Naive Bayes (Gaussian) Credit Card Fraud Detecting Project

Nawal Obaid

9/12/2022

PROBLEM STATEMENT

- Credit card companies need to have the ability to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.
- Datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
- The data contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data.
- Input Features: V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning.
- Output: 1 in case of fraud and 0 otherwise.
- Link to the dataset: https://www.kaggle.com/mlg-ulb/creditcardfraud/home

Import the needed libraries

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Get the data

In [2]:
```
card = pd.read_csv('creditcard.csv')
```

In [3]:
```
card.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.01

1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.22
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.24
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.10
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.00

5 rows × 31 columns

In [4]:
```
card.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
```

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

Out[5]:

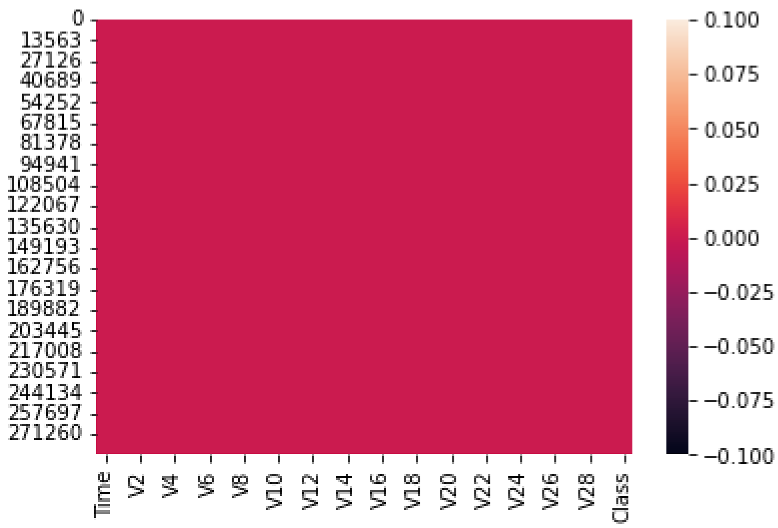
	Time	V1	V2	V3	V4	V5	V6
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	3.918649e-15	5.682686e-16	-8.761736e-15	2.811118e-15	-1.552103e-15	2.040130e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01

75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01

8 rows × 31 columns

In [6]:
```
sns.heatmap(card.isnull())
# no missing vaules
```

Out[6]:
```
<AxesSubplot:>
```



In [7]:
```
is_fraud = card[card['Class']==1]
not_fraud = card[card['Class']==0]
```

In [8]:
```
len(is_fraud)
```

Out[8]:
```
492
```

In [9]:
```
len(not_fraud)
# imbalanced dataset
```

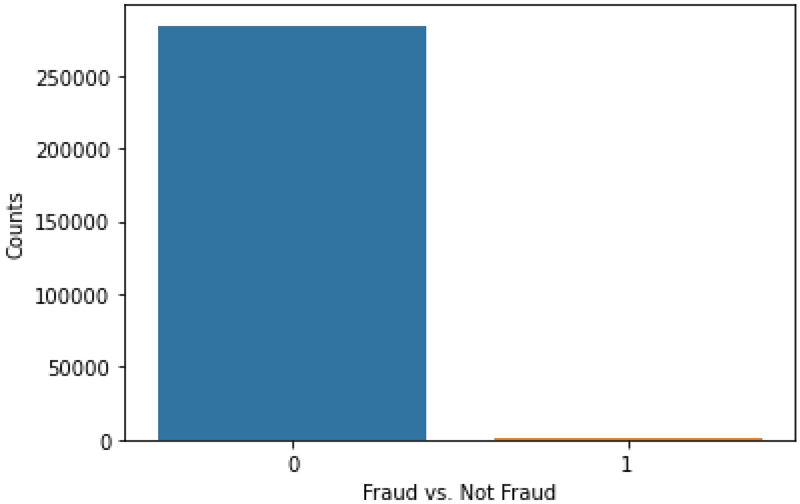
Out[9]:
```
284315
```

In [10]:
```
# let's get counts and percentages!
print('Number of spam emails:',len(is_fraud))
print('Percentage of spam emails:', len(is_fraud)/len(card) * 100,'%')
print('Number of not spam emails:', len(not_fraud))
print('Percentage of not spam emails:', len(not_fraud)/len(card)* 100,'%')
```

```
Number of spam emails: 492
Percentage of spam emails: 0.1727485630620034 %
Number of not spam emails: 284315
Percentage of not spam emails: 99.82725143693798 %
```

In [11]:
```
c = sns.countplot(data = card, x = card['Class'])
# spam is 1, not-spam or ham is 0
c.set_xlabel('Fraud vs. Not Fraud')
c.set_ylabel('Counts')
```

Out[11]:
```
Text(0, 0.5, 'Counts')
```



In [12]:

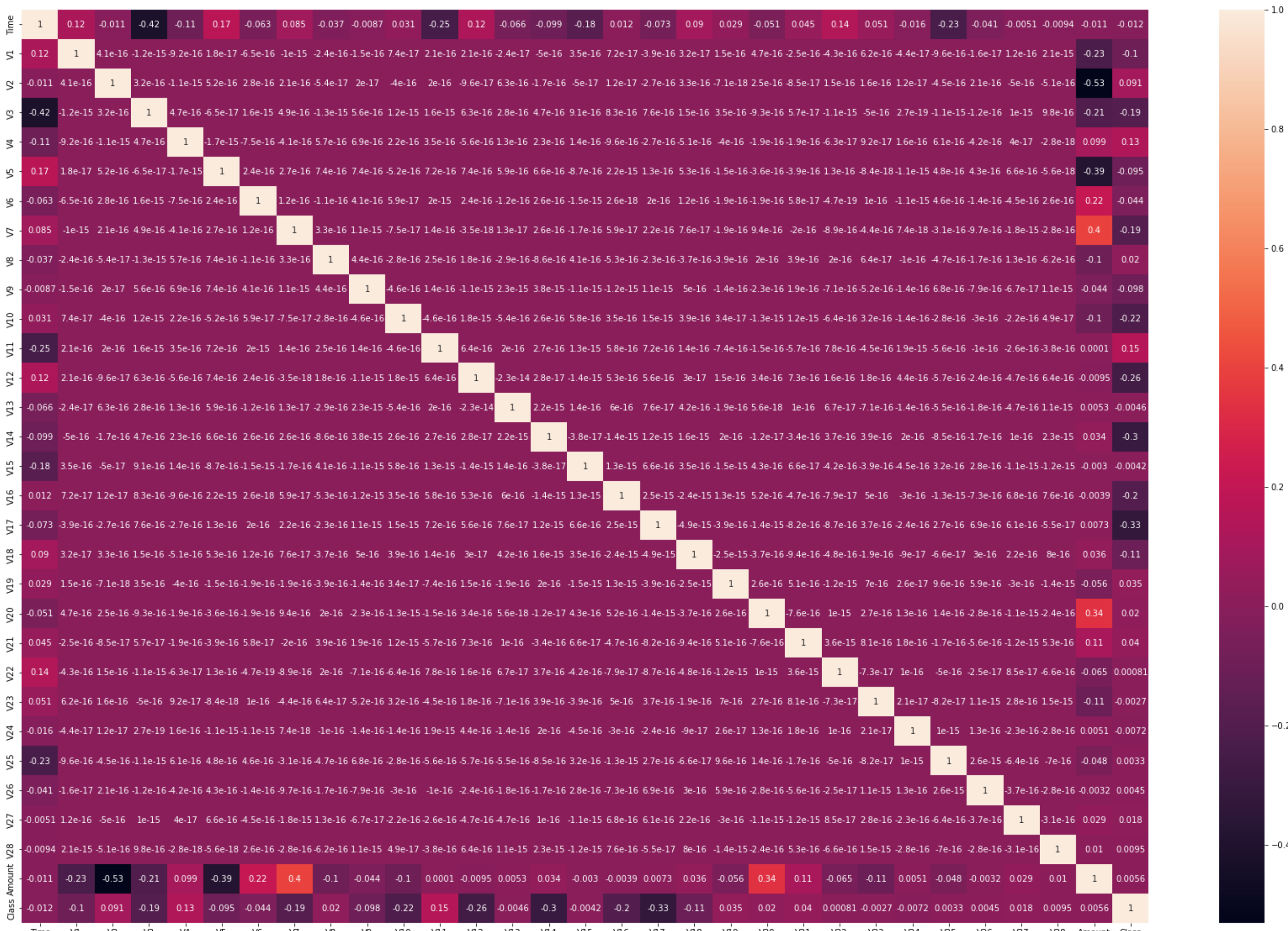
```
corr = card.corr()
```

In [13]:

```
plt.figure(figsize = (30,20))
sns.heatmap(corr, annot = True)
# it seems there is no correlection among columns in this dataset since these columns ar
```

Out[13]:

```
<AxesSubplot:>
```



Feature Engineering

In [14]:

```
# Feature scale/normalize the "Amount" column
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
card['Amount_Norm'] = sc.fit_transform(card['Amount'].values.reshape(-1,1))
```

In [15]:
```
card.head()
```

Out[15]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	0.27
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.63
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.77
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	0.00
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	0.79

5 rows × 32 columns

In [16]:
```
# we do not need the 'Amount' column anymore so we can drop it
card.drop('Amount' , axis = 1, inplace = True)
```

In [17]:
```
card.head()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.01
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.22
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.24
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.10
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.00

5 rows × 31 columns

Train Test Split

In [18]:
```
X = card.drop('Class', axis = 1)
y = card['Class']
```

In [19]:
```
from sklearn.model_selection import train_test_split
```

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [21]:
```
X_train.shape
```

Out[21]:
```
(227845, 30)
```

In [22]:
```
X_test.shape
```

Out[22]:
```
(56962, 30)
```

In [23]:
```
y_train.shape
```

Out[23]:
```
(227845,)
```

In [24]:
```
y_test.shape
```

Out[24]:
```
(56962,)
```

In [25]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
```

In [26]:
```
classifier.fit(X_train, y_train)
```

Out[26]:
```
▾ GaussianNB
GaussianNB()
```

In [27]:
```
predictions = classifier.predict(X_test)
```

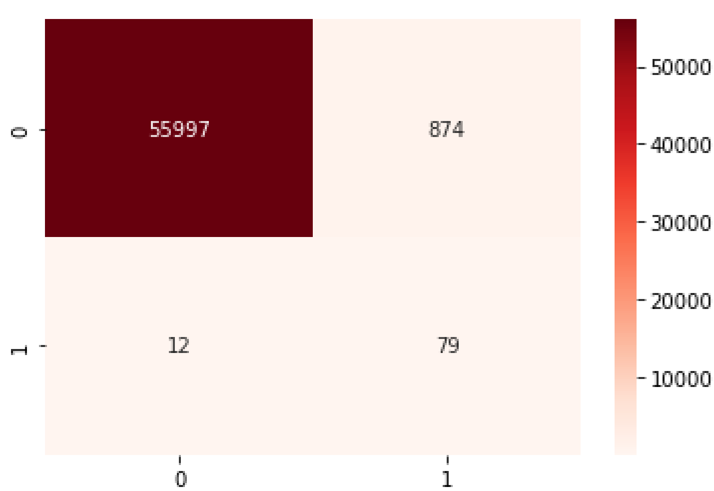
In [28]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [54]:
```
print('*************Classification Report*************************')
print(classification_report(y_test, predictions))
print('***************Confusion Matrix***************************')
print(confusion_matrix(y_test, predictions))
```

```
*************Classification Report*************************
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56871
           1       0.08      0.87      0.15        91

    accuracy                           0.98     56962
   macro avg       0.54      0.93      0.57     56962
weighted avg       1.00      0.98      0.99     56962

***************Confusion Matrix***************************
[[55997   874]
 [   12    79]]
```

In [55]:
```
cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot = True, fmt = "d", cmap ='Reds')
```

Out[55]:
```
<AxesSubplot:>
```



The model misclassified about 401 cases (32 of Error Type II), a big reason is that the dataset is originally imbalanced!!

Improve the model

In [31]:
```
# we can drop the features we are not interested in anymore since they do not make a gre

X = card.drop(['Class','Time','V8', 'V13', 'V15', 'V20', 'V22', 'V23', 'V24','V25', 'V26
y = card['Class']
```

In [40]:
```
# train and test the model again based on the new X
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

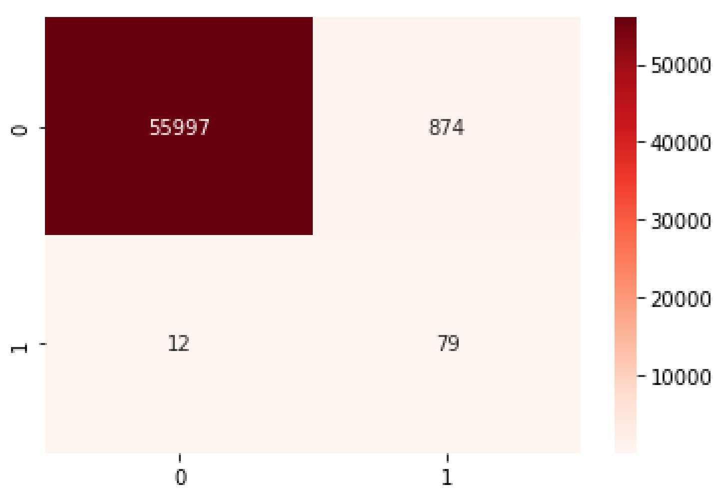
In [45]:
```
GNBclassifier = GaussianNB()
GNBclassifier.fit(X_train, y_train)
predictions = GNBclassifier.predict(X_test)
```

In [48]:
```
print('*************Classification Report************************')
print(classification_report(y_test, predictions))
print('***************Confusion Matrix***************************')
print(confusion_matrix(y_test, predictions))
```

```
*************Classification Report************************
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     56871
           1       0.08      0.87      0.15        91

    accuracy                           0.98     56962
   macro avg       0.54      0.93      0.57     56962
weighted avg       1.00      0.98      0.99     56962

***************Confusion Matrix***************************
[[55997   874]
 [   12    79]]
```

In [56]:
```
cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot = True, fmt = "d", cmap ='Reds')
# The model misclassified 12 cases  of Type II which is kind of improvement
```

Out[56]: <AxesSubplot:>



In [62]:
```
print('Number of fraud cases/points in the testing dataset = ', sum(y_test), ' the model
```

```
Number of fraud cases/points in the testing dataset =  91  the model has misclassified 1
2 cases
```