Gupta, Ajitesh
Choi, Jean
Mavalankar, Aditi Ashutosh
Department of Electrical Engineering and Computer Science
University of California, San Diego

December 6, 2016

HOMEWORK 4

Problem 1. **Nearest Neighbor refresher** Consider that you have a set of 1-Dimensional points with their labels. The tuples of points and labels are : (1, 0), (1.5, 0), (4, 0), (2, 1), (3, 1), (0, 1).

- Estimate the labels of the following points. You have to use the 1-Nearest neighbor algorithm to compute them.

  Solution
  Denote NN as the nearest neighbor.
  (1) 1, the NN of -2 is 0 which has label 1.
  (2) 0, the NN of 5 is 4 which has label 0.
  (3) 0, the NN of 1.6 is 1.5 which has label 0.
  (4) 0, the NN of 0.75 is 1 which has label 0.

- Now that you have got a hook of the algorithm, can you say where the decision boundaries lie ?

  Solution
  First decision boundary lies at $(0+1)/2 = 0.5$. The second decision boundary lies at $(1.5+2)/2 = 1.75$. The third decision boundary lies at $(3+4)/2 = 3.5$.

Problem 2. **PCA** You have been taught about Eigen Decomposition as well as Singular Value Decomposition. Now lets try to find an interesting relationship between them. Consider a set of zero-centered data points X. X has a dimension of $m \times n$. It means that there are m data samples. Each sample is a point in the n-dimensional space.

- Suppose $\sum$ is the covariance matrix of X. What is the relationship between the eigen values of $\sum$ and the singular values of X?

  Solution
  Using the lecture notes, where an $m \times n$ data matrix is of m data pixels and n images, by definition,

  $$\sum_{cov} = \frac{1}{n-1}\sum^{n}(x_i - \mu)(x_i - \mu)^T \tag{0-1}$$

  whereas μ is the average of x_i's.

  $$\mu = \frac{1}{n}\sum^{n} x_i \tag{0-2}$$

  Denote data matrix as X. Since X is a zero-centered:

  $$\sum_{cov} = \frac{1}{n-1}\sum^{n}(x_i - \mu)(x_i - \mu)^T = \frac{1}{n-1}XX^T \tag{0-3}$$

Now, in SVD, singular values of X are the square roots of eigenvalues of XX^T and X^TX. Thus singular values of X are the square roots of eigenvalues of $(n-1)\sum_{cov}$.

- Now suppose you approximate a vector X_i using the top k eigen vectors obtained from PCA.(Assume that $k \leq n$) , where n is the total number of eigen vectors. What fraction of variance is represented by such an approximation ?

  Solution
  Out of n eigenvectors, top k were chosen. Variance, which is an eigenvalue, is σ^2.
  Thus the fraction represented is:

$$\frac{\sigma_1^2 + \sigma_2^2 + ... + \sigma_k^2}{\sigma_1^2 + \sigma_2^2 + ... + \sigma_n^2} \tag{0-4}$$

Problem 3. **Naive Recognition** : This problem involves doing face recognition by comparing raw pixel values of the images. Here, subset0 is the training set and subsets 1 to 4 are the testing sets. We are supposed to use 1-NN (1 Nearest Neighbor) classification.

1. After classifying all the images in the test data, report the average test error (percentage of misclassified images) for each test set. Report average accuracy for each test set.

   Solution Average test error:

   Subset 1 : 5.83%
   Subset 2 : 48.33%
   Subset 3 : 78.57%
   Subset 4 : 86.84%

   Average accuracy:

   Subset 1 : 94.17%
   Subset 2 : 51.67%
   Subset 3 : 21.43%
   Subset 4 : 13.16%

2. Comment on the performance, and if there is any difference between the sets.

   Solution

   We can see that subset 1 has a very good performance, whereas subset 4 has the worst performance. This is because as we observe the testing data, we see that each subset has different lighting conditions for the faces. Subset 0 has almost uniformly distributed lighting, as a result of which comparison of raw pixel values will lead to higher accuracies. However, with each increasing subset, we see that the light moves farther from the source, and its angle changes, due to which not all features can be observed, when we compare just the pixel values, as the light is at a more skewed angle, which affects the illumination on the faces. Thus, subset 2 has a worse performance than subset 1, and similarly, subset 3 has a worse performance than subset 2, and subset 4 has the worst performance.

3. Show any two correctly classified samples. Why does the model work well for those samples?

   Solution

Figure 1: Correctly classified samples

This model works well for these samples as we can see that the lighting is reasonably uniform for both these samples. Due to this, comparison of just the pixel values with the training data values will result in a match.

4. Show any two incorrectly classified samples. Why were they misclassified?
   Solution

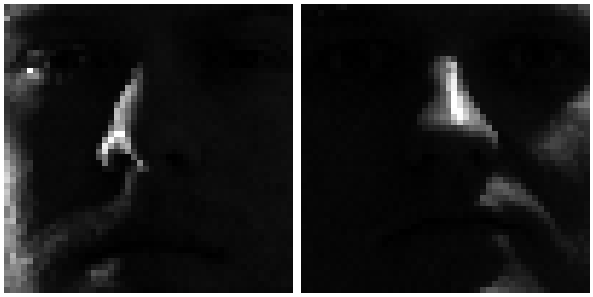

Figure 2: Incorrectly classified samples

These samples were misclassified as we can see that since these pictures have been taken in poor lighting conditions, the pixel values will not match those of the training dataset as there are very few significant pixel values for comparison.

5. Comment on the performance using l_p norm where $p = 1, 3$ and compare them.
   Solution
   The accuracies mentioned using $p = 1$ have been mentioned in part 1 of this question. The performance for $p = 3$ is:
   Testing error:
   Subset 1 : 6.67%
   Subset 2 : 50.83%
   Subset 3 : 80.71%
   Subset 4 : 87.37%
   We see that in this case $p = 1$ performs better than $p = 3$.

Problem 4. **k-NN Recognition** : Instead of using just the nearest neighbor, it is sometimes useful to consider the majority vote of k nearest neighbors. Repeat part 1 using k-NN (k Nearest Neighbors).

1. Test the performance for each dataset using $k = 1$, $k = 3$ and $k = 5$.

   Solution

   For $k = 1$, the performance is:

   Testing error:

   Subset 1 : 5.83%

   Subset 2 : 48.33%

   Subset 3 : 78.57%

   Subset 4 : 86.84%

   For $k = 3$, the performance is:

   Testing error:

   Subset 1 : 9.17%

   Subset 2 : 51.67%

   Subset 3 : 77.14%

   Subset 4 : 86.32%

   For $k = 5$, the performance is:

   Testing error:

   Subset 1 : 6.67%

   Subset 2 : 53.33%

   Subset 3 : 82.14%

   Subset 4 : 85.26%

2. Compare this performance to the previous part i.e. 1-NN classification. Were there improvements? Explain your observations.

   Solution

   $k = 1$ is essentially 1-NN classification. If we see the error rates, we find that we have an improvement on increasing the value of k for some of the testing datasets, and a decrease in performance for some datasets. For instance, there was an increase in the performance for subsets 3 and 4 when we increased the value of k from 1 to 3, but a decrease in that for subsets 1 and 2.

   Again, we see that when we increase the value of k from 3 to 5, we can see that the performance for subset 1 increases, but it is still worse than 1-NN. However, the performance for subset 4 is better than both 1-NN and 3-NN. The performance for subsets 2 and 3 is worse than 1-NN.

   In conclusion, increasing the value of k may not have the desired increase in accuracy for all subsets, and 1-NN may perform better for some subsets, compared to $k > 1$, as we have seen in the above observations.

3. Comment on the performance using l_p norm where $p = 1, 3$ and compare them.

   Solution

   The performance using $p = 1$ has been elaborated in part 1 of this question. The following are the testing errors for $p = 3$.

   For $k = 1$, the performance is:

   Testing error:

   Subset 1 : 6.67%

   Subset 2 : 50.83%

   Subset 3 : 80.71%

Subset 4 : 87.37%

For $k = 3$, the performance is:

Testing error:

Subset 1 : 5.00%

Subset 2 : 52.50%

Subset 3 :82.86%

Subset 4 : 88.42%

For $k = 5$, the performance is:

Testing error:

Subset 1 : 5.83%

Subset 2 : 57.50%

Subset 3 : 82.14%

Subset 4 : 88.95%

The comparison between the two sets of observations for $k = 1$ has already been discussed before. For $k = 3$, we get a much better performance for subset 1, but worse performance for subsets 2, 3 and 4. For $k = 5$, we get a better performance for subset 1, but worse performances for the other subsets of testing data. Here, we see that varying the values of k and p can result in different accuracies for different testing datasets.

Problem 5. **Recognition using Eigenfaces**

- Write a function

  `[W,mu] = eigenTrain(trainset,k)`

  that takes as input a $N \times d$ matrix trainset of vectorized images from subset 0, where N = 70 is the number of training images and d = 2500 is the number of pixels in each training image. Perform PCA on the data and compute the top k = 20 eigenvectors. Return the $k \times d$ matrix of eigenvectors W, and a d-dimensional vector mu encoding the mean of the training images. You should use the matlab command svd (or svds) to find the first k eigenvectors.

  Solution
  Reference the appendix for code.

- Rearrange each of the top 20 eigenvectors you obtained in the previous step into a 2D image of size 50×50. Display these images by appending them together into a 500×100 image (a 10×2 grid of images).

  Solution
  The eigenvectors from above function, eigenTrain, were displayed using mat2gray to scale the image intensity.
  Reference the appendix for code.

Figure 3: Top 20 eigenvectors

- Explain the objective of performing PCA on the training images. What does this achieve?

Solution
We are performing PCA on the training images to extract eigenvectors from them and then map them to points in k-dimensional space. In this way, we can later use these points as references when classifying the test images.

- Select one image per person from subset 0 (e.g., the 5 images person01 01.png, per- son02 01.png, ... person10 01.png). Show what each of these images would look like when using only the top k eigenvectors to reconstruct them, for k = 1, 2, 3, 4, 5, ...10. This reconstruction procedure should project each image into a k-dimensional space, project that k-dimensional space back into a 2500 dimensional space, and finally resize that 2500 vector into a 50 50 image.

  Solution
  We chose the 6th image in 7 images per person. For each strip of images, k increases downward. We placed the original image as the 11th image in each strip for convenience. We can observe that for low k values, the faces are hard to distinguish from each other. As k is increased, the reconstructed faces look more distinct.
  Reference the appendix for the function eigenApprox().

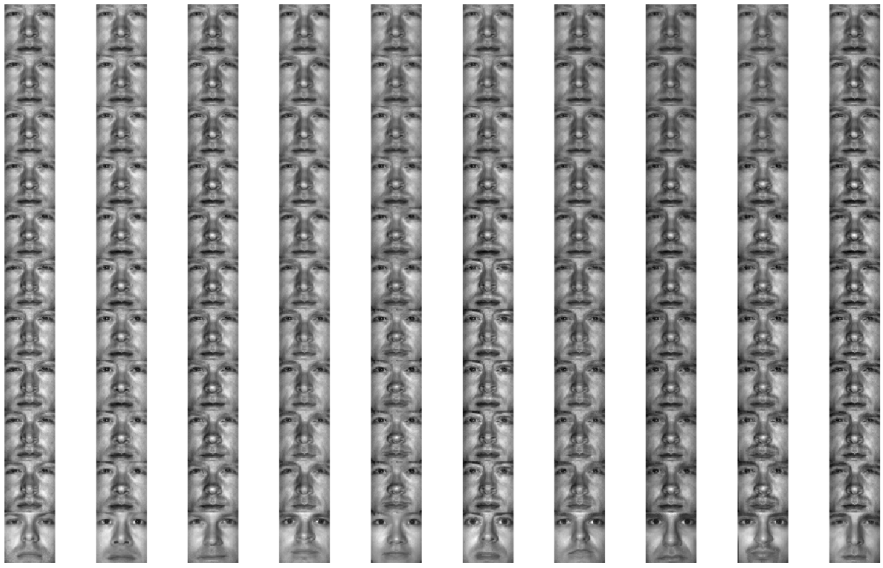

Figure 4: Reconstruction using top k eigenvectors in eigenfaces

- Write a function called

  `testlabels=eigenTest(trainset,trainlabels,testset,W,mu,k)`

  that takes as input :

1. The same N d matrix trainset of vectorized images from subset 0

2. An N dimensional vector trainlabels that encodes the class label of each training image (e.g., 1 for person01, 2 for person02, etc.)

3. An M d matrix testset of M vectorized images from one of the test subsets (1-4)

4. The output of PCA i.e. W and mu, and the number of eigenvectors to use k.

Project each image from trainset and testset onto the space spanned by the first k eigenvectors. For each test image, find the nearest neighbor (1-NN) in the training set using an L2 distance in this lower dimensional space and predict the class label as the class of the nearest training image. Your function should return an M dimensional vector testlabels encoding the predicted class label for each test example. Evaluate eigenTest on each test subset 1-4 separately for values k = 1...20 (so it should be evaluated 4×20 times). Plot the error rate (fraction of incorrect predicted class labels) of each subset as a function of k in the same plot, and use the matlab legend function to add a legend to your plot.

Solution
Reference the appendix for code and the function eigenProjection().

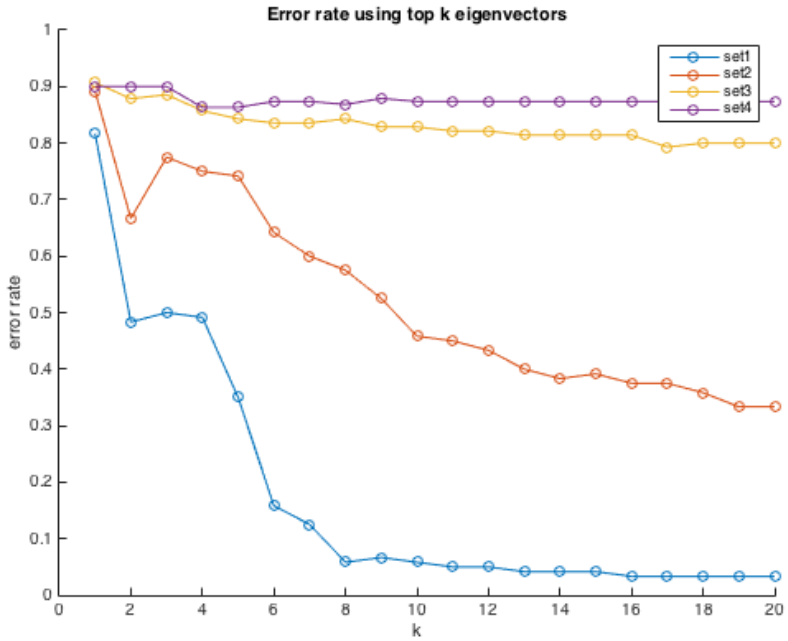

Figure 5: Error rates of sets with respect to k

- Repeat the experiment from the previous step, but throw out the first 4 eigenvectors. That is, use k eigenvectors starting with the 5th eigenvector. Produce a plot similar to

the one in the previous step. How do you explain the difference in recognition performance from the previous part?

Solution
The error rates produced by using k eigenvectors starting from the 5th one are generally less than those using top k eigenvectors. This is because the eigenface method depends on PCA. PCA tends to maximize overall scatter and does not care about within class scatter. A such it retains illumination differences within classes as they have high variance. This causes high within class scatter and as such the real class model is not perfectly formed as ideally same faces should be present together with low scatter. This causes error while classification when examples of same face with different illumination are given. Since illumination is one of the most varying features in the images, it is invariably present in the top-3-4 eigenvectors. So removing them essentially makes our algorithm invariant to the illumination changes. As such throwing out the top 4 eigenvectors actually improved the performance. But caution should be taken while doing this as the top 3-4 eigenvectors also contain other important information which we lose.
Reference the appendix for code and the function eigenProjection().

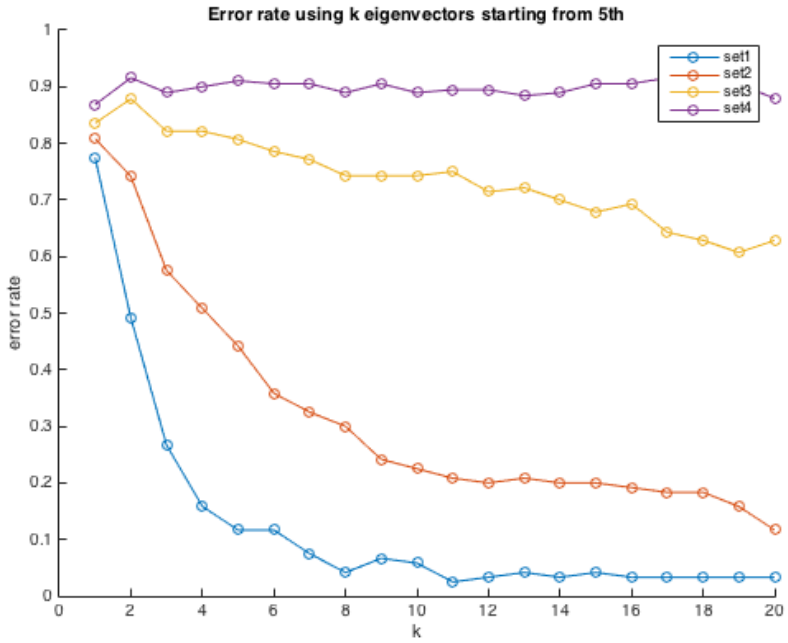

Figure 6: Error rates of sets with respect to k

- Explain any trends you observe in the variation of error rates as you move from subsets 1 to 4 and as you increase the number of eigenvectors. Use images from each subset to reinforce your claims.

Solution
Generally, the error rate increases as test subsets go from 1 through 4. This is expected

as the the faces in a subset are more illuminated and lit from less skewed direction than those in the subsequent subsets. As mentioned above PCA is and subsequently eigenfaces is not robust against illumination variation and as such tends to perform very poor as variation increases and images get darker. Refer to figure below. The partially-lit or dimly lit parts of faces make it hard to classify, thus the error rates are higher in general for subsets with larger index.

Figure 7: Test images from subsets 1-4

Also for each subset, as k is increased, the error rate decreases. This is because more k eigenvectors imply that the faces are being projected onto a space with more dimensions, so less information, which is helpful in classification, is lost during the projection.

Problem 6. **Recognition using Fisherfaces**

- Write a function called $[W, mu] = fisherTrain(trainset, trainlabels, c)$ that takes as input the same $N \times d$ matrix trainset of vectorized images from trainset of vectorized images from subset 0, the corresponding class labels trainlabels, and the number of classes c = 10. Your function should do the following :

  (a) Compute the mean mu of the training data, and use PCA to compute the first N c principal components. Let this be W_{PCA}.
  (b) Use W_{PCA} to project the training data into a space of dimension (Nc).
  (c) Compute the between-class scatter matrix SB and the within class scatter matrix SW on the (N c) dimensional space from the previous space.

(d) Compute W_{FLD}, by solving for the generalized eigenvectors of the (c 1) largest generalized eigenvalues for the problem $S_B w_i = \lambda_i S_w w_i$. You can use the matlab function eig to solve for the generalized eigenvalues of SB and SW.

(e) The fisher bases will be a $W = W_{FLD} W_{PCA}$, where W is $(c1) \times d$ dimensional, W_{FLD} is $(c1) \times (Nc)$ dimensional, and W_{PCA} is $(Nc) \times d$ dimensional.

Solution
Reference the appendix for the code.

- Similar to the Eigenfaces problem, rearrange the top 9 Fisher bases of the previous part into images of size 50×50 and stack them into one big 450×50 image.

Solution
The eigenvectors from above function, fisherTrain, were displayed using mat2gray to scale the image intensity.
Reference the appendix for code.

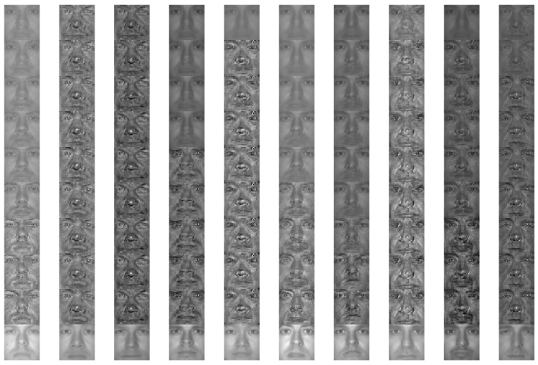

Figure 9: Reconstructed faces using fisherfaces method

- Similar to the eigenfaces problem, perform recognition on the testset with Fisherfaces. Use a nearest neighbor (1-NN) classifier and evaluate results separately for each test set (1-4) for values $k = 1 - 9$. Plot the error rate of each subset as a function of k and use a matlab legend in your plot. Explain any trends in the variation of error rates with different subsets and different values of k, and compare the performance with the Eigenfaces solution.

  Solution

  We see that the error increases as we move from set 1 to set 4. If we look at the images in the subsets, the lighting and illumination vary drastically from set 1 to set 4. The lighting in set 1 is pretty good and face is clearly visible. In fact the images are quite similar to the ones used in the training. As such the algorithm performs very well on them. But as we move to 2nd, 3rd and 4th set the images become progressively darker thus providing. Set 2 is still respectable but by the 4th set there hardly any distinguishing features visible and the images become very dark. As such the accuracy of the algorithm decreases due to lack of distinguishing features.

  As far as variation w.r.t. to k value goes, we see that as k value increases, error decreases. This is simply due to the fact that the more top-k eigenvectors we use to represent the data, the more information we are able to retain and the more distinguishing traits we are able to retain and use. The top eigenvector corresponding to the highest eigenvalue would have the most information but it alone wouldnt be able to distinguish much but as we add a few more basis the representation power increases drastically and hence also the accuracy.

  In comparison to the eigenfaces method the fisherfaces performs significantly better. This is due to the fact that eigenfaces method is based on PCA. PCA takes into account, only the total scatter of the data while reducing the dimensions and does not care about within class scatter and as such has high within class scatter. This prevents it from being illumination invariant as it retains varying illumination information rather than neglecting it. This is evident from the fact the eigenfaces method is able to reconstruct images better (as shown in the provided figure) than the fisherfaces as it retains more of the unnecessary information. This causes problems in recognition when there is significant illumination difference between faces of the same subject. On the other hand, the fisherfaces method is based on LDA which focusses on maximizing the ratio of interclass scatter vs intraclass scatter. This helps it focus on the most distinguishing features of the face rather than

the lighting information. It this thus able to recognize the same faces across illumination differences. Reference the appendix for code.

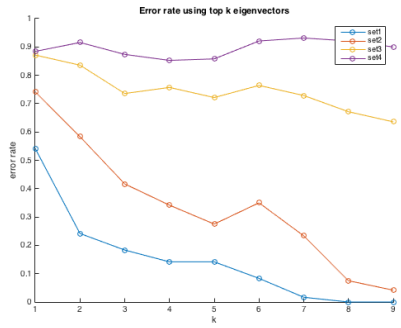

Figure 10: Error rates of sets with respect to k

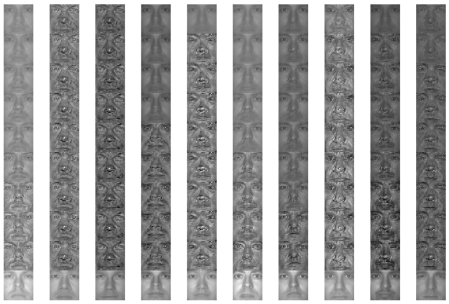

Figure 11: Reconstruction

Appendix

```
[trainset trainlabels]=loadSubset(0);

figure
imshow(drawFaces(trainset, 7));

%% (1) & (2): Test error & Difference between sets
% As the light moves away, the error rate increases
for i = 1:4
    error = classify1( trainset, trainlabels, i, 1 );
    disp(error);
end

%% (5): Norm differences
for i = 1:4
    error = classify1( trainset, trainlabels, i, 3 );
    disp(error);
```

```
end
```

Listing 1: MATLAB source for Question 3

```
function [ error ] = classify1( trainset, trainlabels, testset_num, p )

% Given a testset number:
% load the testset
% use p_norm to calculate distance
% and 1-NN to classify

% load testset
[testset testlabels]=loadSubset(testset_num);
%figure
%imshow(drawFaces(testset, 10));
[num dim] = size(testset);

% iterate
[n d] = size(trainset);

counter = 0;

for x = 1:num
    % initialization
    min = Inf;
    label = -1;

    for y = 1:n

        % using lp-norm formula to find distance by first taking absolute
        % value of the difference betweent the training and testing images.
        diff_img = testset(x, :) - trainset(y, :);
        diff_img = abs(diff_img);
        diff_img = power(diff_img, p);
        S = sum(diff_img);
        distance = power(S, 1./p);

        if distance < min
            min = distance;
            label = trainlabels(y);
        end
    end

    if label == testlabels(x)
        counter = counter + 1;
        %disp(x);disp(label);
    end
end

error = (num - counter) ./ num;

end
```

Listing 2: Classifier function for 1-NN

```
[trainset trainlabels]=loadSubset(0);
[testset testlabels]=loadSubset(1);

figure
imshow(drawFaces(trainset, 7));
figure
imshow(drawFaces(testset, 12));
```

```
8
9  %% 1-NN
10 disp('1-NN');
11 for i = 1:4
12     error = classify1( trainset, trainlabels, i, 3 );
13     disp(error);
14 end
15
16 %% 3-NN
17 disp('3-NN');
18 for i = 1:4
19     error = classifyk( trainset, trainlabels, i, 3, 3 );
20     disp(error);
21 end
22
23 %% 5-NN
24 disp('5-NN');
25 for i = 1:4
26     error = classifyk( trainset, trainlabels, i, 3, 5 );
27     disp(error);
28 end
```

Listing 3: MATLAB source for Question 4

```
1  function [ error ] = classifyk( trainset, trainlabels, testset_num, p , k)
2
3  % Given a testset number:
4  % load the testset
5  % use p_norm to calculate distance
6  % and k-NN to classify
7
8  % load testset
9  [testset testlabels]=loadSubset(testset_num);
10 [num dim] = size(testset);
11
12 % iterate
13 [n d] = size(trainset);
14
15 counter = 0;
16
17 for x = 1:num
18     % initialization
19     NN = zeros(2, n);
20     min = Inf;
21
22     for y = 1:n
23         %{
24         testset(x, :) = testset(x, : ) - mean(testset(x, :));
25         trainset(y, :) = trainset(y, : ) - mean(trainset(y, :));
26         %}
27
28         diff_img = testset(x, : ) - trainset(y, : );
29         diff_img = abs(diff_img);
30         diff_img = power(diff_img, p);
31         S = sum(diff_img);
32         distance = power(S, 1./p);
33
34         NN(1, y) = distance;
35         NN(2, y) = trainlabels(y);
36         if distance < min
37             min = distance;
38             min_label = trainlabels(y);
39         end
```

```
    end

    % find k-NN
    sorted_NN = sortrows(NN',1)';

    %disp(sorted_NN(2, :));
    % find majority
    X = sorted_NN(2, 1:k);
    label = mode(X);

    %{
    M = zeros(1, 10);
    for i = 1:k
        M(1, sorted_NN(2, i)) = M(1, sorted_NN(2, i)) + 1;
    end

    max_value = max(M);
    %label = zeros(1, 10);
    for j = 1:10
        if M(1, j) == max_value
            %label(1, j) = j;
            label = j;
        end
    end
    %}
    % break ties

    if label == testlabels(x)
        counter = counter + 1;
        %disp(x);disp(label);
    %else
        %A = [x label min_label testlabels(x)];
        %disp(A);
        %disp(sorted_NN(2, 1:5));

    end
end

error = (num - counter) ./ num;

end
```

Listing 4: Classifier function for k-NN

```
% load data
[trainset, trainlabels]=loadSubset(0);

% 5.7

figure
for i = 1:4
    [s, l] = loadSubset(i);
    subplot(1, 4, i)
    imshow(drawFaces(s(1, :), 1));

end

figure
imshow(drawFaces(trainset, 7));

%% 5.1 get eigenvectors from training set
[W, mu] = eigenTrain(trainset, 20);
```

```
19
20 %% 5.2 draw eigenfaces
21 figure
22 imshow(mat2gray(drawFaces(W, 2)));
23
24 %% 5.3 purpose of PCA and what does it achieve?
25
26 %% 5.4 approximation
27 figure
28 for i = 1:10
29     set = zeros(11, 2500);
30     subplot(1, 10, i)
31     for k = 1:10
32         x = trainset(7*i-1, :)';
33         I = eigenApprox(x, W, k);
34         set(k, :) = I + mu;
35     end
36     set(11, :) = trainset(7*i, :)' + mu;
37     imshow(mat2gray(drawFaces(set, 1)));
38 end
39
40 %% 5.5
41
42 errors = zeros(4, 20);
43 for set = 1:4
44     for k = 1:20
45         [testset, testlabels]=loadSubset(set);
46         labels = eigenTest(trainset,trainlabels,testset,W,mu,k);
47
48         % error rate
49         [M d] = size(testlabels);
50         counter = 0;
51         for i = 1:M
52             if labels(i, 1) ~= testlabels(i, 1)
53                 counter = counter + 1;
54             end
55         end
56         error = counter ./ M;
57         errors(set, k) = error;
58     end
59 end
60
61 % plot
62 figure
63 x = 1:1:20
64 for set = 1:4
65     hold on
66     y = errors(set, x);
67     plot(x, y, '-o')
68     hold off
69 end
70 title('Error rate using top k eigenvectors')
71 xlabel('k')
72 ylabel('error rate')
73 legend('set1', 'set2', 'set3', 'set4');
74
75 %% 5.6 variation in eigenvectors
76
77 [W, mu] = eigenTrain(trainset, 24);
78 Wnew = zeros(20, 2500);
79 Wnew(1:20, :) = W(5:24, :);
80
```

```
81
82 errors = zeros(4, 20);
83 for set = 1:4
84     for k = 1:20
85         [testset, testlabels]=loadSubset(set);
86         labels = eigenTest(trainset,trainlabels,testset,Wnew,mu,k);
87
88         % error rate
89         [M d] = size(testlabels);
90         counter = 0;
91         for i = 1:M
92             if labels(i, 1) ~= testlabels(i, 1)
93                 counter = counter + 1;
94             end
95         end
96         error = counter ./ M;
97         errors(set, k) = error;
98     end
99 end
100
101 % plot
102 figure
103 x = 1:1:20
104 for set = 1:4
105     hold on
106     y = errors(set, x);
107     plot(x, y, '-o')
108     hold off
109 end
110 title('Error rate using k eigenvectors starting from 5th')
111 xlabel('k')
112 ylabel('error rate')
113 legend('set1', 'set2', 'set3', 'set4');
```

Listing 5: MATLAB source for Question 5

```
1 function [ I ] = eigenApprox( x, W, k )
2
3 % output image approximated by k eigenvectors from W
4 d = 2500;
5 Wproj = zeros(k, d);
6 Wproj(1:k, :) = W(1:k, :);
7
8 x_proj = Wproj * x;
9 xnew = Wproj'* x_proj;
10
11 I = xnew;
12
13 end
```

Listing 6: eigenApprox function

```
1 function [ xnew ] = eigenProjection( x, W, k )
2
3 d = 2500;
4 Wproj = zeros(k, d);
5 Wproj(1:k, :) = W(1:k, :);
6
7 x_proj = Wproj * x;
8 xnew = Wproj'* x_proj;
9
10
```

```
end
```

Listing 7: eigenProjection function

```
function testlabels = eigenTest(trainset,trainlabels,testset,W,mu,k)

[N d] = size(trainset);
[M d] = size(testset);

%figure
%imshow(drawFaces(testset, 10));
testlabels = zeros(M, 1);

for x = 1:M
    min = Inf;
    label = -1;
    test = eigenProjection(testset(x, :)', W, k);

    for y = 1:N
        train = eigenProjection(trainset(y, :)', W, k);
        diff = test(:, 1) - train(:, 1);
        diff = abs(diff);
        S = sum(diff);
        distance = power(S, 1./2);

        if distance < min
            min = distance;
            label = trainlabels(y);
        %elseif distance == min
        %    break
        end
    end

    testlabels(x, 1) = label;

end



end
```

Listing 8: eigenTest function

```
function [W,mu] = eigenTrain(trainset,k)

%% find mu first
n = 70;
d = 2500;
sum = zeros(d, 1);

for i = 1:n
    %trainset(i, :) = trainset(i, : ) - mean(trainset(i, :));
    sum = sum + trainset(i, :)';
end

mu = sum./ n;

%% create data matrix D
D = zeros(d, n);

for i = 1:n
    D(:, i) = trainset(i, :)' - mu;
```

```
end

%% perform SVD to create covariance mtx
[U, S, V] = svd(D);

Covar = (n - 1) * U;

% normalization
for i = 1:n
    Covar(:, i) = Covar(:, i)/norm(Covar(:, i), 2);
end

%% create W of k eigenvectors
W = zeros(k, d);

for i = 1:k
    W(i, :) = Covar(:, i)';
end

end
```

Listing 9: eigenTrain function

```
%%6.1
[trainset trainlabels] = loadSubset(0);
c = 10;

%% 6.1
[W, mu] = fisherTrain(trainset, trainlabels, c);

%% 6.2
figure
imshow(mat2gray(drawFaces(W, 1)));

%% 6.3
errors = zeros(4, 9);
for set = 1:4
    for k = 1:9
        [testset, testlabels]=loadSubset(set);
        labels = eigenTest(trainset,trainlabels,testset,W,mu,k);

        % error rate
        [M d] = size(testlabels);
        counter = 0;
        for i = 1:M
            if labels(i, 1) ~= testlabels(i, 1)
                counter = counter + 1;
            end
        end
        error = counter ./ M;
        errors(set, k) = error;
    end
end

% plot
figure
x = 1:1:9
for set = 1:4
    hold on
    y = errors(set, x);
    plot(x, y, '-o')
    hold off
end
```

```
title('Error rate using top k eigenvectors')
xlabel('k')
ylabel('error rate')
legend('set1', 'set2', 'set3', 'set4');

%% Approximation
figure
for i = 1:10
    set = zeros(10, 2500);
    subplot(1, 10, i)
    for k = 1:9
        x = trainset(7*i-1, :)';
        I = eigenApprox(x, W, k);
        set(k, :) = I + mu;
    end
    set(10, :) = trainset(7*i, :)' + mu;
    imshow(mat2gray(drawFaces(set, 1)));
end
```

Listing 10: MATLAB source for Question 6

```
function [W, mu] = fisherTrain(trainset, trainlabels, c)

[N, d] = size(trainset);

% (1) Find W_pca
k = N - c;
[W_pca, m] = eigenTrain(trainset, k);

% (2) Project the training data onto dim(N-c)
train_proj = W_pca * trainset';

% (3-1) Compute SB
SB = zeros(k, k);

% compute mu
mu = zeros(k, 1);
for i = 1:N
    mu = mu + train_proj(:, i);
end
mu = mu ./ N;

for j = 1:c
    % compute SB_i
    chi_i = 7;
    sum_i = zeros(k, 1);

    for i = 1:7
        sum_i = sum_i + train_proj(:, i + (j-1)*7);
    end

    mu_i = sum_i ./ chi_i;
    SB_i = chi_i * (mu_i - mu) * (mu_i - mu)';
    SB = SB + SB_i;
end

% (3-2) Compute SW
SW = zeros(k, k);

for j = 1:c
    % compute SW_i
    chi_i = 7;
    sum_i = zeros(k, 1);
```

```
43
44      for i = 1:7
45          sum_i = sum_i + train_proj(:, i + (j-1)*7);
46      end
47
48      mu_i = sum_i ./ chi_i;
49
50      for i = 1:7
51          x_k = train_proj(:, i + (j-1)*7)
52          SW_i = (x_k - mu_i) * (x_k - mu_i)';
53          SW = SW + SW_i;
54      end
55  end
56
57  % (4) Compute W_FLD
58  %[V,D] = eig(inv(SW) * SB);
59
60  [W_FLD, D] = eig(SB, SW);
61
62  % sort
63  [D, i] = sort(diag(D), 'descend');
64  W_FLD = W_FLD(:,i);
65
66  % truncate
67  W_FLD = W_FLD(:,1:c-1);
68
69  % transpose
70  W_FLD = W_FLD';
71
72  % (5) Find W
73  W = W_FLD*W_pca;
74
75  mu = m;
76
77  end
```

Listing 11: fisherTrain function

```
1  function [ img ] = drawFaces(imgs, cols)
2      if nargin < 2, cols = size(imgs,1); end
3      w = 50;
4      h = 50;
5      n = 1;
6
7      rows = ceil(size(imgs,1)/cols);
8      img = zeros(h*rows, w*cols);
9      for i=1:rows
10         for j=1:cols
11             if n <= size(imgs,1)
12                 img(((i-1)*h+1):((i)*h), ((j-1)*w+1):((j)*w)) = reshape(imgs(n,:),h,w
       );
13                 n = n + 1;
14             end
15         end
16     end
17 end
```

Listing 12: given function drawFaces

```
1  function [ imgs, labels ] = loadSubset(subsets, yaleDir)
2  % loadSubset Load a subset or set of subsets from the yale face database,
3  % returning a matrix of images
```

```
%
% INPUT:
%   subsets:  the index of the subset to load, or a vector of sebset
%      indices.  For example, loadSubset(2) loads subset2, whereas
%      loadSubset([0,1]) loads subset0 and subset1
%   yaleDir:  the path to the yale dataset directory.  If left blank,
%      defaults to 'yaleBfaces'
%
% OUTPUT:
%   imgs: a NXd matrix of images, where N is the number of images and d is
%      the number of pixels in each image
%   labels: a vector of length N, storing the person or class ID of each
%      image

    if nargin < 2,  yaleDir = 'yaleBfaces'; end
    imgs = [];
    labels = [];
    for i=1:length(subsets)
        subsetDir = sprintf('%s/subset%d/', yaleDir, subsets(i));
        files = dir(subsetDir);
        for j=1:length(files)
            [person count] = sscanf(files(j).name, 'person%02d');
            if count == 1
                img = im2double(imread([subsetDir files(j).name]));
                imgs = [imgs; img(:)'];
                labels = [labels; person];
            end
        end
    end
end
```

Listing 13: given function loadSubset

Submitted by Gupta, Ajitesh
Choi, Jean
Mavalankar, Aditi Ashutosh on December 6, 2016.