

Detecting Credit Card Fraud Using Supervised Machine Learning Models

Abstract

Financial fraud is an escalating issue with lasting repercussions for the financial industry. While numerous techniques have been developed to address this challenge faced by various companies, data mining has proven effective in automating the analysis of vast, complex financial datasets. Additionally, data mining has played a crucial role in detecting credit card fraud in online transactions.

Credit card fraud detection is a data mining challenge due to two key factors: the dynamic nature of normal and fraudulent behavior profiles, and the highly skewed distribution of fraud data. This paper evaluates the performance of several machine learning models, including Decision Tree, Random Forest, SVC, XGBoost, K-Nearest Neighbors, and Logistic Regression, on a highly imbalanced credit card fraud dataset. The dataset, comprising 284,786 transactions from European cardholders, is used to test these models on both raw and pre-processed data.

The performance of the models is evaluated using metrics such as accuracy, sensitivity, specificity, and precision. The results show that the optimal accuracy for the classifiers are as follows: Logistic Regression achieved 94.4%, Decision Tree 91.9%, Random Forest 92.9%, K-Nearest Neighbors 93.9%, SVC 93.4%, and XGBoost 94.95%.

Table of Contents

Chapter		Description	Page No.
1		INTRODUCTION:	
	1.1	Description	1
	1.2	Problem Formulation	4
	1.3	Proposed Solution	4
	1.4	Scope of the project	6
2		LITERATURE SURVEY	7
3		SYSTEM DESIGN	
	3.1	Functional Requirements	8
	3.2	Non functional Requirements	8
	3.3	Specific Requirements	8
	3.4	Use-Case Diagram and description	9
4		IMPLEMENTATION DETAILS	
	6.1	Algorithms/ Methods used	11
	6.2	Working of project	15
5		RESULT ANALYSIS	23
6		FUTURE SCOPE	25
7		CONCLUSION	25
8		REFERENCES	26

List of Figures

Figure No.	Figure Caption	Page No.
1	System Architecture	4
2	Activity Diagram	8
3	Flow Diagram	8
4	Use-Case Diagram	9
5	Class Diagram	9
6	Logistic Curve	10
7	SVM Model Graph	11
8	Decision Tree	12
9	Distribution of Transaction Amount and Time	16
10	Learning Curves of Models	22

1. Introduction

1.1 Description

Financial fraud is an increasingly critical issue with far-reaching consequences for governments, corporate organizations, and the financial industry. As the reliance on internet technology has grown, so have credit card transactions, both online and offline. Unfortunately, this rise has also led to a surge in credit card fraud. Given the widespread use of credit card payments, recent computational methods have been developed to address this growing problem. Numerous fraud detection solutions are available for sectors such as credit card services, retail, e-commerce, and insurance. Among these, data mining is one of the most effective techniques used to tackle credit card fraud detection.

Since it's impossible to be completely certain of the true intent behind a transaction, data mining algorithms offer an efficient way to identify potential fraud based on available data. Fraud detection in credit card transactions involves classifying transactions into two categories: legitimate and fraudulent. Various techniques, including Genetic Algorithms, Artificial Neural Networks, Frequent Itemset Mining, and Migrating Birds Optimization, have been applied to address this issue. Comparative analyses of models such as Logistic Regression, SVC, KNN, XGBoost, Decision Tree, and Random Forest have also been conducted.

Credit card transaction datasets are typically scarce, highly imbalanced, and skewed. The selection of optimal features and appropriate metrics is critical in evaluating the performance of fraud detection models on these imbalanced datasets. Several challenges make credit card fraud detection difficult, such as the dynamic nature of fraudulent behavior, which often mimics legitimate transactions. Additionally, performance is influenced by the choice of sampling approach, feature selection, and detection techniques.

The results from experiments show that Logistic Regression achieved an accuracy of 94.4%, SVC 93.4%, KNN 93.9%, Decision Tree 91.9%, and Random Forest 92.9%. XGBoost produced the highest accuracy at 94.95%. However, an analysis of learning curves revealed that XGBoost, Random Forest, and Decision Tree models tend to overfit, while KNN showed better generalization and learning ability. As a result, KNN is concluded to be the most suitable model for the system despite XGBoost's higher accuracy.

1.2 Problem Formulation

The problem formulation consists of just one sentence and should make it clear to everyone what research problem, you aim to address and to whom and where it is relevant.

Problem Formulation for our project:

Are the existing techniques used for detecting credit card frauds correctly and accurately providing efficient results or can it be improved using Machine Learning? Thus, our project tries to predict credit-card frauds using Machine Learning as it is believed to provide better results as compared to the existing techniques used to detect these frauds.

1.3 Proposed System

These are the proposed techniques used in this paper for detecting the frauds in credit card system. The comparison are made for different machine learning algorithms such as Logistic Regression, Decision Trees, Random Forest, to determine which algorithm suits best and can be adapted by credit card merchants for identifying fraud transactions. The Figure shows the architectural diagram for representing the overall system framework.[5]

Algorithm steps:	
Step 1:	Read the Dataset.
Step 2:	Random Sampling is done on the data set to make it balanced.
Step 3:	Divide the dataset into two parts i.e., Train dataset and Test dataset.
Step 4:	Accuracy and performance metrics has been calculated to know the efficiency for different algorithms.
Step 5:	Then retrieve the best algorithm based on efficiency for the given dataset.

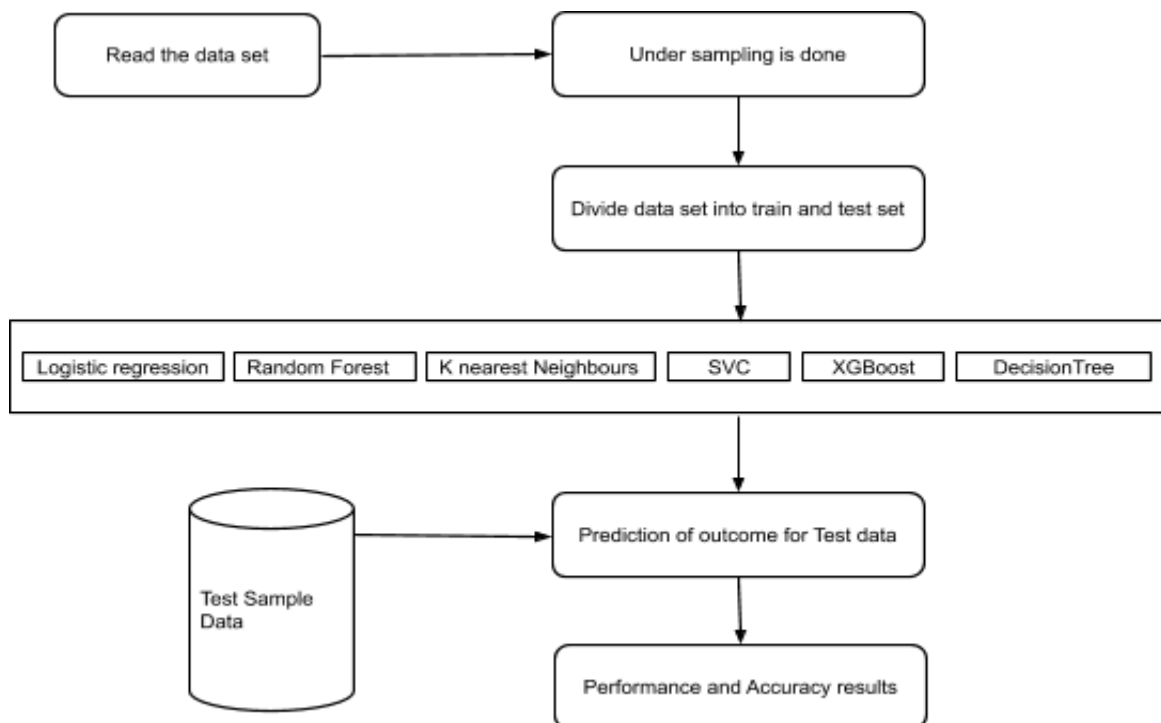


Figure 1: System Architecture

1.4 Scope of the Project

Credit card fraud detection is a very popular but also a difficult problem to solve. Firstly, due to issue of having only a limited amount of data, credit card makes it challenging to match a pattern for dataset. Secondly, there can be many entries in dataset with truncations of fraudsters which also will fit a pattern of legitimate behaviour. Also, the problem has many constraints. Firstly, data sets are not easily accessible for public and the results of researches are often hidden and censored, making the results inaccessible and due to this it is challenging to benchmarking for the models built. Secondly, the improvement of methods is more difficult by the fact that the security concern imposes a limitation to exchange of ideas and methods in fraud detection, and especially in credit card fraud detection. Lastly, the data sets are continuously evolving and changing making the profiles of normal and fraudulent behaviours always different, that is, a legit transaction in the past may be a fraud in present or vice versa. We evaluate six advanced data mining approaches, Logistic Regression, K-Nearest Neighbours, Support Vector Classifiers, Decision Tree and Random Forests and XGBoost and then a collative comparison is made to evaluate which model performed best.

2. Literature Survey

Credit card fraud detection has drawn a lot of research interest and a number of techniques, with special emphasis on neural networks, data mining and distributed data mining have been suggested.

There have been various studies on credit card fraud detection. Machine learning and related methods are most commonly used, which include artificial neural networks, rule-induction techniques, decision trees, logistic regression, and support vector machines[1]

Behrouz, Emad and Sahil [2] have proposed credit card fraud detection using supervised machine learning algorithms. They have employed these algorithms to implement a super classifier using ensemble learning methods. Also, they compared and discussed the performance of various supervised machine learning algorithms that exist in literature against the super classifier that they implemented.

Navanshu Khare and Saad Yunus Sait[3] have explained the mathematics and working of every algorithm we have used in this paper.

Siddhartha Bhattacharyya and 4 others [6] in their paper in 2011 did a detailed comparative study of Support vector machine and random forest along with logistic regression. They concluded through experiments that Random Forest technique shows most accuracy followed by Logistic Regression and Support Vector Machine.

Y. Sahin and E. Duman [7] proposed fraud detection in credit card using a combination of Support Vector Machines and Decision Trees. Decision Trees outperformed SVMs when the size of data set was small but with increase in size of dataset SVM reached accuracy over decision trees.

3. System Design

3.1 Functional Requirements

- The model should be able to give accurate and trustworthy predictions.
- The application must show graphical visualization of the predicted results and data in general.
- The user should be able to enter the input values for prediction.

3.2 Non Functional Requirements

Non Functional Requirements will describe how a system should behave and what limits are constrained on its functionality. It generally specifies the system's quality attributes or characteristics.

- Availability: The system should be available to any transaction verification system.
- Correctness: The accuracy of the system should be as maximum as possible for better prediction.
- Maintainability: The system should maintain correct history of records.
- Usability: The system should satisfy a maximum number of banking system needs.

3.3 Specific Requirements

Software requirements

- | | |
|----------------------|-------------------------------|
| 1. Languages | : Python-3 |
| 2. Operating Systems | : Windows, Linux, etc. |
| 3. Back End Software | : Anaconda, Jupyter notebook. |

Hardware Requirements:

- | | |
|--------------------|---------------------------|
| 1. CPU | : Intel Pentium IV 600MHz |
| 2. Hard disk space | : 20 GB or more |
| 3. Memory | : 4 GB RAM |

3.4 Block Diagram, Use Case, class diagram, etc

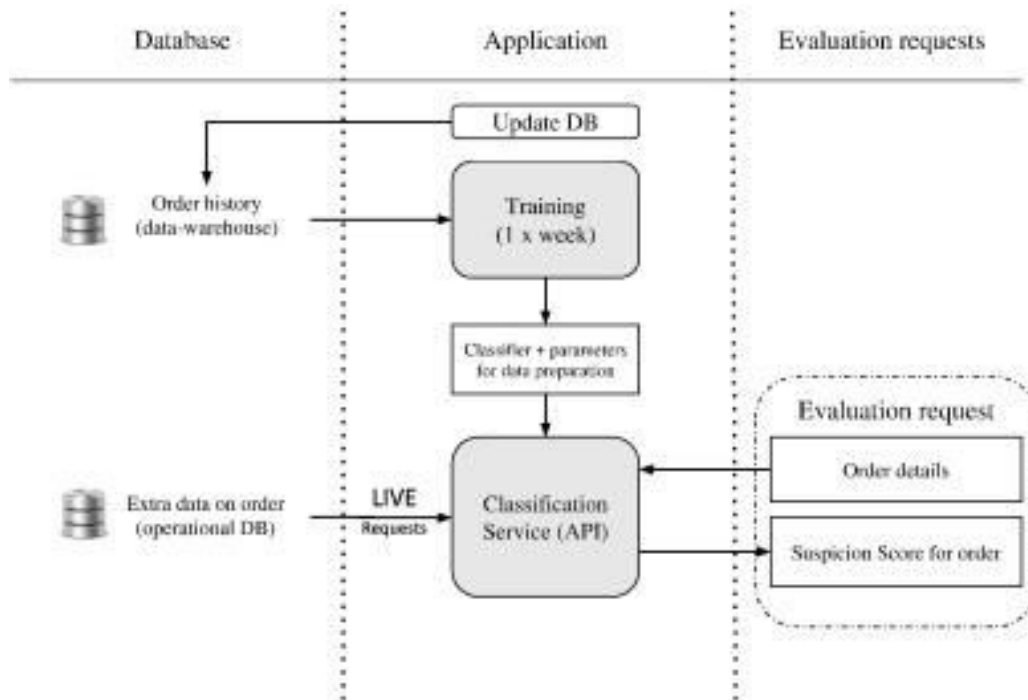


Figure 2: Activity diagram

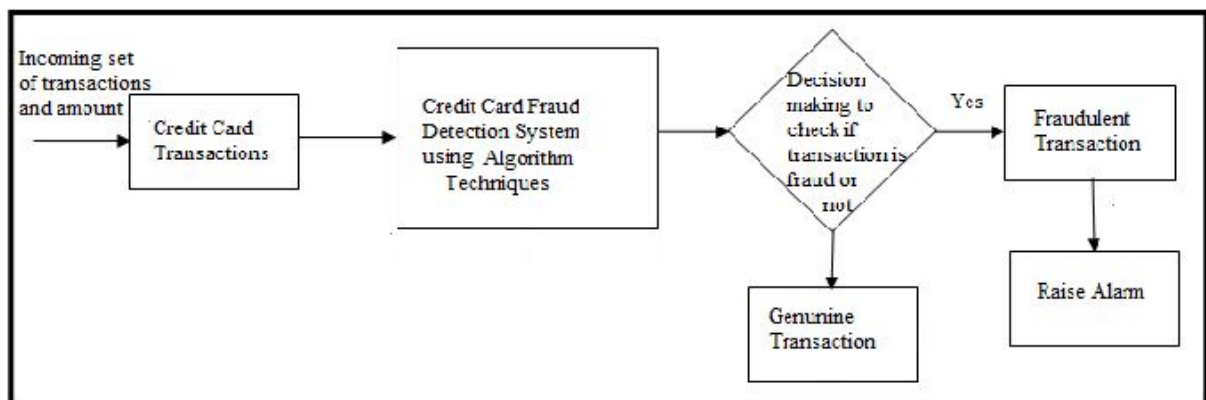


Figure 3: Flow Diagram

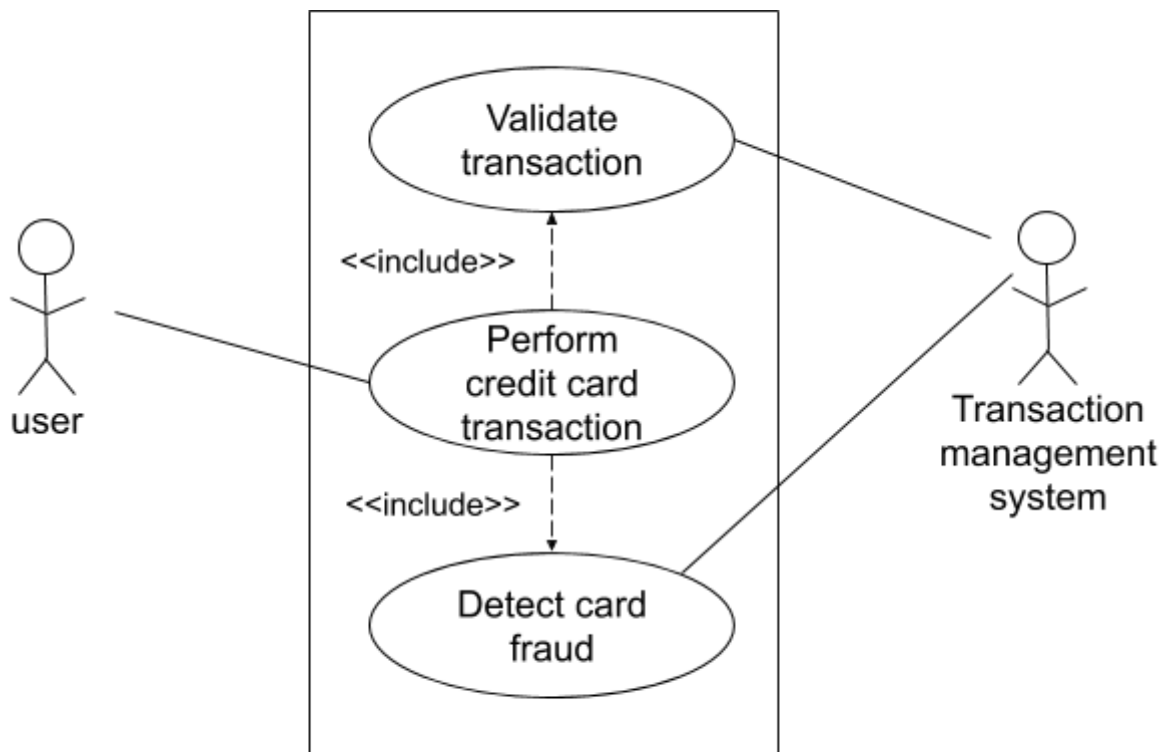


Figure 4: Use Case diagram for fraud detection

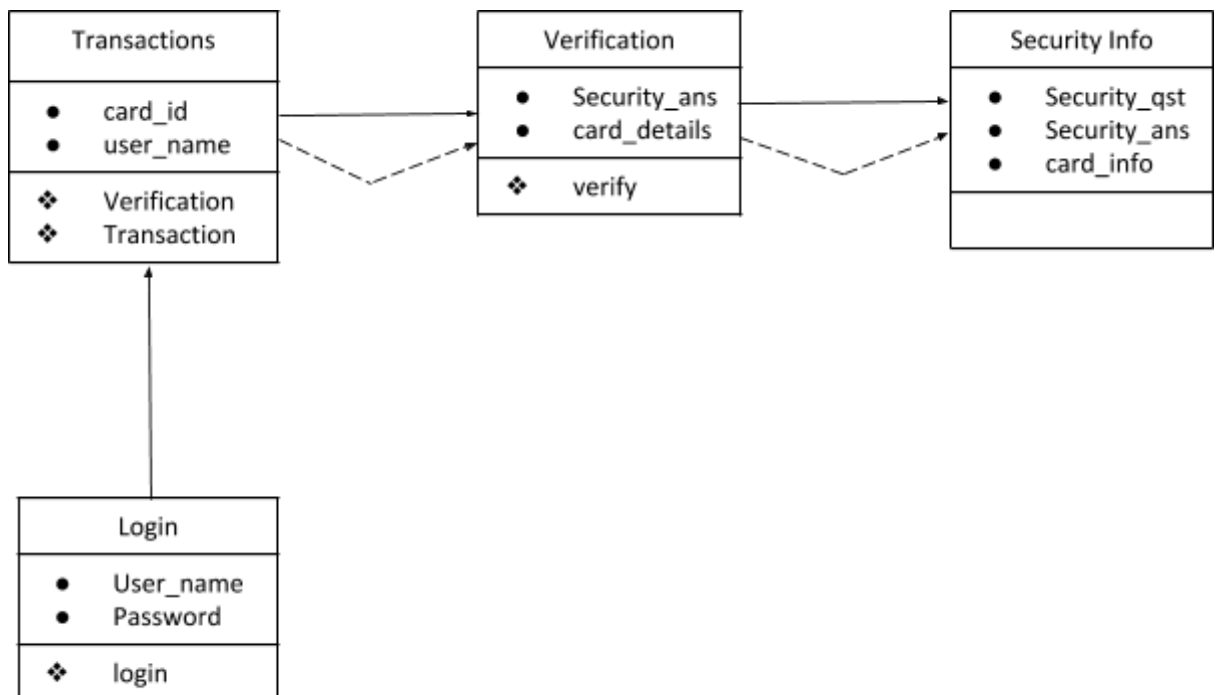


Figure 5: Class Diagram

4. Implementation Details

4.1 Algorithms

Ability of system to automatically learn and improve from experience without being explicitly programmed is called machine learning and it focuses on the development of computer programs that can access data and use it to learn by themselves. And classifier can be stated as an algorithm that is used to implement classification especially in concrete implementation, it also refers to a mathematical function implemented by algorithm that will map input data into category. It is an instance of supervised learning i.e. where training set of correctly identified observations is available.

A. Logistic Regression

Logistic Regression is a supervised classification method that returns the probability of binary dependent variable that is predicted from the independent variable of dataset i.e. logistic regression predicts the probability of an outcome which has two values, either zero or one, no or yes and false or true. Logistic regression has similarities to linear regression, but, in linear regression a straight line is obtained, logistic regression shows a curve. The use of one or several predictors or independent variable is on what prediction is based, logistic regression produces logistic curves which plots the values between zero and one.

Logistic Regression is a regression model where the dependent variable is categorical and analyzes the relationship between multiple independent variables. There are many types of logistic regression model such as binary logistic model, multiple logistic model, binomial logistic models. Binary Logistic Regression model is used to estimate the probability of a binary response based on one or more predictors.

$$p = \frac{e^{\alpha + \beta_n X}}{1 + e^{\alpha + \beta_n X}}$$

Above equation represents the logistic regression in mathematical form.

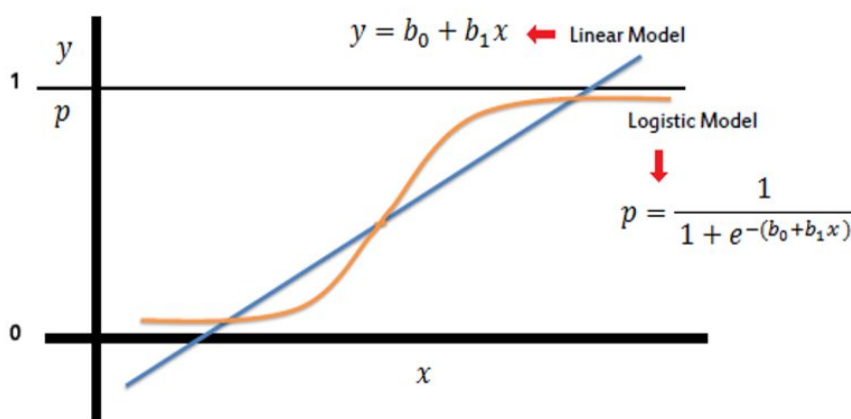


Figure 6: Logistic Curve

This graph shows the difference between linear regression and logistic regression where logistic regression shows a curve but linear regression represents a straight line.

B. SVM Model (Support Vector Machine)

SVM is a one of the popular machine learning algorithm for regression, classification. It is a supervised learning algorithm that analyses data used for classification and regression. SVM modeling involves two steps, firstly to train a data set and to obtain a model & then, to use this model to predict information of a testing data set. A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane where SVM model represents the training data points as points in space and then mapping is done so that the points which are of different classes are divided by a gap that is as wide as possible. Mapping is done in to the same space for new data points and then predicted on which side of the gap they fall.

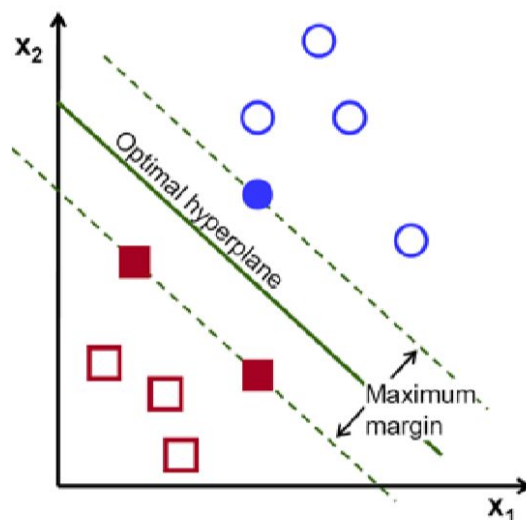


Figure 7: SVM Model Graph

In SVM algorithm, plotting is done as each data item is taken as a point in n-dimensional space where n is number of features, with the value of each feature being the value of a particular coordinate. Then, classification is performed by locating the hyperplane that separates the two classes very well.

C. Decision Tree

Decision tree is an algorithm that uses a tree like graph or model of decisions and their possible outcomes to predict the final decision, this algorithm uses conditional control statement. A Decision tree is an algorithm for approaching discrete-valued target functions, in which decision tree is denoted by a learned function. For inductive learning these types of algorithms are very famous and have been successfully applied to a broad range of tasks.

We give label to a new transaction that is whether it is legit or fraudulent for which class label is unknown and then transaction value is tested against the decision tree, and after that from root node to output/class label for that transaction a path is traced.

$$Entropy(S) = \sum_{i=1} -p_i \log_2 p_i$$

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Decision rules determines the outcome of the content of leaf node. In general rules have the form of ‘If condition 1 and condition 2 but not condition 3 then outcome’. Decision tree helps to determine the worst, best and expected values for different scenarios, simplified to understand and interpret and allows addition of new possible scenarios.

Steps for making a decision tree are: to calculate the entropy of every attribute using the dataset in problem, then, dataset is divided into subsets using the attribute for which gain is maximum or entropy is minimum, after that, to make a decision tree node containing that attribute and lastly recursion is performed on subsets using remaining attributes to create a decision tree.

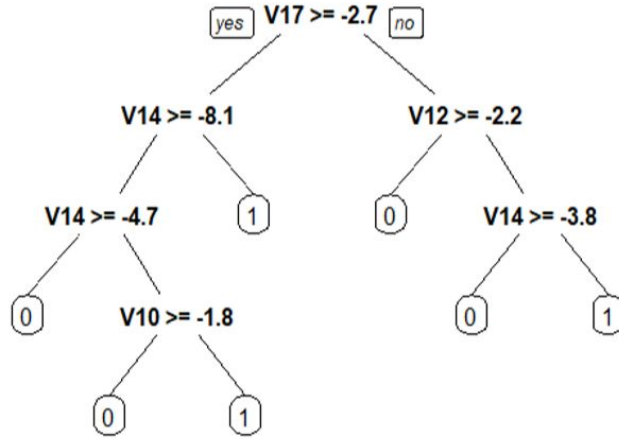


Figure 8: Decision tree

D. Random Forest

Random Forest is an algorithm for classification and regression. Summarily, it is a collection of decision tree classifiers. Random forest has advantage over decision tree as it corrects the habit of overfitting to their training set. A subset of the training set is sampled randomly so that to train each individual tree and then a decision tree is built, each node then splits on a feature selected from a random subset of the full feature set. Even for large data sets with many features and data instances training is extremely fast in random forest and because each tree is trained independently of the others. The Random Forest algorithm has been found to provides a good estimate of the generalization error and to be resistant to overfitting. Random forest ranks the importance of variables in a regression or classification problem in a natural way can be done by Random Forest.

E. K-Nearest Neighbour Classifier

The k-nearest neighbour is an instance based learning which carries out its classification based on a similarity measure, like Euclidean, Manhattan or Minkowski distance functions. The first two distance measures work well with continuous variables while the third suits categorical variables. The Euclidean distance measure is used in this study for the kNN classifier. The Euclidean distance (D_{ij}) between two input vectors (X_i, X_j) is given by:

$$D_{ij} = \sqrt{\sum_{k=1}^n (X_{ik} - X_{jk})^2} \quad k=1,2,\dots,n$$

For every data point in the dataset, the Euclidean distance between an input data point and current point is calculated. These distances are sorted in increasing order and k items with lowest distances to the input data point are selected. The majority class among these items is found and the classifier returns the majority class as the classification for the input point. Parameter tuning for k is carried out for k = 1, 3, 5, 7, 9, 11 and k = 3 showed optimal performance. Thus, value of k = 3 is used in the classifier.

F. XGBoost

XGBoost is a short form for Extreme Gradient Boosting. Boosting is a sequential process. Multiple trees are created and the information of the first tree is fed as input to the second tree so that it improves the prediction in subsequent iterations. Basically it is an additive tree model where it adds new trees that complement the already built ones. XGBoost handles missing values and it works only for numeric data.

4.2 Working of the project

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Dataset

The datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numeric input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the

first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise. There are no "Null" values, so we don't have to work on ways to replace values.

```
...: dataset.describe()
Out[1]:
```

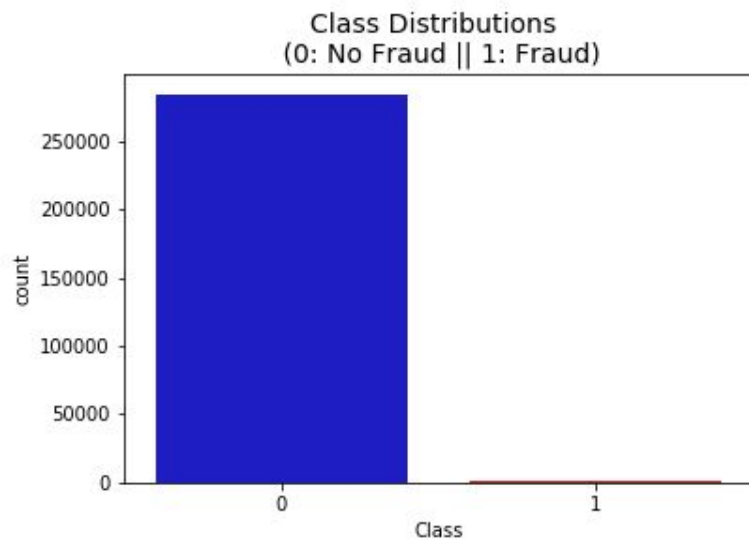
	Time	V1	...	Amount	Class
count	284807.000000	2.848070e+05	...	284807.000000	284807.000000
mean	94813.859575	3.919560e-15	...	88.349619	0.001727
std	47488.145955	1.958696e+00	...	250.120109	0.041527
min	0.000000	-5.640751e+01	...	0.000000	0.000000
25%	54201.500000	-9.203734e-01	...	5.600000	0.000000
50%	84692.000000	1.810880e-02	...	22.000000	0.000000
75%	139320.500000	1.315642e+00	...	77.165000	0.000000
max	172792.000000	2.454930e+00	...	25691.160000	1.000000

```
In [2]: dataset.head()
Out[2]:
```

	Time	V1	V2	V3	...	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	...	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	...	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	...	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	...	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	...	0.219422	0.215153	69.99	0

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

Why directly running prediction on the dataset is not a good idea?



No frauds 99.83% of dataset

Frauds 0.17% of dataset

Notice how imbalanced is our original dataset is! Most of the transactions are non-fraud. If we use this data frame as the base for our predictive models and analysis, we might get a lot of errors and our algorithms will probably overfit since it will "assume" that most transactions are not fraud. But we don't want our model to assume, we want our model to detect patterns that give signs of fraud!

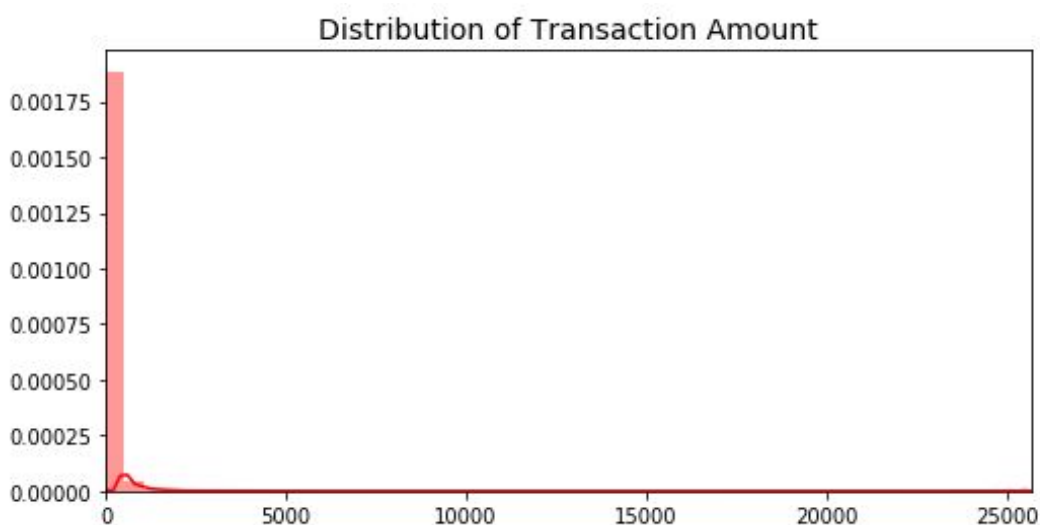
Credit card dataset is highly imbalanced dataset because it carries more legitimate transactions as compared to the fraudulent one. That means prediction will get very high accuracy score without detecting a fraud transaction. When we apply any classification algorithm directly on the dataset we get the following results.

Directly applying the classification:

KNearest [[56864 0] [93 5]] accuracy_score: 0.9983673326077034 miscalssification: 0.0016326673922966162 Sensitivity : 1.0 Specificity : 0.05102040816326531 precision_score: 1.0 f1_score: 0.09708737864077671	XGBClassifier [[56858 6] [18 80]] accuracy_score: 0.9995786664794073 miscalssification: 0.0004213335205927038 Sensitivity : 0.9998944850872257 Specificity : 0.8163265306122449 precision_score: 0.9302325581395349 f1_score: 0.8695652173913043
LogisticRegression [[56853 11] [46 52]] accuracy_score: 0.9989993328885924 miscalssification: 0.0010006671114075605 Sensitivity : 0.9998065559932471 Specificity : 0.5306122448979592 precision_score: 0.8253968253968254 f1_score: 0.6459627329192548	RandomForestClassifier [[56862 2] [21 77]] accuracy_score: 0.9995962220427653 miscalssification: 0.00040377795723467447 Sensitivity : 0.9999648283624085 Specificity : 0.7857142857142857 precision_score: 0.9746835443037974 f1_score: 0.8700564971751412

The results deal with a problem called accuracy paradox. The ACCURACY PARADOX is the paradoxical finding that accuracy is not a good metric for predictive models when classifying in predictive analytics. This is because a simple model may have a high level of accuracy but be too crude to be useful. For example, if the incidence of category A is dominant, being found in 99% of cases, then predicting that every case is category A will have an accuracy of 99%. Precision and recall are better measures in such cases. The underlying issue is that there is a class imbalance between the positive class and the negative class. Prior probabilities for these classes need to be accounted for in error analysis. Precision and recall help, but precision too can be biased by very unbalanced class priors in the test sets.

Also, by seeing the distributions, we can have an idea how skewed are these features. There are techniques that can help the distributions be less skewed which will be implemented in this report in the future.



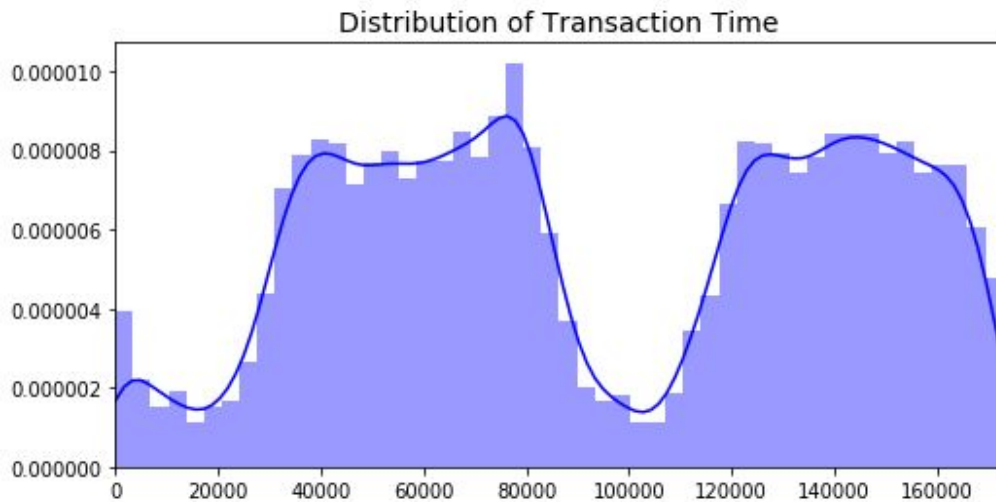


Figure 9: Distribution of Transaction Amount and Time

To handle this kind of problem one better way is to class distribution, i.e., sampling minority classes. In sampling minority, class training example can be increased in proportion to the majority class to raise the chance of correct prediction by the algorithm.

Scaling and Distributing:

In this phase, we will first scale the columns comprise of Time and Amount. Time and amount should be scaled as the other columns. On the other hand, we need to also create a subsample of the data frame in order to have an equal amount of Fraud and Non-Fraud cases, helping our algorithms better understand patterns that determines whether a transaction is a fraud or not.

In this scenario, our subsample will be a data frame with a 50/50 ratio of fraud and non-fraud transactions. Meaning our sub-sample will have the same amount of fraud and non-fraud transactions.

We need to create a sub sample because we saw that the original data frame was heavily imbalanced! Using the original data frame will cause the following issues:

1. Overfitting: Our classification models will assume that in most cases there are no frauds! What we want for our model is to be certain when a fraud occurs.
2. Wrong Correlations: Although we don't know what the "V" features stand for, it will be useful to understand how each of this feature influence the result (Fraud or No Fraud) by having an imbalance data frame we are not able to see the true correlations between the class and features.

We use RobustScaler to scale time and amount, the reason being that they scale features using statistics that are robust to outliers.

This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Median and interquartile range are then stored to be used on later data using the transform method.

Standardization of a dataset is a common requirement for many machine learning estimators. Typically, this is done by removing the mean and scaling to unit variance.

However, outliers can often influence the sample mean / variance in a negative way. In such cases, the median and the interquartile range often give better results.

As RobustScaler, QuantileTransformer is robust to outliers in the sense that adding or removing outliers in the training set will yield approximately the same transformation on held out data. But contrary to RobustScaler, QuantileTransformer will also automatically collapse any outlier by setting them to the a priori defined range boundaries (0 and 1).

Splitting the Data (Original DataFrame)

Summary:

- There are 492 cases of fraud in our dataset so we can randomly get 492 cases of non-fraud to create our new sub data frame.
- We concatenate the 492 cases of fraud and non-fraud, creating a new sub-sample.

Scaled amount and scaled time are the columns with scaled values:

```
from sklearn.preprocessing import StandardScaler, RobustScaler
rob_scaler = RobustScaler()
dataset['scaled_amount'] = rob_scaler.fit_transform(dataset['Amount'].values.reshape(-1,1))
dataset['scaled_time'] = rob_scaler.fit_transform(dataset['Time'].values.reshape(-1,1))
dataset.drop(['Time','Amount'], axis=1, inplace=True)
scaled_amount = dataset['scaled_amount']
scaled_time = dataset['scaled_time']
dataset.drop(['scaled_amount', 'scaled_time'], axis=1, inplace=True)
dataset.insert(0, 'scaled_amount', scaled_amount)
dataset.insert(1, 'scaled_time', scaled_time)
# Amount and Time are Scaled!
```

```
In [6]: dataset.head()
```

```
Out[6]:
```

	scaled_amount	scaled_time	V1	...	V27	V28	Class
42948	-0.167680	-0.509675	-0.256652	...	-0.547145	-0.380498	0
74625	-0.293579	-0.340923	-0.357039	...	0.243853	0.085171	0
236178	-0.296793	0.751959	-0.443426	...	-0.665420	-0.268262	0
192669	0.041920	0.529940	2.334822	...	0.006936	-0.071813	0
15736	1.089779	-0.675866	-23.914101	...	1.458076	0.430315	1

Before proceeding with the Random Under Sampling technique we have to separate the original data frame for testing purposes, although we are splitting the data when implementing Random Under Sampling or Oversampling techniques, we want to test our models on the original testing set, not on the testing set created by either of these techniques. The main goal is to fit the model either with the data frames that were under sample and oversample (in order for our models to detect the patterns), and test it on the original testing set.

One of the most common and simplest strategies to handle imbalanced data is to under sample the majority class.

Oversampling the minority class can result in overfitting problems if we oversample before cross-validating.

```
In [8]: X = dataset.drop('Class', axis=1)
...: y = dataset[['Class']]
...: from sklearn.model_selection import StratifiedShuffleSplit
...: sss = StratifiedShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
...: for train_index, test_index in sss.split(X, y):
...:     print("Train:", train_index, "Test:", test_index)
...:     original_Xtrain, original_Xtest = X.iloc[train_index], X.iloc[test_index]
...:     original_ytrain, original_ytest = y.iloc[train_index], y.iloc[test_index]
...:
...:     original_Xtrain = original_Xtrain.values
...:     original_Xtest = original_Xtest.values
...:     original_ytrain = original_ytrain.values
...:     original_ytest = original_ytest.values
...:     train_unique_label, train_counts_label = np.unique(original_ytrain,
return_counts=True)
...:     test_unique_label, test_counts_label = np.unique(original_ytest,
return_counts=True)
...:     print('-' * 100)
...:     print('Label Distributions: \n')
...:     print(train_counts_label/ len(original_ytrain))
...:     print(test_counts_label/ len(original_ytest))
```

```

Train: [265518 180305 42664 ... 29062 13766 17677] Test: [263020 11378 147283 ... 274532 269819 64170]
Train: [ 72227 114282 16818 ... 264471 191914 284017] Test: [202638 32978 128121 ... 244024 127667 48318]
Train: [ 20895 114622 167683 ... 244502 178972 218506] Test: [284352 82483 90981 ... 171224 168807 271602]
Train: [122248 181660 194400 ... 104631 277586 29432] Test: [225673 63348 68025 ... 279451 77554 76043]
Train: [241684 223467 136928 ... 86495 160550 49633] Test: [157557 204860 83760 ... 251478 178967 216850]

```

Label Distributions:

```

[0.99827075 0.00172925]
[0.99827955 0.00172045]

```

Under Sampling :

In this phase of the project we will implement "*Under Sampling*" which basically consists of removing data in order to have a more balanced dataset and thus avoiding our models to overfitting.[4]

- The first thing we have to do is determine how imbalanced is our class (use "value_counts()" on the class column to determine the amount for each label)
- Once we determine how many instances are considered fraud transactions (Fraud = "1") , we should bring the non-fraud transactions to the same amount as fraud transactions (assuming we want a 50/50 ratio), this will be equivalent to 492 cases of fraud and 492 cases of non-fraud transactions.
- After implementing this technique, we have a sub-sample of our data frame with a 50/50 ratio with regards to our classes. Then the next step we will implement is to shuffle the data to see if our models can maintain a certain accuracy every time, we run this script.

Note: The main issue with "Random Under-Sampling" is that we run the risk that our classification models will not perform as accurate as we would like to since there is a great deal of information loss (bringing 492 non-fraud transaction from 284,315 non-fraud transaction)


```

In [9]: dataset = dataset.sample(frac=1)
...: # amount of fraud classes 492 rows.
...: fraud_df = dataset.loc[dataset['Class'] == 1]
...: non_fraud_df = dataset.loc[dataset['Class'] == 0][:497]
...: normal_distributed_df = pd.concat([fraud_df, non_fraud_df])
...: # Shuffle dataframe rows
...: new_df = normal_distributed_df.sample(frac=1, random_state=42)
...: new_df.head()
...: new_df.describe()
...: print('Distribution of the Classes in the subsample dataset')
...: print(new_df['Class'].value_counts()/len(new_df))

```

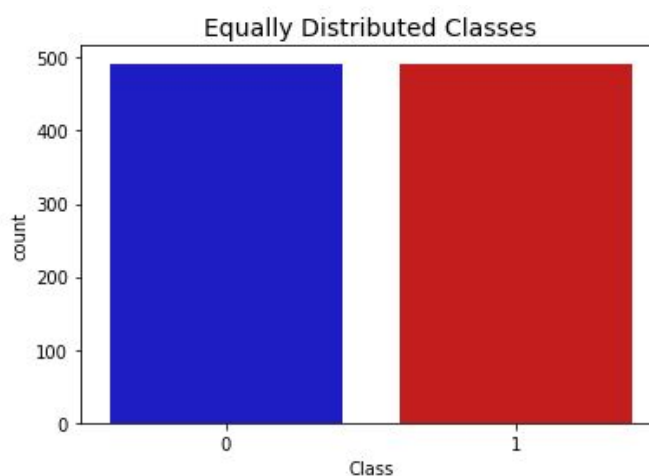
Distribution of the Classes in the subsample dataset

0 0.502528

1 0.497472

Name: Class, dtype: float64

Now that we have our dataframe correctly balanced, we can go further with our **analysis** and **data preprocessing**.



Now that we have sampled our data set, we apply our supervised learning models.

```

In [26]: classifiers = {
...:     "RandomForestClassifier": RandomForestClassifier(n_estimators=300),
...:     "LogisticRegression": LogisticRegression(),
...:     "KNearest": KNeighborsClassifier(),
...:     "Support Vector Classifier": SVC(),
...:     "DecisionTreeClassifier": DecisionTreeClassifier(),
...:     "XGBClassifier": XGBClassifier()
...: }

```

```

In [30]: for key, classifier in classifiers.items():
...:     classifier.fit(X_train, y_train)
...:     y_pred = classifier.predict(X_test)
...:     #metrics
...:     print("Classifiers: ", classifier.__class__.__name__)
...:     cm=confusion_matrix(y_test, y_pred)
...:     print(cm)
Classifiers:  DecisionTreeClassifier
[[84  7]
 [11 96]]
Classifiers:  RandomForestClassifier
[[90  1]
 [14 93]]
Classifiers:  LogisticRegression
[[88  3]
 [13 94]]
Classifiers:  KNeighborsClassifier
[[90  1]
 [16 91]]
Classifiers:  SVC
[[90  1]
 [14 93]]
Classifiers:  XGBClassifier
[[86  5]
 [12 95]]

```

5. Result Analysis

Learning Curves:

- The wider the gap between the training score and the cross-validation score, the more likely your model is overfitting (high variance).
- If the score is low in both training and cross-validation sets this is an indication that our model is underfitting (high bias)
- K Nearest Neighbour Classifier shows the best score in both training and cross-validating sets.

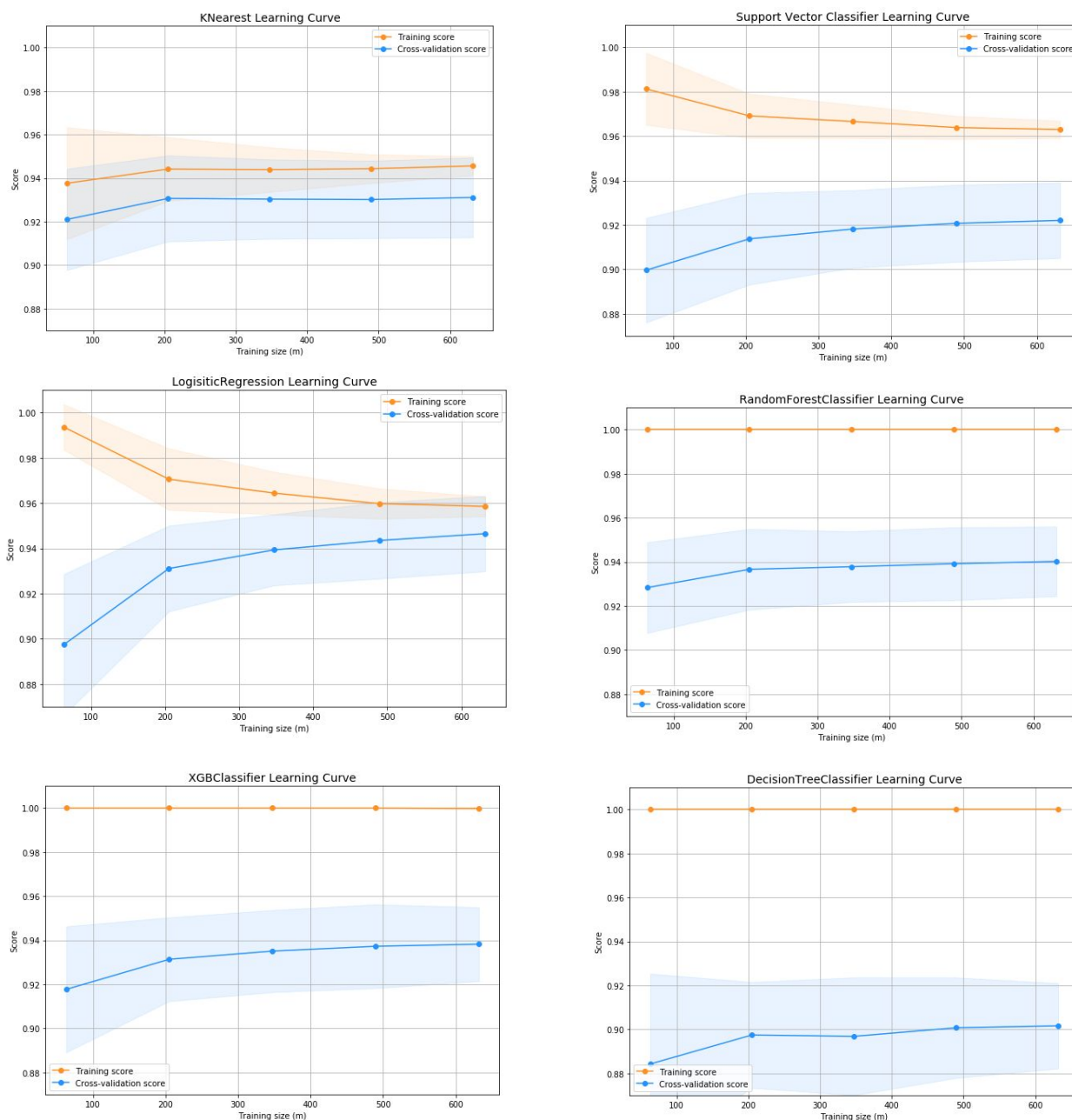


Figure 10: Learning Curves of Models

Metrics:

In order to compare various techniques we calculate the true positive, true negative, false positive and false negative generated by a system or an algorithm and use these in quantitative measurements to evaluate and compare performance of different systems.

True Positive (TP) is number of transactions that were fraudulent and were also classified as fraudulent by the system. True Negative (TN) is number of transactions that were legitimate and were also classified as legitimate. False Positive (FP) is number of transactions that were legitimate but were wrongly classified as fraudulent transactions. False Negative (FN) is number of transactions that were fraudulent but were wrongly classified as legitimate transactions by the system.

Table 1: Basic Metrics for sampled data distribution

Metrics	Classifiers					
	Random Forest	Decision Tree	XGBoost	KNearest	SVC	Logistic Regression
True Positive	90	86	90	91	87	90
False Positive	1	5	1	0	4	1
False Negative	13	11	9	12	9	10
True Negative	94	96	98	95	98	97

Table 2: Accuracy results for sampled data distribution

Metrics	Classifiers					
	Random Forest	Decision Tree	XGBoost	KNearest	SVC	Logistic Regression
Accuracy	0.9293	0.9192	0.9495	0.9394	0.9343	0.9444
Sensitivity	0.8785	0.8972	0.9159	0.8878	0.9159	0.9065
Specificity	0.9890	0.9341	0.98901	1.0	0.9560	0.9890
Precision	0.9895	0.9505	0.9899	1.0	0.9608	0.9898
F1 score	0.93069	0.9231	0.9515	0.9406	0.9378	0.9463

From the above table it is clear that the XGboost gives the highest accuracy followed by Logistic regression, SVC and KNN.

6. Future scope

With increasing number of bank fraudulency and cyber crime cases, need of a secure testing system is on rise. And this is a direct solution to this problem. It can be extended to a duplex verification of not only a customer(debit-ant) but also of the seller(credit-ant). It can be taken and used on a regular basis just like OTP. It can be used to even assess past transaction in database to find whether certain transactions were fraudulent or not and also would be able to produce evidence in such cases. Also optimization techniques on the proposed models and testing on new models can be done.

7. Conclusion

Although there are several fraud detection techniques available today, none is able to detect all frauds completely when they are actually happening, they usually detect it after the fraud has been committed. This happens because a very minuscule number of transactions from the total transactions are actually fraudulent in nature. So we need a technology that can detect the fraudulent transaction when it is taking place so that it can be stopped then and there and that too in a minimum cost. So the major task of today is to build an accurate, precise and fast detecting fraud detection system for credit card frauds that can detect not only frauds happening over the internet like phishing and site cloning but also tampering with the credit card itself i.e. it signals an alarm when the tampered credit card is being used. The major drawback of all the techniques is that they are not guaranteed to give the same results in all environments. They give better results with a particular type of dataset and poor or unsatisfactory results with other type. Thus, the results are purely dependent on the dataset type used.

In our undersample data, our model is unable to detect for a large number of cases, the non fraud transactions correctly and instead, mis-classifies those non fraud transactions as fraud cases. Imagine that people that were making regular purchases got their card blocked due to the reason that our model classified that transaction as a fraud transaction, this will be a huge disadvantage for the financial institution. The number of customer complaints and customer dissatisfaction will increase. The next step of this analysis will be to do an outlier removal on our undersample dataset and see if our accuracy in the test set improves.

In this paper, Machine learning technique like Logistic regression, Decision Tree, Random forest, K-Nearest Neighbours, SVC and XGBoost classifiers were used to detect the fraud in credit card system. Sensitivity, Specificity, accuracy and error rate are used to evaluate the performance for the proposed system. From the experiments, the result that has been concluded is that Logistic regression has an accuracy of 94.4% while SVC shows accuracy of 93.4%, KNN with 93.9% and Decision tree shows accuracy of 91.9% and Random forest shows accuracy of 92.9% but the best results are obtained by XGBoost with a precise accuracy of 94.95%. However when the learning curves of all the classifiers are evaluated we see that XGBoost overfits along with Random forest and decision tree and only KNN is able to learn whereas others underfit. Hence we conclude that KNN is the best model for our system.