assignment_eigenfaces

October 17, 2016

1 [WS14/15] Mathematics for Robotics and Control: Assignment 003 - Eigenfaces

First we will setup this notebook so that figures and plots can be shown in the notebook page.

```
In [9]: try:
            shell = get_ipython()
            shell.enable_pylab("inline")
        except NameError:
            pass

        import numpy
        import matplotlib
        from matplotlib import pylab, mlab, pyplot
        np = numpy
        plt = pyplot

        from IPython.display import display
        from IPython.core.pylabtools import figsize, getfigs
        import IPython
        from pylab import *
        from numpy import *
```

Hint: Before you start solving the assignment, you might want to check the following *numpy* functions:

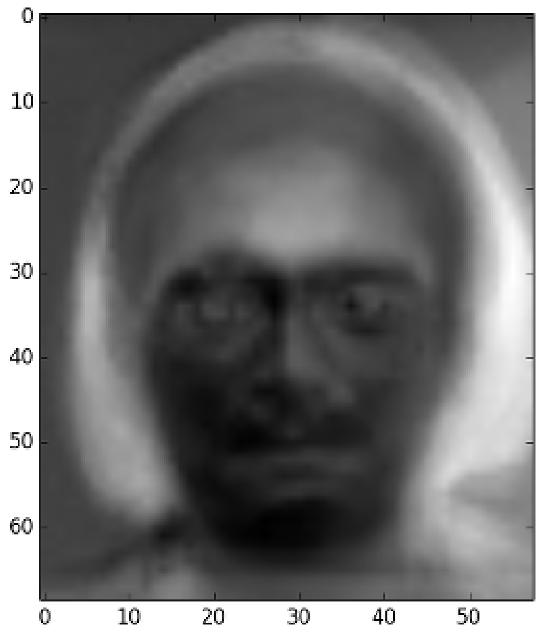
```
PIL.Image.open
scipy.misc.imresize
scipy.spatial.distance
numpy.linalg.eigh
```

1.1 Eigenfaces

Eigenvectors have many applications which are not limited to obtaining surface normals from a set of point clouds. In this assignment, you are asked to write your own little facial recognition library. Take a look at the following image:

```
In [10]: IPython.core.display.Image("images/ef0001.png")
```

```
Out[10]:
```



This is what is called an *eigenface*. An eigenface really is nothing else than an eigenvector, in this case reshaped for plotting. Eigenfaces can be used in facial recognition, allowing a robot to distinguish between different persons, but it can also be applied to other use cases such as voice or gesture recognition.

1. **Read the Scholarpedia article about Eigenfaces.**
2. **Implement the eigenface algorithm described in the Scholarpedia article in a Python object (a class) that exposes (at least) two methods:**
   1. **A method calculating eigenfaces given a number of images and a subject id uniquely identifying the subject in the picture as parameters.**
   2. **A method returning the subject id of the most similar face, for each face in a list of query faces given as parameter.**
3. ** Dataset for the assignment has been provided in the "data" folder. Images in the test folder should be used for enrolling the faces. The validity of the working of the eigen based face recognition should be tested using the images in the training folder**

Use the signatures shown below for your implementation. The ** recognize_face ** function returns the variable of the recognized ids which is of datatype **list**. Please refer the test case provided for loading the images and testing your code.

Hint: You might want to research how Principal Component Analysis (PCA) works.

```
In [21]: class FaceRecognition(object):
             def enroll_faces(self, image_filenames, subject_ids):
                 #Implement code here
                 from PIL import Image
                 # create an empty array of 45045 side to store
                 #array of all images in single array.
                 self.X_matrix = np.empty((45045))
                 # All images are converted into np.array
                 #and stacked into single matrix.
                 for i in image_filenames:
                     img = Image.open(i)
                     imggray = img.convert('LA')
                     img_matrix = np.array(list(imggray.getdata(band=0)), float)
                     #print img_matrix
                     self.X_matrix = np.row_stack((self.X_matrix,img_matrix))

                 #removing extra zero rows.
                 self.X_matrix = np.delete(self.X_matrix,(0), axis=0)
                 #print X_matrix.shape

                 #Now compute mean of all images in X_matrix.
                 mean_oftrained_images = np.mean(self.X_matrix,0)
                 #print mean.shape

                 #Compute adjusted  mean.
                 mean_adjust = self.X_matrix - mean_oftrained_images
                 #print mean_adjust.shape
                 #print mean_adjust.T.shape

                 # Now applying SVD
                 #mean_adjust.T is trasposed so all eigen faces are in U matrix.
                 # full_matrices = False because when it is true it gives
                 #memory error.
                 U,S,V = np.linalg.svd(mean_adjust.T, full_matrices=False)
                 self.U = U
                 #print U.shape
                 #plt.figure(figsize=(4,4))

                 #plt.imshow(U[:,0].reshape(231,195),cmap='gray')

                 #Computing weights of all trained images.
                 self.weight_oftrained_images = np.dot(mean_adjust,self.U)
                 #print weight.shape

                 #Reconstruct image according to given weights.
                 recon = mean_oftrained_images + \
                 np.dot(self.weight_oftrained_images[0,:],self.U.T)
                 #plt.imshow(recon.reshape(231,195), cmap='gray')
```

```
        self.subject_ids = subject_ids



        #raise NotImplementedError()

    def recognize_faces(self, image_filenames):
        from PIL import Image

        from scipy.spatial import distance


        #create empty matrix and convert all images into +
        #array, stack them row-wise.
        Y_matrix = np.empty((45045))
        for j in image_filenames:
            img = Image.open(j)
            imggray = img.convert('LA')
            img_matrix = np.array(list(imggray.getdata(band=0)), float)
            Y_matrix = np.row_stack((Y_matrix,img_matrix))

        Y_matrix = np.delete(Y_matrix,(0), axis=0)
        #compute mean of test folder images.
        mean_ofcurrent_image = np.mean(Y_matrix,0)
        #Adjusted  mean of test folder images.
        mean_adjust = Y_matrix - mean_ofcurrent_image

        # Now applying SVD
        U,S,V = np.linalg.svd(mean_adjust.T, full_matrices=False)
        #print U.shape

        #Computing weights by dot product of adjusted mean and
        #eigen vector(U matrix).
        weight_ofcurrent_images = np.dot(mean_adjust,self.U)

        #Getting length of weight matrix of test +\
        #images so can be used for loop.
        length_of_current_images = len(weight_ofcurrent_images)
        #print length_of_current_images
        #weights in test image are 55.
        #Getting length of weight matrix of trained images
        length_of_trained_images = len(self.weight_oftrained_images)
        #weights of trained images are 110.

        #print length_of_trained_images

        #Computing Eucleadian distance of --
        #current image and trained image
```

```
# So length of Euclidean distance ---
#matrix will be 55 *110.

#First weight of 1st image in test images ----
#is taken, and euclidean distance is computed + \
#with weight of every image in trained image.

outer_eucl_dist_matrix = np.array([0])
for k in range(length_of_current_images):
    inner_eucl_dist_matrix = np.array([0])

    for m in range(length_of_trained_images):
        euc_matrix = +\
        distance.euclidean(weight_ofcurrent_images[k,:],+\
                           self.weight_oftrained_images[m,:])
        #print euc_matrix
        if np.all(inner_eucl_dist_matrix == 0):
            inner_eucl_dist_matrix = +\
            np.add(inner_eucl_dist_matrix, np.array(euc_matrix))
        else:
            inner_eucl_dist_matrix = +\
            np.column_stack((inner_eucl_dist_matrix,+\
                             np.array(euc_matrix)))

    if np.all(outer_eucl_dist_matrix == 0):
        outer_eucl_dist_matrix = +\
        np.add(outer_eucl_dist_matrix,+\
               np.array(inner_eucl_dist_matrix))
    else:
        outer_eucl_dist_matrix = +\
        np.row_stack((outer_eucl_dist_matrix,+\
                      np.array(inner_eucl_dist_matrix)))
#print outer_eucl_dist_matrix.shape

# Now for recognized Ids.
#Each row in outer_eucl_dist_matrix has 110 ids,
#so we need to take index of minimum image.

index = outer_eucl_dist_matrix.argmin(axis = 1)
recognized_ids = np.array([0])
for m in index:
    if np.all (recognized_ids == 0):
        recognized_ids = +\
        np.add(recognized_ids, np.array(self.subject_ids[m]))
    else:
        recognized_ids = +\
        np.column_stack((recognized_ids,+\
                         np.array(self.subject_ids[m])))
```

```
                print recognized_ids

                return recognized_ids  #This is of list datatype
                #raise NotImplementedError()
```

```
In [ ]:
```

2 Testing your code

```
In [22]: import os
         import glob
         import numpy as np
         import operator


         #Loading the Training dataset images
         training_image_filenames = sorted(glob.iglob('./data/training/*.pgm'))

         #Loading the Test dataset images
         test_image_filenames = sorted(glob.iglob('./data/test/*.pgm'))

         #creating a Lambda function to extract the filename
         #the filename of the image is the subject id
         subject_number = lambda filename: int(os.path.basename(filename)[7:9])

         #extracting the filename using the lambda function
         train_subject_ids = list (map(subject_number, training_image_filenames))
         test_subject_ids = list (map(subject_number, test_image_filenames))

         print (test_subject_ids)
         face_recognition = FaceRecognition()

         face_recognition.enroll_faces(training_image_filenames, train_subject_ids)
         recognized_ids = face_recognition.recognize_faces(test_image_filenames)

         different_results = np.array(test_subject_ids) - np.array(recognized_ids)
         print (different_results)
         positives = (different_results == 0).sum()

         assert positives >= 41
```

```
[1, 1, 1, 1, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 6, 6, 6, 6, 6, 7, 7, 7, 7, 8, 8,
[[ 1  1  2  1  2  3  3 14  4  4 14  4  5  5  9  5  6  6  2  6  6  7  7  7
   7  8  2  8  9  9  9  9  9 10 10 10 11 11 11 11 11 12 12 12 13 13 14 14
   4 14 15  3 15 15 15]]
[[  0   0  -1   0   0   0   0 -11   0   0 -10   0   0   0  -4   0   0   0
    4   0   0   0   0   0   0   0   6   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0   0   0  10   0   0  12   0   0
    0]]
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```