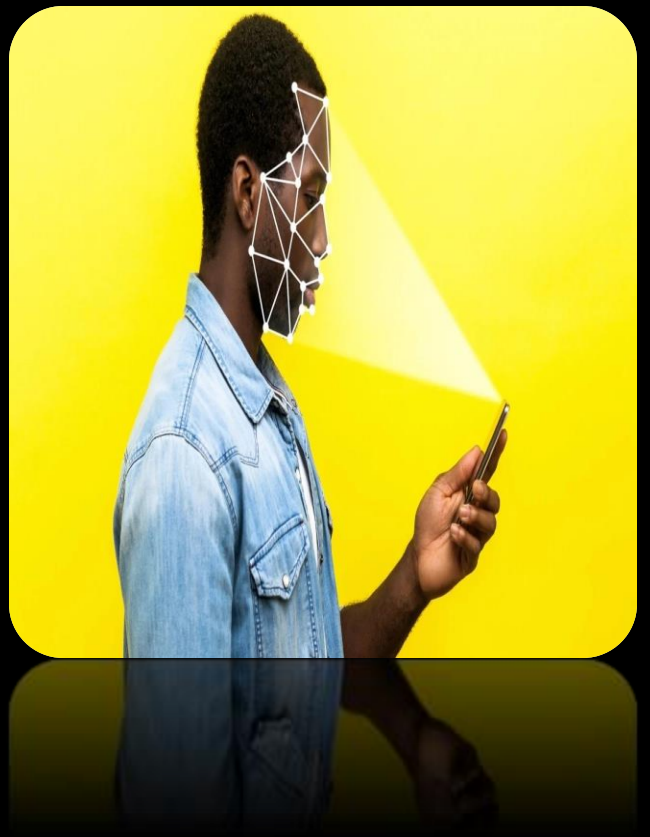
Facial Recognition

Ponnam Rahul

Contents

Introduction

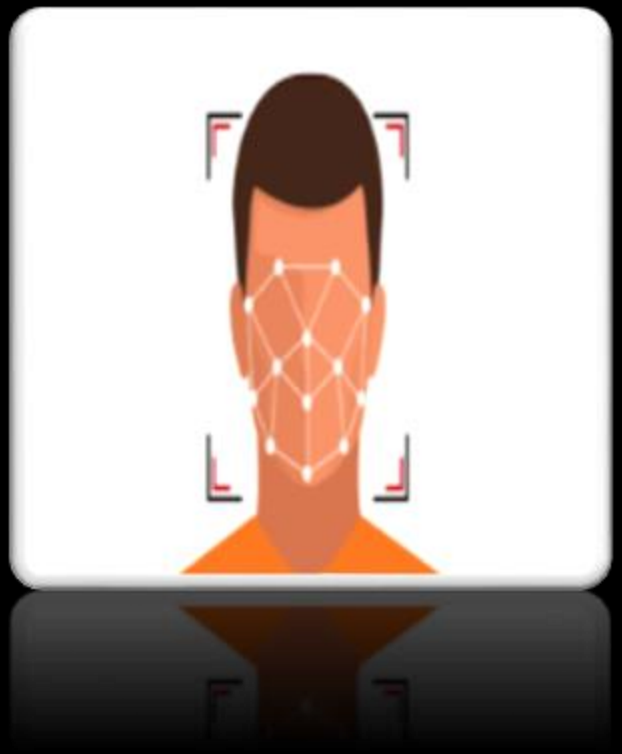
- Eigen faces have a wide range of applications beyond just facial recognition. They can be used to analyze and classify other types of images.
- In addition, eigen faces can be combined with other techniques such as neural networks to create even more powerful image recognition systems.



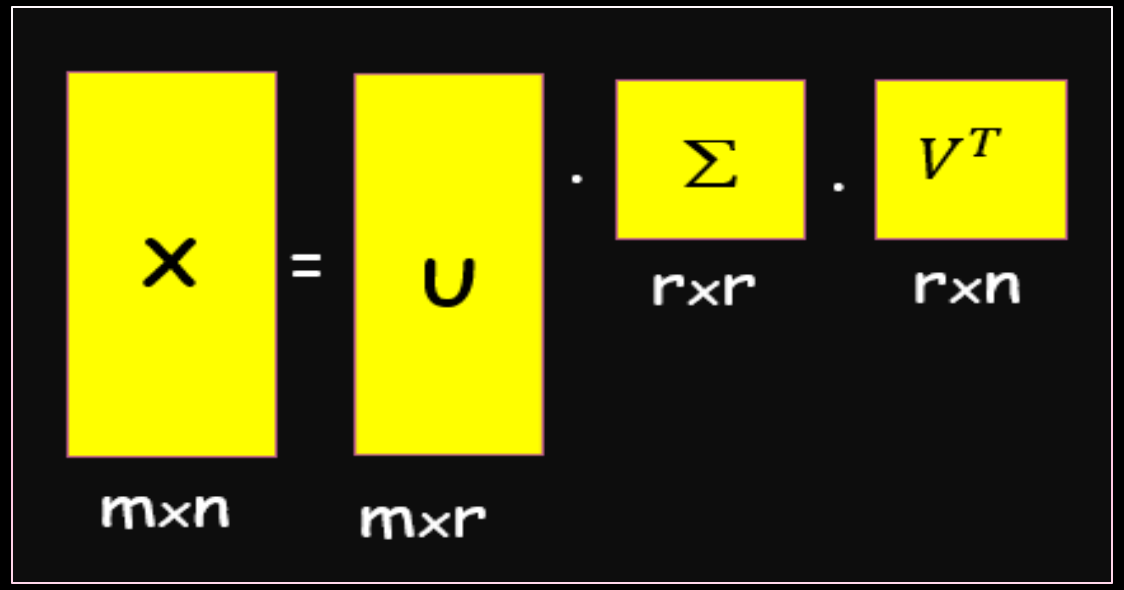
Introduction

- The basic idea behind eigen faces is to represent a set of images as linear combinations of other images in the same set.

- This allows for efficient storage and retrieval of facial data, as well as the ability to recognize faces even when they are presented in different orientations or lighting conditions.

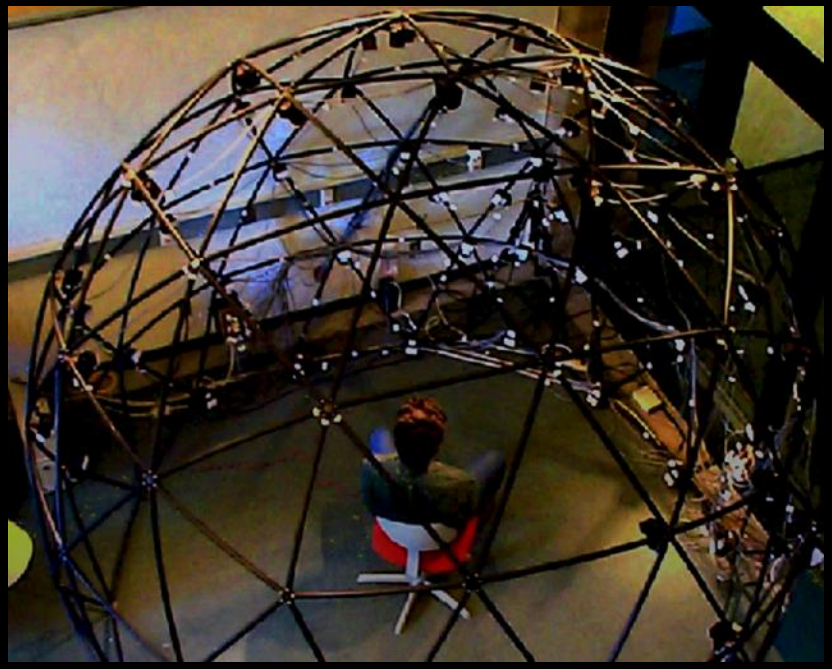


Singular Value Decomposition (SVD)

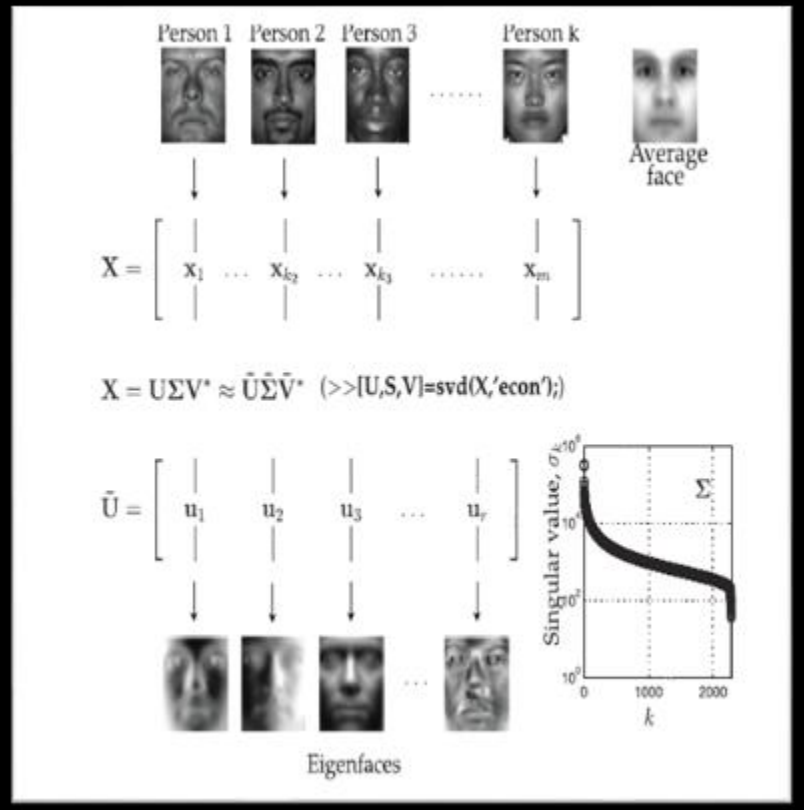


Eigen faces

- To use eigen faces for facial recognition, a set of training images must first be used to generate the eigen vectors and faces.
- The images have been captured for 38 different people from 64 different angles.
- Face images are then being stretched to column vectors and all face-vectors are horizontally concatenated to construct a face matrix.



- The eigen vectors are sorted in order of descending eigen values.
- This reduces the dimensionality of the image space.
- Any face image can be represented as a vector of coefficients, corresponding to the contribution of each eigen face.
- To perform Principal Component Analysis (PCA), an average face needs to be calculated by finding the mean of all dimensions.



Code and explanation (Training data set)

```python
import matplotlib.pyplot as plt
import numpy as np
import os
import scipy.io
plt.rcParams['figure.figsize'] = [10, 10]
plt.rcParams.update({'font.size': 18})

mat_contents = scipy.io.loadmat(os.path.join('..','DATA','allFaces.mat'))
faces = mat_contents['faces']
m = int(mat_contents['m'])
n = int(mat_contents['n'])
nfaces = np.ndarray.flatten(mat_contents['nfaces'])

allPersons = np.zeros((n*6,m*6))
count = 0

for j in range(6):
    for k in range(6):
        allPersons[j*n : (j+1)*n, k*m : (k+1)*m] = np.reshape(faces[:,np.sum(nfaces[:count])],(m,n)).T
        count += 1

img = plt.imshow(allPersons)
img.set_cmap('gray')
plt.axis('off')
plt.show()
```

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  import os
4  import scipy.io
5  plt.rcParams['figure.figsize'] = [10, 10]
6  plt.rcParams.update({'font.size': 18})
```

- The first 4 lines in the code are libraries used and the next two lines are the dimensions of images.

```
8  mat_contents = scipy.io.loadmat(os.path.join('..','DATA','allFaces.mat'))
9  faces = mat_contents['faces']
10 m = int(mat_contents['m'])
11 n = int(mat_contents['n'])
12 nfaces = np.ndarray.flatten(mat_contents['nfaces'])
```

- The above 4 lines extract the data from data set as nxm matrices.

```
14  allPersons = np.zeros((n*6,m*6))
15  count = 0
16
17  for j in range(6):
18      for k in range(6):
19          allPersons[j*n : (j+1)*n, k*m : (k+1)*m] = np.reshape(faces[:,np.sum(nfaces[:count])],(m,n)).T
20          count += 1
21
22  img = plt.imshow(allPersons)
23  img.set_cmap('gray')
24  plt.axis('off')
25  plt.show()
```

- The first 36 persons faces has been taken in the form of 6x6 matrix to train the data.
- For arranging the faces in proper ascending order the loops has been used.
- The last four lines will show the image of all persons in grayscale.

output

- These are the images of 36 different persons from same angle (6x6 matrix format).

Code and explanation (Training data set)

```
for person in range(len(nfaces)):
    subset = faces[:,sum(nfaces[:person]) : sum(nfaces[:(person+1)])]
    allFaces = np.zeros((n*8,m*8))

    count = 0

    for j in range(8):
        for k in range(8):
            if count < nfaces[person]:
                allFaces[j*n:(j+1)*n,k*m:(k+1)*m] = np.reshape(subset[:,count],(m,n)).T
                count += 1

    img = plt.imshow(allFaces)
    img.set_cmap('gray')
    plt.axis('off')
    plt.show()
```

```
for person in range(len(nfaces)):
    subset = faces[:,sum(nfaces[:person]) : sum(nfaces[:(person+1)])]
    allFaces = np.zeros((n*8,m*8))

    count = 0
```

- The above 5 lines will extract the 64 images of each person in matrix 8x8 format.

```
    for j in range(8):
        for k in range(8):
            if count < nfaces[person]:
                allFaces[j*n:(j+1)*n,k*m:(k+1)*m] = np.reshape(subset[:,count],(m,n)).T
                count += 1

    img = plt.imshow(allFaces)
    img.set_cmap('gray')
    plt.axis('off')
    plt.show()
```

- The above loop is used to rearrange the data of all 64 images in ascending order and also to show faces.

output

- Here we get total of 36 matrices of 8x8 matrix, the first three matrices are shown.

Principal Component Analysis (PCA)

- Principal component analysis (PCA) is a technique for analyzing large datasets containing a high number of dimensions/features per observation, increasing the interpretability of data while preserving the maximum amount of information.
- Using PCA wea can plot the data of the faces and we can compare the faces.
- First we will find the **Mean Subtracted Matrix** from **SVD**, **B = X – Xi**.
- Then we will use **Covariance C = B′B**, and **Principal Component Value U = BV.**

Code and explanation (Training data set)

```
 1  import matplotlib.pyplot as plt
 2  import numpy as np
 3  import os
 4  import scipy.io
 5  plt.rcParams['figure.figsize'] = [8, 8]
 6  plt.rcParams.update({'font.size': 18})
 7  mat_contents = scipy.io.loadmat(os.path.join('..','DATA','allFaces.mat'))
 8  faces = mat_contents['faces']
 9  m = int(mat_contents['m'])
10  n = int(mat_contents['n'])
11  nfaces = np.ndarray.flatten(mat_contents['nfaces'])
12  # We use the first 36 people for training data
13  trainingFaces = faces[:,:np.sum(nfaces[:36])]
14  avgFace = np.mean(trainingFaces,axis=1) # size n*m by 1
15  # Compute eigenfaces on mean-subtracted training data
16  X = trainingFaces - np.tile(avgFace,(trainingFaces.shape[1],1)).T
17  U, S, VT = np.linalg.svd(X,full_matrices=0)
18  fig1 = plt.figure()
19  ax1 = fig1.add_subplot(121)
20  img_avg = ax1.imshow(np.reshape(avgFace,(m,n)).T)
21  img_avg.set_cmap('gray')
22  plt.axis('off')
23  ax2 = fig1.add_subplot(122)
24  img_u1 = ax2.imshow(np.reshape(U[:,0],(m,n)).T)
25  img_u1.set_cmap('gray')
26  plt.axis('off')
27  plt.show()
```

```
1  import matplotlib.pyplot as plt
2  import numpy as np
3  import os
4  import scipy.io
5  plt.rcParams['figure.figsize'] = [8, 8]
6  plt.rcParams.update({'font.size': 18})
7  mat_contents = scipy.io.loadmat(os.path.join('..','DATA','allFaces.mat'))
8  faces = mat_contents['faces']
9  m = int(mat_contents['m'])
10 n = int(mat_contents['n'])
11 nfaces = np.ndarray.flatten(mat_contents['nfaces'])
```

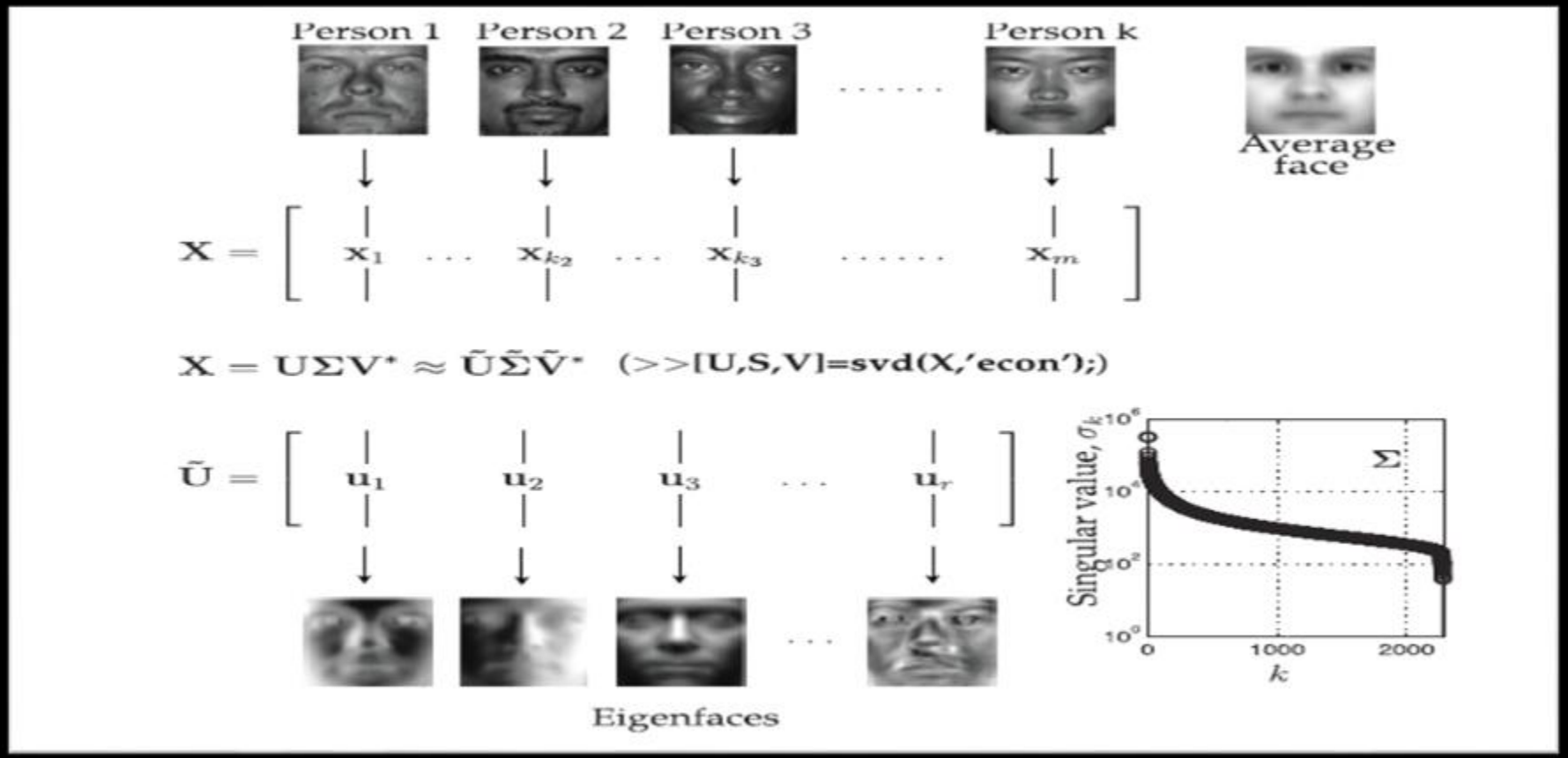
- The above lines are used to extract the date from data set (Yale B data set) and to arrange the faces in matrix format(8x8).

```
12 # We use the first 36 people for training data
13 trainingFaces = faces[:,:np.sum(nfaces[:36])]
14 avgFace = np.mean(trainingFaces,axis=1) # size n*m by 1
```

- Now we are training the first 36 persons images from all 64 angles to obtain the average face.

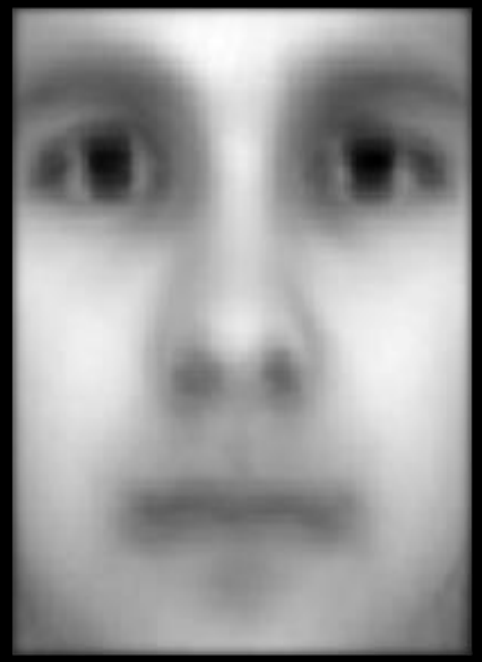
```
15  # Compute eigenfaces on mean-subtracted training data
16  X = trainingFaces - np.tile(avgFace,(trainingFaces.shape[1],1)).T
17  U, S, VT = np.linalg.svd(X,full_matrices=0)
18  fig1 = plt.figure()
19  ax1 = fig1.add_subplot(121)
20  img_avg = ax1.imshow(np.reshape(avgFace,(m,n)).T)
21  img_avg.set_cmap('gray')
22  plt.axis('off')
23  ax2 = fig1.add_subplot(122)
24  img_u1 = ax2.imshow(np.reshape(U[:,0],(m,n)).T)
25  img_u1.set_cmap('gray')
26  plt.axis('off')
27  plt.show()
```

- Here we are computing the eigen faces on mean subtracted training data using singular value decomposition(SVD).
- From this code we will find and plot both the average face and eigen face of 36 persons.


Person 1
Person 2
Person 3
Person k
Average face
$\mathbf{X} = \begin{bmatrix} | & | & | & | \\ \mathbf{x}_1 & \cdots & \mathbf{x}_{k_2} & \cdots & \mathbf{x}_{k_3} & \cdots & \mathbf{x}_m \\ | & | & | & | \end{bmatrix}$
$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^*$ (>>[U,S,V]=svd(X,'econ');)
$\tilde{\mathbf{U}} = \begin{bmatrix} | & | & | & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \mathbf{u}_3 & \cdots & \mathbf{u}_r \\ | & | & | & | \end{bmatrix}$
Eigenfaces
Singular value, σ_k
Σ
10^6
10^4
10^2
10^0
0
1000
2000
k

output

- This is the average face of 36 matrices of 8x8 matrix.
- To perform Principal Component Analysis (PCA), an average face needs to be calculated by finding the mean of all dimensions.
- The average face can be reshaped as an image after computing.



Code and explanation (Testing data set)

```
## Now show eigenface reconstruction of image that was omitted from test set

testFace = faces[:,np.sum(nfaces[:36])] # First face of person 37
plt.imshow(np.reshape(testFace,(m,n)).T)
plt.set_cmap('gray')
plt.title('Original Image')
plt.axis('off')
plt.show()

testFaceMS = testFace - avgFace
r_list = [25, 50, 100, 200, 400, 800, 1600]

for r in r_list:
    reconFace = avgFace + U[:,:r]  @ U[:,:r].T @ testFaceMS
    img = plt.imshow(np.reshape(reconFace,(m,n)).T)
    img.set_cmap('gray')
    plt.title('r = ' + str(r))
    plt.axis('off')
    plt.show()
```

```
## Now show eigenface reconstruction of image that was omitted from test set

testFace = faces[:,np.sum(nfaces[:36])] # First face of person 37
plt.imshow(np.reshape(testFace,(m,n)).T)
plt.set_cmap('gray')
plt.title('Original Image')
plt.axis('off')
plt.show()
```

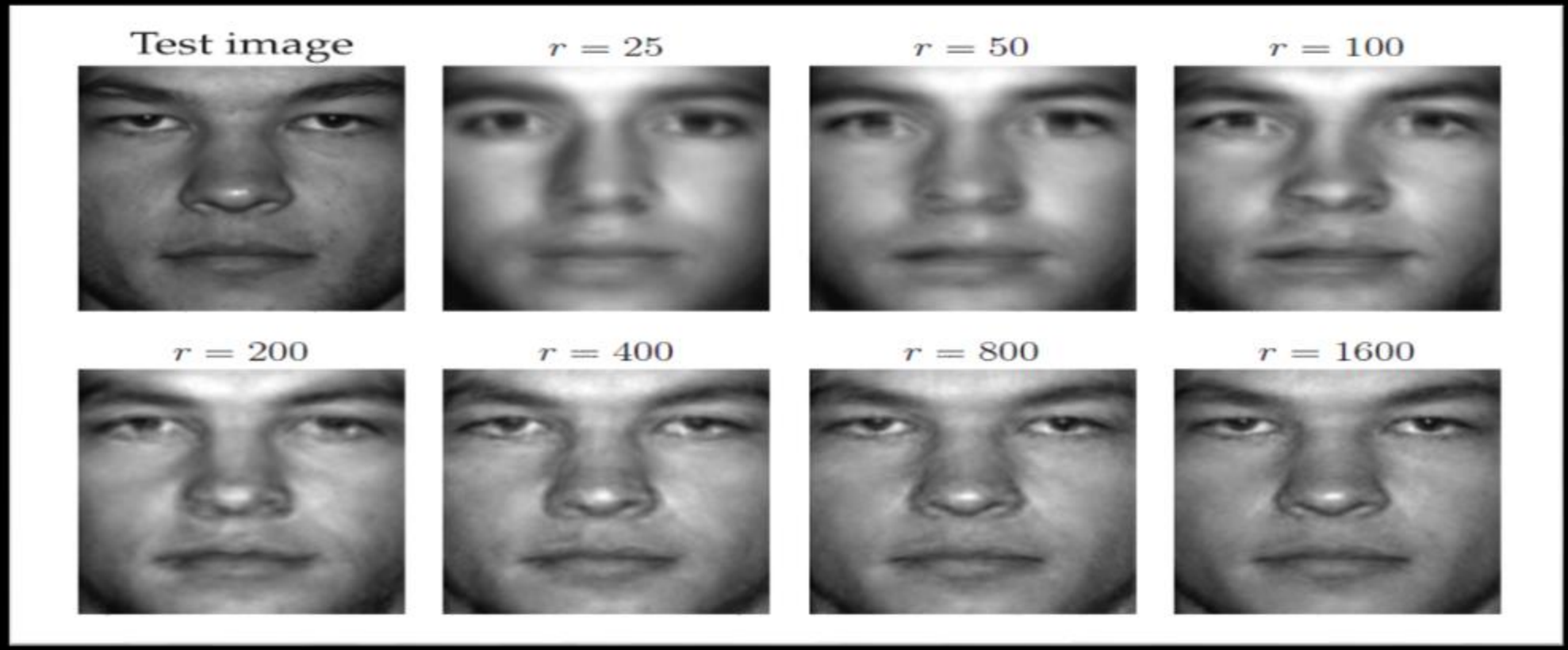
- Here we are testing the image of 37th person by using the average face, which is the output of previous data.

```
testFaceMS = testFace - avgFace
r_list = [25, 50, 100, 200, 400, 800, 1600]

for r in r_list:
    reconFace = avgFace + U[:,:r]  @ U[:,:r].T @ testFaceMS
    img = plt.imshow(np.reshape(reconFace,(m,n)).T)
    img.set_cmap('gray')
    plt.title('r = ' + str(r))
    plt.axis('off')
    plt.show()
```

- We are testing the image at different values of **r** (order of image), as we increase the value of **r**, we get more and more accurate image.

output



Code and explanation (Testing data set)

```
## Project person 2 and 7 onto PC5 and PC6

P1num = 2 # Person number 2
P2num = 7 # Person number 7

P1 = faces[:,np.sum(nfaces[:(P1num-1)]):np.sum(nfaces[:P1num])]
P2 = faces[:,np.sum(nfaces[:(P2num-1)]):np.sum(nfaces[:P2num])]

P1 = P1 - np.tile(avgFace,(P1.shape[1],1)).T
P2 = P2 - np.tile(avgFace,(P2.shape[1],1)).T

PCAmodes = [5, 6] # Project onto PCA modes 5 and 6
PCACoordsP1 = U[:,PCAmodes-np.ones_like(PCAmodes)].T @ P1
PCACoordsP2 = U[:,PCAmodes-np.ones_like(PCAmodes)].T @ P2

plt.plot(PCACoordsP1[0,:],PCACoordsP1[1,:],'d',Color='k',label='Person 2')
plt.plot(PCACoordsP2[0,:],PCACoordsP2[1,:],'^',Color='r',label='Person 7')

plt.legend()
plt.show()
```

```
## Project person 2 and 7 onto PC5 and PC6

P1num = 2 # Person number 2
P2num = 7 # Person number 7

P1 = faces[:,np.sum(nfaces[:(P1num-1)]):np.sum(nfaces[:P1num])]
P2 = faces[:,np.sum(nfaces[:(P2num-1)]):np.sum(nfaces[:P2num])]

P1 = P1 - np.tile(avgFace,(P1.shape[1],1)).T
P2 = P2 - np.tile(avgFace,(P2.shape[1],1)).T
```

- Here we are taking the 2 persons from the **X** matrix (person 2 and 7)and making their individual average face.

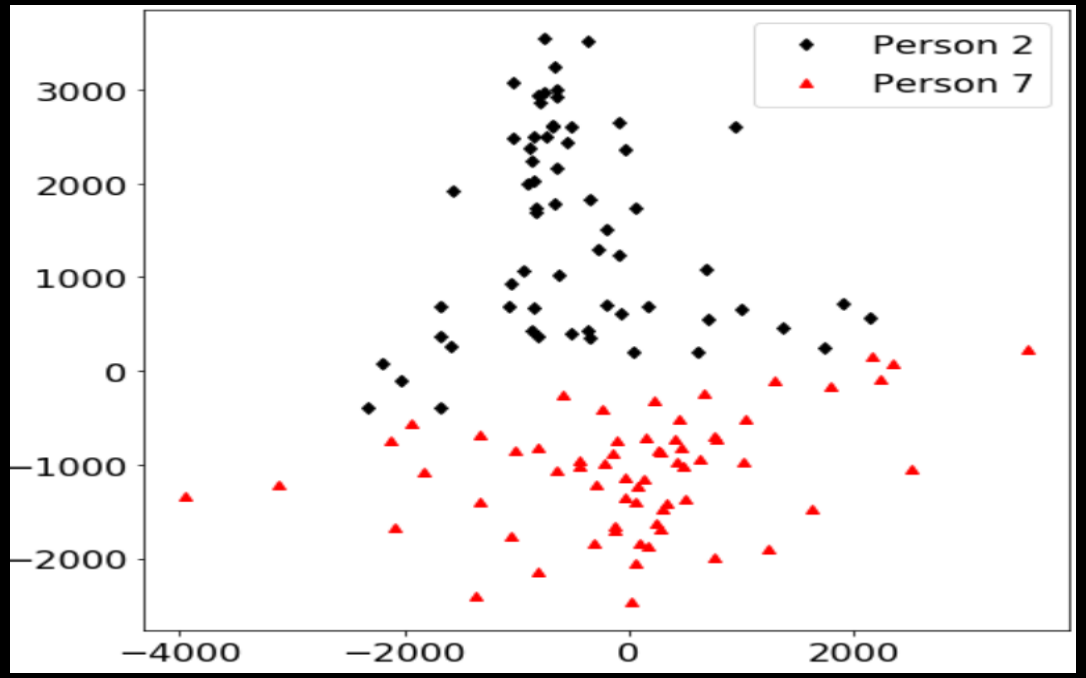
```
PCAmodes = [5, 6] # Project onto PCA modes 5 and 6
PCACoordsP1 = U[:,PCAmodes-np.ones_like(PCAmodes)].T @ P1
PCACoordsP2 = U[:,PCAmodes-np.ones_like(PCAmodes)].T @ P2

plt.plot(PCACoordsP1[0,:],PCACoordsP1[1,:],'d',Color='k',label='Person 2')
plt.plot(PCACoordsP2[0,:],PCACoordsP2[1,:],'^',Color='r',label='Person 7')

plt.legend()
plt.show()
```

- After taking the images of person 2 and 7, we are plotting the data of images into PCA 5 and PCA 6 (5^{th} , 6^{th} column of **U**)

output



Person 2

Person 7

- The plot is against PCA 5 vs PCA 6 of person 2 and 7, if we compare the facial images from any angle of these two persons we get similar kind of separation.

- Here we can see the data was separated (from graph), so from this we can say that the images are completely different.

- But if consider the data of twins, we will not have a clear separation of data, although there might be small separations.

- The data set and source code has been take from Yale B data set and databookuw.com.

- 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.