**Project – Exam1 Source code**

**CSEE 5590/CS490: Python and Deep Learning**

# **Group 8:**

Medavarapu Sumanth: 29

Roshna Toke: 50

Salkuti Sandeep Reddy: 41

Patchala Sai Chand: 34

Ramineni Pardha Saradhi: 38

### **Introduction:**

This project is about five different questions which we must solve using given methods and importing respective libraries and methods.

### **Objectives:**

The objective of this project is it will cover all the concepts which we have learned in the class. To learn different algorithms and its advantages, disadvantages, and its accuracy. We can compare best algorithm which suits to our dataset.

**Source code :**

**1)**

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  
dataset = pd.read\_csv('creditcard.csv',sep=',')  
dataset.head()  
dataset.info() #checking any null val  
dataset.isnull().values.any()  
## Get the Fraud and the normal dataset  
frauddata = dataset[dataset['Class']==1]  
normaldata = dataset[dataset['Class']==0]  
print(frauddata.shape,normaldata.shape)  
frauddata.Amount.describe()  
normaldata.Amount.describe()  
## Taking only some sample of data as dataset is too large  
data= dataset.sample(frac = 0.1,random\_state=1)  
data.shape  
#number of fraud and valid transactions in new dataset  
Fd = data[data['Class']==1]  
Vd = data[data['Class']==0]  
outlier\_fraction = len(Fd)/float(len(Vd))  
print(Fd.shape,Vd.shape)  
print(outlier\_fraction)  
print("Fraud Cases : {}".format(len(Fd)))  
print("Valid Cases : {}".format(len(Vd)))  
#independent and Dependent Features  
# Define a random state  
state = np.random.RandomState(42)  
X = data.iloc[:,:-1]  
y = data.iloc[:,-1]  
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)  
from sklearn.model\_selection import train\_test\_split  
  
train, test = train\_test\_split(data, test\_size=.2)  
##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, nu=outlier\_fraction, gamma=0.1,  
 max\_iter=-1)  
}  
from sklearn.svm import SVC # "Support Vector Classifier"   
n\_outliers = len(Fd)  
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))

**2)**

import pandas as pd  
from sklearn.preprocessing import StandardScaler  
from sklearn.cluster import KMeans  
import matplotlib.pyplot as plt  
import numpy as np  
import seaborn as sns  
  
sns.set(style="white", color\_codes=True)  
import warnings  
warnings.filterwarnings("ignore")  
  
df = pd.read\_csv('Customers.csv')  
print('\n', df.head())  
  
df.drop(["CustomerID"], axis=1, inplace=True)  
  
# Checking Null vaues  
nulls = pd.DataFrame(df.isnull().sum().sort\_values(ascending=False)[:25])  
nulls.columns = ['Null Count']  
print('\n', nulls)  
  
# grouping based on Spending score  
ss1\_20 = df["Spending Score (1-100)"][(df["Spending Score (1-100)"] >= 1) & (df["Spending Score (1-100)"] <= 20)]  
ss21\_40 = df["Spending Score (1-100)"][(df["Spending Score (1-100)"] >= 21) & (df["Spending Score (1-100)"] <= 40)]  
ss41\_60 = df["Spending Score (1-100)"][(df["Spending Score (1-100)"] >= 41) & (df["Spending Score (1-100)"] <= 60)]  
ss61\_80 = df["Spending Score (1-100)"][(df["Spending Score (1-100)"] >= 61) & (df["Spending Score (1-100)"] <= 80)]  
ss81\_100 = df["Spending Score (1-100)"][(df["Spending Score (1-100)"] >= 81) & (df["Spending Score (1-100)"] <= 100)]  
  
ssx = ["1-20", "21-40", "41-60", "61-80", "81-100"]  
ssy = [len(ss1\_20.values), len(ss21\_40.values), len(ss41\_60.values), len(ss61\_80.values), len(ss81\_100.values)]  
plt.figure(figsize=(15,6))  
sns.barplot(x=ssx, y=ssy, palette="nipy\_spectral\_r")  
plt.title("Spending Scores")  
plt.xlabel("Score")  
plt.ylabel("Number of Customer Having the Score")  
plt.show()  
  
# grouping based on Annual Income  
ai0\_30 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 0) & (df["Annual Income (k$)"] <= 30)]  
ai31\_60 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 31) & (df["Annual Income (k$)"] <= 60)]  
ai61\_90 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 61) & (df["Annual Income (k$)"] <= 90)]  
ai91\_120 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 91) & (df["Annual Income (k$)"] <= 120)]  
ai121\_150 = df["Annual Income (k$)"][(df["Annual Income (k$)"] >= 121) & (df["Annual Income (k$)"] <= 150)]  
  
aix = ["$ 0 - 30,000", "$ 30,001 - 60,000", "$ 60,001 - 90,000", "$ 90,001 - 120,000", "$ 120,001 - 150,000"]  
aiy = [len(ai0\_30.values), len(ai31\_60.values), len(ai61\_90.values), len(ai91\_120.values), len(ai121\_150.values)]  
plt.figure(figsize=(15,6))  
sns.barplot(x=aix, y=aiy, palette="Set2")  
plt.title("Annual Incomes")  
plt.xlabel("Income")  
plt.ylabel("Number of Customer")  
plt.show()  
  
# converting Male=0 & Female=1  
p = df.iloc[:,:1]  
def tran\_mathscore(p):  
 if p == 'Male':  
 return 0  
 if p == 'Female':  
 return 1  
  
# K-means Clustering  
wcss = []  
for k in range(1,11):  
 kmeans = KMeans(n\_clusters=k, init="k-means++")  
 df['Gender'] = df['Gender'].apply(tran\_mathscore)  
 kmeans.fit(df.iloc[:,1:])  
 wcss.append(kmeans.inertia\_)  
plt.figure(figsize=(12,6))  
plt.grid()  
plt.plot(range(1,11),wcss, linewidth=2, color="red", marker ="8")  
plt.title('the elbow method')  
plt.xlabel("No. of Clusters : K Value")  
plt.xticks(np.arange(1,11,1))  
plt.ylabel("WCSS")  
plt.show()  
  
# standardization  
scaler = StandardScaler()  
  
scaler.fit(df.iloc[:,1:])  
x\_scaler = scaler.transform(df.iloc[:,1:])  
  
# KMeans after standarization and Clustering, K=5  
km = KMeans(n\_clusters=5)  
km.fit(x\_scaler)  
y\_cluster\_kmeans= km.predict(x\_scaler)  
from sklearn import metrics  
score = metrics.silhouette\_score(x\_scaler, y\_cluster\_kmeans)  
print("\n Silhouette score is: ", score)  
  
  
# Visualization - Scatter the plots with seaborn  
  
# Box plot of spending score and annual income to better visualize the distribution range.  
plt.figure(figsize=(15,6))  
plt.subplot(1,2,1)  
sns.boxplot(y=df["Spending Score (1-100)"], color="red")  
plt.subplot(1,2,2)  
sns.boxplot(y=df["Annual Income (k$)"])  
plt.show()  
  
  
km = KMeans(n\_clusters=5)  
clusters = km.fit\_predict(df.iloc[:, 1:])  
df["label"] = clusters  
import matplotlib.pyplot as plt  
  
fig = plt.figure(figsize=(20, 10))  
ax = fig.add\_subplot(111, projection='3d')  
ax.scatter(df.Age[df.label == 0], df["Annual Income (k$)"][df.label == 0], df["Spending Score (1-100)"][df.label == 0], c='blue', s=60)  
ax.scatter(df.Age[df.label == 1], df["Annual Income (k$)"][df.label == 1], df["Spending Score (1-100)"][df.label == 1], c='red', s=60)  
ax.scatter(df.Age[df.label == 2], df["Annual Income (k$)"][df.label == 2], df["Spending Score (1-100)"][df.label == 2], c='green', s=60)  
ax.scatter(df.Age[df.label == 3], df["Annual Income (k$)"][df.label == 3], df["Spending Score (1-100)"][df.label == 3], c='orange', s=60)  
ax.scatter(df.Age[df.label == 4], df["Annual Income (k$)"][df.label == 4], df["Spending Score (1-100)"][df.label == 4], c='purple', s=60)  
ax.view\_init(30, 185)  
plt.xlabel("Age")  
plt.ylabel("Annual Income (k$)")  
ax.set\_zlabel('Spending Score (1-100)')  
plt.show()

**3)**

import warnings  
warnings.filterwarnings('ignore')  
import numpy as np  
import pandas as pd  
import sklearn  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
from sklearn.linear\_model import LinearRegression  
from sklearn import preprocessing  
data=pd.read\_csv("weather.csv") #Loading dataset  
data.head()  
data.shape  
data.isnull().any() #checking for null values in all columns  
data['Precip Type'].value\_counts()  
  
data.loc[data['Precip Type'].isnull(),'Precip Type']='rain' #replacing null value in column with rain  
data.isnull().any()  
  
data.loc[data['Precip Type']=='rain','Precip Type']=1  
data.loc[data['Precip Type']=='snow','Precip Type']=0  
data.drop(['Summary', 'Daily Summary','Formatted Date'], axis=1, inplace=True)  
  
X=data.drop(['Temperature (C)'],axis=1) #taking X and y values which is dependent and independent features  
y=data['Temperature (C)']  
  
X\_train,X\_test,y\_train,y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1) #splitting training and testing data  
#linear regression model  
model=LinearRegression()  
model.fit(X\_train,y\_train)  
pred=model.predict(X\_test)  
print(np.mean((pred-y\_test)\*\*2)) #calculating error using linear regression  
pd.DataFrame({'actualvalue':y\_test,  
 'predictedvalue':pred,  
 'difference':(y\_test-pred)})  
  
#polynomial regression model  
from sklearn.preprocessing import PolynomialFeatures  
poly=PolynomialFeatures(degree=4) #calculating error using polynomial  
tran=poly.fit\_transform(X\_train)  
poly.fit(tran,y\_train)  
model=LinearRegression()  
model.fit(tran,y\_train)  
pred=model.predict(poly.fit\_transform(X\_test))  
print(np.mean((pred-y\_test)\*\*2))  
pd.DataFrame({'actualvalue':y\_test,  
 'predictedvalue':pred,  
 'difference':(y\_test-pred)})  
  
#Random Forest Regression model  
from sklearn.ensemble import RandomForestRegressor  
reg1=RandomForestRegressor(max\_depth=50,random\_state=42,n\_estimators=100)  
reg1.fit(X\_train,y\_train)  
pred1=reg1.predict(X\_test)  
print(np.mean(pred1-y\_test)\*\*2) #calculating error rate using mean value of predicted minus test value  
pd.DataFrame({'actualvalue':y\_test,  
 'predictedvalue':pred1,  
 'difference':(y\_test-pred1)})  
  
accuracy = reg1.score(X\_train,y\_train) #As randomforest yields lesser errorrate using it as our model  
print(accuracy)  
  
accuracy = reg1.score(X\_test,y\_test)  
print(accuracy)

**4)**

#imported all libraries required  
import pandas as pd  
import re  
import nltk  
from nltk.corpus import stopwords  
from nltk.stem import PorterStemmer  
from nltk.stem.wordnet import WordNetLemmatizer  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.feature\_extraction.text import TfidfVectorizer,TfidfTransformer  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.metrics import confusion\_matrix,accuracy\_score  
  
#reading dataset or csv file  
data=pd.read\_csv("spam.csv",encoding='latin-1') #reading dataset  
#data cleaning and preprocessing  
ps=PorterStemmer() # using porter stemmer for base words  
wordnet\_lemmatizer = WordNetLemmatizer()  
ls=[]  
for i in range(0,len(data)):  
 newdata=re.sub('[^a-zA-Z]',' ',data['Text'][i]) #removing all unnecessary data except a-zA-Z caharcters  
 newdata=newdata.lower() #lowering all words to small alphabets  
 newdata=newdata.split() #splitting each word  
  
 #newdata=[ps.stem(word) for word in newdata if not word in stopwords.words('english')]  
 newdata = [wordnet\_lemmatizer.lemmatize(word) for word in newdata if not word in stopwords.words('english')]  
 newdata=''.join(newdata) #joining words after stemming to newdata  
 ls.append(newdata)  
#print("after stemming: \n",ls)  
#creating bag of words model  
countvec= CountVectorizer(max\_features=5000) #document matrix with top features about 5000 are used  
X=countvec.fit\_transform(ls).toarray()  
tfidf\_Vect = TfidfVectorizer()  
X\_tfidf = tfidf\_Vect.fit\_transform(ls).toarray()  
#print(countvec)  
#print(X)  
y=pd.get\_dummies(data['Class'])  
#print(y)  
y=y.iloc[:,1].values  
#train\_test\_split  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.20,random\_state=42)  
X\_train,X\_test,y\_train,y\_test=train\_test\_split(X\_tfidf,y,test\_size=0.20,random\_state=42)  
#training model using NaiveBayes  
NB=MultinomialNB().fit(X\_train,y\_train) #using naivebayes classification technique  
pred=NB.predict(X\_test)  
conf=confusion\_matrix(y\_test,pred)  
print("confusion matrix is:",conf)  
score=accuracy\_score(y\_test,pred) #calculating accuracy score on test and predicted data  
print("accuracy score is:",score)

**5)**

import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
# ---------------------5. a) Data Analysis --------------------------------#  
# Loading the data from adult Dataset  
adultData = pd.read\_csv('adult.csv')  
adultData.drop("income",axis=1)  
incomeLabel = adultData['income']  
  
# Finding the Values which are missing  
print("Number of missing values:\n", format(adultData.isnull().sum()))  
  
# Eliminate NAN values  
adultData.dropna(axis = 0, inplace= True)  
  
# Encode the categorial features  
trainData = pd.get\_dummies(adultData)  
  
from sklearn.model\_selection import train\_test\_split  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(trainData,incomeLabel)  
  
  
# ---------------------5. b) Applying Naive Bayes, SVM and KNN--------------------------------#  
  
# ----------- creating the Gaussian Naive Bayes object ---------------#  
from sklearn.naive\_bayes import GaussianNB  
gnb = GaussianNB()  
  
# Training the Model  
gnb.fit(X\_train,y\_train)  
trainScore = gnb.score(X\_train,y\_train)  
  
# Predicting the Output  
testScore = gnb.score(X\_test,y\_test)  
print(f'\n Gaussian Naive Bayes : Training score - {trainScore} and Test score (Accuracy) - {testScore}')  
  
  
# ----------- creating the KNN object -------------------------------#  
from sklearn.neighbors import KNeighborsClassifier  
knn = KNeighborsClassifier(n\_neighbors=5)  
  
# Training the Model  
knn.fit(X\_train,y\_train)  
trainScore = knn.score(X\_train,y\_train)  
  
# Predicting the Output  
testScore = knn.score(X\_test,y\_test)  
print(f'\n K Neighbors : Training score - {trainScore} and Test score (Accuracy) - {testScore}')  
  
  
# ----------- creating the SVM object -------------------------------#  
  
from sklearn import svm  
from sklearn.preprocessing import StandardScaler  
  
# Training Model  
scaler = StandardScaler()  
scaler.fit(trainData,incomeLabel)  
X\_train\_scaled = scaler.transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)  
# Linear SVM Kernel  
svc = svm.SVC(kernel='linear')  
  
# Training the Model  
svc.fit(X\_train\_scaled,y\_train)  
trainScore = svc.score(X\_train\_scaled,y\_train)  
  
# Predicting the Output  
testScore = svc.score(X\_test\_scaled, y\_test)  
  
print(f'\n The result of SVM (Linear) is: Training score - {trainScore} and Test score (Accuracy) - {testScore}')  
  
  
# ---------------------5. c) SVM using Linear and Non-Linear Kernel --------------------------------#  
  
svc\_scores = []  
kernels = ['linear', 'poly', 'rbf', 'sigmoid']  
for i in range(len(kernels)):  
 svc\_classifier = svm.SVC(kernel = kernels[i])  
 svc\_classifier.fit(X\_train\_scaled, y\_train)  
 svc\_scores.append(svc\_classifier.score(X\_test\_scaled, y\_test))  
  
y\_pos = np.arange(len(kernels))  
plt.bar(y\_pos, svc\_scores, color=['black', 'red', 'green', 'blue'])  
plt.title('Performance of Linear and Non-Linear SVM Kernel')  
plt.xticks(y\_pos, kernels)  
print(f'\n The result of Non-Linear SVM is: Test score (Accuracy)')  
print(f'\n {kernels}')  
print(f'\n {svc\_scores}')  
plt.show()