**Task one:**

A= (22, 1,42,10)

B= (20,0,36,8)

a) Euclidean Distance = √(a1-b1)2 + (a2-b2)2 +………(an-bn)2

= √ (22-20) 2 +(1-0) 2 + (42-36) 2 +(10-8) 2

= √ (2) 2 + (1) 2 +(6) 2 +(2) 2

= √ 4 +1+36+4

= √45

= 6.7082

b) Manhattan distance = |a1-b1|+ |a2-b2| + ………|an-bn|

= |22-20| + |1-0| + |42-36| + |10-8|

= |2| + |1| + |6| +|2|

= 11

d1 = {0,4,10,8,0,5,0}

d2 = {5,19,7,16,0,0,32}

Cosine Similarity = x.y / ||x|| \* ||y||

Perform dot product of x and y,

x.y = 0 x 5 + 4 x19 + 10 x 7 + 8 x 16 + 0 x 0 + 5 x 0 + 0 x 32

= 0 +76 +70 +128+0+0+0

= 274

Calculate ||x||

||x|| = √ 02 +42 +102 +82 +02+52

= √ 16+100+64+25

= √205

= 14.3178

Calculate ||y||

||y|| = √ 52 +192 +72 +162 +02+02 +322

= √ 25+361+49+256 + 1024

= √1715

= 41.4125

Putting values in the formula we get,

Cosine Similarity = 274/ (14.3178 x 41.4125)

= 0.4621

**Task two:**

1. Given information from the example is:

*p*( Disease = true ) = 0.01

*p*( Test = pos | Disease = true ) = 0.95

*p*( Test = neg | Disease = false ) = 0.90

*p*( Test = neg | Disease = true ) = 0.05

*p*( Test = pos | Disease = false ) = 0.10

*p*( Disease = false ) = 0.99

Bayes theorem is:

*p(A|B) = p(B|A) \* p(A) / p(B)*

Let’s denote event A = presence of disease

and B = positive test of disease

By using above information in the Bayes form:

*p*(Disease=True | Test=Pos) = *p*(Test=Pos| Disease=True) \* *p*(Disease =True) / *p*(Test=Pos)

-------(eq 1)

By using conditional probability principle,

*p*(Test=Pos) = *p*(Test=Pos | Disease =True) \* *p*(Disease =True) +

*p*(Test=Pos| Disease =False) \* *p*(Disease =False)

-------(eq 2)

We have *p*(Test = neg | Disease = false ) = 0.90 (true negative rate, or specificity) by using this we can calculate *p*(Test=Pos | Disease =False)

*p*(Test=Pos | Disease =False) = 1- p(Test = neg | Disease = false )

= 1- 0.90

= 0.1

We already  have below information,

*p*(Test = pos | Disease = true ) = 0.95

*p*(Disease = true ) = 0.01

*p*(Disease = false ) = 0.99

Putting above calculated values in the (eq 2),

*p*(Test=Pos) = (0.95 \*0.01) + (0.1\* 0.99)

= 0.0095 + 0.099

=0.1085

Now we have everything to put in (eq 1),

*p*(Disease=True | Test=Pos) = *p*(Test=Pos| Disease=True) \* *p*(Disease =True) / *p*(Test=Pos)

= 0.95\*0.01/ *p*(Test=Pos)

= 0.95\*0.01/0.1085

= 0.0095/0.1085

= 0.0875576

Hence, the probability you actually have the disease, given the test was positive is 0.0875576.

2.

*p*(Disease = true | Test = pos ) / *p*( Disease = true ) = 0.0875576 / 0.01

=8.75576

According to this person has 8.7% chance that he has disease if test is positive.

**Task three:**

Python programs perform PCA and plot the data:

**class PCA.py**

"""

This Class computes the PCA from scratch

@author Sayali Kudale

"""

import numpy as np

import matplotlib.pyplot as plt

"""

This method generates the random points

@param n , m

n: # dimentions

m: # points

"""

def generateRandom2DData(n, m):

random = np.random.RandomState(1)

uniform=random.rand(2, 2)

normal=random.randn(n, m)

X = np.dot(uniform,normal ).T

return X

"""

This method plot the matrix on graph

@param x

x: 2D matrix

"""

def plotOrigionalData(X):

fig, ax = plt.subplots(figsize = (12, 8))

ax.scatter(X[:, 0], X[:, 1],facecolors = 'b', edgecolors = 'b')

plt.title("Origional Data", fontsize = 15)

plt.xlabel("X")

plt.ylabel("Y")

"""

This method performs PCA algorithm

@param x

x: input matrix

"""

def performPca(X):

# mean centering

mean = np.mean(X, axis=0)

X\_Center = (X - mean)

# covariance matrix

cov = np.cov(X\_Center.T)

# eigen vectors

eig\_vals, eig\_vecs = np.linalg.eig(cov)

return X\_Center, eig\_vals,eig\_vecs

"""

This method sorts the eigen vectors in increasing order

@param eig\_vals, eig\_vec

eig\_vals: eigen values

eig\_vec: eigen Vector

"""

def sortEigenVectors(eig\_vals,eig\_vec):

sorted\_index = np.argsort(eig\_vals)[::-1]

sorted\_eigenvectors = eig\_vec[:,sorted\_index]

return sorted\_eigenvectors

"""

This method plots the PCA data and reconstructed data

@param X, X\_pca , X\_recon, twoDplot

X: origional matrix

X\_pca: matrix after PCA computation

X\_recon : reconstructed matrix after using PCA

twoDplot : whether PCA plot or reconstructed matrix plot

"""

def plotData(X,X\_pca, X\_recon,twoDplot=False):

plt.figure(figsize = (12, 8))

plt.xlabel('X')

plt.ylabel('Y')

if twoDplot == False:

plt.title("First Principle Component")

plt.plot(X\_pca, np.zeros\_like(X\_pca),'o',color = 'red')

else:

plt.title("Reconstruction: Dimensionality Reduced")

plt.xlabel('X')

plt.ylabel('Y')

plot = plt.scatter(X[:, 0], X[:, 1],facecolors = 'b', edgecolors = 'b',label = "Actual")

plot = plt.scatter(X\_recon[:, 0], X\_recon[:, 1],facecolors = 'k', edgecolors = 'k', label = 'Projected')

plt.legend()

"""

This method reduces the dimentions of origional data by using eigen vector information

@param X, eig\_vec , com

X: origional matrix

eig\_vec: eigen vector

com : # of components

"""

def reduceDimension(X, eig\_vec, com):

reduce\_mat = eig\_vec[:, :com]

z = np.dot(X, reduce\_mat)

return z

"""

This method recovers the origional points using reduced dimention matrix

@param z, eig\_vec , com

z: reduced matrix

eig\_vec: eigen vector

com : # of components

"""

def recoverX(z, eig\_vec, com):

reduce\_mat = eig\_vec[:, :com]

X\_rec = np.dot(z, reduce\_mat.T)

return X\_rec

"""

method to show the plotted data

"""

def show():

plt.show()

**Program main:**

""" This Class is the starting point of program and driver class

@author Sayali Kudale

"""

import pca as pca

import numpy as np

import matplotlib.pyplot as plt

n =2

m=1000

component=1

X = pca.generateRandom2DData(n,m)

pca.plotOrigionalData(X)

X\_Center, eig\_vals, eig\_vecs= pca.performPca(X)

sorted\_eigenvectors= pca.sortEigenVectors(eig\_vals,eig\_vecs)

z = pca.reduceDimension(X\_Center, sorted\_eigenvectors, component)

pca.plotData(X,z,X,twoDplot = False)

X\_recov = pca.recoverX(z, sorted\_eigenvectors, component)

pca.plotData(X\_Center,z,X\_recov,twoDplot = True)

pca.show()

**Report on the task of Principle component Analysis :**

The purpose of performing this task was to get the first principal component of the two-dimensional points in a space. After principal component is computed then by using this information points are reconstructed in the two-dimensional space.

Followed below steps to perform this task in using python:

**Step 1: Random generation of points**

The matrix of 2-dimensional points is randomly generated by using numpy random function. While generating this matrix positive linearity between the x and y co-ordinate of the point This is because it will generate the positive covariance and will achieve the direction of graph mentioned in a question. This matrix is plotted using matplotlib pyplot library. Figure 1 shows the generated points.

Chart, scatter chart

Description automatically generated

Figure 1: Original data

**Step 2: PCA Algorithm**

1. Data standardization:

To perform PCA first step is data standardization; mean is calculated in each dimension and it is substracted from original value to gather the data around 0. Below formula is used.

X\_Center= X- mean

1. Calculate covariance:

From covariance we can understand how values of each dimensions are dependent on each other. For above task we receive the positive covariance; it means that when value of x dimension increases, value of y dimension also increases and vice versa. This covariance is calculated using numpy library.

1. Calculate eigen values and eigen vectors:

Eigen values and Eigen vectors are calculated using the numpy library. These values will be used to calculate the PCA.

1. Sorting the Eigen Vectors:

Eigen vectors are sorted in descending order because most significant values should be taken as first component and so on.

1. Compute the principal component:

In this step most significant eigen values are taken and dimensions are reduced to form the principal component. The Figure 2 is the first principal component.

Chart

Description automatically generated with medium confidence

Figure 2: The first principal component

**Step 3: Reconstruct the original values:**

Matrix multiplication of centered matrix and principal component matrix is done to reconstruct the original data.

Chart, scatter chart

Description automatically generated

Figure 3: Reconstructed points after PCA

In the Figure 3 the original data is shown in blue color and reduced features after applying the PCA are denoted in black color. Data points in the black color are considered as the significant data points and other data points are removed.

From performing this task understanding of PCA is clearer. It is always preferrable to have most important, concise and significant features rather than a greater number of unsignificant features. This technique can be used to perform the features reduction without losing the important information from the dataset.

**Task four:**

**Python code for PCA implementation:**

**Class pca\_USPS.py**

""" This Class is the driver class to perform PCA on given dataset

@author Sayali Kudale

"""

import scipy.io

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.cm as cm

import pca as pca

component=10 # PCA component

imageId=2998 #This is the row number of required image in given 3000 rows

"""

This method to extract the images from dataset matrix

one image is extracted and converted into 16x16 matrix and plotted on a graph

@param mat, imageId

mat: dataset matrix

imageID: the row number in a matrix

returns: it plots the given image and returns the matrix

"""

def loadAndPlotImage(mat, imageId):

A=(mat['A'])

A1=A[imageId]

X=np.reshape(A1,(16,16))

fig = plt.figure()

ax1 = fig.add\_subplot(111)

ax1.imshow(X,cmap='gray')

return A

"""

This method converts the given matrix into 3000,256 format

and extract the image at specified post and plot that image dimension onto the graph

@param X\_recov

X\_recov : reduced matrix of image after applying PCA

returns: returns the image matrix in 16x16 dimension

"""

def plotReducedImage(X\_recov,imageId):

X\_reshape=np.reshape(X\_recov,(3000,256))

A1=X\_reshape[imageId]

X=np.reshape(A1,(16,16))

fig = plt.figure()

ax1 = fig.add\_subplot(111)

ax1.imshow(X,cmap='gray')

return X\_reshape

"""

This method calculates the recosntruction error using mean square error method

@param X, X\_reshape

X : origional matrix of 3000X256 dimension

X\_reshape: reconstructed matrix of original data

"""

def calReconstructionError(X,X\_reshape):

mse = np.mean((X - X\_reshape)\*\*2)

print("Reconstruction Error for PCA component {} is {}" .format(component,mse))

# load the file

mat = scipy.io.loadmat('USPS')

X=loadAndPlotImage(mat,imageId)

X\_Center, eig\_vals, eig\_vecs = pca.performPca(X)

sorted\_eigenvectors= pca.sortEigenVectors(eig\_vals,eig\_vecs)

z = pca.reduceDimension(X\_Center, sorted\_eigenvectors, component)

X\_recov = pca.recoverX(z, sorted\_eigenvectors, component)

X\_reshape=plotReducedImage(X\_recov,imageId)

# mean square error

calReconstructionError(X,X\_reshape)

pca.show()

**class PCA.py (same class is used in task no 3)**

"""

This Class computes the PCA from scratch

@author Sayali Kudale

"""

import numpy as np

import matplotlib.pyplot as plt

"""

This method generates the random points

@param n , m

n: # dimentions

m: # points

"""

def generateRandom2DData(n, m):

random = np.random.RandomState(1)

uniform=random.rand(2, 2)

normal=random.randn(n, m)

X = np.dot(uniform,normal ).T

return X

"""

This method plot the matrix on graph

@param x

x: 2D matrix

"""

def plotOrigionalData(X):

fig, ax = plt.subplots(figsize = (12, 8))

ax.scatter(X[:, 0], X[:, 1],facecolors = 'b', edgecolors = 'b')

plt.title("Origional Data", fontsize = 15)

plt.xlabel("X")

plt.ylabel("Y")

"""

This method performs PCA algorithm

@param x

x: input matrix

"""

def performPca(X):

# mean centering

mean = np.mean(X, axis=0)

X\_Center = (X - mean)

# covariance matrix

cov = np.cov(X\_Center.T)

# eigen vectors

eig\_vals, eig\_vecs = np.linalg.eig(cov)

return X\_Center, eig\_vals,eig\_vecs

"""

This method sorts the eigen vectors in increasing order

@param eig\_vals, eig\_vec

eig\_vals: eigen values

eig\_vec: eigen Vector

"""

def sortEigenVectors(eig\_vals,eig\_vec):

sorted\_index = np.argsort(eig\_vals)[::-1]

sorted\_eigenvectors = eig\_vec[:,sorted\_index]

return sorted\_eigenvectors

"""

This method plots the PCA data and reconstructed data

@param X, X\_pca , X\_recon, twoDplot

X: origional matrix

X\_pca: matrix after PCA computation

X\_recon : reconstructed matrix after using PCA

twoDplot : whether PCA plot or reconstructed matrix plot

"""

def plotData(X,X\_pca, X\_recon,twoDplot=False):

plt.figure(figsize = (12, 8))

plt.xlabel('X')

plt.ylabel('Y')

if twoDplot == False:

plt.title("First Principle Component")

plt.plot(X\_pca, np.zeros\_like(X\_pca),'o',color = 'red')

else:

plt.title("Reconstruction: Dimensionality Reduced")

plt.xlabel('X')

plt.ylabel('Y')

plot = plt.scatter(X[:, 0], X[:, 1],facecolors = 'b', edgecolors = 'b',label = "Actual")

plot = plt.scatter(X\_recon[:, 0], X\_recon[:, 1],facecolors = 'k', edgecolors = 'k', label = 'Projected')

plt.legend()

"""

This method reduces the dimentions of origional data by using eigen vector information

@param X, eig\_vec , com

X: origional matrix

eig\_vec: eigen vector

com : # of components

"""

def reduceDimension(X, eig\_vec, com):

reduce\_mat = eig\_vec[:, :com]

z = np.dot(X, reduce\_mat)

return z

"""

This method recovers the origional points using reduced dimention matrix

@param z, eig\_vec , com

z: reduced matrix

eig\_vec: eigen vector

com : # of components

"""

def recoverX(z, eig\_vec, com):

reduce\_mat = eig\_vec[:, :com]

X\_rec = np.dot(z, reduce\_mat.T)

return X\_rec

"""

method to show the plotted data

"""

def show():

plt.show()

**2 & 3. Results of implementation and Reconstruction Error:**

|  |  |  |
| --- | --- | --- |
|  | **Image** | **Reconstruction Error** |
| Original Image -1  (3000th image in the given dataset) | **Graphical user interface  Description automatically generated** |  |
| Image after applying PCA  Com p=10 | **Graphical user interface  Description automatically generated** | 0.5888828439108547 |
| Image after applying PCA  Com p=50 | **Graphical user interface  Description automatically generated** | 0.44002007124983206 |
| Image after applying PCA  Com p=100 | **Graphical user interface  Description automatically generated with medium confidence** | 0.40520359692881364 |
| Image after applying PCA  Com p=150 | **Graphical user interface  Description automatically generated** | 0.39347608954142044 |
| Image after applying PCA  Com p=200 | **Graphical user interface, application  Description automatically generated** | 0.388387974478863 |
| Original Image -2  (2999th image in the given dataset) | **Graphical user interface, application  Description automatically generated** |  |
| Image after applying PCA  Com p=10 | **Graphical user interface  Description automatically generated** | 0. 5888828439108547 |
| Image after applying PCA  Com p=50 | **Graphical user interface  Description automatically generated** | 0.44002007124983206 |
| Image after applying PCA  Com p=100 | **Graphical user interface, application  Description automatically generated** | 0.40520359692881364 |
| Image after applying PCA  Com p=150 | **Graphical user interface, application  Description automatically generated** | 0.39347608954142044 |
| Image after applying PCA  Com p=200 | **Graphical user interface, application  Description automatically generated** | 0.388387974478863 |

**4.Discussion on the task:**

In this task we performed PCA to reduce huge number of data points and still retain the meaningful information. During this PCA implementation data points which account for most variance is retained, and other data points are removed. The data points in final result are highly correlated to one another.

In this task the images used for visualization are of 16x16 dimensions. The After applying PCA we got the 256 eigen values and corresponding eigen vectors. These Eigen vectors are sorted in descending order so that the eigen vector which contain the high variance information can be selected as the first principal component and so on.

The first component contains the most variance and when we get all the variance information by including a greater number of PCA components then we can get the greater number of data points and remove unimportant data points that does not add value to the data. As shown in the above images even though we have removed the most of the unwanted data points, we can still see the number which is the important data point for this data.

The Mean square error (MSE) is used to evaluate the performance of PCA model. The MSE is calculated by taking mean of squares of the difference of original data and reconstructed data after applying PCA. MSE values are mentioned in above table for each component. From those values we can say that MSE decreases when number of components increases. This is because loss is decreases when number of components increases in PCA model. Hence, MSE is used to measure the amount of loss in reconstructed data of PCA and original data.

Overall, from above analysis it is clear that PCA is feature reduction algorithm used to retain only meaningful information in data and remove the too many features that are causing noise and complications while processing.