A Project on

Credit Card Fraud Prediction

By

Shweta Sanjay Suryawanshi

May, 2022

Contents

1 Introduction **1**

2 Dataset **2**

3 Tasks **3**

3.1 find the variables that are most useful in prediction 3

3.2 Implementing PCA 8

3.3 Draft the classification models 12

3.3.1 Logistic Regression 12

3.3.2 Support Vector Machine 14

3.3.3 Decision Tree Classification 15

Chapter 1

Introduction

Fraud Detection Using Machine Learning deploys a machine learning (ML) model and an example dataset of credit card transactions to train the model to recognize fraud patterns. Machine Learning refers to the process of training a computer program to build a statistical model based on data. The goal of machine learning (ML) is to turn data and identify the key patterns out of data or to get key insights.

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Tasks:

1. Out of all the 30 independent input variables, find the variables that are most useful in predicting 'Class'.

2. Try implementing PCA (whether it is useful or not) to reduce no. of independent variables.

3. Draft the following classification models to predict 'Class': Logistic regression, SVC, Decision Tree Classifier.

Chapter 2

Dataset

The dataset 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. The fraud detection problem may be framed as a classification problem, of which the goal is to predict the discrete label 0 or 1 where 0 generally suggest that a transaction is non-fraud and 1 suggest that the transaction seems to be fraud.

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-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

We will going to create the notebook which will show us how to use the machine learning techniques on the banking dataset and find out the fraud detection methods, with the help of model.

Chapter 3

Tasks

3.1 find the variables that are most useful in prediction

First we will going to import some librarys then read the "creditcard.csv" file to see what it contains and do some operation on data to get basic information about data like shape,dtypes,info etc., After that we will find correlation for each column and on the basis of highly correlated column we decide target variable.

Let's import librarys

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
%matplotlib inline
```

Read the “creditcard.csv” file to see what it contains.

```
data = pd.read_csv('creditcard.csv')
data
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798
...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643

284807 rows × 31 columns

You can see from the output that the “creditcard.csv” file contains the V1,V2,...V28,Time,Amount and Class attributes.

Here we use shape for representing the dimension of dataset.

```
data.shape
```

```
(284807, 31)
```

```
data.keys()
```

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

Now,using describe() method to view some basic statistical details for rating like count, mean, standard deviation,etc.,

```
data.describe()
```

V9	...	V21	V22	V23	V24	V25	V26	V27	V28	
70e+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	284807
)24e-15	...	1.537294e-16	7.959909e-16	5.367590e-16	4.458112e-15	1.453003e-15	1.699104e-15	-3.660161e-16	-1.206049e-16	8E
32e+00	...	7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	5.212781e-01	4.822270e-01	4.036325e-01	3.300833e-01	25(
07e+01	...	-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	-1.029540e+01	-2.604551e+00	-2.256568e+01	-1.543008e+01	(
)76e-01	...	-2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	-3.171451e-01	-3.269839e-01	-7.083953e-02	-5.295979e-02	5
373e-02	...	-2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	1.659350e-02	-5.213911e-02	1.342146e-03	1.124383e-02	22
390e-01	...	1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	3.507156e-01	2.409522e-01	9.104512e-02	7.827995e-02	77
99e+01	...	2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691

Here, we used dtypes to get data type for each columns. and then called isnull() method for checking any null entry in the dataset. but, here all entries are non-null so, dataset is fullfil.

```
data.dtypes.head()
```

```
Time    float64
V1      float64
V2      float64
V3      float64
V4      float64
dtype: object
```

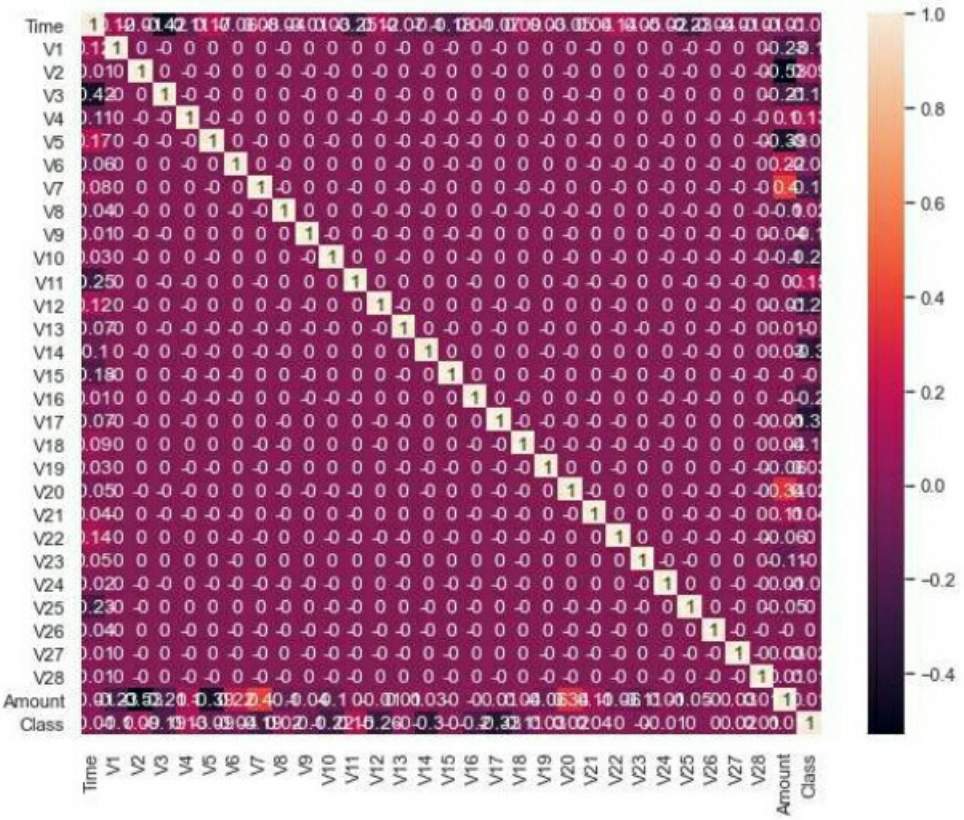
```
data.isnull().sum().head()
```

```
Time    0
V1      0
V2      0
V3      0
V4      0
dtype: int64
```

Call corr() to find correlation for each variable

```
sns.set(rc={'figure.figsize':(10,8)})
cm = data.corr().round(2)
sns.heatmap(data=cm, annot = True)
```

```
<AxesSubplot:>
```



```
corr = data.corr()
corr.head()
```

'6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28
e-)2	8.471437e- 02	-3.694943e- 02	-8.660434e- 03	...	4.473573e- 02	1.440591e- 01	5.114236e- 02	-1.618187e- 02	-2.330828e- 01	-4.140710e- 02	-5.134591e- 03	-9.412688e- 03
e- 6	1.991550e- 15	-9.490675e- 17	2.169581e- 16	...	-1.755072e- 16	7.477367e- 17	9.808705e- 16	7.354269e- 17	-9.805358e- 16	-8.621897e- 17	3.208233e- 17	9.820892e- 16
e- 6	3.966486e- 16	-4.413984e- 17	-5.728718e- 17	...	8.444409e- 17	2.500830e- 16	1.059562e- 16	-8.142354e- 18	-4.261894e- 17	2.601622e- 16	-4.478472e- 16	-3.676415e- 16
e- 5	2.168574e- 15	3.433113e- 16	-4.233770e- 16	...	-2.971969e- 17	4.648259e- 16	2.115206e- 17	-9.351637e- 17	4.771164e- 16	6.521501e- 16	6.239832e- 16	7.726948e- 16
e- 6	1.556330e- 16	5.195643e- 16	3.859585e- 16	...	-9.976950e- 17	2.099922e- 16	6.002528e- 17	2.229738e- 16	5.394585e- 16	-6.179751e- 16	-6.403423e- 17	-5.863664e- 17

So, as compare to other variable corelation of class variable is high. Therefor we take "Class" variable as a target

```
classes=data['Class'].value_counts()
normal_share=classes[0]/data['Class'].count()*100
fraud_share=classes[1]/data['Class'].count()*100
print('Normal percentage',normal_share)
print('Fraud percentage',fraud_share)
```

```
Normal percentage 99.82725143693798
Fraud percentage 0.1727485630620034
```

Let's separate data in x and y

```
x = data.copy().drop('Class', axis = 1)
y = data.copy()['Class']
```

3.2 Implementing PCA

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components.

- It is used to reduce dimensionality of data to increase performance and remove all those data which are not important.
- When we have multiple features we cannot plot our model but using principal components we can plot out model data easily.

StandardScaler :

StandardScaler is an important technique that is mainly performed as a preprocessing step before many machine learning models, in order to standardize the range of functionality of the input dataset. StandardScaler is used to resize the distribution of values so that the mean of the observed values is 0 and the standard deviation is 1.

After StandardScaler we done PCA to reduce the dimension of dataset.

```
: x = StandardScaler().fit_transform(x)
```

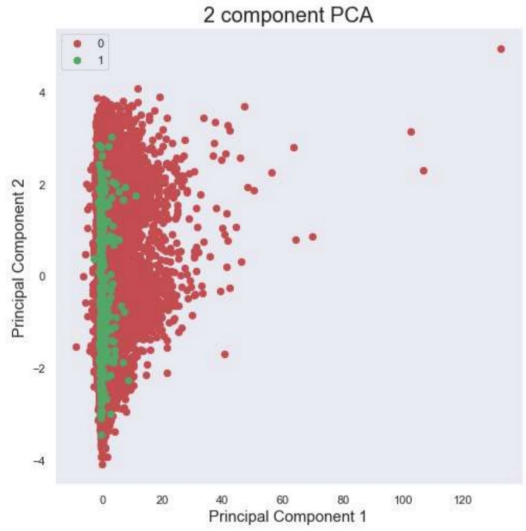
```
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
principalDf.head()
```

	principal component 1	principal component 2
0	0.400244	-2.561714
1	-0.388551	-2.079959
2	1.848546	-2.539917
3	0.311768	-1.775098
4	-0.003214	-1.491605

```
finalDf = pd.concat([principalDf, data[['Class']]], axis = 1)
finalDf.head()
```

	principal component 1	principal component 2	Class
0	0.400244	-2.561714	0
1	-0.388551	-2.079959	0
2	1.848546	-2.539917	0
3	0.311768	-1.775098	0
4	-0.003214	-1.491605	0

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = [0,1]
colors = ['r', 'g']
for Class, color in zip(targets,colors):
    indicesToKeep = finalDf['Class'] == Class
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 40)
ax.legend(targets)
ax.grid()
```




```
pca.explained_variance_ratio_
```

```
array([0.06527159, 0.05610597])
```

Uses of PCA:

- It is used to find inter-relation between variables in the data.
- It is used to interpret and visualize data.
- As the number of variables are decreasing it makes further analysis simpler.
- It's often used to visualize genetic distance and relatedness between populations.

3.3 Draft the classification models

3.3.1 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. Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.

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.

```
: feature = finalDf[['principal component 1', 'principal component 2']]

  target = finalDf['Class']

: X_train,X_test,Y_train,Y_test=train_test_split(feature,target,test_size= 0.2,random_state=4)
```

```
: model = LogisticRegression(max_iter=10e6)
  model.fit(X_train, Y_train)

: LogisticRegression(max_iter=10000000.0)

: Y_pred = model.predict(X_test)

: print('Accuracy:', metrics.accuracy_score(Y_test, Y_pred))

  Accuracy: 0.9982268881008391
```

```
#classification_report
print(metrics.classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.00      0.00      0.00       101

    accuracy                           1.00     56962
   macro avg       0.50      0.50      0.50     56962
weighted avg       1.00      1.00      1.00     56962
```

Accuracy is a metric used in classification problems used to tell the percentage of accurate predictions. We calculate it by dividing the number of correct predictions by the total number of predictions.

3.3.2 Support Vector Machine

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

```
model = svm.SVC(kernel ='linear')
model.fit(X_train,Y_train)
```

```
SVC(kernel='linear')
```

```
Y_pred=model.predict(X_test)
```

```
print('Accuracy: ', metrics.accuracy_score(Y_test, Y_pred)*100,'%')
```

SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. We carry out plotting in the n-dimensional space. Value of each feature is also the value of the specific coordinate. Then, we find the ideal hyperplane that differentiates between the two classes.

```
#classification_report
print(metrics.classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.00      0.00      0.00       101

    accuracy                           1.00     56962
   macro avg       0.50      0.50      0.50     56962
weighted avg       1.00      1.00      1.00     56962
```

3.3.3 Decision Tree Classification

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. The decisions or the test are performed on the basis of features of the given dataset.

It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. 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. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm. A decision tree simply asks a question, and based on the

answer (Yes/No), it further split the tree into subtrees.

```
clf = DecisionTreeClassifier()
clf.fit(X_train, Y_train)
```

```
DecisionTreeClassifier()
```

```
Y_pred = clf.predict(X_test)
```

```
print('Accuracy: ', metrics.accuracy_score(Y_test, Y_pred))
```

```
Accuracy:  0.9968224430321969
```

```
#classification_report
print(metrics.classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56861
           1       0.07      0.08      0.07       101

    accuracy                           1.00     56962
   macro avg       0.53      0.54      0.54     56962
weighted avg       1.00      1.00      1.00     56962
```

Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Conclusion: Precision and recall value of Decision tree classifier is high as compair to other two classification models. That means Decision tree classifier gives accurate prediction. So Decision tree classifier is best for this dataset

■✠■✠■