GOWTHAM MALLULA **700745432**

CRN-23921

Assignment 5

GitHub Link: https://github.com/MallulaGowtham/Assignment5

Video Link: https://drive.google.com/file/d/1ttoi3GvCGJcHU5GFqp7TiHuqNtpcUUEa/view?usp=share_link

Q1) Principal Component Analysis :

we have to import the useful libraries

In [1]:
```
#importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn import preprocessing, metrics
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
sns.set(style="white", color_codes=True)
import warnings
warnings.filterwarnings("ignore")
```

Also we read the csv file and print the info about of the file

In [2]:
```python
#1. Principal Component Analysis
#a. Apply PCA on CC dataset.
#b. Apply k-means algorithm on the PCA result and report your observation if the silhouette score has improved or not?
#c. Perform Scaling+PCA+K-Means and report performance.

#Reading the csv file and printing the info about file
dataset_pd = pd.read_csv("CC GENERAL.csv")
dataset_pd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   CUST_ID                           8950 non-null   object 
 1   BALANCE                           8950 non-null   float64
 2   BALANCE_FREQUENCY                 8950 non-null   float64
 3   PURCHASES                         8950 non-null   float64
 4   ONEOFF_PURCHASES                  8950 non-null   float64
 5   INSTALLMENTS_PURCHASES            8950 non-null   float64
 6   CASH_ADVANCE                      8950 non-null   float64
 7   PURCHASES_FREQUENCY               8950 non-null   float64
 8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
 10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
 11  CASH_ADVANCE_TRX                  8950 non-null   int64  
 12  PURCHASES_TRX                     8950 non-null   int64  
 13  CREDIT_LIMIT                      8949 non-null   float64
 14  PAYMENTS                          8950 non-null   float64
 15  MINIMUM_PAYMENTS                  8637 non-null   float64
 16  PRC_FULL_PAYMENT                  8950 non-null   float64
 17  TENURE                            8950 non-null   int64  
dtypes: float64(14), int64(3), object(1)
memory usage: 1.2+ MB
```

We have printed the first five rows of the dataset

In [3]:
```python
#To print first five rows of the dataset to inspect data format
dataset_pd.head()
```

Out[3]:

NCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
3333	0.000000	0	2	1000.0	201.802084	139.509787	0.000000	12
0000	0.250000	4	0	7000.0	4103.032597	1072.340217	0.222222	12
0000	0.000000	0	12	7500.0	622.066742	627.284787	0.000000	12
0000	0.083333	1	1	7500.0	0.000000	NaN	0.000000	12
0000	0.000000	0	1	1200.0	678.334763	244.791237	0.000000	12

We have checked if there is any missing values present in the dataset

In [4]:
```
#checking missing values in dataset
dataset_pd.isnull().any()
```

Out[4]:
```
CUST_ID                             False
BALANCE                             False
BALANCE_FREQUENCY                   False
PURCHASES                           False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
CASH_ADVANCE                        False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
CREDIT_LIMIT                         True
PAYMENTS                            False
MINIMUM_PAYMENTS                     True
PRC_FULL_PAYMENT                    False
TENURE                              False
dtype: bool
```

From above output missing values are there in dataset, so we eliminate missing values by selecting numeric columns of dataset and replace the missing values with mean of respective columns

In [5]:
```
# Select numeric columns of the dataset
numeric_columns = dataset_pd.select_dtypes(include=[np.number]).columns.tolist()

# Replace missing values with mean of the respective columns
dataset_pd[numeric_columns] = dataset_pd[numeric_columns].fillna(dataset_pd[numeric_columns].mean())

dataset_pd.isnull().any()
```

Out[5]:
```
CUST_ID                             False
BALANCE                             False
BALANCE_FREQUENCY                   False
PURCHASES                           False
ONEOFF_PURCHASES                    False
INSTALLMENTS_PURCHASES              False
CASH_ADVANCE                        False
PURCHASES_FREQUENCY                 False
ONEOFF_PURCHASES_FREQUENCY          False
PURCHASES_INSTALLMENTS_FREQUENCY    False
CASH_ADVANCE_FREQUENCY              False
CASH_ADVANCE_TRX                    False
PURCHASES_TRX                       False
CREDIT_LIMIT                        False
PAYMENTS                            False
MINIMUM_PAYMENTS                    False
PRC_FULL_PAYMENT                    False
TENURE                              False
dtype: bool
```

Next we extract the input features and output labels from pandas dataframe and we print the shapes, here x is input features and y is output labels

In [6]:
```
# Extracting input features and output labels from the pandas dataframe and printing their shapes
x = dataset_pd.iloc[:,1:-1]
y = dataset_pd.iloc[:,-1]
print(x.shape,y.shape)
```

```
(8950, 16) (8950,)
```

1a) Apply PCA on CC dataset

In [7]:
```
#1.a Apply PCA on CC Dataset
pca = PCA(3)
x_pca = pca.fit_transform(x)
principalDf = pd.DataFrame(data = x_pca, columns = ['principal component 1', 'principal component 2', 'principal compor
finalDf = pd.concat([principalDf, dataset_pd.iloc[:,-1]], axis = 1)
finalDf.head()
```

Out[7]:

	principal component 1	principal component 2	principal component 3	TENURE
0	-4326.383979	921.566882	183.708383	12
1	4118.916665	-2432.846346	2369.969289	12
2	1497.907641	-1997.578694	-2125.631328	12
3	1394.548536	-1488.743453	-2431.799649	12
4	-3743.351896	757.342657	512.476492	12

1b) Apply k-means algorithm on the PCA result and report your observation if the silhouette score has improved or not?

In [8]:
```
#1.b Apply K Means on PCA Result
X = finalDf.iloc[:,0:-1]
y = finalDf.iloc[:,-1]
```

In [10]:
```
# This is the k in kmeans
nclusters = 3
km = KMeans(n_clusters=nclusters)
km.fit(X)

# predict the cluster for each data point
y_cluster_kmeans = km.predict(X)


# Summary of the predictions made by the classifier
print(classification_report(y, y_cluster_kmeans, zero_division=1))
print(confusion_matrix(y, y_cluster_kmeans))


train_accuracy = accuracy_score(y, y_cluster_kmeans)
print("\nAccuracy for our Training dataset with PCA:", train_accuracy)


#Calculate sihouette Score
score = metrics.silhouette_score(X, y_cluster_kmeans)
print("Sihouette Score: ",score)
"""
Sihouette Score- ranges from −1 to +1 , a high value indicates that the object is well matched to its own cluster and p
"""
```

```
              precision    recall  f1-score   support

           0       0.00      1.00      0.00       0.0
           1       0.00      1.00      0.00       0.0
           2       0.00      1.00      0.00       0.0
           6       1.00      0.00      0.00     204.0
           7       1.00      0.00      0.00     190.0
           8       1.00      0.00      0.00     196.0
           9       1.00      0.00      0.00     175.0
          10       1.00      0.00      0.00     236.0
          11       1.00      0.00      0.00     365.0
          12       1.00      0.00      0.00    7584.0

    accuracy                           0.00    8950.0
   macro avg       0.70      0.30      0.00    8950.0
weighted avg       1.00      0.00      0.00    8950.0

[[   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [ 175   28    1    0    0    0    0    0    0    0]
 [ 173   15    2    0    0    0    0    0    0    0]
 [ 169   27    0    0    0    0    0    0    0    0]
 [ 149   26    0    0    0    0    0    0    0    0]
 [ 188   47    1    0    0    0    0    0    0    0]
 [ 284   78    3    0    0    0    0    0    0    0]
 [5389 2069  126    0    0    0    0    0    0    0]]

Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.5109307274319468
```

Out[10]:
```
'\nSihouette Score- ranges from −1 to +1 , a high value indicates that the object is well matched to its own cluster
and poorly matched to neighboring clusters.\n'
```

1c) Perform Scaling+PCA+K-Means and report performance.

In [11]:
```
#1.c Scaling +PCA + KMeans
x = dataset_pd.iloc[:,1:-1]
y = dataset_pd.iloc[:,-1]
print(x.shape,y.shape)
```

```
(8950, 16) (8950,)
```

Here we do the scaling

In [12]:
```
#Scaling
scaler = StandardScaler()
scaler.fit(x)
X_scaled_array = scaler.transform(x)
#PCA
pca = PCA(3)
x_pca = pca.fit_transform(X_scaled_array)
principalDf = pd.DataFrame(data = x_pca, columns = ['principal component 1', 'principal component 2','principal compone
finalDf = pd.concat([principalDf, dataset_pd.iloc[:,-1]], axis = 1)
finalDf.head()
```

Out[12]:

	principal component 1	principal component 2	principal component 3	TENURE
0	-1.718893	-1.072938	0.535696	12
1	-1.169306	2.509318	0.627979	12
2	0.938414	-0.382598	0.161220	12
3	-0.907503	0.045859	1.521691	12
4	-1.637830	-0.684973	0.425696	12

Here we extract the features and target variable from finalDf dataframe and X contains all columns of dataframe except last one, y contains values from the last column

In [13]:
```
#Extraction of the features and target variable from the finalDf dataframe.
#X contains all the columns of the dataframe except the last one
X = finalDf.iloc[:,0:-1]
#y contains the values from the last column.
y = finalDf["TENURE"]
print(X.shape,y.shape)
```

```
(8950, 3) (8950,)
```

we perform k-means

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.34,random_state=0)
nclusters = 3
# this is the k in kmeans
km = KMeans(n_clusters=nclusters)
km.fit(X_train,y_train)


# predict the cluster for each training data point
y_clus_train = km.predict(X_train)

# Summary of the predictions made by the classifier
print(classification_report(y_train, y_clus_train, zero_division=1))
print(confusion_matrix(y_train, y_clus_train))

train_accuracy = accuracy_score(y_train, y_clus_train)
print("Accuracy for our Training dataset with PCA:", train_accuracy)

#Calculate sihouette Score
score = metrics.silhouette_score(X_train, y_clus_train)
print("Sihouette Score: ",score)

"""
Sihouette Score- ranges from -1 to +1 , a high value indicates that the object is well matched to its own cluster and p
"""
```

```
              precision    recall  f1-score   support

           0       0.00      1.00      0.00       0.0
           1       0.00      1.00      0.00       0.0
           2       0.00      1.00      0.00       0.0
           6       1.00      0.00      0.00     139.0
           7       1.00      0.00      0.00     135.0
           8       1.00      0.00      0.00     128.0
           9       1.00      0.00      0.00     118.0
          10       1.00      0.00      0.00     151.0
          11       1.00      0.00      0.00     262.0
          12       1.00      0.00      0.00    4974.0

    accuracy                           0.00    5907.0
   macro avg       0.70      0.30      0.00    5907.0
weighted avg       1.00      0.00      0.00    5907.0

[[   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   4   30  105    0    0    0    0    0    0    0]
 [   1   26  108    0    0    0    0    0    0    0]
 [   4   28   96    0    0    0    0    0    0    0]
 [   2   27   89    0    0    0    0    0    0    0]
 [   6   38  107    0    0    0    0    0    0    0]
 [  11   66  185    0    0    0    0    0    0    0]
 [ 735  842 3397    0    0    0    0    0    0    0]]
Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.38140423993908845
```

ut[14]:
```
'\nSihouette Score- ranges from -1 to +1 , a high value indicates that the object is well matched to its own cluster
and poorly matched to neighboring clusters.\n'
```

In [15]:
```
# predict the cluster for each testing data point
y_clus_test = km.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_clus_test, zero_division=1))
print(confusion_matrix(y_test, y_clus_test))

train_accuracy = accuracy_score(y_test, y_clus_test)
print("\nAccuracy for our Training dataset with PCA:", train_accuracy)

#Calculate sihouette Score
score = metrics.silhouette_score(X_test, y_clus_test)
print("Sihouette Score: ",score)

"""
Sihouette Score- ranges from -1 to +1 , a high value indicates that the object is well matched to its own cluster and p
"""
```

```
              precision    recall  f1-score   support

           0       0.00      1.00      0.00       0.0
           1       0.00      1.00      0.00       0.0
           2       0.00      1.00      0.00       0.0
           6       1.00      0.00      0.00      65.0
           7       1.00      0.00      0.00      55.0
           8       1.00      0.00      0.00      68.0
           9       1.00      0.00      0.00      57.0
          10       1.00      0.00      0.00      85.0
          11       1.00      0.00      0.00     103.0
          12       1.00      0.00      0.00    2610.0

    accuracy                           0.00    3043.0
   macro avg       0.70      0.30      0.00    3043.0
weighted avg       1.00      0.00      0.00    3043.0

[[   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0]
 [   3   21   41    0    0    0    0    0    0    0]
 [   0   12   43    0    0    0    0    0    0    0]
 [   1   10   57    0    0    0    0    0    0    0]
 [   0   22   35    0    0    0    0    0    0    0]
 [   5   17   63    0    0    0    0    0    0    0]
 [   4   30   69    0    0    0    0    0    0    0]
 [ 395  450 1765    0    0    0    0    0    0    0]]

Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.383642891892748
```

ıt[15]:
```
'\nSihouette Score- ranges from -1 to +1 , a high value indicates that the object is well matched to its own cluster
and poorly matched to neighboring clusters.\n'
```

2) Use pd_speech_features.csv

In [16]:
```
# 2.Use pd_speech_features.csv
# a. Perform Scaling
# b. Apply PCA (k=3)
# c. Use SVM to report performance

dataset_pd = pd.read_csv('pd_speech_features.csv')
dataset_pd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 756 entries, 0 to 755
Columns: 755 entries, id to class
dtypes: float64(749), int64(6)
memory usage: 4.4 MB
```

2a) Perform Scaling

In [20]:
```
#2.a Scaling Data
scaler = StandardScaler()
X_Scale = scaler.fit_transform(X)
```

2b) Apply PCA (k=3)

In [21]:
```
#2.b Apply PCA with k =3
pca3 = PCA(n_components=3)
principalComponents = pca3.fit_transform(X_Scale)

principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2','Pri

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

Out[21]:

	principal component 1	principal component 2	Principal Component 3	class
0	-10.047372	1.471074	-6.846408	1
1	-10.637725	1.583749	-6.830980	1
2	-13.516185	-1.253544	-6.818701	1
3	-9.155083	8.833601	15.290901	1
4	-6.764470	4.611469	15.637123	1

2c) Use SVM to report performance

In [23]:

```
#2.c Using Support Vector Machine's (SVM)

from sklearn.svm import SVC

svmClassifier = SVC()
svmClassifier.fit(X_train, y_train)

y_pred = svmClassifier.predict(X_test)

# Summary of the predictions made by the classifier
print(classification_report(y_test, y_pred, zero_division=1))
print(confusion_matrix(y_test, y_pred))
# Accuracy score
glass_acc_svc = accuracy_score(y_pred,y_test)
print('accuracy is',glass_acc_svc )

#Calculate sihouette Score
score = metrics.silhouette_score(X_test, y_pred)
print("Sihouette Score: ",score)
```

```
              precision    recall  f1-score   support

           0       0.67      0.42      0.51        62
           1       0.84      0.93      0.88       196

    accuracy                           0.81       258
   macro avg       0.75      0.68      0.70       258
weighted avg       0.80      0.81      0.79       258

[[ 26  36]
 [ 13 183]]
accuracy is 0.810077519379845
Sihouette Score:  0.2504463624937735
```

3) Apply Linear Discriminant Analysis (LDA) on Iris.csv dataset to reduce dimensionality of data tok=2.

In [24]:
```
#3.Apply Linear Discriminant Analysis (LDA) on Iris.csv dataset to reduce dimensionality of data to k=2.
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
dataset_iris = pd.read_csv('Iris.csv')
dataset_iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [26]:
```
x = dataset_iris.iloc[:,1:-1]
y = dataset_iris.iloc[:,-1]
print(x.shape,y.shape)
```

```
(150, 4) (150,)
```

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

In [28]:
```
#performs data preprocessing by standardizing the input features and encoding the target variable
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
le = LabelEncoder()
y = le.fit_transform(y)
```

In [29]:
```
# Perform Linear Discriminant Analysis on the training data to reduce dimensionality to 2 components
# and transform the training and test data to the reduced space
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=2)
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
# Print the shape of the transformed training and test data
print(X_train.shape,X_test.shape)
```

```
(105, 2) (45, 2)
```

4) Briefly identify the difference between PCA and LDA

Ans.

Both LDA and PCA are techniques used for dimensionality reduction, which means reducing the number of features or variables in a dataset while retaining the most important information. They both use linear transformations to convert the original data into a lower dimensional space.

PCA is an unsupervised learning algorithm that identifies the directions of maximum variance in the data, regardless of any class labels. It generates new features, called principal components, which are orthogonal (not correlated) and capture the largest variance in the data. The first principal component captures the most variability in the data, the second captures the second most, and so on.

LDA, on the other hand, is a supervised learning algorithm that aims to maximize the separability between different classes in the data. It identifies linear discriminants that maximize the variance between different categories while minimizing the variance within each category. LDA does this by considering the class labels in the data and finding the directions of maximum class separability.

Therefore, the main difference between PCA and LDA is that PCA is focused on capturing the maximum variance in the data, while LDA is focused on finding the directions that best separate different classes. PCA does not take into account any differences between classes, while LDA explicitly considers the class labels to find the optimal discriminative directions.