Credit Card Fraud Detection:
A Classification Analysis

1921012 - Jeel Shah

ABSTRACT

- Credit Card Fraud Detection is a consistently developing threat with far outcomes in the money related industry.
- With the quick improvement of electronic business, the quantity of exchanges by credit cards are expanding quickly.
- Misrepresentation can be recognized by dissecting spending conduct of clients from past exchange information.
- The goal of this paper is to give relative investigation of various methods to recognize extortion.

INTRODUCTION

- Credit card misrepresentation can be characterized as "the unapproved utilization of a person's charge card or card data to make buys, or to expel reserves from the cardholder's record".
- The wrongdoing of Credit card extortion starts when somebody either takes a credit or check card, or falsely gets the card number and other record data important for the card to be utilized effectively

COMPONENTS OF CREDIT CARD FRAUD

- Credit card Theft
- Credit card counterfeiting
- Credit card Fraud

LITERATURE REVIEW

Different Fraud location Models (FDMs) have been proposed in past years. Each model had demonstrated its great work with a one of a kind dataset, however no model fitting for all datasets. There are different models for distinguishing extortion in Credit card:

- Dempster–Shafer theory and Bayesian learning:
- Detection of Credit Card Fraud using Fuzzy
- Bayesian and Neural Networks
- Fraud detection using Decision Tree and SVM
- Game-Theory Approach and many more..

DIFFERENT TYPE OF FRAUD TECHNIQUES

Merchant Related Frauds

- Vendor Collusion
- Triangulation

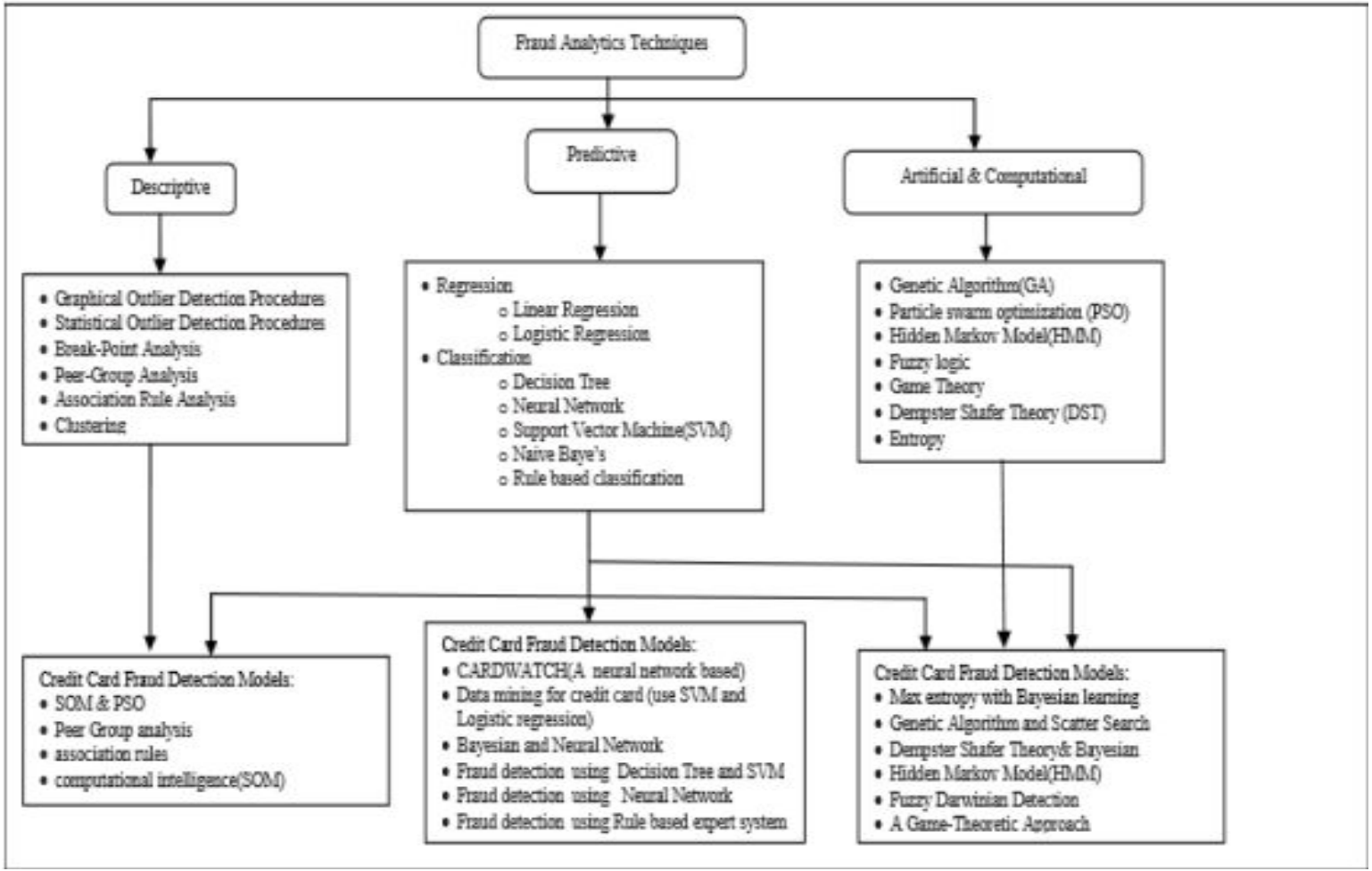
Web Related Frauds

- Site cloning
- False Merchant areas

OTHER FRAUD TECHNIQUES

- Lost/Stolen Cards
- Account Takeover
- Cardholder-Not-Present (CNP)
- Fake and Counterfeit Cards
- Erasing the alluring strip
- Making a fraud card
- Skimming
- Phishing

CLASSIFICATION OF FRAUD ANALYTICS TECHNIQUES



Implementation

DETAILS

Dataset Information

Dataset : creditcard.csv

Source : Kaggle

No. of Attributes : 30

No. of Instances : 284807

Algorithms applied on dataset

- Decision Tree Classifier
- Logistic Regression
- KNN

Code:

```
#import libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from termcolor import colored as cl # text customization
```

```
df=pd.read_csv('creditcard.csv') #import dataset
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
df.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

```
df.head(10)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.468177	-0.470
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.635558	0.463
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946	2.345865	-2.890
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924	-0.631418	-1.059
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670	0.175121	-0.451
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	-0.371407	1.341262	0.359894	-0.358091	-0.137134	0.517617	0.401
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	-0.099254	-1.416907	-0.153826	-0.751063	0.167372	0.050144	-0.443
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	1.249376	-0.619468	0.291474	1.757964	-1.323865	0.686133	-0.076
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	-0.410430	-0.705117	-0.110452	-0.286254	0.074355	-0.328783	-0.210
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	-0.366846	1.017614	0.836390	1.006844	-0.443523	0.150219	0.739

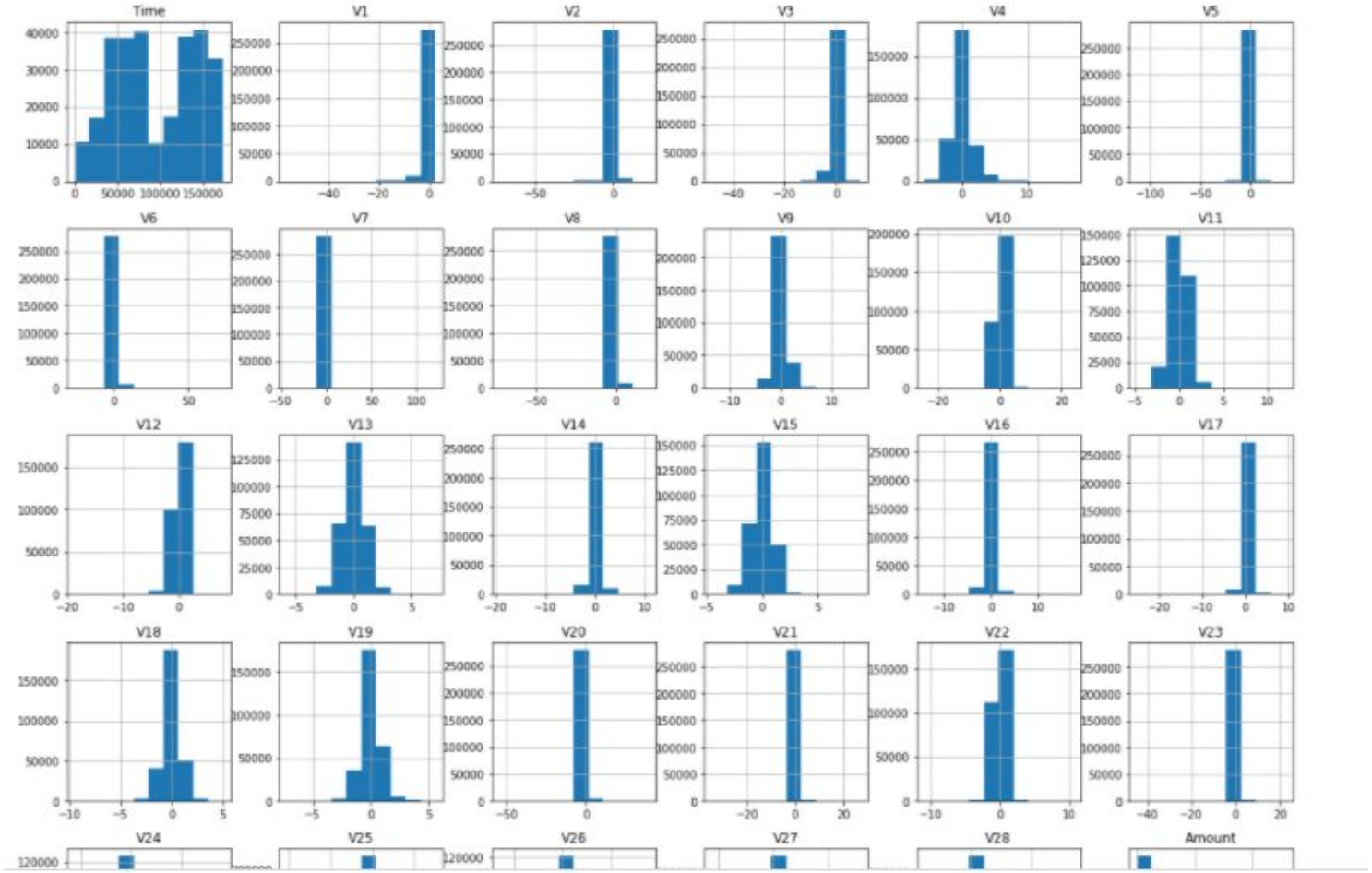
```
df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028e-16	-3.137024e-15	1.768627e-15	9.
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00	1.088850e+00	1.
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01	-2.458826e+01	-4.
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01	-5.354257e-01	-7.
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02	-9.291738e-02	-3.
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01	4.539234e-01	7.
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01	2.374514e+01	1.

```
df.shape
```

```
(284807, 31)
```

```
df.hist(figsize=(20,20))
plt.show()
```



```
cases = len(df)
nonfraud_count = len(df[df.Class == 0])
fraud_count = len(df[df.Class == 1])
fraud_percentage = round(fraud_count/nonfraud_count*100, 2)



print(cl('CASE COUNT', attrs=['bold']))
print('----------------------------------------')
print('Total number of cases : {}'.format(cases))
print('Number of Non-fraud cases : {}'.format(nonfraud_count))
print('Number of fraud cases : {}'.format(fraud_count))
print('Percentage of fraud cases : {}'.format(fraud_percentage),'%')
print('Percentage of Non-Fraud cases :', round(df['Class'].value_counts()[0]/len(df) * 100,2), '% ')
print('----------------------------------------')
```

```
CASE COUNT
----------------------------------------
Total number of cases : 284807
Number of Non-fraud cases : 284315
Number of fraud cases : 492
Percentage of fraud cases : 0.17 %
Percentage of Non-Fraud cases : 99.83 %
----------------------------------------
```

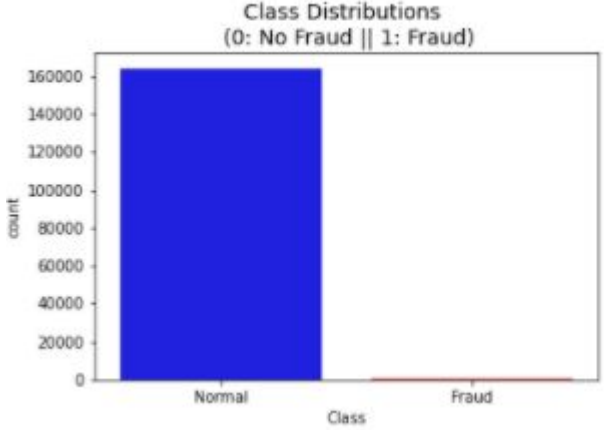
```
nonfraud_cases = df[df.Class == 0]
fraud_cases = df[df.Class == 1]

print(cl('CASE AMOUNT STATISTICS', attrs=['bold']))
print('-----------------------------------------------')
print('NON-FRAUD CASE AMOUNT STATS')
print('-----------------------------------------------')
print(nonfraud_cases.Amount.describe())
print('-----------------------------------------------')
print('FRAUD CASE AMOUNT STATS')
print('-----------------------------------------------')
print(fraud_cases.Amount.describe())
print('-----------------------------------------------')
```

CASE AMOUNT STATISTICS

```
-----------------------------------------------
NON-FRAUD CASE AMOUNT STATS
-----------------------------------------------
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
-----------------------------------------------
FRAUD CASE AMOUNT STATS
-----------------------------------------------
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
-----------------------------------------------
```

```
colors = ["blue", "red"]
LABELS = ["Normal", "Fraud"]
sns.countplot('Class', data=df, palette=colors)
plt.title('Class Distributions \n (0: No Fraud || 1: Fraud)', fontsize=14)
plt.xticks(range(2), LABELS)
```

```
[Text(0, 0, 'Normal'), Text(0, 0, 'Fraud')])
```



Splitting dataset into train and test

```
from sklearn.preprocessing import StandardScaler # data normalization
from sklearn.model_selection import train_test_split # data split
```

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

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

```
print(cl('X_train : ', attrs = ['bold']), X_train[:1])
print(cl('\nX_test : ', attrs = ['bold']), X_test[0:1])
print(cl('\ny_train : ', attrs = ['bold']), y_train[0:10])
print(cl('\ny_test  : ', attrs = ['bold']), y_test[0:10])
```

```
X_train :  [[ 1.30747000e+05  2.04716304e+00  1.07986610e-01 -1.80651506e+00
   7.27331875e-02  2.48370682e-01 -1.74483674e+00  7.12448447e-01
  -4.88842031e-01 -1.02709460e-01  1.81618007e-01  1.63122533e+00
   1.17614185e+00 -2.41013415e-01  1.04099217e+00 -3.28446112e-01
  -6.73208022e-01 -2.45851702e-01 -3.31582037e-01  1.13071634e-01
  -2.61379891e-01  2.41017135e-01  8.22618416e-01  2.29997150e-02
   5.49867922e-01  3.22173126e-01  1.91755203e-01 -8.50246556e-02
  -8.42920007e-02  7.70000000e-01]]

X_test :  [[ 1.25821000e+05 -3.23333572e-01  1.05745525e+00 -4.83411518e-02
  -6.07204308e-01  1.25982115e+00 -9.17607168e-02  1.15910150e+00
  -1.24334606e-01 -1.74639536e-01 -1.64440065e+00 -1.11886302e+00
   2.02647310e-01  1.14596495e+00 -1.80235956e+00 -2.47177932e-01
  -6.09453515e-02  8.46605738e-01  3.79454387e-01  8.47262245e-01
   1.86409421e-01 -2.07098267e-01 -4.33890272e-01 -2.61613283e-01
  -4.66506063e-02  2.11512300e-01  8.29721214e-03  1.08494430e-01
   1.61139167e-01  4.00000000e+01]]

y_train :  [0 0 0 0 0 0 0 0 0 0]

y_test  :  [0 0 0 0 0 0 0 0 0 0]
```

Feature scaling

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
print(cl('X_train : ', attrs = ['bold']), X_train)
print(cl('\nX_test : ', attrs = ['bold']), X_test)
```

```
X_train :  [[ 2.13095172  1.20192078 -0.04839928 ... -0.18262243 -0.18590735
  -0.34466583]
 [ 1.60843287  1.13721891 -0.80285121 ... -0.06691768 -0.08680639
   0.1106864 ]
 [ 2.20638035 -0.5393537  -0.31161051 ... -0.38774645 -0.34646262
   0.27186345]
 ...
 [ 1.48417251  1.07524941 -1.08393429 ... -0.09215102 -0.11195668
   0.33683932]
 [ 0.62079657 -0.21307634  0.43191634 ...  0.21324488  0.37220944
  -0.28574261]
 [-0.66569765  0.61187667 -0.11997698 ...  0.03738649  0.0631412
   0.03055884]]

X_test :  [[-0.5682658   0.79636937 -0.44150776 ...  0.02850074  0.02699337
  -0.18549297]
 [-0.47913341  0.73839498 -0.24590939 ... -0.0974579   0.01508115
  -0.34582826]
 [-0.17278666  0.7120017  -0.69981147 ... -0.13798873  0.04924422
   0.27486973]
 ...
 [ 0.9587971  -0.97309762  0.37169258 ... -1.43878169 -1.04482915
  -0.31075491]
 [-0.02914801 -0.86958458 -0.5905282  ... -0.29307754 -0.13099926
   0.66199929]
 [ 2.11597409  0.7914549   1.54809538 ...  0.19911124 -0.32376829
  -0.26193282]]
```

Decision Tree Classifier

```
#Training Decision Tree Classification model on the Training set
from sklearn.tree import DecisionTreeClassifier # Decision tree algorithm
from sklearn import tree

classifier = DecisionTreeClassifier( criterion = 'entropy', random_state = 0)
model=classifier.fit(X_train, y_train)
```

```
import matplotlib.pyplot as plt
tree.plot_tree(classifier,filled=True)
plt.show()
```

```
#predictions
dt_pred = classifier.predict(X_test)
print(dt_pred)
```

```
[0 0 0 ... 0 0 0]
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

cm = confusion_matrix(y_test, dt_pred)
print('Confusion MAtrix: \n',cm)

acc=accuracy_score(y_test, dt_pred)
print('\nAccuracy:',acc)

print('\nClassification Report: \n\n',classification_report(y_test,dt_pred))
```

```
Confusion MAtrix:
 [[71041    41]
 [   28    92]]

Accuracy: 0.9990309260975815

Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71082
           1       0.69      0.77      0.73       120

    accuracy                           1.00     71202
   macro avg       0.85      0.88      0.86     71202
weighted avg       1.00      1.00      1.00     71202
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression # Logistic regression algorithm
```

```
lr = LogisticRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
```

```
lr_pred
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
cm2 = confusion_matrix(y_test, lr_pred)
print('Confusion MAtrix: \n',cm2)

acc2=accuracy_score(y_test, lr_pred)
print('\nAccuracy:',acc2)

print('\nClassification Report: \n\n',classification_report(y_test,lr_pred))
```

```
Confusion MAtrix:
 [[71057    25]
 [   39    81]]

Accuracy: 0.9991011488441336

Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71082
           1       0.76      0.68      0.72       120

    accuracy                           1.00     71202
   macro avg       0.88      0.84      0.86     71202
weighted avg       1.00      1.00      1.00     71202
```

KNN

```
from sklearn.neighbors import KNeighborsClassifier # KNN algorithm
n = 5

knn = KNeighborsClassifier(n_neighbors = n)
knn.fit(X_train, y_train)
knn_pred = knn.predict(X_test)
```

```
knn_pred
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

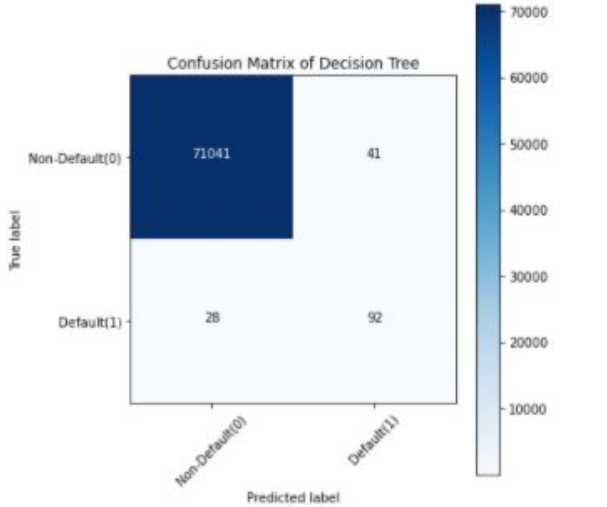
```
cm1 = confusion_matrix(y_test, knn_pred)
print('Confusion MAtrix: \n',cm1)

acc1=accuracy_score(y_test, knn_pred)
print('\nAccuracy:',acc1)

print('\nClassification Report: \n\n',classification_report(y_test,knn_pred))
```

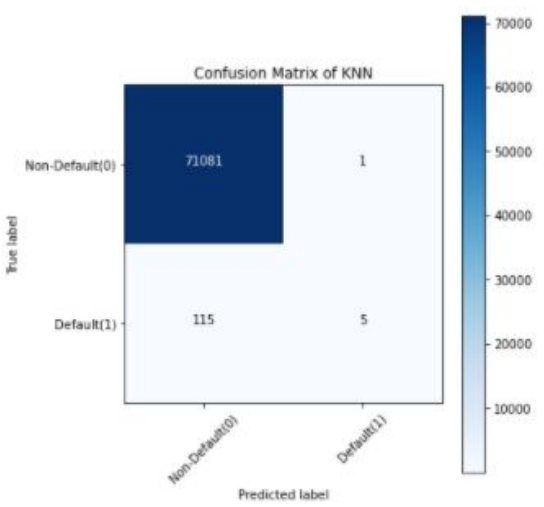
```
Confusion MAtrix:
 [[71081     1]
 [  115     5]]

Accuracy: 0.9983708322799921

Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00     71082
           1       0.83      0.04      0.08       120

    accuracy                           1.00     71202
   macro avg       0.92      0.52      0.54     71202
weighted avg       1.00      1.00      1.00     71202
```

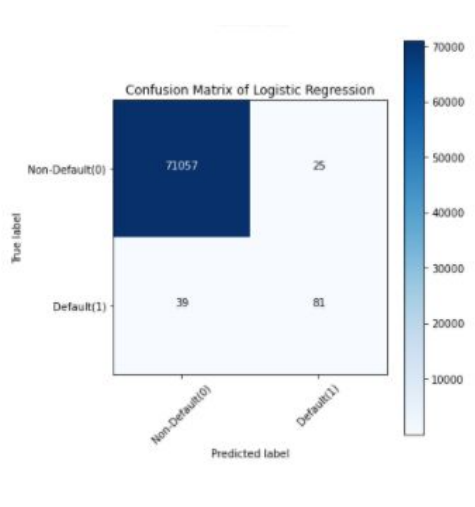
Comparing the results

```
print(cl('ACCURACY SCORE' , attrs = ['bold']))
print('Accuracy score of the Decision Tree model is {}'.format(accuracy_score(y_test, dt_pred)))
print('----------------------------------------------------------------------------')
print('Accuracy score of the KNN model is {}'.format(accuracy_score(y_test, knn_pred)))
print('----------------------------------------------------------------------------' )
print('Accuracy score of the Logistic Regression model is {}'.format(accuracy_score(y_test, lr_pred)))
```

```
ACCURACY SCORE
Accuracy score of the Decision Tree model is 0.9990309260975815
----------------------------------------------------------------------------
Accuracy score of the KNN model is 0.9983708322799921
----------------------------------------------------------------------------
Accuracy score of the Logistic Regression model is 0.9991011488441336
```







CONCLUSION

This paper presents arrangement of charge card the challenges looked via cardholder and in addition the card guarantor, verity of misrepresentation executed by the people

Future Scope

The future scope is not mentioned by the author but the author could suggest KNN model as one of the techniques .
so here we consider it as a future scope.

References

http://ieeexplore.ieee.org/document/8653770

Thank You

1921012 - Jeel Shah