Pattern Recognition Coursework One

Lizhang Lin (CID: 00840705) & Lida Jin (CID: 00835494)
Department of Electrical and Electronic Engineering
Imperial College London

In face recognition, the goal is to develop a real-time system that can detect a person's face and then recognize him or her by comparing characteristics of the person's face with facial database of known individuals (training data). The process includes face detection, face alignment, feature representation and classification.

However, this coursework only focuses on feature representation and classification using PCA and multi-class SVM.

1. Data Partition

The facial data set provided are in the form of vectors in a very high dimensional space (2576×1), converted from profile images that are centred and of same size. It consists of 52 different people (classes), and 10 variations for each person.

In this case, using conventional validation may not generalize the model with small data (10 face data for each person), resulting in significant loss in modelling and testing capability. A fair approach to retain performance in this case is to use cross-validation as a powerful general technique. We chose K-fold cross validation method for data partitioning where K is 10. With K of value 10, the data set is divided into 10 subsets, and each subset consists of 52 randomly generated face images from 52 different people. One of the subsets is used as the test data set and the other 9 subsets are put together to form a training data set. Then the k results from the folds can then be averaged to compute a single estimation.

The advantage of using K-fold cross validation method is that every data point has a chance to be in a test set exactly once, and in a training set K-1 times. We can thus minimize the variance to improve performance as K increases. However, the downside is that it takes K times as much computation because the training algorithm reruns K times from scratch.

This trade-off between performance and computational time needs to be considered when choosing the value of K.

2. Feature Extraction & Feature Selection

2.1. PCA (AA^T)

PCA is applied to the training data which are partitioned using the K-fold cross validation method we just described.

The mean face vector is given by

$$\overline{\mathrm{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathrm{x}_n. \quad (1)$$

Simply subtracting the computed mean vector from training data set, we obtained the deviation matrix, denoted as A.

The covariance matrix S can then be computed by

$$S_1 = \frac{1}{N}AA^T \quad (2)$$

Where A is a matrix of deviation from the mean of face data in the training data set. The dimension of the computed covariance matrix is 2576×2576. Therefore, it has 2576 eigenvectors of dimension 2576×1 and 2576 eigenvalues. Note that A is a matrix over real numbers, which has a dimensional of 2576×468, and the rank of A is 468. The property of linear algebra states that if A is a matrix over real numbers then the rank of A and the rank of its corresponding Gram matrix are equal. Therefore, the rank of Gram matrix S_1 is 468, and we should expect at most 468 eigenvectors with non-zero eigenvalues to be used for the model. This can be further demonstrated by figure 1 & 2 where the 468th largest eigenvalue of matrix S drops to zero.

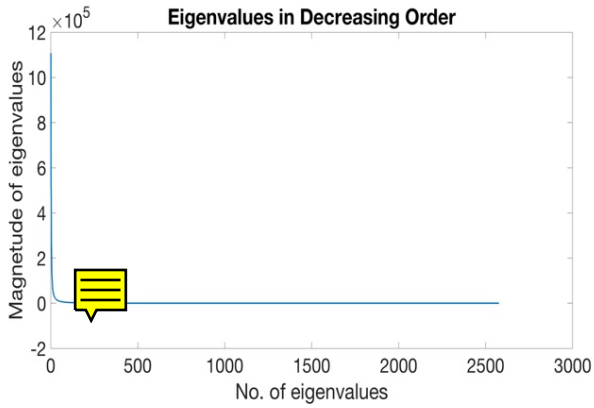

Figure 1 Eigenvalues of S_1 in descending order

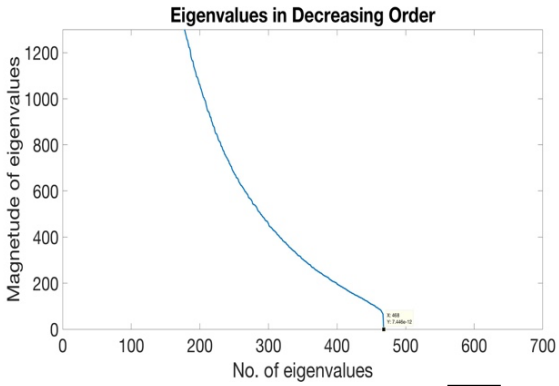

Figure 2 Zoom-in Figure 1

2.2. Feature Selection

Principle components with higher eigenvalues summarise more information about the training data set. In practice, applying all the principal components for face recognition is not desirable because it requires extensive amount of computation. In addition, it will very likely cause overfitting. We should also note that although selecting a very small number of principal components for face recognition would significantly reduce computation time, the model would be under-fitted because it learns little information from the training data set.

We therefore carefully considered the above trade-offs at feature selection stage, and decided to pick a reasonable number of eigenvectors with largest eigenvalues such that they summarise 92% of the total variance represented by the training but require not much computation time. There is nothing special about the value 92% because our intent was to pick a value in between 90% and 95%.

The selected number of eigenvectors for face recognition can be computed by equation

$$variance\% = \frac{\lambda_1+\lambda_2+\cdots\cdots+\lambda_n}{\lambda_1+\lambda_2+\cdots\cdots+\lambda_{468}} = 92\% \qquad (3)$$

where λ is the eigenvalue, and the subscripts indicate their ranks in a descending order (e.g. λ_1 is the largest eigenvalue), and n is the value we want to compute. Thus, we selected 85 largest eigenvectors. This number was applied throughout the rest of the PCA part of the coursework.

2.2. Alternative Method for PCA (A^TA)

The method we used in part II to compute covariance matrix is not practical as the covariance matrix S_1 $(D\times D)$ has very large dimensions. Alternatively, we compute covariance matrix S_2 by equation

$$S_2 = \frac{1}{N}A^TA \qquad (4)$$

This significantly lowers the amount of computation by reducing the dimension of covariance matrix S to 468×468 $(N\times N)$.

There are only 468 eigenvectors but all of them have non-zero eigenvalues. And the non-zero eigenvalues obtained here are identical to those obtained from the covariance matrix S_1. The mathematical proof is as follows.

The eigenvectors v of covariance matrix S_2 is

$$S_2v = \mu v \qquad (5)$$

where μ is the eigenvalue and v is the eigenvector.

Substitute equation (4) into (5) gives

$$\frac{1}{N}A^TA\ v = \mu v \qquad (6)$$

Multiply both sides of equation (6) by Matrix A

$$\frac{1}{N}AA^TAv = \mu Av \qquad (7)$$

Substitute equation (2) into equation (6) gives

$$S_1Av = \mu Av \qquad (8)$$

Let $u = Av$, then

$$S_1u = \mu u \qquad (9)$$

Comparing equation (8) with (4) we can conclude that covariance matrix S_1 and S_2 have the same eigenvalues.

Now consider the eigenvectors of S_1 and S_2. Figure 3 & 4 picked some eigenfaces generated from the corresponding normalised eigenvectors of S_1 and S_2 respectively, arranged in descending eigenvalue order. It is obvious that corresponding eigenfaces from S_1 and S_2 are of same profile. This implies that the uncorrelated eigenvectors extracted from the training images using two different PCA methods spans the same eigenspace. You might notice that the brightness of some eigenfaces in figure 3 & 4, such as eigenface 3, are opposite to each other. We think the possible explanation for this is that some of the eigenvectors of S_1 and S_2 are in opposite directions though they lie in the same eigenspace.

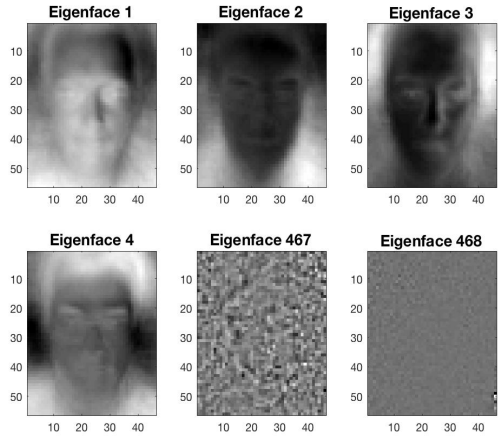

Figure 3: eigenfaces from S_1 (AA^T)

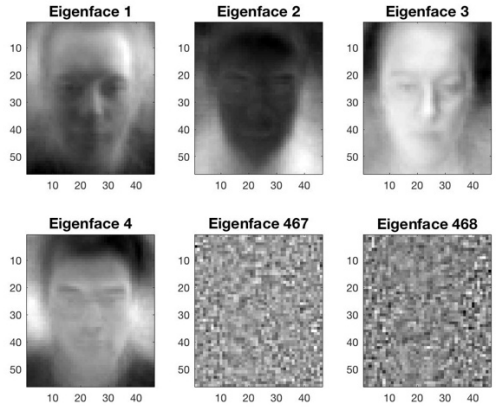

Figure 4 eigenfaces from S_2 (A^TA)

To conclude our analyses in this subsection, both methods of PCA yield eigenvectors that lie in the same eigenspace, and are of same eigenvalues. Applying PCA using covariance matrix $\frac{1}{N}A^TA$ is a more efficient method as it reduces computational time for computing covariance matrix. At the same time, it is equally reliable since the principal components extracted using the alternative method span the same eigenspace as compared to the original method. This method will therefore be used throughout the rest part of the coursework.

3. Face Reconstruction

In our experiment, there were 468 PCA bases learnt from the training images. In theory, we could obtain better reconstruction result if larger number of eigenvectors were used. Also, we should expect perfect reconstruction for training images but less significant results for test images if all the 468 eigenvectors are used for reconstruction. This is because PCA only extracts feature from training images. Some features contained in test images were not included in the training images.

The above statements can be justified by figure 5 & 6, where different number of eigenvectors were used for the reconstructions of test and training images respectively. From the results, we obtained less reconstruction errors as the number of PCA bases used gradually increases. In addition, looking at figure 5, it is obvious that perfect reconstructions can be achieved for training images when applying all the PCA bases for the process. However, perfect reconstruction is not possible for test images shown in figure 6.

Figure 5 Varying the Number of Eigenvectors Used for Training Image Reconstruction

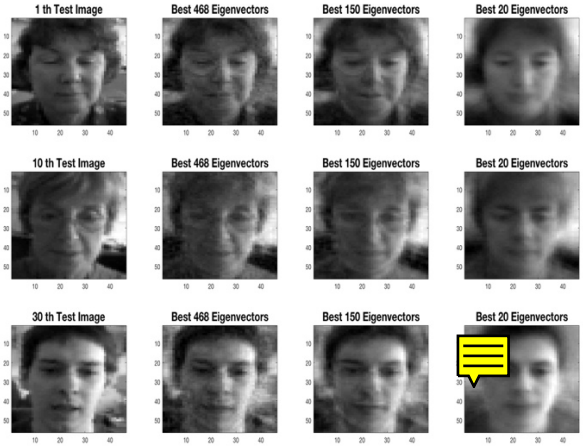

Figure 6 Varying the Number of Eigenvectors Used for Test Image Reconstruction

The above analysis about the relationship between reconstruction error and the number of eigenvectors used can be further summarised and proved by figure 7. The reconstruction error in the plot is calculated as the Euclidean distance between the original data and its estimate.

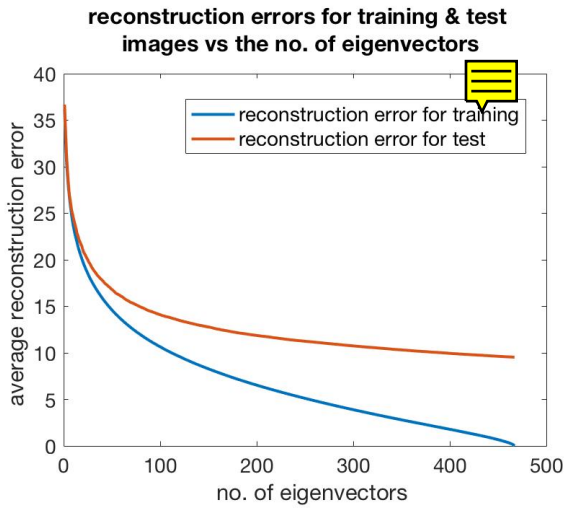

Figure 7 Reconstruction error for training & test images against the number of eigenvectors used

4. Face Recognition – Nearest Neighbour

Having the learnt model, nearest neighbour (NN) classification method was performed to recognize an unknown face image from the test data set.

Normalised test images are projected onto the eigen-subspace spanned by the training data set. Their projections are then compared with that of all training images. The training class that has the smallest distance to the test image is then assigned.

The recognition accuracy of the method varies with the number of eigenvectors used to form the eigen-subspace. We thus plotted figure 8 to illustrate the relationship of accuracy and time for computation against the number of eigenvectors. The accuracy is computed as the average of results from the 10 folds. It is worth noting that in figure 8, average accuracy increases significantly at small number of eigenvectors, and then smoothly reaches a threshold of 62.69% when all the number of eigenvectors are applied. The amount of computational time has a linear relationship against the number of eigenvectors that are selected for face recognition.

Remember that we selected 85 best eigenvectors for face recognition, which corresponds to an average of 61.35% success rate which is approximately the same as that of peak value. In addition, given linear relationship between computational time and the number of eigenvectors, our choice of eigenvectors brings more than 5 times reduction in terms of computational time. Thus, the figure further justifies that our choice of eigenvectors for face recognition has a good trade-off between computational time and accuracy.

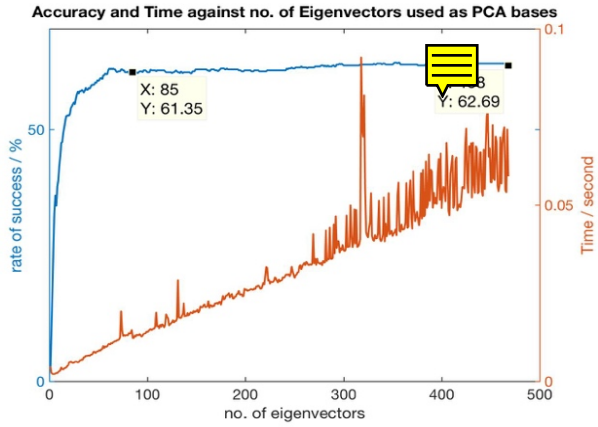

Figure 8 Accuracy & time of NN method against the number of eigenvectors used for face recognition

Figure 9 and 10 are examples of success and failure cases respectively. Test images for both cases are the image of 1st and 2nd people (classes) respectively. For the failure case, it is obvious that the predicated image and actual image looks very similar in terms of their face structure; therefore, after projecting the test face image into the eigen-subspace, the NN method finds that the Euclidean distance between the projection of the actual image and the predicted image in the training data set is minimised.

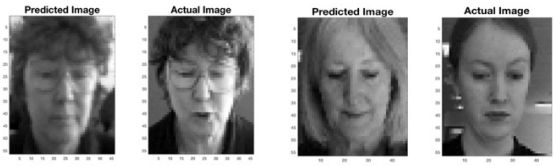

Figure 9 Successful case

Figure 10 Failure case

Figure 11 is the confusion matrix for the nearest neighbour classification method where the columns and rows are predicted results and actual results respectively. It conveniently visualises the performance of the NN method. For example, the success and failure case we just mentioned can be observed from the second row (counting from bottom to top) of the confusion matrix.

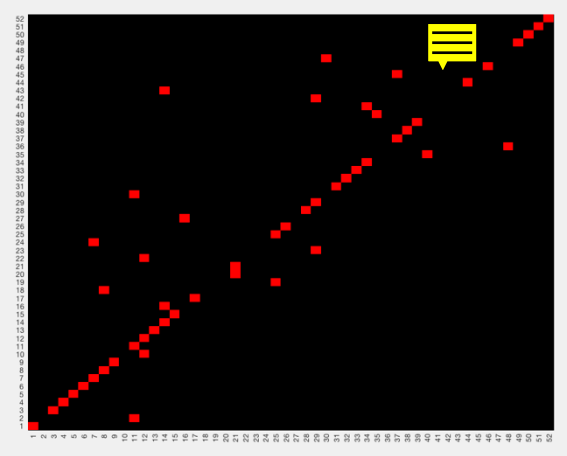

Figure 11 Confusion matrix for the NN method

5. Support Vector Machine

Two-class Support Vector Machines (SVM) can be combined to solve multi-class classification problems by adopting either the one-versus-all (OvA) or the one-versus-one (OvO) method.

For an M-class problem, OvA algorithm trains M separate SVMs so that each is a binary classification distinguishing that class from the rest. Recognition is achieved by applying each SVM to the query image and finding the class with maximum output value (probability). In contrary, the OvO algorithm distinguishes between all possible pairs of classes and generates $\frac{M\times(M-1)}{2}$ classifiers. Each SVM votes on the test data and the category with the highest votes would be assigned. However, OvO is much slower, of computational complexity $\mathcal{O}(N^2)$, whereas for OvA it is only $\mathcal{O}(N)$ with N being the number of classes.

5.1. Different Kernel Functions

In some cases, w the given data-set is non-linear, the Kernel Trick can be used to map data points to a higher dimension so that it can be linearly separated by a decision hyperplane. The Kernel functions investigated in this report include linear, polynomial and radial basis function (RBF) kernel.

The linear kernel is defined as $k(x,x') = x^T x$, which has computational complexity of $\mathcal{O}(K)$ where K is the dimension of input. It is faster than the other kernel functions due to its simplicity. However, there is no change to the classification accuracy after varying the error weighting constant C as this kernel type cannot over fit.

The polynomial kernel is defined as $k(x,x') = (\gamma x^T x' + coef)^M$, and the complexity is $\mathcal{O}(K\times N)$. Where M is the order of polynomial, a larger value of M projects the data set onto a higher dimension, and γ determines the shape of the hyperplane.

The RBF kernel is defined as $k(x,x') = e^{-\gamma(\|x^T - x'\|)^2}$ with $\mathcal{O}(N^2)$ computational complexity. The kernel scale σ, which is equal to $\frac{1}{\gamma}$, determines the size of region around the support vector which it has influence on. A large value of sigma will make the support vector have more influence over a larger area and vice versa.

To perform face recognition, the same data partition as previous were used and the performance were compared with PCA based NN classification in section 4. Two types of feature vectors, namely the raw-intensity vectors and PCA coefficients, were tested with both OvA and OvO SVMs respectively. For each combination, an exhaustive search for different parameters was used to maximise the accuracy.

5.2. PCA Coefficients vs Raw-intensity Vectors

By comparing PCA coefficients and raw-intensity vectors as inputs, a trade-off between execution time and accuracy must be considered. PCA coefficients is much faster, which is as expected because it has less dimension compared to that of raw data (2576). However, accuracy is generally lower than using raw-intensity vectors as inputs. For RBF using OvA, PCA coefficient achieved an average accuracy of 92.31% whereas that obtained by raw-intensity vectors was 98.08%.

5.3. Comparison of OvA and OvO

With all kernel functions, the OvA algorithm consistently outperforms OvO, this is because OvA is a less complex algorithm and is less likely to overfit than OvO. It also has a significantly lower computational complexity, therefore we concluded that OvA is a better multi-class SVM method to use. Thus, the results shown in this section were mainly focused on OvA. Hence, by varying the input vectors and analysing different types of kernel functions with varied range of parameters, the results are summarised in Figure 12 below. OvA results showed that polynomial kernel gives the best averaged accuracy, 98.08%, for both PCA and raw-intensity vectors inputs. The C value used was 50 and the order of polynomial was 2 in both cases. RBF also achieved similar level of accuracy for raw data inputs but was reduced to 92.31% for PCA. Linear kernel has the worst performance, obtaining 94.23% and 88.46% for raw data and PCA respectively. For OvO, the results are 84.62%, 88.46% and 86.54% for linear, polynomial and RBF kernel respectively using PCA inputs.

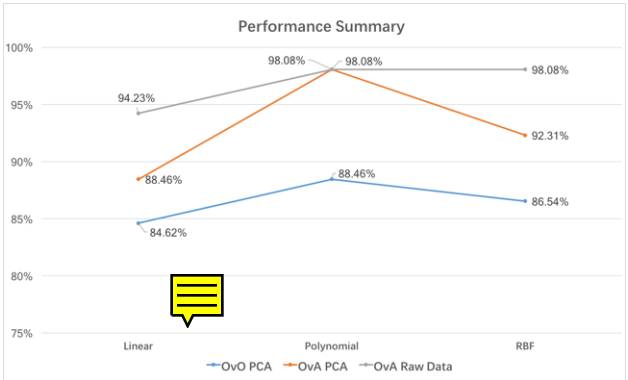

Figure 12 Performance summary of different kernel functions

The process of parameter selection and investigation is identical for all cases involving polynomial and RBF kernels. An example is shown below in Figure 13, which is the accuracy plot with different C and Sigma values for RBF kernel using OvO algorithm. When the σ value is very small, the classifier tends to overfit thus accuracy is lowered. The optimal σ here is 54. In addition, a small value of C (optimally 2) limits the error penalty which in turn limits overfitting.

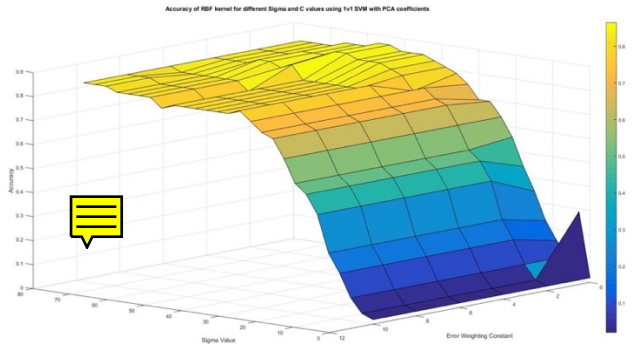

Figure 13 Accuracy plot of RBF kernel for different Sigma and C values using OvO SVM with PCA coefficients

Figure 14 below shows the confusion matrix using OvA SVM with PCA coefficients for different types of kernels compared with NN classification. The results show clearly that all kernel types outperform the NN method

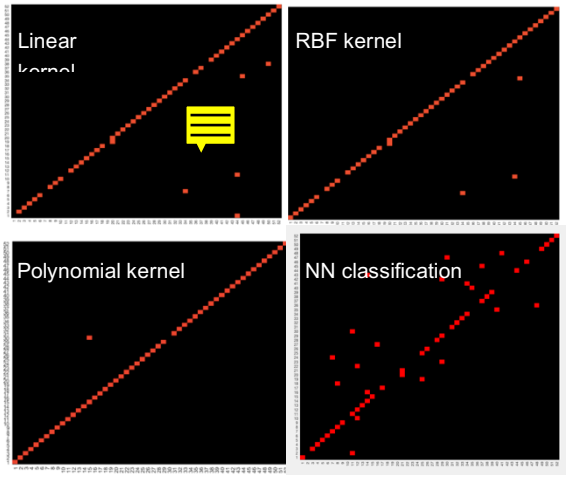

Figure 14 Confusion matrix for different kernel with OvA compared with NN classification method

An example of decision boundary of RBF kernel and support vectors is shown in Figure 15. The classification is between class 1 and 2 and all the support vectors are highlighted with a green square. It should be noted that PC1 and PC2 are the largest two eigen vectors as it contains the most amount of information.

As for the time efficiency, there exists a trade-off between the number of PCA coefficients used and the execution time required. The number of PCA coefficients used is directly proportional to the accuracy of classification, since a larger number of eigenvectors gives the classifiers more features and information. The results for RBF kernel is summarised in Figure 16. The data cursors indicate that applying only 85 PCA coefficients gives 81.54% which is approximately 10% less than the accuracy of applying all PCA coefficients. In terms of time efficiency, the former requires four times less amount of computational time, only 0.74 second. Further, the relationship between computational time and the number of PCA coefficients used is what we expected. As mentioned previously, the time complexity for RBF kernel is $\mathcal{O}(N^2)$.

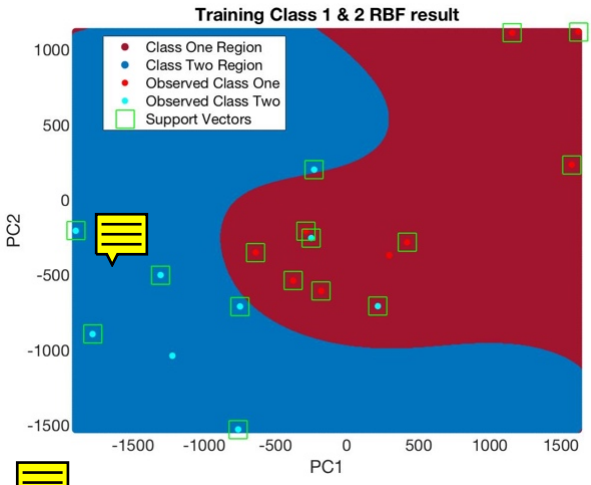

Figure 15 Support vectors for Class 1 and Class 2 Classification

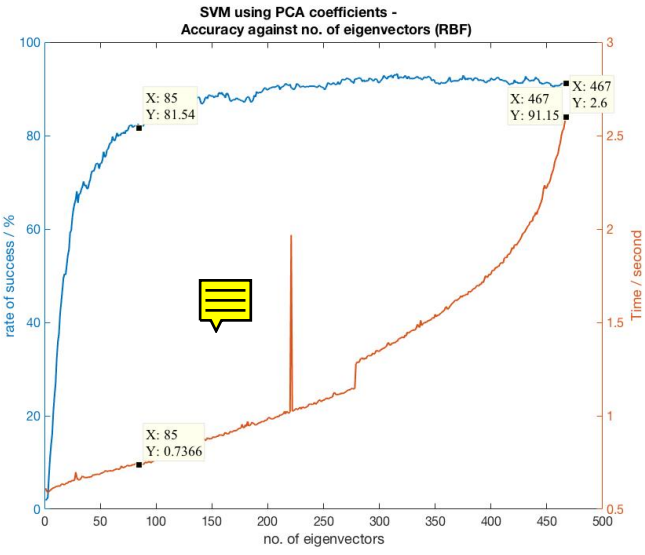

Figure 16 Accuracy and Computation time against number of Eigenvectors

Appendix

```
clc
clear
close all

load face.mat


%10-fold crossvalidation
%10 items in each class and 9 data into training set, 1 into test set
k=10;                                %Define ratio of partition, k is the
proportion sorted into test set
c = cvpartition(l,'Kfold',k);        %Create partition object


figure;
accuracy_1vAll = zeros(10,468,size();
elapsedTime_1vAll = zeros(10,468);

for j=1:10
    TestIdx=test(c,j);                    %Create index list for test set
    TrainingIdx=training(c,j);            %Index list for training set
    testData=X(:,TestIdx);
    trainData=X(:,TrainingIdx);

    %find mean face image for training data
    mean_face = mean(trainData,2); %return a column vector which is the mean
of training data
    phi_train = trainData - repmat(mean_face,[1,468]); %Obtain train data
    S = (phi_train' * phi_train)/size(phi_train,2); %A'A

    %compute and normalise the eigenvectors of covariance matrix S
    [eig_vec, eig_val] = eig(S);
    eig_vec = phi_train * eig_vec;
    eig_vec = normc(eig_vec);
    [eig_val_sort, eig_val_sort_index] = sort(diag(eig_val),'descend');



        for M = 1:size(trainData,2)
            %compute the best M eigenvectors
            tic;
            M_eig_vec = eig_vec(:, eig_val_sort_index(1:M));
                train_projection = phi_train' * M_eig_vec;

                test_projection = testData'* M_eig_vec;

                train = train_projection;
                test = test_projection;

            decision_val = zeros(52,52);
            %% 1vAll
            for i = 1:52
            % creating different labels for each loop
                label_test = -ones(52,1);
                label_test(i) = 1;
                label_train = -ones(468,1);
                label_train(((i-1)*9+1):(i*9)) = 1;


            svm_1vAll = fitcsvm(...
```

```
            train, ...
            label_train, ...
            'KernelFunction', 'rbf', ...
            'PolynomialOrder', [], ...
            'KernelScale', sigma_1vAll, ...
            'BoxConstraint',C_1vAll, ...
            'Standardize', true, ...
            'ClassNames', [1; -1]);

            [~,score] = predict(svm_1vAll,test);
            decision_val(:,i) = score(:,1);
            end
            [x,result] = max(decision_val);
            accuracy_1vAll(j,M) = 1 - (nnz(result - (1:52))/52);

            elapsedTime_1vAll(j,M)=toc;



    %% 1v1
        label=zeros(52,1326);
        a = 1;
        for i = 1:52-1
            for k = (i+1):52
                category_i = train(((i-1)*9+1):(i*9),:);
                category_k = train(((k-1)*9+1):(k*9),:);
                train_i = vertcat(category_i,category_j);
                label_train = [i i i i i i i i i k k k k k k k k]';
                svm_1v1 = fitcsvm(...
                train_i, ...
                label_train, ...
                'KernelFunction', 'rbf', ...
                'PolynomialOrder', [], ...
                'KernelScale', sigma_1v1, ...
                'BoxConstraint',C_1v1, ...
                'Standardize', true, ...
                'ClassNames', [i;k]);
                [label(:,a),~] = predict(svm_1v1,test);
                a = a+1;
            end
        end

        predictResult = mode(label,2);
        correct_num = 0;

        for n = 1:52
            if (predictResult(n,1) == n)
            correct_num = correct_num +1;
            end
        end

        accuracy_1v1 = correct_num/52;

    end

end
% plot accuracy (success rate)
[ax, h1, h2] =
plotyy([1:468],mean(accuracy_1vAll)*100,[1:468],mean(elapsedTime_1vAll));
title('Accuracy against no. of Eigenvectors used as PCA bases');
xlabel('no. of eigenvectors');
ylabel('rate of success / %');
axes(ax(2)); ylabel('Time / second');
```

```
set(ax(2),'fontsize',16);
set(ax(1),'fontsize',16);
set(ax(2),'linewidth',1);
set(ax(1),'linewidth',1);
set(h1,'linewidth',2);
set(h2,'linewidth',2);
alldatacursors = findall(gcf,'type','hggroup');
set(alldatacursors,'FontSize',18)

%%  Exhaustive Search Loop

grange = 0.01:20:500;
crange = 0.001:50:2000;

accuracy = zeros(size(grange,2),1);
acc_all = zeros(size(grange,2),size(crange,2));

b = 1;
tic
for c = crange
    a = 1;

    for g = grange
        decision_val = zeros(52,52);

        for i = 1:52
        % creating different labels for each loop
            label_test = -ones(52,1);
            label_test(i) = 1;
            label_train = -ones(468,1);
            label_train(((i-1)*9+1):(i*9)) = 1;


        svm_1vAll = fitcsvm(...
        train, ...
        label_train, ...
        'KernelFunction', 'RBF', ...
        'PolynomialOrder', [], ...
        'KernelScale', g, ...
        'BoxConstraint',c, ...
        'Standardize', true, ...
        'ClassNames', [1; -1]);

        [label,score] = predict(svm_1vAll,test);
        decision_val(:,i) = score(:,1);
        end


        [x,result] = max(decision_val);
        accuracy(a) = 1 - (nnz(result - (1:52))/52);
        a = a+1;
    end

    acc_all(:,b) = accuracy;
    b = b+1;
end
toc
```