CSE 252A Homework 4

Dominique Meyer A99079487, Hao-en Sung A53204772, Tsung-han Lee A53219632

January 12, 2018

1 Nearest Neighbour refresher [4 pts]

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 neighbour algorithm to compute them. [2pts]

  (1) -2

  (2) 5

  (3) 1.6

  (4) 0.75

- Now that you have got a hook of the algorithm, can you say where the decision boundaries lie ? [2pts]

Sol. Part 1:

Using the given points and their respective labels, it was derived what the closest point to the sample points were.

(1) -2 : label 1
(2) 5 : label 0
(3) 1.6 : label 0
(4) 0.75 : label 0

Part 2:

It can therefore be concluded that the decision boundaries lie at the following places: 0.5, 1.75, 3.5. Any sample point less than 0.5 would be classified as label 1. Between 0.5 and 1.75, the label would be 0. Between 1.75 and 3.5, the label would be 1. Anything greater than 3.5 would be label 0. □

2 PCA [6 pts]

You have been taught about Eigen Decomposition as well as Singular Value Decomposition. Now let's 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 Σ is the covariance matrix of X. What is the relationship between the singular values of Σ and the eigen values of X. [4pts]
- 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 ? [2pts]

Sol. Part 1:

Given the matrix $X(mxn)$ which has m data samples in a n-dimensional space, we can derive the variance matrix of X, $\sum$, by having the variances on the diagonal of the matrix and the covariances between the different components in the other matrix elements:

$$Var(X) = E[(X - Ex)(X - EX)^T] = \begin{bmatrix} Var(X_1) & \cdots & Cov(X_1, X_n) \\ \vdots & \ddots & \vdots \\ Cov(X_n, X_1) & \cdots & Var(X_n) \end{bmatrix}$$

The n x n covariance matrix of X can then be calculated using the population variables:

$$\sum = (m-1)^{-1} \begin{bmatrix} \sum_{i=1}^m (X_{i1} - \bar{X}_1)^2 & \cdots & \sum_{i=1}^m (X_{i1} - \bar{X}_1)(X_{in} - \bar{X}_n) \\ \vdots & \ddots & \vdots \\ \sum_{i=1}^m (X_{in} - \bar{X}_n)(X_{i1} - \bar{X}_1) & \cdots & \sum_{i=1}^m (X_{in} - \bar{X}_n)^2 \end{bmatrix}$$

Which can also be written as:

$$\sum = X^T X/(m-1)$$

As the covariance matrix is symmetric, it can be diagonalized.

$$\sum = VLV^T$$

Where V is the matrix of eigenvectors and L is the diagonal matrix with eigenvalues. When projecting the data onto the principal axes, we get the principal components. After performing singular value decomposition, we obtain the decomposition:

$$X = USV^T$$

Where S is the diagonal matrix of singular values s.

$$\sum = VSU^T USV^T/(m-1) = V((S^2)/(m-1)V^T)$$

As such, the singular values, s_i of the data matrix X are related to the eigenvalues $\lambda_i$of the covariance matrix by:

$$\lambda_i = s_1^2/(m-1)$$

Part 2:

When using the first k of n eigenvectors from PCA, the fraction of variance is represented by the following equation:

$$V_k = (\sum_{i=1}^{k} \lambda_i)/(\sum_{i=1}^{n} \lambda_i)$$

□

3 Naive Recognition [9 pts]

For this first problem, you will do face recognition by comparing the raw pixel values of the images. Let subset 0 be the train set, and report classification accuracy on test sets 1 to 4. Use the nearest neighbor classifier (1-NN) (Reference notes : Link).

1. (3 points) Once you have classified all images in a test set, report the average test error (percentage of misclassified test images). Report the average accuracy for each test set, i.e. 4 numbers
2. (2 points) Comment on the performance, does it make sense? Is there any difference between the sets?
3. (1 point) Show any two correctly classified examples and explain why the model works well for these?
4. (1 point) Show any two misclassified examples and explain why these might have been misclassified.
5. (2 points) Also comment on the performance using l_p norm where $p = 1, 3$ and compare them.

Sol. Part 1:

The accuracy are shown as below with 6-digit precision:

Test set 1: 0.941667
Test set 2: 0.516667
Test set 3: 0.192857
Test set 4: 0.152632

Part2:

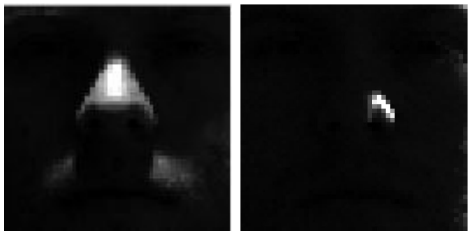
The performance makes sense because the knn classifier relates the test image pixel values to the sample images. When the images are more obscured, it becomes harder to identify which is the closest sample image. Set 1 has front illuminated faces while set 2 has partially side illuminated and set 3 and 4 have very obscure side illumination. It therefore makes sense as to why the performance is the worst in set 3 and 4 and pretty decent in set 1.

Part 3:

The classification of images 2 and 27 from set 1 (as seen below) were successful because both images showed the face clearly, with front on lighting illumination, light enough for the pixel values to be varying rather than all black, as is the case in later datasets.

Part 4:

The classification of images 1 and 18 from set 4 (as seen below) were unsuccessful because they provided insufficient pixel values which could be correlated to the



Part 5:

The performance when p=1,3 in l_p is very slightly affected. Test set 1 performance did not change at all, while set 2 decreased slightly when p=3 compared to when p=1 or p=2. Set 3 showed a slight increase of 2 percent when p=1 compared to p=2, but p=3 was the same as p=2. Set 4 had performance decrease of both p=1 and p=3 compared to p=2, but again only a few percent. Overall, we can see that the change in p has negligible effect in the performance of face classification, given these datasets.

□

4 k-NN Recognition [7 pts]

1. (3 points) Test the performance for each test dataset using k = 1, k = 3 and k = 5.

Sol. I implement the KNN algorithm, which is recorded as Code 3 in Appendix. I then generate the experiment result for $k \in \{1, 3, 5\}$ and $p = 2$, as shown in Table 1. The script for problem 4 can be found as Code 4. □

2. (2 points) Compare the performance to the corresponding 1-NN performance, did it improve? Explain your observation.

Sol. Here is an implementation detail: how to deal with majority vote when there is a tie. For my implementation, I just use the default majority function *mode* provided by *MATLAB*. That may be the reason why $k = 3$ or $k = 5$ will sometimes degrade the model performance. From the experiment results, there is no clear improvement trend of using $k = 3$ or $k = 5$. □

3. Also comment on the performance using lp norm where p = 1, 3 and compare them.

Sol. I then consider $k \in \{1, 3, 5\}$ and $p \in \{1, 3\}$, as shown in Table 1. It is noticeable that the experiment performance strongly depends on the dataset itself, and there is no clear improvement tendency when considering L_1 and L_3 norm. □

	Test Data 1	Test Data 2	Test Data 3	Test Data 4
(k=1, p=2)	0.941667	0.516667	0.192857	0.152632
(k=3, p=2)	0.941667	0.475000	0.178517	0.121053
(k=5, p=2)	0.950000	0.466667	0.171429	0.110526
(k=1, p=1)	0.941667	0.516667	0.214286	0.131579
(k=3, p=1)	0.908333	0.483333	0.228571	0.136842
(k=5, p=1)	0.933333	0.466667	0.171429	0.14.7368
(k=1, p=3)	0.933333	0.491667	0.192857	0.126316
(k=3, p=3)	0.950000	0.475000	0.171429	0.115789
(k=5, p=3)	0.941667	0.425000	0.178571	0.110526

Table 1: Test Data Performance K-NN Accuracy

5 Recognition using Eigenfaces [18 Pts]

1.(4 points) 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.

Sol. Follow the instruction and sudo code in the lecture, I first find the mean image (in the vector form) of the training set. Later, I subtract the mean image for each image in the training set. By performing SVD on it, I will get the Eigenvectors of the training set, and select the top k as the output. My *eigenTrain* implementation is recorded as Code 5. □

2. (2 points) 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).

Sol. Using the output from the previous step, I rearrange the original $k \times d$ matrix into a 500×100 matrix to display the images (a 10×2 grid of images), where k is the top k eigenvectors, d is the dimension of the vectorized image. The display top k eigenfaces is described as the following sequence.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ . & . \\ . & . \\ 17 & 18 \\ 19 & 20 \end{bmatrix}$$

Since some values in the eigenvectors may be negative, I normalize the eigenvectors by shifting the value and scaling all the vectors in the same amount to avoid the color beyond black, i.e. value<0. The result are shown in Fig. 1. The implementation is recorded as Code 6 and 7.

□

3. (2 points) Explain the objective of performing PCA on the training images. What does this achieve?

Sol. The purpose of PCA is to transform the set of images with possible correlation into a set of values of linearly uncorrelated variables or vectors. In this case, the eigenvectors can also be called eigenfaces, which describe the features of the face in independent way. PCA can help to find the most important features, i.e. eigenfaces, among the training set. It also can help to reduce the feature space by selecting top k features, and thus reduce the computation cost in the following question when classifying the faces, i.e. face recognition. In addition, by using the PCA, we can reconstruct the image. See the next question. □

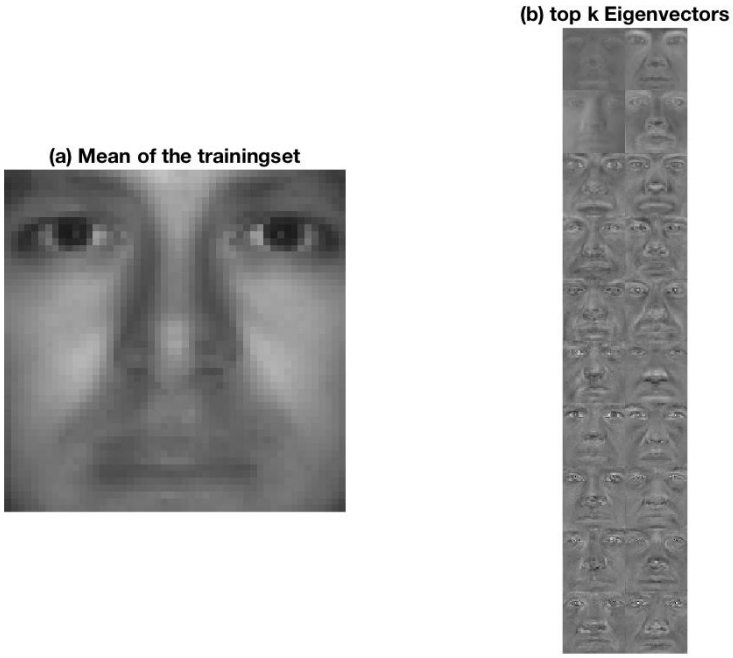


Figure 1: Mean face and Top 20 Eigenfaces

4. (2 points) 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.

Sol. For each selected image from the training set for each person, I subtract the mean image from them. Then, using dot product of the matrix to project them into top k-dimensional eigenspace to find the weighting for each eigenvectors. Finally, multiply the weighting by eigenvectors set to project the k-dimension space back into a 2500 dimensional space. Then reshape it to a 50×50 image. The result are shown in Fig. 2 and Fig. 3. The implementation is recorded as Code 8.

□

5. (4 points) Write a function called testlabels=eigenTest(trainset,trainlabels,testset,W,mu,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.

Sol. Follow the instruction and in the lecture, I first subtract the mean image for each image in the training set and testing set. Using dot product of the matrix to project them into top k-dimensional eigenspace. Then use the kNN classification code in the Problem 4 to perform 1-NN classification. Finally, output the result of the 1-NN classification.
For each testing set (i.e. subset1, subset2, subset3, subset4), I compute the error rate of the

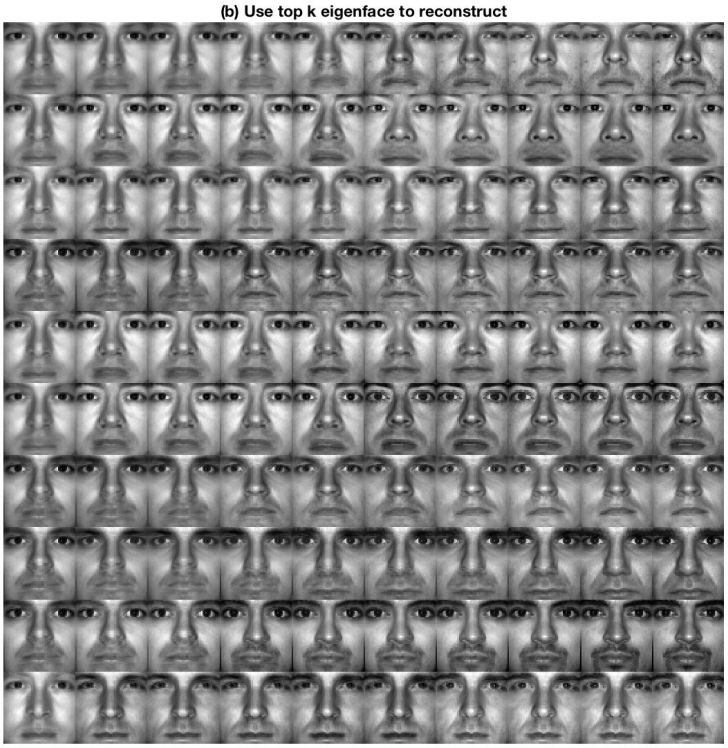


Figure 2: Reconstruction

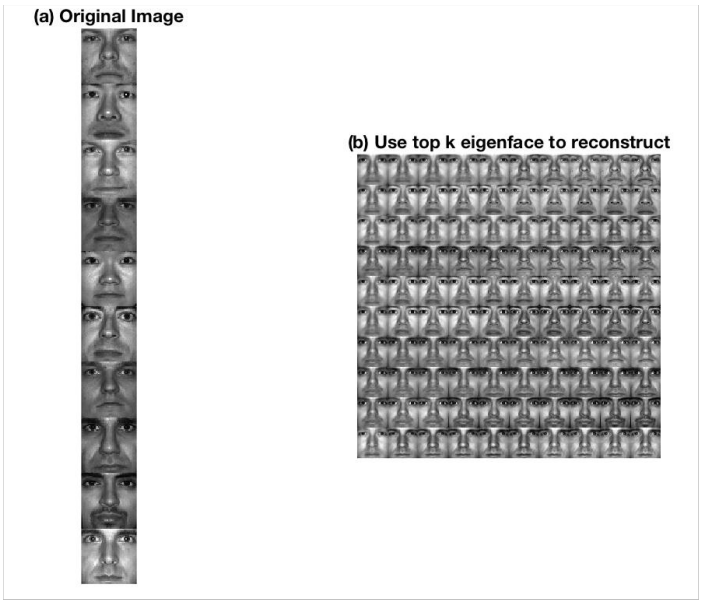


Figure 3: Reconstruction

classification result for each k, where $k = 1, 2, 3, ..., 20$. The result are shown in Fig.4. My *eigenTest* implementation is recorded as Code 9.

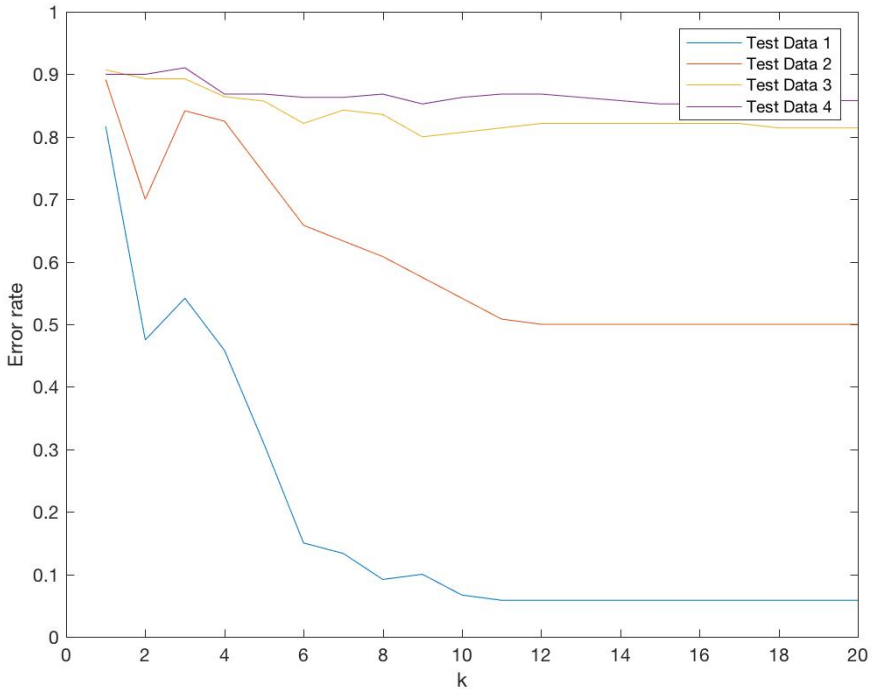


Figure 4: Error rate curve

□

6. (2 points) 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?

Sol. As the instruction, I throw away the first four eigenvectors, which means to start from the fifth eigenvectors. Then use the same procedure in the previous question to produce the error rate versus different k for all the four testing set. The result are shown in Fig. 5. The implementation is recorded as Code 10.
From the result Fig. 4 and Fig. 5, it is noticeable that the error rate decreases for testing set 1,2 and 3. When dealing with classification problem, all the things we want is to distinguish between different class. That is, projecting the testing data into a space, in which the training data can be discrete among the different class and dense in the same class. In this problem, because the top-ordered eigenfaces share the similarity between all the faces, they may not good at discriminating different faces (the classification space is not discrete enough among distinct class). That's why after discarding the first four eigenfaces, the error rate decreases.
As for the testing set 4, since the image is dark in half of the faces (Fig.6), it is not suitble to use the features generated from the image taken in the environment that is full of light, the error rate does not decrease even after throwing out the first four eigenfaces.

□

7. (2 points) 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.

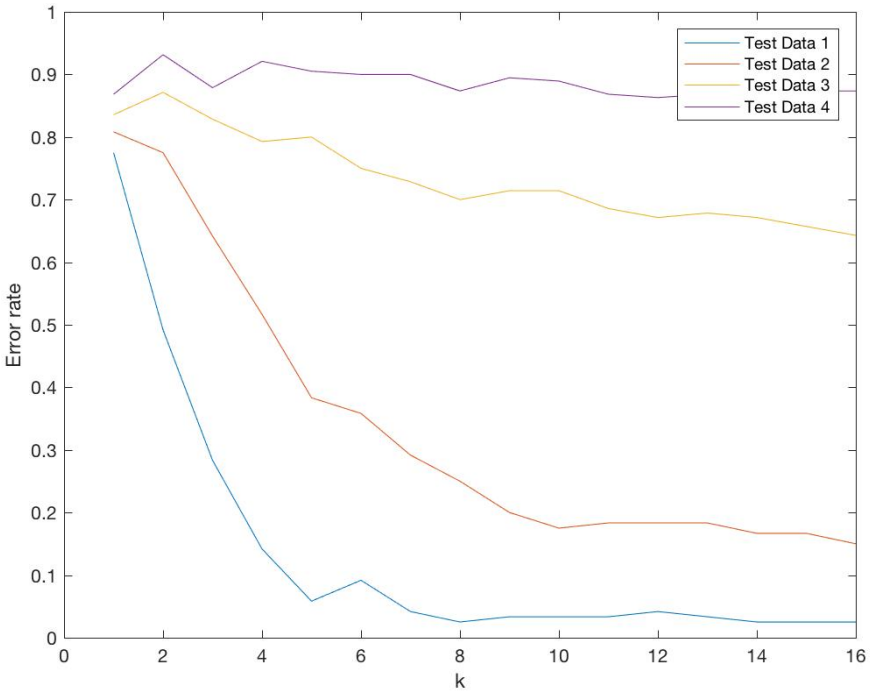


Figure 5: Error rate curve if discard the first four eigenfaces

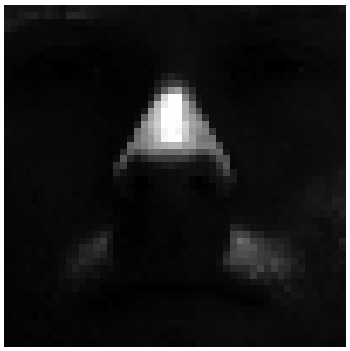


Figure 6: Testing image from subset4

Sol. Generally speaking, for testing set 1,2,and 3, increasing k can help to reduce the error rate, because involving more features, i.e. eigenfaces, is helpful for discriminating distinct classes. However, when moving from the subset 1 to 4, the error rate increases. This is because the light source is rotating from the hard front to the side of the face, as subset goes from 1 to 4. This will cause the shadow to block out the clues or features of the corresponding faces, only the nose is still full of light enough. See the following figures, Fig.7, Fig.8, Fig.9, and Fig.10.

Figure 7: Testing image from subset1

Figure 8: Testing image from subset2

Figure 9: Testing image from subset3

□

6 Recognition using Fisherfaces [14 Pts]

1. (8 points) 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 subset 0, the corresponding class labels trainlabels, and the number of classes $c = 10$.

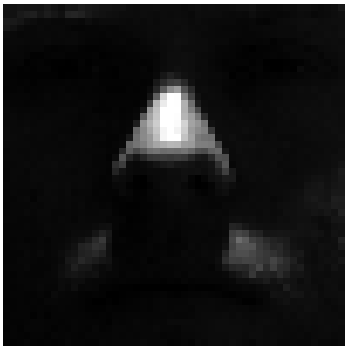


Figure 10: Testing image from subset4

Sol. Follow the pseudo code in lecture, I first calculate the PCA and map the training dataset into one subspace with $n - c$ dimension with W_{PCA}. Later, I generate W_{LDA} with S_W and S_B. At the end, I return the result $W_{PCA} \cdot W_{LDA}$. My *fisherTrain* implementation is recorded as Code 11. □

2. (2 points) As in the Eigenfaces exercise, rearrange the top 9 Fisher bases you obtained in the previous part into images of size 50×50 and stack them into one big 450×50 image.

Sol. I rearrange the output matrix of *fisherTrain* W from dimension 9×2500 into 450×50. Top 9 fisher bases are shown in Fig. 11.

□

3. (4 points) As in the eigenfaces exercise, perform recognition on the testset with Fisherfaces. As before, use a nearest neighbor classifier (1-NN), and evaluate results separately for each test subset 1-4 for values $k = 1...9$. Plot the error rate 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. Explain any trends you observe in the variation of error rates with different subsets and different values of k, and compare performance to the Eigenface method.

Sol. It is not so hard to implement *fisherTest*. I just subtract the training set and testing set by the value μ and then apply K-NN function with $k = 1$ and $p = 2$. The implementation of fisherTest can be found as Code 12; while the overall script for problem 6 can be found as Code 13 in Appendix.

It is clear that all datasets except the fourth one reduce the error rate with the growing of used components. It is quite reasonable for the first three datasets that the model improves its performance with more given information. My guess for the poor performance of fourth dataset is that there are simply too many noises for model to identify the useful information.

For the comparison between Eigenfaces and Fisherfaces, it is not surprising that Fisherfaces has better performance. The reason is that LDA further considers the relationships between different classes; while PCA may drops the needed information for classifying instances.

□

Figure 11: Top 9 Fisher Bases

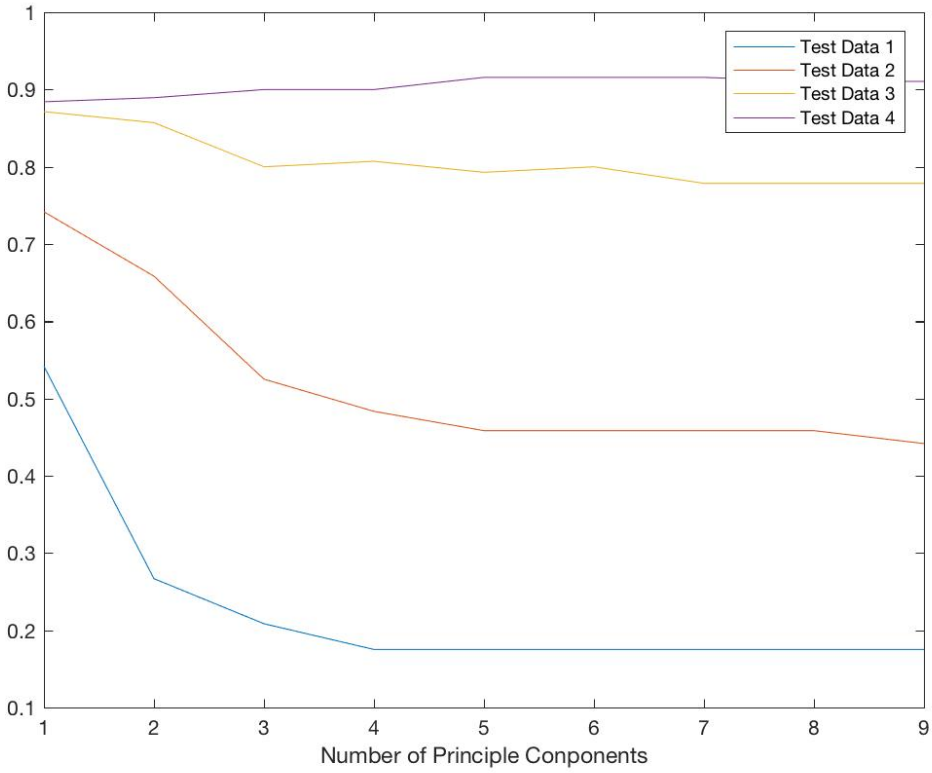


Figure 12: Fisher Algorithm Performance (Error Rate)

7 Appendix

```
%% define and initialize variables
counter=0;
k=1;
p=2;

%% loading data into variables
[trainset trainlabels]=loadSubset(0);
[set_1 set_1_labels]=loadSubset(1);
[set_2 set_2_labels]=loadSubset(2);
[set_3 set_3_labels]=loadSubset(3);
[set_4 set_4_labels]=loadSubset(4);

%% success calculation from knn classifier for set 1
class1= knn(trainset,trainlabels, set_1, k, p);
results1= horzcat(set_1_labels, class1);
for i=1:size(set_1,1)
    if results1(i,1) == results1(i,2)
        counter = counter + 1;
    end
 end
success1=100-(counter/120*100)
counter=0;

%% success calculation from knn classifier for set 2

class2= knn(trainset ,trainlabels, set_2 ,k, p);
```

```
results2= horzcat(set_2_labels, class2);
for i=1:size(set_2,1)
    if results2(i,1) == results2(i,2)
        counter = counter + 1;
    end
end
success2=100-(counter/120*100)
counter=0;

%% success calculation from knn classifier for set 3

class3= knn(trainset ,trainlabels, set_3 ,k, p);
results3= horzcat(set_3_labels, class3);
for i=1:size(set_3,1)
    if results3(i,1) == results3(i,2)
        counter = counter + 1;
    end
end
success3=100-(counter/120*100)
counter=0;

%% success calculation from knn classifier for set 4

class4= knn(trainset ,trainlabels, set_4 ,k, p);
results4= horzcat(set_4_labels, class4);
for i=1:size(set_4,1)
    if results4(i,1) == results4(i,2)
        counter = counter + 1;
    end
end
success4=100-(counter/120*100)

%% show example datasets and images

imshow(drawFaces(trainset, 10));
imshow(drawFaces(set_1, 10));
imshow(drawFaces(set_2, 10));
imshow(drawFaces(set_3, 10));
imshow(drawFaces(set_4, 10));

%correct examples
imshow(reshape(set_1(2,:),50,50))
imshow(reshape(set_1(27,:),50,50))

%incorrect examples
imshow(reshape(set_4(1,:),50,50))
imshow(reshape(set_4(18,:),50,50))
```

Listing 1: Problem 3: Running Script

```
%% knn classifier function

function tt_p = knn(tn_x, tn_y, tt_x, k, p)
%create matrices
n_tn = size(tn_x, 1);
n_tt = size(tt_x, 1);
tt_p = zeros(n_tt, 1);

%iterattively compare pixel values to each sample values
```

```
for i = 1:n_tt
    dis = zeros(n_tn, 2);
    for j = 1:n_tn
        dis(j, 1) = norm(tn_x(j, :) - tt_x(i, :), p);
        dis(j, 2) = tn_y(j);
    end
%sort according to distances
    [~, idx] = sort(dis(:, 1));
    dis = dis(idx, :);

    tt_p(i) = mode(dis(1:k, 2));
end
```

Listing 2: Problem 3: knn script

```
function tt_p = kNN(tn_x, tn_y, tt_x, k, p)

n_tn = size(tn_x, 1);
n_tt = size(tt_x, 1);
tt_p = zeros(n_tt, 1);

for i = 1:n_tt
    dis = zeros(n_tn, 2);
    for j = 1:n_tn
        dis(j, 1) = norm(tn_x(j, :) - tt_x(i, :), p);
        dis(j, 2) = tn_y(j);
    end

    [~, idx] = sort(dis(:, 1));
    dis = dis(idx, :);

    tt_p(i) = mode(dis(1:k, 2));
end
```

Listing 3: Problem 4: KNN Algorithm

```
%% load dataset
[tn_x, tn_y] = loadSubset(0, '../dat/yaleBfaces');
[tt_1_x, tt_1_y] = loadSubset(1, '../dat/yaleBfaces');
[tt_2_x, tt_2_y] = loadSubset(2, '../dat/yaleBfaces');
[tt_3_x, tt_3_y] = loadSubset(3, '../dat/yaleBfaces');
[tt_4_x, tt_4_y] = loadSubset(4, '../dat/yaleBfaces');

%% 4-1 and 4-3
kLst = [1, 3, 5];
pLst = [2, 1, 3];
for i = 1:length(kLst)
    for j = 1:length(pLst)
        tt_1_p = kNN(tn_x, tn_y, tt_1_x, kLst(i), pLst(j));
        fprintf('Test 1 with (k=%d, p=%d): %f\n', ...
            kLst(i), pLst(j), calAcc(tt_1_y, tt_1_p));
        tt_2_p = kNN(tn_x, tn_y, tt_2_x, kLst(i), pLst(j));
        fprintf('Test 2 with (k=%d, p=%d): %f\n', ...
            kLst(i), pLst(j), calAcc(tt_2_y, tt_2_p));
        tt_3_p = kNN(tn_x, tn_y, tt_3_x, kLst(i), pLst(j));
        fprintf('Test 3 with (k=%d, p=%d): %f\n', ...
            kLst(i), pLst(j), calAcc(tt_3_y, tt_3_p));
        tt_4_p = kNN(tn_x, tn_y, tt_4_x, kLst(i), pLst(j));
        fprintf('Test 4 with (k=%d, p=%d): %f\n', ...
```

```
            kLst(i), pLst(j), calAcc(tt_4_y, tt_4_p));
    end
end
```

Listing 4: Problem 4: Running Script

```
%% (a)
function [W, mu] = eigenTrain(trainset, k)
    mu = mean(trainset);

    X_hat = trainset - mu;

    [nRows, ~] = size(X_hat);
    C = X_hat' * X_hat / (nRows-1);
    [U,~,~] = svd(C);
    W = U(:,1:k)';

    %[~, ~, V] = svd(X_hat, 'econ');
    %W = V(:,1:k)';

end
```

Listing 5: Problem 5-1: eigenTrain

```
%% (b) Display Image in 500x100 matrix
function DispImg = display_b(W)
    [nRows, ~] = size(W);
    i = 1;
    while(i<=nRows)
        TmpA = reshape(W(i,:), [50,50]);
        TmpB = reshape(W(i+1,:), [50,50]);
        RowMat = horzcat(TmpA, TmpB);
        if (i==1)
            DispImg = RowMat;
        else
            DispImg = vertcat(DispImg, RowMat);
        end
        i = i+2;
    end
end
```

Listing 6: Problem 5-2: Rearrange

```
%% Normalize the vector to [0,1] scale
function mat = normal(W)
    [nRows, nCols] = size(W);
    mat = zeros(nRows, nCols);

    Max = max(W(:));
    Min = min(W(:));
    for i = 1:nRows
        for j = 1:nCols
            mat(i,j) = (W(i,j) - Min) / (Max-Min);
        end
    end

end
```

Listing 7: Problem 5-2: normal

```
%% (d)
function Img = Reconstruct(imgIn, mu, W)
    [nRows, ~] = size(W);
    Img = mu;
    y = W * (imgIn-mu)';
    for i = 1:nRows
        Img = y(i) * W(i,:) + Img;
    end
end
```

Listing 8: Problem 5-4: reconstruct

```
%% (e)
function testlabels = eigenTest(tn_x, tn_y, tt_x, W, mu, k)
    % select subset of W
    sW = W(1:k, :);

    % remap function to subspace
    tn_x = (tn_x - mu) * sW';
    tt_x = (tt_x - mu) * sW';

    % apply kNN
    testlabels = kNN(tn_x, tn_y, tt_x, k, 2);
end
```

Listing 9: Problem 5-5: eigenTest

```
% ***** Prob.5 EigenFaces *****

%% (a) Eigenfaces
k = 20;
[imgs, labels] = loadSubset(0, 'data/yaleBfaces');
[W, mu] = eigenTrain(imgs, k);

%% (b) Display in 10x2 grid of images
% Normalize W
NormW = normal(W);
DispImg = display_b(NormW);

%% ---- Plotting (a) & (b) ----
OutputFileName = '5-1.jpg';

graph = figure('visible', 'off');
%set(gcf,'units','points','position',[0,0,800,310]);

% Draw mean faces
subplot (1,2,1);
imshow(drawFaces(mu));
title('(a) Mean of the trainingset');

% Display in 10x2 grid of images
subplot(1,2,2);
imshow(DispImg);
title('(b) top k Eigenvectors');

%saveas(graph, strcat('Output/', OutputFileName));


```

```
%% (d) Reconstruction
k_prime = 10;

for i = 1:10
    index = 7*(i-1) + 1 ;

    for j = 1:k_prime
        RecImg_sub = Reconstruct(imgs(index,:), mu, W(1:j, :));
        if (j==1)
            RecImg_tmp = reshape( normal(RecImg_sub), [50,50]);
        else
        RecImg_tmp = horzcat(RecImg_tmp, reshape( normal(RecImg_sub), [50,50]));
        end
    end

    if (i==1)
            OriImg = reshape( imgs(index,:), [50,50]);
            RecImg = RecImg_tmp;
%            RecImg_sub1 = Reconstruct(imgs(index,:), mu, W(1:k_prime,:));
%            RecImg = reshape( normal(RecImg_sub1), [50,50]);
    else

        OriImg = vertcat(OriImg, reshape( imgs(index,:), [50,50]) );
        RecImg = vertcat(RecImg, RecImg_tmp);
    end
end


%% ---- Plotting (d) ----
OutputFileName = '5-2.jpg';

graph = figure('visible', 'off');
%set(gcf,'units','points','position',[0,0,800,310]);

% Draw mean faces
subplot (1,2,1);
imshow(OriImg);
title('(a) Original Image');

% Display in 10x2 grid of images
subplot(1,2,2);
imshow(RecImg);
title('(b) Use top k eigenface to reconstruct');

saveas(graph, strcat('Output/', OutputFileName));

%% (e)
[tt_1_x, tt_1_y] = loadSubset(1, 'data/yaleBfaces');
[tt_2_x, tt_2_y] = loadSubset(2, 'data/yaleBfaces');
[tt_3_x, tt_3_y] = loadSubset(3, 'data/yaleBfaces');
[tt_4_x, tt_4_y] = loadSubset(4, 'data/yaleBfaces');

tt_1_err = zeros(1, 20);
tt_2_err = zeros(1, 20);
tt_3_err = zeros(1, 20);
tt_4_err = zeros(1, 20);

for k = 1:20
```

```
    tt_1_p = eigenTest(imgs, labels, tt_1_x, W, mu, k);
    tt_1_err(k) = 1 - calAcc(tt_1_y, tt_1_p);

    tt_2_p = eigenTest(imgs, labels, tt_2_x, W, mu, k);
    tt_2_err(k) = 1 - calAcc(tt_2_y, tt_2_p);

    tt_3_p = eigenTest(imgs, labels, tt_3_x, W, mu, k);
    tt_3_err(k) = 1 - calAcc(tt_3_y, tt_3_p);

    tt_4_p = eigenTest(imgs, labels, tt_4_x, W, mu, k);
    tt_4_err(k) = 1 - calAcc(tt_4_y, tt_4_p);
end

%% ---- Plotting (e) ----
OutputFileName = '5-3.jpg';

graph = figure('visible', 'off');
plot(1:20, tt_1_err, 1:20, tt_2_err, 1:20, tt_3_err, 1:20, tt_4_err);
legend('Test Data 1', 'Test Data 2', 'Test Data 3', 'Test Data 4');
ylabel('Error rate');
xlabel('k');
saveas(graph, strcat('Output/', OutputFileName));

%% (f)
sW = W(5:20,:);
tt_1_err = zeros(1, 16);
tt_2_err = zeros(1, 16);
tt_3_err = zeros(1, 16);
tt_4_err = zeros(1, 16);

for k = 1:16
    tt_1_p = eigenTest(imgs, labels, tt_1_x, sW, mu, k);
    tt_1_err(k) = 1 - calAcc(tt_1_y, tt_1_p);

    tt_2_p = eigenTest(imgs, labels, tt_2_x, sW, mu, k);
    tt_2_err(k) = 1 - calAcc(tt_2_y, tt_2_p);

    tt_3_p = eigenTest(imgs, labels, tt_3_x, sW, mu, k);
    tt_3_err(k) = 1 - calAcc(tt_3_y, tt_3_p);

    tt_4_p = eigenTest(imgs, labels, tt_4_x, sW, mu, k);
    tt_4_err(k) = 1 - calAcc(tt_4_y, tt_4_p);
end

%% ---- Plotting (f) ----
OutputFileName = '5-4.jpg';

graph = figure('visible', 'off');
plot(1:16, tt_1_err, 1:16, tt_2_err, 1:16, tt_3_err, 1:16, tt_4_err);
legend('Test Data 1', 'Test Data 2', 'Test Data 3', 'Test Data 4');
ylabel('Error rate');
xlabel('k');
saveas(graph, strcat('Output/', OutputFileName));
```

Listing 10: Problem 5: Running Script

```
function [W, mu] = fisherTrain(tn_x, tn_y, c)

%% parameters
```

```
[n, ~] = size(tn_x);

%% PCA
[w_pca, mu] = eigenTrain(tn_x, n-c);
tn_x = (tn_x-mu) * w_pca;

%% LDA
muLst = zeros(c, n-c);
c_size = zeros(c, 1);
for i = 1:c
    muLst(i, :) = mean(tn_x(tn_y == i, :), 1);
    c_size(i) = sum(tn_y == i);
end
muAll = mean(tn_x, 1);

sb = zeros(n-c, n-c);
for i = 1:c
    bd = muLst(i, :) - muAll;
    sb = sb + c_size(i) * (bd' * bd);
end

sw = zeros(n-c, n-c);
for i = 1:n
    wd = tn_x(i, :) - muLst(tn_y(i), :);
    sw = sw + wd' * wd;
end

[V, ~] = eigs(sb, sw, c-1);
%V = fliplr(V);
w_lda = V(:, 1:(c-1));

W = w_lda' * w_pca';
```

Listing 11: Problem 6-1: FisherTrain

```
function tt_p = fisherTest(tn_x, tn_y, tt_x, W, mu, k)

% select subset of W
sW = W(1:k, :);

% remap function to subspace
tn_x = bsxfun(@minus, tn_x, mu) * sW';
tt_x = bsxfun(@minus, tt_x, mu) * sW';

% apply 1NN
tt_p = kNN(tn_x, tn_y, tt_x, 1, 2);
```

Listing 12: Problem 6-3: FisherTest

```
%% load dataset
[tn_x, tn_y] = loadSubset(0, '../dat/yaleBfaces');
[tt_1_x, tt_1_y] = loadSubset(1, '../dat/yaleBfaces');
[tt_2_x, tt_2_y] = loadSubset(2, '../dat/yaleBfaces');
[tt_3_x, tt_3_y] = loadSubset(3, '../dat/yaleBfaces');
[tt_4_x, tt_4_y] = loadSubset(4, '../dat/yaleBfaces');

%% 6-1
[W, mu] = fisherTrain(tn_x, tn_y, 10);

```

```
%% 6-2
st_W = zeros(450, 50);
for i = 1:9
    lt = 50 * (i-1) + 1;
    rt = 50 * i;
    st_W(lt:rt, :) = mat2gray(reshape(W(i, :), [50, 50]));
end
res = figure('visible', 'off');
imshow(st_W);
saveas(res, strcat('../res/fisherBases', '.jpg'));

% %% 6-3
tt_1_err = zeros(1, 9);
tt_2_err = zeros(1, 9);
tt_3_err = zeros(1, 9);
tt_4_err = zeros(1, 9);
for k = 1:9
    tt_1_p = fisherTest(tn_x, tn_y, tt_1_x, W, mu, k);
    tt_1_err(k) = 1 - calAcc(tt_1_y, tt_1_p);
    tt_2_p = fisherTest(tn_x, tn_y, tt_2_x, W, mu, k);
    tt_2_err(k) = 1 - calAcc(tt_2_y, tt_2_p);
    tt_3_p = fisherTest(tn_x, tn_y, tt_3_x, W, mu, k);
    tt_3_err(k) = 1 - calAcc(tt_3_y, tt_3_p);
    tt_4_p = fisherTest(tn_x, tn_y, tt_4_x, W, mu, k);
    tt_4_err(k) = 1 - calAcc(tt_4_y, tt_4_p);
end
res = figure('visible', 'off');
plot(1:9, tt_1_err, 1:9, tt_2_err, 1:9, tt_3_err, 1:9, tt_4_err);
legend('Test Data 1', 'Test Data 2', 'Test Data 3', 'Test Data 4');
xlabel('Number of Principle Conponents');
saveas(res, strcat('../res/fisherTest', '.jpg'));
```

Listing 13: Problem 6: Running Script