Coursework on PCA and Multi class SVM for Face Recognition

Ashish Pandey
Dept. of Electrical and Electronics Engineering
Imperial College London
ashish.pandey17@imperial.ac.uk
CID-01383450

Ilias Chrysovergis
Dept. of Electrical and Electronics Engineering
Imperial College London
ilias.chrysovergis17@imperial.ac.uk
CID-01449042

Abstract

Criticality of face recognition in multiple applications such as law enforcement, security surveillance, video indexing and access control amongst others calls for a robust solution applicable in various situations. Starting with the Eigenface Algorithm for face reconstruction and recognition using PCA, which is the standard technique for dimension reduction and feature selection, and Nearest Neighbor classifier we study the eigenvectors and eigenvalues obtained. The effect on distortion error during face reconstruction and recognition accuracy while performing face recognition is observed by varying associated parameters. Effect of varying distribution between training and testing datasets on observed results is also discussed in our report. Support Vector Machine (SVM) provides superior performance in generalization and in tackling high-dimensional data. Owing to its advantages, realization of OVO and OVR multi-class SVM for enhanced face recognition is done in MATLAB©. Results for both extensions are compared on basis of recognition accuracy, SVM parameters i.e. kernel type, kernel parameters, C and time-efficiency of SVM training/testing amongst other. Sample success and failure images are provided for both techniques discussed.

1. Introduction

Considerable importance in the recent past has been given to problems pertaining to face recognition and face reconstruction, owing to their application in multitude of domains such as security surveillance, access control, mugshot recognition, and human computer interaction [1]. Immense research in this field has resulted in development of multiple algorithms. A few popular ones include PCA [2], SVM [3], AAM [4] and HMM [5]. In our report we will limit our focus to discussions on PCA and multi-class SVM. In essence, face recognition is a high-dimensional pattern recognition problem. Even a small face image of size [64x64] pixels, transforms it to a 4096-dimensional problem. Effective and efficient solution to the dimensionality problem has been found using several approaches. One of them is to use eigenspace-based method which reduces the dimensionality of the input face space. PCA is one such technique which transforms face images from higher dimensional space into lower dimensional space by finding appropriate vectors that accurately describe the face images. With increasing dimensionality and non-linearity of face images and demand for a more accurate face recognition system, Support Vector Machine (SVM) was developed and in the recent years has become increasingly popular. Operating essentially as a Pattern Classifier it aims to separate data-points of two different classes using a hyperplane. In our report we use multi-class extensions to SVM, One V/s One (OVO) and One V/s Rest (OVR), for face-recognition. This is implemented using the LibSVM toolbox [6].

Our report discusses both PCA and multi-class SVM in detail by presenting sample face-recognition and reconstruction results and by observing differences in results on varying associated parameters. This report is organized as follows. Section 2 discusses Q1 of the coursework and Section 3 discusses Q2-1 of the coursework. Section 3 presents a conclusion to our report.

2. Eigenfaces and their applications using PCA

Eigenface approach based methods operate by projecting the input face images onto a lower dimensional space wherein associated vectors are extracted to perform face recognition. Literature concerning different types of eigenface based methods exist. The major difference occurs based on choice of projection algorithm, distance metric and classifier employed. In this section of our report we use Principal Component Analysis (PCA), proposed by Turk and Pentland, as projection algorithm, Euclidian distance metric and NN classifier. Fig. 1 describes the various steps involved in face recognition using Sample data with *C* classes. To accomplish our objectives, we use the provided data which has 520 face images, with 10 face images per class. Each face image is [56x46] pixels making it a 2576-dimensional problem. We proceed by dividing face images between Training data set, which will be used to train our model, and Testing data set, which will be used to test the accuracy and various other parameters of our designed model.

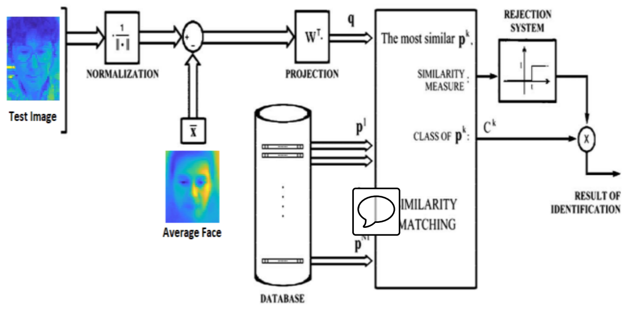

Figure 1: General Process-flow of eigenface based method for face recognition [7].

The data points should be partitioned in such a manner that it allows the model to have moderate variance in estimated parameters by allocating sufficient training data and satisfactory variance in obtained performance statistics by allocating sufficient testing data. The Pareto principle (also known as the 80/20 rule) [7] help us in partitioning the face data. We take 8 images per class as part of testing dataset and keep 2 images per class for testing of our model.

To apply PCA to face images in training set we require eigenfaces, which are eigenvectors of the covariance matrix S. Each face image has been converted to a column vector of size [2576x1]. The equations leading to S are given below:

$$\overline{\mathrm{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathrm{x}_n \tag{1}$$

$$\varphi_n = \mathrm{x}_n - \overline{x} \tag{2}$$

$$A = [\varphi_1, \varphi_2, \varphi_3, \ldots, \varphi_N\,] \tag{3}$$

$$\mathrm{S} = \frac{1}{N} A A^T \tag{4}$$

Where, $\overline{\mathrm{x}}$ is the average face vector and N is number of images in training dataset. The eigenvectors and their corresponding eigenvalue are computed using eigen decomposition of S. We obtain 2576 eigenvectors, each of size [2576x1]. The obtained eigenvectors are arranged in decreasing order based on their corresponding eigenvalues. Largest eigenvector contains maximum information about the facial features. Fig. 2 shows what is contained in these eigenvectors.

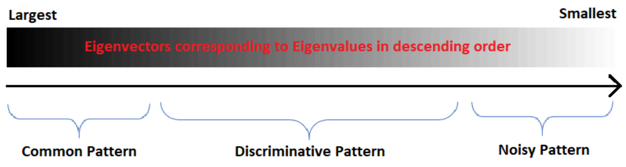

Figure 2: Features represented by eigenvectors.

It is observed that out of 2576 eigenvectors, we have 416 eigenvectors with eigenvalues > 1 and 1443 eigenvectors with non-zero eigenvalues. The number of eigenvectors chosen for face recognition has been chosen using Kaiser Criterion [8] which states to drop all eigenvectors with eigenvalues under 1.0 (corresponding to eigenvectors representing less variations). So, we chose 416 eigenvectors for face recognition. Another technique to compute the covariance matrix S is by following below given equation:

$$\mathrm{S} = \frac{1}{N} A^T A \tag{5}$$

For the sake of simplicity, we will refer to the calculation of S using (4) as Method-1 and using (5) as Method-2. Eigen decomposition of S from (5) gives us just 416 eigenvectors of size (416x1) and its corresponding eigenvalues as opposed to 2576 eigenvectors and eigenvalues for method-1. The eigenvectors and eigenvalues obtained in Method-1 and Method-2 are not identical. On comparing the first 416 eigenvalues from both methods, we observe that there exists a minor difference of the order 10^{-10}, which is illustrated in fig. 3. We represent the magnitude of eigenvalues corresponding to 416 largest eigenvectors, for both methods, in fig. 4.

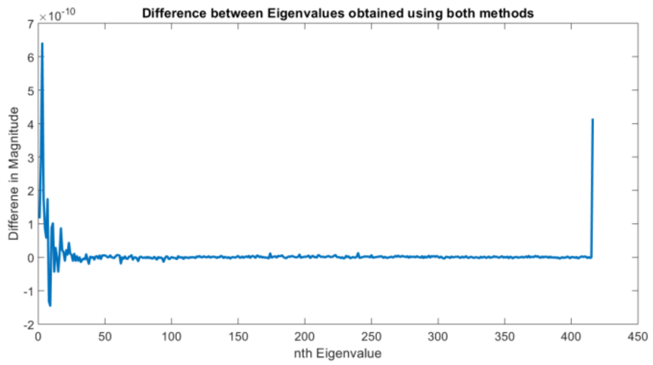

Figure 3: Difference in magnitude of eigenvalues obtained using methods 1 and 2.

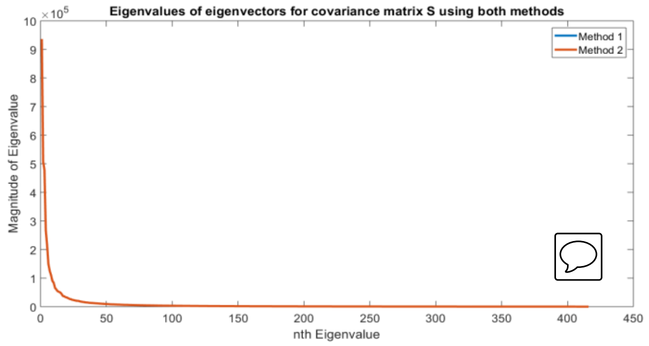

Figure 4: Plot of magnitude of eigenvalues corresponding to 416 largest eigenvectors for both methods.

The average image $\overline{\mathrm{x}}$ is independent of the method chosen for computing S. We represent $\overline{\mathrm{x}}$ in fig. 5. The time taken to compute S and then perform its eigen decomposition for N= 416 is 3.061 secs for Method-1 and is merely 0.018 secs for Method-2. This speed-up in computation occurs because of difference in methods and becomes important and starts to act as a differentiator when we deal with face images of even higher dimensions and with more samples in training set. It is however equally true that size and number of eigenvectors obtained using method-2 are much smaller and lesser than obtained using method-1.

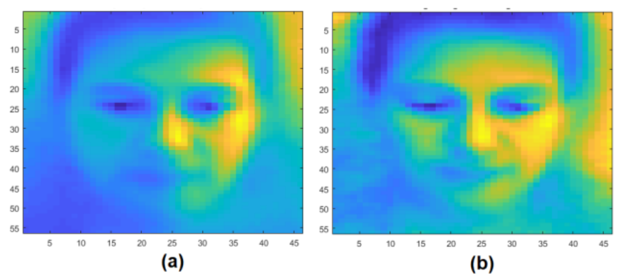

Figure 5: Average image of (a) Training Set (b) Testing Set.

We now attempt to reconstruct face images using eigenvectors, also referred to as bases, and the average face image. The number of bases chosen for face reconstruction defines the goodness of reconstruction. More bases used results in a better face reconstruction as we can incorporate large number of common face patterns as well as the discriminative face patterns for an image class. For $N = 416$, if we vary the number of bases used for reconstructing the face, we observe that the average normalized error per pixel becomes almost zero when we use all available bases with eigenvalues > 1. Fig. 6 represents our discussion in a graphical format. We present the outcome of face reconstruction for two sample images from training dataset and one sample image from testing data set in fig. 7. It is evident that the quality of reconstructed image improves as we start to use more bases for reconstruction. It is imperative to mention in relation to fig.7 that with $N = 416$, we have $M = 104$ face images in the testing dataset and the eigen decomposition of Covariance Matrix of these M images gives us 104 bases with eigenvalues > 1.

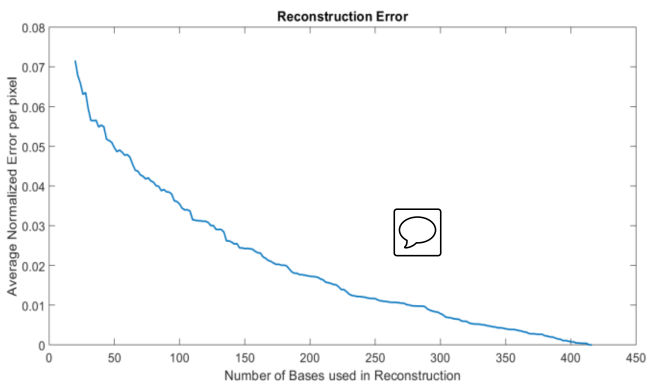

Figure 6: Variation in reconstruction error with the number of bases used for face reconstruction.

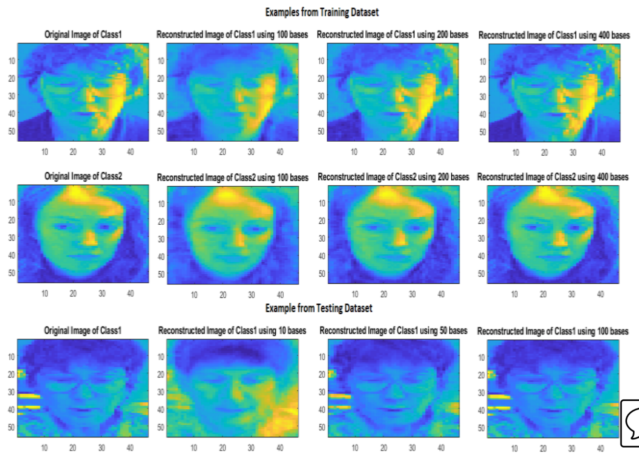

Figure 7: Examples for face reconstruction with varying number of bases used.

Proceeding to the problem of face recognition we make use of PCA as the projection algorithm, NN as the classifier and Euclidian distance metric. The sample success and failure example and the confusion matrix for $N = 416$ and $N = 468$ is given is fig. 8.

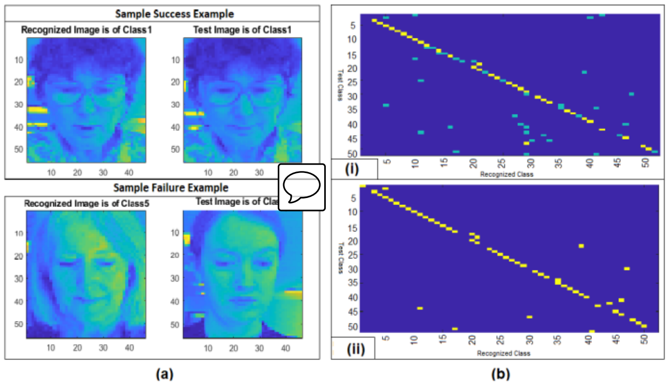

Figure 8: (a) Sample Success and Failure face recognition example, (b) Confusion Matrix for face recognition (i) N=416, bases= 415 (ii) N= 468, bases=465.

On having 90% of face images in training dataset and using all bases with eigenvalues >1 for face recognition we have 75% recognition accuracy, similarly for $N = 416$ and using all bases with eigenvalues >1 for face recognition we have 67% recognition accuracy. Fig. 9 shows the variation in recognition accuracy on changing the number of face images in training dataset and bases used for face reconstruction. We can use Mahalanobis distance metric instead the Euclidian distance metric which increases the recognition accuracy. The total time taken for face recognition is 13.799 secs with N =468, 12.677 secs with N =416 and 12.345 secs with N =364. The time taken appears to be independent of the number of bases used for each N.

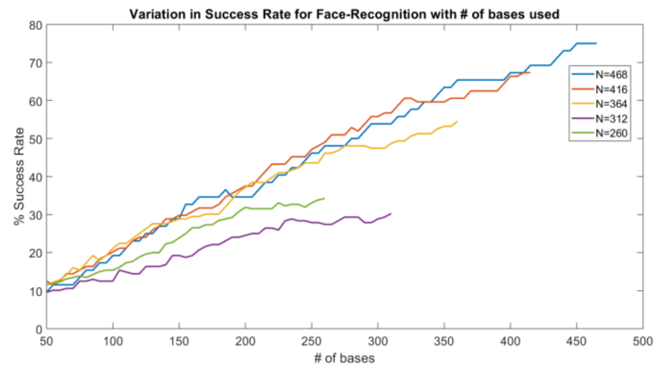

Figure 9: Variation in Recognition Accuracy with $\boldsymbol{N}$ and Bases used for face recognition.

3. Multi-class SVM for Face Recognition

Support Vector Machine (SVM) is a binary classification algorithm which provides sparse solution in high dimensional problems by utilizing labels of the training data. In order to do so we find a hyperplane in a N-dimensional space that separates data in two classes. Biggest advantage of this method is that it provides a sparse solution, because the hyperplane is only depended to the support vectors. This results in manifold reduction in computational power used to classify new unlabeled data as compared to other classification algorithms, such as the NN. The classification function for SVM is the following:

$$y(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b \tag{6}$$

Where, $\boldsymbol{x}$ is a new data point that needs to be classified, **w** and b, are the hyperplane parameters and $\boldsymbol{y}$ is the output of the algorithm. When the data are not linearly separable in the feature space a mapping $\varphi : \boldsymbol{x} \to \varphi(\boldsymbol{x})$ is applied to transform the data into a higher dimensional space, where the data can be separated [9]. This leads to introduction of the kernel function, which enables the formulation of SVM in that second feature space. The kernel function is defined as follows:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^T\varphi(\boldsymbol{x}') \tag{7}$$

Where, $\boldsymbol{x}$ corresponds to a new data point and $\boldsymbol{x}'$ to the training data. Finally, the SVM decision function is the following one:

$$y(\boldsymbol{x}) = \sum_{m=1}^{M} a_m t_m k(\boldsymbol{x}, \boldsymbol{x_m}) + b \tag{8}$$

Where, $a_m \geq 0$ and $\boldsymbol{x_m}$ is the training data and t_n are the labels. Most a_m parameters are equal to zero, and thus sparsity is achieved. Since, the problem discussed in this report has 52 different classes we make use of multi-class extensions of SVM, the code for which can be found in the Appendix to this report. One-Versus-Rest and the One-Versus-One approach extensions have been implemented. The SVM functions from the LibSVM library have been used for training and testing. The provided kernel types are Linear, Polynomial, Radial Basis Function (RBF) and Sigmoid kernel. Before proceeding we scale the feature vectors in testing and training datasets. By using the default coefficients provided by LibSVM for N = 416, we get the following recognition accuracy:

Table I: Comparison of Recognition Accuracy, Testing and Training time with different kernel types with default parameters.

	Linear	Polynomial	RBF	Sigmoid
OVR Using Scaled Data				
Recognition Accuracy (%)	94.3	81.7	83.6	83.6
Training Time (s)	5.98	4.68	4.77	5.16
Testing Time (s)	1.41	1.06	0.95	0.97
OVO Using Scaled Data				
Recognition Accuracy (%)	82.7	78.8	64.4	61.5
Training Time (s)	6.02	5.12	5.09	5.23
Testing Time (s)	1.57	1.21	1.13	0.98

The table given above compares different kernels on the basis of recognition accuracy and testing and training time. A quick look at the values obtained for face recognition by SVM from almost all kernels (for both OVA and OVR) makes it obvious that using SVM gives better performance compared to PCA-NN classification algorithm used in the previous section. The advantage of scaling data before proceeding with face-recognition can be appreciated by comparing recognition accuracy for scaled and unscaled data using RBF kernel. With feature vectors scaled in range [0,1] we have 83.6% recognition accuracy for OVR and 64.4 % for OVO, whereas by proceeding with unscaled data we get recognition accuracy of 1.9% for OVR and 31.7 % for OVO. Similar weird results are obtained for testing and training time parameters. Somewhat similar results are obtained on proceeding with unscaled data using Sigmoid Kernel. This happens because these kernels are highly non-linear and transform the data in an even higher dimensional space. SVM with Linear Kernel provides best recognition accuracy amongst different kind of kernels, this shows that the data samples are linearly separable.

From our initial discussion it is quite evident that in our case, OVR approach has better performance compared to OVO approach. However, it is essential to mention that the OVO approach is much less sensitive to the problems of imbalanced datasets but is much more computationally expensive. In the figures given below, we present a comparison of recognition accuracy obtained using different degree of polynomial kernel for both multi class extensions to LibSVM.

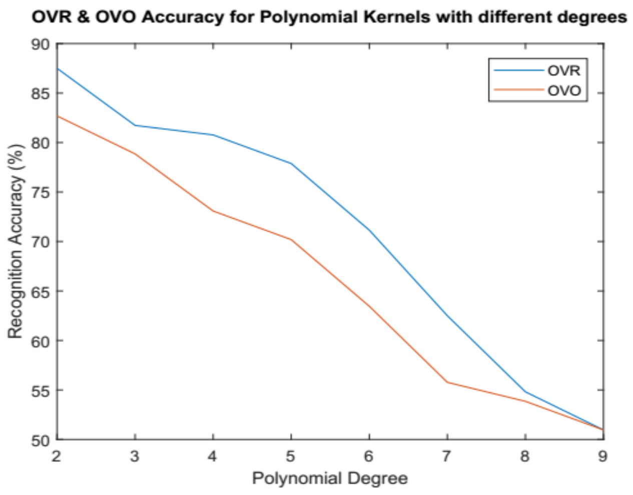

Figure 10: OVR & OVO Recognition Accuracy for Polynomial kernel with variation in degree.

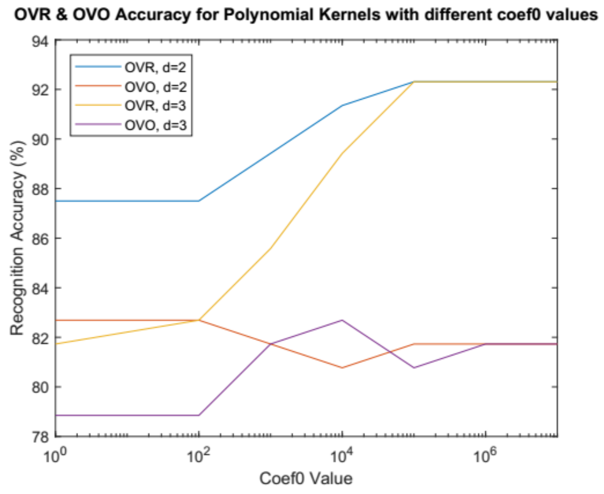

Figure 11: OVR & OVO Accuracy for Polynomial kernel with different coef0 values.

Using the results obtained in fig. 10 and 11, we can obtain best value of degree and coef0 for the polynomial kernel. We now focus our discussion for RBF and Sigmoid kernels. For these kernels we will observe variation in Recognition Accuracy and Testing and Training time by choosing different value of C. Value of parameter C is used during the training phase and it controls how much outliers are taken into account in calculating Support Vectors. It essentially controls the trade-off between training error and the flatness of the solution. A larger value of C results in a lower final training error. But the value of C increased to a larger extent puts us at a risk of losing generalization properties of the classifier, because it will try to fit as best as possible all the training points.

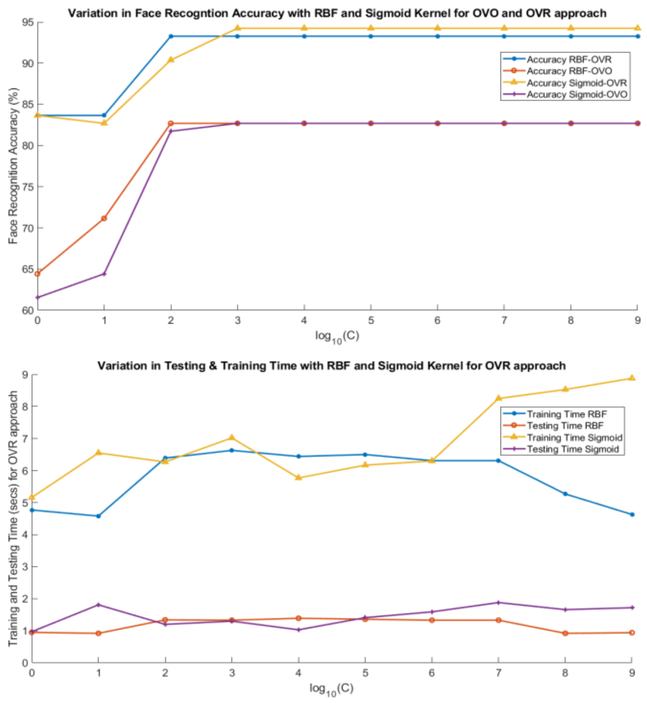

Figure 12: Variation in recognition accuracy and Testing & Training time for RBF and Sigmoid kernel with parameter C.

For each SVM trained, a fraction of the provided feature vectors is chosen as the corresponding support vector. The number of support vectors vary for different SVM's for the OVR and OVO approaches as well as by using different kernel and parameters. These support vectors are the most representative pictures of the training face images. For example, with the OVR approach the first SVM has 41 different support vectors out of the 416 face images. Four out of them are presented in the following figure:

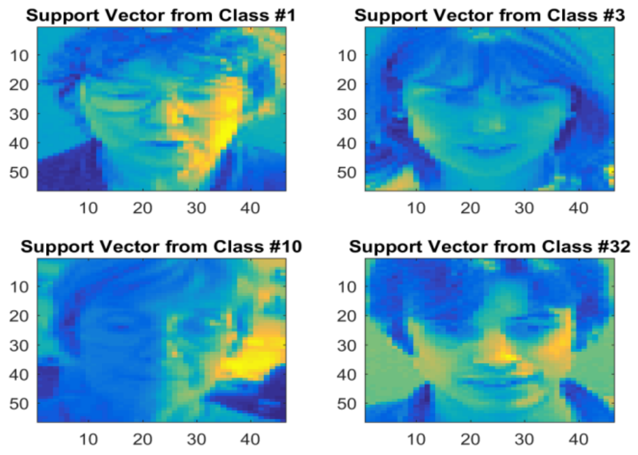
Support Vector from Class #1
Support Vector from Class #3
Support Vector from Class #10
Support Vector from Class #32

Figure 13: Examples of Support Vectors from OVO approach.

Confusion matrix is a very useful graphical method to visualize the result of classification for different testing samples. Confusion matrix for both multi-class SVM extensions i.e. OVO and OVR for N=416 and linear kernel is shown in fig. 14. We also display sample success and failure case for face recognition in fig. 15.

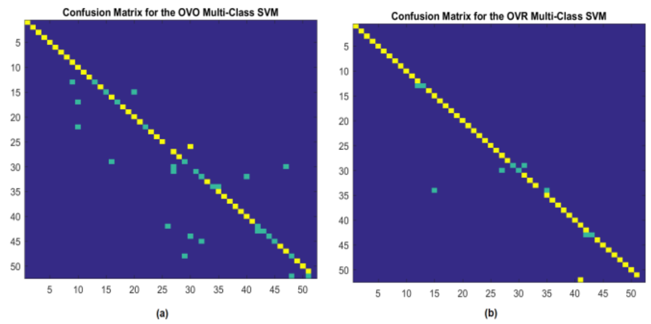

Figure 14: Confusion Matrix for Face Recognition using raw intensity vectors (a) OVO (b) OVR.

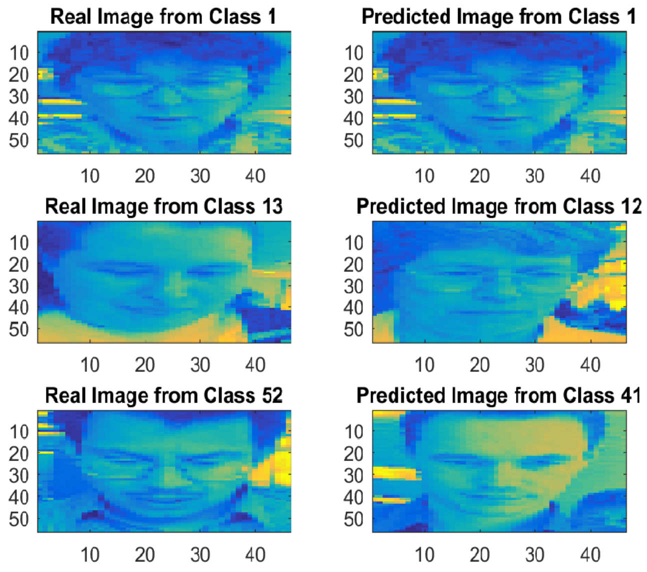
Real Image from Class 1
Predicted Image from Class 1
Real Image from Class 13
Predicted Image from Class 12
Real Image from Class 52
Predicted Image from Class 41

Figure 15: Sample Success and Failure Images.

Feature Vectors being used in training and testing multi-class SVM for face recognition are the raw-intensity vectors (obtained by raster-scanning pixel values of face images). We will now base our discussion by using PCA coefficients as feature vectors. On performing multiple experiments, we get nearly same recognition accuracy and testing and training time by using PCA coefficients as observed in Table I by using raw intensity vectors. Similar performance is obtained because even now the size of face images is same, the only difference being in use of PCA coefficients. The advantage of using PCA coefficients is that we can reduce testing and training time for our SVM model by choosing only top Eigenvectors to train our model. Confusion matrix representing the spread of input test class and output recognized class has been shown in fig. 16 for both OVR and OVO approaches.

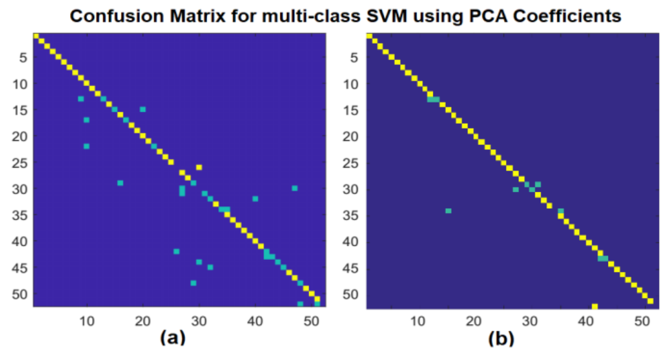

Figure 16: Confusion Matrix for Face Recognition using PCA Coefficients (a) OVO (b) OVR.

For SVM a very important parameter that affects outcome is margin. It is determined by calculating distance of decision surface to the closest data point. Maximizing the margin is a good approach for linearly separable data points, as observed in the figure given below that points near the decision surface represent very uncertain classification decisions.

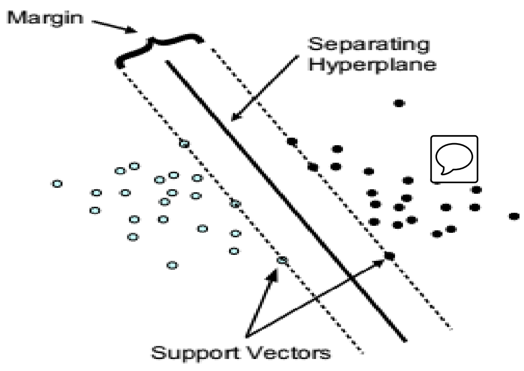

Figure 17: Margin and Support Vectors (Linearly Separable Data) [12].

4. Conclusion

We have discussed two of the most popular face recognition techniques i.e. Eigenfaces using PCA and multi-class SVM in our report. For each technique we have presented and discussed results obtained by varying several associated parameters. It can easily be concluded that for same value of N, multi-class SVM has a better performance than Eigenfaces method using PCA in terms of recognition accuracy (~94% for OVR-SVM with linear kernel compared to ~67% for PCA; N =416) and reduced testing & training time.

References

[1] Chellappa R, Wilson C L, and Sirohey S., Human and machine recognition of faces: A survey, Proceedings of the IEEE, 1995, 83(5), pp. 705-740.

[2] M. Turk, A. Pentland, Eigenfaces for Recognition, Journal of Cognitive Neuroscience, Vol. 3, No. 1, 1991, pp. 71-86.

[3] G. Guo, S.Z. Li, K. Chan, Face Recognition by Support Vector Machines, Proc. of the IEEE International Conference on Automatic Face and Gesture Recognition, 26-30 March 2000, Grenoble, France, pp. 196-201.

[4] T.F. Cootes, C.J. Taylor, Statistical Models of Appearance for Computer Vision, Technical Report, University of Manchester, 125 pages.

[5] A.V. Nefian, M.H. Hayes III, Hidden Markov Models for Face Recognition, Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP'98, Vol. 5, 12-15 May 1998, Seattle, Washington, USA, pp. 2721-2724.

[6] Chih-Chung Chang and Chih-Jen Lin, LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[7] Javier Ruiz-del-Solar and Pablo Navarrete, Eigenspace-Based Face Recognition: A Comparative Study of Different Approaches, IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews, Aug. 2005, Vol. 35, No. 3, pp. 315-325.

[8] Isabelle Guyon, A scaling law for the validation-set training-set size ratio, Technical Report AT&T Bell Laboratories, Berkeley, California, USA, 11 pages.

[9] Norman Cliff, The Eigenvalues-Greater-Than-One Rule and the Reliability of Components, Psychological Bulletin, 1988, Vol. 103, No. 2, pp. 276-279.

[10] T.K Kim, Pattern Recognition Lecture Notes, 2017, Imperial College London.

[11] Wikipedia, Hyperparameters, Date Accessed, 3/12/2017, https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning).

[12] David Meyer, Support Vector Machines, The Interface to libsvm in package e1071, FH Technikum Wien, Austria, Feb. 2017.

APPENDIX

MATLAB© code for Multi Class SVM for face recognition

```
%% Multi-Class SVM
clc; close all; clear all
addpath(genpath('libsvm'))

load('face.mat'); % I -> 520 Labels, X
-> 520 image of 2576 dimensions

p = 8;
q = 10 - p;
[train_data, test_data] =
partition_data(X, p);
N = p/10*520; % number of training
instances
M = (10-p)/10*520; % number of testing
instances

% For Raw intensity vectors as Feature
Vector X

train_data = train_data';
test_data=test_data';
[train_labels, test_labels] =
partition_data(l, p );

% For PCA Coefficients as Feature
vector X
% Uncomment the next 10 lines if we
want to use PCA coefficients

% x_average = 1/N*sum(train_data,2);
% phi = train_data - x_average;
% S1 = 1/N*(phi*phi');
% [U1, lamda1] = eig(S1,'vector');
% omega = phi'*U1;
% phi_test = test_data-x_average;
% omega_test = phi_test'*U1;
% train_data = omega;
% test_data = omega_test;
% [train_labels, test_labels] =
partition_data(l, p );

% Data Normalization
train_data=(train_data -
repmat(min(train_data,[],1),size(train_
data,1),1))*spdiags(1./(max(train_data,
[],1)-
min(train_data,[],1))',0,size(train_dat
a,2),size(train_data,2));
test_data=(test_data -
repmat(min(test_data,[],1),size(test_da
ta,1),1))*spdiags(1./(max(test_data,[],
1)-
min(test_data,[],1))',0,size(test_data,
2),size(test_data,2));

%% One versus the rest
acc=[0 0 0 0 0 0 0 0 0 0];
train_time=[0 0 0 0 0 0 0 0 0 0];
test_time=[0 0 0 0 0 0 0 0 0 0];

for c=0:9
    lab_train= -1*ones(N,1);
    lab_test= -1*ones(M,1);
    predicted_labels = zeros(M,52);
    accuracy = zeros(3,52);
    dec_values = zeros(M,52);
    OVR_nSVs = zeros(52,1);
    t = cputime;

    % Train 52 different SVMs
    for i = 1:52
        lab_train((i-1)*p+1:(i-1)*p+p)
= ones(p,1);
        lab_test((i-1)*(10-p)+1:i*(10-
p)) = ones(10-p,1);
        model_linear{i} =
svmtrain(lab_train, train_data, ['-t 3
-c ',num2str(10^c)]); %linear SVM
        lab_train((i-1)*p+1:(i-1)*p+p)
= -1*ones(p,1);
        lab_test((i-1)*(10-p)+1:i*(10-
p)) = -1*ones(10-p,1);
    end
    e_training = cputime-t;
    train_time(c+1)=e_training;
    t = cputime;

    % Test the Data
    for i = 1:52
        lab_test((i-1)*(10-p)+1:i*(10-
p)) = ones(10-p,1);
        [predict_label_L, accuracy_L,
dec_values_L] = svmpredict(lab_test,
test_data, model_linear{i});
        predicted_labels(:,i) =
predict_label_L;
        accuracy(:,i) = accuracy_L;
        dec_values(:,i) = dec_values_L;
        lab_test((i-1)*(10-p)+1:i*(10-
p)) = -1*ones(10-p,1);
    end

    e_testing = cputime-t;
    test_time(c+1)=e_testing;
    [~, index] = max(dec_values, [],
2);
    errors = 0;
    for j = 1:M
        if index(j) ~= test_labels(j)
            errors = errors + 1;
        end
    end
    OVR_accuracy = (M-errors)/M*100;
    acc(1,c+1)=OVR_accuracy;

end

%% One versus one
accovo=[];
for c=0:9
    training_set = zeros(2*p, 2576);
    training_labels = ones(2*p,1);
    training_labels(p+1:2*p) = -
1*ones(p,1);

    predicted_labels = zeros(M, 52);
    counter = 0;
    for i = 1:52
        training_set(1:p,:) =
train_data((i-1)*p+1:(i-1)*p+p,:);
        for j = i+1:52
            counter = counter + 1;
            training_set(p+1:2*p,:) =
train_data((j-1)*p+1:(j-1)*p+p,:);
            model_linear2{counter} =
svmtrain(training_labels,
training_set,['-t 3 -c
',num2str(10^c)]); %linear SVM
            [predict_label_L,
accuracy_L, dec_values_L] =
svmpredict(lab_test, test_data,
model_linear2{counter});

            for k = 1:M
                if predict_label_L(k)
== 1

predicted_labels(k,i) =
predicted_labels(k,i) + 1;
                else

predicted_labels(k,j) =
predicted_labels(k,j) + 1;
                end
            end
        end
    end
    [~, index] = max(predicted_labels,
[], 2);
    errors = 0;
    for j = 1:M
        if index(j) ~= test_labels(j)
            errors = errors + 1;
        end
    end
    OVO_accuracy = (M-errors)/M*100;
    accovo(c+1)=OVO_accuracy;
end
```