CS-663 Assignment 4 Q4

Soham Naha (193079003)
Akshay Bajpai (193079002)
Mohit Agarwala (19307R004)

November 6, 2020

4

In this part, you will implement a mini face recognition system. Download the ORL face database from the homework folder. It contains 40 sub-folders, one for each of the 40 subjects/persons. For each person, there are ten images in the appropriate folder named 1.pgm to 10.pgm. The images are of size 92 by 110 each. Each image is in the pgm format. You can view the images in this format, either through MATLAB or through image viewers like IrfanView on Windows, or xv/display/gimp on Unix. Though the face images are in different poses, expressions and facial accessories, they are all roughly aligned (the eyes are in roughly similar locations in all images). For the first part of the assignment, you will work with the images of the first 32 people. For each person, you will include the first six images in the training set (that is the first 6 images that appear in a directory listing as produced by the dir function of MATLAB) and the remaining four images in the testing set. You implement the recognition system by using the svd function of MATLAB on an appropriate data matrix. Record the recognition rate using squared difference between the eigencoefficients while testing on all the images in the test set, for $k \in \{1, 2, 3, 5, 10, 15, 20, 30, 50, 75, 100, 150, 170\}$. Plot the rates in your report in the form of a graph. Now modify the required few lines of the code but using the eig function of MATLAB (on the L matrix as defined in class) instead of svd.

Repeat the same experiment (using just the svd routine) on the Yale Face database from the homework folder. This database contains 64 images each of 38 individuals (labeled from 1 to 39, with number 14 missing). Each image is in pgm format and has size 192 by 168. The images are taken under different lighting conditions but in the same pose. Take the first 40 images of every person for training and test on the remaining 24 images (that is the first 40 images that appear in a directory listing as produced by the dir function of MATLAB). Plot in your report the recognition rates for $k \in \{1, 2, 3, 5, 10, 15, 20, 30, 50, 60, 65, 75, 100, 200, 300, 500, 1000\}$ based on (a) the squared difference between all the eigen coefficients and (b) the squared difference between all *except* the three eigen coefficients corresponding to the eigen vectors with the three largest eigenvalues. [30 points]

myFaceRecognizer.py

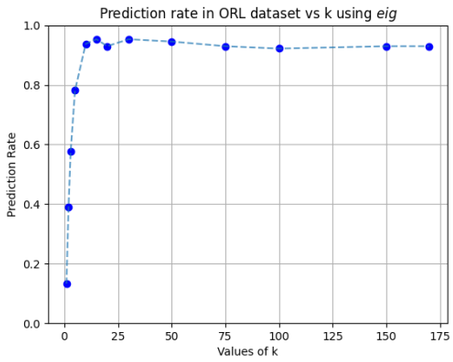
```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
import sys

def find_mean(array):
    """Find the mean of train data
    :param array : the train image matrix
    :output mean : the mean image vector
    """
    mean = np.mean(array,axis=0)
    return mean

def im2double(im):
```

```
    """Normalizes the image to be in range 0-1
    :param im : input image
    :output out : min-max normalized output
    """
    min_val = np.min(im)
    max_val = np.max(im)
    out = (im.astype('float') - min_val) / (max_val - min_val)

    return out

def read_train_images_ORL(file_path,subjects,images_per_subject):
    """Reads the ORL train image data and returns a stacked array
    :param file_path: the folder path of ORL dataset
    :param subjects: the number of subjects under training consideration
    :param images_per_subject: the number of images to consider for each
                               test subject
    :output image_array: stacked array output
    """
    image_array = []
    for i in range(1,subjects+1):
        folder_path = os.path.join(file_path,"s"+str(i))
        #print(folder_path)
        for j in range(1,images_per_subject+1):
            filepath = os.path.join(folder_path,str(j)+".pgm")
            im = cv2.imread(filepath,0)
            image_array.append(im2double(im).ravel())

    return np.array(image_array)

def read_test_images_ORL(file_path,subjects,images_per_subject,starting_index):
    """Reads the ORL test image data and returns a stacked array
    :param file_path: the folder path of ORL dataset
    :param subjects: the number of subjects under test
    :param images_per_subject: the number of images to consider for each
                               test subject
    :param starting_index: starting image number for test images
    :output image_array: stacked array output
    """
    image_array = []
    for i in range(1,subjects+1):
        folder_path = os.path.join(file_path,"s"+str(i))
        for j in range(starting_index,starting_index+images_per_subject):
            filepath = os.path.join(folder_path,str(j)+".pgm")
            im = cv2.imread(filepath,0)
            image_array.append(im2double(im).ravel())

    return np.array(image_array)

def read_train_images_YALE(file_path,images_per_subject):
    """Reads the CroppedYale train image data and returns a stacked array
    :param file_path: the folder path of ORL dataset
    :param images_per_subject: the number of images to consider for each
                               test subject
    :output image_array: stacked array output
    """
    image_array = []
    folder_list = os.listdir(file_path)
    for i in folder_list:
        folder_path = os.path.join(file_path,i)
        files = os.listdir(folder_path)
        for j in files[:images_per_subject]:
            filepath = os.path.join(folder_path,j)
            im = cv2.imread(filepath,0)
            image_array.append(im2double(im).ravel())
    image_array = np.array(image_array)
```

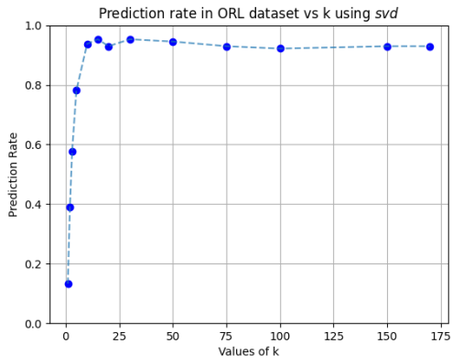
```
    return image_array

def read_test_images_YALE(file_path, starting_index):
    """Reads the CroppedYale train image data and returns a stacked array
    :param file_path: the folder path of ORL dataset
    :param starting_index: the starting number of the images to consider for each
                                test subject
    :output image_array: stacked array output
    """
    image_array = []
    folder_list = os.listdir(file_path)
    for i in folder_list:
        folder_path = os.path.join(file_path,i)
        files = os.listdir(folder_path)
        for j in files[starting_index:]:
            filepath = os.path.join(folder_path,j)
            im = cv2.imread(filepath,0)
            image_array.append(im2double(im).ravel())
    image_array = np.array(image_array)

    return image_array



def subtract_mean(mean,train,test):
    """Subtracts the mean of the training data from both the
    train and the test stacked array
    :param mean: the mean of the training data
    :param train: the train images stack array
    :param test: the test images stack array
    :output train_mean: mean subtracted train stack
    :output test_mean: mean subtracted test stack
    """
    train_mean = train-mean
    test_mean = test-mean

    return train_mean, test_mean

def normalize_vecs(array):
    """Normalizes the Unitary vector matrix
    :param array: the unitary matrix of vectors from svd
    :output array: the normalized array
    """
    _,c = array.shape
    for i in range(c):
        array[:,i] = array[:,i]/(np.sqrt(np.sum(np.square(array[:,i]))))

    return array

def calculate_svd(array):
    """Calculates svd of the train stack
    :param array: the train stack
    :output U: the left unitary matrix of the svd output
    """
    U,sigma,V_T = np.linalg.svd(array.T,full_matrices=False)
    U = normalize_vecs(U)
    return U

def calculate_evd(array):
    """Calculates the eigen value and eigen vectors and returns the eigen vectors
    :param array: the train stack
    :output V: the normalized unitary vector matrix
    """
    L_train = np.matmul(array,array.T)
```

```
    eig_val,W = np.linalg.eig(L_train)
    V = np.matmul(array.T,W)
    V = normalize_vecs(V)

    return V

def calculate_alpha(V,train,test):
    """Calculates the reconstruction matrix alpha for train and test images
    :param V: the unitary matrix from decomposition of train stack
    :param train: the train stack
    :param test: the test stack
    """
    alpha_train = np.matmul(V.T,train.T)
    alpha_test = np.matmul(V.T,test.T)

    return alpha_train, alpha_test

def calculate_and_plot_prediction_rates(train, test, alpha_train, alpha_test, ks, \
                                        dataset, method, light=False):
    """Calculates the prediction rates and plots according to the ks supplied
    :param train: the train stack
    :param test: the test stack
    :param alpha_train: the reconstruction coefficient matrix for the train images
    :param alpha_test: the reconstruction coefficient matrix for test images
    :param ks: the list of k (#of coefficients of reconstruction) to consider
    :param dataset: the name of the dataset under consideration(YALE or ORL)
    :param method: the method used to calculate the unitary matrix (eig or svd)
    :param light: whether to discard the lighting effects of the image
                  (default:  False: means not to discard)
    """
    prediction_rate = []
    for k in ks:
        correct_prediction_count = 0
        for counter, ele in enumerate(alpha_test.T):
            if light:
                index = np.argmin(np.sum(np.square((alpha_train.T[:,3:3+k]-alpha_test.T[counter,
                                                                3:3+k])),axis=1))
            else:
                index = np.argmin(np.sum(np.square((alpha_train.T[:,:k]-alpha_test.T[counter,:k]))
                                                ,axis=1))
            if (counter//test == index//train):
                correct_prediction_count += 1
        prediction_rate.append(correct_prediction_count/float(alpha_test.shape[1]))

    if light:
            print("Prediction Rate for dataset {} using process {} without light is
                {}.".format(dataset,method,prediction_rate))
    else:
        print("Prediction Rate for dataset {} using process {} is
                {}.".format(dataset,method,prediction_rate))

    plt.figure()
    plt.plot(ks,prediction_rate,"bo")
    plt.plot(ks,prediction_rate,alpha=0.7,linestyle='dashed')
    plt.ylabel('Prediction Rate')
    plt.xlabel('Values of k')
    plt.ylim(ymin=0,ymax=1)
    if light:
        plt.title(r"Prediction rate in {} dataset vs k using ${}$ removing lighting effects".format(dataset,method
    else:
        plt.title(r"Prediction rate in {} dataset vs k using ${}$".format(dataset,method))
    plt.grid()
    if light:
        plt.savefig("../images/PredRate_"+dataset+"_"+method+"_withoutLighting.png")
    else:
```

```
        plt.savefig("../images/PredRate_"+dataset+"_"+method+"_normal.png")



def ORL_data(path, subjects, train_images_per_subject, test_images_per_subject, ks):
    """The operation pipeline for the training and testing in ORL Dataset
    :param path: path to ORL dataset
    :param subjects: the number of subjects under training consideration
    :param train_images_per_subject: the number of images to consider for each
                                train subject
    :param test_images_per_subject: the number of images to consider for each
                                test subject
    :param ks: the list of k (#of coefficients of reconstruction) to consider
    """
    train_images = read_train_images_ORL(path,subjects,train_images_per_subject)
    test_images = read_test_images_ORL(path,subjects,test_images_per_subject, \
                                       starting_index=train_images_per_subject+1)
    train_mean = find_mean(train_images)
    train_mean_subtracted, test_mean_subtracted = subtract_mean(train_mean, train_images, \
                                                    test_images)
    eigen_vectors = calculate_evd(train_mean_subtracted)
    alpha_eig_train, alpha_eig_test = calculate_alpha(eigen_vectors, train_mean_subtracted, \
    test_mean_subtracted)
    calculate_and_plot_prediction_rates(train_images_per_subject,test_images_per_subject, \
                                        alpha_eig_train, alpha_eig_test,
                                        ks,dataset="ORL",method="eig")
    unitary_vectors = calculate_svd(train_mean_subtracted)
    alpha_svd_train, alpha_svd_test = calculate_alpha(unitary_vectors, train_mean_subtracted,
                                test_mean_subtracted)
    calculate_and_plot_prediction_rates(train_images_per_subject,test_images_per_subject, \
                                        alpha_svd_train, alpha_svd_test,
                                        ks,dataset="ORL",method="svd")


def YALE_data(path, train_images_per_subject, test_images_per_subject, ks):
    """The operation pipeline for the training and testing in CroppedYale Dataset
    :param path: path to CroppedYale dataset
    :param train_images_per_subject: the number of images to consider for each
                                train subject
    :param test_images_per_subject: the number of images to consider for each
                                test subject
    :param ks: the list of k (#of coefficients of reconstruction) to consider
    """
    train_images = read_train_images_YALE(path,train_images_per_subject)
    test_images = read_test_images_YALE(path, starting_index=train_images_per_subject)
    train_mean = find_mean(train_images)
    train_mean_subtracted, test_mean_subtracted = subtract_mean(train_mean, train_images, test_images)
    unitary_vectors = calculate_svd(train_mean_subtracted)
    alpha_svd_train, alpha_svd_test = calculate_alpha(unitary_vectors, train_mean_subtracted, \
                                test_mean_subtracted)
    calculate_and_plot_prediction_rates(train_images_per_subject, test_images_per_subject, \
                        alpha_svd_train, alpha_svd_test, ks, dataset="Yale", method="svd")
    calculate_and_plot_prediction_rates(train_images_per_subject, test_images_per_subject, \
                alpha_svd_train, alpha_svd_test, ks, dataset="Yale", method="svd", light=True)


if __name__=="__main__":
    ORL_PATH = "../../ORL"
    ORL_k = [1, 2, 3, 5, 10, 15, 20, 30, 50, 75, 100, 150, 170]
    ORL_data(ORL_PATH,subjects=32,train_images_per_subject=6,test_images_per_subject=4,ks=ORL_k)
    YALE_PATH = "../../CroppedYale"
    YALE_k = [1,2,3,5,10,15,20,30,50,60, 65,75,100,200,300,500,1000]
    YALE_data(YALE_PATH, train_images_per_subject=40, test_images_per_subject=24, ks=YALE_k)
```

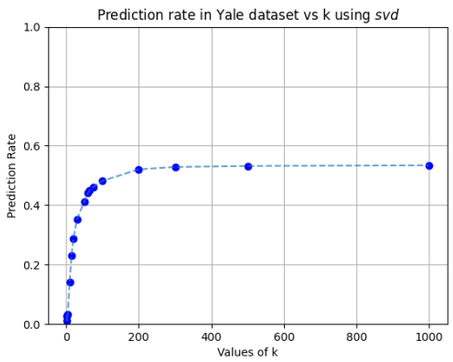


[a]

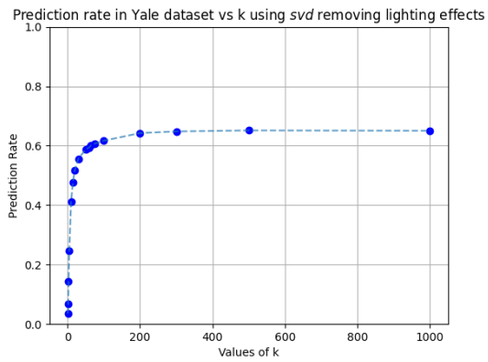


[b]

Figure 1: (a) Prediction Rate vs k for ORL Database using *eig* (b) Prediction Rate vs k for ORL Database using *svd*



[a]



[b]

Figure 2: (a) Prediction Rate vs k for CroppedYale Dataset using *svd* (b) Prediction Rate vs k for CroppedYale Database using *svd* without the first three eigen - vectors

The Recorded Prediction Rates

```
For ORL Dataset :
ks = [1, 2, 3, 5, 10, 15, 20, 30, 50, 75, 100, 150, 170]
Prediction Rate for dataset ORL using process eig is
[0.1328125, 0.390625, 0.578125, 0.78125, 0.9375, 0.953125, 0.9296875, 0.953125, 0.9453125,
0.9296875, 0.921875, 0.9296875, 0.9296875].
Prediction Rate for dataset ORL using process svd is
[0.1328125, 0.390625, 0.578125, 0.78125, 0.9375, 0.953125, 0.9296875, 0.953125, 0.9453125,
0.9296875, 0.921875, 0.9296875, 0.9296875].
**************************************************
For YALE Dataset :
ks = [1,2,3,5,10,15,20,30,50,60, 65,75,100,200,300,500,1000]
Prediction Rate for dataset Yale using process svd is
[0.026815642458100558, 0.027932960893854747, 0.012290502793296089, 0.0335195530726257,
0.1418994413407821,0.23016759776536314, 0.2860335195530726, 0.35195530726256985,
0.4111731843575419, 0.4402234636871508, 0.45027932960893857, 0.46145251396648046,
0.48156424581005586, 0.5206703910614525, 0.5284916201117319,
0.5318435754189944, 0.5340782122905028].
Prediction Rate for dataset Yale using process svd without light is
[0.034636871508379886, 0.06927374301675977, 0.1430167597765363, 0.24581005586592178,
0.4122905027932961, 0.4759776536312849, 0.5162011173184358, 0.5564245810055866,
0.5865921787709497, 0.5944134078212291, 0.6011173184357542, 0.6067039106145251,
0.6167597765363129, 0.6424581005586593, 0.6480446927374302,
0.6513966480446928, 0.6502793296089385].
```