**ML ASSIGNMENT 5**

**Name: varna nemulla**

**ID: 700744920**

## **In-class programming:**

1. **Principal Component Analysis**

**a. Apply PCA on the CC dataset.**

**b. Apply the 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.**

**Source Code:**

# import 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")

df = pd.read\_csv("/Users/varnanemulla/Documents/ML Assignment 5/CC GENERAL.csv")

df.head()

#Reading the CC General file

**Screenshots of the output:**

Graphical user interface, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

1. **Apply PCA on CC dataset.**

**Source Code:**

#Datasets can be analyzed with PCA so that redundant features can be removed without losing too much information.

pca = PCA(3) #Instantiate PCA

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, df.iloc[:,-1]], axis = 1)

finalDf.head()

#PCA(3)- performs principal component analysis (PCA) on dataset x, reducing the dimensionality of the data from the original number of features to 3 principal components.

#fit\_transform()- method of the PCA object is called on the data x to obtain a transformed version of the data, where each observation is represented by its three principal components.

#principalDf- represents the transformed data x\_pca and three principal components

#finalDf- concatenating principalDf with the last column of the original DataFrame df using pd.concat(). This is likely done to include the target variable (the variable being predicted) with the transformed data.

**Screenshots of the output:**

Text

Description automatically generated

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

**Source Code:**

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

y = finalDf.iloc[:,-1]

print(X.shape,y.shape)

#X- predictor variable- contains all rows of finalDf except for the last column, representing the principal components generated by PCA

#y- target variable- contains only the last column of finalDf, representing the target variable.

nclusters = 3 # this is the k in kmeans

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))

#finding the accuracy

train\_accuracy = accuracy\_score(y, y\_cluster\_kmeans)

print("\nAccuracy for our Training dataset with PCA:", train\_accuracy)

#Calculating sihouette Score

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

print("Sihouette Score: ",score) #ranges from -1 to +1, high value shows that it is matched more

# predict the cluster for each data point

**Screenshots of the output:**

Graphical user interface, text, application, email

Description automatically generated

Table

Description automatically generated

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

**Source Code:**

x = df.iloc[:,1:-1]

y = df.iloc[:,-1]

print(x.shape,y.shape)

## Scale the dataset; This is very important before you apply PCA

scaler = StandardScaler()

scaler.fit(x)

X\_scaled\_array = scaler.transform(x)

# Instantiate PCA

pca = PCA(3)

# Determine transformed features

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, df.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)

#Calculating sihouette Score

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

print("Sihouette Score: ",score) #ranges from -1 to +1, high value shows that it is matched more

# 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 Testing dataset with PCA:", train\_accuracy)

#Calculating sihouette Score

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

print("Sihouette Score: ",score) #ranges from -1 to +1, high value shows that it is matched more

#First scale the data Applies the fit\_transform() method of the StandardScaler instance to the feature matrix X to perform feature scaling.

#This method first computes the mean and standard deviation of each feature in X, and then scales the features such that they have zero mean and unit variance

#Then apply PCA to reduce the dimensionality to 3 components.

#Then split the data into training and testing sets using the train\_test\_split() function.

#Perform K-means clustering on the training set and test set and predict the cluster for each training data point.

#Finally, evaluate the performance of the clustering on the training & training set using classification\_report(), confusion\_matrix(), accuracy\_score(), and silhouette\_score() functions from sklearn.metrics.

**Screenshots of the output:**

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text

Description automatically generatedGraphical user interface, text, table

Description automatically generated

1. **Use pd\_speech\_features.csv.**

**Source Code:**

df\_pd = pd.read\_csv("/Users/varnanemulla/Documents/ML Assignment 5/pd\_speech\_features.csv")

df\_pd.head()

df\_pd.isnull().any()

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

Y = df\_pd['class'].values

# this codes represents dropping the target variable class from the main data frame and creating a new data frame X

# Y returns the class column from the main data frame

**Screenshots of the output:**

Graphical user interface, text, application

Description automatically generated

1. **Perform Scaling**

**Source Code:**

#Scaling Data

scaler = StandardScaler()

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

#StandardScaler to scale the input X, this is important as it ensures that all the features are on the same scale and prevents features with larger magnitude from dominating the distance calculations

#Applies the fit\_transform() method of the StandardScaler instance to the feature matrix X to perform feature scaling

**Screenshots of the output:**

Graphical user interface, text, application

Description automatically generated

**b. Apply PCA (k=3)**

**Source Code:**

# 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 Component 3'])

finalDf = pd.concat([principalDf, df\_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=0.3,random\_state=0)

**Screenshots of the output:**

Graphical user interface, text, application

Description automatically generated

**c. Use SVM to report performance.**

**Source Code:**

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)

#It then trains an SVM classifier on the training set, predicts the classes for the test set using the trained classifier, and evaluates the performance using a classification report, confusion matrix, accuracy score, and silhouette score.

**Screenshots of the output:**

Graphical user interface, text, application

Description automatically generated

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

**Source Code:**

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

df\_iris = pd.read\_csv("/Users/varnanemulla/Documents/ML Assignment 5/Iris.csv")

df\_iris.head()

df\_iris.isnull().any()

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

y = df\_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)

#fit and transform the scaler object on our training data and only transform our test data.

#LabelEncoder to encode our target variable y into numerical values.

#(LDA) to perform dimensionality reduction on our input features x. Here, we are reducing the number of input features to 2 using n\_components=2

#we transform our training and test data using the fit\_transform and transform methods of the LDA object respectively

**Screenshots of the output:**

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

1. **Briefly identify the difference between PCA and LDA**

**Explanation:**

PCA (Principal Component Analysis) and LDA (Linear Discriminant Analysis) are popular machine learning techniques for dimensionality reduction. However, they have different purposes and methods:

Purpose: PCA is used for unsupervised learning and finds the directions of maximum variance in a dataset. It reduces the number of features by transforming the original dataset into a new coordinate system, where the features are uncorrelated and sorted by their variance. PCA is commonly used for data compression, visualization, and noise reduction. On the other hand, LDA is used for supervised learning and aims to find the linear combinations of features that best separate the classes. It reduces the number of features by projecting the original dataset onto a lower-dimensional space while maximizing class separability. LDA is commonly used for feature extraction, pattern recognition, and classification.

Method: PCA operates by finding the eigenvectors and eigenvalues of the covariance matrix of the data. The eigenvectors represent the directions of maximum variance, and the eigenvalues represent the amount of variance explained by each eigenvector. PCA selects the top k eigenvectors, where k is the desired dimensionality of the reduced dataset. LDA, on the other hand, maximizes the between-class scatter and minimizes the within-class scatter of the data. It involves finding the eigenvectors and eigenvalues of the product of two matrices: the between-class scatter matrix and the within-class scatter matrix. LDA selects the top k eigenvectors that correspond to the largest eigenvalues.

**Description:** In this ICP I did Principal Component Analysis and Applied PCA on the CC dataset. Then, I applied the k-means algorithm to the PCA result and reported my observation. the silhouette score has improved. Performed Scaling+PCA+K-Means and report performance. Used pd\_speech\_features.csv and Performed Scaling then, Applied PCA (k=3). Used SVM to report performance. Applied Linear Discriminant Analysis (LDA) on the Iris.csv dataset to reduce the dimensionality of data tok=2. Briefly explained the difference between PCA and LDA.

**GitHub Link:** [https://github.com/VarnaNemulla/ML-Assignment-5](file:///Users/varnanemulla/Downloads/ML%20ASSIGNMENT%205.docx)

**Video Link:** [https://drive.google.com/file/d/1ElLSqE8vn\_1grfKx\_d47ER53AZeD1sQ3/view?usp=sharing](file:///Users/varnanemulla/Downloads/ML%20ASSIGNMENT%205.docx)