MACHINE LEARNING ASSIGNMENT 5

NAME : ASHOK SAI SUDIREDDY
ID : 700734963
Video Link :
https://drive.google.com/file/d/1tHq3AE4MbdWvXxZXG4xXp97HFuS_E8zN/view?usp=share_link
GitHub Link :
https://github.com/AshokSai1999/Machine-Learning.git

Q1) Principal Component Analysis

First we import the 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")
```

Then we read the csv file and print the info about file

In [2]:
```
#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
```

Next we print the first five rows of dataset to inspect data format

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

Out[3]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667
	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000
	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000
	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333
	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333

Then we check if there is any missing values in dataset

```
In [5]: #checking missing values in dataset
        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                         True
        PAYMENTS                            False
        MINIMUM_PAYMENTS                     True
        PRC_FULL_PAYMENT                    False
        TENURE                              False
        dtype: bool
```

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

```
In [6]: # 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[6]: 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 [7]:
```
# 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 [8]:
```
#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 component 3'])
finalDf = pd.concat([principalDf, dataset_pd.iloc[:,-1]], axis = 1)
finalDf.head()
```

Out[8]:

	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 [9]:
```
#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 poorly mat
"""
```

```
              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]
 [   1  175   28    0    0    0    0    0    0    0]
 [   2  173   15    0    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]
 [   1  189   46    0    0    0    0    0    0    0]
 [   3  284   78    0    0    0    0    0    0    0]
 [ 126 5393 2065    0    0    0    0    0    0    0]]

Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.5111604907756386
```

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 perform Scaling and PCA

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 component 3'])
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	-1.072941	0.535646	12
1	-1.169307	2.509325	0.628153	12
2	0.938413	-0.382592	0.161552	12
3	-0.907503	0.045860	1.521728	12
4	-1.637829	-0.684978	0.425596	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,)
```

Here 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 poorly matc
"""
```

```
              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]
 [ 105    4   30    0    0    0    0    0    0    0]
 [ 108    1   26    0    0    0    0    0    0    0]
 [  96    4   28    0    0    0    0    0    0    0]
 [  89    2   27    0    0    0    0    0    0    0]
 [ 107    6   38    0    0    0    0    0    0    0]
 [ 185   11   66    0    0    0    0    0    0    0]
 [3393  739  842    0    0    0    0    0    0    0]]
Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.3812067642530423
```

Out[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 poorly matc
"""
```

```
              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]
 [  41    3   21    0    0    0    0    0    0    0]
 [  42    1   12    0    0    0    0    0    0    0]
 [  57    1   10    0    0    0    0    0    0    0]
 [  35    0   22    0    0    0    0    0    0    0]
 [  63    5   17    0    0    0    0    0    0    0]
 [  69    4   30    0    0    0    0    0    0    0]
 [1763  397  450    0    0    0    0    0    0    0]]

Accuracy for our Training dataset with PCA: 0.0
Sihouette Score:  0.3833211693140143
```

Out[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','Principal Co

         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	1.471072	-6.846399	1
1	-10.637725	1.583745	-6.830974	1
2	-13.516185	-1.253546	-6.818693	1
3	-9.155084	8.833575	15.290922	1
4	-6.764470	4.611447	15.637136	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:  0.2504463621666034
```

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

```
In [25]: #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 [27]: x = dataset_iris.iloc[:,1:-1]
y = dataset_iris.iloc[:,-1]
print(x.shape,y.shape)
```

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

```
In [28]: X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=0)
```

```
In [29]: #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 [30]: # 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.