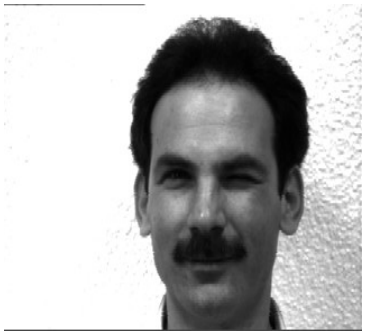
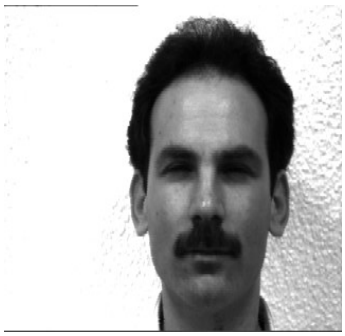
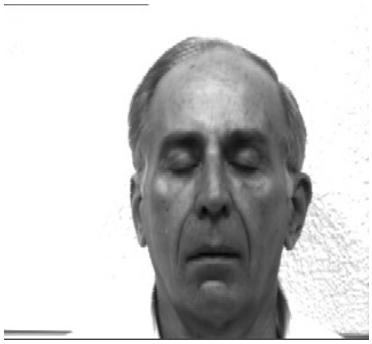
Pramod Sivaram Kaushik(201407538)
Mohd Salman Khan(201305513)
Sajal Sharma(201305526)

Image Representation & Recognition

Dataset:
The dataset is taken from Original Yale Face Dataset (http://vision.ucsd.edu/content/yale-face-database). The Yale Face Database (size 6.4MB) contains 165 grayscale images in GIF format of 15 individuals. There are 11 images per subject, one per different facial expression or configuration: center-light, w/glasses, happy, left-light, w/no glasses, normal, right-light, sad, sleepy, surprised, and wink.

Some of these images are given below :



Dividing the dataset using 5 - fold Cross Validation:

The following code is used to perform 5 – fold cross validation :

```
def LoadImages(partition,imagepath):
    totalImages = len(imagepath)
    for i in range(totalImages):
        if i > (partition*20)*totalImages/100 or i < (((partition-1)*20))*totalImages/100:
            image = Image.open(imagepath[i])
            trainData.append(numpy.array(image).flatten())
            trainLabel.append(imagepath[i].split("/")[6].split(".")[0])
        else:
            image = Image.open(imagepath[i])
            testData.append(numpy.array(image).flatten())
            testLabel.append(imagepath[i].split("/")[6].split(".")[0])
```

Obtaining Eigen Faces and Mean Face

The following function calculates the Eigen Faces and the mean face :

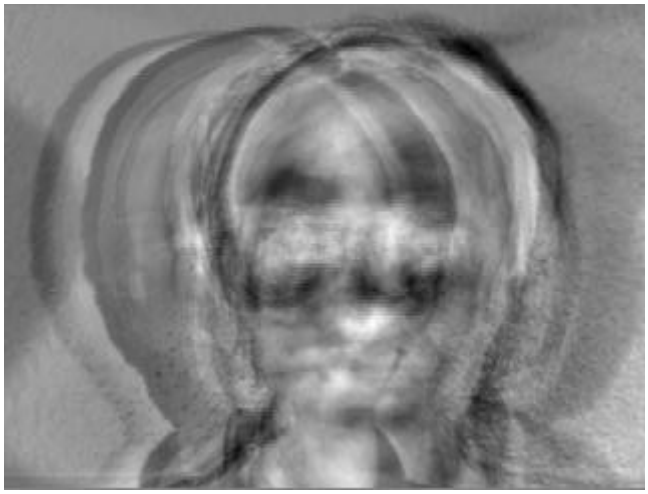
```
def PCA(data):
    # mean
    mu = numpy.mean(data, 0)
    # mean adjust the data
    ma_data = data - mu
    # run SVD
    e_faces, sigma, v = linalg.svd(ma_data.transpose(), full_matrices=False)
    # compute weights for each image
    weights = numpy.dot(ma_data, e_faces)
    return e_faces, weights, mu
```

“data” is the training data that is passed to the PCA function. It returns e_faces(Eigen Faces), Weights(no. Of training faces * no. Of training faces) and Mean.

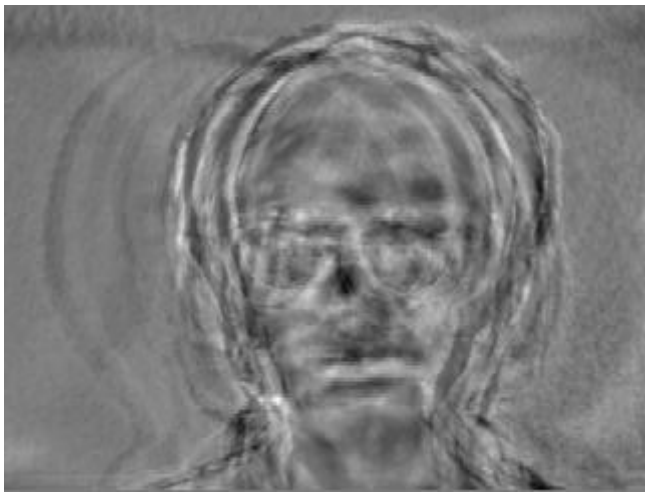
The mean face so obtained is given below :

The eigen faces so developed are equal to the total number of training samples, some of the eigen faces are given below :

Eigen Face 1



Eigen Face 24



Eigen Face 63

Reconstruction from the Weights Matrix and Eigen Vectors

The following code is used to reconstruct the Images :

```
reconstructed = []
for p in range(trainingData.shape[0]):
        reconstructed.append(Reconstruct(p, e_faces, trainingWeights, mu,
                                         len(trainingWeights)))
        imgID = p
        SaveImage(outDIR, "reconstructed/" + str(p), imgID,imgDims, reconstructed[p])
```

The results obtained were then comapared with the original image :

Reconstructed Image

Original Image

1 NN Identification Code with results :

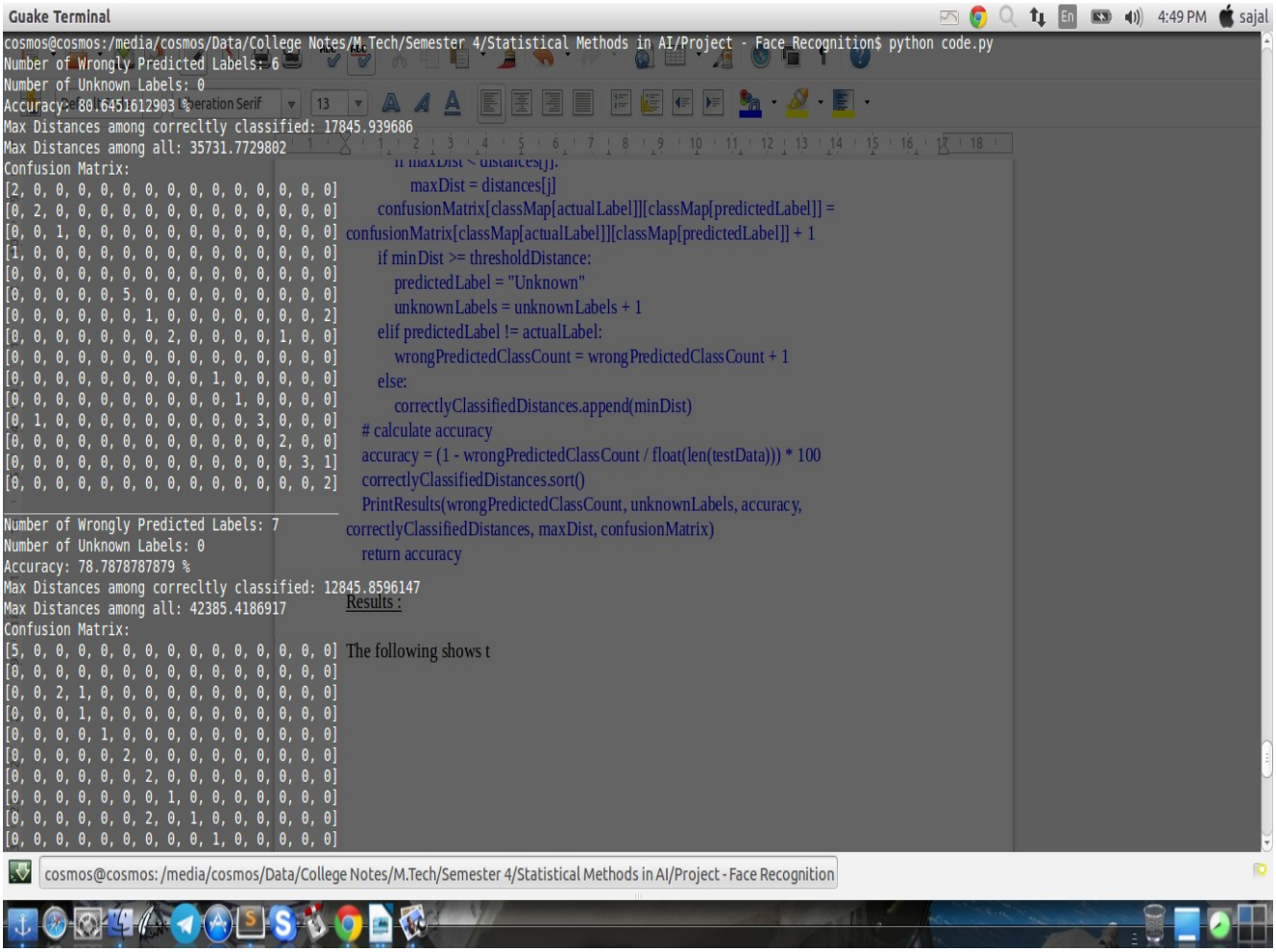
The following is 1 NN code for predicting class labels for test data :

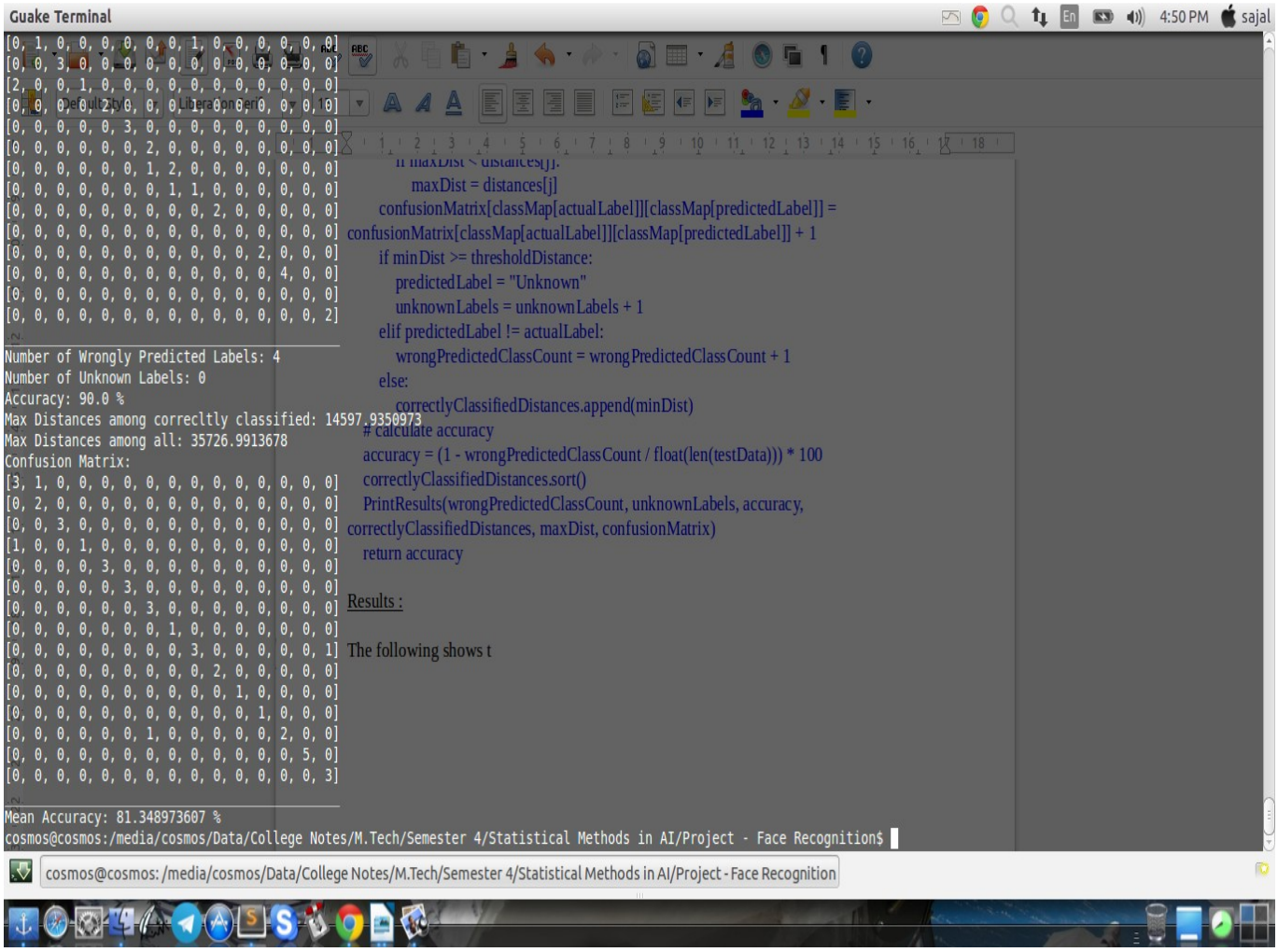
```
def EuclideanDistance(instance1, instance2, length):
        distance = 0
        for x in range(length):
                distance += pow((instance1[x] - instance2[x]), 2)
        return math.sqrt(distance)

def PredictLabelsFromTestData(testData, noOfClasses, mu, e_faces, trainingWeights,
testLabels, trainingLabels, classMap, thresholdDistance):
    correctlyClassifiedDistances = []
    confusionMatrix = [[0 for i in xrange(noOfClasses)] for i in xrange(noOfClasses)]
    wrongPredictedClassCount = 0
    unknownLabels = 0
```

```
    for i in range(len(testData)):
        testWeight=InputWeight(testData[i],mu,e_faces)
        distances = []
        for x in range(len(trainingWeights)):
            dist = EuclideanDistance(testWeight, trainingWeights[x], len(testWeight))
            #dist = Mahanalobis(testWeight, trainingWeights[x])
            distances.append(dist)
        actualLabel = testLabels[i]
        predictedLabel = ""
        minDist = sys.maxint
        maxDist = -sys.maxint
        for j in range(len(distances)):
            if minDist > distances[j]:
                minDist = distances[j]
                predictedLabel = trainingLabels[j]
            if maxDist < distances[j]:
                maxDist = distances[j]
        confusionMatrix[classMap[actualLabel]][classMap[predictedLabel]] =
confusionMatrix[classMap[actualLabel]][classMap[predictedLabel]] + 1
        if minDist >= thresholdDistance:
            predictedLabel = "Unknown"
            unknownLabels = unknownLabels + 1
        elif predictedLabel != actualLabel:
            wrongPredictedClassCount = wrongPredictedClassCount + 1
        else:
            correctlyClassifiedDistances.append(minDist)
    # calculate accuracy
    accuracy = (1 - wrongPredictedClassCount / float(len(testData))) * 100
    correctlyClassifiedDistances.sort()
    PrintResults(wrongPredictedClassCount, unknownLabels, accuracy,
correctlyClassifiedDistances, maxDist, confusionMatrix)
    return accuracy
```

Results :

The following images show the results obtained :





The results obtained were :

Iteration 1
Number of Wrongly Predicted Labels: 6
Number of Unknown Labels: 0
Accuracy: 80.6451612903 %
Max Distances among correcltly classified: 17845.939686
Max Distances among all: 35731.7729802
Confusion Matrix:
[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2]
[0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]

Iteration 2
Number of Wrongly Predicted Labels: 7
Number of Unknown Labels: 0
Accuracy: 78.7878787879 %
Max Distances among correcltly classified: 12845.8596147
Max Distances among all: 42385.4186917
Confusion Matrix:
[5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 1]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

Iteration 3
Number of Wrongly Predicted Labels: 7
Number of Unknown Labels: 0
Accuracy: 76.6666666667 %
Max Distances among correcltly classified: 13430.776403
Max Distances among all: 44870.0955506
Confusion Matrix:
[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 3, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]

[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

Iteration 4
Number of Wrongly Predicted Labels: 6
Number of Unknown Labels: 0
Accuracy: 80.6451612903 %
Max Distances among correclty classified: 14713.371206
Max Distances among all: 36198.3732523
Confusion Matrix:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 2, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2]

Iteration 5
Number of Wrongly Predicted Labels: 4
Number of Unknown Labels: 0
Accuracy: 90.0 %
Max Distances among correclty classified: 14597.9350973
Max Distances among all: 35726.9913678
Confusion Matrix:
[3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3]

Mean Accuracy: 81.348973607 %

The average **Accuracy** obtained is between **80-85%**.

The complete Code:

```
import os
import sys
import pdb
import glob
import math
import numpy
import random
from sets import Set
import scipy as scipy
from scipy.misc import *
from scipy import linalg

# This function loads the images, divides the data intro training and test set
def LoadImages(directory, split):
    # get a list of all the picture filenames
    gifs = glob.glob(directory + '/*.gif')
    # uncomment the below line when trying an unknown file
    #extraGif = glob.glob("/media/cosmos/Data/College Notes/M.Tech/Semester 4/Statistical Methods in
AI/Project - Face Recognition/1.gif")
    classMap = {}
    testGIF = []
    allLabels = []
    testLabels = []
    trainingGIF = []
    trainingLabels = []
    for i in range(len(gifs)):
        if random.random() < split:
            trainingGIF.append(gifs[i])
            l = gifs[i].split("/");
            labelName = l[len(l)-1].split(".")[0][-2:]
            trainingLabels.append(labelName)
            allLabels.append(labelName)
        else:
            testGIF.append(gifs[i])
            l = gifs[i].split("/");
            labelName = l[len(l)-1].split(".")[0][-2:]
            testLabels.append(labelName)
            allLabels.append(labelName)
    # uncomment the below 2 lines when trying an unknown file
    #testGIF.append(extraGif[0])
    #testLabels.append("un")
    trainingImgs = numpy.array([imread(i, True).flatten() for i in trainingGIF])
    testImgs = numpy.array([imread(i, True).flatten() for i in testGIF])
    # creating a list of class labels
    allLabels = set(allLabels)
    noOfClasses = len(allLabels)
    sortedLabels = []
    for i in allLabels:
        sortedLabels.append(i)
```

```
    sortedLabels = sorted(sortedLabels)
    # creating a mapping for confusion matrix
    j = 0
    for i in sortedLabels:
        classMap[i] = j
        j = j + 1
    return trainingImgs,testImgs,trainingLabels,testLabels,noOfClasses,classMap

def EuclideanDistance(instance1, instance2, length):
            distance = 0
            for x in range(length):
                        distance += pow((instance1[x] - instance2[x]), 2)
            return math.sqrt(distance)

def Mahanalobis(x, y):
    return scipy.spatial.distance.mahalanobis(x,y,np.linalg.inv(np.cov(x,y)))

# Run Principal Component Analysis on the input data.
# INPUT  : data    - an n x p matrix
# OUTPUT : e_faces -
#          weights -
#          mu      -
def PCA(data):
    # mean
    mu = numpy.mean(data, 0)
    # mean adjust the data
    ma_data = data - mu
    # run SVD
    e_faces, sigma, v = linalg.svd(ma_data.transpose(), full_matrices=False)
    # compute weights for each image
    weights = numpy.dot(ma_data, e_faces)
    return e_faces, weights, mu

# This function calculates the weight of the test data
def InputWeight(testData, mu, e_faces):
    ma_data = testData - mu
    weights = numpy.dot(ma_data, e_faces)
    return weights

# Reconstruct an image using the given number of principal components.
def Reconstruct(imgIDx, e_faces, weights, mu, npcs):
            # dot weights with the eigenfaces and add to mean
            recon = mu + numpy.dot(weights[imgIDx, 0:npcs], e_faces[:, 0:npcs].T)
            return recon

# Saves the image in the given directory
def SaveImage(outDIR, subdir, imgID, imgDims, data):
            directory = outDIR + "/" + subdir
            if not os.path.exists(directory): os.makedirs(directory)
            imsave(directory + "/image_" + str(imgID) + ".jpg", data.reshape(imgDims))

# Prints the final results
def PrintResults(wrongPredictedClassCount, unknownLabels, accuracy, correctlyClassifiedDistances,
maxDist, confusionMatrix):
    print "Number of Wrongly Predicted Labels:",wrongPredictedClassCount
    print "Number of Unknown Labels:",unknownLabels
    print "Accuracy:",accuracy,"%"
```

```
    print "Max Distances among correcltly
classified:",correctlyClassifiedDistances[len(correctlyClassifiedDistances) - 1]
    print "Max Distances among all:",maxDist
    PrintConfusionMatrix(confusionMatrix)

# Prints the confustion matrix
def PrintConfusionMatrix(confusionMatrix):
    print "Confusion Matrix:"
    for i in range(0, len(confusionMatrix)):
        print confusionMatrix[i]

# Predicts the class of test data
def PredictLabelsFromTestData(testData, noOfClasses, mu, e_faces, trainingWeights, testLabels,
trainingLabels, classMap, thresholdDistance):
    correctlyClassifiedDistances = []
    confusionMatrix = [[0 for i in xrange(noOfClasses)] for i in xrange(noOfClasses)]
    wrongPredictedClassCount = 0
    unknownLabels = 0
    for i in range(len(testData)):
        testWeight=InputWeight(testData[i],mu,e_faces)
        distances = []
        for x in range(len(trainingWeights)):
            dist = EuclideanDistance(testWeight, trainingWeights[x], len(testWeight))
            #dist = Mahanalobis(testWeight, trainingWeights[x])
            distances.append(dist)
        actualLabel = testLabels[i]
        predictedLabel = ""
        minDist = sys.maxint
        maxDist = -sys.maxint
        for j in range(len(distances)):
            if minDist > distances[j]:
                minDist = distances[j]
                predictedLabel = trainingLabels[j]
            if maxDist < distances[j]:
                maxDist = distances[j]
        confusionMatrix[classMap[actualLabel]][classMap[predictedLabel]] =
confusionMatrix[classMap[actualLabel]][classMap[predictedLabel]] + 1
        if minDist >= thresholdDistance:
            predictedLabel = "Unknown"
            unknownLabels = unknownLabels + 1
        elif predictedLabel != actualLabel:
            wrongPredictedClassCount = wrongPredictedClassCount + 1
        else:
            correctlyClassifiedDistances.append(minDist)
    # calculate accuracy
    accuracy = (1 - wrongPredictedClassCount / float(len(testData))) * 100
    correctlyClassifiedDistances.sort()
    PrintResults(wrongPredictedClassCount, unknownLabels, accuracy, correctlyClassifiedDistances,
maxDist, confusionMatrix)
    return accuracy

def main():
    inDIR  = "/media/cosmos/Data/College Notes/M.Tech/Semester 4/Statistical Methods in AI/Project - Face
Recognition/yalefaces/yalefaces"
    outDIR = "/media/cosmos/Data/College Notes/M.Tech/Semester 4/Statistical Methods in AI/Project - Face
Recognition/yalefaces/SampleOutput"
    imgDims = (243, 320)
```

```
    split = 0.8
    noOfDimensions = 100
    thresholdDistance = 18000.0
    trainingData, testData, trainingLabels, testLabels, noOfClasses, classMap = LoadImages(inDIR, split)
    e_faces, trainingWeights, mu = PCA(trainingData)
            # save mean photo
    imsave(outDIR + "/mean.gif", mu.reshape(imgDims))
    # save each eigenface as an image
    for i in range(e_faces.shape[1]):
                        SaveImage(outDIR, "eigenfaces", i, imgDims, e_faces[:,i])
            # reconstruct each face image using an increasing number of principal components
    reconstructed = []
    for p in range(trainingData.shape[0]):
        reconstructed.append(Reconstruct(p, e_faces, trainingWeights, mu, len(trainingWeights)))
        imgID = p
        SaveImage(outDIR, "reconstructed/" + str(p), imgID,imgDims, reconstructed[p])
    # Predicting Classes for test data
    accuracy = PredictLabelsFromTestData(testData, noOfClasses, mu, e_faces, trainingWeights, testLabels,
trainingLabels, classMap, thresholdDistance)
    return accuracy

total = 0.0
for i in range(0, 5):
    total = total + main()
    print "_____"
print "Mean Accuracy:",total / 5,"%"
```

Interesting Results:

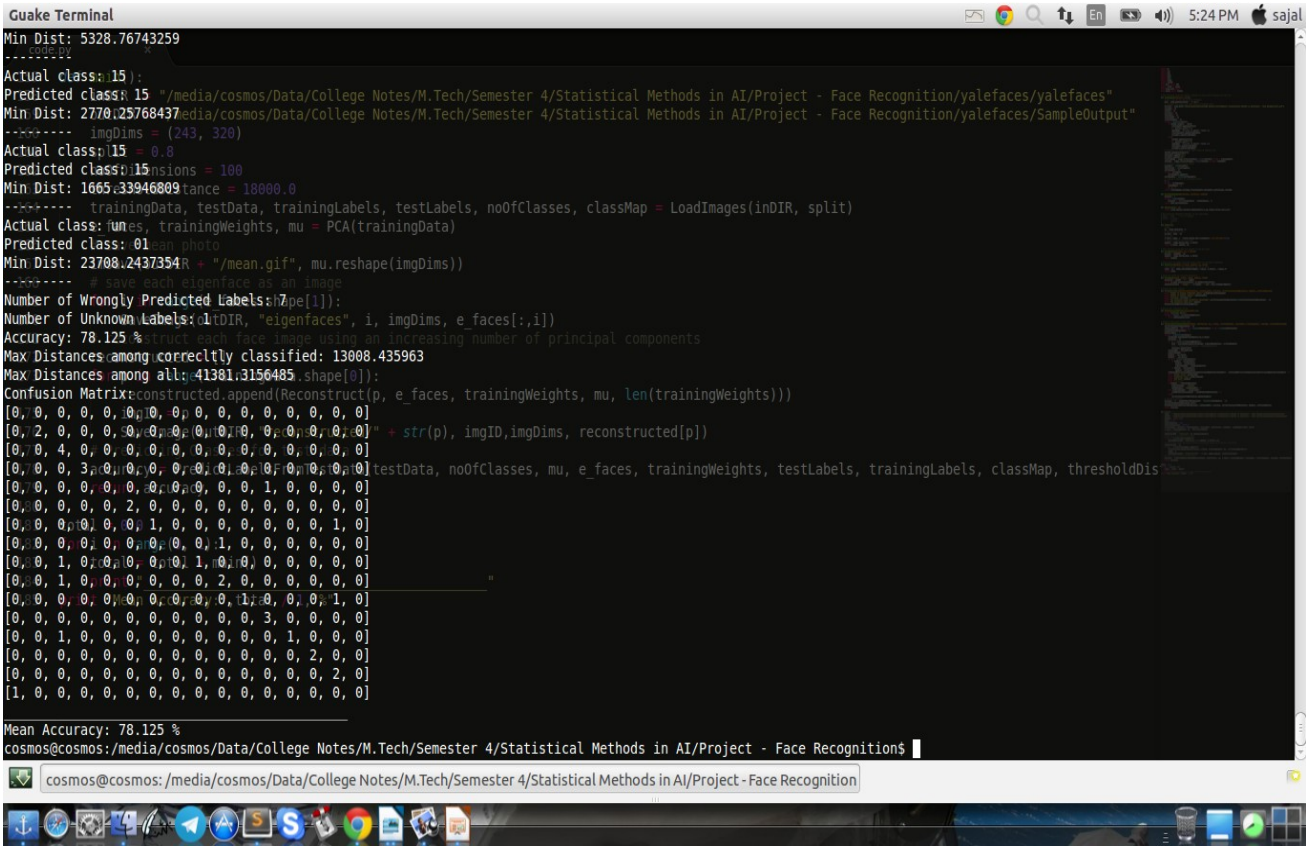
1. If someone doesn't want his/her face to be recognised, he should smile, tilt his head, and blink vigorously while the photograph is being taken. Doing so will confuse the training model and the person won't be recognised correctly(Read it from one of the blog online).

2. If different lighting conditions for enrolment and query then eigenfaces have difficulty in determining the correct face. Bright light causes image saturation.

3. Unknown Classification :
When the given image is dfferent from the training samples, it should be classified as Unknown rather than getting wrongly classified, we have taken the threshold value coming from euclidean distance to be 18000(1.5 times the maximum distance from correctly classified samples), any image whose minimum distance from training images is greater than 18000 will be classified as “Unknown”.

Image used :

Now this is not a facial image and should be tagged as “Unknown”, and hence the code as well classified it as “Unknown”.



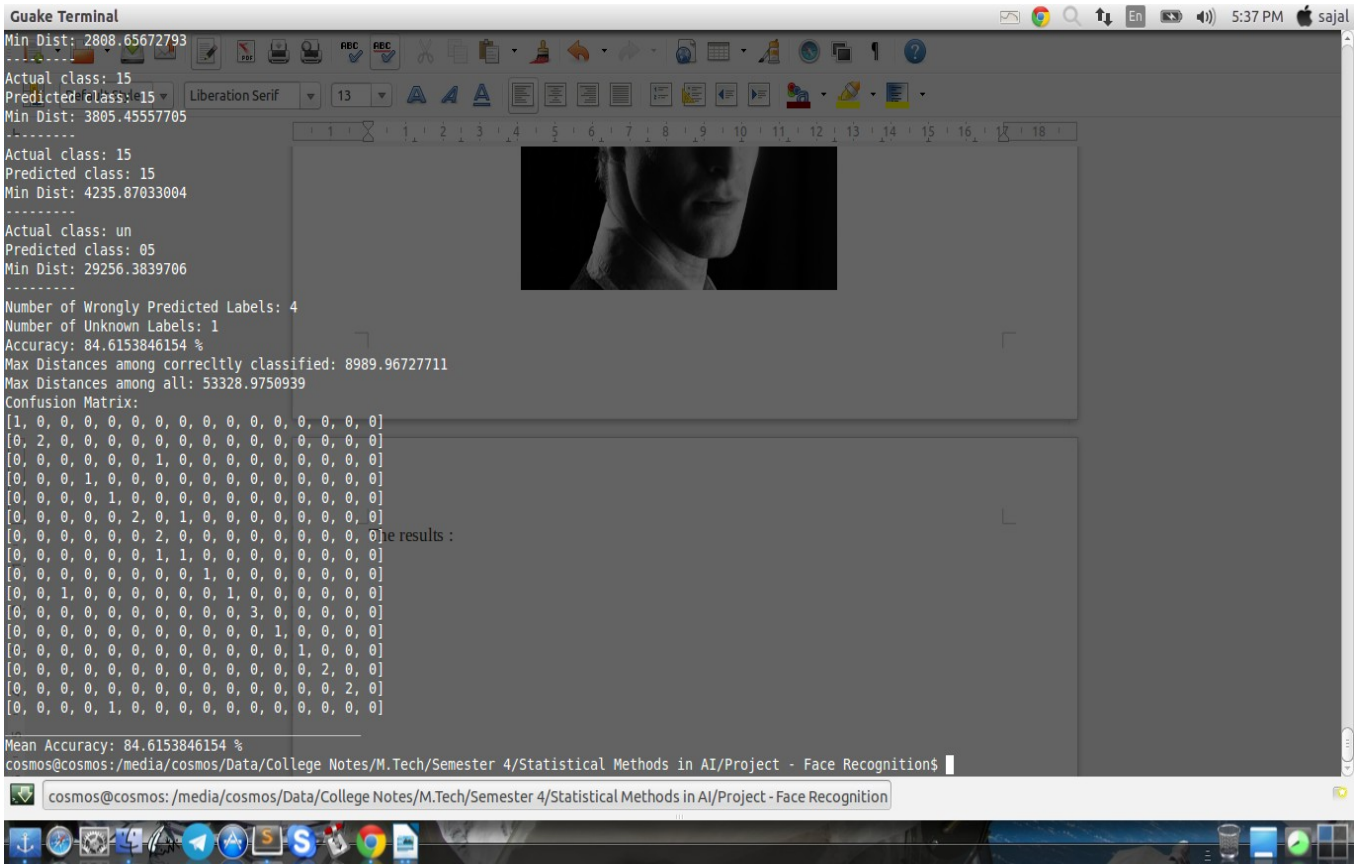
Actual class: un
Predicted class: 01
Min Dist: 23708.2437354

Number of Wrongly Predicted Labels: 7
Number of Unknown Labels: 1
Accuracy: 78.125 %
Max Distances among correcltly classified: 13008.435963
Max Distances among all: 41381.3156485
Confusion Matrix:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

And then tried with a new facial image of Daniel Radcliffe which was not there in the training set, here are the results obtained :

The image :

The results :



Actual class: un
Predicted class: 05
Min Dist: 29256.3839706

Number of Wrongly Predicted Labels: 4
Number of Unknown Labels: 1
Accuracy: 84.6153846154 %
Max Distances among correcltly classified: 8989.96727711
Max Distances among all: 53328.9750939
Confusion Matrix:
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]