**Machine Learning (Assignment # 5)**

**Name: Sindhu Rajanala**

**Student id: 700741228**

**1. Principal Component Analysis**

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

**Code:**

import math

import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn import preprocessing

import seaborn as sns

sns.set(style="white", color\_codes=True)

import warnings

warnings.filterwarnings("ignore")

df= pd.read\_csv("CC.csv")

df.head()

df.shape

output:

Graphical user interface, text, application, email

Description automatically generated

df['TENURE'].value\_counts()

output:

Graphical user interface, application

Description automatically generated with medium confidence

x = df.iloc[:,[1,2,3,4]]

y = df.iloc[:,-1]

le = preprocessing.LabelEncoder()

df['CUST\_ID'] = le.fit\_transform(df.CUST\_ID.values)

pca = PCA(n\_components=2)

principalComponents = pca.fit\_transform(x)

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

finalDf = pd.concat([principalDf, df[['TENURE']]], axis = 1)

finalDf.head()

output:

Graphical user interface, text, application

Description automatically generated

Description :

Principal Component Analysis is a statistical procedure that allows you to summarize the information content in large data tables by means of a smaller set of “summary indices” that can be more easily visualized and analyzed.

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

**Code:**

from sklearn.cluster import KMeans

nclusters = 2

km = KMeans(n\_clusters=nclusters)

km.fit(x)

y\_cluster\_kmeans = km.predict(x)

from sklearn import metrics

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

print(score)

output:

Graphical user interface

Description automatically generated

Description: silhouette score has improved when we applied K-Mean algorithm.

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

**code:**

scaler = StandardScaler()

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

pca2 = PCA(n\_components=2)

principalComponents = pca2.fit\_transform(X\_Scale)

principalDf1 = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])

finalDf1 = pd.concat([principalDf1, df[['TENURE']]], axis = 1)

finalDf1.head()

from sklearn.cluster import KMeans

nclusters = 2 # this is the k in kmeans

km = KMeans(n\_clusters=nclusters)

km.fit(X\_Scale)

y\_cluster\_kmeans = km.predict(X\_Scale)

from sklearn import metrics

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

print(score)

output:

Graphical user interface, text, application, email

Description automatically generated

Description:

Sihouette 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**

**code:**

dataset\_pd = pd.read\_csv('pd\_speech\_features.csv')

dataset\_pd.info()

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

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

scaler = StandardScaler()

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

Output:

Graphical user interface, text, application, email

Description automatically generated

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

**code:**

pca1 = PCA(n\_components=3)

principalComponents = pca1.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()

output:

Graphical user interface, text

Description automatically generated

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

**Code:**

from sklearn.model\_selection import train\_test\_split

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

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

svmClassifier = SVC()

svmClassifier.fit(X\_train, y\_train)

y\_pred = svmClassifier.predict(X\_test)

print(classification\_report(y\_test, y\_pred, zero\_division=1))

print(confusion\_matrix(y\_test, y\_pred))

glass\_acc\_svc = accuracy\_score(y\_pred,y\_test)

print('accuracy is',glass\_acc\_svc )

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

print("Sihouette Score: ",score)

output:

Graphical user interface, text, email

Description automatically generated

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

**Code:**

df = pd.read\_csv("iris.csv")

df.head()

stdsc = StandardScaler()

X\_train\_std = stdsc.fit\_transform(df.iloc[:,range(0,4)].values)

class\_le = LabelEncoder()

y = class\_le.fit\_transform(df['Species'].values)

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

lda = LinearDiscriminantAnalysis(n\_components=2)

X\_train\_lda = lda.fit\_transform(X\_train\_std,y)

data=pd.DataFrame(X\_train\_lda)

data['class']=y

data.columns=["LD1","LD2","class"]

data.head()

markers = ['s', 'x', 'o']

colors = ['r', 'b', 'g']

sns.lmplot(x="LD1", y="LD2", data=data, hue='class', markers=markers, fit\_reg=False, legend=False)

plt.legend(loc='upper center')

plt.show()

output:

Graphical user interface, text, application

Description automatically generated

Description:

Linear Discriminant Analysis is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modelling differences in groups i.e. separating two or more classes.

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

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.

PCA- It 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- LDA finds the linear discriminants in order to maximize the variance between the different categories while minimizing the variance within the class.

as 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.