Task5Credit-card-fraud-detection

August 4, 2024

1 CodSoft DataScience Internship

1.1 Task 5: CREDIT CARD FRAUD DETECTION

[9]:
```
# importing libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[3]:
```
# loading dataset
df = pd.read_csv('creditcard.csv')
df.head()
```

[3]:
```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  …       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

Statistical vaues

[4]:
```
df.describe()
```

[4]:
```
                Time            V1            V2            V3            V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean    94813.859575  1.168375e-15  3.416908e-16 -1.379537e-15  2.074095e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00  1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00  7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00  1.687534e+01

                 V5            V6            V7            V8            V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   9.604066e-16  1.487313e-15 -5.556467e-16  1.213481e-16 -2.406331e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00  1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  1.559499e+01

       …           V21           V22           V23           V24  \
count  …  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
mean   …  1.654067e-16 -3.568593e-16  2.578648e-16  4.473266e-15
std    …  7.345240e-01  7.257016e-01  6.244603e-01  6.056471e-01
min    … -3.483038e+01 -1.093314e+01 -4.480774e+01 -2.836627e+00
25%    … -2.283949e-01 -5.423504e-01 -1.618463e-01 -3.545861e-01
50%    … -2.945017e-02  6.781943e-03 -1.119293e-02  4.097606e-02
75%    …  1.863772e-01  5.285536e-01  1.476421e-01  4.395266e-01
max    …  2.720284e+01  1.050309e+01  2.252841e+01  4.584549e+00

                V25           V26           V27           V28         Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  284807.000000
mean   5.340915e-16  1.683437e-15 -3.660091e-16 -1.227390e-16      88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01     250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01       0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02       5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02      22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02      77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01   25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
```

```
25%          0.000000
50%          0.000000
75%          0.000000
max          1.000000

[8 rows x 31 columns]
```

Checking the null values

[5]:
```
df.isnull().sum()
```

[5]:
```
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
```

we can see the dataset is very clean

Extracting information about all columns

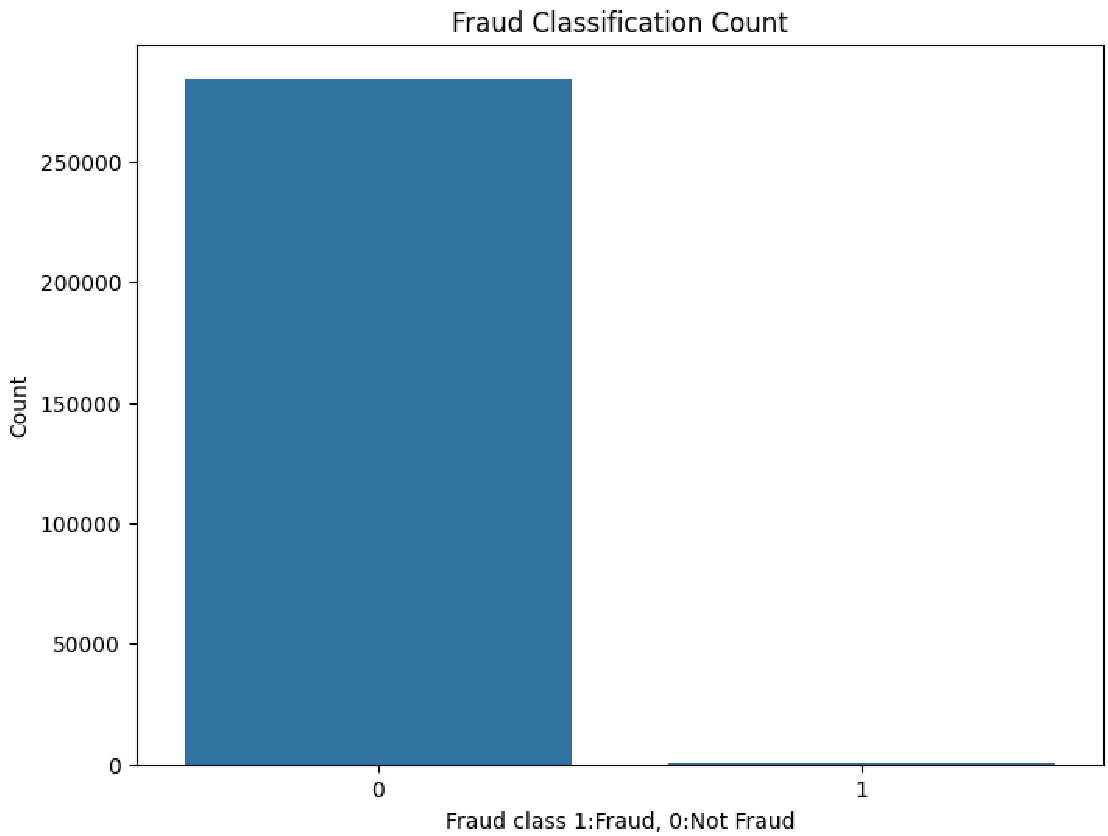
[6]:
```
df.info()
```

```
<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
```

1.2 Data visualization

[10]:
```
fit = plt.figure(figsize=(8,6))
sns.countplot(x = 'Class', data = df)
plt.xlabel('Fraud class 1:Fraud, 0:Not Fraud')
plt.ylabel('Count')
```

```
plt.title("Fraud Classification Count")
plt.show()
```



1.3 Setting Up Dataset For Model Development

[13]:
```
X = df.drop(['Class'],axis = 1)
X.head()
```

[13]:

	Time	V1	V2	V3	V4	V5	V6	V7	\
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	

	V8	V9	…	V20	V21	V22	V23	V24	\
0	0.098698	0.363787	…	0.251412	-0.018307	0.277838	-0.110474	0.066928	
1	0.085102	-0.255425	…	-0.069083	-0.225775	-0.638672	0.101288	-0.339846	
2	0.247676	-1.514654	…	0.524980	0.247998	0.771679	0.909412	-0.689281	
3	0.377436	-1.387024	…	-0.208038	-0.108300	0.005274	-0.190321	-1.175575	

```
4 -0.270533  0.817739  …  0.408542 -0.009431  0.798278 -0.137458  0.141267

        V25       V26       V27       V28  Amount
0  0.128539 -0.189115  0.133558 -0.021053  149.62
1  0.167170  0.125895 -0.008983  0.014724    2.69
2 -0.327642 -0.139097 -0.055353 -0.059752  378.66
3  0.647376 -0.221929  0.062723  0.061458  123.50
4 -0.206010  0.502292  0.219422  0.215153   69.99

[5 rows x 30 columns]
```

[15]:
```python
Y = df['Class']
Y.head()
```

[15]:
```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

1.3.1 Splitting data in the train and test set

[16]:
```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(X,Y,test_size=0.2,random_state␣
 ↪= 5)
```

1.4 Model Development

1.4.1 KNeighbor-Nearest Classification

[17]:
```python
from sklearn.neighbors import KNeighborsClassifier
neighbor = KNeighborsClassifier(n_neighbors = 5)
```

Training tahe model

[18]:
```python
neighbor.fit(x_train,y_train)
```

[18]:
```
KNeighborsClassifier()
```

Making predictions

[27]:
```python
prediction1 = neighbor.predict(x_test)
```

[35]:
```python
print(prediction1)
```

```
[0 0 0 … 0 0 0]
```

Model Evaluation

[37]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
accuracy = accuracy_score(y_test,prediction1)
conMatrix = confusion_matrix(y_test,prediction1)
```

[42]:
```
print("Accuracy score is {:.2f}%".format(accuracy*100))
```

```
Accuracy score is 99.82%
```

[43]:
```
print("ConfusionMatrix:\n",conMatrix)
```

```
ConfusionMatrix:
 [[56856     2]
 [   98     6]]
```

The Model is 99.82% accurate to detect the fraud.

1.5 LogisticRegression Classification Model

[44]:
```
from sklearn.linear_model import LogisticRegression
```

[55]:
```
LR = LogisticRegression(C = 0.001,solver = 'liblinear')
```

Fitting the data

[56]:
```
LR.fit(x_train,y_train)
```

[56]:
```
LogisticRegression(C=0.001, solver='liblinear')
```

Making prediction

[57]:
```
LRpredict = LR.predict(x_test)
print(LRpredict)
```

```
[0 0 0 … 0 0 0]
```

Model Evaluation

[58]:
```
print("Accuracy of the Model is  {:.2f}%.".
 ↪format(accuracy_score(y_test,LRpredict)*100))
```

```
Accuracy of the Model is  99.89%.
```

[59]:
```
print("ConfusionMatrix :\n",confusion_matrix(y_test,LRpredict))
```

```
ConfusionMatrix :
 [[56841    17]
 [   47    57]]
```

This model gives more accurate prediction than KNN model

1.6 DecisionTree Classifier

[63]:
```
from sklearn.tree import DecisionTreeClassifier
import sklearn.tree as tree
```

[61]:
```
DST = DecisionTreeClassifier()
```

fitting the model

[62]:
```
DST.fit(x_train,y_train)
```

[62]:
```
DecisionTreeClassifier()
```

Prediction using test data set

[65]:
```
DSprediction = DST.predict(x_test)
```

[66]:
```
print(DSprediction)
```

```
[0 0 0 … 0 0 0]
```

Model Evaluation

[69]:
```
print("Confusion Matrix:")
confusion_matrix(y_test,DSprediction)
```

```
Confusion Matrix:
```

[69]:
```
array([[56848,    10],
       [   19,    85]])
```

[71]:
```
print('Accuracy of the model is {:.2f}%.'.
 ↪format(accuracy_score(y_test,DSprediction)*100))
```

```
Accuracy of the model is 99.95%.
```

2 Conclusions

From the above models the most accurate model is decision tree model so DecisionTreeClassifier is the best predictor for Fraud Detection in Credit card.

3 Author

Suresh Tamang

Internship Id: CS11WX331283

[ ]:
```

```