CREDIT CARD FRAUD DETECTION USING PYTHON

CONTENT

1. INTRODUCTION

In the recent years, the fast development of e-commerce technologies made it possible for people to select the most desirable items in terms of suggested price, quality and quantity among various services, facilities, shops and stores from all around the world. However, it also made it easier for fraudsters to abuse this huge opportunity. As credit card has become the most popular mode of payment, the fraudulent activities using credit card payment technologies are rapidly increasing as a result. Credit card fraud is usually caused either by card owner's negligence with his data or by a breach in a website's security. Fraudulent credit cards are usually destroyed by the criminals after several successful payments, just before a victim realizes the problem and reports it. Fraud detection is a set of activities that are taken to prevent money or property from being obtained through false pretenses. In this machine learning project, we solve the problem of detecting credit card fraud transactions using different packages Pandas, NumPy and few other python libraries.

Credit card transaction fraud costs billions of dollars to card issuers every year. A well-developed fraud detection system with a state-of-the-art fraud detection model is regarded as essential to reducing fraud losses. The main contribution of our work is the development of a fraud detection system for detecting fraud transactions using machine learning and python programming. Based on a real-life dataset from one of the largest commercial banks in US, we conduct a comparative study to assess the effectiveness of the proposed framework. Our proposed methodology is an effective and feasible mechanism for credit card fraud detection. By the implication of our proposed methodology system can efficiently identify fraudulent transactions it helps protect customers' interests and reduce fraud losses and regulatory costs.

The main purpose of supervised credit card fraud detection task is to build a machine learning model on the current transactional credit card payments data including fraudulent and non-fraudulent transactions and use it to make decision on new incoming transaction to find out if it is fraudulent or not.

2. CHAPTER 1

2.1 DATA DESCRIPTION

- The dataset contains transactions made by credit cards from January 1, 2019 to October 3, 2020. The dataset taken from the Kaggle website contains 23 rows and 1048574 columns.
- Out of 23 rows, there are 13 numerical data and 10 categorical data. Name and type of data included are given below:

  - num: Numerical
  - cat: Categorical

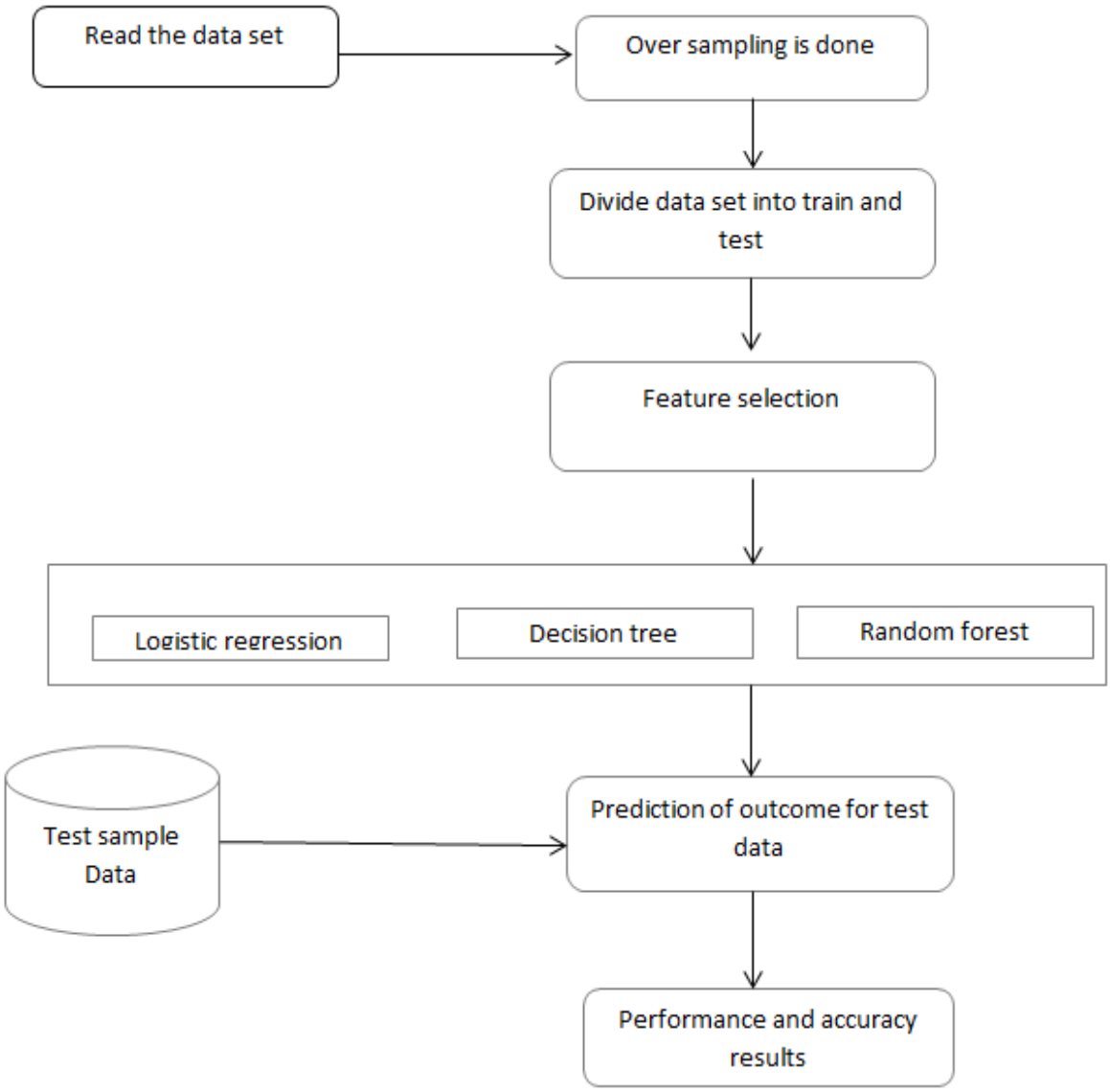
1. Index - Unique identifier for each row(num)
2. Transdate_Transtime - Transaction date and time(num)
   Example: 2020-06-21 12:14:25
3. cc_num - Credit card number of customer(num)
4. Example: 2291163933867244
5. Merchant – Merchant name(cat)
6. Category – Purpose of transaction(cat)
   Example: home, travel, personal care, entertainment
7. amt – Amount of transaction(num)
   Example: 10.37k
8. first – First name of credit card holder(cat)
9. last – Last name of credit card holder(cat)
10. gender - Gender of credit card holder(cat)
    M: Male
    F: Female
11. street – Street address of credit card holder(cat)
12. city – City of credit card holder(cat)
13. state – State of credit card holder(cat)
14. zip – Zip of credit card holder(num)
15. Lat – Latitude location of credit card holder(num)
16. long – Longitude of credit card holder(num)
17. city_pop – City population of credit card holder(num)
18. job – Job of credit card holder(cat)
19. dob – Date of birth of credit card holder(num)
20. trans_num – Transaction number
    combination of characters and numbers
    Example: 0b242abb623afc578575680df30655b9
21. unix_time – UNIX time of transaction(num)
22. merch_lat – Latitude location of merchant(num)
23. merch_long – Longitude location of merchant(num)
24. is_fraud – Checking whether it is fraud or not(num)

- Required files like NumPy, pandas, matplotlib, seaborn, datetime are imported. There are no missing values found in the dataset.
- For predicting customer frauds, columns namely trans_num, first, last, unix_time, dob, street, city, zip are of no use, so we drop it. Sometimes distance from the customer's home location to the merchant's location can prove out to be main reason for fraud, so taking the difference of longitude and latitude of respective columns. After finding displacement columns Lat, long, merch_lat, merch_long, lat_diff, long_diff is dropped.
- Segregating city population tab on the basis of less dense, adequately dense, densely populated. Recency column is divided into segments but first converting them from seconds to minutes then to number of hours passed.
- Then we initialize a separate data containing fraud transactions to analyze trends. Fraud transaction is analyzed based on state, gender, recency and checking fraud intensity respective to amount.
- We use transaction amount, transaction time, recency and displacement to predict fraud transactions.
- Data is then divided as train and test data. Detection engine uses the following ML classifiers: Decision Tree (DT), Random Forest (RF), Logistic Regression (LR) to predict the credit card fraud

2.2 METHODOLOGY

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: Feature selection are applied for the proposed models.
Step 5: Accuracy and performance metrics has been calculated to know the efficiency for different algorithms.
Step6: Then retrieve the best algorithm based on efficiency for the given dataset.



3. CHAPTER 2

3.1. ALGORITHMS USED IN THIS PROJECT

The proposed techniques are 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 gives suits best and can be adapted by credit card merchants for identifying fraud transactions.

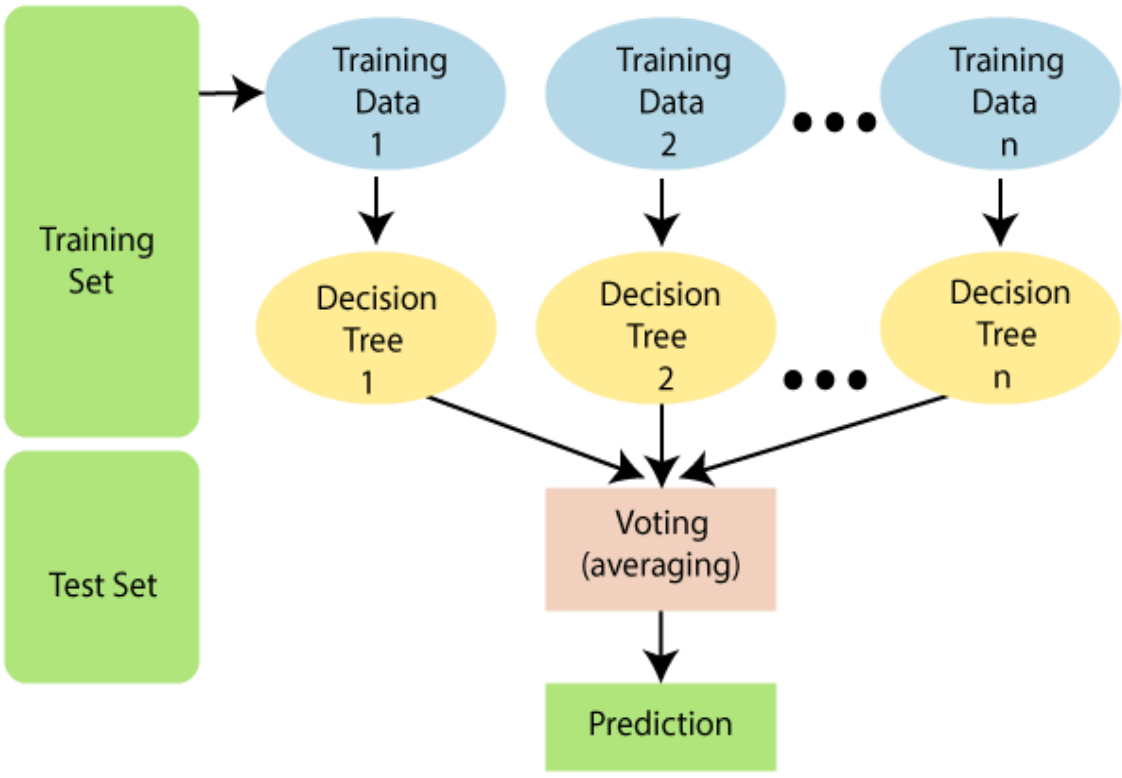
3.1.1. RANDOM FOREST:

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of over fitting.

The below diagram explains the working of the Random Forest algorithm:



Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

Advantages of Random Forest

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Code for random forest

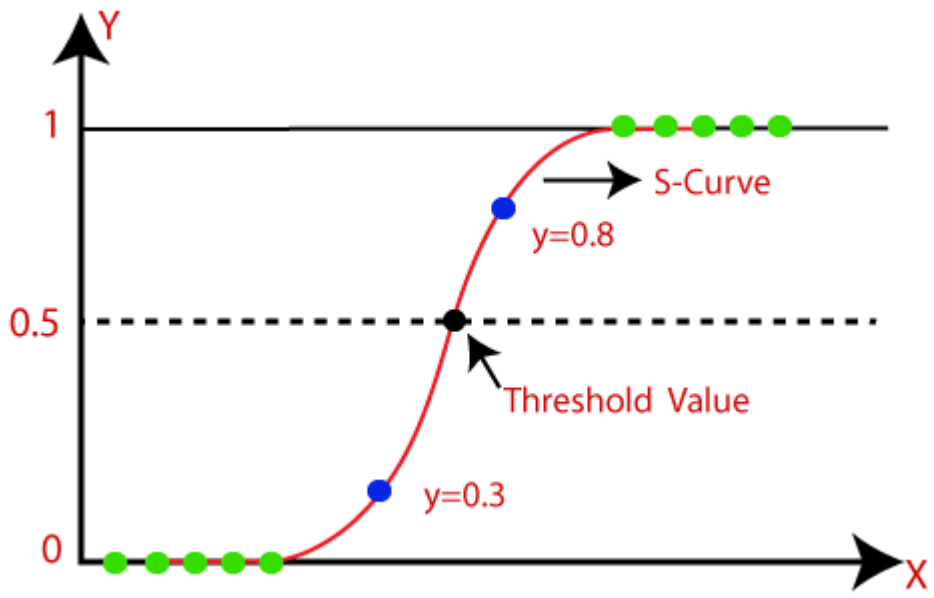
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
data = pd.read_csv('Salaries.csv')
print(data)
x= df.iloc [:, : -1]
```

```
y= df.iloc [:, -1 :]

# Fitting Random Forest Regression to the dataset
# import the regressor
from sklearn.ensemble import RandomForestRegressor

# create regressor object
regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)

# fit the regressor with x and y data
regressor.fit(x, y)
Y_pred = regressor.predict(np.array([6.5]).reshape(1, 1))
```

3.1.2. LOGISTIC REGRESSION :

Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables. Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1). Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification. The below image is showing the logistic function:



Assumptions for Logistic Regression:

The dependent variable must be categorical in nature.

The independent variable should not have multi-collinearity.

Logistic Regression Equation:

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

- We know the equation of the straight line can be written as:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

- In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$; 0 for y= 0, and infinity for y=1

- But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Advantages of Logistic Regression:

- Logistic regression is easier to implement, interpret, and very efficient to train.
- It makes no assumptions about distributions of classes in feature space.
- It can easily extend to multiple classes(multinomial regression) and a natural probabilistic view of class predictions.
- It can interpret model coefficients as indicators of feature importance.

Disadvantages of Logistic Regression:

- If the number of observations is lesser than the number of features, Logistic Regression should not be used, otherwise, it may lead to overfitting.
- It constructs linear boundaries.
- The major limitation of Logistic Regression is the assumption of linearity between the dependent variable and the independent variables.
- Non-linear problems can't be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios.

Code for logistic regression:

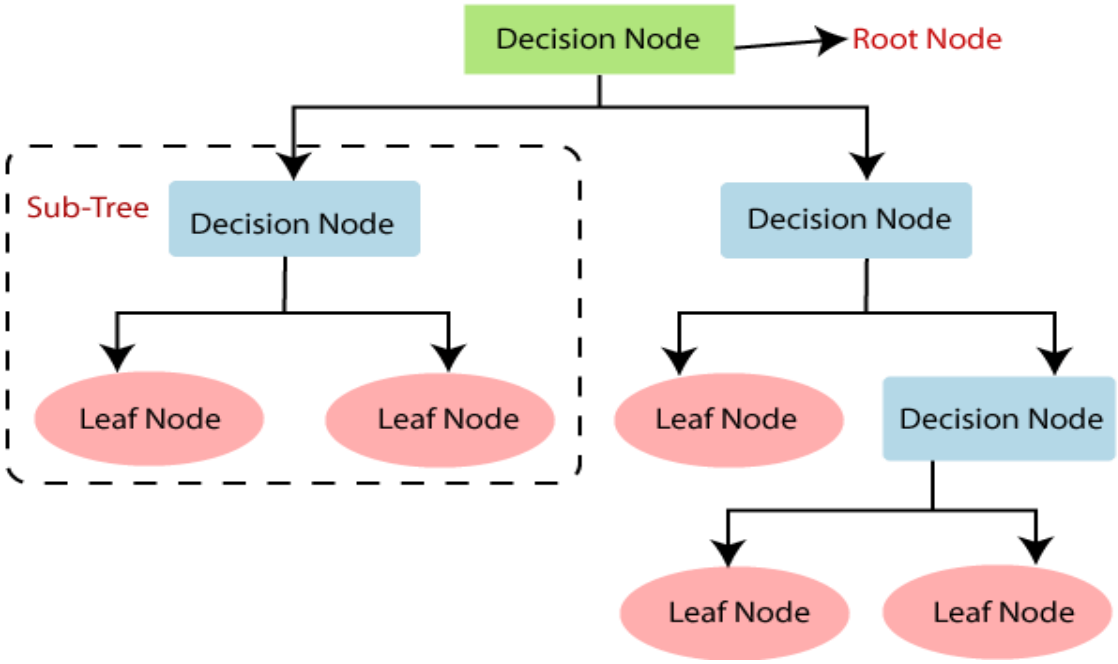
```
import matplotlib.pyplot as plt #importing libraries
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

x = np.arange(10).reshape(-1, 1)#get data
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])

>>> x #data in two dimension
array([[0],
[1],
[2],
[3],
[4],
[5],
[6],
[7],
[8],
[9]])
>>> y
array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
model = LogisticRegression(solver='liblinear', random_state=0)#create a model and train
model.fit(x, y)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100,
multi_class='warn', n_jobs=None, penalty='l2',
random_state=0, solver='liblinear', tol=0.0001, verbose=0,
warm_start=False)
model = LogisticRegression(solver='liblinear', random_state=0).fit(x, y)
>>> model.classes_
array([0, 1])
>>> model.predict_proba(x)#evaluate the model
array([[0.74002157, 0.25997843],
[0.62975524, 0.37024476],
[0.5040632 , 0.4959368 ],
[0.37785549, 0.62214451],
[0.26628093, 0.73371907],
[0.17821501, 0.82178499],
[0.11472079, 0.88527921],
[0.07186982, 0.92813018],
```

```
[0.04422513, 0.95577487],
[0.02690569, 0.97309431]])
>>> model.predict(x)
array([0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
>>> model.score(x, y)
0.9
>>> confusion_matrix(y, model.predict(x))#confusion matrix
array([[3, 1],
[0, 6]])
```

3.1.3. Decision Tree

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into sub trees.

Below diagram explains the general structure of a decision tree:



Decision Tree Terminologies:

Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.

Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

Branch/Sub Tree: A tree formed by splitting the tree.

Pruning: Pruning is the process of removing the unwanted branches from the tree.

Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

Working of decision tree:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).

Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Advantages of the Decision Tree:

It is simple to understand as it follows the same process which a human follow while making any decision in real-life.

It can be very useful for solving decision-related problems.

It helps to think about all the possible outcomes for a problem.

There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree:

The decision tree contains lots of layers, which makes it complex.

It may have an overfitting issue, which can be resolved using the Random Forest algorithm.

For more class labels, the computational complexity of the decision tree may increase.

Code for decision tree

```
    # Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

# Function importing Dataset
def importdata():
balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
'databases/balance-scale/balance-scale.data',
sep= ',', header = None)

# Printing the dataswet shape
print ("Dataset Length: ", len(balance_data))
print ("Dataset Shape: ", balance_data.shape)

# Printing the dataset obseravtions
print ("Dataset: ",balance_data.head())
return balance_data

# Function to split the dataset
def splitdataset(balance_data):

# Separating the target variable
X = balance_data.values[:, 1:5]
Y = balance_data.values[:, 0]

# Splitting the dataset into train and test
X_train, X_test, y_train, y_test = train_test_split(
X, Y, test_size = 0.3, random_state = 100)

return X, Y, X_train, X_test, y_train, y_test

# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):

# Creating the classifier object
clf_gini = DecisionTreeClassifier(criterion = "gini",
random_state = 100,max_depth=3, min_samples_leaf=5)

# Performing training
clf_gini.fit(X_train, y_train)
return clf_gini

# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):

# Decision tree with entropy
clf_entropy = DecisionTreeClassifier(
```

```
criterion = "entropy", random_state = 100,
max_depth = 3, min_samples_leaf = 5)

# Performing training
clf_entropy.fit(X_train, y_train)
return clf_entropy


# Function to make predictions
def prediction(X_test, clf_object):

# Predicton on test with giniIndex
y_pred = clf_object.predict(X_test)
print("Predicted values:")
print(y_pred)
return y_pred

# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):

print("Confusion Matrix: ",
confusion_matrix(y_test, y_pred))

print ("Accuracy : ",
accuracy_score(y_test,y_pred)*100)

print("Report : ",
classification_report(y_test, y_pred))

# Driver code
def main():

# Building Phase
data = importdata()
X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
clf_gini = train_using_gini(X_train, X_test, y_train)
clf_entropy = tarin_using_entropy(X_train, X_test, y_train)

# Operational Phase
print("Results Using Gini Index:")

# Prediction using gini
y_pred_gini = prediction(X_test, clf_gini)
cal_accuracy(y_test, y_pred_gini)

print("Results Using Entropy:")
# Prediction using entropy
y_pred_entropy = prediction(X_test, clf_entropy)
```

```
cal_accuracy(y_test, y_pred_entropy)


# Calling main function
if __name__=="__main__":
main()
```

3.1.4 Comparison of Algorithms:

Random forest VS Logistic Regression:

- LR Can't learn non linear decision boundaries and has high bias as compared to RF which has low bias and is flexible enough to learn highly nonlinear decision boundaries. The variance in RF is also reduced due to bootstrapping and voting.
- Most of the limitations of linear regression are applied to LR, that are -heteroskedasticity, serial correlation, and non-normality of error terms. All of them contribute to the standard errors of the estimated parameters.
- Can't handle missing values unlike RF which is immune to it as its underlyings are decision trees.
- In LR, features need to be scaled and normalized unlike RF which is unaffected by it.
- Appropriate features must be selected before fitting the model unlike RF which select features in its decision tress. Or as an alternative the Logistic Regression model should be regularized with lasso to select the features.
- When classes are completely separable, the estimation of parameters becomes unstable in LR due to the use of logistic function which then becomes close to a Step Function and it forces the derivatives to be infinite and hence becomes computationally unstable. So, LR works when the classes are almost linearly separable but not exactly. In RF there is no problem if the classes are completely separable. Rather it helps to reduce the computations when appropriate tree pruning methods are used.

Logistic Regression	Random Forest
Path analysis approach, uses a generalized linear equation to describe the directed dependencies among a set of variables.	Top-down induction based approach to classification and prediction. Averages many decision trees (CARTs) together.
A number of statistical assumptions must be met.	No statistical assumptions; can handle multicollinearity.
Overfitting a concern (rule of ten), as well as outliers.	Robust to overfitting and outliers.
Final model should be parsimonious and balanced.	Final model depends on the strength of the trees in the forest and the correlation between them.
A number of complementary measures can be used to assess goodness of fit (i.e., -2LL, ~R^2, HL).	Random inputs and random features tend to produce better results in RFs (Breiman, 2001).
Logit link function: $\ln\left(\frac{\hat{p}_i}{1-\hat{p}_i}\right) = \beta_1 X_i + \beta_0$	CART Gini impurity algorithm: $\sum_{i=1}^{J} p_i(1-p_i) = \sum_{i=1}^{J}(p_i - p_i^2) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} p_i^2 = 1 - \sum_{i=1}^{J} p_i^2$

Random forest vs Decision tree

- Decision tree algorithm are prone to errors like overfitting, error due to bias and variance. on other hand random forest Builds a robust model.It does not suffer from overfitting problem.Can use for both classification and regression problems.
- The predictions made by decision tree algorithms are less accurate compared to random forest . random forest gives highly accurate predictions .It is Powerful than other non-linear models.

Random Forest	Decision Tree
It is a group of decision trees combined together to give output.	It is a tree-like decision-making diagram.
Prevents Overfitting.	Possibility of Overfitting.
Gives accurate results.	Gives less accurate result.
Hard to interpret.	Simple and easy to interpret.
More Computation	Less Computation
Complex Visualization.	Simple to visualize.
Slow to process.	Fast to process.

3.1.5 PROGRAM CODE

Import libraries

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
```

Import the dataset

```
df = pd.read_csv('/content/drive/MyDrive/CREDICT CARD FRAUD DETECTIONS .csv')
df.head()
df.shape()
```

Getting more details of the dataset

```
df.info()
#looks like there are no missing values found in the dataset.
df.isna().sum()
#Checking the highly imbalanced target variable.
df['is_fraud'].value_counts()
import seaborn as sns
sns.set_theme(style="darkgrid")
ax = sns.countplot(x="is_fraud", data=df)
genuine_percentage = df['is_fraud'].value_counts()[0]/(len(df))
fraud_percentage = df['is_fraud'].value_counts()[1]/(len(df))
d = {'Genuine': genuine_percentage, 'Fraud': fraud_percentage}
percentages = pd.DataFrame(data = d, index=[0])
percentages
sns.barplot(data = percentages)
df.head()
# we cannot work on trans_num as there is no unique pattern, so dropping it
df = df.drop("trans_num",1)
# checking cc_num columns
df.cc_num.value_counts()
# checking first and last name columns
df['first'].value_counts()
df["last"].value_counts()
# we can see the first and last names of customers are not unique while the cc_num is, so we will use cc_num to distinct between customers
# and since first and last are now of no use of ours so we will drop them
df = df.drop(columns=["first","last"])
# we can have look on unix time, unix time is generally the number of seconds passed from the UNIX EPOCH i.e. 00:00:00 UTC on 1 January 1970
# we can use this to know the recency of transactions of same cc_num
df["recency"] = df.groupby(by="cc_num")["unix_time"].diff()
```

```
# checking null values of recency
df["recency"].isnull().sum()
# we are getting null values because as 983 because there are 983 unique values of cards, this means whenever the cc_num group changes
# python makes the first value of every group null, so making them as starting payment, we will initialize null values to -1
df.loc[df.recency.isnull(),["recency"]] = -1
#checking null values again
df.isnull().sum()
# converting trans_date_trans_time to datetime
df["trans_date_trans_time"] = pd.to_datetime(df["trans_date_trans_time"])
df= df.drop("unix_time",1)
# we even don't need Unnamed: 0 as it is the index only and we have index present with us
df = df.drop(columns=["Unnamed: 0"])
df.dob.value_counts()
# in predicting customer frauds, dob doesn't play any role that whether there will be a fraud with customer or not, so removing dob column too
df = df.drop("dob",1)
# sometimes distance from the customer's home location to the merchant's location can prove out to be main reason for fraud, so taking the
# difference of longitude and lattitude of respective columns
df["lat_diff"] = abs(df.lat - df.merch_lat)
df["long_diff"] = abs(df["long"] - df["merch_long"])
# we have used abs function so that we get proper distance diiference in positive as abs makes negative values positive and used as a mod function
df.head()
# now since we have the difference, it is estimated that difference between each degree of longitude and lattitude is 69 miles(approx)
# or 110 kilometers (approx), so taking displacement into account as it will be difficult to calculate distance between merchant's location
# or customer's location so applying pythogoras theorem
df["displacement"] = np.sqrt(pow((df["lat_diff"]*110),2) + pow((df["long_diff"]*110),2))
# here we have applied pythogoras theorem and we have multiplied with 110 because each degree of longitude and lattitude is 69 miles(approx)
# or 110 kilometers apart
# now since we got the displacement so longitudes and lattitudes columns are of no use now, so we can remove them
df = df.drop(columns = ["lat","long","merch_lat","merch_long","lat_diff","long_diff"])
# since state contains both city and zip code and street comes under city, so we can move with state column and drop street, city and zip
# we can work with cities through their population parameter, as names of cities cannot implement whether a fraud will be done or not, while
# population of a city can.
df= df.drop(columns = ["city","zip","street"])
df.info()
# checking displacement column
df.displacement.describe()
# now we can bin the displacement into near, far and very far records
```

```
# if merchant lies between the range of 0-45 then it is near, while above 45 but below 90 will be far and rest can be very far
df.loc[(df["displacement"]<45),["location"]] = "Nearby"
df.loc[((df["displacement"]>45) & (df["displacement"]<90)),["location"]] = "Far Away"
df.loc[(df["displacement"]>90),["location"]] = "Long Distance"
df.info()
# checking location column
df.location.value_counts(normalize=True)
# Although date part in column trans_date_trans_time is not important because that is historical data, but time part of the component is important
# so creating a column of time
df["Time"] = pd.to_datetime(df["trans_date_trans_time"],"%H:%M").dt.time
# converting Time column to datetime
df["Time"] = pd.to_datetime(df["trans_date_trans_time"]).dt.hour
# segregating city_population tab on the basis of less dense, adequately densed, densely populated
df.loc[(df["city_pop"]<10000),["city_pop_segment"]] = "Less Dense"
df.loc[((df["city_pop"]>10000) & (df["city_pop"]<50000)),["city_pop_segment"]]= "Adequately Dense"
df.loc[(df["city_pop"]>50000),["city_pop_segment"]] = "Densely populated"
df.city_pop_segment.value_counts(normalize = True)
df = df.drop("city_pop",1)
#dividing recency column into segments but first converting them from seconds to minutes
df.recency = df.recency.apply(lambda x: float((x/60)/60))
#dividing recency to segments based on number of hours passed
df.loc[(df["recency"]<1),["recency_segment"]] = "Recent_Transaction"
df.loc[((df["recency"]>1) & (df["recency"]<6)),["recency_segment"]] = "Within 6 hours"
df.loc[((df["recency"]>6) & (df["recency"]<12)),["recency_segment"]] = "After 6 hours"
df.loc[((df["recency"]>12) & (df["recency"]<24)),["recency_segment"]] = "After Half-Day"
df.loc[(df["recency"]>24),["recency_segment"]] = "After 24 hours"
df.loc[(df["recency"]<0),["recency_segment"]] = "First Transaction"
df.recency_segment.value_counts(normalize = True)
# examining category column
df.category.value_counts()
df.columns
df.head()
# let's initialize a separate data containing fraud transactionsto analyze trends
data_fraud = df[df["is_fraud"]==1]
data_fraud.head()
# checking fraud transactions peak hours
sns.kdeplot(data_fraud["Time"])
plt.xlim(left = 0,right = 24)
plt.show()
# lets have a look about the fraud transactions done in cities according to their populations
sns.countplot(data_fraud["city_pop_segment"])
plt.show()
# let's have a look on state-wise fraud transactions
plt.figure(figsize = [8,10])
sns.countplot(y=data_fraud.state)
```

```
plt.show()
# let's analyze the the date when maximum frauds have happened
sns.kdeplot(data_fraud.trans_date_trans_time)
plt.xticks(rotation = 90)
plt.show()
# checking fraud transactions based on recency
sns.countplot(y=data_fraud.recency_segment)
plt.show()
sns.kdeplot(data_fraud[data_fraud["recency_segment"] == "Recent_Transaction"].Time)
plt.xlim(left=0,right=24)
plt.show()
# Checking which category have highest number of frauds
sns.countplot(y=data_fraud.category)
plt.show()
# let's check how far frauds happens from customer's residence
sns.countplot(data_fraud.location)
plt.show()
# let's have a look on fraud transactions based on Gender
sns.countplot(data_fraud.gender)
plt.show()
# Let's checkwhat is the fraud intensity respective to amount
sns.kdeplot(data_fraud.amt)
plt.xlim(left=0)
plt.show()
# let's visualize this cycle with respect to gender
sns.kdeplot(data_fraud.amt,hue=df.gender)
plt.xlim(left=0)
plt.show()
data=df.drop(columns=["trans_date_trans_time","cc_num","merchant","job","state","category","g
ender","recency_segment","city_pop_segment","location"])
data.head()
d = data[data["is_fraud"]==1]
d.head()
data.info()
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
data[["amt","Time","displacement","recency"]]=sc.fit_transform(data[["amt","Time","displaceme
nt","recency"]])
d = data[data["is_fraud"]==1]
d.head()
a=data.drop(columns=["is_fraud"])
a.head()
```

Convert the data into data frames format

```
x = a.iloc[:].values
```

```
y = data.loc[:,["is_fraud"]].values
print(x)
print(y)
```

Decide the amount of data for training data and testing data

```
from sklearn.model_selection import train_test_split
x_train,x_test , y_train ,y_test = train_test_split(x,y,test_size=0.4 , random_state=0)
```

Logistic regression model

```
from sklearn.linear_model import LogisticRegression
log_clf = LogisticRegression(random_state=0).fit(x_train,np.ravel (y_train))
```

Accuracy of Logistic Regression

```
from sklearn.metrics import accuracy_score
train_y_pred = log_clf.predict(x_train)
test_y_pred = log_clf.predict(x_test)
print(accuracy_score(y_test, test_y_pred))
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, test_y_pred)
```

Decision tree model

```
from sklearn.tree import DecisionTreeClassifier
tree_clf = DecisionTreeClassifier().fit(x_train, y_train)
```

Accuracy of decision tree

```
#Test Accuracy
test_y_pred = tree_clf.predict(x_test)
print(accuracy_score(y_test, test_y_pred))
#Confusion Matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, test_y_pred)
```

Random forest model

```
from sklearn.ensemble import RandomForestClassifier
rf_clf = RandomForestClassifier().fit(x_train,np.ravel(y_train))
```

Accuracy of random forest

```
#Test Accuracy
test_y_pred = rf_clf.predict(x_test)
print(accuracy_score(y_test, test_y_pred))
```

```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, test_y_pred)
models = []
```

Plotting accuracy of logistic regression ,Decision tree and random forest

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
models.append(('LR', LogisticRegression()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('RF', RandomForestClassifier()))
names = []
scores = []
for name, model in models:
model.fit(x_train, np.ravel(y_train))
y_pred = model.predict(x_test)
scores.append(accuracy_score(y_test, y_pred))
names.append(name)
tr_split = pd.DataFrame({'Name': names, 'Score': scores})
print(tr_split)
import seaborn as sns
axis = sns.barplot(x = 'Name', y = 'Score', data =tr_split )
axis.set(xlabel='Classifier', ylabel='Accuracy')
for p in axis.patches:
height = p.get_height()
axis.text(p.get_x() + p.get_width()/2, height + 0.005, '{:1.4f}'.format(height), ha="center")
plt.show()
```

Prediction model using Random forest

```
data2=[[0.936726,-0.67569,0.738772,-1.877811]]
data3=[[1.388081,      -0.476154       ,1.239504,-1.437726]]
predict_fraud=rf_clf.predict(data3)
print(predict_fraud)
if(predict_fraud==1):
print("fraud")
else:
print("genuine")
```

4. CHAPTER

4.1 RESULTS OF DATA ANALYSIS AND INTERPRETATION

Step 1: Import required libraries and data set

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
```

```
[ ] df = pd.read_csv('/content/drive/MyDrive/CREDICT CARD FRAUD DETECTIONS .csv')
    df.head()
```

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...	lat	long	city_pop	
0	0	01-01-2019 00:00	2.703190e+15	fraud_Rippin, Kub and Mann	misc_net	4.97	Jennifer	Banks	F	561 Perry Cove	...	36.0788	-81.1781	3495	Psycholog counsell
1	1	01-01-2019 00:00	6.304230e+11	fraud_Heller, Gutmann and Zieme	grocery_pos	107.23	Stephanie	Gill	F	43039 Riley Greens Suite 393	...	48.8878	-118.2105	149	Spe educatio needs teac
										594					

Step 2: Getting more details about the data set.

```
df.shape
```

```
(1048575, 23)
```

+ Code + Text

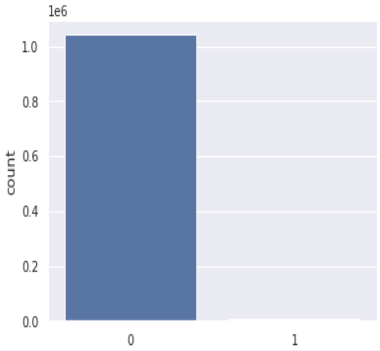
```
[ ] #Getting more details of the dataset.
    df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 23 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   Unnamed: 0             1048575 non-null  int64
 1   trans_date_trans_time  1048575 non-null  object
 2   cc_num                 1048575 non-null  float64
 3   merchant               1048575 non-null  object
 4   category               1048575 non-null  object
 5   amt                    1048575 non-null  float64
 6   first                  1048575 non-null  object
 7   last                   1048575 non-null  object
 8   gender                 1048575 non-null  object
 9   street                 1048575 non-null  object
 10  city                   1048575 non-null  object
 11  state                  1048575 non-null  object
 12  zip                    1048575 non-null  int64
 13  lat                    1048575 non-null  float64
```

```
#Checking the highly imbalanced target variable.
df['is_fraud'].value_counts()
```

```
0    1042569
1       6006
Name: is_fraud, dtype: int64
```

```
import seaborn as sns
sns.set_theme(style="darkgrid")
ax = sns.countplot(x="is_fraud", data=df)
```

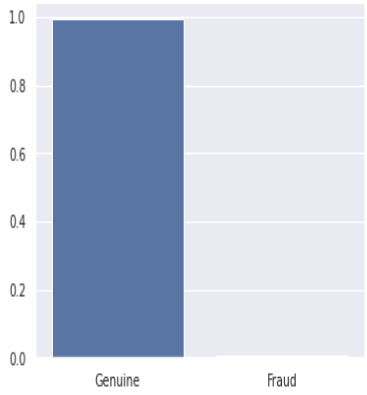



```
genuine_percentage = df['is_fraud'].value_counts()[0]/(len(df))
fraud_percentage = df['is_fraud'].value_counts()[1]/(len(df))
d = {'Genuine': genuine_percentage, 'Fraud': fraud_percentage}
percentages = pd.DataFrame(data = d, index=[0])
percentages
```

	Genuine	Fraud
0	0.994272	0.005728

```
sns.barplot(data = percentages)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc359d29550>
```



Find new column displacement using latitude and longitude

```
[ ]  #  now since we have the difference, it is estimated that difference between each degree of longitude and lattitude is 69 miles(approx)
     # or 110 kilometers (approx), so taking displacement into account as it will be difficult to calculate distance between merchant's location
     # or customer's location so applying pythogoras theorem

     df["displacement"] = np.sqrt(pow((df["lat_diff"]*110),2) + pow((df["long_diff"]*110),2))

     # here we have applied pythogoras theorem and we have multiplied with 110 because each degree of longitude and lattitude is 69 miles(approx)
     # or 110 kilometers apart
```

```
[ ] # now since we got the displacement so longitudes and lattitudes columns are of no use now, so we can remove them
    df = df.drop(columns = ["lat","long","merch_lat","merch_long","lat_diff","long_diff"])
```

```
    # since state contains both city and zip code and street comes under city, so we can move with state column and drop street, city and zip
    # we can work with cities through their population parameter, as names of cities cannot implement whether a fraud will be done or not, while
    # population of a city can.
    df= df.drop(columns = ["city","zip","street"])
    df.info()
```

Categories the displacement into near ,far and very far

```
[ ] # now we can bin the displacement into near, far and very far records
    # if merchant lies between the range of 0-45 then it is near, while above 45 but below 90 will be far and rest can be very far

    df.loc[(df["displacement"]<45),["location"]] = "Nearby"
    df.loc[((df["displacement"]>45) & (df["displacement"]<90)),["location"]] = "Far Away"
    df.loc[(df["displacement"]>90),["location"]] = "Long Distance"
    df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 13 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   trans_date_trans_time  1048575 non-null  datetime64[ns]
 1   cc_num                 1048575 non-null  float64
 2   merchant               1048575 non-null  object
 3   category               1048575 non-null  object
 4   amt                    1048575 non-null  float64
 5   gender                 1048575 non-null  object
 6   state                  1048575 non-null  object
 7   city_pop               1048575 non-null  int64
 8   job                    1048575 non-null  object
 9   is_fraud               1048575 non-null  int64
 10  recency                1048575 non-null  float64
 11  displacement           1048575 non-null  float64
 12  location               1048575 non-null  object
dtypes: datetime64[ns](1), float64(4), int64(2), object(6)
memory usage: 104.0+ MB
```

Converting time column to datetime

Classifying city_population column on the basis of less dense ,adequately densed and densely populated

```
[ ] # Although date part in column trans_date_trans_time is not important because that is historical data, but time part of the component is important
    # so creating a column of time

    df["Time"] = pd.to_datetime(df["trans_date_trans_time"],"%H:%M").dt.time
```

```
[ ] # converting Time column to datetime
    df["Time"] = pd.to_datetime(df["trans_date_trans_time"]).dt.hour
    # segregating city_population tab on the basis of less dense, adequately densed, densely populated
    df.loc[(df["city_pop"]<10000),["city_pop_segment"]] = "Less Dense"
    df.loc[((df["city_pop"]>10000) & (df["city_pop"]<50000)),["city_pop_segment"]] = "Adequately Dense"
    df.loc[(df["city_pop"]>50000),["city_pop_segment"]] = "Densely populated"
```

```
[ ] df.city_pop_segment.value_counts(normalize = True)
```

```
Less Dense          0.699429
Densely populated   0.187494
Adequately Dense    0.113076
Name: city_pop_segment, dtype: float64
```

Dividing recency to segments based on number of hour passed.

```
[ ]  #dividing recency to segments based on number of hours passed
    df.loc[(df["recency"]<1),["recency_segment"]] = "Recent_Transaction"
    df.loc[((df["recency"]>1) & (df["recency"]<6)),["recency_segment"]] = "Within 6 hours"
    df.loc[((df["recency"]>6) & (df["recency"]<12)),["recency_segment"]] = "After 6 hours"
    df.loc[((df["recency"]>12) & (df["recency"]<24)),["recency_segment"]] = "After Half-Day"
    df.loc[(df["recency"]>24),["recency_segment"]] = "After 24 hours"
    df.loc[(df["recency"]<0),["recency_segment"]] = "First Transaction"
    df.recency_segment.value_counts(normalize = True)
```

```
Within 6 hours      0.415753
After 6 hours       0.188987
Recent_Transaction  0.169538
After Half-Day      0.144539
After 24 hours      0.080284
First Transaction   0.000899
Name: recency_segment, dtype: float64
```

Analyzing trends of fraud transactions.

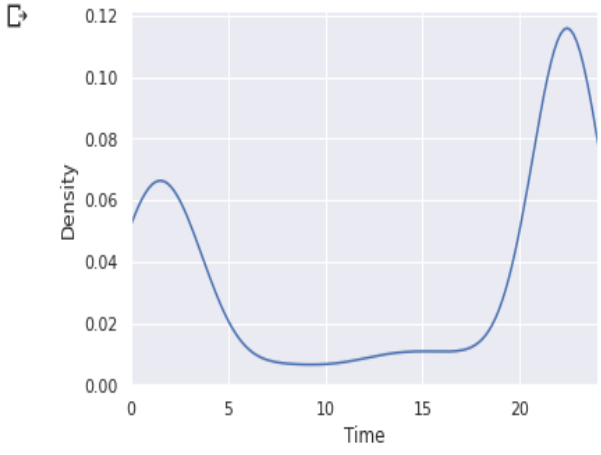
```
[ ] # let's initialize a separate data containing fraud transactionsto analyze trends
    data_fraud = df[df["is_fraud"]==1]
```

```
data_fraud.head()
```

	trans_date_trans_time	cc_num	merchant	category	amt	gender	state	job	is_fraud	recency	displacement	location	Time	city_p
2449	2019-02-01 01:06:00	4.613310e+12	fraud_Rutherford-Mertz	grocery_pos	281.06	M	NC	Soil scientist	1	12.249167	76.922817	Far Away	1	
2472	2019-02-01 01:47:00	3.401870e+14	fraud_Jenkins, Hauck and Friesen	gas_transport	11.52	F	TX	Horticultural consultant	1	-0.000278	86.017410	Far Away	1	Dense
2523	2019-02-01 03:05:00	3.401870e+14	fraud_Goodwin-Nitzsche	grocery_pos	276.31	F	TX	Horticultural consultant	1	1.298333	45.388989	Far Away	3	Dense
2546	2019-02-01 03:38:00	4.613310e+12	fraud_Erdman-Kertzmann	gas_transport	7.03	M	NC	Soil scientist	1	2.523889	41.168829	Nearby	3	
2553	2019-02-01 03:55:00	3.401870e+14	fraud_Koepp-Parker	grocery_pos	275.73	F	TX	Horticultural consultant	1	0.840000	45.403601	Far Away	3	Dense

Here checking fraud transaction peak hours

```
# checking fraud transactions peak hours

sns.kdeplot(data_fraud["Time"])
plt.xlim(left = 0,right = 24)
plt.show()
```

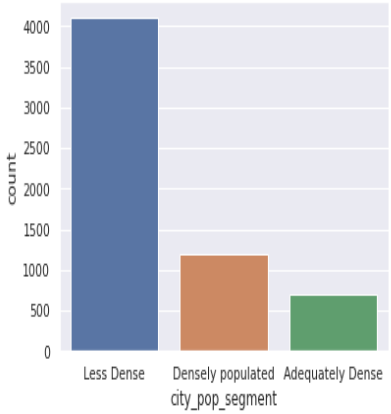


We can have a look that peak hours for fraud transactions starts from 07:00 P.M. to 5:00 A.M. which means transactions done in night needs to be scrutinised more and to be checked with the customer

Fraud transactions done in cities according to their population

```
[ ] # lets have a look about the fraud transactions done in cities according to their populations
    sns.countplot(data_fraud["city_pop_segment"])
    plt.show()
```

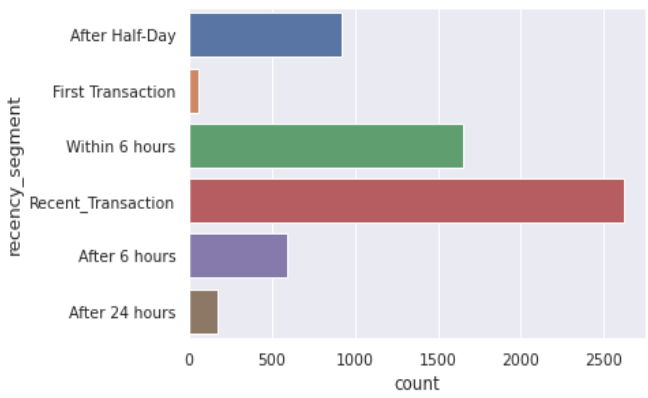
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only vali
  FutureWarning
```



Its seen that less dense cities are more prone to frauds as compared to more densed cities while on the other hand, adequately densed cities are less prone to frauds as compared to densely populated cities

Fraud transaction based on recency

```
# checking fraud transactions based on recency
sns.countplot(y=data_fraud.recency_segment)
plt.show()
```

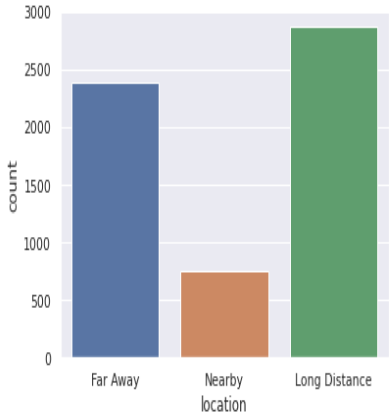


It can be observed that recent transactions are less in number overall while they are termed as more fraud, while other parameters like After Half-Day or Within 6 hours termed as fraud because probably customer have done a genuine transaction in the day, while the fraud transaction which was done through his/her card might be during the night time, as was seen earlier that peak hours for fraud transactions are in the night.

Fraud transaction according to displacement

```
# let's check how far frauds happens from customer's residence
sns.countplot(data_fraud.location)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only vali
  FutureWarning
```



It can be seen that if the merchant's location is far away from customer's location, then the probability of transaction being a fraud increases

After analyzing all these trends we choose more desired attributes for detecting fraud transactions.

```
[ ] data = df.drop(columns=["trans_date_trans_time","cc_num","merchant","job","state","category","gender","recency_segment","city_pop_segment","location"])
```

```
[ ] data.head()
```

	amt	is_fraud	recency	displacement	Time
0	4.97	0	-0.000278	96.011245	0
1	107.23	0	-0.000278	29.954105	0
2	220.11	0	-0.000278	107.342986	0
3	45.00	0	-0.000278	101.178169	0
4	41.96	0	-0.000278	95.535506	0

Amt ,is_fraud ,recency ,displacement and time are the columns choosen.

STEP 3: Converting data into data frame format

```
[ ] x = a.iloc[:].values

    y = data.loc[:,["is_fraud"]].values
```

```
[ ] print(x)
    print(y)
```

```
[[-0.40830493 -0.67568975  0.37693403 -1.87781142]
 [ 0.2310128  -0.67568975 -1.73236046 -1.87781142]
 [ 0.93672555 -0.67568975  0.73877195 -1.87781142]
 ...
 [-0.30639921 -0.21507863  2.04717703  0.46931103]
 [-0.37985886 -0.46480165  1.89882117  0.46931103]
 [-0.39680147 -0.50700056 -0.48028019  0.46931103]]
[[0]
 [0]
 [0]
 ...
 [0]
 [0]
 [0]]
```

STEP 4: Decide the amount of data for training data and testing data

```
[ ] from sklearn.model_selection import train_test_split
    x_train,x_test , y_train ,y_test = train_test_split(x,y,test_size=0.4 , random_state=0)
```

We choose 60% data for training purpose and remaining for testing purpose.

STEP 5: Logistic regression model

```
[111] from sklearn.model_selection import train_test_split
      x_train,x_test , y_train ,y_test = train_test_split(x,y,test_size=0.4 , random_state=0)

[112] from sklearn.linear_model import LogisticRegression
      log_clf = LogisticRegression(random_state=0).fit(x_train,np.ravel (y_train))

[113] from sklearn.metrics import accuracy_score
      train_y_pred = log_clf.predict(x_train)

[114] test_y_pred = log_clf.predict(x_test)

      print(accuracy_score(y_test, test_y_pred))

      0.9936056076103283
```

Step 1: START
Step 2: Reading the dataset. pd.read.csv (file name) # reads the dataset file
Step 3: Data cleaning and preprocessing of data
•Resampling the data as normal and fraud class i.e. normal = 0 and fraud =1 under sampling of data is done
•Data is scaled (if any null value then eliminated) and normalized.
•Dataset is splitted into two set as train data and test data using split () on training data is used to split the data.
Step 4: Training the data using the LOGISTIC REGRESSION algorithm
•Logistic regression is called as log_clf() # which predicts whether transaction fraud or nonfraud using given data.
Step 5: Calculating the fraud transactions and valid transactions, then calculating the accuracy and stored in the respective locations

Step 6: STOP

Here we get 0.994 % accuracy.

STEP 6: Decision Tree model

```
[118] from sklearn.tree import DecisionTreeClassifier

      tree_clf = DecisionTreeClassifier().fit(x_train, y_train)
```

```
[119] #Test Accuracy
      test_y_pred = tree_clf.predict(x_test)

      print(accuracy_score(y_test, test_y_pred))

      0.9939703883842358
```

Step 1: START
Step 2: Reading the dataset. pd.read.csv (file name) # reads the dataset file
Step 3: Data cleaning and preprocessing of data
•Resampling the data as normal and fraud class i.e. normal = 0 and fraud =1 under sampling of data is done
•Data is scaled (if any null value then eliminated) and normalized.
•Dataset is splitted into two set as train data and test data using split () on training data is used to split the data.
Step 4: Training the data using the DECISION TREE algorithm
•Decision tree is called as tree_clf() # which predicts whether transaction fraud or nonfraud using given data.
Step 5: Calculating the fraud transactions and valid transactions, then calculating the accuracy and stored in the respective locations

Step 6: STOP

Here we get 0.9945% accuracy.

Random Forest model

```
[123] from sklearn.ensemble import RandomForestClassifier
      rf_clf = RandomForestClassifier().fit(x_train,np.ravel(y_train))
```

```
[124] #Test Accuracy
      test_y_pred = rf_clf.predict(x_test)

      print(accuracy_score(y_test, test_y_pred))

      0.9961614572157452
```

Step 1: START
Step 2: Reading the dataset. pd.read.csv (file name) # reads the dataset file
Step 3: Data cleaning and preprocessing of data
•Resampling the data as normal and fraud class i.e. normal = 0 and fraud =1 under sampling of data is done
•Data is scaled (if any null value then eliminated) and normalized.
•Dataset is splitted into two set as train data and test data using split () on training data is used to split the data.
Step 4: Training the data using the RANDOM FOREST algorithm.
•Random forest is called as rf_clf() # which predicts whether transaction fraud or nonfraud using given data.
Step 5: Calculating the fraud transactions and valid transactions, then calculating the accuracy and stored in the respective locations

Step 6: STOP

Here we get 0.996% accuracy

The basic performance measures derived from the confusion matrix. The confusion matrix is a 2 by 2 matrix table contains four outcomes produced by the binary classifier. Various measures such as sensitivity, specificity, accuracy and error rate are derived from the confusion matrix.

Confusion matrix

Accuracy: Accuracy is calculated as the total number of two correct predictions(A+B) divided by the total number of the dataset(C+D).It is calculated as (1-error rate).

Accuracy=A+B/C+D
Whereas,
A=True Positive
B=True Negative
C=Positive
D=Negative
Error rate: Error rate is calculated as the total number of two incorrect predictions(F+E) divided by the total number of the dataset(C+D).

Error rate=F+E/C+D
Whereas,
E=False Positive F=False Negative
C=Positive D=Negative
Sensitivity: Sensitivity is calculated as the number of correct positive predictions(A) divided by the total number of positives(C).

Sensitivity=A/C
Specificity: Specificity is calculated as the number of correct negative predictions(B) divided by the total number of negatives(D).

Specificity=B/D.

Accuracy, Error-rate, Sensitivity and Specificity are used to report the performance of the system to detect the fraud in the credit card.
In this paper, three machine learning algorithms are developed to detect the fraud in credit card system. To evaluate the algorithms, 60% of the dataset is used for training and 40% is used for testing and validation. The accuracy result is shown for logistic regression; Decision tree and random forest classifier are 99.3, 99.4, and 99.6 respectively. The comparative results show that the Random forest performs better than the logistic regression and decision tree techniques.

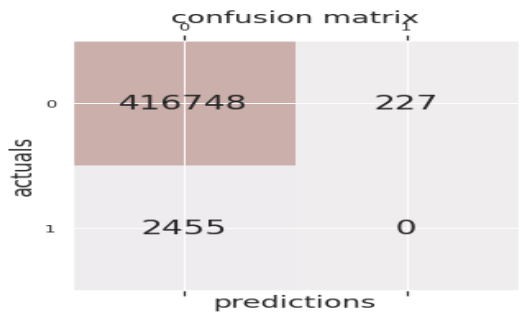
CONFUSION MATRIX AND INTERPRETATION

LOGISTIC REGRESSION:

```
[115] from sklearn.metrics import confusion_matrix

      c=confusion_matrix(y_test, test_y_pred)
      print(c)
```

```
[[416748    227]
 [  2455      0]]
```

```
#print confusion matrix using matplotlib
fig,ax=plt.subplots(figsize=(5,5))
ax.matshow(c,cmap=plt.cm.Oranges,alpha=0.3)
for i in range(c.shape[0]):
  for j in range(c.shape[1]):
     ax.text(x=j,y=i,s=c[i,j],va='center',ha='center',size='xx-large')
plt.xlabel('predictions',fontsize=18)
plt.ylabel('actuals',fontsize=18)
plt.title('confusion matrix',fontsize=18)
plt.show()
```




```
[117] from sklearn.metrics import precision_score,recall_score
      print('precision :%.3f'%precision_score(y_test, test_y_pred))
      print('recall :%.3f'%recall_score (y_test, test_y_pred))
      print('accuracy :%.3f'%accuracy_score (y_test, test_y_pred))
```

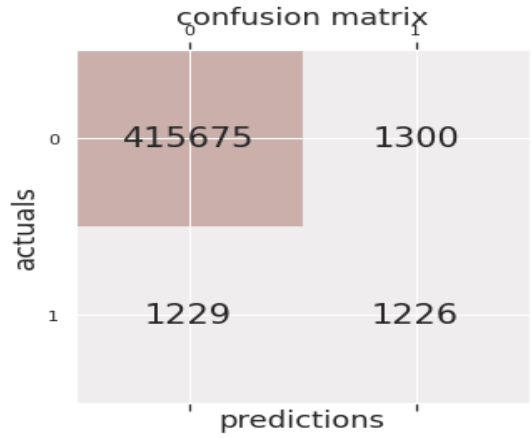
```
precision :0.000
recall :0.000
accuracy :0.994
```

DECISION TREE:

```
[120] #Confusion Matrix
      from sklearn.metrics import confusion_matrix

      c=confusion_matrix(y_test, test_y_pred)
      print(c)
```

```
[[415675   1300]
 [  1229   1226]]
```

```
[121] #print confusion matrix using matplotlib
      fig,ax=plt.subplots(figsize=(5,5))
      ax.matshow(c,cmap=plt.cm.Oranges,alpha=0.3)
      for i in range(c.shape[0]):
        for j in range(c.shape[1]):
          ax.text(x=j,y=i,s=c[i,j],va='center',ha='center',size='xx-large')
      plt.xlabel('predictions',fontsize=18)
      plt.ylabel('actuals',fontsize=18)
      plt.title('confusion matrix',fontsize=18)
      plt.show()
```




```
[122] from sklearn.metrics import precision_score,recall_score
      print('precision :%.3f'%precision_score(y_test, test_y_pred))
      print('recall :%.3f'%recall_score (y_test, test_y_pred))
      print('accuracy :%.3f'%accuracy_score (y_test, test_y_pred))
```

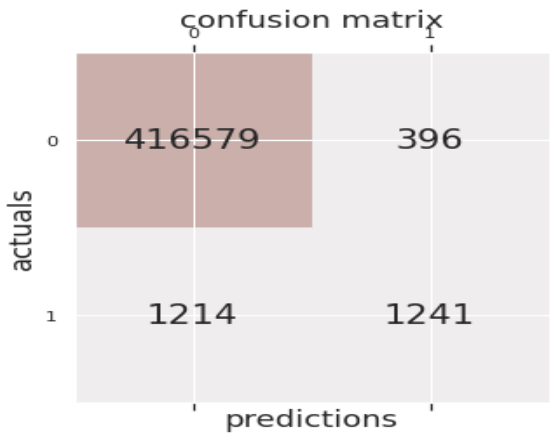
```
precision :0.485
recall :0.499
accuracy :0.994
```

RANDOM FOREST:

```
[125] #Confusion Matrix
      from sklearn.metrics import confusion_matrix

      c=confusion_matrix(y_test, test_y_pred)
      print(c)
```

```
[[416579    396]
 [  1214   1241]]
```

```
#print confusion matrix using matplotlib
fig,ax=plt.subplots(figsize=(5,5))
ax.matshow(c,cmap=plt.cm.Oranges,alpha=0.3)
for i in range(c.shape[0]):
  for j in range(c.shape[1]):
    ax.text(x=j,y=i,s=c[i,j],va='center',ha='center',size='xx-large')
plt.xlabel('predictions',fontsize=18)
plt.ylabel('actuals',fontsize=18)
plt.title('confusion matrix',fontsize=18)
plt.show()
```



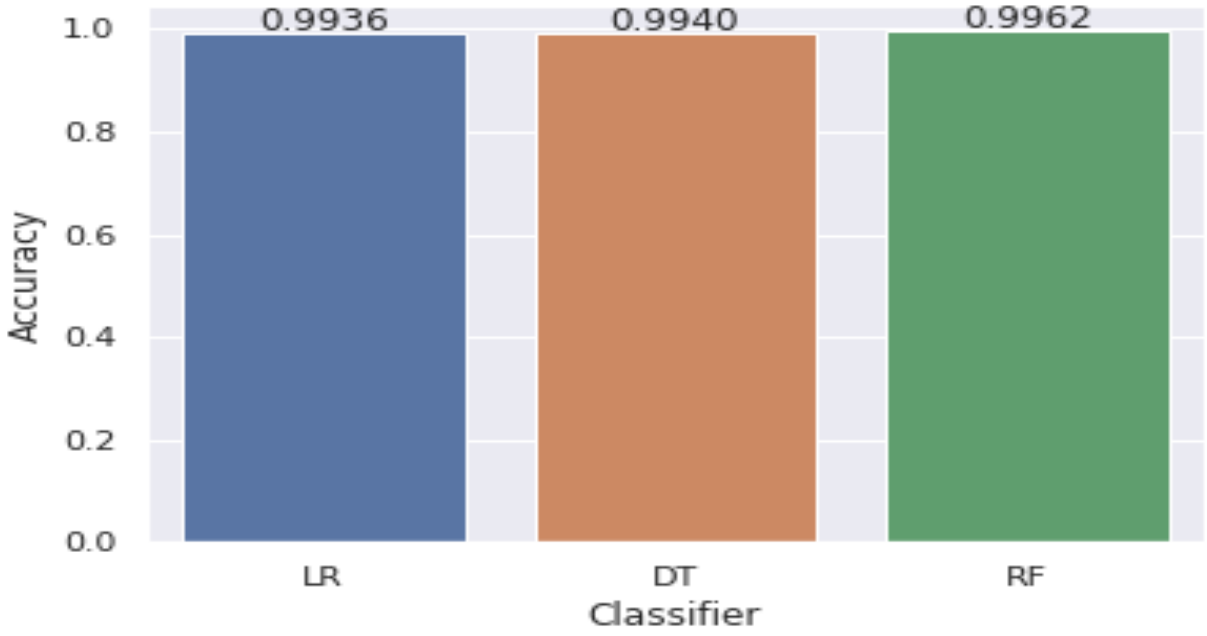

```
[127] from sklearn.metrics import precision_score,recall_score
      print('precision :%.3f'%precision_score(y_test, test_y_pred))
      print('recall :%.3f'%recall_score (y_test, test_y_pred))
      print('accuracy :%.3f'%accuracy_score (y_test, test_y_pred))
```

```
precision :0.758
recall :0.505
accuracy :0.996
```

Performance analysis for three different algorithms

Feature Selection	Logistic regression	Decision tree	Random forest
Accuracy	0.9936	0.9939	0.996
Error rate	0.006	0.006	0.003

Represent the plot of accuracy obtained using logistic regression, decision tree and random forest



In this paper, Machine learning technique like Logistic regression, Decision Tree and Random forest were used to detect the fraud in credit card system. Accuracy is used to evaluate the performance for the proposed system. The accuracy for logistic regression, Decision tree and random forest classifier are 99.3, 96.4, and 99.6 respectively. By comparing all the three methods, found that random forest classifier is better than the logistic regression and decision tree. Thus, we use random forest to build our model for prediction.

PREDICTIONMODEL

Predicting fraud transactions

```
[ ]  data2=[[0.936726,-0.67569,0.738772,-1.877811]]
     data3=[[1.388081, -0.476154 ,1.239504,-1.437726]]
```

```
[ ]  predict_fraud=rf_clf.predict(data3)
     print(predict_fraud)
     if(predict_fraud==1):
       print("fraud")
     else:
       print("genuine")
```

```
[1]
fraud
```

Predicting genuine transactions

```
predict_fraud=rf_clf.predict(data2)
print(predict_fraud)
if(predict_fraud==1):
  print("fraud")
else:
  print("genuine")
```

```
[0]
genuine
```

Random forest is used as the prediction model and it is called as rf_clf() , which predicts whether transaction fraud or non-fraud using given data.

5. CONCLUSION

The key objective of any credit card fraud detection system is to identify suspicious events and report them to an analyst while letting normal transactions be automatically processed. This article proposes the implementation of a hybrid approach that makes use of unsupervised outlier scores to extend the feature set of a fraud detection classifier. The novelty of the contribution, beyond its applications in real and sizeable datasets of credit card transactions, is the implementation and assessment of different levels of granularity for the definition of an outlier score.

In this paper, we studied applications of machine learning like Logistic regression, Random Forest with boosting, decision tree and shows that it proves accurate in deducting fraudulent transaction and minimizing the number of false alerts. If these algorithms are applied into bank credit card fraud detection system, the probability of fraud transactions can be predicted soon after credit card transactions.

We investigated the data, checking for data unbalancing, visualizing the features, and understanding the relationship between different features. We then investigated two predictive models. The data was split into two parts, a train set and a test set. We started with Logistic Classifier and then with Random Forest Classifier, for which we obtained an AUC code of 0.81 and 0.85, respectively, when predicting the target for the test set. By comparing all the three methods, we found that random forest classifier with boosting technique is better than the logistic regression and decision methods.

Random Forest is a supervised machine learning algorithm that is used widely in classification and regression problems. It is great with high dimensional data since we are working subsets of data. It is faster to train than decision tree because we are working only on subset of features in this model, so we can easily work with hundreds of features. Future scope from the above comparative analysis of the various credit card fraud detection techniques it is clear that Random Forest with Boosting technique performs best in this scenario.

6. REFERENCES

1. A Novel Feature Engineering Methodology for Credit Card Fraud Detection with a Deep Learning Architecture

DOI: https://doi.org/10.1016/j.ins.2019.05.023

2. Combining Unsupervised and Supervised Learning in Credit Card Fraud Detection

DOI:https://doi.org/10.1016/j.ins.2019.05.042

3. Credit card fraud detection using artificial neural network

DOI: www.elsevier.com/locate/gltp