Pattern Recognition Report

Orion Mathews
Imperial College London
omm15@ic.ac.uk
01053855

Paul Roever
Imperial College London
pr2214@ic.ac.uk
00948609

Abstract

Face recognition is a complex pattern recognition problem. Principle component analysis (PCA) is a standard technique used in face recognition, however discriminative information between classes may be lost in projection. To maximally separate data from different classes linear discriminant analysis (LDA) is preformed. LDA often provides better face recognition accuracy than PCA. A drawback of LDA is that there must be more training examples than features. Fisher-faces is a method used to solve this problem by initially using PCA to reduce the number of features and then preforming LDA. Face recognition performance can be further improved by fusing together an ensemble of different models.

1. Introduction

A set of faces can be described by a list of features. However, some features may be related and therefore redundant. PCA is used to extract the relevant information in the data set by obtaining the features that show the most data variation. Therefore the data can be represented using fewer features as the unnecessary correlations among the original set of features is removed.

PCA can be used for face recognition using a classifier such as nearest neighbours (NN). However, PCA simply finds the features that are most useful for representing data and may discard discriminating information between the classes. Linear discriminant analysis (LDA) finds the features that best separate data of different classes and usually offers better performance.

Each face image is 46x56 pixels and has been raster-scanned so that it forms a D dimensional column vector (where $D = 46 * 56 = 2576$ and is regarded as the number of features describing each face). The data set X contains 520 images of 52 different people, consequently there are 10 images of each person's face.

2. Cross Validation

Cross validation is used to assess how well the predictive model will generalize to unseen data. This is achieved by splitting the data set into two independent subsets, the training data set and the test data set. The test data remains unknown to the model, allowing the model's accuracy to be tested on new data. Cross validation reveals problems such as models under-fitting or over-fitting the data, due to the models being too simple or complex.

The data set X is partitioned into training data X_{train} and test data X_{test} by defining the number of images per person that will be used for training. The number of training images per person is defined as 7, causing the data set to be split in a ratio of 7:3 training data to test data. Therefore models are trained on 70% of the data (364 images = N_{train}) and tested on the remaining 30% (156 images = N_{test}).

For implementation of data partitioning see lines 19 to 28 in the source code (appendix A).

3. Eigenfaces (PCA)

3.1. PCA applied to training data set (X_{train})

Each face is a D-dimensional vector in the image feature space. A subspace is computed such that the projections of the data points onto the subspace have maximum data variance, this subspace is called the face-space (M dimensional, M << D). The face-space is spanned by the eigenfaces (the features that characterise variations among face images). Eigenfaces are the principle components of the set of face images, that is, they are the eigenvectors that correspond the the M largest eigenvalues of the face data's covariance matrix S.

$$S = \frac{1}{N_{train}} A * A^T \tag{1}$$

where N_{train} is the number of training images and A is the normalised training data.

Eigenvalues represent the amount of data variation associated with a certain direction. The larger the eigenvalue the more the data varies in that direction. An eigenvalue of zero means there is no data variation in that direction. Since S is real and symmetric all its eigenvalues are real. The eigenvectors associated with the M largest eigenvalues are chosen as it ensures the chosen directions have maximum data variance. When implementing face recognition with PCA and NN classification, M is chosen so that the sum of the first M eigenvalues is above 99% of the total sum of all the eigenvalues, therefore more that 99% of the information is retained. Given this condition, M is calculated to be 243. See lines 54 to 57 in source code for implementation. Figure 1 shows the value of each eigenvalue.

To map the training faces to the face-space, the mean face of the training data (see figure 2) is subtracted from each face and the result is projected onto the face-space. For the detailed procedure see slides 15 to 20 of the "Face Recognition by Eigenfaces: Subspace, PCA" lecture notes. Figure 3 shows the top 15 eigenfaces (the eigenvectors corresponding to the 15 largest eigenvalues of S).

The number of non-zero eigenvalues is the number of directions in which the data varies and is equivalent to the rank of the matrix S. $A \in \mathbb{R}^{D*N_{train}}$ and has columns consisting of the normalised training images.

$$rank(A) \leq min(D, N_{train}) = N_{train} \tag{2}$$

$$rank(A) = rank(A^T A) = rank(AA^T) \tag{3}$$

From equations (1), (2) and (3) it is clear that

$$rank(S) = rank(A) \leq N_{train} = 364 \tag{4}$$

As A is X_{train} normalised, the maximum number of non-zero eigenvalues can be further bounded to

$$rank(S) \leq N_{train} - 1 = 363 \tag{5}$$

Therefore the number of non-zero eigenvalues is the number of training faces -1 (N_{train} - 1).

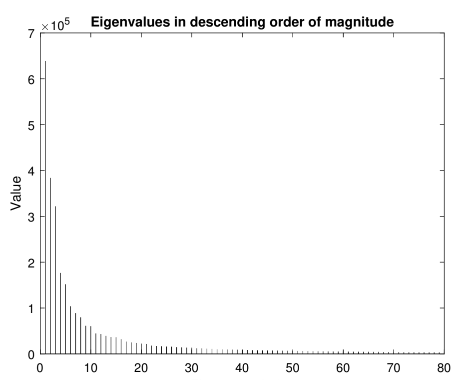


Figure 1. Eigenvalues in descending order

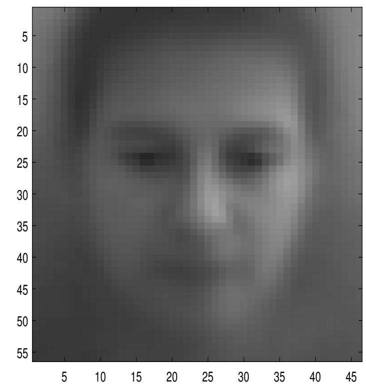


Figure 2. Average training data face

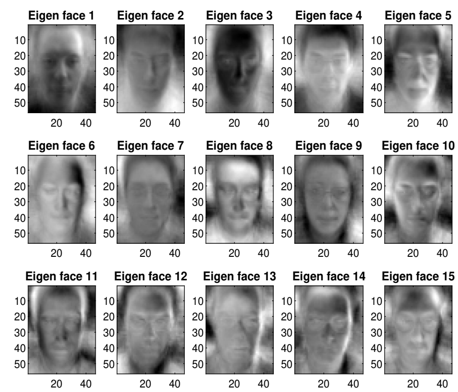


Figure 3. Eigenfaces corresponding to largest 15 eigenvalues.

3.2. Low dimension computation of the face-space

Using (1) to compute S means that $S \in \mathbb{R}^{D*D}$, as $D = 2576$ this matrix is very large, hence eigenvalue-eigenvector decomposition of S is computationally expensive. Therefore, we consider the eigenvalue-eigenvector decomposition of

$$\frac{1}{N_{train}} A^T A \tag{6}$$

$$(\frac{1}{N_{train}} A^T A) * \boldsymbol{V_i} = \lambda_i \boldsymbol{V_i}$$

Multiplying by A on the left:

$$(\frac{1}{N_{train}} AA^T A) * \boldsymbol{V_i} = \lambda_i A * \boldsymbol{V_i}$$

Using equation (1) and applying a simple substitution:

$$SA * \boldsymbol{V_i} = \lambda_i A * \boldsymbol{V_i}$$

$$S\boldsymbol{U_i} = \lambda_i \boldsymbol{U_i} \tag{7}$$

From equation (7) and identity (3) the non-zero eigenvalues of AA^T and $A^T A$ are identical. The corresponding eigenvectors can be obtained as $\boldsymbol{U_i} = A * \boldsymbol{V_i}$.

Using (6) to compute the data co-variance matrix results in $S \in \mathbb{R}^{N_{train}*N_{train}}$; as $N_{train} = 364 << 2576$, the matrix is much smaller than if S is computed using (1). The benefits of having a smaller co-variance matrix is that the time taken to compute the eigenvectors and eigenvalues is much less. For $S \in \mathbb{R}^{D*D}$ the time taken for eigenvalue-eigenvector decomposition is approximately 12.59 seconds compared to around 0.090 seconds when $S \in \mathbb{R}^{N_{train}*N_{train}}$. A possible shortcoming of this method is that the reconstruction error appears to be larger when using the low dimensional S. Overall the much faster computational speed outweighs this drawback and all further analysis will be carried out with the low-dimensional data covariance matrix.

3.3. Face reconstruction

Each face can be reconstructed by a linear combination of the eigenfaces. Perfect reconstruction of the original data set can theoretically be achieved if all eigenvectors are used. However using all eigenfaces would be computationally expensive and as shown in table 1 an average face reconstruction accuracy of 91.4% can be achieved with only 100 eigenfaces. As expected the more eigenfaces that are used in reconstruction the higher the average reconstruction accuracy, this is illustrated in figure 4 as well as in table 1.

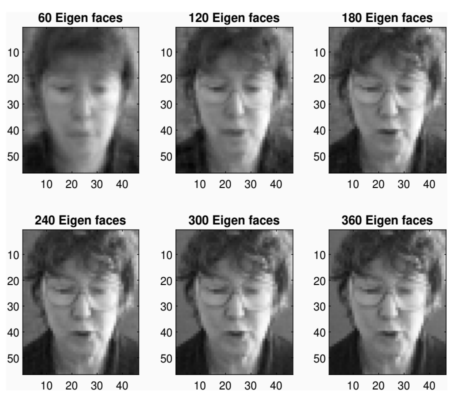


Figure 4. Face reconstruction using different number of eigenfaces

Number of eigenfaces used in reconstruction (M)	Average reconstruction accuracy (%)
40	87.1
80	90.4
120	92.3
160	93.8
200	95.1
240	96.2
280	97.3

Table 1. Table of eigenfaces used in reconstruction and corresponding accuracies

3.4. Nearest Neighbor (NN) classification

Face recognition using nearest neighbor classification is preformed by computing the face-space with all the training data, projecting the new image onto the face-space and classifying the face by comparing its position in the face-space with the positions of known individuals. The new image is assigned to the same class as the training image that is closest to it in the face-space.

As the number of eigenfaces learned increases so does the prediction accuracy (shown in figure 5). Initially the prediction accuracy increases rapidly but plateau's after reaching approximately 63%. After 100 learned eigenfaces there is very little increase in prediction accuracy. With 243 learned eigenfaces, recognition is 63.5% accurate. Figure 7 shows the confusion matrix for NN classification using 100 eigenfaces.

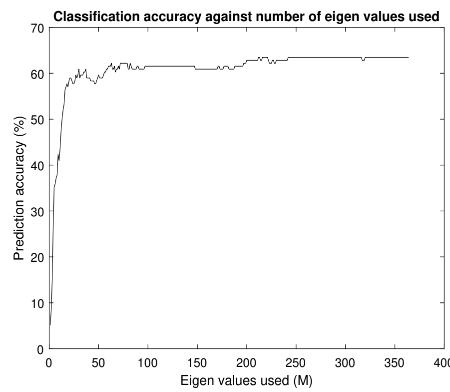


Figure 5. NN classification accuracy against number of eigenfaces learned.

Figure 6. Example of failure case. PCA with NN classification miss-classified the individual on the left. The image on the right is the image that was classed as most similar, though it is a completely different individual.

3.5. Alternative classification method

Alternatively, a face-space can be computed for each person. The new image would then be projected onto each of the face-spaces and the reconstruction error can be measured. The assigned class would be the one with least reconstruction error.

Face recognition using this method is 74.4% accurate where as the maximum accuracy for NN classification was 61.9%; both were averaged over five repetitions with differently shuffled data. The difference is that in this alternative method similarity to all images is taken into account instead of classifying based on the single most similar image. For comparison the confusion matrix for this alternate method of classification is included in appendix C.

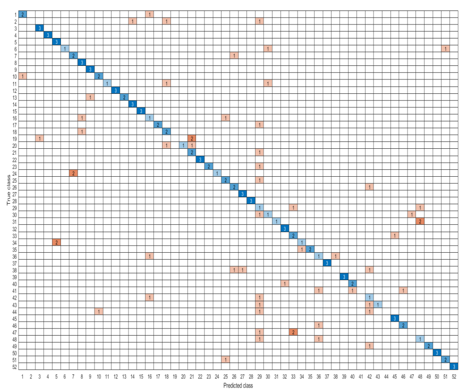


Figure 7. PCA with NN classification confusion matrix (M=100). For full figure see appendix C.

4. Combined Generative and Discriminative Subspace Learning

PCA finds the directions that maximise the projected data variance. These directions are the columns of the projection matrix W_{PCA}. W_{PCA} is found by solving the PCA objective function given in equation (8), where W is a matrix of projection directions and S is the data covariance matrix.

$$\begin{aligned} W_{PCA} = \arg\max_{W} \quad & \left| W^T * S * W \right| \\ \text{with} \quad & W^T * W = 1 \end{aligned} \tag{8}$$

LDA finds the directions that maximise the separation of data from different classes. These directions are the columns of the projection matrix W_{LDA}. W_{LDA} is found by solving equation (9), i.e. maximising the between class scatter while minimizing the withing class scatter. S_B is the between class scatter matrix and S_W is the within class scatter matrix.

$$W_{LDA} = \arg\max_{W} \quad \frac{\left| W^T * S_B * W \right|}{\left| W^T * S_W * W \right|} \tag{9}$$

S_W is invertible if $N_{train} > D$, if this is the case then W_{LDA} is found to be made up of the eigenvectors of $S_W^{-1} S_B$ (see slides 8 to 11 of "Discriminant Analysis: Fisherfaces" lecture notes for proof). If $N_{train} < D$ then S_W is singular and the number of dimensions must be reduced. Therefore PCA is preformed and the data is projected to a lower-dimensioned subspace where $N_{train} > D$ and S_W is invertible. This procedure results in objective function (10).

$$W_{LDA} = \arg\max_{W} \quad \frac{\left| W^T * W_{PCA}^T * S_B * W_{PCA} * W \right|}{\left| W^T * W_{PCA}^T * S_W * W_{PCA} * W \right|} \tag{10}$$

This is equivalent to maximising the numerator while keeping the denominator constant, giving equation (11).

$$\begin{aligned} W_{LDA} = \arg\max_{W} \quad & \left| W^T * W_{PCA}^T * S_B * W_{PCA} * W \right| \\ \text{with} \quad & W^T * W_{PCA}^T * S_W * W_{PCA} * W = k \end{aligned} \tag{11}$$

W_{LDA} can be found using the method of Lagrange multipliers as demonstrated below.

$$\begin{aligned} & L = (W^T * W_{PCA}^T * S_B * W_{PCA} * W) + \\ & \lambda(k - W^T * W_{PCA}^T * S_W * W_{PCA} * W) \end{aligned} \tag{12}$$

Setting gradient with respect to W to zero gives:

$$2W_{PCA}^T * (S_B - \lambda S_W) * W_{PCA} * W = 0 \tag{13}$$

Expansion and factorisation yields

$$(W_{PCA}^T S_W W_{PCA})^{-1} * (W_{PCA}^T S_B W_{PCA}) W = \lambda W \tag{14}$$

From (14), it is clear that W and λ are the eigenvectors and eigenvalues of $(W_{PCA}^T S_W W_{PCA})^{-1} * (W_{PCA}^T S_B W_{PCA})$. The combined PCA-LDA subspace is therefore given by

$$W_{opt} = W_{PCA} * W_{LDA} \tag{15}$$

5. PCA-LDA Face Recognition with NN classification

To perform LDA the within class scatter matrix S_W must be invertible, as described in section 4. Therefore PCA is used to reduce the dimensions of the training data set so that $N_{train} > D$. The training data is then projected onto the low dimension subspace and LDA is preformed on the projected data. See slides 14 to 27 of the "Discriminant Analysis: Fisherfaces" lecture notes for the procedure and lines 38 to 77 in the source code for the implementation (appendix B).

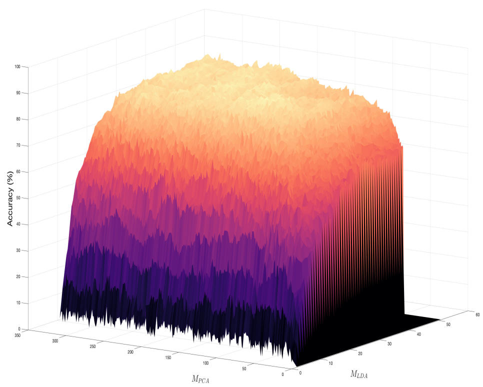


Figure 8. Accuracy heat-map generated by varying the number of eigenvectors used in PCA and LDA. For full image and a heatmap see appendix C.

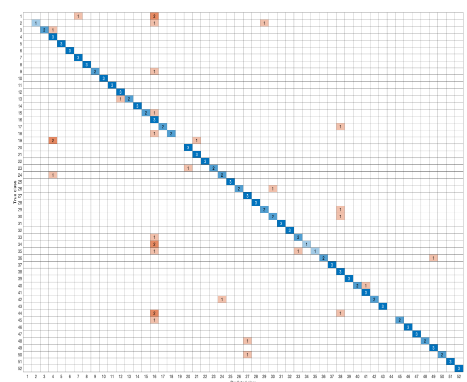


Figure 9. Confusion matrix for PCA-LDA face recognition with NN classifier. M_{PCA} is 312 and M_{LDA} is 51. For full image see appendix C.

Figure 8 displays the relationship between the face recognition accuracy for every combination of the number eigenvectors used in PCA (M_{PCA}) and LDA (M_{LDA}). If M_{PCA} and M_{LDA} are increased simultaneously the recognition accuracy also increases. The initial accuracy increase is rapid as M_{PCA} increase form 1 to 100 and M_{LDA} increases form 1 to 17 but plateaus after an accuracy of approximately 92%. If M_{PCA} is kept constant the recognition accuracy increases as M_{LDA} increases. The opposite is also true but to a lesser extent. M_{PCA} must always be greater than M_{LDA} to ensure that the within class scatter matrix S_W is invertible.

The rank of the within class scatter matrix (S_W) is the total number of images in the training data (N_{train}) minus the number of classes (C), i.e. $rank(S_w) = N_{train} - C$.

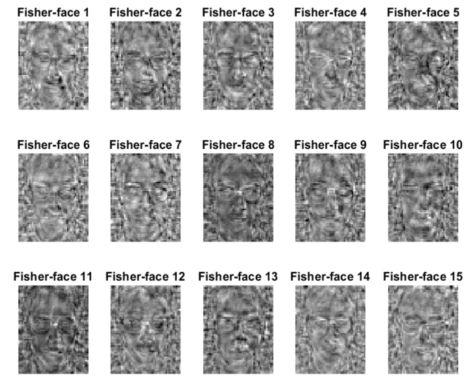


Figure 10. Top 15 Fisher-faces spanning the combined PCA-LDA subspace.

The between class scatter matrix S_B has rank $C - 1$, this is because S_B is a combination of C feature vectors and C feature vectors will define a subspace that has $C - 1$ dimensions or less. Using identity (16) it becomes clear that LDA can reduce dimensionality to at most $C - 1$, since $rank(S_w^{-1}S_b) \leq min(rank(S_w), rank(S_b)) = rank(S_b) = C - 1$.

$$rank(AB) \leq min(rank(A), rank(B)) \tag{16}$$

The confusion matrix for face recognition using PCA-LDA with a NN classifier is shown in figure 10. The model is successful at predicting the majority of classes (2 out of 3 test images are predicted correctly for 46 individuals). The average accuracy is 78.8%. The most miss-classified individuals are 1, 19 and 44. In these three cases all three test images were miss-classified. Figures C and 11 are examples of individuals that have been correctly classified and miss-classified respectively.

For a fixed value of M_{PCA}, PCA-LDA with NN classification gives a higher face recognition accuracy than just PCA with NN classification if M_{LDA} is above a certain value. Figure 12 shows how the accuracy of PCA-LDA changes as a function of M_{LDA} when M_{PCA} is fixed at 243. In general, for a given value of M_{PCA} the face recognition accuracy can be improved by using LDA.

6. PCA-LDA Ensemble learning

The performance of the PCA-LDA model can be improved by fusing an ensemble of different models, each one of which has varying parameters or data subsets. This prevents discriminative information from being discarded and over-fitting of the model. The ensemble of randomly different models is created by training each model on random

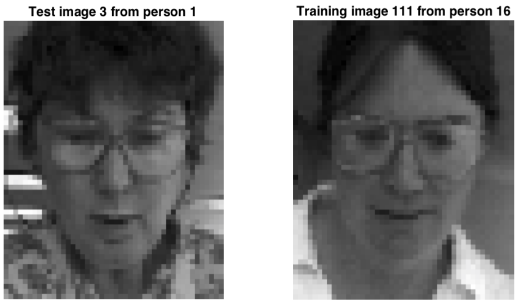


Figure 11. Example of failure case. PCA-LDA with NN classification miss-classified the individual on the left. The image on the right is the image that was classed as most similar, though it is a different individual. See appendix C for a success case.

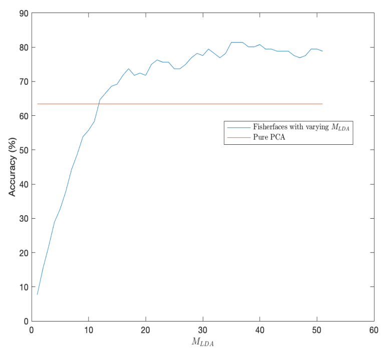


Figure 12. Accuracy of PCA and PCA-LDA when M_{PCA} is 312 and M_{LDA} is varied

subsets of the training data and randomising the model parameters.

One approach, known as bagging (discussed in Wang & Tang, 2006), involves splitting the training data into multiple "bags", each of which contains a subset of the overall training data. Figure 13 explores how many bags to use and how many samples each bag should contain. Although $N_{bags} = 80$ consistently displays slightly higher accuracy than $N_{bags} = 20$, it was much more computationally expensive to use 80 bags. Consequently, 20 bags were used, each containing 250 samples, corresponding to the highest expected accuracy for that number of bags.

Another approach, subspace randomisation[1], is performed by fixing a certain number of the PCA-LDA covariance matrix's eigenvectors (those with the largest eigenvalues). Then, a random subset of the remaining ones is chosen. W_{PCA}, W_{LDA}, and W_{opt} are then calculated from just the selected eigenvectors. The optimal combination of eigenvectors was found to be 130 total with the 45 largest remaining fixed and the remaining ones randomly chosen.

Table 2 shows the various accuracies achieved with the different ensemble methods. The baseline Fisherface performance is given at the top. Both ensemble methods, when combined with majority voting (whichever class is predicted the most), already perform better than the baseline. Note that the best-performing bag and subspace performed worse than the combinations of bags and subspaces, respectively. This implies that the random sampling approach works; each LDA model is tuned to unique aspects of the overall feature space, and combining them improves on their individual performance. Instead of combining the individual models using majority voting, the sum rule can be used to improve accuracy slightly further. See appendix C for PCA-LDA ensemble confusion matrix.

Face recognition and classification is a complex problem. PCA + NN is a good starting point for the task. Combining PCA with LDA gives the Fisherfaces algorithm, which performs much better than "pure" PCA. An ensemble approach randomises certain parameters of multiple discriminant models, allowing certain models to specialise to certain aspects of the recognition problem. Combining these results with a committee machine yields more modest, but still substantial, performance gains.

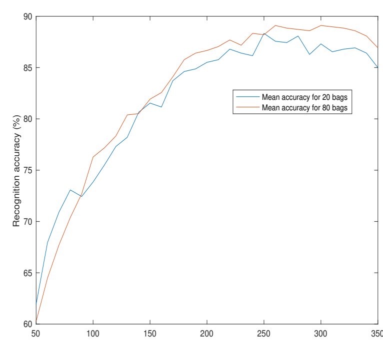


Figure 13. Accuracy of bagging (by majority voting) when $N_{bags} = 20, 80$ as a function of $N_{samples}$ per bag

	Average accuracy (%) over 20 trials
Baseline Fisherface	85.1
Bagging (Majority Voting)	87.7
Random Subspaces (Majority Voting)	89.4
Best-performing Bag	82.2
Best-performing Subspace	85.1
Combined subspaces & bags (Maj. Vot.)	89.9
Combined subspaces & bags (Sum Rule)	90.8

Table 2. Table of average accuracies for different types of ensemble PCA-LDA.

References

[1] Wang, X. & Tang, X., 2006. Random Sampling for Subspace Face Recognition. International Journal of Computer Vision, 70(1), pp.91–104.

A. Source code Q1

```
clear
close all
clc

load('face.mat');
rng(1)

%% Parameters

[D_features, N_faces] = size(X);                    % N_faces is number of faces (520). D_features is number of features/dimensions (2576)
N_faces_per_person = 10;                    % sets the number of faces per person
N_people = N_faces / N_faces_per_person;
train_per_person = 7;                    % sets the split between number of training faces and number of test faces (per person)
test_per_person = N_faces_per_person - train_per_person;
M_max = train_per_person * N_people;                    % M is number of eigen values used (M_max is largest possible M = number of training faces)
img_width = 46;
img_height = 56;

%% Splitting data set (for cross-validation)

train_split = [ones(1, train_per_person),  zeros(1, test_per_person)];
train_split = train_split(randperm(N_faces_per_person));    % shuffles the train and test images
train_indices = logical(repmat(train_split, 1, N_people));  % logical array that chooses which coloumns should be in the training data

l_train = l(:, train_indices);
l_test = l(:, ~train_indices);
X_train = X(:, train_indices);
X_test = X(:, ~train_indices);


%% PCA (S = 1/N * A * A')

X_train_avg = mean(X_train, 2); % compute average face vector (the mean value of each row/feature)
A = X_train - X_train_avg;      % subtract average face vector from each face/coloumn
A_test = X_test - X_train_avg;

tic
S = (1/N_faces)*A*(A');          % Computing covarience matrix
[U, Eval] = eigs(S, N_faces);    % Computing M eigen-vectors and eigen-values of covarience matrix
toc

Eval = real(sum(Eval, 1));

%% Plot Eigenvalues

figure
Eval_plot = stem(Eval);          % shows the value of each eigen value. largest eigen values correspond to best eigen vectors as data has most varience in these directions
set(Eval_plot, 'Marker', 'none');
title('Value of each eigen value in descending order of magnitude')
xlabel('Eigenvalue index')
ylabel('Value')
print -deps Eval_plot

%% Choosing M

M = find(0.99*sum(Eval) < cumsum(Eval));    % finds all indicies where the cumalutive sum of the eigen values are greater than 99% of the total sum of the eigen values
M = M(1);                                    % M is the first element that meets this criteria

%% PCA_low (S = 1/N * A' * A)

tic
S_low = (1/N_faces)*(A')*A;      % low dimensional computation of the eigenspace
[V_low, Eval_low] = eigs(S_low, M_max);
U_low = A*V_low;
U_low = normc(U_low);
toc

Eval_low = real(sum(Eval_low, 1));

%% Reconstruction while varying number of eigen faces

%W_project = U'*A;            % high dimension reconstruction (less error)
%X_train_rec = X_train_avg + U*W_project;

X_face_rec_mat = zeros(D_features, 18);
rec_accuracy_mat = zeros(18, M_max);
Eval_index = zeros(1, 18);
j = 1;
for i = 20:20:M_max
    W_project_low = (U_low(:, 1:i))'*A;
    X_train_rec_low = X_train_avg + U_low(:, 1:i)*W_project_low;

    X_face_rec_mat(:,j)=X_train_rec_low(:,1);

    X_train_rec_error = abs(X_train - X_train_rec_low);
    rec_accuracy = 100 - ((sum(X_train_rec_error)./ sum(X_train)) * 100);
    rec_accuracy_mat(j,:) = rec_accuracy;
```

```
    Eval_index(j) = i;
    j = j +1;
end

rec_accuracy_avg = mean(rec_accuracy_mat, 2);

%% plot fce renonstructions

figure
h = zeros(1, 6);
for i = 1:6
    h(i) = subplot(2, 3, i);
    img_1 = reshape(X_face_rec_mat(:,3*i), [img_height, img_width]);
    image(img_1, 'Parent', h(i));
    title( [num2str(i*60) ' Eigen faces']);
end
colormap(gray(255));
print -deps face_recs

%% plot reconstruction accuracy

figure
stem(Eval_index, rec_accuracy_avg);
title('Reconstruction accuracy as against the number of Eigenfaces used')
xlabel('Number of Eigenfaces used in reconstruction')
ylabel('Reconstruction accuracy (%)')
print -deps Rec_acc


%% KNN classification using PCA

predicted_class_matrix = zeros(M_max, test_per_person * N_people);  % stores predicted classes for each face
accuracy_matrix = zeros(1, M_max);                          % stores accuracies for each number of eigen values used

for i = 1:M_max

    X_train_proj = (A')*U_low(:,1:i);          % Projecting normalised faces onto the face-space. Rows are the projections of normalised faces onto the eigenface (row 1 is projection of face 1)
    X_test_proj = (A_test')*U_low(:,1:i);

    Idx = knnsearch(X_train_proj, X_test_proj);   % Perform a knnsearch between X_train_proj and X_test_proj to find indices of nearest neighbor and puts in coloumn vector

    predicted_class = l_train(Idx);
    prediction_results = predicted_class == l_test;
    prediction_accuracy = (sum(prediction_results)*100)/(test_per_person * N_people);

    predicted_class_matrix(i, :) = predicted_class;
    accuracy_matrix(i) = prediction_accuracy;
end

%X_train_rec = X_train_avg + U_low*X_train_proj';
%Error_rec = vecnorm(X_train - X_train_rec);

%% Plot average training face

figure
img_1 = reshape(X_train_avg, [img_height, img_width]); % takes the average face vector (D x 1) and changes it to an image matrix (W x H)
image(img_1);
colormap(gray(255));
print -deps PCA_avg_face

%%

figure
plot(accuracy_matrix);  % shows can stop at around 100 eigen values
title('Classification accuracy against number of eigen values used')
xlabel('Eigen values used (M)')
ylabel('Accuracy (%)')

figure
%confusionchart(l_test, predicted_class_matrix(100, :)); % selects the results obtained using 100 eigen vectors and creates a confusion chart from them

%% Classification using reconstruction

Error_rec = zeros(N_people, test_per_person * N_people);

for i = 1:N_people
    index = (i - 1) * train_per_person;

    class_train = X_train(:, index + 1:(index + train_per_person));    % takes each class of person

    class_avg = mean(class_train, 2);
    class_normalized = class_train - class_avg;
    S_class = class_normalized'*class_normalized;

    [V_class, Eval_class] = eigs(S_class, train_per_person -1);
    U_class = class_normalized*V_class;
    U_class = normc(U_class);

    test_class_norm = X_test - class_avg;
    test_class_projection = U_class'*test_class_norm;

    X_test_rec = class_avg + U_class*test_class_projection;
    test_difference = X_test - X_test_rec;

    Error_rec(i, :) = vecnorm(test_difference);
end

[Mins, I] = min(Error_rec, [], 1);

figure
%confusionchart(l_test, I);

results_reconst = l_test == I;
```

```
accuracy_reconst = sum(results_reconst)/length(l_test)*100;


%% Plot Error case

img_width = 46;
img_height = 56;

figure
subplot(1, 2, 1);
img_1 = mat2gray(reshape(X_test(:,31), [img_height, img_width])); % takes the average face vector (D x 1) and changes it to an image matrix (W x H)
imshow(img_1, 'InitialMagnification', 'fit');
title('Test image 31 from person 11');

subplot(1, 2, 2);
img_1 = mat2gray(reshape(X_train(:,123), [img_height, img_width])); % takes the average face vector (D x 1) and changes it to an image matrix (W x H)
imshow(img_1, 'InitialMagnification', 'fit');
title('Training image 123 from person 18');

colormap(gray(255));
print -depsc PCA_NN_Error
```

B. Source code Q3 fisherfaces

```
clear
close all
clc

%% setup
load('face.mat');
rng(1)

% dimensions
width = 46;
height = 56;

% set some Ns
N = size(X, 2);
N_faces_per_person = 10;
N_people = N / N_faces_per_person;
N_features = size(X, 1);

% generate train/test split
train = 7;
test = N_faces_per_person - train;

% create logical (boolean) indices to be used for splitting l, X
train_split = [ones(1,train), zeros(1,test)];

% shuffle train/test images
train_split = train_split(randperm(N_faces_per_person));

train_indices = logical(repmat(train_split, [1, N_people]));

% split dataset, train is the ones in the indices, test is the inverse
l_train = l(:, train_indices);
l_test = l(:, ~train_indices);

X_train = X(:, train_indices);
X_test = X(:, ~train_indices);

mean_train_image = mean(X_train, 2);
mean_class_images = zeros(N_features, N_people);

%% calculate S_W, S_B
S_W = zeros(N_people, N_features, N_features);

% create a mean image for each person
for i = 1:N_people
    index = (i - 1) * train;
    current_train = X_train(:, index + 1:(index + train));

    mean_class_image = mean(current_train, 2);
    mean_class_images(:, i) = mean_class_image;

    diffed_class_image = current_train - mean_class_image;

   % form within-class scatter matrix
   S_W(i, :,:) = diffed_class_image * diffed_class_image';
end

S_W = reshape(sum(S_W, 1), [N_features, N_features]);
diffed_class_mean_images = mean_class_images - mean_train_image;

S_B = (diffed_class_mean_images)*(diffed_class_mean_images)';

S_T = S_B + S_W;

M_pca = 312;
% M_pca = rank(S_W);

[W_pca, D_pca] = eigs(S_T, M_pca);

intermediate = (W_pca' * S_W * W_pca)\(W_pca' * S_B * W_pca);

accuracy_mldas = zeros(1, 51);

for M_lda = 1:51
    % M_lda = rank(S_B);

    [W_lda, D_lda] = eigs(intermediate, M_lda);

    W_opt = W_pca * W_lda;

    %% testing W_opt
    proj_train = (X_train - mean_train_image)' * W_opt;
    proj_test = (X_test - mean_train_image)' * W_opt;

    Idx = knnsearch(proj_train, proj_test);

    % use l to determine if we got the right person, results(n) = 0 if we did
    predicted_class = l_train(Idx);
    results = abs(l_test - predicted_class);

    % every time we get it wrong, make it a positive number, then turn it
    % into a logical array, invert and sum it to get the number of correct
    % test examples, divide by the total number of tests and turn into a
    % percentage
    accuracy = sum(~logical(results))/length(l_test)*100;
    accuracy_mldas(M_lda) = accuracy;
end

accuracy_pure_pca = 63.461538461538460;

plot(accuracy_mldas)
hold on
plot(ones(1, 51)* accuracy_pure_pca)
xlabel('$M_{LDA}$', 'Interpreter', 'latex')
ylabel('Accuracy (%)')
legend({'Fisherfaces with varying $M_{LDA}$', 'Pure PCA'}, 'Interpreter', 'latex', 'Location', 'best')

%%
% %% Plot Confusion Chart
%
% figure
% confusionchart(l_test, predicted_class);
```

```
% % print('./figures/q3_confusionmatrix_PCA_LDA.png', '-dpng')
%
% %% Plot Error case
%
% img_width = 46;
% img_height = 56;
%
% figure
% subplot(1, 2, 1);
% img_1 = mat2gray(reshape(X_test(:,3), [img_height, img_width])); % takes the average face vector (D x 1) and changes it to an image matrix (W x H)
% imshow(img_1, 'InitialMagnification', 'fit');
% title('Test image 3 from person 1');
%
% subplot(1, 2, 2);
% img_1 = mat2gray(reshape(X_train(:,111), [img_height, img_width])); % takes the average face vector (D x 1) and changes it to an image matrix (W x H)
% imshow(img_1, 'InitialMagnification', 'fit');
% title('Training image 111 from person 16');
%
% colormap(gray(255));
% % print('./figures/q3_error_case_PCA_LDA.eps', '-depsc','-tiff')

%% Plot Success case

img_width = 46;
img_height = 56;

figure
subplot(1, 2, 1);
img_1 = mat2gray(reshape(X_test(:,143), [img_height, img_width])); % takes the average face vector (D x 1) and changes it to an image matrix (W x H)
imshow(img_1, 'InitialMagnification', 'fit');
title('Test image 143 from person 48');

subplot(1, 2, 2);
img_1 = mat2gray(reshape(X_train(:,336), [img_height, img_width])); % takes the average face vector (D x 1) and changes it to an image matrix (W x H)
imshow(img_1, 'InitialMagnification', 'fit');
title('Training image 336 from person 48');

colormap(gray(255));
print('./figures/q3_success_case_PCA_LDA.png', '-dpng')

%% Plot 15 Biggest Fisherfaces
figure
h = zeros(1, 15);
for i = 1:15
    h(i) = subplot(3, 5, i);
    img_1 = mat2gray(reshape(W_opt(:,i), [img_height, img_width]));
    imshow(img_1, 'InitialMagnification', 'fit');
    title(['Fisherface ' num2str(i)]);
end
colormap(gray(255));
print('./figures/q3_fisherfaces.png', '-dpng')
```

C. Tables & Figures

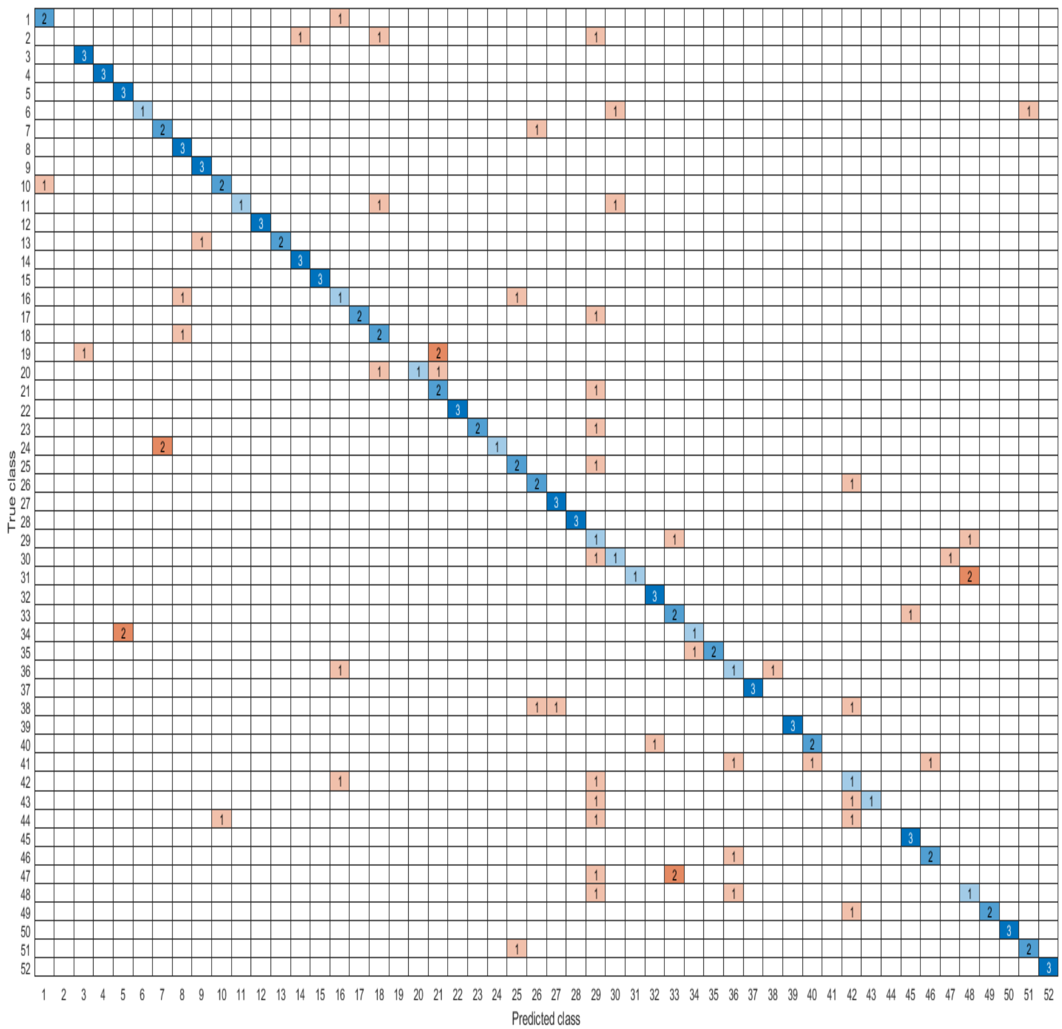
Table of eigenfaces used in reconstruction and corresponding accuracies

Number of eigenfaces used in reconstruction (M)	Average reconstruction accuracy (%)
20	83.7
40	87.1
60	89.0
80	90.4
100	91.4
120	92.3
140	93.1
160	93.8
180	94.5
200	95.1
220	95.7
240	96.2
260	96.8
280	97.3

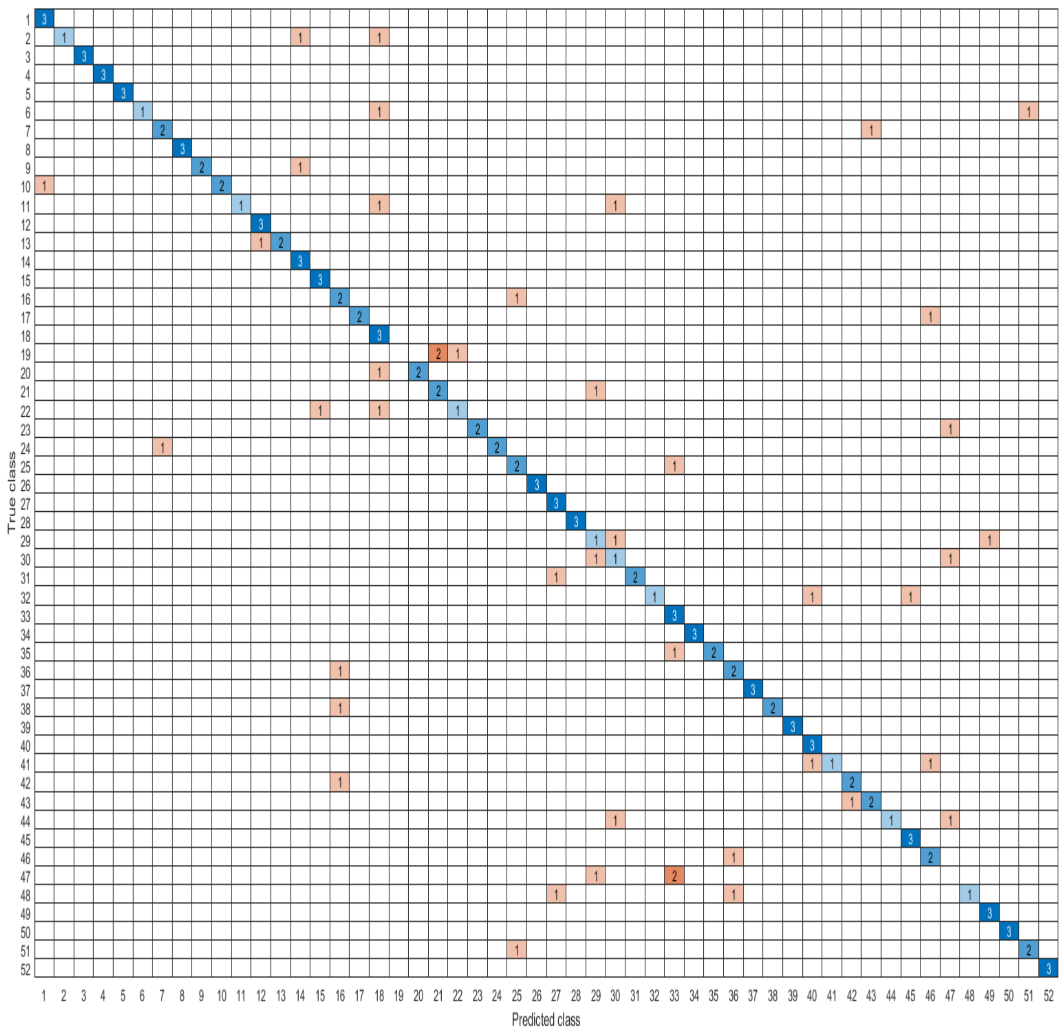
Average accuracies as a function of the number of subspaces used in subspace randomisation

$N_{subspaces}$	Average accuracy (%) over 20 trials
5	86.2
10	84.2
15	90.3
20	86.8
25	88.2
30	87.4
35	86.9
40	86.9
45	88.7
50	89.0
55	90.5
60	88.3
65	85.6
70	89.2
75	89.2
80	86.0
85	89.4
90	90.8
95	88.7
100	87.1

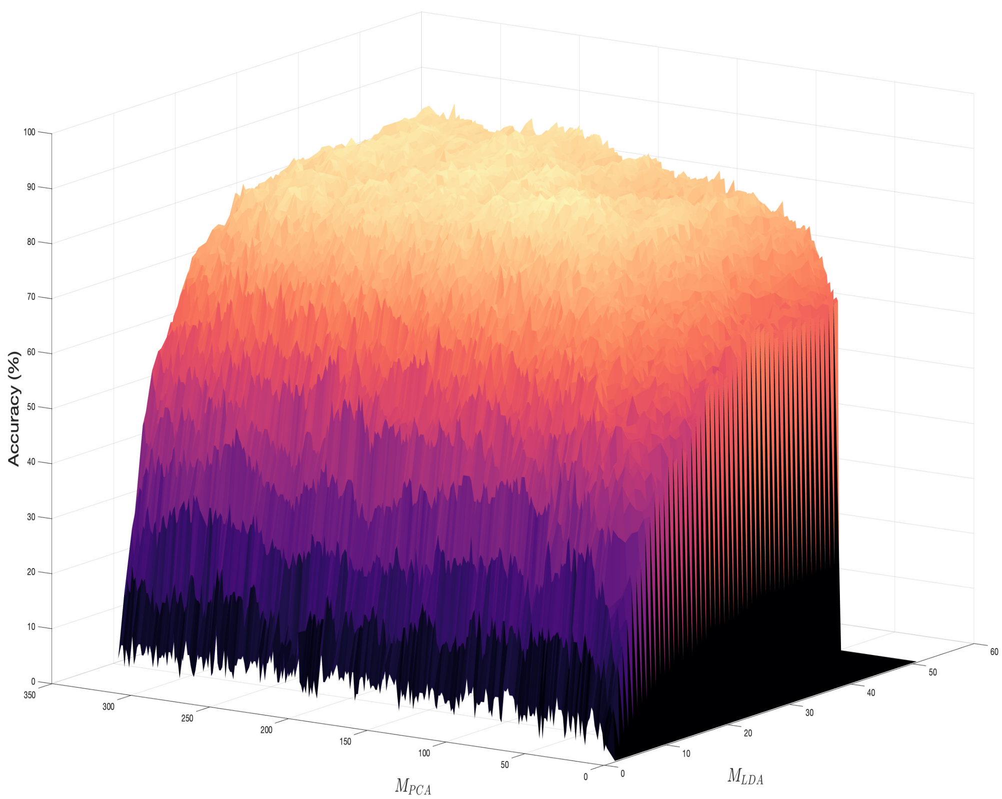
Confusion matrix for PCA using NN classification



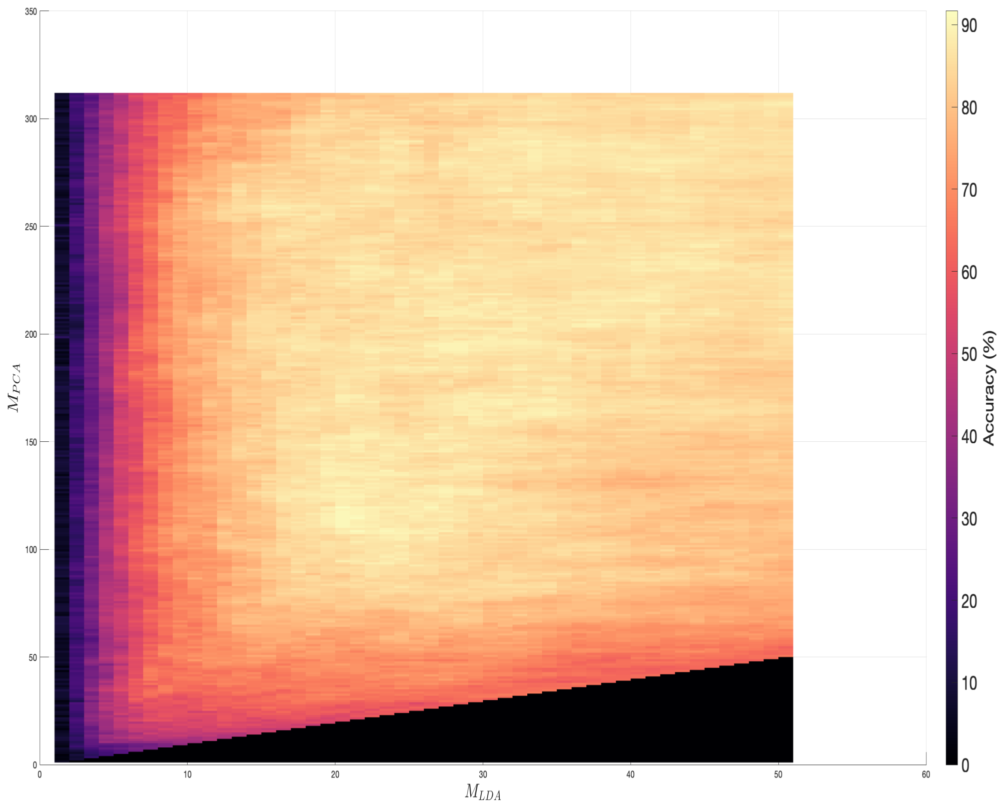
Confusion matrix for PCA using alternative method classification (i.e. using reconstruction errors)



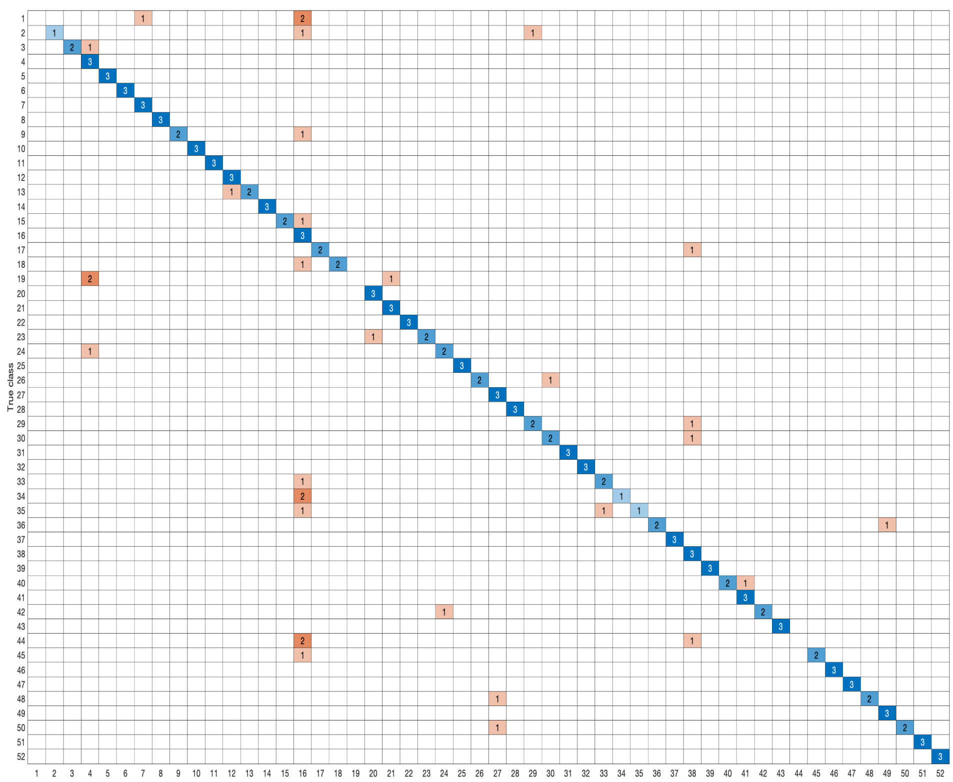
Accuracy heat-map generated by varying the number of eigenvectors used in PCA and LDA



Accuracy heat-map generated by varying the number of eigenvectors used in PCA and LDA



Confusion matrix for PCA-LDA face recognition with NN classifier. M_{PCA} **is 312 and** M_{LDA} **is 51.**

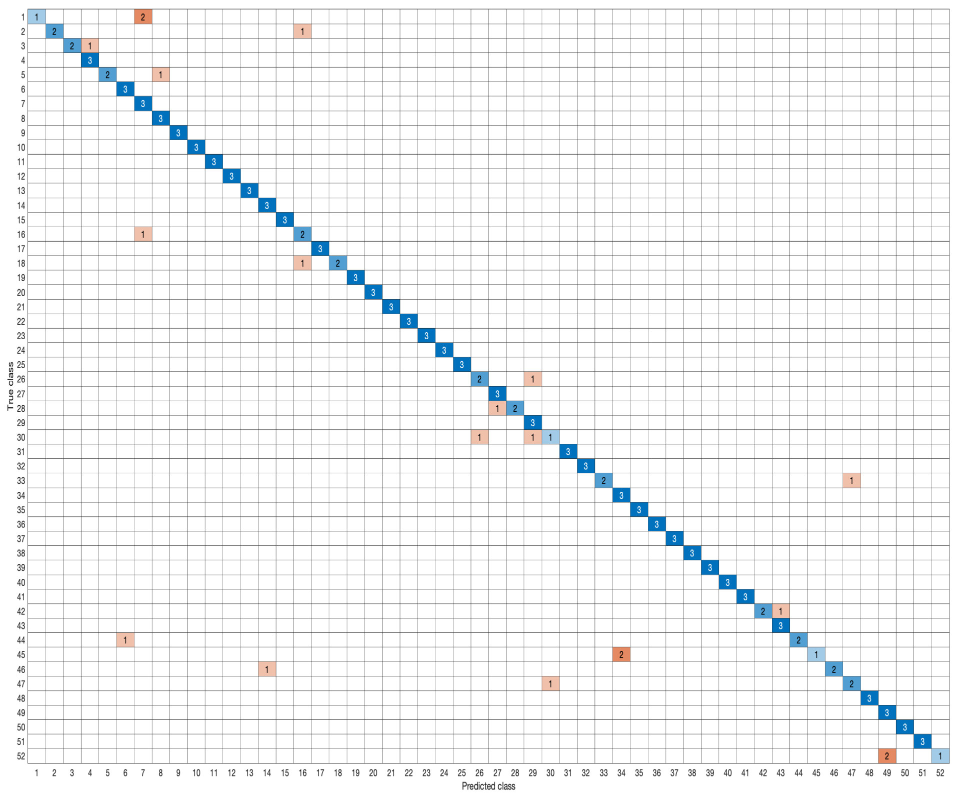


Example of success case for PCA-LDA with NN classification.

Test image 143 from person 48

Training image 336 from person 48

PCA-LDA ensemble confusion matrix



PCA-LDA ensemble confusion matrix