



















































from matplotlib import pyplot as plt

from matplotlib.image import imread

import numpy as np

import os

TRAIN\_IMG\_FOLDER = (r"/content/sample\_data/Training images")

TEST\_IMG\_FOLDER =  (r"/content/sample\_data/TEST")

train\_set\_files = os.listdir(TRAIN\_IMG\_FOLDER)

test\_set\_files = os.listdir(TEST\_IMG\_FOLDER)

width  = 128

height = 128

train\_id\_file = set([f.split('\_')[0] for f in train\_set\_files])

test\_id\_file = set([f.split('\_')[0] for f in train\_set\_files])

print(train\_id\_file <= test\_id\_file)

print('Train Images:')

train\_image\_names = os.listdir(TRAIN\_IMG\_FOLDER)

training\_tensor   = np.ndarray(shape=(len(train\_image\_names), height\*width), dtype=np.float64)

for i in range(len(train\_image\_names)):

    img = plt.imread(os.path.join(TRAIN\_IMG\_FOLDER, train\_image\_names[i]))

    training\_tensor[i,:] = np.array(img, dtype='float64').flatten()

    plt.subplot(5,5,1+i)

    plt.imshow(img, cmap='gray')

    plt.tick\_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')

plt.show()

print('Test Images:')

test\_image\_names = os.listdir(TEST\_IMG\_FOLDER)#[i for i in dataset\_dir if i not in train\_image\_names]

testing\_tensor   = np.ndarray(shape=(len(test\_image\_names), height\*width), dtype=np.float64)

for i in range(len(test\_image\_names)):

    img = imread(os.path.join(TEST\_IMG\_FOLDER, test\_image\_names[i]))

    testing\_tensor[i,:] = np.array(img, dtype='float64').flatten()

    plt.subplot(8,5,1+i)

    plt.imshow(img, cmap='gray')

    plt.subplots\_adjust(right=1.2, top=1.2)

    plt.tick\_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')

plt.show()

mean\_face = np.zeros((1,height\*width))

for i in training\_tensor:

    mean\_face = np.add(mean\_face,i)

mean\_face = np.divide(mean\_face,float(len(train\_image\_names))).flatten()

plt.imshow(mean\_face.reshape(height, width), cmap='gray')

plt.tick\_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')

plt.show()

normalised\_training\_tensor = np.ndarray(shape=(len(train\_image\_names), height\*width))

for i in range(len(train\_image\_names)):

    normalised\_training\_tensor[i] = np.subtract(training\_tensor[i],mean\_face)

for i in range(len(train\_image\_names)):

    img = normalised\_training\_tensor[i].reshape(height,width)

    plt.subplot(5,5,1+i)

    plt.imshow(img, cmap='gray')

    plt.tick\_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')

plt.show()

cov\_matrix=np.cov(normalised\_training\_tensor)

cov\_matrix = np.divide(cov\_matrix,25.0)

print('Covariance Matrix Shape:', cov\_matrix.shape)

#print('Covariance matrix of X: \n%s' %cov\_matrix)

#eigenvalues and eigenvectors

eigenvalues, eigenvectors, = np.linalg.eig(cov\_matrix)

print('eigenvalues.shape: {} eigenvectors.shape: {}'.format(eigenvalues.shape, eigenvectors.shape))

eig\_pairs = [(eigenvalues[index], eigenvectors[:,index]) for index in range(len(eigenvalues))]

# Sort the eigen pairs in descending order:

eig\_pairs.sort(reverse=True)

eigvalues\_sort  = [eig\_pairs[index][0] for index in range(len(eigenvalues))]

eigvectors\_sort = [eig\_pairs[index][1] for index in range(len(eigenvalues))]

sorted\_ind = sorted(range(eigenvalues.shape[0]), key=lambda k: eigenvalues[k], reverse=True)

eigvalues\_sort = eigenvalues[sorted\_ind]

eigvectors\_sort = eigenvectors[sorted\_ind]

train\_set\_files\_sort = np.array(train\_set\_files)[sorted\_ind]

var\_comp\_sum = np.cumsum(eigvalues\_sort)/sum(eigvalues\_sort)

# Show cumulative proportion of varaince with respect to components

print("Cumulative proportion of variance explained vector: \n%s" %var\_comp\_sum)

# x-axis for number of principal components kept

num\_comp = range(1,len(eigvalues\_sort)+1)

plt.title('Cum. Prop. Variance Explain and Components Kept')

plt.xlabel('Principal Components')

plt.ylabel('Cum. Prop. Variance Expalined')

plt.scatter(num\_comp, var\_comp\_sum)

plt.show()

reduced\_data = np.array(eigvectors\_sort[:25]).transpose()

reduced\_data.shape

print(training\_tensor.transpose().shape, reduced\_data.shape)

proj\_data = np.dot(training\_tensor.transpose(),reduced\_data)

proj\_data = proj\_data.transpose()

proj\_data.shape

for i in range(proj\_data.shape[0]):

    img = proj\_data[i].reshape(height,width)

    plt.subplot(5,5,1+i)

    plt.imshow(img, cmap='gray')

    plt.tick\_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')

plt.show()

w = np.array([np.dot(proj\_data,i) for i in normalised\_training\_tensor])

print(w.shape)

def recogniser(test\_image\_names, train\_image\_names,proj\_data,w, t0=2e8, prn=False):

    count        = 0

    num\_images   = 0

    correct\_pred = 0

    result = []

    wts = []

    #False match rate (FMR)

    FMR\_count = 0

    #False non-match rate (FNMR)

    FNMR\_count = 0

    test\_image\_names2 = sorted(test\_image\_names)

    for img in test\_image\_names2:

        #img = imread(os.path.join(TEST\_IMG\_FOLDER, test\_image\_names[i]))

        unknown\_face = plt.imread(os.path.join(TEST\_IMG\_FOLDER, img))

        num\_images += 1

        unknown\_face\_vector = np.array(unknown\_face, dtype='float64').flatten()

        normalised\_uface\_vector = np.subtract(unknown\_face\_vector,mean\_face)

        w\_unknown = np.dot(proj\_data, normalised\_uface\_vector)

        diff  = w - w\_unknown

        norms = np.linalg.norm(diff, axis=1)

        index = np.argmin(norms)

        wts.append([count, norms[index]])

        if prn: print('Input:'+'.'.join(img.split('.')[:2]), end='\t')

        count+=1

        match = img.split('\_')[0] == train\_image\_names[index].split('\_')[0]

        if norms[index] < t0: # It's a face

            if match:

                if prn: print('Matched:' + train\_image\_names[index], end = '\t')

                correct\_pred += 1

                result.append(1)

            else:

                if prn: print('F/Matched:'+train\_image\_names[index], end = '\t')

                result.append(0)

                FMR\_count += 1

        else:

            if match:

                if prn: print('Unknown face!'+train\_image\_names[index], end = '\t')

                FNMR\_count +=1

            else:

                pass

                correct\_pred += 1

        if prn: print(norms[index], end=' ')

        if prn: print()

    FMR = FMR\_count/num\_images

    FNMR = FNMR\_count/num\_images

    print('Correct predictions: {}/{} = {} \t\t'.format(correct\_pred, num\_images, correct\_pred/num\_images), end=' ')

    print('FMR: {} \t'.format(FMR), end=' ')

    print('FNMR: {} \t'.format(FNMR))

    return wts, result, correct\_pred, num\_images, FMR, FNMR

wts, result, correct\_pred, num\_images, FMR, FNMR =recogniser(test\_image\_names, train\_image\_names,proj\_data,w, t0=2e8, prn=True)

def rg(r):

    if r: return 'g'

    else: return 'r'

cl = [rg(r) for r in result]

x=[x[0] for x in wts]

y=[y[1] for y in wts]

plt.scatter(x,y, color=cl, label = 'Distance measure (true ang false pred.)')

x2=[x[0] for x in wts]

y2=[2.7e7 for y in wts]

plt.plot(x2,y2, label = 'Empirical error threshold')

plt.legend()

plt.grid()

plt.show()

CPR\_list, t0\_list, FMR\_list, FNMR\_list = [], [] , [] , []

for t0 in np.linspace(start=0, stop=1e8, num=20):

    print('{:e}'.format(t0), end=' ')

    wts, result, correct\_pred, num\_images, FMR, FNMR = recogniser(test\_image\_names, train\_image\_names,proj\_data,w, t0)

    CPR\_list.append(correct\_pred/num\_images)

    t0\_list.append(t0)

    FMR\_list.append(FMR)

    FNMR\_list.append(FNMR)

x1=t0\_list

y1=FMR\_list

x2=t0\_list

y2=FNMR\_list

x3=t0\_list

y3=CPR\_list

plt.plot(x1,y1, ls='--', color='r', label='FMR',)

plt.plot(x2,y2, ls='-.', color='b', label='FNMR')

plt.plot(x3,y3, color='g', label='Correct prediction using threshold')

plt.grid()

plt.legend()

count        = 0

num\_images   = 0

correct\_pred = 0

def Visualization(img, train\_image\_names,proj\_data,w, t0):

    global count,highest\_min,num\_images,correct\_pred

    unknown\_face        = plt.imread(os.path.join(TEST\_IMG\_FOLDER, img))

    num\_images          += 1

    unknown\_face\_vector = np.array(unknown\_face, dtype='float64').flatten()

    normalised\_uface\_vector = np.subtract(unknown\_face\_vector,mean\_face)

    plt.subplot(40,2,1+count)

    plt.imshow(unknown\_face, cmap='gray')

    plt.title('Input:'+'.'.join(img.split('.')[:2]))

    plt.tick\_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')

    count+=1

    w\_unknown = np.dot(proj\_data, normalised\_uface\_vector)

    diff  = w - w\_unknown

    norms = np.linalg.norm(diff, axis=1)

    index = np.argmin(norms)

    plt.subplot(40,2,1+count)

    if norms[index] < t0: # It's a face

        match = img.split('\_')[0] == train\_image\_names[index].split('\_')[0]

        #if img.split('.')[0] == train\_image\_names[index].split('.')[0]:

        if match:

            #plt.title('Matched:'+'.'.join(train\_image\_names[index].split('.')[:2]), color='g')

            plt.title('Matched:', color='g')

            plt.imshow(imread(os.path.join(TRAIN\_IMG\_FOLDER, train\_image\_names[index])), cmap='gray')

             #img = plt.imread(os.path.join(TRAIN\_IMG\_FOLDER, train\_image\_names[i]))

            correct\_pred += 1

        else:

            #plt.title('Matched:'+'.'.join(train\_image\_names[index].split('.')[:2]), color='r')

            plt.title('False matched:', color='r')

            plt.imshow(imread(os.path.join(TRAIN\_IMG\_FOLDER, train\_image\_names[index])), cmap='gray')

    else:

        #if img.split('.')[0] not in [i.split('.')[0] for i in train\_image\_names] and img.split('.')[0] != 'apple':

        if img.split('\_')[0] not in [i.split('\_')[0] for i in train\_image\_names]:

            plt.title('Unknown face', color='g')

            correct\_pred += 1

        else:

            plt.title('Unknown face', color='r')

    plt.tick\_params(labelleft='off', labelbottom='off', bottom='off',top='off',right='off',left='off', which='both')

    plt.subplots\_adjust(right=1.2, top=2.5)

    count+=1

fig = plt.figure(figsize=(5, 30))

test\_image\_names2 = sorted(test\_image\_names)

for i in range(len(test\_image\_names2)):

    Visualization(test\_image\_names2[i], train\_image\_names,proj\_data,w, t0=2.7e7)

plt.show()