**Assignment - 5**

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.

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")

dataset\_CC = pd.read\_csv('CC.csv')

dataset\_CC.info()

dataset\_CC.head()

dataset\_CC.isnull().any()

dataset\_CC.fillna(dataset\_CC.mean(), inplace=True)

dataset\_CC.isnull().any()

x = dataset\_CC.iloc[:,1:-1]

y = dataset\_CC.iloc[:,-1]

print(x.shape,y.shape)

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\_CC.iloc[:,-1]], axis = 1)

finalDf.head()

X = finalDf.iloc[:,0:-1]

y = finalDf.iloc[:,-1]

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 silhouette Score

score = metrics.silhouette\_score(X, y\_cluster\_kmeans)

print("Silhouette Score: ",score)

Silhouette 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.

x = dataset\_CC.iloc[:,1:-1]

y = dataset\_CC.iloc[:,-1]

print(x.shape,y.shape)

#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\_CC.iloc[:,-1]], axis = 1)

finalDf.head()

X = finalDf.iloc[:,0:-1]

y = finalDf["TENURE"]

print(X.shape,y.shape)

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 silhouette Score

score = metrics.silhouette\_score(X\_train, y\_clus\_train)

print("Silhouette Score: ",score)

Silhouette 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.

# 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 silhouette Score

score = metrics.silhouette\_score(X\_test, y\_clus\_test)

print("Silhouette Score: ",score)

Silhouette 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.

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()

dataset\_pd.head()

dataset\_pd.isnull().any()

X = dataset\_pd.drop('class',axis=1).values

y = dataset\_pd['class'].values

scaler = StandardScaler()

X\_Scale = scaler.fit\_transform(X)

pca3 = PCA(n\_components=3)

principalComponents = pca3.fit\_transform(X\_Scale)

principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2','Principal Component 3'])

finalDf = pd.concat([principalDf, dataset\_pd[['class']]], axis = 1)

finalDf.head()

X = finalDf.drop('class',axis=1).values

y = finalDf['class'].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.34,random\_state=0)

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 silhouette Score

score = metrics.silhouette\_score(X\_test, y\_pred)

print("Silhouette Score: ",score)

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()

dataset\_iris.isnull().any()

x = dataset\_iris.iloc[:,1:-1]

y = dataset\_iris.iloc[:,-1]

print(x.shape,y.shape)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.3, random\_state=0)

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

le = LabelEncoder()

y = le.fit\_transform(y)

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(X\_train.shape,X\_test.shape)

4. Briefly identify the difference between PCA and LDA

PCA reduces the features into a smaller subset of orthogonal variables, called principal components – linear combinations of the original variables. The first component captures the largest variability of the data, while the second captures the second largest, and so on. LDA finds the linear discriminants in order to maximize the variance between the different categories while minimizing the variance within the class. Both LDA and PCA rely on linear transformations and aim to maximize the variance in a lower dimension. PCA is an unsupervised learning algorithm while LDA is a supervised learning algorithm. This means that PCA finds directions of maximum variance regardless of class labels while LDA finds directions of maximum class separability.

**https://github.com/NXI57230/Assignment1\_700725723**