Queen Mary University of London

Department of Electrical Engineering & Computer Science

ECS797P Machine Learning for Visual Data Analysis

Lab 2 – Face Recognition Using Eigenfaces

Hasan Emre Erdemoglu

4 March 2021

Getting Started

This laboratory report will examine Face Recognition using Eigenfaces. The implementation will be done in MATLAB, following Turk and Pentland's paper. The dataset provided consists of 200 training and 70 test images of size 23 by 28. The template code given with this lab assignment is saved to a root folder "A2", this folder and its sub-directories are added to MATLAB's path.

The assignment consists of 11 parts, where some parts have already been provided within the template. The code will be explained in detail, where necessary. Only the code that I have written will be listed within this report for clarity purposes. Additional comments are added to the given code skeleton for further clarification on the implementation.

Lab 2: Face Recognition Using Eigenfaces

Part 1:

Part 1 is already implemented within the template code. It uses the following functions:

- "loadImagesInDirectory.m": The method takes the path for training images and loads, flattens and returns all 200 images in a matrix of size 200 by 644 (flattening 23 by 28 image).
- "loadTestImagesInDirectory.m": This method takes the path for test images. It loads, flattens and returns all 70 images in a matrix of size 70 by 644 (called "Imagestest" in the assignment). It also produces a one by 70 vector called "Identity", which holds the face label for each image (40 images in total).

Part 2 – 3:

Part 2 is already implemented within the template code. It calculates the training images' means (Using 200 by 644 variable, "Imagestrain"). It subtracts the mean from the images (CenteredVectors) and calculates the covariance matrix for the training images (has a size of 644 by 644). Singular Value Decomposition is applied to normalized training dataset to represent the dataset in terms of the dataset's eigenpairs. Eigenvalues are listed in the diagonal entries of 200 by 644 matrix, "S", and V contains corresponding eigenvectors for the eigenvalues.

Part 4:

Part 4 is already implemented within the template code. It reshapes one by 644 mean vector to 28 by 23 image. The following is the output:

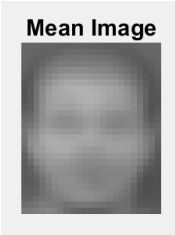

Figure 1. Mean output of Training Set.

Part 5:

This part requires displaying the first 20 eigenfaces. The first 20 eigenfaces are described by the eigenvectors of the largest 20 eigenvalues. MATLAB's Singular Value Decomposition (SVD) method automatically organizes eigenpairs in descending order of eigenvalues. This can be verified by inspecting the variable "S", which is the diagonal eigenvalue matrix.

By matrix operations, the first 20 entries of the matrix "Space" will hold the corresponding eigenvectors of the largest 20 eigenvalues. The following code was written to reshape the flattened image back into its original form.

```
figure;
for i = 1:20
    subplot(4,5,i);
    %  This notation reqd to have show images, otherwise images are B&W.
    imshow(reshape(Space(i,:),[28,23]),[]);
end
sgtitle("First 20 Eigenfaces");
```

The following output is obtained by running the code above:

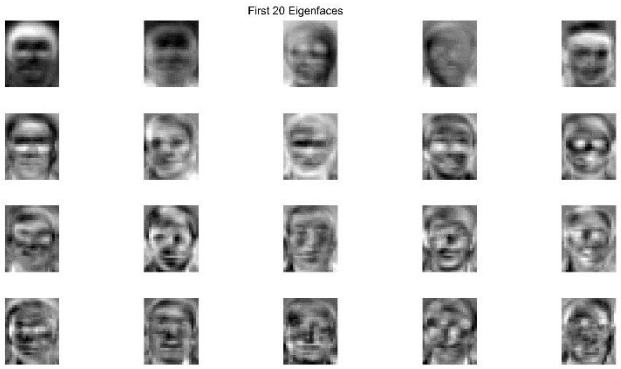

Figure 2. First 20 Eigenfaces for the Training Dataset.

Part 6:

Part 6 is already implemented within the template code. It uses a function called "projectImages.m" to project training and testing datasets, using means and eigenvectors extracted from the training dataset. The code subtracts the mean from the given dataset (train or test) and does a matrix multiplication with the training dataset's eigenvectors to project the images onto the given eigenspace.

Part 7:

Part 7 is already implemented within the template code. Two separate for-statements handle the task. The first two nested for-loop computes the Euclidean distance between all test and training samples one by one. The third, innermost loop calculates the distance between training and testing projections using the first "k" eigenfaces, set to 20 for this question.

The second loop sorts the best matching training images for the test images to be used later. It is done by sorting concerning the distances of training and testing projections.

Part 8:

Part 8 is already implemented within the template code. The output is as follows:

Figure 3. Top six best-matched training images (right) for each testing image (left).

Figure 3 shows the test image projection on the left side with 20 eigenfaces. The best matching training image with 20 eigenfaces is displayed with the projected image index on the right side. The test and training images are visually very close to each other. The labelling for the test follows from 1-nearest neighbours, as the test image is labelled by the training image label closest to the test image projection.

Part 9:

This part requires the computation of the recognition rate using 20 Eigenfaces. Note that Part 10 already partly implement this part within itself, and my implementation will follow from that implementation as well for coherence.

The recognition rate with 20 Eigenfaces is found to be 0.7583 (75.83%). See the comments for a basic explanation. The code for this section is given below, along with a further explanation of the implementation:

```
%% Part 2.9: Compute recognition rate with 20 eigenfaces:
% Part 2.10 uses this section. Code can be modified and migrated here.
recognised_person = zeros(1,40); % 40 different faces in testing images
recognitionrate=zeros(1,5);  % Each sample can be seen at most 5 times; divide to bins
number_per_number=zeros(1,5); % As identity is ordered, save counts in bins of 5.

% Loop over the testing images:
test_idx = 1;
while test_idx < size(Imagestest,1)
    id = Identity(test_idx); % Returns the label for the image, 1:40

    % Extract minimum value and the corresponding index:
    % Note that you extract only the closest index (KNN = 1)
    distmin = Values(id,1); indicemin = Indices(id,1);

    % If the test samples deplete, or the id extracted in the outer and
    % inner loop (Identity(test_idx)) do not match break from inner loop,
    % then update the predictions. Identity is sorted.
    while (test_idx < 70) && (Identity(test_idx) == id)
        % Find the minimum distance and indices (KNN = 1)
        if (Values(test_idx,1) < distmin)
            distmin=Values(test_idx,1);
            indicemin=Indices(test_idx,1);
        end
        % id updates slower than the test samples. Normal as there are 40
        % faces available but 70 samples.
        test_idx = test_idx+1;
    end

    % Log the recognized face (Best matching training image):
    recognised_person(id) = indicemin;

    % TestLabelCounts coming from Part 2.1 histogram. Tells how many
    % samples per each face we have. Same face is in test set 0 to 5 times
    % number_per_number counts how many times a id is seen (1, 2 ,3, 4, 5
    % times) in the test dataset. Will be used to normalize recognition
    % rate. The second while loop passes through the same "id"
    % testLabelCounts(id) times. Bins this and averages later.
    number_per_number(testLabelCounts(id)) = ...
        number_per_number(testLabelCounts(id))+1;

    % Does a transformation on the indexing. Checks whether the labelling is
    % correct by comparing the transformed training label to the testing label.
    % (200/40 = 5 samples per face)
    if (id==floor((indicemin-1)/5)+1) % If correct person is recognized
        recognitionrate(testLabelCounts(id))= ...
            recognitionrate(testLabelCounts(id))+1;
        % Increment the recognition, given
    end
end

% Calculates recognition rate,
for  i=1:5
    recognitionrate(i)=recognitionrate(i)/number_per_number(i);
end
mean_recognition_rate = mean(recognitionrate);
```

The outermost while loop traverses each of the test image projections. The variable "id" is used to identify the face's label for each sample. As values and indices are computed in a previous part for 20 Eigenfaces, the loop fetches the minimum distance and index for the training sample providing that minimum sample for the given ID. Note that only one minimum index and value is selected, implying this classification is done using 1-nearest neighbour. 2nd dimensionality of "Values" and "Indices" is left as 1. These variables were sorted; this means that the first entry corresponds to the training sample's distance and index closest to the corresponding test sample.

The second while loop traverses testing images with the same "id" to identify the minimum distance and the index. Using this information, for each unique face in the test set; the best training sample index can be returned using the variable "recognized_person", which has a size of one by 40.

Because of the dataset properties, each unique face is seen for 1, 2, 3, 4 or 5 times. As the inner while loop iterates over unique faces, the number of occurrences for each unique face is incremented within variable "number_per_number", which has a size of one by five. Finally, the recognition rate is calculated with an if statement, which does a transformation on the training sample's index and compares them with the testing image label. The recognition rate is also incremented if a face is detected correctly; the increment is made on bins 1, 2, 3, 4, or 5, depending on the face's frequency on the test dataset. The transformation is based on how the dataset is ordered.

Finally, the recognition rate is normalized by the "number_per_number" variable. This corresponding variable index (1 to 5) is incremented each time a unique face occurs (1-5) times. If the face is recognized correctly, the recognition rate is incremented in the same fashion. Therefore, the ratio between these two numbers normalizes the recognition rate between zero and 1. The overall recognition rate is the mean of this vector.

Part 10:

Part 10 is already implemented within the template code, and it follows from the previous discussions from Part 7, 8 and 9. The code is looped over one eigenface to 40 eigenfaces to construct the recognition rate against the number of eigenvalues. Note that for 20 eigenfaces, the results for Part 9 and Part 10 are the same. The plot is given below:

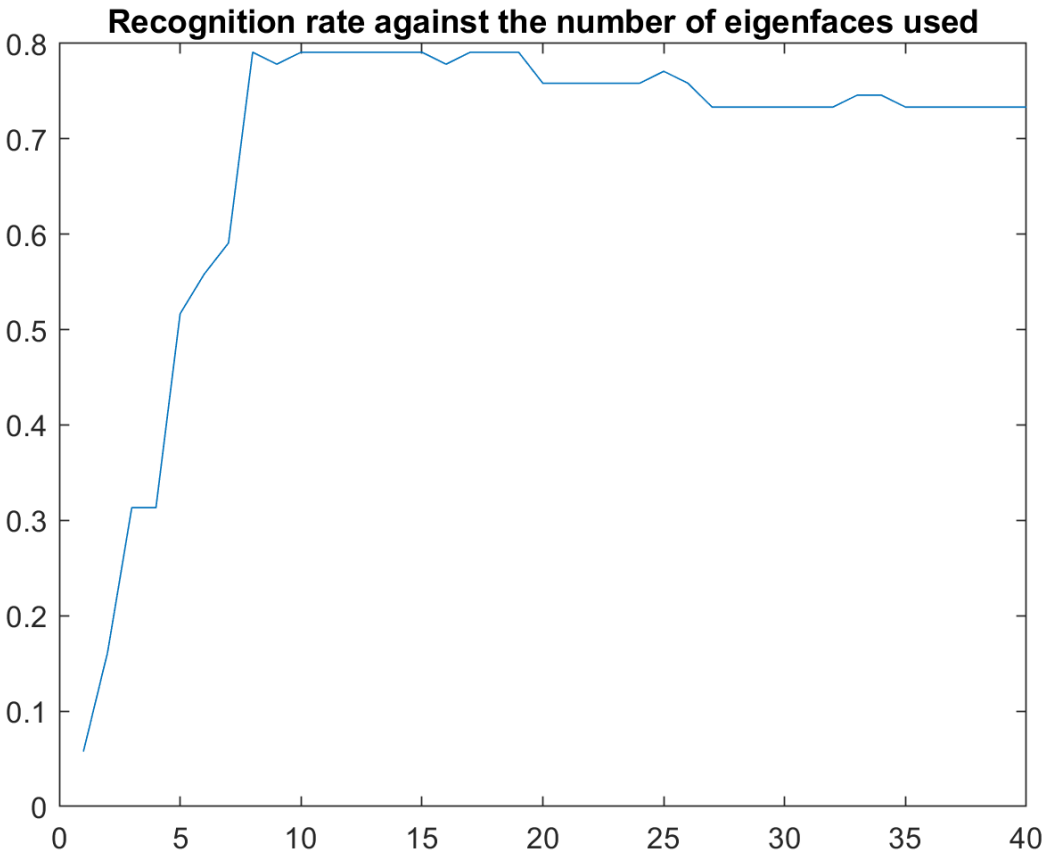

Figure 4. Recognition Rate against the number of Eigenfaces Used

Part 11:

Part 11 investigates using the KNN of k-training images on the recognition rate using 20 Eigenfaces. This part requires slight modification on Part 9, which by default assigns the label using the closest training sample only. I have done optimizations over Part 10. The following output was obtained. Note that there could be some discrepancies with the recognition rate as the implementation flow does not follow Parts 9 and 10.

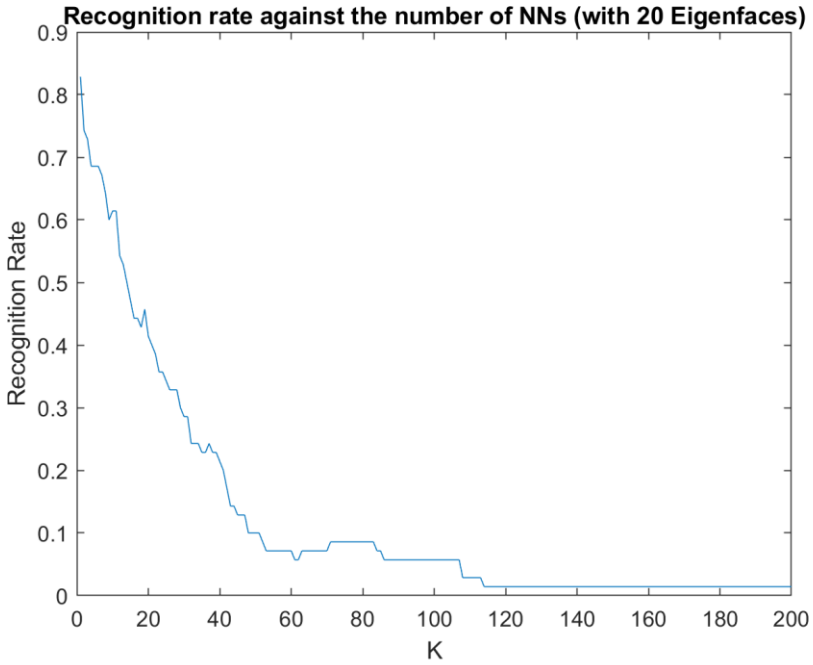

Figure 5. Recognition Rate against the number of Nearest Neighbour Classifiers

From the figure, it is seen that the recognition rate decreases as K gets large. As K is increased, training image projections that are further away from the test image projection are used to classify the test image. As K increase, wrong samples are also considered in KNN's voting scheme causing the test image to be classified incorrectly. For small K, training samples that are further away from the test sample will not be considered; hence the overall accuracy, in this case, would increase.

The following code computes and displays the recognition rate against K:

```
%% Part 2.11: Investigate the effect of K in KNN classifier:
% effect of K: You need to evaluate the effect of K in KNN and
% plot the recognition rate against K. Use 20 eigenfaces here.

% Set to 20 Eigenfaces: Part 2.6 and 2.7
Locationstrain=projectImages (Imagestrain, Means, Space);
Locationstest=projectImages (Imagestest, Means, Space);
Threshold = 20; % Number of eigenfaces to be used.
Distances=zeros(size(Locationstest,1), size(Locationstrain,1));
for i=1:size(Locationstest,1)
    for j=1:size(Locationstrain,1)
        Sum=0;
        for k=1: Threshold
            Sum=Sum+(Locationstrain(j,k)-Locationstest(i,k)).^2;
        end
        Distances(i,j)=Sum;
    end
end
% Sort best matching training images for test images (using min distance):
Values = zeros(size(Locationstest,1), size(Locationstrain,1));
Indices = zeros(size(Locationstest,1), size(Locationstrain,1));
for i=1:size(Locationstest,1)
    [Values(i,:), Indices(i,:)] = sort(Distances(i,:));
end
```

The code above follows from Part 2.6 and 2.7 to set-up the environment to work with 20 Eigenfaces. This code fragment is already discussed in its respective sections.

```
recog_rate=zeros(1,20);
% Redo the experiment at Part 2.9 for 20 Eigenfaces and K from 1 to 20.
for K = 1:size(Imagestrain,1)
    % Indices (70x200) hold values sorted from min to max for each row.
    % i.e, for each test, best fitting training samples are ordered with
    % increasing distance. (x-1)/5+1 is transformation to extract training
    % labels (used in previous sections)
    index_matrix = floor((Indices(:,:)-1)/5)+1;

    % For each test sample check the recognition
    correct_count = 0; % initialize correct count for each K tested.
    for idx = 1:size(Imagestest,1)
        id = Identity(idx); % Returns the label for the image, 1:40

        % Fetch K closest indices, use mode to find which class is assigned
        % If equal mode, smallest one is selected by MATLAB (shortcoming?)
        knn_output = mode(index_matrix(idx, 1:K));

        if (id == knn_output) % The recognition is correct
            correct_count = correct_count + 1;
        end
    end

    % Traversed through all samples, compute recognition rate:
    recog_rate(K) = correct_count ./ size(Imagestest,1);

end

% Plot the Recognition Rate:
figure;
plot(recog_rate);
title('Recognition rate against the number of NNs (with 20 Eigenfaces)');
ylabel('Recognition Rate'); xlabel('K');
```

This code segment is done to compute and display the effect of using different KNN values. It uses the matrix "Indices". "Indices" holds the training samples' index, sorted in ascending distances for each of the test samples. Using the transformation that follows Part 2.10, the actual labels for the training images can be extracted.

A for-loop can be executed for each of the testing samples. This for-loop can investigate first K training sample indices and use mode operation to vote and decide on the test image label. If this label is classified correctly, the recognition rate can be incremented. Note that mode operation returns the mode for the first K training indices associated with the testing sample. If there is a tie, MATLAB automatically picks the smallest value for the mode. This is not a problem, as the nearest neighbours are presented in ascending order, implying that MATLAB will choose the label with the smallest distance to the test image if there is a tie.

This operation is repeated 200 times (all training images are considered to classify the point) to determine the effect of "K" in classification using 20 eigenfaces with an outer loop. Finally, logged recognition rates are plotted.