CPSC 552 – Assignment 7

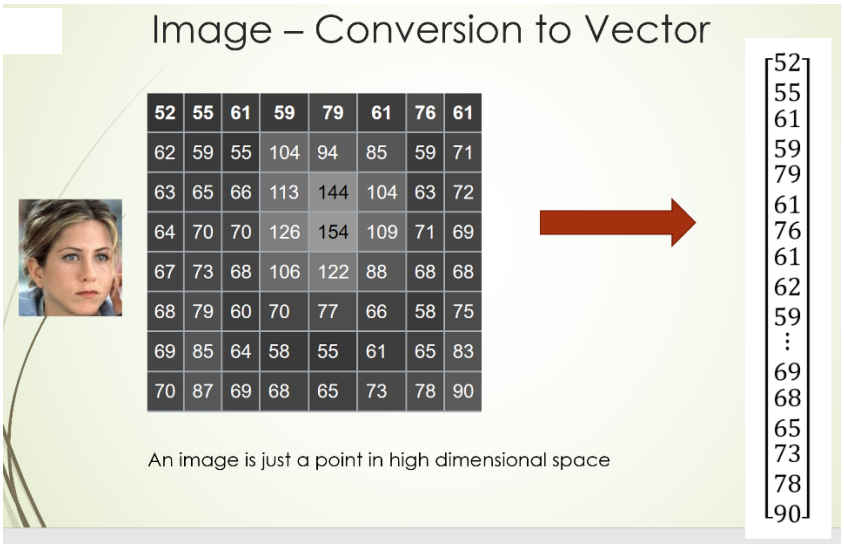
PCA Algorithm for Dimensionality Reduction

PCA (Principle Component Analysis) is a popular algorithm for dimensionality reduction. For high dimensional data, often it is helpful to reduce the dimensions such that the features in the data that help distinguish the data from one another are kept. Another reason for dimensionality reduction is to be able to visualize the data in two or three dimensions. PCA is an unsupervised algorithm, meaning, it does not need to know the class of each data item in reducing the dimensionality.

The steps in the PCA as explained in the lecture are:

Step 1: Assemble the data in a matrix.

For image type of data, usually the pixels belonging to an image (after being converted to gray scale) are converted into a column vector (as shown below), and the different images in the data set are assembled in a matrix with each column representing an image.



For non-image type of data, the data usually is given to us as a comma separated or Excel file where each row represents a data item, and the columns of the matrix represent features. So assembly in a matrix is straightforward.

Step 2: Subtract the mean of all data from each data item. If the data items in the matrix ***X*** were assembled where each column represents a data item, then the mean has to be computed on each row and then subtracted from each column, and vice versa if the data matrix was assembled row wise. For this reason, we will develop a row version of the PCA algorithm and a column version.

Step 3: Compute the Covariance matrix.
Step 4: Compute the Eigen values and the Eigen vectors of the covariance matrix

Step 5: Sort the Eigen values by magnitude and assemble the corresponding Eigen vectors into an EV matrix.
Step 6: Multiply the ***X*** (mean adjusted data matrix) with the EV matrix to create the P matrix.
Step 7: Project each data item (after subtracting the mean of all data) onto the P matrix to obtain the reduced dimensions for the data item.

For details of the above steps, refer to the lecture given in the class.

In Python, the main set of steps for the PCA can be programmed as:

```
    def pca(self,X, num_components=0):
        [d,n] = X.shape   # n - number of images, d = input dimension e.g.
112x92=10304
        if (num_components <= 0) or (num_components > n):
            num_components = n
        mu = X.mean(axis=1)
        X = (X.T - mu).T
        if d<n:
            Cov = np.dot(X,X.T) # covariance matrix
            [eigenvalues , EV] = np.linalg.eigh(Cov)
        else:
            Cov = np.dot(X.T,X)
            [eigenvalues ,eigenvectors] = np.linalg.eigh(Cov)
            EV = np.dot(X,eigenvectors) # it is termed as EigenFace

        # convert each eigenvector to a unit vector by dividing it by its norm
        for i in range(n):
            EV[:,i] = EV[:,i]/np.linalg.norm(EV[:,i])

        #  sort eigenvectors descending by their eigenvalue
        idx = np.argsort(-eigenvalues)
        eigenvalues = eigenvalues[idx]
        EV = EV[:,idx]
        #  select only num_components
        eigenvalues = eigenvalues[0:num_components].copy()
        EV = EV[:,0:num_components].copy()
              return [eigenvalues , EV , mu]
```

We will program the PCA algorithm (both a row version and a column version) for face recognition on the ATT face dataset. This dataset contains 400 images (10 for each person and there are 40 persons). Each image is 112x92 gray scale image. Once we convert each image to a vector, it will become 10304x1 vector. We will assemble 200 images (our training set, 5 images for each person) into a 10304x200 matrix where each column represents one image. After PCA has transformed each data items dimension to e.g., 30 dimensions, we will test the accuracy of face recognition on the remaining 200 images.

Create a Python application called PCAFaceRecog. Add a file called DistanceMetric.py with the following code in it. This file contains two distance metrics i.e., Euclidean distance and the Cosine distance. The idea is that after we reduce the dimensionality of incoming unknown data and we try to compare it with the known 200 reduced

dimension images, we need a distance criteria to see which one of the known 200 images is closer in distance to the unknown image.

```
import numpy as np

class EuclideanDistance():
    def __call__(self , p, q):
        p = np.asarray(p).flatten()
        q = np.asarray(q).flatten()
        return np.sqrt(np.sum(np.power((p-q),2)))

class CosineDistance():
    def __call__(self , p, q):
        p = np.asarray(p).flatten()
        q = np.asarray(q).flatten()
        return -np.dot(p.T,q) / (np.sqrt(np.dot(p,p.T)*np.dot(q,q.T)))
```

Add a class called PCACol with the following code in it. This class implements the column version of the PCA.

```
import numpy as np
import os
import sys
from PIL import Image  # pip install pillow
import matplotlib.cm as cm # pip install matplotlib
import matplotlib.pyplot as plt
from DistanceMetric import *

class PCACol(object):  # column wise data computations
    def __init__(self):
        self.projections = []
        self.dist_metric = EuclideanDistance()

    def asRowMatrix(self,X):
        if len(X) == 0:
            return np.array([])
        mat = np.empty((0, X[0].size), dtype=X[0].dtype)

        for row in X:
            mat = np.vstack((mat, np.asarray(row).reshape(1,-1)))
        return mat

    def asColumnMatrix(self,X):
        if len(X) == 0:
            return np.array([])
        mat = np.empty((X[0].size , 0), dtype=X[0].dtype)
        for col in X:
            mat = np.hstack((mat, np.asarray(col).reshape(-1,1)))
        return mat

    def pca(self,X, num_components=0):
        [d,n] = X.shape   # n - number of images, d = input dimension e.g. 112x92=10304
        if (num_components <= 0) or (num_components > n):
            num_components = n
        mu = X.mean(axis=1)
        X = (X.T - mu).T
```

```
        if d<n:
            Cov = np.dot(X,X.T) # covariance matrix
            [eigenvalues , EV] = np.linalg.eigh(Cov)
        else:
            Cov = np.dot(X.T,X)
            [eigenvalues ,eigenvectors] = np.linalg.eigh(Cov)
            EV = np.dot(X,eigenvectors) # it is termed as EigenFace

        # convert each eigenvector to a unit vector by dividing it by its norm
        for i in range(n):
            EV[:,i] = EV[:,i]/np.linalg.norm(EV[:,i])

        #  sort eigenvectors descending by their eigenvalue
        idx = np.argsort(-eigenvalues)
        eigenvalues = eigenvalues[idx]
        EV = EV[:,idx]
        #  select only num_components
        eigenvalues = eigenvalues[0:num_components].copy()
        EV = EV[:,0:num_components].copy()
        return [eigenvalues , EV , mu]

    def normalize(self,X, low, high , dtype=None):
        X = np.asarray(X)
        minX , maxX = np.min(X), np.max(X)
        # normalize to [0...1].
        X = X - float(minX)
        X = X / float((maxX - minX)) # scale to [low...high].
        X = X * (high -low)
        X = X + low
        if dtype is None:
            return np.asarray(X)
        return np.asarray(X, dtype=dtype)

    def read_images(self,path , sz=None):
        c = 0
        X,y,yseq = [], [], [] # list of images and labels, yseq is sequential ordering
number
        for dirname , dirnames , filenames in os.walk(path):
            for filename in filenames:
                try:
                    fname = path+filename
                    im = Image.open(fname)
                    im = im.convert("L") # resize to given size (if given)
                    if (sz is not None):
                        im = im.resize(sz, Image.ANTIALIAS)
                    imdata = np.asarray(im, dtype=np.uint8)
                    sh = imdata.shape
                    X.append(np.asarray(im, dtype=np.uint8))
                    sh = len(X)
                    label = filename.split('_',1)[0]  # e.g. S10
                    y.append(label)  # use this to determine accuracy
                    yseq.append(c)
                    #y.append(filename)  uncomment this to show individual test
                    c = c + 1
                except IOError:
                    print("I/O error({0}): {1}".format(errno , strerror))
        return [X,y,yseq]
```

```
    def project(self, EF, X, mu=None):
        if mu is None:
            return np.dot(EF.T,X)
        return np.dot(EF.T,(X.T - mu).T).T    # .T to make it e.g. 1x100

    def reconstruct(self, EF, P, mu=None): # P are the coefficients e.g. 1x100
        if mu is None:
            return np.dot(EF,P.T)
        return (np.dot(EF,P.T).T + mu).T

    def create_font(self, fontname='Tahoma', fontsize=10):
        return { 'fontname': fontname , 'fontsize':fontsize }

    def subplot(self,title , images , rows , cols , sptitle="subplot", sptitles=[],
colormap=cm.gray , ticks_visible=True , filename=None):
        fig = plt.figure()
        # main title
        fig.text(.5, .95, title , horizontalalignment='center')
        for i in range(len(images)):
            ax0 = fig.add_subplot(rows ,cols ,(i+1))
            plt.setp(ax0.get_xticklabels(), visible=False)
            plt.setp(ax0.get_yticklabels(), visible=False)
            if len(sptitles) == len(images):
                plt.title("%s #%s" % (sptitle , str(sptitles[i])),
self.create_font('Tahoma',10))
            else:
                plt.title("%s #%d" % (sptitle , (i+1)), self.create_font('Tahoma',10))
            plt.imshow(np.asarray(images[i]), cmap=colormap)
        if filename is None:
            plt.show()
        else:
            fig.savefig(filename)

    def predict(self , EF, X, mu, y, yseq):
        minDist = np.finfo('float').max
        minClass = -1
        index = -1
        Q = self.project(EF, X.reshape(-1,1), mu).T
        for i in range(len(self.projections)):
            dist = self.dist_metric(self.projections[i], Q)
            if dist < minDist:
                minDist = dist
                minClass = y[i]
                index = yseq[i]
        return minClass, index
```

Add a class called PCARow with the following code in it. This class implements the row version of the PCA.

```
import numpy as np
import os
import sys
from PIL import Image  # conda install Pillow  (Pillow is PIL for Python 3)
import matplotlib.cm as cm # conda install matplotlib
import matplotlib.pyplot as plt
```

```
from DistanceMetric import EuclideanDistance

class PCARow(object):   # Row wise computations
    def __init__(self):
        self.projections = []
        self.dist_metric = EuclideanDistance()

    def asRowMatrix(self,X):
        if len(X) == 0:
            return np.array([])
        mat = np.empty((0, X[0].size), dtype=X[0].dtype)

        for row in X:
            mat = np.vstack((mat, np.asarray(row).reshape(1,-1)))
        return mat

    def asColumnMatrix(self,X):
        if len(X) == 0:
            return np.array([])
        mat = np.empty((X[0].size , 0), dtype=X[0].dtype)
        for col in X:
            mat = np.hstack((mat, np.asarray(col).reshape(-1,1)))
        return mat

    def pca(self,X, y, num_components=0):
        [n,d] = X.shape
        if (num_components <= 0) or (num_components >n):
            num_components = n
        mu = X.mean(axis=0)
        X = X - mu
        if n>d:
            C = np.dot(X.T,X)
            [eigenvalues ,EV] = np.linalg.eigh(C)
        else:
            C = np.dot(X,X.T)
            [eigenvalues ,eigenvectors] = np.linalg.eigh(C)
            EV = np.dot(X.T,eigenvectors)
        for i in range(n):
            EV[:,i] = EV[:,i]/np.linalg.norm(EV[:,i])

        #  sort eigenvectors descending by their eigenvalue
        idx = np.argsort(-eigenvalues)
        eigenvalues = eigenvalues[idx]
        EV = EV[:,idx]
        #  select only num_components
        eigenvalues = eigenvalues[0:num_components].copy()
        EV = EV[:,0:num_components].copy()
        return [eigenvalues , EV , mu]

    def read_images(self,path , sz=None):
        c = 0
        X,y,yseq = [], [], [] # list of images and labels
        for dirname , dirnames , filenames in os.walk(path):
            for filename in filenames:
                try:
                    fname = path+filename
                    im = Image.open(fname)
                    im = im.convert("L") # resize to given size (if given)
```

```
                    if (sz is not None):
                        im = im.resize(sz, Image.ANTIALIAS)
                    imdata = np.asarray(im, dtype=np.uint8)
                    sh = imdata.shape
                    X.append(np.asarray(im, dtype=np.uint8))
                    sh = len(X)
                    label = filename.split('_',1)[0]  # e.g. S10
                    y.append(label)  # use this to determine accuracy
                    yseq.append(c)
                    #y.append(filename)  uncomment this to show individual test
                    c = c + 1
                except IOError:
                    print("I/O error({0}): {1}".format(errno , strerror))
                except:
                    print("Unexpected error:", sys.exc_info()[0])
                    raise
        return [X,y,yseq]

    def normalize(self, X, low, high , dtype=None):  # since eignevectors can have
negative values
        X = np.asarray(X)    # to be able to visualize this, we need to scale between
        minX , maxX = np.min(X), np.max(X) # 0 and 1
        # normalize to [0...1].
        X = X - float(minX)
        X = X / float((maxX - minX))
        # scale to [low...high].
        X = X * (high -low)
        X = X + low
        if dtype is None:
            return np.asarray(X)
        return np.asarray(X, dtype=dtype)

    def project(self, EV, X, mu=None):
        if mu is None:
            return np.dot(X,EV)
        return np.dot(X - mu, EV)

    def reconstruct(self, EV, Y, mu=None): #
        if mu is None:
            return np.dot(Y,EV.T)
        return np.dot(Y, EV.T) + mu

    def create_font(self, fontname='Tahoma', fontsize=10):
        return { 'fontname': fontname , 'fontsize':fontsize }

    def subplot(self,title , images , rows , cols , sptitle="subplot", sptitles=[],
colormap=cm. gray , ticks_visible=True , filename=None):
        fig = plt.figure()
        # main title
        fig.text(.5, .95, title , horizontalalignment='center')
        for i in range(len(images)):
            ax0 = fig.add_subplot(rows ,cols ,(i+1))
            plt.setp(ax0.get_xticklabels(), visible=False)
            plt.setp(ax0.get_yticklabels(), visible=False)
            if len(sptitles) == len(images):
                plt.title("%s #%s" % (sptitle , str(sptitles[i])),
self.create_font('Tahoma',10))
            else:
```

```
                plt.title("%s #%d" % (sptitle , (i+1)), self.create_font('Tahoma',10))
            plt.imshow(np.asarray(images[i]), cmap=colormap)
        if filename is None:
            plt.show()
        else:
            fig.savefig(filename)

    def predict(self , EV, X, mu, y, yseq):
        minDist = np.finfo('float').max
        minClass = -1
        index = -1
        Q = self.project(EV, X.reshape(1,-1), mu)
        for i in range(len(self.projections)):
            dist = self.dist_metric(self.projections[i], Q)
            if dist < minDist:
                minDist = dist
                minClass = y[i]
                index = yseq[i]
        return minClass,index
```

In the main PCAFaceRecog.py, type the following code:

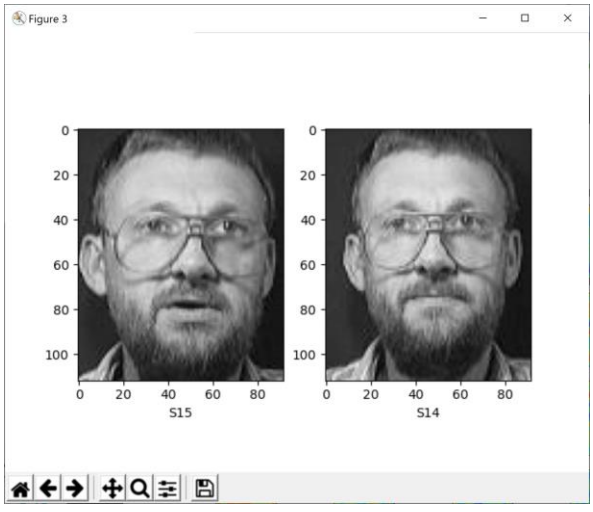
```
import sys
import numpy as np
from PCARow import PCARow
from PCACol import PCACol
import matplotlib.cm as cm
import matplotlib.pyplot as plt

def main():
    pcah = PCACol()  # change to PCARow()  to use row wise version of data
    #------------------column wise version of PCA----------------
    [X,y,yseq] = pcah.read_images("D:/Images/AttDataSet/ATTDataSet/Training/")
    print(X[0].shape)

    [Xtest,ytest,yseqt] = pcah.read_images("D:/Images/AttDataSet/ATTDataSet/Testing/")
    # X is a list f 112x92  2-d arrays, y are the labels
    [E, EV, mu] = pcah.pca(pcah.asColumnMatrix(X),100) # top 100 Eigen vectors
    # E is the Eigen value array, EV is the catenated Eigen vectors, mu is the mean image
    print(EV.shape)

    # turn the first (at most) 16 eigenvectors into grayscale
    # images (note: eigenvectors are stored by column!)
    EF16 = []
    for i in range(min(len(X), 16)):
        e = EV[:,i].reshape(X[0].shape)
        EF16.append(pcah.normalize(e,0,255)) # for visualization purposes
    print(len(EF16))
    pcah.subplot(title="Eigenfaces AT&T Facedatabase", images=EF16, rows=4, cols=4,
sptitle=" Eigenface", colormap=cm.jet, filename="python_pca_eigenfaces.png")

    # reconstruct projections of first n Eigen Faces
    steps=[i for i in range(10, min(len(X), 200), 20)]
    EF10 = []
    for i in range(min(len(steps), 16)):
        numEvs = steps[i]
```

```
        P = pcah.project(EV[:,0:numEvs], X[0].reshape(-1,1), mu)
        R = pcah.reconstruct(EV[:,0:numEvs], P, mu)
        # reshape and append to plots
        R = R.reshape(X[0].shape)
        EF10.append(pcah.normalize(R,0,255))
        # plot them and store the plot to "python_reconstruction.pdf"
    pcah.subplot(title="Reconstruction AT&T Facedatabase", images=EF10, rows=4, cols=4,
           sptitle=" Eigenvectors", sptitles=steps , colormap=cm.gray ,
           filename=" python_pca_reconstruction.png")

    for xi in X:
        pcah.projections.append(pcah.project(EV, xi.reshape(-1,1), mu))
    imtest = 20  # image number to test
    labelPredicted, index = pcah.predict(EV,Xtest[imtest],mu,y,yseq)
    plt.figure()
    plt.subplot(1,2,1)
    plt.imshow(np.asarray(Xtest[index]),cmap=cm.gray)
    plt.xlabel(ytest[25])
    plt.subplot(1,2,2)
    plt.imshow(np.asarray(Xtest[imtest]),cmap=cm.gray)
    plt.xlabel(labelPredicted)
    plt.show()
    print("actual label=" + ytest[imtest] + " label predicted=" + labelPredicted)

    #compute recognition accuracy
    i = 0
    accuracyCount = 0
    for xi in Xtest:
        labelPredicted,index = pcah.predict(EV,xi,mu,y,yseq)
        if (labelPredicted == ytest[i]):
            accuracyCount = accuracyCount+1
        i = i+1
    print("recog accuracy = " + (str)(accuracyCount/i))

    #-------------------------------------------------------------

    #---------------Row version of PCA-----------------------
    #[X,y,yseq] = pcah.read_images("D:/Images/AttDataSet/ATTDataSet/Training/")
    #[Xtest,ytest,yseqt] = pcah.read_images("D:/Images/AttDataSet/ATTDataSet/Testing/")
    ## X is a list f 112x92  2-d arrays, y is a list of numbers
    #[E, EV, mu] = pcah.pca(pcah.asRowMatrix(X), y) # top 100 Eigen vectors
    ## E is the Eigen value array, EV is the catenated Eigen vectors, mu is the mean
image

    ## turn the first (at most) 16 eigenvectors into grayscale
    ## # images (note: eigenvectors are stored by column!)
    #EF16 = []
    #for i in range(min(len(X), 16)):
    #    e = EV[:,i].reshape(X[0].shape)
    #    EF16.append(pcah.normalize(e,0,255))
    #    # plot them and store the plot to "python_eigenfaces.pdf"
    #pcah.subplot(title="Eigenfaces AT&T Facedatabase", images=EF16, rows=4, cols=4,
sptitle=" Eigenface", colormap=cm.jet, filename="python_pca_eigenfaces.png")

    ## reconstruction steps
    #steps=[i for i in range(10, min(len(X), 200), 20)]
    #EF10 = []
    #for i in range(min(len(steps), 16)):
```

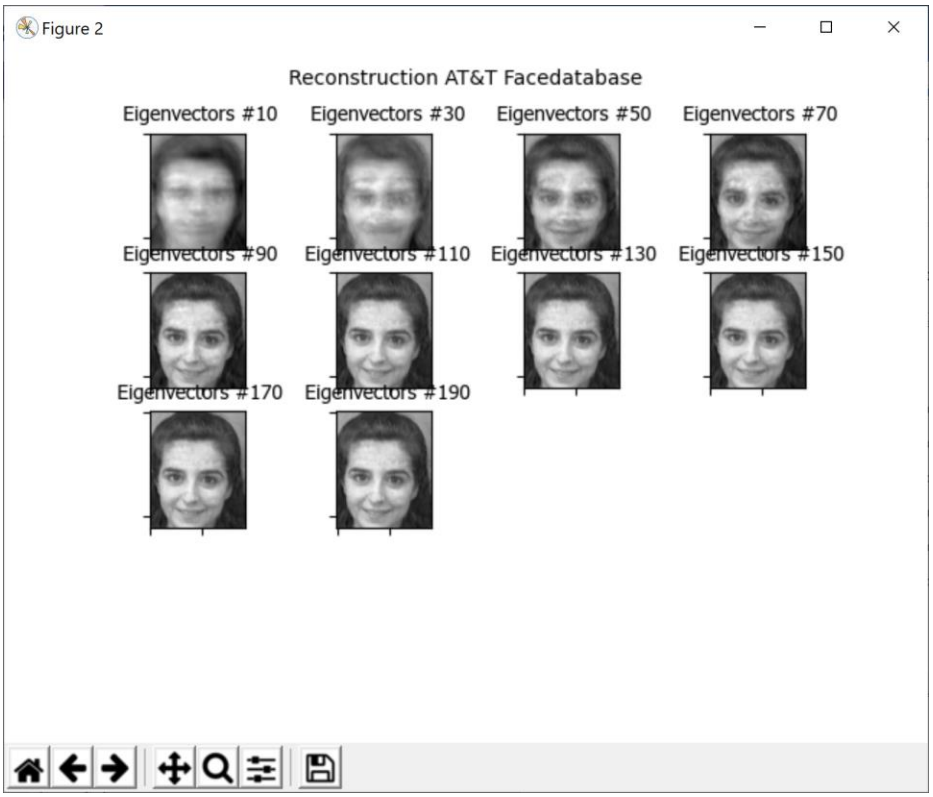
```
    #    numEvs = steps[i]
    #    P = pcah.project(EV[:,0:numEvs], X[0].reshape(1,-1), mu)
    #    R = pcah.reconstruct(EV[:,0:numEvs], P, mu)
    #    # reshape and append to plots
    #    R = R.reshape(X[0].shape)
    #    EF10.append(pcah.normalize(R,0,255))
    #    # plot them and store the plot to "python_reconstruction.pdf"
    #pcah.subplot(title="Reconstruction AT&T Facedatabase", images=EF10, rows=4, cols=4,
    #       sptitle=" Eigenvectors", sptitles=steps , colormap=cm.gray ,
    #       filename=" python_pca_reconstruction.png")

    #for xi in X:
    #    pcah.projections.append(pcah.project(EV, xi.reshape(1,-1), mu))
    #labelPredicted, index = pcah.predict(EV,Xtest[25],mu,y,yseq)
    #plt.figure()
    #plt.subplot(1,2,1)
    #plt.imshow(np.asarray(Xtest[index]),cmap=cm.gray)
    #plt.xlabel(ytest[25])
    #plt.subplot(1,2,2)
    #plt.imshow(np.asarray(Xtest[25]),cmap=cm.gray)
    #plt.xlabel(labelPredicted)
    #plt.show()
    #print("actual label=" + ytest[25] + " label predicted=" + labelPredicted)
    ##compute recognition accuracy
    #i = 0
    #accuracyCount = 0
    #for xi in Xtest:
    #    labelPredicted,index = pcah.predict(EV,xi,mu,y,yseq)
    #    if (labelPredicted == ytest[i]):
    #        accuracyCount = accuracyCount+1
    #    i = i+1
    #print("recog accuracy = " + (str)(accuracyCount/i))

if __name__ == "__main__":
    sys.exit(int(main() or 0))
```

Test the row and the column version of the PCA. Your output will appear as:

Th following output shows test of one unknown picture being matched to one of the pictures of the same person in the known dataset.



The following output shows how different number of PCA components are combined to regenerate an image.



This is how the different Principle components look like for the ATT dataset.