MACHINE LEARNING DIGITAL ASSIGNMENT 2

Name: Om Ashish Mishra

Registration Number: 16BCE0789

Slot: F2

9. Consider a classification problem in Banking domain and implement it using Multi-Layer Perceptron and Support vector Machines.

I. TITLE/HEADER

Banking Domain

II. INTRODUCTION

Ever since starting my journey into data science, I have been thinking about ways to use data science for good while generating value at the same time. Thus, when I came across this data set on Kaggle dealing with credit card fraud detection, I was immediately hooked. The data set has 31 features, 28 of which have been anonymized and are labeled V1 through V28. The remaining three features are the time and the amount of the transaction as well as whether that transaction was fraudulent or not. Before it was uploaded to Kaggle, the anonymized variables had been modified in the form of a PCA (Principal Component Analysis). Furthermore, there were no missing values in the data set. Equipped with this basic description of the data, let's jump into some exploratory data analysis.

III. IMPLEMENTATION

a. Source Code

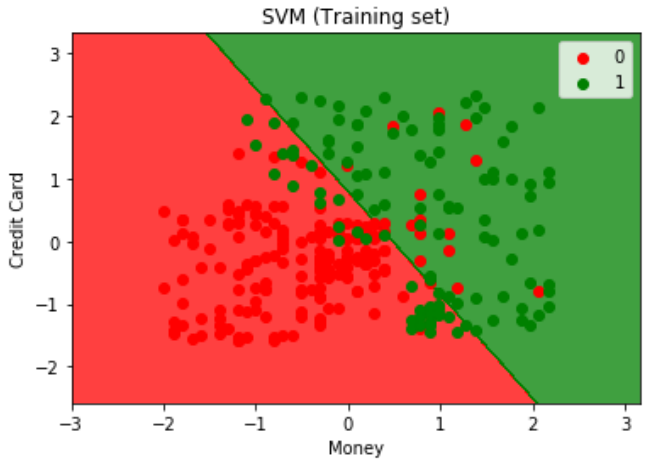
```
# Support Vector Machine (SVM)

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

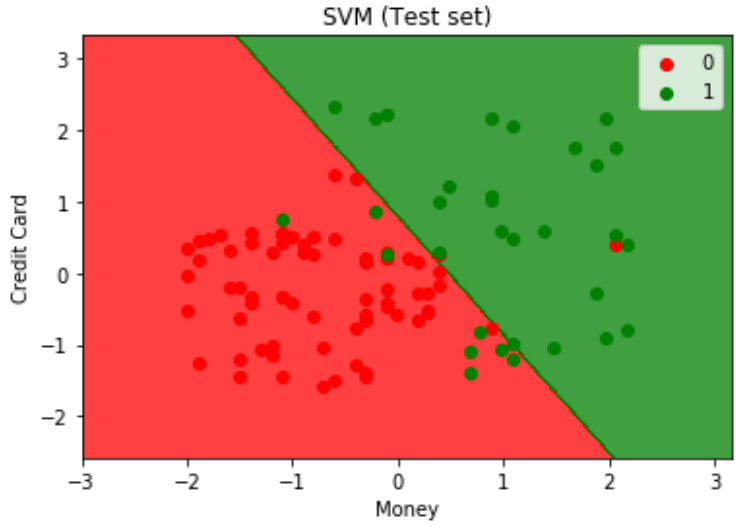
```
# Importing the dataset
dataset = pd.read_csv('Credit_Card_Fraud.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting SVM to the Training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step =
0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Training set)')
```

```
plt.xlabel('Money')
plt.ylabel('Credit Card')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step =
0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(),
X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Test set)')
plt.xlabel('Money')
plt.ylabel('Credit Card')
plt.legend()
plt.show()

# -*- coding: utf-8 -*-
"""
Created on Tue Feb 12 10:35:28 2019

@author: OM MISHRA
"""
#Multi Layer Perceptron

import numpy as np

#Input array of People Bank Account
X=np.array([[1,0,1,0],[1,0,1,1],[0,1,0,1]])

#Output
y=np.array([[1],[1],[0]])

#Sigmoid Function
```

```
def sigmoid (x):
    return (1/(1 + np.exp(-x)))

#Derivative of Sigmoid Function
def derivatives_sigmoid(x):
    return x * (1 - x)

#Variable initialization
epoch=5000 #Setting training iterations
lr=0.1 #Setting learning rate
inputlayer_neurons = X.shape[1] #number of features in data set
hiddenlayer_neurons = 3 #number of hidden layers neurons
output_neurons = 1 #number of neurons at output layer

#weight and bias initialization
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))

for i in range(epoch):

#Forward Propogation
    hidden_layer_input1=np.dot(X,wh)
    hidden_layer_input=hidden_layer_input1 + bh
    hiddenlayer_activations = sigmoid(hidden_layer_input)
    output_layer_input1=np.dot(hiddenlayer_activations,wout)
    output_layer_input= output_layer_input1+ bout
    output = sigmoid(output_layer_input)

#Backpropagation
    E = y-output
    slope_output_layer = derivatives_sigmoid(output)
    slope_hidden_layer = derivatives_sigmoid(hiddenlayer_activations)
    d_output = E * slope_output_layer
    Error_at_hidden_layer = d_output.dot(wout.T)
    d_hiddenlayer = Error_at_hidden_layer * slope_hidden_layer
    wout += hiddenlayer_activations.T.dot(d_output) *lr
    bout += np.sum(d_output, axis=0,keepdims=True) *lr
    wh += X.T.dot(d_hiddenlayer) *lr
    bh += np.sum(d_hiddenlayer, axis=0,keepdims=True) *lr

print (output)
```

b. Snapshots






```
In [4]: runfile('D:/6Sixth Semester/mulyi layer perceptron.py', wdir='D:/6Sixth
Semester')
[[0.97864111]
 [0.96948908]
 [0.04154263]]
```

c. Conclusion

```
In [4]: cm
Out[4]:
array([[66,  2],
       [ 8, 24]], dtype=int64)
```

```
In [5]: from sklearn.metrics import classification_report
   ...: print(classification_report(y_test,y_pred))
              precision    recall  f1-score   support

           0       0.89      0.97      0.93        68
           1       0.92      0.75      0.83        32

   micro avg       0.90      0.90      0.90       100
   macro avg       0.91      0.86      0.88       100
weighted avg       0.90      0.90      0.90       100
```

IV. REFERENCES

1. http://www.crraoaimscs.org/lecture_notes/VR_Lecture_2.pdf
2. https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es1999-463.pdf
3. https://www.researchgate.net/publication/270345247_Selection_of_Support_Vector_Machines_based_classifiers_for_credit_risk_domain
4. https://dl.acm.org/citation.cfm?id=1465151