MACHINE LEARNING Engineer Course

GRADUATION ASSIGNMENT PROJECT

DIVE INTO  CODE

Project: *Credit* ***Cards Fraud Detection in Sierra Leone***

SELF-INTRODUCTION

I AM FODAY BANGURA

- ATTENDED THE UNIVERSITY OF MANAGEMENT AND TECHNOLOGY(UNIMTECH)
- COMPUTER SCIENCE DEPARTMENT
- DEGREE YEAR THREE(3)
- SIERRA LEONE.

University of Idaho

Credit Cards Fraud Detection In Sierra Leone

Introduction

Context:
It is important that credit card companies are able to recognize fraudulent credit card transaction so that customers are not charged for items that they did not purchase. This is a big problem in Sierra Leone.

Content:
I were hoping to real dataset from Sierra Leone but since it was not possible, I used dataset from kaggle in order to build a module so that in the further i can do the same thing for a real dataset of credit card frauds in Sierra Leone.

The datasets contains transactions made by credit cards in September 2013 European cardholders. This dataset presents transactions that occurred in two days, where they have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced.

The positive class (frauds) account for 0.172% of all transactions. In this project, I will be predicting whether a transaction is fraud or not fraud by using predictive models.

It contains only numerical input variables which are the result of the Principal component Analysis(PCA) transformation. Unfortunately, due to confidentiality issues, I cannot provide the original features and more background information about the data.

The dataset given contains features V1 to V28 which are a result of PCA dimensionality reduction to protect user identities and sensitive features (according to the description) are the principal components obtained with PCA, the only features which have not been transformed with PCA are ‘Time’ and ‘Amount’. Feature ‘Time’ contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature ‘Amount’ is the transaction Amount this feature can be use for example-dependent cost-sensitive learning . Feature ‘Class’ is the response variable and it takes the value of 1 in the class if it is fraud and the value of 0 in the class if it is not fraud.

Inspiration:
Identify fraudulent credit card transaction.
Given the class imbalance ratio, I recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC).Confusion matrix accuracy is not meaningful for unbalance classification.

Acknowledgement:
The dataset has been collected and analyzed during a research on big data mining and fraud detection.

```
#Import the required libraries

import numpy as np
import pandas as pd
import sklearn
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report,accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams
rcParams['figure.figsize'] = 14, 8
RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]
import plotly.plotly as py
import plotly.graph_objs as go
import plotly
import plotly.figure_factory as ff
from plotly.offline import init_notebook_mode, iplot
```

```
data = pd.read_csv('../input/creditcard_data.csv')
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
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
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
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
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
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

V12	V13	V14	V15	V16	V17	V18	V19	V20	V21	V22	V23
-0.617801	-0.991390	-0.311169	1.468177	-0.470401	0.207971	0.025791	0.403993	0.251412	-0.018307	0.277838	-0.110474
1.065235	0.489095	-0.143772	0.635558	0.463917	-0.114805	-0.183361	-0.145783	-0.069083	-0.225775	-0.638672	0.101288
0.066084	0.717293	-0.165946	2.345865	-2.890083	1.109969	-0.121359	-2.261857	0.524980	0.247998	0.771679	0.909412
0.178228	0.507757	-0.287924	-0.631418	-1.059647	-0.684093	1.965775	-1.232622	-0.208038	-0.108300	0.005274	-0.190321
0.538196	1.345852	-1.119670	0.175121	-0.451449	-0.237033	-0.038195	0.803487	0.408542	-0.009431	0.798278	-0.137458

V24	V25	V26	V27	V28	Amount	Class
0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

From the above table - There are no missing values in the dataset

In [5]:
```
data.describe()
```

Out[5]:

	Time	V1	V2	V3	V4	V5	V6	V7
count	284806.000000	284806.000000	2.848060e+05	284806.000000	284806.000000	2.848060e+05	284806.000000	284806.000000
mean	94813.585781	0.000002	6.661837e-07	-0.000002	0.000002	4.405007e-08	0.000002	-0.000006
std	47488.004530	1.958699	1.651311e+00	1.516257	1.415871	1.380249e+00	1.332273	1.237092
min	0.000000	-56.407510	-7.271573e+01	-48.325589	-5.683171	-1.137433e+02	-26.160506	-43.557242
25%	54201.250000	-0.920374	-5.985522e-01	-0.890368	-0.848642	-6.915995e-01	-0.768296	-0.554080
50%	84691.500000	0.018109	6.549621e-02	0.179846	-0.019845	-5.433621e-02	-0.274186	0.040097
75%	139320.000000	1.315645	8.037257e-01	1.027198	0.743348	6.119267e-01	0.398567	0.570426
max	172788.000000	2.454930	2.205773e+01	9.382558	16.875344	3.480167e+01	73.301626	120.589494

V8	V9	V10	V11	V12	V13	V14	V15
284806.000000	284806.000000	284806.000000	284806.000000	2.848060e+05	2.848060e+05	2.848060e+05	2.848060e+05
0.000001	-0.000002	0.000003	0.000004	1.106485e-07	6.604254e-07	2.960486e-07	-1.451290e-07
1.194355	1.098634	1.088850	1.020713	9.992031e-01	9.952759e-01	9.585973e-01	9.153176e-01
-73.216718	-13.434066	-24.588262	-4.797473	-1.868371e+01	-5.791881e+00	-1.921433e+01	-4.498945e+00
-0.208628	-0.643098	-0.535422	-0.762485	-4.055742e-01	-6.485416e-01	-4.255807e-01	-5.828853e-01
0.022358	-0.051429	-0.092913	-0.032757	1.400356e-01	-1.356316e-02	5.060308e-02	4.807416e-02
0.327346	0.597140	0.453934	0.739595	6.182391e-01	6.625058e-01	4.931500e-01	6.488220e-01
20.007208	15.594995	23.745136	12.018913	7.848392e+00	7.126883e+00	1.052677e+01	8.877742e+00

V16	V17	V18	V19	V20	V21	V22	V23
284806.000000	284806.000000	2.848060e+05	2.848060e+05	284806.000000	2.848060e+05	284806.000000	284806.000000
0.000001	0.000002	-5.878731e-07	8.992685e-07	-0.000001	-9.166149e-07	-0.000002	-0.000001
0.876254	0.849338	8.381776e-01	8.140418e-01	0.770926	7.345251e-01	0.725702	0.624461
-14.129855	-25.162799	-9.498746e+00	-7.213527e+00	-54.497720	-3.483038e+01	-10.933144	-44.807735
-0.468046	-0.483745	-4.988498e-01	-4.563030e-01	-0.211722	-2.283974e-01	-0.542351	-0.161846
0.066418	-0.065673	-3.639113e-03	3.736578e-03	-0.062481	-2.945020e-02	0.006781	-0.011196
0.523300	0.399676	5.008082e-01	4.589502e-01	0.133034	1.863701e-01	0.528548	0.147641
17.315112	9.253526	5.041069e+00	5.591971e+00	39.420904	2.720284e+01	10.503090	22.528412

V24	V25	V26	V27	V28	Amount	Class
2.848060e+05	284806.000000	284806.000000	2.848060e+05	2.848060e+05	284806.000000	284806.000000
-3.088756e-08	0.000002	0.000003	8.483873e-09	-4.792707e-08	88.349168	0.001727
6.056481e-01	0.521278	0.482225	4.036332e-01	3.300838e-01	250.120432	0.041527
-2.836627e+00	-10.295397	-2.604551	-2.256568e+01	-1.543008e+01	0.000000	0.000000
-3.545895e-01	-0.317142	-0.326979	-7.083961e-02	-5.295995e-02	5.600000	0.000000
4.097671e-02	0.016596	-0.052134	1.342244e-03	1.124381e-02	22.000000	0.000000
4.395270e-01	0.350716	0.240955	9.104579e-02	7.828043e-02	77.160000	0.000000
4.584549e+00	7.519589	3.517346	3.161220e+01	3.384781e+01	25691.160000	1.000000

[3]:
```
data1= data.sample(frac = 0.1,random_state=1)
data1.shape
```

[3]: (28481, 31)

[123]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284806 entries, 0 to 284805
Data columns (total 31 columns):
Time       284806 non-null float64
V1         284806 non-null float64
V2         284806 non-null float64
V3         284806 non-null float64
V4         284806 non-null float64
V5         284806 non-null float64
V6         284806 non-null float64
V7         284806 non-null float64
V8         284806 non-null float64
V9         284806 non-null float64
V10        284806 non-null float64
V11        284806 non-null float64
V12        284806 non-null float64
V13        284806 non-null float64
V14        284806 non-null float64
V15        284806 non-null float64
V16        284806 non-null float64
V17        284806 non-null float64
V18        284806 non-null float64
V19        284806 non-null float64
V20        284806 non-null float64
V21        284806 non-null float64
V22        284806 non-null float64
V23        284806 non-null float64
V24        284806 non-null float64
V25        284806 non-null float64
V26        284806 non-null float64
V27        284806 non-null float64
V28        284806 non-null float64
Amount     284806 non-null float64
Class      284806 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

From the above table - There are no missing values in the dataset

```
data.describe()
```

[5]:

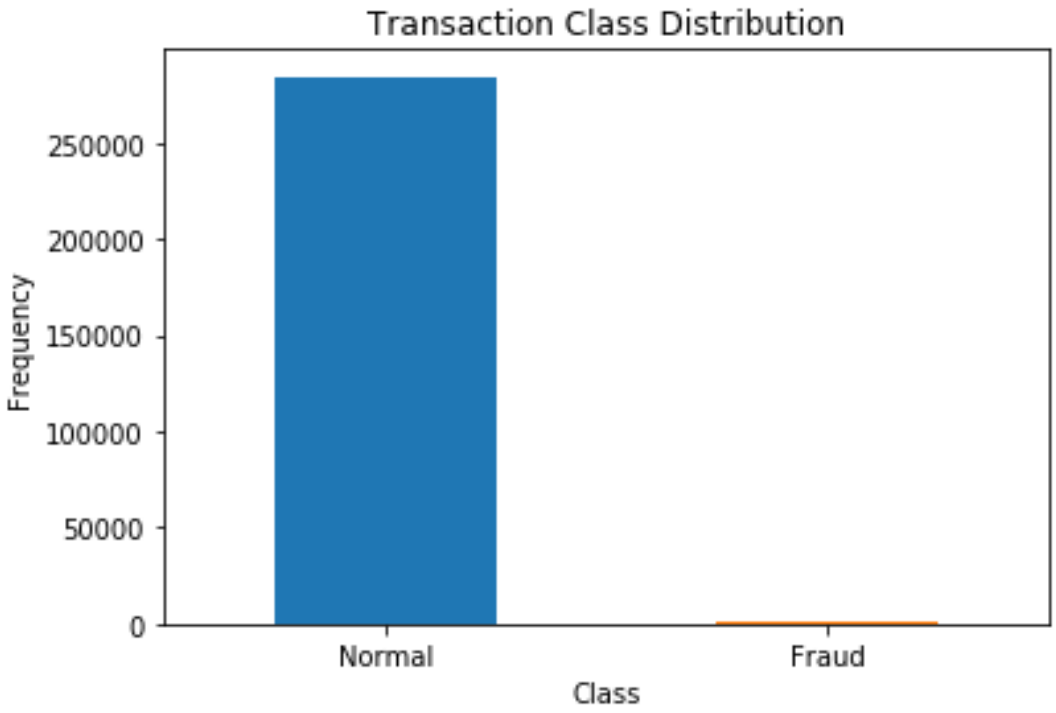
	Time	V1	V2	V3	V4	V5	V6	V7	V8
count	284806.000000	284806.000000	2.848060e+05	284806.000000	284806.000000	2.848060e+05	284806.000000	284806.000000	284806.000000
mean	94813.585781	0.000002	6.661837e-07	-0.000002	0.000002	4.405007e-08	0.000002	-0.000006	0.000001
std	47488.004530	1.958699	1.651311e+00	1.516257	1.415871	1.380249e+00	1.332273	1.237092	1.194355
min	0.000000	-56.407510	-7.271573e+01	-48.325589	-5.683171	-1.137433e+02	-26.160506	-43.557242	-73.216718
25%	54201.250000	-0.920374	-5.985522e-01	-0.890368	-0.848642	-6.915995e-01	-0.768296	-0.554080	-0.208628
50%	84691.500000	0.018109	6.549621e-02	0.179846	-0.019845	-5.433621e-02	-0.274186	0.040097	0.022358
75%	139320.000000	1.315645	8.037257e-01	1.027198	0.743348	6.119267e-01	0.398567	0.570426	0.327346
max	172788.000000	2.454930	2.205773e+01	9.382558	16.875344	3.480167e+01	73.301626	120.589494	20.007208

V9	V10	V11	V12	V13	V14	V15	V16	V17	V18
284806.000000	284806.000000	284806.000000	2.848060e+05	2.848060e+05	2.848060e+05	2.848060e+05	284806.000000	284806.000000	2.848060e+05
-0.000002	0.000003	0.000004	1.106485e-07	6.604254e-07	2.960486e-07	-1.451290e-07	0.000001	0.000002	-5.878731e-07
1.098634	1.088850	1.020713	9.992031e-01	9.952759e-01	9.585973e-01	9.153176e-01	0.876254	0.849338	8.381776e-01
-13.434066	-24.588262	-4.797473	-1.868371e+01	-5.791881e+00	-1.921433e+01	-4.498945e+00	-14.129855	-25.162799	-9.498746e+00
-0.643098	-0.535422	-0.762485	-4.055742e-01	-6.485416e-01	-4.255807e-01	-5.828853e-01	-0.468046	-0.483745	-4.988498e-01
-0.051429	-0.092913	-0.032757	1.400356e-01	-1.356316e-02	5.060308e-02	4.807416e-02	0.066418	-0.065673	-3.639113e-03
0.597140	0.453934	0.739595	6.182391e-01	6.625058e-01	4.931500e-01	6.488220e-01	0.523300	0.399676	5.008082e-01
15.594995	23.745136	12.018913	7.848392e+00	7.126883e+00	1.052677e+01	8.877742e+00	17.315112	9.253526	5.041069e+00

V19	V20	V21	V22	V23	V24	V25
2.848060e+05	284806.000000	2.848060e+05	284806.000000	284806.000000	2.848060e+05	284806.000000
8.992685e-07	-0.000001	-9.166149e-07	-0.000002	-0.000001	-3.088756e-08	0.000002
8.140418e-01	0.770926	7.345251e-01	0.725702	0.624461	6.056481e-01	0.521278
-7.213527e+00	-54.497720	-3.483038e+01	-10.933144	-44.807735	-2.836627e+00	-10.295397
-4.563030e-01	-0.211722	-2.283974e-01	-0.542351	-0.161846	-3.545895e-01	-0.317142
3.736578e-03	-0.062481	-2.945020e-02	0.006781	-0.011196	4.097671e-02	0.016596
4.589502e-01	0.133034	1.863701e-01	0.528548	0.147641	4.395270e-01	0.350716
5.591971e+00	39.420904	2.720284e+01	10.503090	22.528412	4.584549e+00	7.519589

V26	V27	V28	Amount	Class
284806.000000	2.848060e+05	2.848060e+05	284806.000000	284806.000000
0.000003	8.483873e-09	-4.792707e-08	88.349168	0.001727
0.482225	4.036332e-01	3.300838e-01	250.120432	0.041527
-2.604551	-2.256568e+01	-1.543008e+01	0.000000	0.000000
-0.326979	-7.083961e-02	-5.295995e-02	5.600000	0.000000
-0.052134	1.342244e-03	1.124381e-02	22.000000	0.000000
0.240955	9.104579e-02	7.828043e-02	77.160000	0.000000
3.517346	3.161220e+01	3.384781e+01	25691.160000	1.000000

```
#Determine the number of fraud and valid transactions in the entire dataset

count_classes = pd.value_counts(data['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency");
```



[7]:
```
#Assigning the transaction class "0 = NORMAL  & 1 = FRAUD"
Normal = data[data['Class']==0]
Fraud = data[data['Class']==1]
```

[8]:
```
Normal.shape
```

[8]: (284314, 31)

[9]:
```
Fraud.shape
```

[9]: (492, 31)

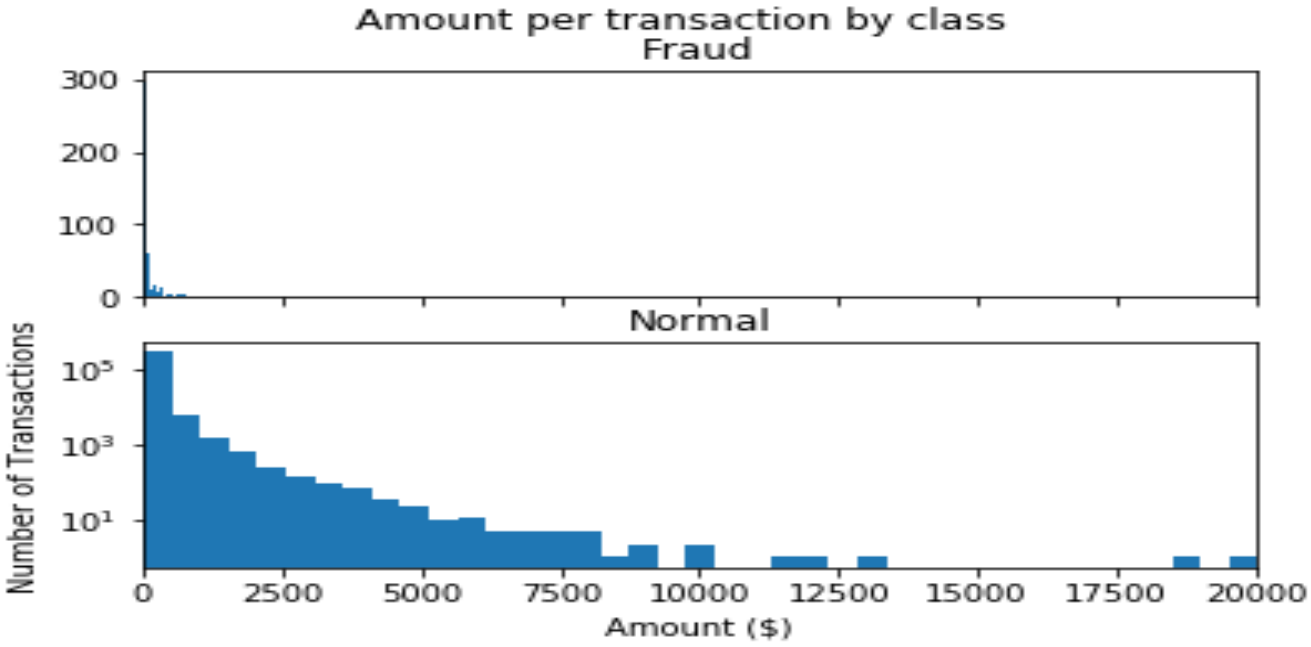
[10]:
```
#How different are the amount of money used in different transaction classes?

Normal.Amount.describe()
```

[10]:
```
count    284314.000000
mean         88.290570
std         250.105416
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

[11]:
```
#How different are the amount of money used in different transaction classes?

Fraud.Amount.describe()
```

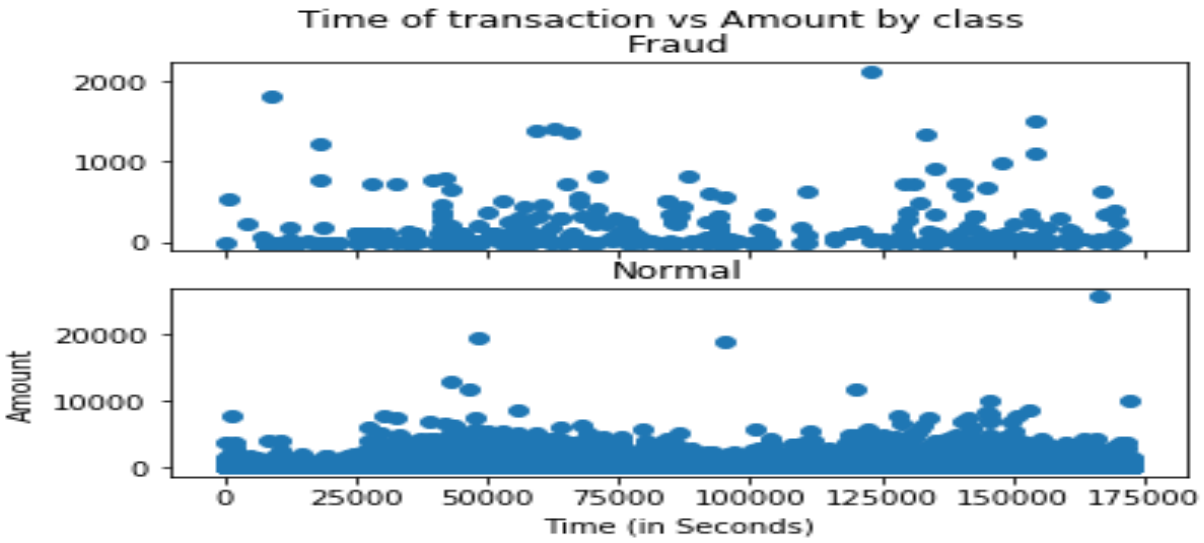
[11]:
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

```
#Let's have a more graphical representation of the data

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per transaction by class')
bins = 50
ax1.hist(Fraud.Amount, bins = bins)
ax1.set_title('Fraud')
ax2.hist(Normal.Amount, bins = bins)
ax2.set_title('Normal')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plt.yscale('log')
plt.show();
```



Do fraudulent transactions occur more often during certain time frame ? Let us find out with a visual representation.

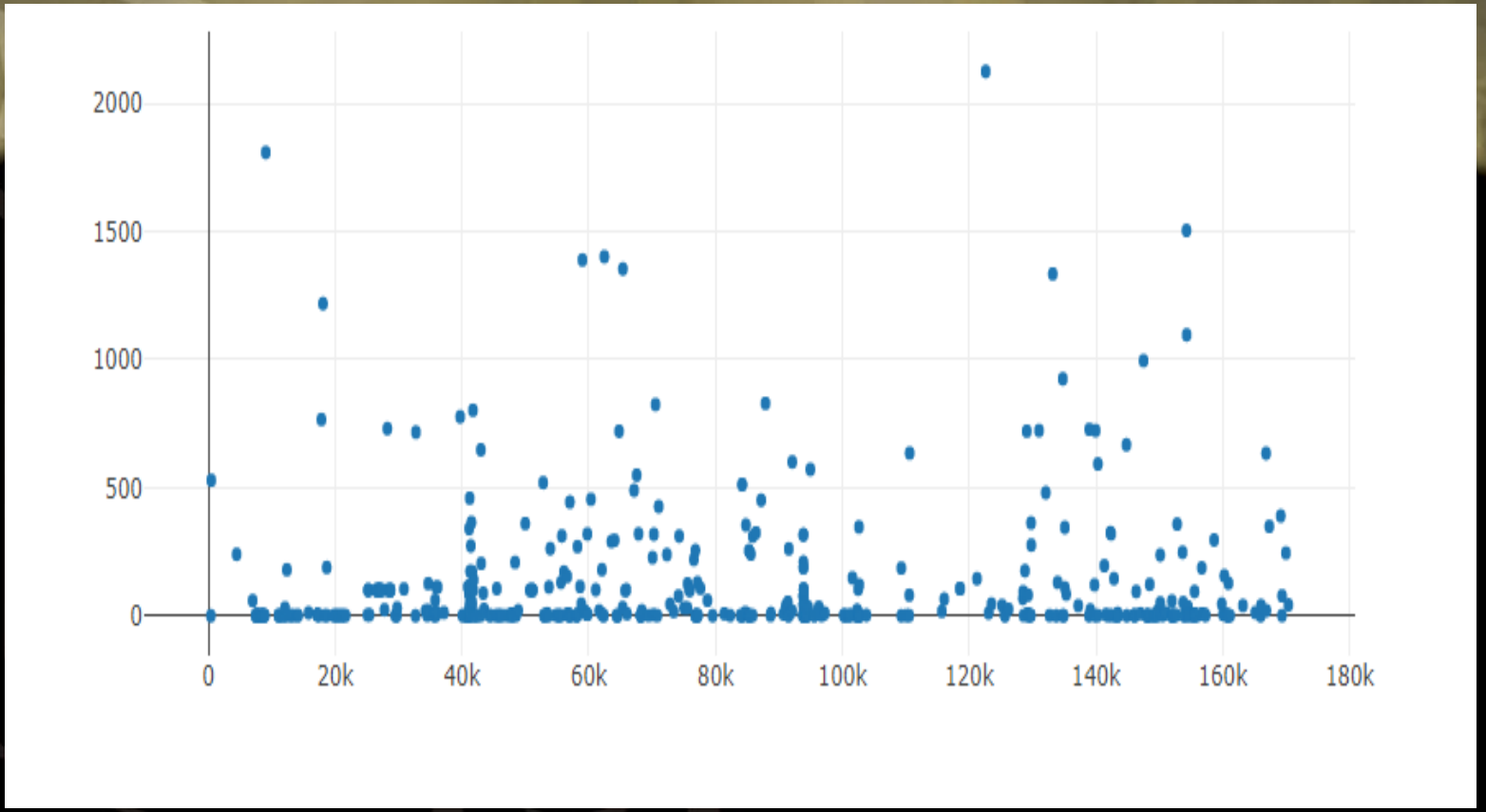
```
#Graphical representation of the data

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(Fraud.Time, Fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(Normal.Time, Normal.Amount)
ax2.set_title('Normal')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show();
```



In [14]:

```
init_notebook_mode(connected=True)
plotly.offline.init_notebook_mode(connected=True)
```

```
# Create a trace

trace = go.Scatter(
    x = Fraud.Time,
    y = Fraud.Amount,
    mode = 'markers'
)
data = [trace]

plotly.offline.iplot({
    "data": data
})
```



Doesn't seem like the time of transaction really matters here as per above observation. Now let us take a sample of the dataset for out modelling and prediction

[15]:
```
data1.shape
```

[15]:
```
(28481, 31)
```

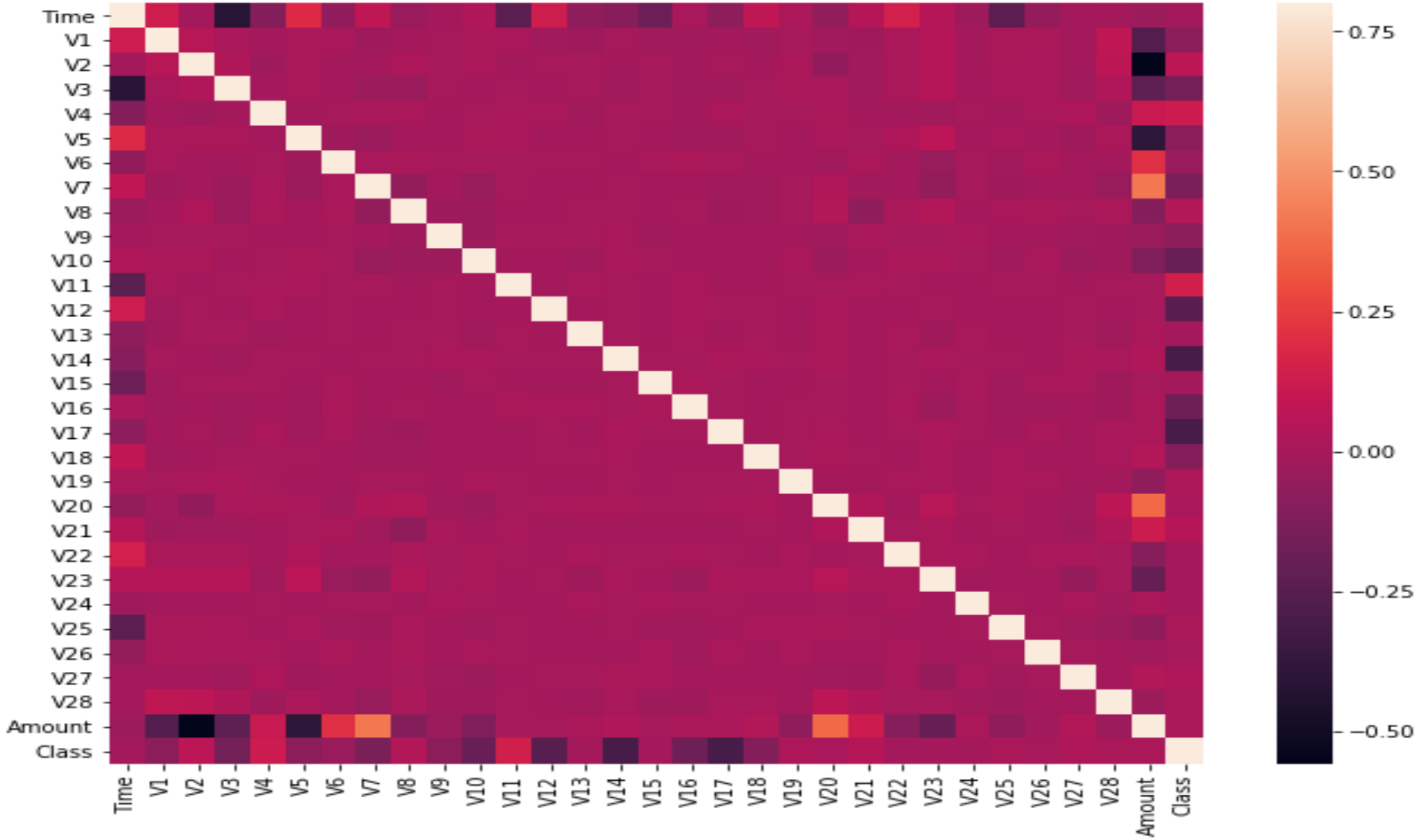
[16]:
```
#Determine the number of fraud and valid transactions in the dataset.

Fraud = data1[data1['Class']==1]
Valid = data1[data1['Class']==0]
outlier_fraction = len(Fraud)/float(len(Valid))
```

[17]:
```
#Now let us print the outlier fraction and no of Fraud and Valid Transaction cases

print(outlier_fraction)
print("Fraud Cases : {}".format(len(Fraud)))
print("Valid Cases : {}".format(len(Valid)))
```

```
0.0016529506928325245
Fraud Cases : 47
Valid Cases : 28434
```

[18]:

```
#Correlation Matrix

correlation_matrix = data1.corr()
fig = plt.figure(figsize=(12,9))
sns.heatmap(correlation_matrix,vmax=0.8,square = True)
plt.show()
```



The above correlation matrix shows that none of the V1 to V28 PCA components have any correlation to each other however if we observe Class has some form positive and negative correlations with the V components but has no correlation with Time and Amount.

[19]:

```
#Get all the columns from the dataframe

columns = data1.columns.tolist()
# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Class"]]
# Store the variable we are predicting
target = "Class"
# Define a random state
state = np.random.RandomState(42)
X = data1[columns]
Y = data1[target]
X_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))
# Print the shapes of X & Y
print(X.shape)
print(Y.shape)
```

```
(28481, 30)
(28481,)
```

Model Prediction:
Now it is time to start building the model .The types of algorithms that I am using to try to do anomaly detection on this dataset are as follows:

Isolation Forest Algorithm:
One of the new techniques to detect anomalies is called Isolation Forests. The algorithm is based on the fact that anomalies are data points that are few and different. As a result of these properties, anomalies are subjected to a mechanism called isolation.

This method is highly useful and is fundamentally different from all existing methods. It introduces the use of isolation as a more effective and efficient means to detect anomalies than the commonly used basic distance and density measures. Moreover, this method is an algorithm with a low linear time complexity and a small memory requirement. It builds a good performing model with a small number of trees using small sub-samples of fixed size, regardless of the size of a data set.

Typical machine learning methods tend to work better when the patterns they try to learn are balanced, meaning the same amount of good and bad behaviors are present in the dataset.

How Isolation Forests Work The Isolation Forest algorithm isolates observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature. The logic argument goes: isolating anomaly observations is easier because only a few conditions are needed to separate those cases from the normal observations. On the other hand, isolating normal observations require more conditions. Therefore, an anomaly score can be calculated as the number of conditions required to separate a given observation.

The way that the algorithm constructs the separation is by first creating isolation trees, or random decision trees. Then, the score is calculated as the path length to isolate the observation.

Local Outlier Factor(LOF) Algorithm:
The LOF algorithm is an unsupervised outlier detection method which computes the local density deviation of a given data point with respect to its neighbors. It considers as outlier samples that have a substantially lower density than their neighbors. The number of neighbors considered, (parameter neighbors) is typically chosen 1) greater than the minimum number of objects a cluster has to contain, so that other objects can be local outliers relative to this cluster, and 2) smaller than the maximum number of close by objects that can potentially be local outliers. In practice, such information's are generally not available, and taking neighbors = 20 appears to work well in general.

Define the outlier detection methods

```
#Define the outlier detection methods

classifiers = {
    "Isolation Forest":IsolationForest(n_estimators=100, max_samples=len(X),
                                       contamination=outlier_fraction,random_state=state, verbose=0),
    "Local Outlier Factor":LocalOutlierFactor(n_neighbors=20, algorithm='auto',
                                              leaf_size=30, metric='minkowski',
                                              p=2, metric_params=None, contamination=outlier_fraction),
    "Support Vector Machine":OneClassSVM(kernel='rbf', degree=3, gamma=0.1,nu=0.05,
                                         max_iter=-1, random_state=state)

}
```

Fit the model

```
#Fit the model

n_outliers = len(Fraud)
for i, (clf_name,clf) in enumerate(classifiers.items()):
    #Fit the data and tag outliers
    if clf_name == "Local Outlier Factor":
        y_pred = clf.fit_predict(X)
        scores_prediction = clf.negative_outlier_factor_
    elif clf_name == "Support Vector Machine":
        clf.fit(X)
        y_pred = clf.predict(X)
    else:
        clf.fit(X)
        scores_prediction = clf.decision_function(X)
        y_pred = clf.predict(X)
    #Reshape the prediction values to 0 for Valid transactions , 1 for Fraud transactions
    y_pred[y_pred == 1] = 0
    y_pred[y_pred == -1] = 1
    n_errors = (y_pred != Y).sum()
    # Run Classification Metrics
    print("{}: {}".format(clf_name,n_errors))
    print("Accuracy Score :")
    print(accuracy_score(Y,y_pred))
    print("Classification Report :")
    print(classification_report(Y,y_pred))
```

Isolation Forest: 69
Accuracy Score :
0.9975773322565921
Classification Report :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	28434
1	0.27	0.28	0.27	47
micro avg	1.00	1.00	1.00	28481
macro avg	0.63	0.64	0.64	28481
weighted avg	1.00	1.00	1.00	28481

Local Outlier Factor: 93
Accuracy Score :
0.9967346652154068
Classification Report :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	28434
1	0.02	0.02	0.02	47
micro avg	1.00	1.00	1.00	28481
macro avg	0.51	0.51	0.51	28481
weighted avg	1.00	1.00	1.00	28481

Support Vector Machine: 8411
Accuracy Score :
0.7046803131912504
Classification Report :

	precision	recall	f1-score	support
0	1.00	0.71	0.83	28434
1	0.00	0.34	0.00	47
micro avg	0.70	0.70	0.70	28481
macro avg	0.50	0.52	0.42	28481
weighted avg	1.00	0.70	0.83	28481

Observations :

Isolation Forest detected 69 errors versus Local Outlier Factor detecting 93 errors vs. SVM detecting 8411 errors

Isolation Forest has a 99.75% more accurate than LOF of 99.67% and SVM of 70.46

When comparing error precision & recall for 3 models , the Isolation Forest performed much better than the LOF as we can see that the detection of fraud cases is around 27 % versus LOF detection rate of just 2 % and SVM of 0

So overall Isolation Forest Method performed much better in determining the fraud cases which is around 30%.

We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense. We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases

The neural network we fitted on the oversampled dataset has lower recall than the previous neural network. However, the precision has also significantly improved.

Conclusion

When we are building a classifier, both precision and recall are important, but it all comes down to the task we are given. In this particular project, higher recall means we can detect more frauds which leads to a more safer platform, whereas higher precision means that we can correctly detect the fraud cases which leads to customer satisfaction (from avoiding their accounts getting blocked). Moreover, we can still improve on our oversampled dataset by removing outliers, and also fine tuning our neural network models.

I couldn’t use dataset of Sierra Leone, but in the further, since I have built a model, I am planning to use a real dataset from Cards Fraud in Sierra Leone.

MACHINE LEARNING GRADUATION ASSIGNMENT

Project

End of Session

DIVE INTO CODE

FODAY BANGURA