CV Assignment 3

Bayes Implementation of EigenFaces for Face Recognition

Ritu Ann Roy George - B170106EC

In [7]:

```
import os
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import cv2
import matplotlib.image as mpimg
import glob
%matplotlib inline
```

Extract and Vectorise Images from Yale Database

In [8]:

```
#collect images as classes based on the person ie 15 classes
def yale_db():
    img1 = []
    all_images=[]
    store_location = glob.glob("D:/AnacondaProjects/yalefaces/yalefaces_asgt2/subject*")
    for loc in store_location:
            image = mpimg.imread(loc)
            image = cv2.resize(image,(50,50))
            img1.append(image)
    img = np.array(img1)
    img_train_class1 = []
    img_test1 = []
    for i in range(15):   #15 ppl
        img_train1 = []
        for j in range(i,i+10): #10 training images for 1 person
            img_train1.append(img[10*i+j])
        img_train=np.array(img_train1)
        img_train_class1.append(img_train)
        img_test1.append(img[10*i+10+i])
    img_train_class=np.array(img_train_class1)
    img_test=np.array(img_test1)
    return img_train_class,img_test

def vectorize_class_allimg(img):
    if len(img) == 0:
        return np.array([])
    return np.reshape(img,(img.shape[0]*img.shape[1],img.shape[2]*img.shape[3]))
```

```
def vectorize_class(img):
    if len(img) == 0:
        return np.array([])
    return np.reshape(img,(img.shape[0],img.shape[1],img.shape[2]*img.shape[3]))

def vectorize_img(img):
    if len(img) == 0:
        return np.array([])
    return np.reshape(img,(img.shape[0],img.shape[1]*img.shape[2]))

img_train_class_orig,img_test_orig=yale_db()
img_train=vectorize_class_allimg(img_train_class_orig)
img_train_class=vectorize_class(img_train_class_orig)
img_test=vectorize_img(img_test_orig)
```

PCA and EigenFaces Computation

In [9]:

```
dif = []
for clas in img_train_class:
    for i in range(len(clas)):
        for j in range(i,len(clas)):
            dif.append(clas[i] - clas[j])
dif=np.array(dif)
mean_dif=0

mean_i = np.mean(dif,axis=0)
std_i = np.std(dif,axis=0)
c = []
for i in dif:
    c.append((i-mean_i)/std_i)
c = np.array(c)
ncovar = np.cov(np.transpose(c))
evalu, evect = np.linalg.eig(ncovar)

indices = np.argsort (- evalu)
evalu  = evalu[indices]
evect = evect[:, indices]
```

Prediction

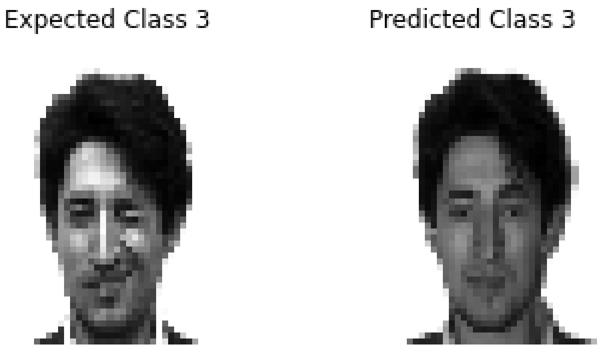
In [10]:

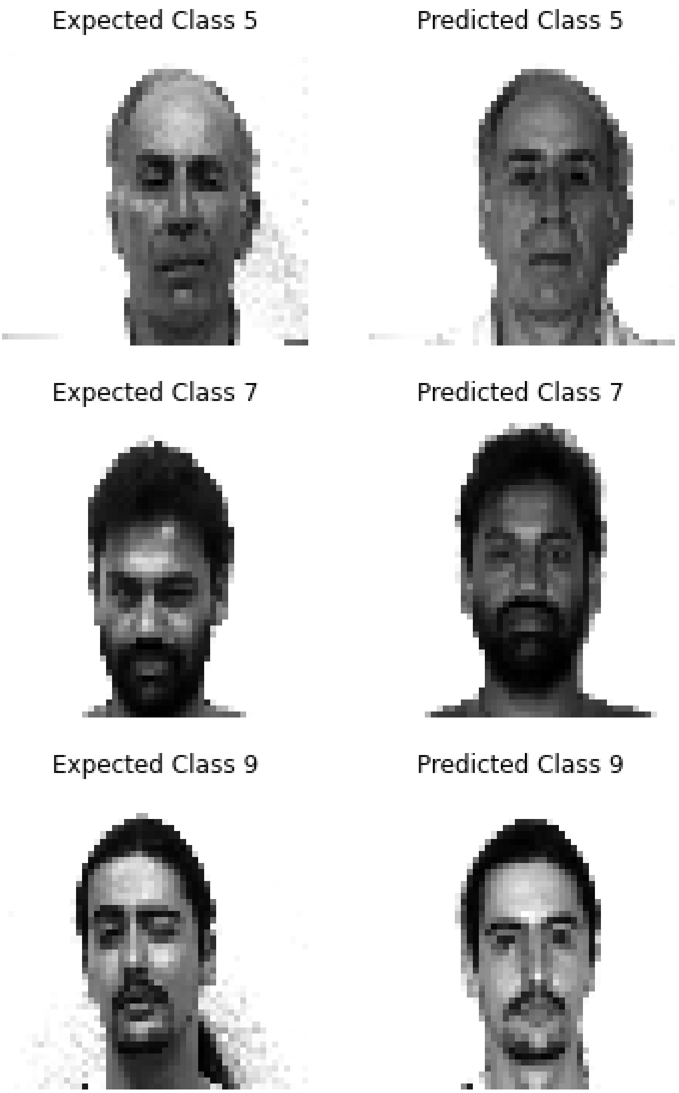
```
N = 20

def coef(img):
    return (np.power(evalu[:N],-1/2) * (evect.T[:N] * img).T).T
```

```
eigface_set = []
for clas in img_train_class:
    eigface = 0
    for i in range(len(clas)):
        eigface += coef(clas[i]) / len(clas)
    eigface_set.append(eigface)
eigface_set = np.array(eigface_set)

def predict(img):
    allps = []
    for i in range(eigface_set.shape[0]):
        pr = np.exp(- 0.5 * np.linalg.norm(coef(img)- eigface_set[i,:,:]))
        allps.append(pr)
    pred = np.argmax(allps) + 1
    return pred
```

In [6]:

```
for k in range(2,img_test.shape[0],2):
    plt.subplot(1,2,1)
    plt.imshow(np.reshape(img_test[k],(50,50)),cmap='gray')
    plt.title('Expected Class {}'.format(k+1))
    plt.axis('off')
    plt.subplot(1,2,2)
    c=predict(img_test[k])
    plt.imshow(np.reshape(img_train_class[c-1][0],(50,50)),cmap='gray')
    plt.title('Predicted Class {}'.format(c))
    plt.axis('off')
    plt.show()
```





Expected Class 5

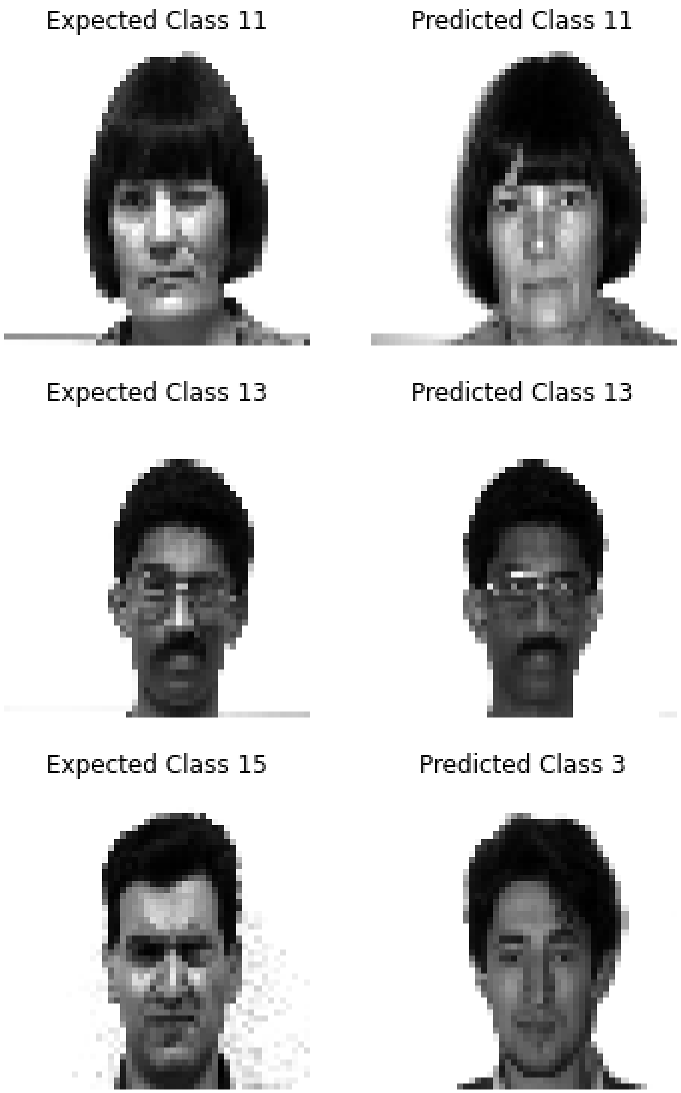
Predicted Class 5

Expected Class 7

Predicted Class 7

Expected Class 9

Predicted Class 9



Expected Class 11

Predicted Class 11

Expected Class 13

Predicted Class 13

Expected Class 15

Predicted Class 3