IE406 Machine Learning

Lab Assignment - 6

Group 28

201801015: Shantanu Tyagi
201801076: Shivani Nandani
201801407: Pratvi Shah
201801408: Arkaprabha Baerjee

Question 1

Given the following data use Principal Component Analysis to reduce the feature dimension from 3 to 1 also show eigen values. (Do manual calculation)

X_1	X_2	X_3
8	14	3
13	9	6
4	3	15
7	2	1
19	8	4
5	18	11

Answer

In order to reduce dimensions from 3 to 1 we need to follow the below mentioned steps:

1. Calculate the mean and standard deviation for each feature/column(Xi) of A where,

   $$A = \begin{bmatrix} 8 & 14 & 3 \\ 13 & 9 & 6 \\ 4 & 3 & 15 \\ 7 & 2 & 1 \\ 19 & 8 & 4 \\ 5 & 18 & 11 \end{bmatrix}$$

   $$M_i(\text{Mean}) = \frac{\sum_{j=1}^{6} A_{ji}}{6}$$

   $$S_i(\text{Standard deviation}) = \sqrt{\frac{\sum_{j=1}^{6}(A_{ji} - M_i)^2}{6}}$$

   M = [9.33 9.00 6.67]

   S = [5.185 5.6568 4.8534]

2. After we calculate the mean and variance of each column we standardize the elements of each column by subtracting the mean and dividing by the standard deviation

   $$B_i = \frac{A_i - M}{S} \text{ for all } i \in [1, 6]$$

$$B = \begin{bmatrix} -0.257 & 0.88 & -0.755 \\ 0.707 & 0.00 & -0.137 \\ -1.028 & -1.06 & 1.717 \\ -0.449 & -1.23 & -1.1675 \\ 1.864 & -0.177 & -0.549 \\ -0.835 & 1.591 & 0.892 \end{bmatrix}$$

3. After reducing the columns to zero mean we can move on to calculate the covariance matrix which for our case will be of 3x3 dimension as there are 3 features

   C = Covariance(B)

   $C_{ij} = \dfrac{B' \times B}{n-1}$ where n=6, $B' = transpose(B)$

   $$C = \begin{bmatrix} 1.2 & -0.0477 & -0.5827 \\ -0.0477 & 1.2 & 0.0947 \\ -0.5827 & 0.0947 & 1.2 \end{bmatrix}$$

4. Now we find the eigen vectors and eigen values corresponding to this covariance matrix C. This was calculated using numpy.linalg.eig(C) and were found to be:

   Eigen values = [1.79971072 0.61531314 1.18497614]

   $$\text{Eigen vectors} = \begin{bmatrix} -0.69401025 & 0.70241371 & -0.15800237 \\ 0.16586365 & -0.05756869 & -0.98446691 \\ 0.70059904 & 0.70943698 & 0.07655164 \end{bmatrix}$$

5. As we need to reduce the 3 dimension to a single one we just retain the eigen vector corresponding to the maximum eigen value, because higher the eigen value higher is the variance in the direction of that eigen vector. Therefore, we only retain the second eigen vector in order to form the projection matrix W:

   $$W = \begin{bmatrix} -0.69401025 \\ 0.16586365 \\ 0.70059904 \end{bmatrix}$$

6. Now in order to transform 3D to single dimension we project the standardized matrix(B) onto the projection matrix W:

   $$B \times W = \begin{bmatrix} -0.257 & 0.88 & -0.755 \\ 0.707 & 0.00 & -0.137 \\ -1.028 & -1.06 & 1.717 \\ -0.449 & -1.23 & -1.1675 \\ 1.864 & -0.177 & -0.549 \\ -0.835 & 1.591 & 0.892 \end{bmatrix}_{6\times3} \times \begin{bmatrix} -0.69401025 \\ 0.16586365 \\ 0.70059904 \end{bmatrix}_{3\times1} = \begin{bmatrix} -0.20423593 \\ -0.58697404 \\ 1.74081123 \\ -0.71095189 \\ -1.70802694 \\ 1.46937756 \end{bmatrix}_{6\times1}$$

7. Final answer of PCA = $$\begin{bmatrix} -0.20423593 \\ -0.58697404 \\ 1.74081123 \\ -0.71095189 \\ -1.70802694 \\ 1.46937756 \end{bmatrix}_{6\times1}$$

Observation/ Justification

For the above case we had to reduce the dimensions rom 3 to 1 hence only the eigen-vector with highest eigen values was chosen while constructing projection matrix. But had it been 3D to 2D transformation we would choose the eigen vectors corresponding to the 2 highest eigen-values i.e., eigen-vectors 1 and 3 in our case. In order to regenerate the 3D data points post PCA we can just do (ans$\times W^T$).

Question 2

A classic application of PCA is to project the 3-D point cloud onto a plane that could still retain the information or essence of the point distribution. Given a 3-D point cloud, estimate an optimal plane, onto which if the 3D data points when projected would still retain the essential information. We provide you with 5 different point clouds P_1.txt to P_5.txt containing the 3D coordinates of points in space. Perform PCA on these point clouds to obtain their projection on a 2D plane and visualize the results using python libraries (like matplotlib).

Answer

For this question we first load the 3D data points from each file. Then we standardize the data-points in order to perform principal component analysis. By performing PCA we reduce the 3D data to 2D by considering only those two features that cover maximum variance. Post this we regenerate the 3D data points by performing inverse transform to calculate the reconstruction error incurred due to feature selection(PCA).

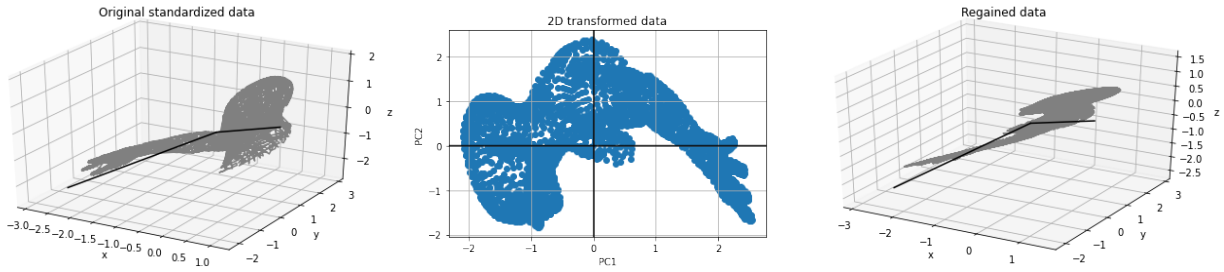
Code

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
%matplotlib inline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.metrics import mean_squared_error as mse
from mpl_toolkits import mplot3d

#Function to standardize data
def standardize(df1):
    scaler=StandardScaler()
    df1=scaler.fit_transform(df1)
    return pd.DataFrame(df1)

#Function to perform PCA
def pca_func(data):
  pca = PCA(n_components=2,svd_solver='full')
  pca=pca.fit(data)

  #Transform data to 2D
  data_encoded=pca.transform(data)

  #Regenerate 3D data from 2 features
  data_decoded=pd.DataFrame(pca.inverse_transform(data_encoded))
  comp=pca.components_

  #Principal components
  print('Components:')
  print(comp)

  #Plot original data
  fig = plt.figure(figsize=[8,5])
  ax = plt.axes(projection='3d')
  ax.plot3D(data[0],data[1], data[2],'gray')
  for i in range(len(pca.components_)):
      exp_var = pca.explained_variance_[i]
      component = pca.components_[i]
      p_component_with_length = component*3*np.sqrt(exp_var)
      PC = list(zip(pca.mean_,pca.mean_ + p_component_with_length))
      ax.plot(PC[0],PC[1],PC[2],'k')
  plt.title('Original normalized data')
  ax.set_xlabel('x')
  ax.set_ylabel('y')
  ax.set_zlabel('z')
  plt.show()

  #Plot regained data after performing PCA
  fig = plt.figure(figsize=[8,5])
  ax = plt.axes(projection='3d')
  ax.plot3D(data_decoded[0],data_decoded[1], data_decoded[2],'gray')
  for i in range(len(pca.components_)):
      exp_var = pca.explained_variance_[i]
```

```
        component = pca.components_[i]
        p_component_with_length = component*3*np.sqrt(exp_var)
        PC = list(zip(pca.mean_,pca.mean_ + p_component_with_length))
        ax.plot(PC[0],PC[1],PC[2],'k')
    plt.title('Regained data')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
    plt.show()

    #Plot 2D figure
    plt.scatter(data_encoded[:,0],data_encoded[:,1])
    plt.axhline(y=0,color='k')
    plt.axvline(x=0,color='k')
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.title('2D transformed data')
    plt.grid(True)
    plt.show()

    #Calculate error
    rmse = mse(data, data_decoded, squared=False)
    print('RMSE between original and regained data after PCA: ',rmse)


#Read data
data1 = pd.read_csv('P_1.txt', sep=" ", header=None)
data2 = pd.read_csv('P_2.txt', sep=" ", header=None)
data3 = pd.read_csv('P_3.txt', sep=" ", header=None)
data4 = pd.read_csv('P_4.txt', sep=" ", header=None)
data5 = pd.read_csv('P_5.txt', sep=" ", header=None)

#Standardize the data for performing PCA
data1=standardize(data1)
data2=standardize(data2)
data3=standardize(data3)
data4=standardize(data4)
data5=standardize(data5)
cols=data1.columns

### P_1
pca_func(data1)
###P_2
pca_func(data2)
###P_3
pca_func(data3)
###P_4
pca_func(data4)
###P_5
pca_func(data5)
```

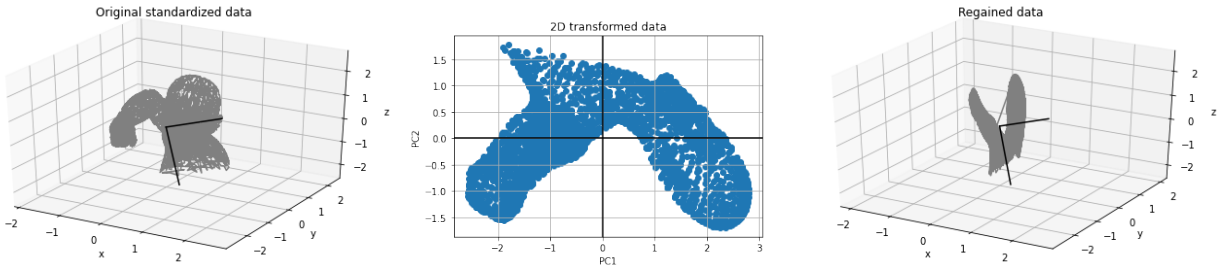
Listing 1: Question 2

Result



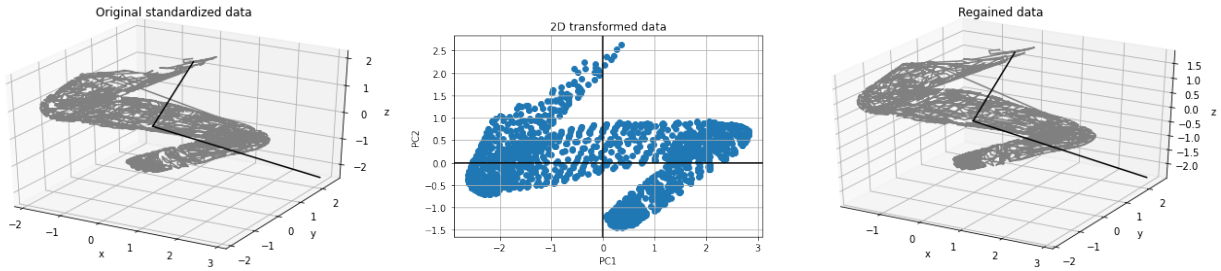
(a) (b) (c)

Figure 1: P_1.txt



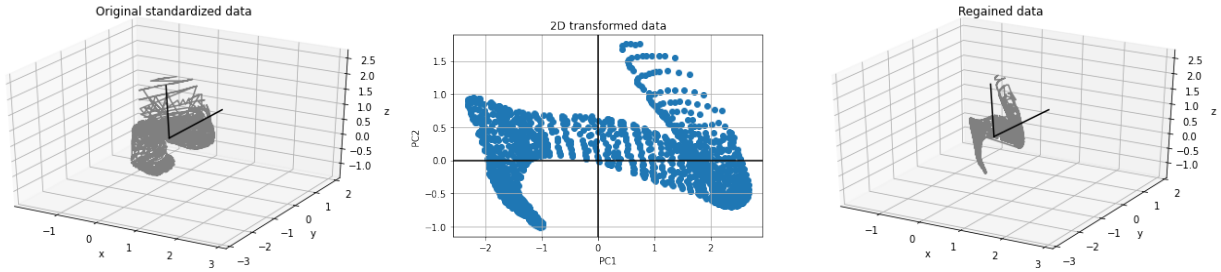
(a) (b) (c)

Figure 2: P_2.txt



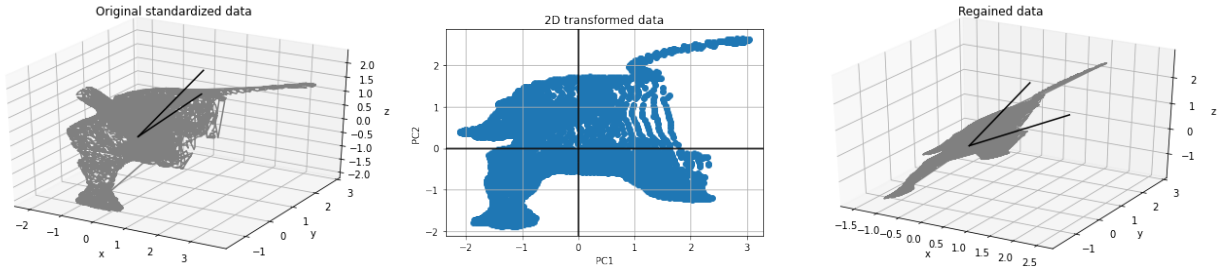
(a) (b) (c)

Figure 3: P_3.txt



(a) (b) (c)

Figure 4: P_4.txt



(a) (b) (c)

Figure 5: P_5.txt

In the above plots the black vectors represent the principal components. After reconstructing the final 3D plot post PCA we calculate the reconstruction error by evaluating RMSE and R^2 score for each point clouds.

File Name	RMSE	R^2 score
P_1	0.22275	0.95038
P_2	0.33047	0.89078
P_3	0.217206	0.952821
P_4	0.20048	0.959806
P_5	0.438327	0.807869

Table 1: RMSE and R^2 score

Observation/ Justification

We could draw the following observations:

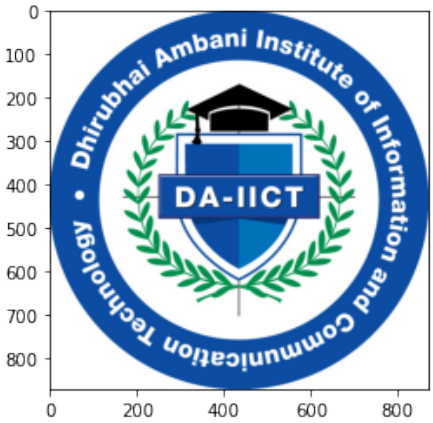
- The first principal component we obtain are majorly in descending order of the explained variance along that direction.
- The reconstructed 3D image post PCA shows that the points majorly lie on the plane formed by PC1 and PC2 and hence paves the way for the reconstruction error.
- RMSE values are low and R^2 values are considerable good(above 0.8) suggesting that performing PCA did not have a significant loss in information hence, indicating the goodness of fit.

Question 3

Another classic example of PCA comes in image compression. The human eye cannot perceive the minute and frequent changes in the image. A typical smartphone camera takes a 5MP image on average. We will utilize PCA and analyze how the reduction in the no. of principal components or dimensions affects the visual quality of the image. Given an image, apply PCA to investigate the effect of reducing the dimensions on the visual quality of the image. You will use the given image and show the analysis.

Answer

For this question we first load the image file. Then we split it into its RGB channels and then standardize the data in order to perform principal component analysis on each channel. By performing PCA we reduce the dimensions significantly by considering only those two features that cover maximum variance. Post this we regenerate the image by performing inverse transform to calculate the reconstruction error incurred due to feature selection(PCA).



(a)

Figure 6: Original image

Code

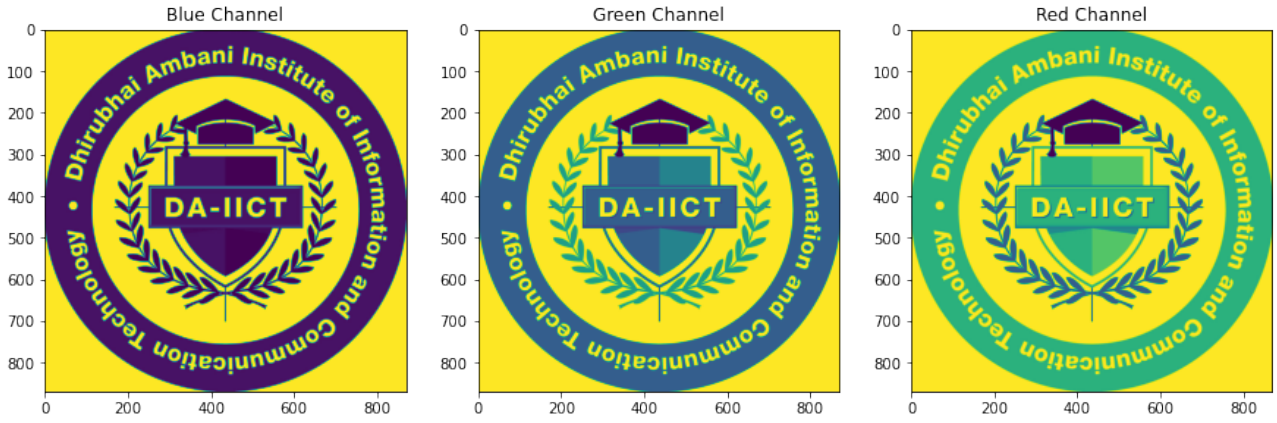
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import cv2
from scipy.stats import stats
import matplotlib.image as mpimg
from  sklearn.metrics  import  mean_squared_error  as mse

img = cv2.cvtColor(cv2.imread('da_image.jpg'), cv2.COLOR_BGR2RGB)
plt.imshow(img)
plt.show()
print(img.shape)

#Splitting into channels
blue,green,red = cv2.split(img)
```

```
18
19 # Plotting the images
20 fig = plt.figure(figsize = (15, 7.2))
21 fig.add_subplot(131)
22 plt.title("Blue Channel")
23 plt.imshow(blue)
24 fig.add_subplot(132)
25 plt.title("Green Channel")
26 plt.imshow(green)
27 fig.add_subplot(133)
28 plt.title("Red Channel")
29 plt.imshow(red)
30 plt.show()
31
32 # ensuring data is between 0 to 1
33 df_blue = blue/255
34 df_green = green/255
35 df_red = red/255
36
37 blue_variance_ratios = []
38 green_variance_ratios = []
39 red_variance_ratios = []
40
41 # Doing PCA for all 3 channels
42 for i in range(10,870,10):
43   pca_b = PCA(n_components=i)
44   pca_b.fit(df_blue)
45   pca_g = PCA(n_components=i)
46   pca_g.fit(df_green)
47   pca_r = PCA(n_components=i)
48   pca_r.fit(df_red)
49   trans_pca_b = pca_b.transform(df_blue)
50   trans_pca_g = pca_b.transform(df_green)
51   trans_pca_r = pca_b.transform(df_red)
52   blue_variance_ratios.append(sum(pca_b.explained_variance_ratio_))
53   green_variance_ratios.append(sum(pca_g.explained_variance_ratio_))
54   red_variance_ratios.append(sum(pca_r.explained_variance_ratio_))
55
56 x_line = []
57 for i in range(10,870,10):
58     x_line.append(i)
59 plt.title("Blue Channel")
60 plt.xlabel("Dimensions Considered")
61 plt.ylabel("Variance Explained")
62 plt.plot(x_line,blue_variance_ratios)
63 plt.grid()
64 plt.show()
65 plt.title("Green Channel")
66 plt.xlabel("Dimensions Considered")
67 plt.ylabel("Variance Explained")
68 plt.plot(x_line,green_variance_ratios,color='green')
69 plt.grid()
70 plt.show()
71 plt.title("Red Channel")
72 plt.xlabel("Dimensions Considered")
73 plt.ylabel("Variance Explained")
74 plt.plot(x_line,red_variance_ratios,color='red')
75 plt.grid()
76 plt.show()
77
78 def CompressAndDisplayImage(dimensions):
79   pca_b = PCA(n_components=dimensions)
80   pca_b.fit(df_blue)
81   trans_pca_b = pca_b.transform(df_blue)
82   pca_g = PCA(n_components=dimensions)
83   pca_g.fit(df_green)
84   trans_pca_g = pca_g.transform(df_green)
85   pca_r = PCA(n_components=dimensions)
86   pca_r.fit(df_red)
87   trans_pca_r = pca_r.transform(df_red)
88
89   print(f"Blue Channel Variance: {sum(pca_b.explained_variance_ratio_)}")
90   print(f"Green Channel Variance: {sum(pca_g.explained_variance_ratio_)}")
91   print(f"Red Channel Variance: {sum(pca_r.explained_variance_ratio_)}")
92
```

```
  b_arr = pca_b.inverse_transform(trans_pca_b)
  g_arr = pca_g.inverse_transform(trans_pca_g)
  r_arr = pca_r.inverse_transform(trans_pca_r)

  img_reduced= (cv2.merge((b_arr, g_arr, r_arr)))

  fig = plt.figure(figsize = (10, 7.2))
  plt.title(f'Post PCA ({dimensions} dimensions)')
  plt.imshow(img_reduced)
  plt.show()

  print(mse(df_blue , b_arr , squared=False))
  print(mse(df_green , g_arr , squared=False))
  print(mse(df_red , r_arr , squared=False))

CompressAndDisplayImage(200)
CompressAndDisplayImage(100)
CompressAndDisplayImage(50)
CompressAndDisplayImage(10)
CompressAndDisplayImage(5)
CompressAndDisplayImage(2)
```

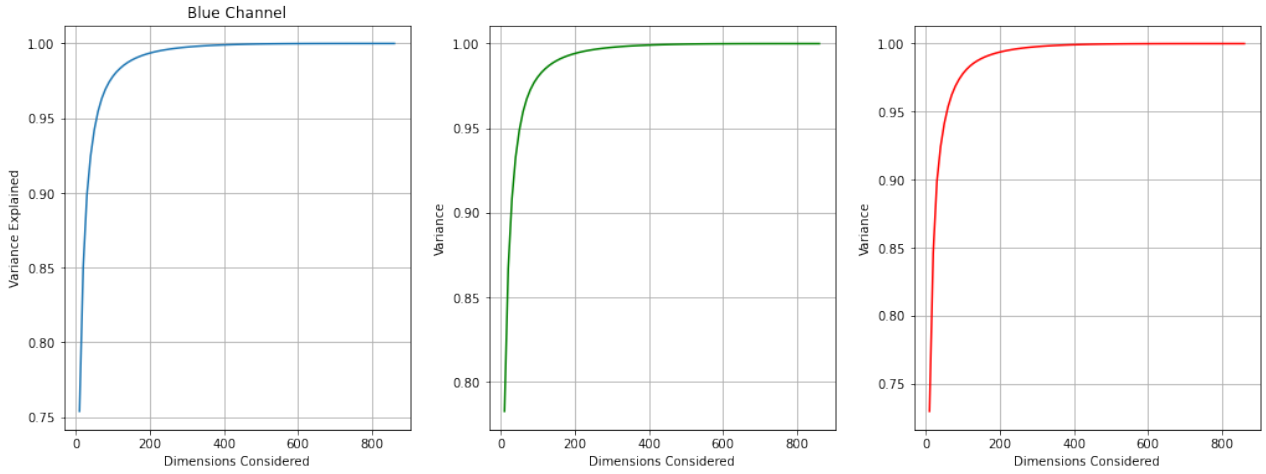
Listing 2: Question 3

Result



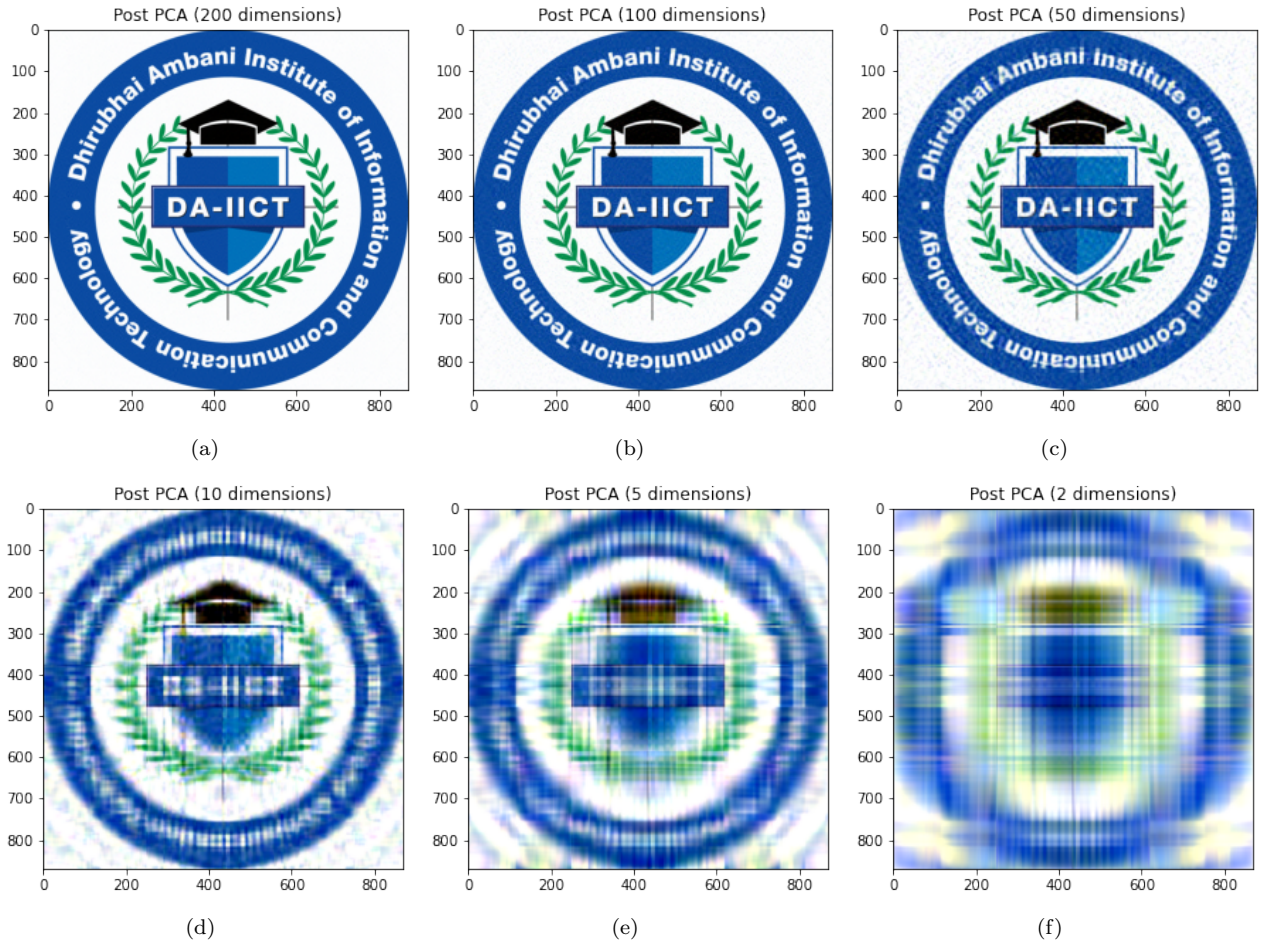
(a)

Figure 7: RGB channels of the image



(a) (b) (c)

Figure 8: Variance VS Dimensions plots of each color channel



(a) (b) (c)

(d) (e) (f)

Figure 9: Reconstructed images considering lesser dimensions

Dimensions	Blue Variance Explained	Green Variance Explained	Red Variance Explained
200	0.9935648226837049	0.9941616565664149	0.9937959902748918
100	0.9779432517901917	0.9802967864880041	0.977847744796052
50	0.9423032980246341	0.9487310465998462	0.9413689828250666
10	0.7538841839537315	0.7825686583827102	0.7297300272813159
5	0.628864418099541	0.6555120231603824	0.5761178869817015
2	0.3832129939545568	0.42068274651318016	0.33822686728695284

Table 2: Variance Explained

Dimensions	Blue RMSE	Green RMSE	Red RMSE
200	0.03628799070104658	0.025171460512205462	0.0175091265028994
100	0.0670588181377301	0.046157278873273956	0.03294640724137046
50	0.10835934101538819	0.07451383636470789	0.05323306621979239
10	0.22337024018093732	0.15282446891207815	0.11343895031462171
5	0.2756272089262313	0.19334849728941825	0.14168458679812998
2	0.35666386847049236	0.2529646395655778	0.1778214097607088

Table 3: RMSE

Observation/ Justification

We could draw the following observations:

- The first principal component we obtain are majorly in descending order of the explained variance along that direction.

- We observe that 100 dimensions give us nearly 98% variance on all color channels. As the dimensions are reduced below 50, we see a significant difference in the reconstructed image.
- RMSE values are low for dimensions more than 50 suggesting that performing PCA did not have a significant loss in information hence, indicating the goodness of fit.

Question 4

Load the Data matrix `faceimages.mat` given to you. There are 400 face images of size 112×92.

a Last column of the data is the class label. Column 1 to 10304 are 112×92 pixel.

b Each row represents one image.

c Construct the data matrix and Mean-center the data.

d Construct covariance matrix for the data.

e Solve eigenvalue problem to find projection matrix.

- Find low dimensional representation of Face images
- Draw energy curve of PCA for the given data
- Reconstruct any face image with 5, 10, 50 and 100 principal components.
- Compute the reconstruction error for the same.

Answer

For this question, we load the `faceimages.mat` and remove the labels column. The remaining data (400×10304) is mean-centered. Using inbulit functions, we calculate covariance, eigenvalues and eigenvectors. We then perform PCA for varying number of components using `sklearn.decomposition.PCA` and calculate the error (in terms of RMSE) for the reconstructed images.

Code

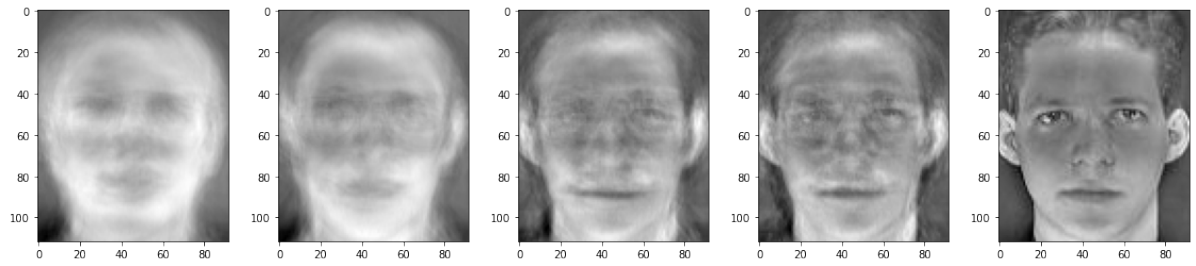
```
import h5py
import numpy as np
from numpy.core.einsumfunc import _parse_possible_contraction
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.base import _pprint
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

f = h5py.File('faceimages.mat', 'r')
data = f.get('data')
data = np.array(data).T
labels = data[:, -1]
data = data[:, :-1]

print(data.shape)

print(np.unique(labels))

data = data - np.mean(data, axis=0)
print(data.shape)

pca = PCA()
pca.fit_transform(data.T)
pca_variance_ratio = pca.explained_variance_ratio_
print(pca_variance_ratio.shape)

plt.figure(figsize=(8, 6))
plt.plot(pca_variance_ratio, 'b.--')
plt.ylabel('Explained Variance ratio')
plt.xlabel('Principal components')
plt.show()

```

```
cov_matrix = (np.dot(data.T, data))/(np.size(data, 0)-1)
print(cov_matrix.shape)

print(cov_matrix)

eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
print(eigenvalues.shape, eigenvectors.shape)

print(eigenvalues)

print(eigenvectors)


def n_compPCA(feat, n_comp):
    pca = PCA(n_components=n_comp)
    print('Explained variance ratio: ', sum(pca.explained_variance_ratio_))
    return pca


"""n_components = 5"""

pca = n_compPCA(data, 5)

trans_data = pca.inverse_transform(pca.fit_transform(data))

print(trans_data.shape)

plt.imshow(trans_data[0].reshape((92, 112)).T, cmap=plt.cm.gray)
plt.show()

reconstruction_error = mean_squared_error(data, trans_data, squared=False)
print(reconstruction_error)

"""n_components = 10"""

pca = n_compPCA(data, 10)

trans_data = pca.inverse_transform(pca.fit_transform(data))

print(trans_data.shape)

plt.imshow(trans_data[0].reshape((92, 112)).T, cmap=plt.cm.gray)
plt.show()

reconstruction_error = mean_squared_error(data, trans_data, squared=False)
print(reconstruction_error)

"""n_components = 50"""

pca = n_compPCA(data, 50)

trans_data = pca.inverse_transform(pca.fit_transform(data))

print(trans_data.shape)

plt.imshow(trans_data[0].reshape((92, 112)).T, cmap=plt.cm.gray)
plt.show()

reconstruction_error = mean_squared_error(data, trans_data, squared=False)
print(reconstruction_error)

"""n_components = 100"""

pca = n_compPCA(data, 100)

trans_data = pca.inverse_transform(pca.fit_transform(data))

print(trans_data.shape)

plt.imshow(trans_data[0].reshape((92, 112)).T, cmap=plt.cm.gray)
plt.show()

reconstruction_error = mean_squared_error(data, trans_data, squared=False)
print(reconstruction_error)

```

```
"""n_components = 400"""

pca = n_compPCA(data, 400)

trans_data = pca.inverse_transform(pca.fit_transform(data))

print(trans_data.shape)

plt.imshow(trans_data[0].reshape((92, 112)).T, cmap=plt.cm.gray)
plt.show()

reconstruction_error = mean_squared_error(data, trans_data, squared=False)
print(reconstruction_error)
```

Listing 3: Question 4

Result



(a) (b) (c) (d) (e)

Figure 10: Reconstructed images for varying `n_components`
(a) `n_components=5`, (b) `n_components=10`, (c) `n_components=50`, (d) `n_components=100`, (e) `n_components=400`

`n_components`	Explained Variance Ratio	RMSE
5	0.48052	28.40129
10	0.59959	24.93487
50	0.81583	16.91369
100	0.88995	13.06288
400	0.99998	1.265848×10^{-13}

Table 4: Explained Variance Ratio and RMSE

Observation/ Justification

We could draw the following observations:

- From Figure 11 and Table 4, it is clear that as the number of components increase, the error reduces and the reconstructed image is closer to the actual image before PCA.
- With 400 components, which is much less than half the total number of features, the reconstruction error is almost negligible.

Question 5

Apply PCA on the MNIST dataset. Plot cumulative sum of variance with no of components and find the minimum no. of component for 85% variance.

Answer

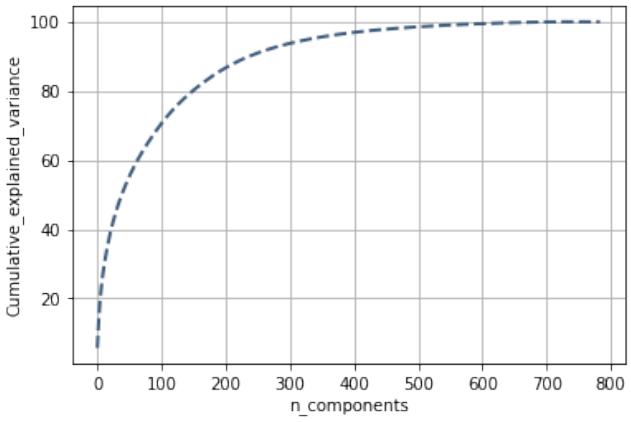
In this question the sklearn PCA function has been used to plot the cumulative variance plot and to find out the number of components required to explain 85% variance.

Code

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import fetch_openml
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import StandardScaler
from sklearn import decomposition

mnist = fetch_openml('mnist_784')
x = mnist.data
y = mnist.target
normalizedData = StandardScaler().fit_transform(x)
print(normalizedData.shape)

pca = decomposition.PCA()
pca.n_components = 784
pca_data = pca.fit_transform(normalizedData)
percentage_var_explained = pca.explained_variance_ / np.sum(pca.explained_variance_) * 100
cum_var_explained = np.cumsum(percentage_var_explained)

plt.figure(1, figsize=(6, 4))
plt.clf()
plt.plot(cum_var_explained, linewidth=2, linestyle='--',color = '#3F5D7D')
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()

pca = decomposition.PCA(n_components=0.85)
principalComponents = pca.fit_transform(normalizedData)
print("Net % Variance explained by Principal Components is :" + str(sum(pca.
    explained_variance_ratio_)*100))
print("No of components to explain " + str(85)+"% Variance is" ,pca.explained_variance_ratio_.
    shape)
```

Listing 4: Question 5

Result



(a)

Figure 11: Cumulative variance plot

Observation/ Justification

- One can observe the initial few principal components explain the most variance. Latter principal components explain very little variance.
- We require the first 186 principal components to explain 85%variance.