraud transactions is less than 4% .
The following table describes the final results for both three models in case of 50,000 sample

Models	Model 1	Model 2	Model 3
Components	RFECV + Random Forest Classifier	PCA + Adaboost Classifier (with SVC)	LDA with Bagging classifier with base KNN
Accuracy	0.98	1	0.96
F_score	0.593	0.54	0.31
Advatages	This model doesn't affect by imbalanced data, the ability to know the exact features used by model	It is diffult to interpret the features selected by this model because it is a different combination of the original ones	KNN is a very trivial algorithm that is easy to interpret
Disadvantage	Very complicated	Takes very long time	but in case of high dimensions it is very weak learner classifer Long execution time

Comparison of F -Score in these three models:

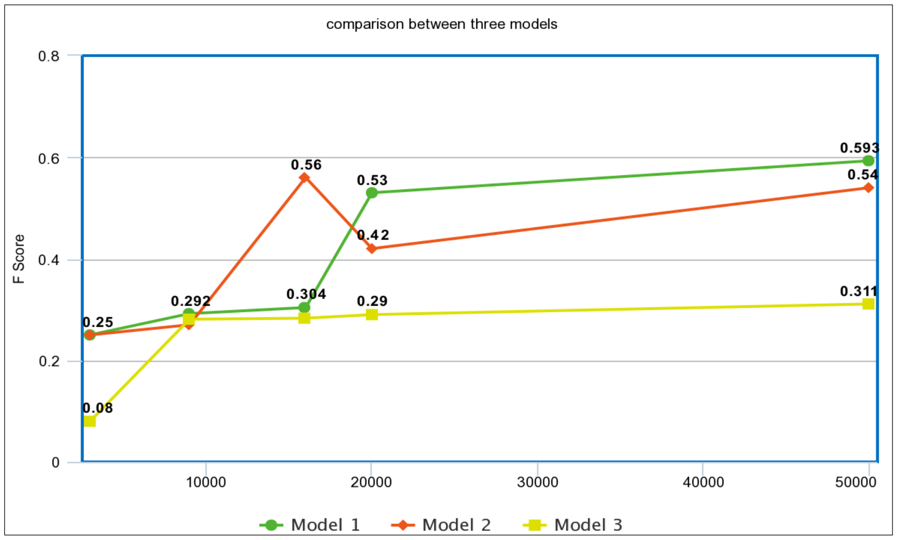


Figure 35 Performance of all models in terms of F- score

After these three models are run on credit card fraud dataset and performance of each model is evaluated with help of confusion matrix. The confusion matrix tells how the samples in training and testing models are correctly classified.
The models are evaluated based F-score as required. We explained previously the confusion matrix on defining the performance metric of the proposed models. The figures in each model section shows confusion matrix for each proposed technique. We find random forest classifier model works better as compared to the bagging classifier with base classifies KNN model because decision tree model bisects the dataset points into smaller and smaller region according to the splitting of attribute based on *Gini* index, whereas KNN fits a single point to each neighborhood which is a very hard task in higher dimensional data and it is very sensitive to noisy data even with the bagging classifier. Moreover, it is sensitive to the outlier data.
to the logistic regression as the outlier points are not handle effectively, in that case decision tree performs better. So, the comparison now is between model 1 and model 2 they have close values of f -scores in sample sizes I tried. However, model 2 is very time consuming along the 500 K data which is a very bad performance.
Finally , after run model 1 on the whole fraud dataset I got exactly 0.594 f- score and the confusion matrix shows that we have a true positive sample are 103906 sample and 1771 true negative, false positive 2089 and false negative equals to 331.For this model the precision equals to 0.98 in case of fraud transactions and 0.84 for non-fraud transactions

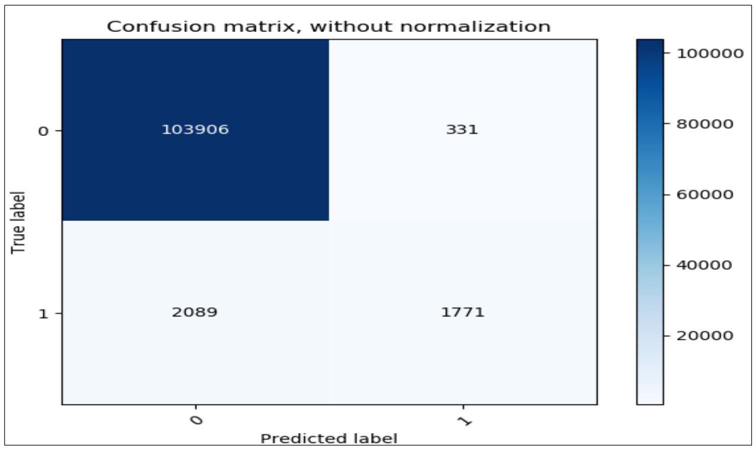


Figure 36 Confusion matrix of model 1 on whole dataset

3.7 Conclusion

The credit card fraud detection is becoming important topic of research, as different types of fraudulent activities cost billions of dollars every year. Of all types, credit card fraud is the most common and costly one which has raised huge concern globally. Detecting credit card fraud is enormously challenging. Several problems make it hard to find solutions for fraud detection, among which the class imbalance problem is one of the major ones. In this project we have proposed a robust model to process large volume of data, the functionality of model can be extended to extract real time data from different desperate sources. The extracted data is then used to build strong analytical model. To improve the analytical accuracy of fraud prediction, we have implemented three different analytical techniques. These analytical models are run on credit card dataset and accuracy of analytical model is evaluated with help of confusion matrix. Among the three models, random forest decision tree performs best in terms of f- score. We encountered a few problems during the study which need further investigation. The first is finding an optimized random forest structure.

On the other hand, the time consumption of the data training phase is difficult to deal with. Moreover, finding optimized parameters such as the number of base estimators or maximum depth of trees. Size of the data was an obvious limitation of my study as we could not conduct our study on my machine for many experiments

3.8 Integration of the proposed model in a software system

I can think about a design of a closed loop system that runs beneath the banking software (or credit card companies). This system consists of four main blocks or (stages)

- Online purchase verification server.
- Load balancer
- Classifier gateway (using my proposed model)
- Distributed databases

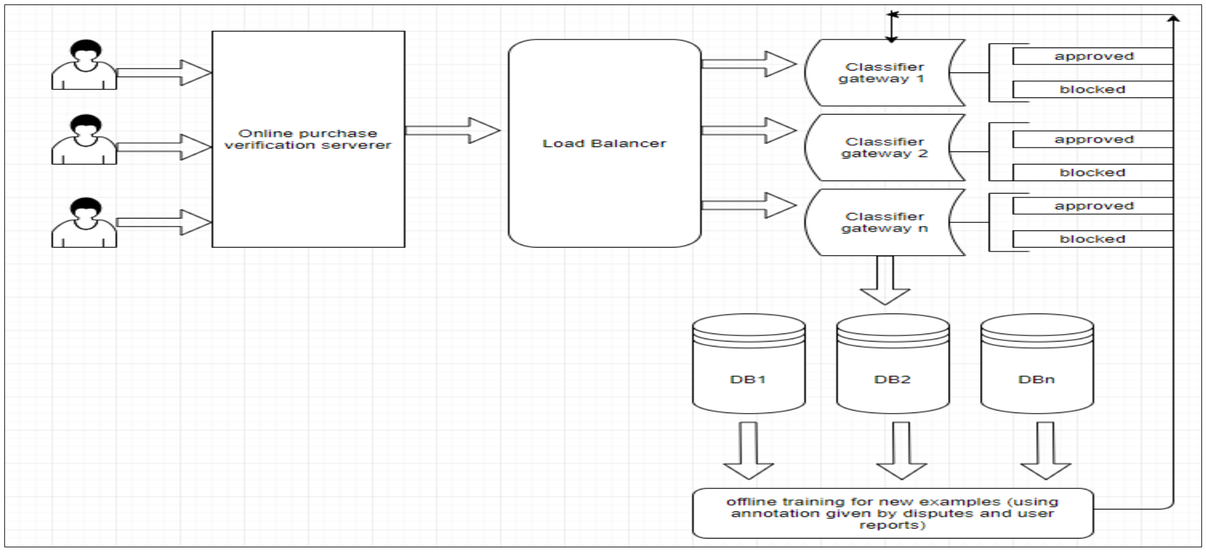


Figure 37 System design for automated fraud detection system

1) <u>Online purchase verification server</u>

I can describe it as a server that receives all the transactions that take place by card holders and verify the data of the user account user with two Ids. First is the
login ID for simply login in and doing the basic credit card transactions. Second Id is provided to the user that is the Profile-ID and its password using which the user can securely update his profile or if the users' card is stolen
All the information about credit card (Like Credit card number, credit card CVV number, credit card Expiry month and year, name on credit card etc.) will be checked with credit card database.

2) <u>Load Balancer</u>

A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.
In this manner, a load balancer performs the following functions:

- Distributes client requests or network load efficiently across multiple servers
- Ensures high availability and reliability by sending requests only to servers that are online
- Provides the flexibility to add or subtract servers as demand dictates

3) <u>Classifier Gateway</u>

This block will consist of the proposed model for classification of the transactions. Each transaction entering the database such as withdrawal, deposit, and any card transaction is treated by the classifier. The similarity between a customer’s present transaction and a known fraud scenario indicates the same fraud may occur again. Suspected transactions are flagged within seconds for further investigations. As I draw in the *figure 37* this is a closed loop system which implies that this output that the transaction is fraud or not is fed back to the classifier in case it is misclassified to enhance the classifier performance. (In other words, all transactions that disputed by users and reported to be fraud ,however, the system doesn’t classify that as a fraud one will be fed back to the classifier so that the classifier becomes more reliable and robust.

4) <u>Databases:</u>

Database management systems are specifically designed for the storage and retrieval of (large amounts of) data. Banks handle large amounts of data. A DBMS enables them to store that data, operate on it, and retrieve it when needed, fast enough for their and their customer’s needs. The database includes transactional data, historical data, and set of stored procedures. Transactions that are marked suspicious by the algorithm are kept in separate table.

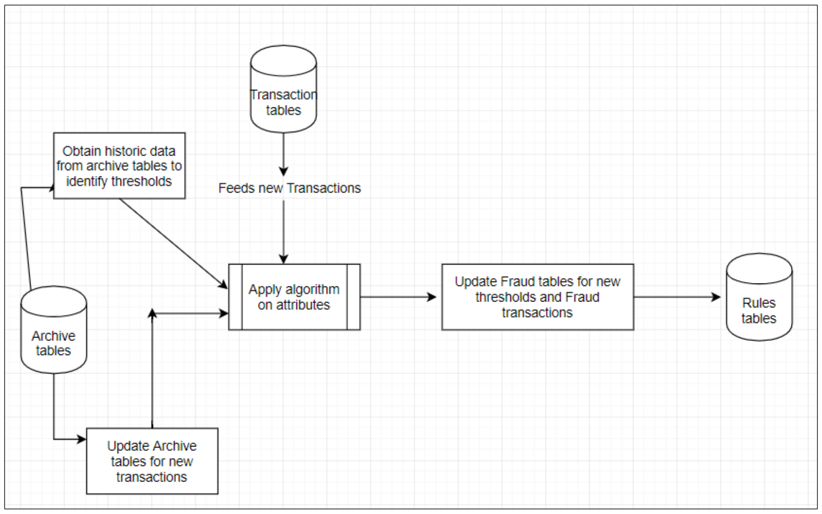

Transaction tables
Obtain historic data from archive tables to identify thresholds
Feeds new Transactions
Apply algorithm on attributes
Update Fraud tables for new thresholds and Fraud transactions
Rules tables
Archive tables
Update Archive tables for new transactions

Figure 38 Database architecture

Then the feedback of the database out is to fed to the model to start offline training with the new inserted transactions. Then, this proposed system can be considered a self learning system whereas the model is customized based on recent and past transactions. The suggested model is dynamic. Thus, the system is providing necessary support system to end users and credit card companies to freely use Plastic and electronic money.

References :

[1] Zareapoor, M.,Shamsolmoali, P. (2015). “Application of credit card fraud detection based on bagging ensemble classifier “ ICCC-2015

[2] Bhusari,V.,Patil, S. (2011).”Application of Hidden Markov Model in Credit Card Fraud Detection” .”IJDPS Vol.2,No.6”

[3] Casalino, G. , Castellano , G. and Mencar, C. “Credit card fraud detection by dynamic incremental semi-supervised
fuzzy clustering”