

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

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import cv2
4 import os
5 import sys
6
7 def find_mean(array):
8     """Find the mean of train data
9     :param array : the train image matrix
10    :output mean : the mean image vector
11    """
12    mean = np.mean(array,axis=0)
13    return mean
14
15 def im2double(im):
```

```

16     """Normalizes the image to be in range 0-1
17     :param im : input image
18     :output out : min-max normalized output
19     """
20     min_val = np.min(im)
21     max_val = np.max(im)
22     out = (im.astype('float') - min_val) / (max_val - min_val)
23
24     return out
25
26 def read_train_images_ORL(file_path, subjects, images_per_subject):
27     """Reads the ORL train image data and returns a stacked array
28     :param file_path: the folder path of ORL dataset
29     :param subjects: the number of subjects under training consideration
30     :param images_per_subject: the number of images to consider for each
31     test subject
32     :output image_array: stacked array output
33     """
34     image_array = []
35     for i in range(1, subjects+1):
36         folder_path = os.path.join(file_path, "s"+str(i))
37         #print(folder_path)
38         for j in range(1, images_per_subject+1):
39             filepath = os.path.join(folder_path, str(j)+".pgm")
40             im = cv2.imread(filepath, 0)
41             image_array.append(im2double(im).ravel())
42
43     return np.array(image_array)
44
45 def read_test_images_ORL(file_path, subjects, images_per_subject, starting_index):
46     """Reads the ORL test image data and returns a stacked array
47     :param file_path: the folder path of ORL dataset
48     :param subjects: the number of subjects under test
49     :param images_per_subject: the number of images to consider for each
50     test subject
51     :param starting_index: starting image number for test images
52     :output image_array: stacked array output
53     """
54     image_array = []
55     for i in range(1, subjects+1):
56         folder_path = os.path.join(file_path, "s"+str(i))
57         for j in range(starting_index, starting_index+images_per_subject):
58             filepath = os.path.join(folder_path, str(j)+".pgm")
59             im = cv2.imread(filepath, 0)
60             image_array.append(im2double(im).ravel())
61
62     return np.array(image_array)
63
64 def read_train_images_YALE(file_path, images_per_subject):
65     """Reads the CroppedYale train image data and returns a stacked array
66     :param file_path: the folder path of ORL dataset
67     :param images_per_subject: the number of images to consider for each
68     test subject
69     :output image_array: stacked array output
70     """
71     image_array = []
72     folder_list = os.listdir(file_path)
73     for i in folder_list:
74         folder_path = os.path.join(file_path, i)
75         files = os.listdir(folder_path)
76         for j in files[:images_per_subject]:
77             filepath = os.path.join(folder_path, j)
78             im = cv2.imread(filepath, 0)
79             image_array.append(im2double(im).ravel())
80     image_array = np.array(image_array)

```

```

81
82     return image_array
83
84 def read_test_images_YALE(file_path, starting_index):
85     """Reads the CroppedYale train image data and returns a stacked array
86     :param file_path: the folder path of ORL dataset
87     :param starting_index: the starting number of the images to consider for each
88     test subject
89     :output image_array: stacked array output
90     """
91     image_array = []
92     folder_list = os.listdir(file_path)
93     for i in folder_list:
94         folder_path = os.path.join(file_path,i)
95         files = os.listdir(folder_path)
96         for j in files[starting_index:]:
97             filepath = os.path.join(folder_path,j)
98             im = cv2.imread(filepath,0)
99             image_array.append(im2double(im).ravel())
100     image_array = np.array(image_array)
101
102     return image_array
103
104
105
106 def subtract_mean(mean,train,test):
107     """Subtracts the mean of the training data from both the
108     train and the test stacked array
109     :param mean: the mean of the training data
110     :param train: the train images stack array
111     :param test: the test images stack array
112     :output train_mean: mean subtracted train stack
113     :output test_mean: mean subtracted test stack
114     """
115     train_mean = train-mean
116     test_mean = test-mean
117
118     return train_mean, test_mean
119
120 def normalize_vecs(array):
121     """Normalizes the Unitary vector matrix
122     :param array: the unitary matrix of vectors from svd
123     :output array: the normalized array
124     """
125     _,c = array.shape
126     for i in range(c):
127         array[:,i] = array[:,i]/(np.sqrt(np.sum(np.square(array[:,i]))))
128
129     return array
130
131 def calculate_svd(array):
132     """Calculates svd of the train stack
133     :param array: the train stack
134     :output U: the left unitary matrix of the svd output
135     """
136     U,sigma,V_T = np.linalg.svd(array.T,full_matrices=False)
137     U = normalize_vecs(U)
138     return U
139
140 def calculate_evd(array):
141     """Calculates the eigen value and eigen vectors and returns the eigen vectors
142     :param array: the train stack
143     :output V: the normalized unitary vector matrix
144     """
145     L_train = np.matmul(array,array.T)

```

```

146     eig_val,W = np.linalg.eig(L_train)
147     V = np.matmul(array.T,W)
148     V = normalize_vecs(V)
149
150     return V
151
152 def calculate_alpha(V,train,test):
153     """Calculates the reconstruction matrix alpha for train and test images
154     :param V: the unitary matrix from decomposition of train stack
155     :param train: the train stack
156     :param test: the test stack
157     """
158     alpha_train = np.matmul(V.T,train.T)
159     alpha_test = np.matmul(V.T,test.T)
160
161     return alpha_train, alpha_test
162
163 def calculate_and_plot_prediction_rates(train, test, alpha_train, alpha_test, ks, \
164                                       dataset, method, light=False):
165     """Calculates the prediction rates and plots according to the ks supplied
166     :param train: the train stack
167     :param test: the test stack
168     :param alpha_train: the reconstruction coefficient matrix for the train images
169     :param alpha_test: the reconstruction coefficient matrix for test images
170     :param ks: the list of k (#of coefficients of reconstruction) to consider
171     :param dataset: the name of the dataset under consideration(YALE or ORL)
172     :param method: the method used to calculate the unitary matrix (eig or svd)
173     :param light: whether to discard the lighting effects of the image
174     (default: False: means not to discard)
175     """
176     prediction_rate = []
177     for k in ks:
178         correct_prediction_count = 0
179         for counter, ele in enumerate(alpha_test.T):
180             if light:
181                 index = np.argmin(np.sum(np.square((alpha_train.T[:,3:3+k]-alpha_test.T[counter,
182                                                         3:3+k])),axis=1))
183             else:
184                 index = np.argmin(np.sum(np.square((alpha_train.T[:, :k]-alpha_test.T[counter, :k])),
185                                                         ,axis=1))
186             if (counter//test == index//train):
187                 correct_prediction_count += 1
188         prediction_rate.append(correct_prediction_count/float(alpha_test.shape[1]))
189
190     if light:
191         print("Prediction Rate for dataset {} using process {} without light is
192               {}".format(dataset,method,prediction_rate))
193     else:
194         print("Prediction Rate for dataset {} using process {} is
195               {}".format(dataset,method,prediction_rate))
196
197     plt.figure()
198     plt.plot(ks,prediction_rate,"bo")
199     plt.plot(ks,prediction_rate,alpha=0.7,linestyle='dashed')
200     plt.ylabel('Prediction Rate')
201     plt.xlabel('Values of k')
202     plt.ylim(ymin=0,ymax=1)
203     if light:
204         plt.title(r"Prediction rate in {} dataset vs k using {}$ removing lighting effects".format(dataset,method))
205     else:
206         plt.title(r"Prediction rate in {} dataset vs k using {}$".format(dataset,method))
207     plt.grid()
208     if light:
209         plt.savefig("../images/PredRate_"+dataset+"_"+method+"_withoutLighting.png")
210     else:

```

```

211     plt.savefig("../images/PredRate_"+dataset+"_"+method+"_normal.png")
212
213
214
215 def ORL_data(path, subjects, train_images_per_subject, test_images_per_subject, ks):
216     """The operation pipeline for the training and testing in ORL Dataset
217     :param path: path to ORL dataset
218     :param subjects: the number of subjects under training consideration
219     :param train_images_per_subject: the number of images to consider for each
220     train subject
221     :param test_images_per_subject: the number of images to consider for each
222     test subject
223     :param ks: the list of k (#of coefficients of reconstruction) to consider
224     """
225     train_images = read_train_images_ORL(path, subjects, train_images_per_subject)
226     test_images = read_test_images_ORL(path, subjects, test_images_per_subject, \
227                                       starting_index=train_images_per_subject+1)
228     train_mean = find_mean(train_images)
229     train_mean_subtracted, test_mean_subtracted = subtract_mean(train_mean, train_images, \
230                                                                test_images)
231     eigen_vectors = calculate_evd(train_mean_subtracted)
232     alpha_eig_train, alpha_eig_test = calculate_alpha(eigen_vectors, train_mean_subtracted, \
233                                                       test_mean_subtracted)
234     calculate_and_plot_prediction_rates(train_images_per_subject, test_images_per_subject, \
235                                       alpha_eig_train, alpha_eig_test,
236                                       ks, dataset="ORL", method="eig")
237     unitary_vectors = calculate_svd(train_mean_subtracted)
238     alpha_svd_train, alpha_svd_test = calculate_alpha(unitary_vectors, train_mean_subtracted,
239                                                       test_mean_subtracted)
240     calculate_and_plot_prediction_rates(train_images_per_subject, test_images_per_subject, \
241                                       alpha_eig_train, alpha_eig_test,
242                                       ks, dataset="ORL", method="svd")
243
244
245 def YALE_data(path, train_images_per_subject, test_images_per_subject, ks):
246     """The operation pipeline for the training and testing in CroppedYale Dataset
247     :param path: path to CroppedYale dataset
248     :param train_images_per_subject: the number of images to consider for each
249     train subject
250     :param test_images_per_subject: the number of images to consider for each
251     test subject
252     :param ks: the list of k (#of coefficients of reconstruction) to consider
253     """
254     train_images = read_train_images_YALE(path, train_images_per_subject)
255     test_images = read_test_images_YALE(path, starting_index=train_images_per_subject)
256     train_mean = find_mean(train_images)
257     train_mean_subtracted, test_mean_subtracted = subtract_mean(train_mean, train_images, test_images)
258     unitary_vectors = calculate_svd(train_mean_subtracted)
259     alpha_svd_train, alpha_svd_test = calculate_alpha(unitary_vectors, train_mean_subtracted, \
260                                                       test_mean_subtracted)
261     calculate_and_plot_prediction_rates(train_images_per_subject, test_images_per_subject, \
262                                       alpha_svd_train, alpha_svd_test, ks, dataset="Yale", method="svd")
263     calculate_and_plot_prediction_rates(train_images_per_subject, test_images_per_subject, \
264                                       alpha_svd_train, alpha_svd_test, ks, dataset="Yale", method="svd", light=True)
265
266
267 if __name__ == "__main__":
268     ORL_PATH = ".././ORL"
269     ORL_k = [1, 2, 3, 5, 10, 15, 20, 30, 50, 75, 100, 150, 170]
270     ORL_data(ORL_PATH, subjects=32, train_images_per_subject=6, test_images_per_subject=4, ks=ORL_k)
271     YALE_PATH = ".././CroppedYale"
272     YALE_k = [1, 2, 3, 5, 10, 15, 20, 30, 50, 60, 65, 75, 100, 200, 300, 500, 1000]
273     YALE_data(YALE_PATH, train_images_per_subject=40, test_images_per_subject=24, ks=YALE_k)

```

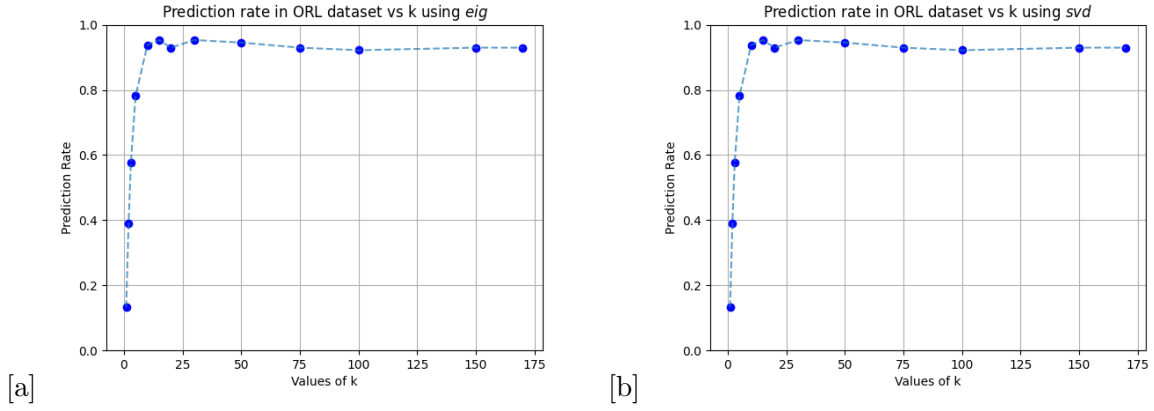


Figure 1: (a) Prediction Rate vs k for ORL Database using *eig* (b) Prediction Rate vs k for ORL Database using *svd*

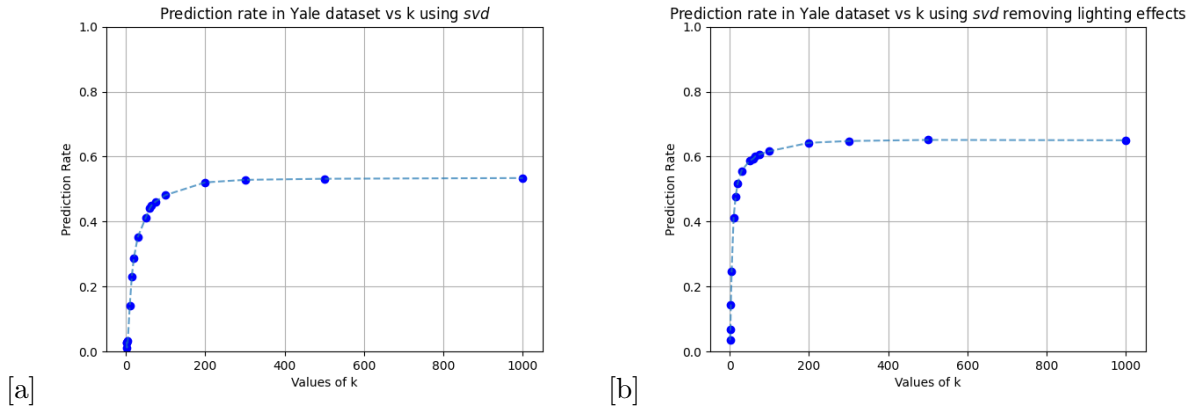


Figure 2: (a) Prediction Rate vs k for CroppedYale Dataset using *svd* (b) Prediction Rate vs k for CroppedYale Database using *svd* without the first three eigen - vectors

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].