**CS6301 MACHINE LEARNING LAB WEEK 10 – DIMENSIONALITY REDUCTION**

**SRIHARI. S – 2018103601**

**Date**: 19-04-2021 Monday

**Aim**: To implement Principal Component Analysis and Linear Discriminant analysis and visualize datasets with a large number of features.

**Principal Component Analysis:**

PCA is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.It reduces the number of variables of a data set, while preserving as much information as possible.

**Algorithm:**

• Write N datapoints xi = (x1i, x2i, . . . , xMi) as row vectors

• Put these vectors into a matrix X (which will have size N ×M)

• Centre the data by subtracting off the mean of each column, putting it into matrix B

• Compute the covariance matrix C = 1/N BT B

• Compute the eigenvalues and eigenvectors of C, so V-1 CV = D, where V holds the eigenvectors of C and D is the M ×M diagonal eigenvalue matrix.

• Sort the columns of D into order of decreasing eigenvalues, and apply the same order to the columns of V

• Reject those with eigenvalue less than some n (eta), leaving L dimensions in the data

**Linear Discriminant Analysis:**

It is a dimensionality reduction technique. It is used as a pre-processing step in Machine Learning and applications of pattern classification. The goal of LDA is to project the features in higher dimensional space onto a lowerdimensional space in order to avoid the curse of dimensionality and also reduce resources and dimensional costs. LDA is a supervised classification technique that is considered a part of crafting competitive machine learning models. This category of dimensionality reduction is used in areas like image recognition and predictive analysis in marketing.

**Algorithm:**

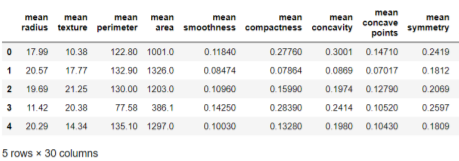
LDA focuses primarily on projecting the features in higher dimension space to lower dimensions. You can achieve this in three steps:

* Firstly, you need to calculate the separability between classes which is the distance between the mean of different classes. This is called the between-class variance.
* Secondly, calculate the distance between the mean and sample of each class. It is also called the within-class variance.
* Finally, construct the lower-dimensional space which maximizes the between-class variance and minimizes the within-class variance. P is considered as the lowerdimensional space projection, also called Fisher’s criterion.

**Implementation of PCA:**

**Url:** <https://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+(diagnostic)>

**Dataset Description:** Breast cancer wisconsin (diagnostic) dataset



**Data Set Characteristics:**

• Number of Instances: 569

• Number of Attributes: 30 numeric, predictive attributes and the class

• Attribute Information:

- radius (mean of distances from center to points on the perimeter)

- texture (standard deviation of gray-scale values)

- smoothness (local variation in radius lengths)

- compactness (perimeter^2 / area - 1.0)

- concavity (severity of concave portions of the contour)

- concave points (number of concave portions of the contour)

- symmetry

-fractal dimension ("coastline approximation" - 1)

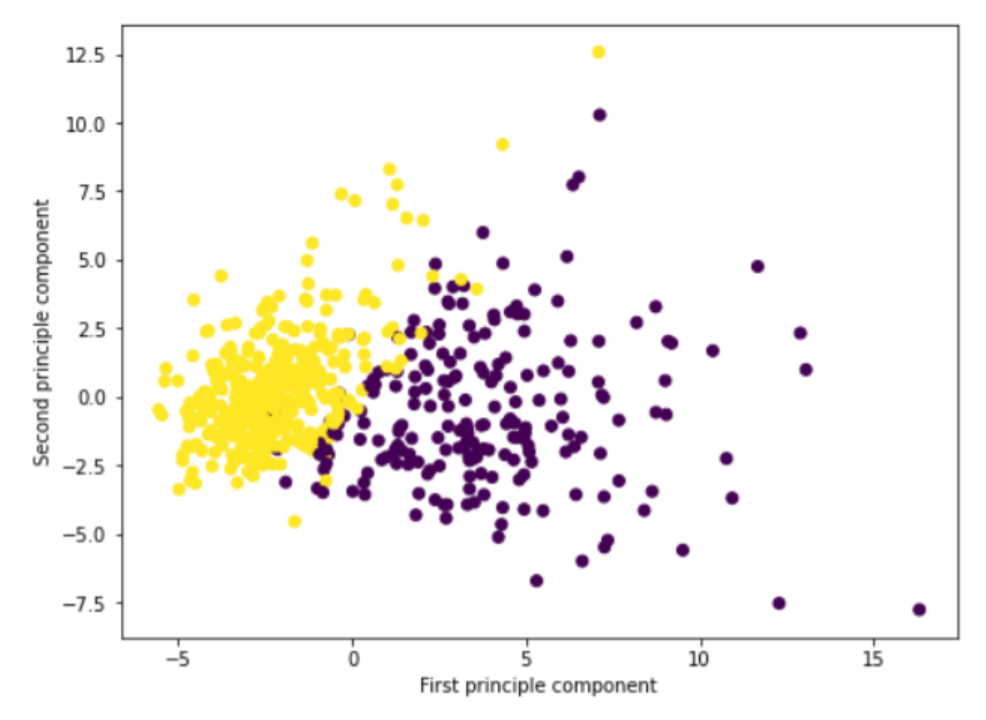
• The mean, standard error, and "worst" or largest (mean of the threeworst/largest values) of these features were computed for each image, resulting in 30 features. For instance, field 0 is Mean Radius, field10 is Radius SE, field 20 is Worst Radius.

• Class

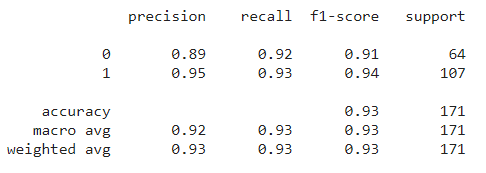
- WDBC-Malignant

- WDBC-Benign

**OUTPUT:**







**CODE:**

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

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

from sklearn.neighbors import KNeighborsClassifier

%matplotlib inline

from sklearn.datasets import load\_breast\_cancer

cancer=load\_breast\_cancer()

X=cancer.data

y=cancer.target

df=pd.DataFrame(cancer['data'],columns=cancer['feature\_names'])

df.head()

from sklearn.preprocessing import StandardScaler

scaler=StandardScaler()

scaler.fit(df)

scaled\_data=scaler.transform(df)

from sklearn.decomposition import PCA

pca=PCA(n\_components=2)

pca.fit(scaled\_data)

x\_pca=pca.transform(scaled\_data)

scaled\_data.shape

x\_pca.shape

scaled\_data

x\_pca

plt.figure(figsize=(8,6))

plt.scatter(x\_pca[:,0],x\_pca[:,1],c=cancer['target'])

plt.xlabel('First principle component')

plt.ylabel('Second principle component')

X\_train\_new, X\_test\_new, y\_train, y\_test = train\_test\_split(x\_pca, y, test\_size = 0.3,

random\_state=20, stratify=y)

knn\_pca = KNeighborsClassifier(7)

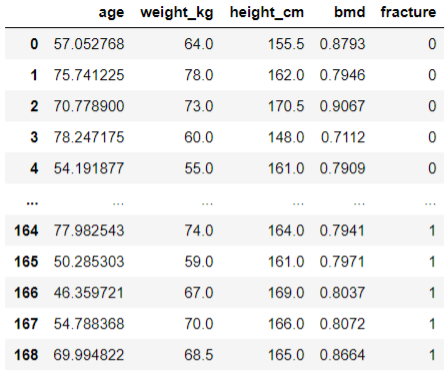
knn\_pca.fit(X\_train\_new,y\_train)

print("Train Accuracy ",knn\_pca.score(X\_train\_new,y\_train)\*100,"%")

print("Test Accuracy ",knn\_pca.score(X\_test\_new,y\_test) \*100,"%")

**Implementation Of LDA:**

**Dataset Used:** bmd.csv (bone mineral density)



The file bmd.csv contains 169 records of bone densitometries (measurement of bone mineral density). The following variables were collected:

• id – patient’s number

• age – patient’s age

• fracture – hip fracture (fracture / no fracture)

• weight\_kg – weight measured in Kg

• height\_cm – height measure in cm

• waiting\_time – time the patient had to wait for the densitometry (in minutes)

• bmd – bone mineral density measure in the hip

#Import libraries

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.neighbors import KNeighborsClassifier

#Load dataset

n\_components = 2

data = pd.read\_csv('bmd.csv')

data = data[['age','weight\_kg','height\_cm','bmd','fracture']]

data

#Normalizing the attributes and encoding labels

from sklearn.preprocessing import StandardScaler

stdsc = StandardScaler()

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

from sklearn.preprocessing import LabelEncoder

class\_le = LabelEncoder()

y = class\_le.fit\_transform(data['fracture'].values)

#Between class variance

S\_W = np.zeros((4,4))

for i in range(2):

S\_W += np.cov(X\_train\_std[y==i].T)

S\_W

#Distance between mean and sample of class

N=np.bincount(y)

vecs=[]

[vecs.append(np.mean(X\_train\_std[y==i],axis=0)) for i in range(2)]

mean\_overall = np.mean(X\_train\_std, axis=0)

S\_B=np.zeros((4,4))

for i in range(2):

S\_B += N[i]\*(((vecs[i]-mean\_overall).reshape(4,1)).dot(((vecs[i]-

mean\_overall).reshape(1,4))))

S\_B

#Display eigen values

eigen\_vals, eigen\_vecs = np.linalg.eig(np.linalg.inv(S\_W).dot(S\_B))

eigen\_pairs = [(np.abs(eigen\_vals[i]), eigen\_vecs[:,i]) for i in range(len(eigen\_vals))]

eigen\_pairs = sorted(eigen\_pairs,key=lambda k: k[0], reverse=True)

print('Eigenvalues in decreasing order:\n')

for eigen\_val in eigen\_pairs:

print(eigen\_val[0])

#Finding LD1 & LD2

tot = sum(eigen\_vals.real)

discr = [(i / tot) for i in sorted(eigen\_vals.real, reverse=True)]

cum\_discr = np.cumsum(discr)

W=np.hstack((eigen\_pairs[0][1][:,].reshape(4,1),eigen\_pairs[1][1][:, ].reshape(4,1))).real

X\_train\_lda = X\_train\_std.dot(W)

#Adding LD1 & LD2 Value to dataframe

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

data['class']=y

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

data.head()

#Visualizing the data after LDA

import seaborn as sns

markers = ['x','o']

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

legend=False, palette='rainbow')

plt.legend(loc='upper center')

plt.show()

#KNN Classifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[['LD1','LD2']], data['class'],

test\_size=0.20)

knn= KNeighborsClassifier(n\_neighbors=3)

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

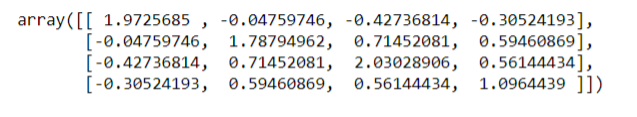
y\_pred = knn.predict(X\_test)

print("Train accuracy ",knn.score(X\_train,y\_train)\*100,"%")

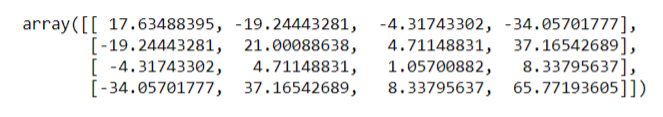
print("Test accuracy ",knn.score(X\_test,y\_test)\*100,”%”)

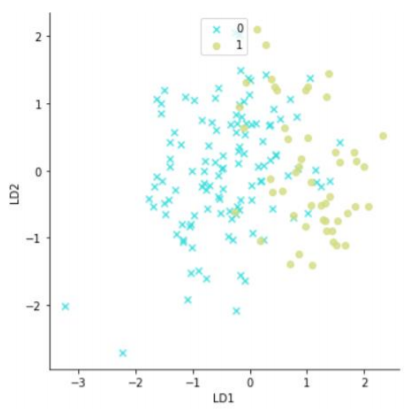
**OUTPUT**

**Between Class Variance**



**Within class variance**





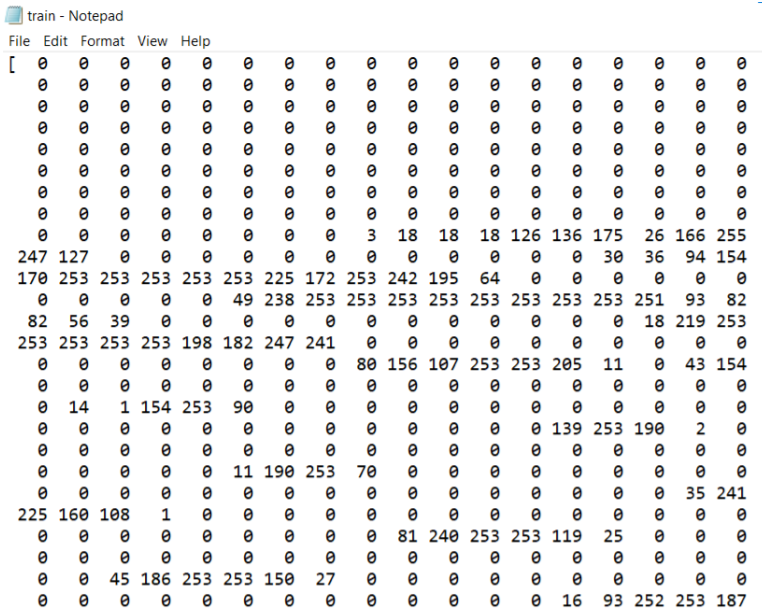


**Implementation of PCA and LDA on a dataset having more than 100 columns:**

**Dataset:** MNIST Dataset

The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems which contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset.

**Input:**



**PCA USING MNIST:**

#Import Library

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

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

from sklearn.neighbors import KNeighborsClassifier

%matplotlib inline

#Import dataset

data = pd.read\_csv("mnist\_train.csv")

X = data.drop('label', axis=1)

y = data.label

data.head()

#Normalizing Dataset using StandardScaler

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

scaled\_data=scaler.fit\_transform(X)

scaled\_data

#Initializing PCA

from sklearn.decomposition import PCA

pca=PCA(n\_components=2)

pca.fit(scaled\_data)

x\_pca=pca.transform(scaled\_data)

scaled\_data.shape

x\_pca.shape

scaled\_data

x\_pca

#Printing transformed data

#Visualizing data after applying PCA

plt.figure(figsize=(8,6))

plt.scatter(x\_pca[:,0],x\_pca[:,1],c=cancer['target'])

plt.xlabel('First principle component')

plt.ylabel('Second principle component')

#KNN Classifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state=20,

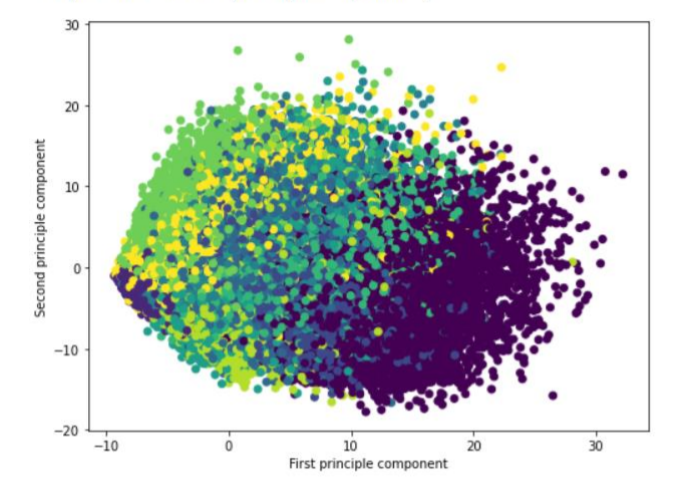
stratify=y)

knn = KNeighborsClassifier(3)

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

print("Train Accuracy",knn.score(X\_train,y\_train)\*100,"%")

print("Test Accuracy ",knn.score(X\_test,y\_test) \*100,"%")





**LDA USING MNIST**

#Import library

import matplotlib.pyplot as plt

from sklearn import datasets, svm, metrics

import pandas as pd

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

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

from sklearn.neighbors import KNeighborsClassifier

%matplotlib inline

#Loading data

data = pd.read\_csv("mnist\_train.csv")

X = data.drop('label', axis=1)

y = data.label

target\_names = data.label

data.head()

# Create a classifier: a Fisher's LDA classifier

lda = LinearDiscriminantAnalysis(n\_components=2, solver='eigen', shrinkage=0.1)

lda = lda.fit(X, y)

X\_r\_lda = lda.transform(X)

# Visualize transformed data on learnt discriminant coordinates

plt.figure(figsize=[13,6])

for i, target\_name in zip([0,1,2,3,4,5,6,7,8,9], target\_names):

plt.scatter(X\_r\_lda[y == i, 0], X\_r\_lda[y == i, 1], alpha=.8,label=target\_name,

marker='$%.f$'%i)

plt.xlabel('Discriminant Coordinate 1')

plt.ylabel('Discriminant Coordinate 2')

plt.tight\_layout()

#KNN Classifier

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state=20,

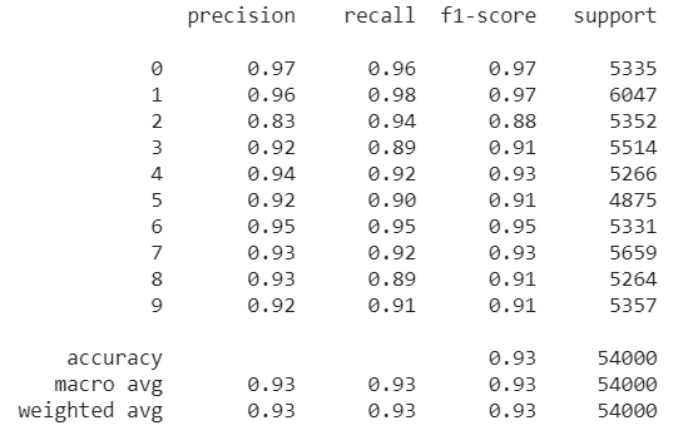
stratify=y)

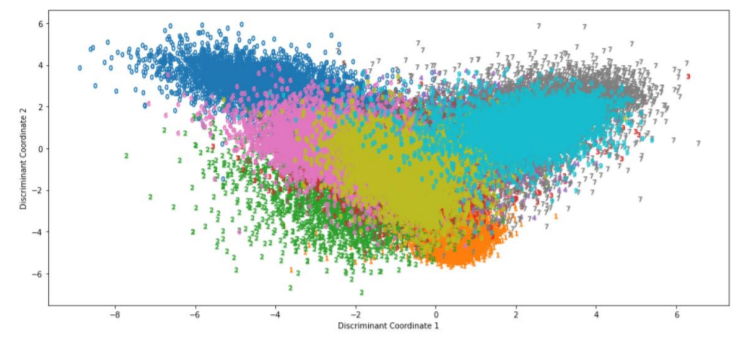
knn = KNeighborsClassifier(7)

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

print("Train score ",knn.score(X\_train,y\_train),"%")

print("Test score ",knn.score(X\_test,y\_test),"%")







**TABULAR INFERENCE**

**LINEAR DISCRIMINANT ANALYSIS**

|  |  |  |
| --- | --- | --- |
| **DATASET** | **Training Accuracy** | **Testing Accuracy** |
| **MNIST** | **97.63%** | **96.68%** |
| **BONE MINERAL DENSITY** | **92.59%** | **85.29%** |

**PRINCIPAL COMPONENT ANALYSIS**

|  |  |  |
| --- | --- | --- |
| **DATASET** | **Training Accuracy** | **Testing Accuracy** |
| **MNIST** | **98.5%** | **96.95%** |
| **BREAST CANCER DATASET** | **95.72%** | **92.98%** |

|  |  |  |  |
| --- | --- | --- | --- |
| **DATASET** | **Precision** | **Recall** | **F1-Score** |
| **MNIST** | **0.95** | **0.94** | **0.94** |
| **BREAST CANCER DATASET** | **0.92** | **0.93** | **0.93** |

**Inference: Thus, PCA and LDA are implemented and the performance metrics are recorded. The higher dimensional data has been visualized on a lower dimension.**