[[1]](#footnote-1)

Performance of PCA and SVM on Human Face Recognition

Ge Gao, Kezhen Liu

*Abstract*— This report analyses on various aspects of performance of face recognition by several classification approaches, including Principal Component Analysis (PCA), Support Vector Machines (SVM) and SVM by using PCA coefficients. The latter two methods achieved 98.08% accuracy in our test, while SVM by using PCA coefficients saves four fifths execution time comparing to SVM by its own. We concluded SVM achieves highest accuracy and, though SVM using PCA coefficients is less computationally expensive, its accuracy can never exceed SVM.

# INTRODUCTION

Nowadays human face recognition has been an important subject in computer vision and machine learning. A few recognition intelligence has been created and proved its effectiveness in the past half century. In this report we will compare and discuss upon various aspects of performance including accuracy, execution time, recognition margin etc. of Principal Component Analysis (PCA) and Support Vector Machine (SVM) on distinguishing human faces.

# Data Partition

Here we have a piece of 520 faces as our raw data, each of which is presented in a 56-by-46 grayscale format. Among those faces, we found that every successive ten faces belong to a single person, referring to Figure 1. To partition the data, for each ten faces belonging to the same person, we took nine of them as training set and the remaining as test set.

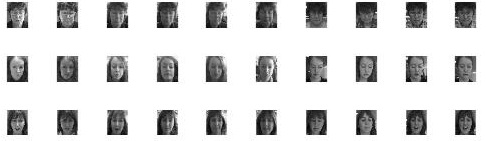


Figure First thirty faces in our raw data

# principal components analysis

We first applied Principal components analysis to our training set. In order to obtain the principal components (eigenfaces) among training set, we calculated the eigenvectors and eigenvalues of the data covariance matrix S, where

,

N is the number of raw data,

**A** is the difference of each face vector from the mean face, as shown in Figure 2.



Figure Visualization of mean face of our data set

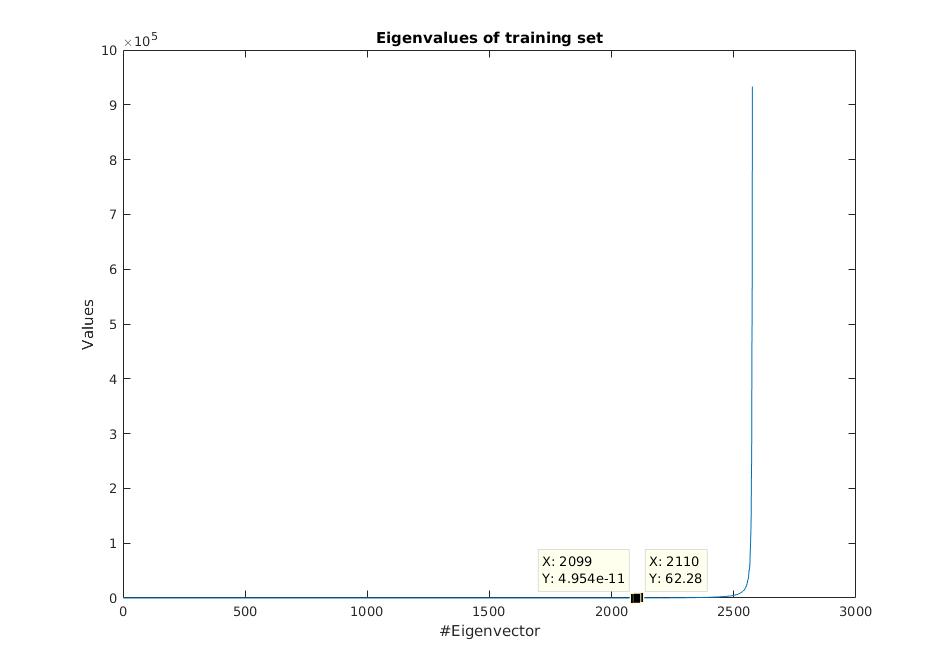
We obtained a total number of 2576 eigenvectors, each of which is associated with an eigenvalue. 

Figure The eigenvalue associated with each eigenvector

Covariance matrix evaluates on the difference between each pair of raw face instance. An eigenvector with large eigenvalue stands for a feature where our data set faces tend to have large variance on, and vice versa. In Figure 3, we found that there is an eigenvalue drop between the 2099th and 2110th eigenvector, from 62.28 down to 4.95e-11, telling us that the faces in our data sets behave no much difference on those features represented by the first 2099 eigenvectors. As a result, to save computational power, we only select the largest 467 eigenvectors as our principal components and discard others.

## Reduce Covariance Matrix Dimension

In the previous section we have a covariance matrix whose dimension equals to the number of attributes and, in our case, equals 2576. Calculating such a large dimension matrix consumes a lot of execution time. Instead, we can construct the covariance matrix with much smaller size while obtaining the same set of principal components.

More specifically, instead of calculating

, ignoring the coefficient, we compute

and the same eigenfaces can be obtained by

,

Where *u* is the same set of principal components and *v* is the eigenvectors of **S**’.

Here we obtained 468 eigenvectors and eigenvalues which are exactly identical as the largest 468 eigenvectors and eigenvalues in last sections. Table 1 summaries the average time consumed on calculating eigenvectors out of ten trials.

Table Average execution time computing eigenfaces

|  |  |
| --- | --- |
| Strategy | Time consumed |
|  | 1.511074 s |
|  | 0.042874 s |

From Table 1 it can be easily deduced that the second strategy is much more timing efficient. However it must be noted that the second strategy still has some inherent limitations that it can only calculate a number of eigenvectors equal to the number of data we have. Consider the case that the number of significant eigenvectors (eigenfaces) exceeds the data count by much, then we are suffering from a loss in number of principal components obtained if the second strategy is applied.

## Face Image Reconstruction

We now applied face Image reconstruction by using PCA. Here we selected two images from training set and one from test set. Figure 4, 5, 6 shows the respective reconstructed image, by using a different number of bases.

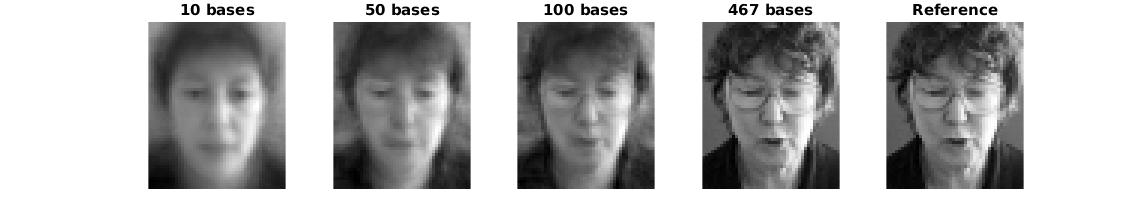


Figure Reconstructed image one using 10, 50, 100, 467 bases

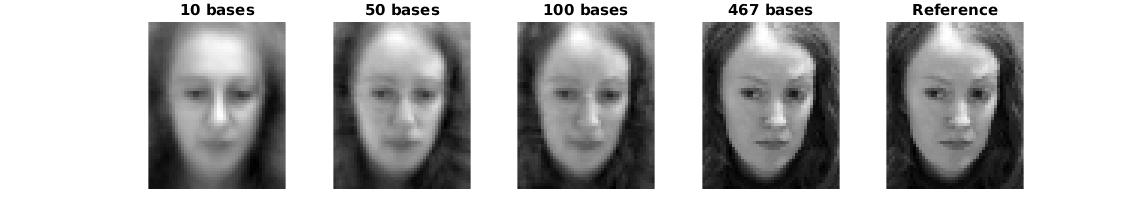


Figure Reconstructed image two using 10, 50, 100, 467 bases

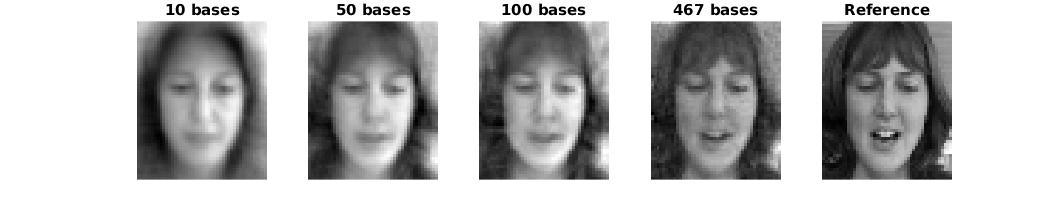


Figure Reconstructed image three using 10, 50, 100, 467 bases, which is an instance from Test Set

Here we saw that at 10 bases, the reconstructed image is barely recognizable, while at 467 bases, the reconstructed image is very much identical to the original image. One another point to note is that in the third set of reconstruction, the regenerated face from test set has more variation with reference image comparing to other reconstructed ones from training set. Table 2 gives a much more intuitive way measuring how different each reconstructed image from reference image, evaluated by the summing absolute difference at each pixel.

Table The sum of absolute difference on each pixel value between reconstructed faces and reference faces

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 10 bases | 50 bases | 100 bases | 467 bases |
| Image 1 | 45778 | 32084 | 24651 | 1.4568e-09 |
| Image 2 | 48881 | 26309 | 18632 | 2.1294e-09 |
| Image 3 | 36743 | 24455 | 21337 | 14196 |

Among those data present in Table 2, in Image 3 there shows a considerable difference between reconstructed face and reference face comparing to Image 1 and 2. The reason is, we constructed test set sample face by projecting into the training set principal components. However we can still seem a tendency that the difference decreases with more and more bases we used during reconstruction.

## PCA-based Face Recognition

In this section, we applied PCA based face recognition to our test set. To reinterpret our experiment settings, our training set contains 467 faces while test set contains 52 faces. Our principal components consists of 467 pre-computed most significant eigenfaces calculated from previous sections. The PCA coefficients are obtained by the projection value of each test face into the principal components. Then the final prediction is made by assigning each test face coefficients to its nearest target class neighbour.

Through our testing, the recognition process takes an average of 0.215614 seconds out of ten trials. Recognition results behave exact coherence out of these trials and have a constant accuracy of 75%. Figure 7 shows the obtained confusion matrix.

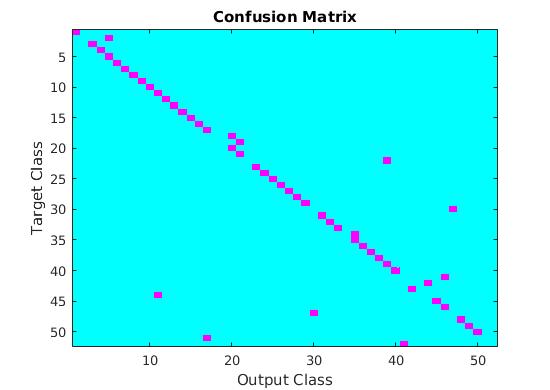


Figure The confusion matrix of PCA prediction results

Figure 8 and 9 show an example of successful and failed recognition case respectively.

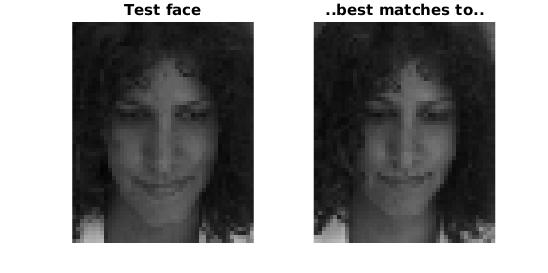


Figure An example of successful prediction



Figure An example of misclassification

## Improve Execution Speed

The same accuracy can be achieved with less execution time by using fewer principal components. As shown in Table 3, the accuracy remains the same until the number of eigenfaces used drops below 50, implying that in our case the most significant 50 eigenfaces do a major role in distinguishing between different faces.

Table The accuracy and execution time by using different number of eigenfaces

|  |  |  |
| --- | --- | --- |
| No of eigenfaces | Accuracy | Execution Time |
| 467 | 0.75 | 0.215614 s |
| 100 | 0.75 | 0.078719 s |
| 50 | 0.75 | 0.066534 s |
| 49 | 0.7308 | 0.066383 s |
| 45 | 0.7308 | 0.065439 s |
| 40 | 0.7115 | 0.064243 s |

# SVM multiclass face recognition

SVM is another classification approach which classifies classes by trying to linearly separate two target classes. A mapping from original feature space to a higher (or infinite) dimension space is usually applied in order to establish a linear classification boundary. There are two approaches to implement a multi-class SVM, one is to train several One-versus-all (OVA) machines and the other is to train one-versus-one (OVO) machines. We will cover both approaches in the following sections.

To apply SVM to our face data, we followed a traditional procedure listed as following throughout the entire testing process [1].

• Transform data into a LIBSVM compatible format

• Scale the data so that data lies in [-1,1]

• Apply RBF, Linear and polynomial models

• Elaborate on model parameters

• Test

## One-versus-all machines

To summaries the principal of OVA machines, we trained a total of 52, equal to the number of target classes, machines by the pixel values from training set and each of them will be applied to the same test set. A classification is made by assigning each test instance to the target class with highest decision values.

Table one summaries the performance of our SVM machine, with default parameters used.

Table Performance of SVM using different kernel types with default parameters

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Accuracy | Training Time | Test Time | Total Time | Minimum Margin |
| Linear | 0.9423 | 4.988 s | 0.435 s | 5.423 s | 0.1522 |
| Polynomial | 0.8654 | 4.866 s | 0.425 s | 5.291 s | 0.0035 |
| RBF | 0.8846 | 4.949 s | 0.416 s | 5.365 s | 0.0035 |

Throughout those three kernel types we used in the above, we will then emphasize more on RBF model. There are a few reasons why we would favour RBF models. First RBF model handles better the non-linear relationship between class labels and attributes. In our case it is easy to deduce that the pixel values must have a non-linear relation with class classification. Moreover, linear model can be considered as a special case of RBF model. The same performance by linear model with penalty parameter C can also be achieved by some RBF model with parameter C and gamma. [1]

### Under-fit and over-fit on penalty parameter C

The penalty constant C measures the cost imposed on misclassified points in training set. With a large penalty value, a hard margin which tries to place all the training samples into its correct target class is generated and thus could possibly cause over-fit on data model. The table below summaries the performance of SVM with different C parameters.

Table Performance of RBF model using different penalty values C

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C of RBF model | Accuracy | Training Time /sec | Test Time /sec | Total Time /sec | Minimum Margin |
| 1 | 0.8846 | 4.949 | 0.416 | 5.365 | 0.0035 |
| 8 | 0.9038 | 5.557 | 0.477 | 6.034 | 4.4653e-04 |
| 32 | 0.9231 | 5.917 | 0.484 | 6.401 | 1.8558e-04 |
| 128 | 0.9615 | 5.928 | 0.481 | 6.409 | 1.3576e-04 |
| 512 | 0.9615 | 5.929 | 0.471 | 6.400 | 1.3576e-04 |

At the beginning, increasing C parameter leads to a higher accuracy, meaning that our first a few model under-fits. With increasing C, the accuracy and minimum margin freeze, telling us that the decision boundary no longer changes, no matter how large the misclassification penalty is.

Same procedure applied on another parameter gamma, where gamma is the free parameter in RBF Gaussian function. Gamma is inversely proportional to the variance and thus small gamma values leads to large variance and more curly decision boundary. The following table demonstrates performance of SVM machine with different gamma values.

Table Performance of RBF model using different gamma value

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Gamma of RBF model | C of RBF model | Accuracy | Total Time /sec | Minimum Margin | Com-ment |
| 0.1 | 32 | 0.5577 | 44.08 | 0.0018 | Under-fit |
| 3.8820e-04 | 32 | 0.9231 | 6.401 | 1.8558e-04 |  |
| 4.8828e-04 | 32 | 0.9615 | 6.056 | 2.0969e-04 | Best |
| 1e-06 | 32 | 0.9038 | 5.351 | 1.0936e-04 | Over-fit |

In this case we observed a clear under-fit, best, over-fit tendency in above table. At first, decreasing gamma value leads to better accuracy and larger margin. Upon reaching best point, where C equals to 32 and gamma equals to 4.8828e-04, rising gamma value squeezes on margin and drops accuracy, showing a typical over-fit behaviour.

### Best parameters

By applying exponential grid search on parameter C and gamma, we found the best pair of parameters is C=32, gamma=4.8828e-04. Note that in Table x, such pair achieved best prediction accuracy among all the trials (96.15%) and exceeds the accuracy by other models (Linear, polynomial).

### Can it still be improved?

Though an accuracy of 96.15% has exceeded the accuracy of PCA based approach by much, it can still be improved without sacrificing much on execution time. We are aware that in an OVA SVM, the training set is highly unbalanced, i.e. out of fifty-two faces there is only one instance classified into the minor class and all the remaining fifty-one instances belong to the other major class. Discrimination on misclassification penalty leads to a softer margin for major class and harder margin for minor class so as to improve the accuracy of minor class classification. The following tables summaries parameters of our best performance model, which achieves a 98% accuracy.

Table The final set of parameters which achieved highest accuracy

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Training Instance Ratio of class | Base penalty parameter C | Gamma Parameter | Weighting of parameter C on minor class | Weighting of parameter C on major class |
| 51:1 | 32 | 4.8828e-04 | 51 | 1 |

Table The best performance SVM achieved

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Accuracy | Training Time | Test Time | Total Time | Minimum Margin |
| 0.9808 | 5.345 s | 0.466 s | 5.811 s | 1.7396e-04 |

Figure 10, 11 and 12 shows the confusion matrix, failure and success case of face recognition respectively. Note that in each case an image of the most significant support vector, i.e. the support vector with highest coefficient value, from output class is attached. Note that we can observe quite much similarity between test face and support vector visualization not only in success case but also in failure case.

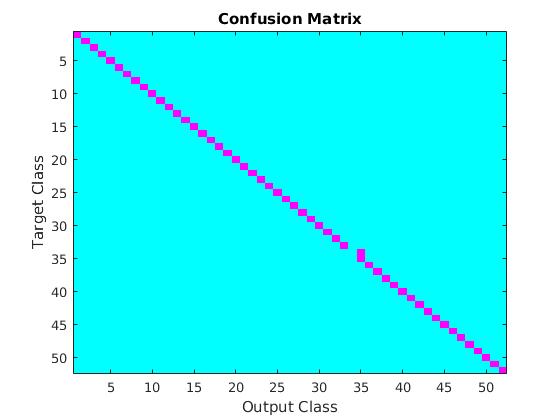


Figure Confusion matrix of best-performed SVM

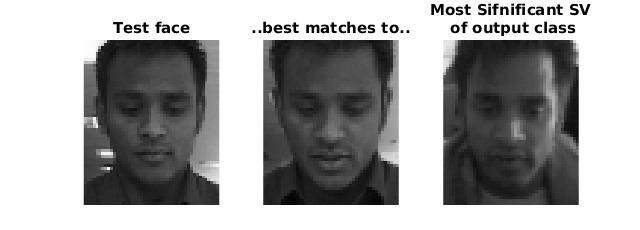


Figure An example of success recognition



Figure An example of misclassification

## One-versus-one machines

Another approach to apply multi-class SVM is to train models of one class versus each one another class and thus we will have a total of N(N-1)/2 models, where N is the number of target classes. Each model makes a prediction on each test face. The final output class is given by the class with the most vote from the entire model set.

The Table 9 summaries the best performance of face recognition by using different type of kernels.

Table Best performance of OVO machines using different kernels

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kernel Type | Parameters with best performance | Accuracy | Minimum Margin | Total time /sec |
| Linear | C=1 | 0.9038 | 1.0087 | 6.473 |
| Polynomial | Gamma=1, coef0=0, degree=2 | 0.9231 | 1.2182e+03 | 6.897 |
| RBF | C=2, gamma= 0.0078125 | 0.8462 | 0.0055 | 8.601 |

In OVO SVM, the linear kernel and polynomial kernel behave far better than RBF kernel in terms of accuracy. The reason is, the number of features (pixels, 2576) exceeds the number of instance (18 instances per training set) by much. One may be aware that if we have a large dimension feature and small number of samples, it will be much easier to linearly separate classes, possibly even without mapping data to a higher dimension space. In our case, without any dimension transformation (Linear Kernel), or with mapping to a slightly higher dimension (Polynomial Kernel with degree 2), the SVM is already able to linearly separate classes quite well, without mapping it to an infinite dimension (RBF model).

## Comparison between OVA machines and OVO machines

Through testing we observed that OVA machines have a better prediction accuracy (98.08%) than OVO machines (92.31%). The reason behind can be deduced qualitatively. Consider, in every OVA machine, each test set instance can be classified to one of its two target classes in a model. Each model makes a valid decision. However in OVO machines, for a single test set instance, only N out of N(N+1)/2 models are able to give a valid classification, while others, instead of making supportive prediction, produce only distraction, as their training class does not contain the test variable. These distractions lower the prediction accuracy.

# Multiclass svm using pca coefficients

Instead of using pixel as features, it is also approachable to use PCA coefficients we computed in the previous sections as features. The advantage is, dimensions of feature vector drops to around one fourths of previous approach, so as support vectors. Smaller support vectors reduce the time consumed for training and testing data. However due to the fact that PCA inevitably drops some of the features of test variable during reducing data dimension, theoretically SVM using PCA coefficients can never exceed the performance of one using pixels and potentially impair the SVM performance.

Table Best performance of SVM using PCA coefficients by different kernels

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Kernel Type | Parameters with best performance | Accuracy | Minimum Margin | Total time /sec |
| Linear | C=16 | 0.9423 | 4.5661e-04 | 0.786 |
| Polynomial | Gamma=0.25, coef0=0,  degree=2 | 0.9423 | 0.0225 | 0.790 |
| RBF | C=32, gamma=0.25 | 0.9808 | 2.4121e-04 | 0.964 |

Referring to Table 10 we found that, by using parameters listed above, SVM using PCA coefficients achieves the same performance as the one using pixel values, with only one fifth execution time consumed. Note that for this RBF model, unlike the previous case, discrimination of data set does NOT further improves the accuracy. We concluded the reasons are, first it can never break its theoretical accuracy limitation as we mentioned earlier, and also PCA eliminates some of the aliasing element by removing those features that does not show much variance among training set, and thus reduce the chance of over-fit.

The only failure case is exactly the same one as we presented in previous section. Please refer to Figure 10 ,11 and 12 for confusion matrix, success and failure cases.

# Conclusion

In this report we have discussed three approaches of face recognition, PCA with NN classifier, SVM and SVM using PCA coefficients as features. PCA with NN classifier gives the worst accuracy while the others have a much higher accuracy (98.08%). Although SVM and SVM using PCA coefficients have the same accuracy in THIS case, it is obvious that SVM is more versatile and robust, but also more computationally demanding, as SVM using PCA coefficients drops some of the raw information and theoretically can never exceed the former.

# References

1. Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin, *A Practical Guide to Support Vector Classification* [*https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf*](https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf)

# appendix

## Raw code

### To plot data faces:

clear;

load('face.mat');

A=zeros(56,46);

%{

for i=1:56

A(i,:)=X(1+(i-1)\*46:i\*46,1);

end

%}

%I(520);

I=zeros(56,46,520);

for j=1:520

for i=1:46

A(:,i)=X(1+(i-1)\*56:i\*56,j);

end

I(:,:,j) = mat2gray(A, [0 256]);

%subplot(26,20,j)

%imshow(I(:,:,j));

end

for j=1:30

subplot(3,10,j)

imshow(I(:,:,j));

end

### Q1\_a

clear;

TRAIN\_NUM = 468;

TEST\_NUM = 52;

EIGVEC\_NUM = 467;

load('face.mat');

trainSet = zeros(2576, TRAIN\_NUM);

testSet = zeros(2576, TEST\_NUM);

testSetIndex = 1;

trainSetIndex = 1;

for i=1:520

res = rem(i,10);

if res == 0

testSet(:,testSetIndex)=X(:,i);

testSetIndex = testSetIndex + 1;

else

trainSet(:,trainSetIndex)=X(:,i);

trainSetIndex = trainSetIndex + 1;

end

end

trainMean = mean(trainSet.').';

%To plot the mean face

temp=zeros(56,46);

for i=1:46

temp(:,i)=trainMean(1+(i-1)\*56:i\*56);

end

meanFaceImg = mat2gray(temp, [0 256]);

imshow(meanFaceImg);

for i = 1:TRAIN\_NUM

trainSet(:,i) = trainSet(:,i)-trainMean;

end

tic

trainCov = (trainSet\*(trainSet.'))./TRAIN\_NUM;

[eig\_vec, eig\_val] = eig(trainCov);

toc

mEigVal = zeros(1,EIGVEC\_NUM);

mEigVec = zeros(2576,EIGVEC\_NUM);

mEigIndex = 1;

for i=1:2576

if eig\_val(i,i) > 1

mEigVal(1,mEigIndex) = eig\_val(i,i);

mEigVec(:,mEigIndex) = eig\_vec(:,i);

mEigIndex = mEigIndex+1;

end

end

%{

%To plot the entire eigenvalues

plotEigValues=zeros(2576,1);

for i=1:2576

plotEigValues(i)=eig\_val(i,i);

end

plot(1:2576,plotEigValues);

title('Eigenvalues of training set');

ylabel('Values');

xlabel('#Eigenvector');

% To show the eigenfaces

Img=zeros(56,46,EIGVEC\_NUM);

A=zeros(56,46);

for j=1:EIGVEC\_NUM

for i=1:46

A(:,i)=mEigVec(1+(i-1)\*56:i\*56,j);

end

Img(:,:,j) = mat2gray(A, [min(min(A)) max(max(A))]);

end

imshow(Img(:,:,467));

%}

### Q1\_b:

clear;

TRAIN\_NUM = 468;

TEST\_NUM = 52;

EIGVEC\_NUM = 467;

load('face.mat');

trainSet = zeros(2576, TRAIN\_NUM);

testSet = zeros(2576, TEST\_NUM);

testSetIndex = 1;

trainSetIndex = 1;

for i=1:520

res = rem(i,10);

if res == 0

testSet(:,testSetIndex)=X(:,i);

testSetIndex = testSetIndex + 1;

else

trainSet(:,trainSetIndex)=X(:,i);

trainSetIndex = trainSetIndex + 1;

end

end

%save('Q1\_b\_DataSet.mat', 'trainSet','testSet');

trainMean = mean(trainSet.').';

for i = 1:TRAIN\_NUM

trainSet(:,i) = trainSet(:,i)-trainMean;

end

tic

S = ((trainSet.')\*trainSet)./TRAIN\_NUM;

[V,D] = eig(S);

mEigVec = trainSet \* V(:,TRAIN\_NUM-EIGVEC\_NUM+1:TRAIN\_NUM);

toc

mD=zeros(TRAIN\_NUM,1);

for i=1:TRAIN\_NUM

mD(i) = D(i,i);

end

for i=1:EIGVEC\_NUM

mEigVec(:,i) = mEigVec(:,i)/norm(mEigVec(:,i));

end

% To show the eigenfaces

Img=zeros(56,46,EIGVEC\_NUM);

A=zeros(56,46);

for j=1:EIGVEC\_NUM

for i=1:46

A(:,i)=mEigVec(1+(i-1)\*56:i\*56,j);

end

Img(:,:,j) = mat2gray(A, [min(min(A)) max(max(A))]);

end

imshow(Img(:,:,EIGVEC\_NUM));

%mEigVec=fliplr(mEigVec);

%save('Q1\_b\_EigVec.mat', 'mEigVec');

%mEigVal=flipud(mD(2:468));

%save('Q1\_b\_EigVal.mat', 'mEigVal');

### Q2\_a:

clear;

TRAIN\_NUM = 468;

TEST\_NUM = 52;

EIGVEC\_NUM = 467;

load('Q1\_b\_EigVec.mat');

load('Q1\_b\_DataSet.mat');

trainMean = mean(trainSet.').';

trainSetDiff = zeros(2576,TRAIN\_NUM);

for i = 1:TRAIN\_NUM

trainSetDiff(:,i) = trainSet(:,i)-trainMean;

end

baseNumIndex = 1;

reconTrainImg = zeros(2576,4,3); %(pixcel, different number of bases, img)

reconTrain = zeros(EIGVEC\_NUM,4,3); %(bases, different number of bases, img)

for baseNum=[10 50 100 EIGVEC\_NUM]

imgIndex=1;

for i=[1 1+int32(TRAIN\_NUM/52) 1+2\*int32(TRAIN\_NUM/52)] % img number in train set

reconTrain(1:baseNum,baseNumIndex,imgIndex) = (trainSetDiff(:,i).' \* mEigVec(:,EIGVEC\_NUM-baseNum+1:EIGVEC\_NUM)).';

imgIndex = imgIndex+1;

end

for i=1:3

%for j=1:baseNum %change 364 here

reconTrainImg(:,baseNumIndex,i) = (reconTrain(1:baseNum,baseNumIndex,i).' \* mEigVec(:,EIGVEC\_NUM-baseNum+1:EIGVEC\_NUM).').';

%end

reconTrainImg(:,baseNumIndex,i) = reconTrainImg(:,baseNumIndex,i)+trainMean;

end

baseNumIndex = baseNumIndex + 1;

end

% To evaluate on the difference between recon and ref image

diff=zeros(4,3);

for i=1:4

for j=1:3

diff(i,j)= sum(abs(reconTrainImg(:,i,j)-...

trainSet(:,(j-1)\*int32(TRAIN\_NUM/52)+1)));

end

end

% To show the plot

IMG\_TO\_PLOT=2; %change 1 here

Img=zeros(56,46,4);

A=zeros(56,46);

basesUsed=[10 50 100 EIGVEC\_NUM];

for j=1:4

for i=1:46

A(:,i)=reconTrainImg(1+(i-1)\*56:i\*56,j,IMG\_TO\_PLOT);

end

Img(:,:,j) = mat2gray(A, [min(min(A)) max(max(A))]);

subplot(1,5,j);

imshow(Img(:,:,j));

hold on;

titleStr=sprintf('%d bases',basesUsed(j));

title(titleStr);

end

subplot(1,5,5);

ref = zeros(56,46);

for i=1:46

ref(:,i)=trainSet(1+(i-1)\*56:i\*56,(IMG\_TO\_PLOT-1)\*int32(TRAIN\_NUM/52)+1);

end

Img1 = mat2gray(ref, [min(min(ref)) max(max(ref))]);

imshow(Img1);

title('Reference');

### Q2\_a on test set:

clear;

TRAIN\_NUM = 468;

TEST\_NUM = 52;

EIGVEC\_NUM = 467;

load('Q1\_b\_EigVec.mat');

load('Q1\_b\_DataSet.mat');

trainMean = mean(trainSet.').';

testSetDiff = zeros(2576,TEST\_NUM);

for i = 1:TEST\_NUM

testSetDiff(:,i) = testSet(:,i)-trainMean;

end

baseNumIndex = 1;

reconTrainImg = zeros(2576,4,3); %(pixcel, different number of bases, img)

reconTrain = zeros(EIGVEC\_NUM,4,3); %(bases, different number of bases, img)

for baseNum=[10 50 100 EIGVEC\_NUM]

imgIndex=1;

for i=[1 1+int32(TEST\_NUM/52) 1+2\*int32(TEST\_NUM/52)] % img number in train set

reconTrain(1:baseNum,baseNumIndex,imgIndex) = (testSetDiff(:,i).' \* mEigVec(:,EIGVEC\_NUM-baseNum+1:EIGVEC\_NUM)).';

imgIndex = imgIndex+1;

end

for i=1:3

%for j=1:baseNum %change 364 here

reconTrainImg(:,baseNumIndex,i) = (reconTrain(1:baseNum,baseNumIndex,i).' \* mEigVec(:,EIGVEC\_NUM-baseNum+1:EIGVEC\_NUM).').';

%end

reconTrainImg(:,baseNumIndex,i) = reconTrainImg(:,baseNumIndex,i)+trainMean;

end

baseNumIndex = baseNumIndex + 1;

end

% To evaluate on the difference between recon and ref image

diff=zeros(4,3);

for i=1:4

for j=1:3

diff(i,j)= sum(abs(reconTrainImg(:,i,j)-...

testSet(:,(j-1)\*int32(TEST\_NUM/52)+1)));

end

end

% To show the plot

IMG\_TO\_PLOT=3; %change 1 here

Img=zeros(56,46,4);

A=zeros(56,46);

basesUsed=[10 50 100 EIGVEC\_NUM];

for j=1:4

for i=1:46

A(:,i)=reconTrainImg(1+(i-1)\*56:i\*56,j,IMG\_TO\_PLOT);

end

Img(:,:,j) = mat2gray(A, [min(min(A)) max(max(A))]);

subplot(1,5,j);

imshow(Img(:,:,j));

hold on;

titleStr=sprintf('%d bases',basesUsed(j));

title(titleStr);

end

subplot(1,5,5);

ref = zeros(56,46);

for i=1:46

ref(:,i)=testSet(1+(i-1)\*56:i\*56,(IMG\_TO\_PLOT-1)\*int32(TEST\_NUM/52)+1);

end

Img1 = mat2gray(ref, [min(min(ref)) max(max(ref))]);

imshow(Img1);

title('Reference');

### Q2\_b:

clear;

TRAIN\_NUM = 468;

TEST\_NUM = 52;

EIGVEC\_NUM = 467;

load('Q1\_b\_EigVec.mat');

load('Q1\_b\_DataSet.mat');

baseNum=EIGVEC\_NUM;

trainMean = mean(trainSet.').';

trainSetDiff = zeros(2576,TRAIN\_NUM);

testSetDiff = zeros(2576,TEST\_NUM);

principleEigvec=zeros(2576,baseNum);

% a=zeros(100,1);

ax=zeros(baseNum,TRAIN\_NUM);

ay=zeros(baseNum,TEST\_NUM);

for i = 1:baseNum

principleEigvec(:,i) = mEigVec(:,i);

end

tic

for i = 1:TRAIN\_NUM

trainSetDiff(:,i) = trainSet(:,i)-trainMean;

ax(:,i)=principleEigvec.'\*trainSetDiff(:,i);

%xbar(:,i)=principleEigvec\*a + trainMean;

end

for i = 1:TEST\_NUM

testSetDiff(:,i) = testSet(:,i)-trainMean;

ay(:,i)=principleEigvec.'\*testSetDiff(:,i);

%ybar(:,i)=principleEigvec\*a + trainMean;

end

d2=zeros(int32(TRAIN\_NUM/52),1);

d1=zeros(52,1);

diff=zeros(TEST\_NUM,1);

outputCl=zeros(TEST\_NUM,1);

for i=1:TEST\_NUM % for each test face

for j=1:52 % for each target class

for k=1:int32(TRAIN\_NUM/52) % for each face in a single target class

d2(k)=norm(ax(:,(j-1)\*int32(TRAIN\_NUM/52)+k)-ay(:,i));

end

%{

if i==4 && j == 4;

disp(' ');

end

%}

d1(j)=min(d2);

end

[diff(i),outputCl(i)]=min(d1);

end

toc

% Construct target class

targetCl=zeros(TEST\_NUM,1);

for i=1:52

for j=1:int32(TEST\_NUM/52)

targetCl((i-1)\*int32(TEST\_NUM/52)+j)=i;

end

end

% Plot confusion matrix

confusion=confusionmat(targetCl,outputCl);

imagesc(confusion);

colormap cool;

title('Confusion Matrix');

xlabel('Output Class');

ylabel('Target Class');

% To calculate accuracy

match=0;

for i=1:TEST\_NUM

if targetCl(i)==outputCl(i)

match=match+1;

end

end

accuray=match/TEST\_NUM

%{

% Plot some specific faces

load('face.mat');

wrong=zeros(56,46);

test=zeros(56,46);

for i=1:46

wrong(:,i)=X(1+(i-1)\*56:i\*56,38);

test(:,i)=X(1+(i-1)\*56:i\*56,40);

end

I(:,:) = mat2gray(test, [0 256]);

subplot(1,2,1);

imshow(I(:,:));

title('Test face');

I(:,:) = mat2gray(wrong, [0 256]);

subplot(1,2,2);

imshow(I(:,:));

title('..best matches to..');

%}

### Q3, pixel input, OVO

clear;

TRAIN\_NUM = 468;

TEST\_NUM = 52;

config

OPTION\_STR='-t 1 -q -g 1 -r 0 -d 2';

load('Q1\_b\_DataSet.mat');

trainSet=(trainSet-128)./128;

%train\_scale\_inst=sparse(((trainSet.')-128)./128);

train\_scale\_lable=double(-1.\*ones(2\*int32(size(trainSet,2)/52),1));

train\_scale\_lable(1:int32(size(trainSet,2)/52))=ones(int32(size(trainSet,2)/52),1);

tic

%Train each model

modelIndex=52\*51/2;

modelLabel2faceID=int32(zeros(2,modelIndex)); %+1,-1

for i=51:-1:1

for j=i+1:52

train\_scale\_inst=sparse(trainSet(:,[(i-1)\*int32(size(trainSet,2)/52)+1:i\*int32(size(trainSet,2)/52) (j-1)\*int32(size(trainSet,2)/52)+1:j\*int32(size(trainSet,2)/52)]).');

model(modelIndex)=svmtrain(train\_scale\_lable, train\_scale\_inst, OPTION\_STR);

modelLabel2faceID(:,modelIndex)=[i,j];

modelIndex=modelIndex-1;

end

end

toc

disp('Start testing...');

tic

test\_scale\_inst=sparse(((testSet.')-128)./128);

predict\_label=zeros(size(testSet,2),52\*51/2); %test\_face#, model#

accuracy=zeros(3,52\*51/2); %[accuracy,mean square error,squared correlation coeff],model#

dec\_values=zeros(size(testSet,2),52\*51/2); %test\_face#,model#

%Test the entire test set by each model

for i=1:52\*51/2 %for each model

test\_scale\_lable=double(-1.\*ones(size(test\_scale\_inst,1),1)); %we don't use predictor label here

%test\_scale\_lable((i-1)\*size(test\_scale\_inst,1)/52+1:i\*size(test\_scale\_inst,1)/52)=ones(size(test\_scale\_inst,1)/52,1);

[predict\_label(:,i), accuracy(:,i), dec\_values(:,i)] = svmpredict(test\_scale\_lable, test\_scale\_inst, model(i),'-q');

end

overall\_predict=ones(size(testSet,2),1);

for i=1:size(testSet,2) %for each test face, count the vote

vote=zeros(52,1);

for j=1:52\*51/2 %for each model, go through its decision value and make vote

if dec\_values(i,j)>=0

vote(modelLabel2faceID(1,j))=vote(modelLabel2faceID(1,j))+1;

else

vote(modelLabel2faceID(2,j))=vote(modelLabel2faceID(2,j))+1;

end

end

[M,indexMaxVote]=max(vote);

overall\_predict(i)=indexMaxVote;

end

toc

% Construct reference class label

targetCl=zeros(TEST\_NUM,1);

for i=1:52

for j=1:int32(TEST\_NUM/52)

targetCl((i-1)\*int32(TEST\_NUM/52)+j)=i;

end

end

% Plot confusion matrix

confusion=confusionmat(targetCl,overall\_predict);

imagesc(confusion);

colormap cool;

title('Confusion Matrix');

xlabel('Output Class');

ylabel('Target Class');

% Calculate on accuracy

match=0;

for i=1:TEST\_NUM

if overall\_predict(i)==targetCl(i)

match=match+1;

end

end

accuracy=match/TEST\_NUM

% To calculate margin

margin=zeros(52,2);

for i=1:52

margin(i,1)=1/norm((model(i).sv\_coef(1:model(i).nSV(1)).')\*model(i).SVs(1:model(i).nSV(1),:));

margin(i,2)=1/norm((model(i).sv\_coef(model(i).nSV(1):model(i).totalSV).')\*model(i).SVs(model(i).nSV(1):model(i).totalSV,:));

end

minMargin=min(min(margin))

%{

% Plot some specific faces

load('face.mat');

wrong=zeros(56,46);

test=zeros(56,46);

svPlot=zeros(56,46);

for i=1:46

wrong(:,i)=X(1+(i-1)\*56:i\*56,379); %342

test(:,i)=X(1+(i-1)\*56:i\*56,380); %340

svPlot(:,i)=model(38).SVs(6,1+(i-1)\*56:i\*56).';

end

I(:,:) = mat2gray(test, [0 256]);

subplot(1,3,1);

imshow(I(:,:));

title('Test face');

I(:,:) = mat2gray(wrong, [0 256]);

subplot(1,3,2);

imshow(I(:,:));

title('..best matches to..');

subplot(1,3,3);

I(:,:) = mat2gray(svPlot, [-1 1]);

imshow(I(:,:));

title({'Most Sifnificant SV', ' of output class'});

%}

### Q3 pixel input, OVA

clear;

TRAIN\_NUM = 468;

TEST\_NUM = 52;

config

OPTION\_STR='-t 2 -q -c 32 -g 0.00048828125 -w1 51 -w-1 1';

% -c 32 -g 0.00048828125 -w1 9 -w-1 1

load('Q1\_b\_DataSet.mat');

train\_scale\_inst=sparse(((trainSet.')-128)./128);

tic

%Train each model

for i=52:-1:1 %for each person

train\_scale\_lable=double(-1.\*ones(size(train\_scale\_inst,1),1));

train\_scale\_lable((i-1)\*size(train\_scale\_inst,1)/52+1:i\*size(train\_scale\_inst,1)/52)=ones(size(train\_scale\_inst,1)/52,1);

model(i)=svmtrain(train\_scale\_lable, train\_scale\_inst, OPTION\_STR);

end

toc

disp('Start testing...');

tic

test\_scale\_inst=sparse(((testSet.')-128)./128);

predict\_label=zeros(size(testSet,2),52); %test\_face#, model#

accuracy=zeros(3,52); %[accuracy,mean square error,squared correlation coeff],model#

dec\_values=zeros(size(testSet,2),52); %test\_face#,model#

%Test the entire test set by each model

for i=1:52 %for each person

test\_scale\_lable=double(-1.\*ones(size(test\_scale\_inst,1),1));

test\_scale\_lable((i-1)\*size(test\_scale\_inst,1)/52+1:i\*size(test\_scale\_inst,1)/52)=ones(size(test\_scale\_inst,1)/52,1);

[predict\_label(:,i), accuracy(:,i), dec\_values(:,i)] = svmpredict(test\_scale\_lable, test\_scale\_inst, model(i),'-q');

end

overall\_predict=ones(size(dec\_values,1),1);

for i=1:size(dec\_values,1) %for each test face

localMaxDec=dec\_values(i,1);

for j=1:52 %for each model, go through the decision values of that face and find the max

if dec\_values(i,j)>localMaxDec

localMaxDec=dec\_values(i,j);

overall\_predict(i)=j;

end

end

end

toc

% Construct reference class label

targetCl=zeros(TEST\_NUM,1);

for i=1:52

for j=1:int32(TEST\_NUM/52)

targetCl((i-1)\*int32(TEST\_NUM/52)+j)=i;

end

end

% Plot confusion matrix

confusion=confusionmat(targetCl,overall\_predict);

imagesc(confusion);

colormap cool;

title('Confusion Matrix');

xlabel('Output Class');

ylabel('Target Class');

% Calculate on accuracy

match=0;

for i=1:TEST\_NUM

if overall\_predict(i)==targetCl(i)

match=match+1;

end

end

accuracy=match/TEST\_NUM

% To calculate margin

margin=zeros(52,2);

for i=1:52

margin(i,1)=1/norm((model(i).sv\_coef(1:model(i).nSV(1)).')\*model(i).SVs(1:model(i).nSV(1),:));

margin(i,2)=1/norm((model(i).sv\_coef(model(i).nSV(1):model(i).totalSV).')\*model(i).SVs(model(i).nSV(1):model(i).totalSV,:));

end

minMargin=min(min(margin))

% Plot some specific faces

load('face.mat');

wrong=zeros(56,46);

test=zeros(56,46);

svPlot=zeros(56,46);

for i=1:46

wrong(:,i)=X(1+(i-1)\*56:i\*56,379); %342

test(:,i)=X(1+(i-1)\*56:i\*56,380); %340

svPlot(:,i)=model(38).SVs(6,1+(i-1)\*56:i\*56).';

end

I(:,:) = mat2gray(test, [0 256]);

subplot(1,3,1);

imshow(I(:,:));

title('Test face');

I(:,:) = mat2gray(wrong, [0 256]);

subplot(1,3,2);

imshow(I(:,:));

title('..best matches to..');

subplot(1,3,3);

I(:,:) = mat2gray(svPlot, [-1 1]);

imshow(I(:,:));

title({'Most Sifnificant SV', ' of output class'});

### Q3 to generate PCA coeff

clear;

TRAIN\_NUM = 468;

TEST\_NUM = 52;

EIGVEC\_NUM = 467;

load('Q1\_b\_EigVec.mat');

load('Q1\_b\_DataSet.mat');

baseNum=EIGVEC\_NUM;

trainMean = mean(trainSet.').';

trainSetDiff = zeros(2576,TRAIN\_NUM);

testSetDiff = zeros(2576,TEST\_NUM);

trainEigVal=zeros(baseNum,TRAIN\_NUM);

testEigVal=zeros(baseNum,TEST\_NUM);

confusion=zeros(52,52,int32(TEST\_NUM/52));

for i = 1:TRAIN\_NUM

trainSetDiff(:,i) = trainSet(:,i)-trainMean;

trainEigVal(:,i)=mEigVec.'\*trainSetDiff(:,i);

end

for i = 1:TEST\_NUM

testSetDiff(:,i) = testSet(:,i)-trainMean;

testEigVal(:,i)=mEigVec.'\*testSetDiff(:,i);

end

save('Q3\_PCA\_coeff','trainEigVal', 'testEigVal');

### Q3 PCA input, OVO

clear;

config

OPTION\_STR='-t 0';

load('Q3\_PCA\_coeff.mat');

rangeTrainEigVal=max(max(trainEigVal))-min(min(trainEigVal));

rangeTestEigVal=max(max(testEigVal))-min(min(testEigVal));

range=max([rangeTrainEigVal rangeTestEigVal]);

trainEigVal=(trainEigVal-range/2)./(range/2);

%train\_scale\_inst=sparse(((trainSet.')-128)./128);

train\_scale\_lable=double(-1.\*ones(2\*int32(size(trainEigVal,2)/52),1));

train\_scale\_lable(1:int32(size(trainEigVal,2)/52))=ones(int32(size(trainEigVal,2)/52),1);

%Train each model

modelIndex=52\*51/2;

modelLabel2faceID=int32(zeros(2,modelIndex)); %+1,-1

for i=51:-1:1

for j=i+1:52

train\_scale\_inst=sparse(trainEigVal(:,[(i-1)\*int32(size(trainEigVal,2)/52)+...

1:i\*int32(size(trainEigVal,2)/52) ...

(j-1)\*int32(size(trainEigVal,2)/52)+1:j\*int32(size(trainEigVal,2)/52)]).');

model(modelIndex)=svmtrain(train\_scale\_lable, train\_scale\_inst, OPTION\_STR);

modelLabel2faceID(:,modelIndex)=[i,j];

modelIndex=modelIndex-1;

end

end

disp('Start testing...');

test\_scale\_inst=sparse(((testEigVal.')-range/2)./(range/2));

predict\_label=zeros(size(testEigVal,2),52\*51/2); %test\_face#, model#

accuracy=zeros(3,52\*51/2); %[accuracy,mean square error,squared correlation coeff],model#

dec\_values=zeros(size(testEigVal,2),52\*51/2); %test\_face#,model#

%Test the entire test set by each model

for i=1:52\*51/2 %for each model

test\_scale\_lable=double(-1.\*ones(size(test\_scale\_inst,1),1)); %we don't use predictor label here

%test\_scale\_lable((i-1)\*size(test\_scale\_inst,1)/52+1:i\*size(test\_scale\_inst,1)/52)=ones(size(test\_scale\_inst,1)/52,1);

[predict\_label(:,i), accuracy(:,i), dec\_values(:,i)] = svmpredict(test\_scale\_lable, test\_scale\_inst, model(i));

end

overall\_predict=ones(size(testEigVal,2),1);

for i=1:size(testEigVal,2) %for each test face, count the vote

vote=zeros(52,1);

for j=1:52\*51/2 %for each model, go through its decision value and make vote

if dec\_values(i,j)>=0

vote(modelLabel2faceID(1,j))=vote(modelLabel2faceID(1,j))+1;

else

vote(modelLabel2faceID(2,j))=vote(modelLabel2faceID(2,j))+1;

end

end

[M,indexMaxVote]=max(vote);

overall\_predict(i)=indexMaxVote;

end

### Q3 PCA input OVA

clear;

TRAIN\_NUM = 468;

TEST\_NUM = 52;

config

OPTION\_STR='-t 1 -q -r 0 -g 0.25 -d 2';

%-t 2 -q -c 32 -g 0.25

load('Q3\_PCA\_coeff.mat');

rangeTrainEigVal=max(max(trainEigVal))-min(min(trainEigVal));

rangeTestEigVal=max(max(testEigVal))-min(min(testEigVal));

range=max([rangeTrainEigVal rangeTestEigVal]);

train\_scale\_inst=sparse(((trainEigVal.')-range/2)./(range/2));

tic

%Train each model

for i=52:-1:1 %for each person

train\_scale\_lable=double(-1.\*ones(size(train\_scale\_inst,1),1));

train\_scale\_lable((i-1)\*size(train\_scale\_inst,1)/52+1:i\*size(train\_scale\_inst,1)/52)=ones(size(train\_scale\_inst,1)/52,1);

model(i)=svmtrain(train\_scale\_lable, train\_scale\_inst, OPTION\_STR);

end

toc

disp('Start testing...');

tic

test\_scale\_inst=sparse(((testEigVal.')-range/2)./(range/2));

predict\_label=zeros(size(testEigVal,2),52); %test\_face#, model#

accuracy=zeros(3,52); %[accuracy,mean square error,squared correlation coeff],model#

dec\_values=zeros(size(testEigVal,2),52); %test\_face#,model#

%Test the entire test set by each model

for i=1:52 %for each person

test\_scale\_lable=double(-1.\*ones(size(test\_scale\_inst,1),1));

test\_scale\_lable((i-1)\*size(test\_scale\_inst,1)/52+1:i\*size(test\_scale\_inst,1)/52)=ones(size(test\_scale\_inst,1)/52,1);

[predict\_label(:,i), accuracy(:,i), dec\_values(:,i)] = svmpredict(test\_scale\_lable, test\_scale\_inst, model(i),'-q');

end

overall\_predict=ones(size(dec\_values,1),1);

for i=1:size(dec\_values,1) %for each test face

localMaxDec=dec\_values(i,1);

for j=1:52 %for each model, go through the decision values of that face and find the max

if dec\_values(i,j)>localMaxDec

localMaxDec=dec\_values(i,j);

overall\_predict(i)=j;

end

end

end

toc

% Construct reference class label

targetCl=zeros(TEST\_NUM,1);

for i=1:52

for j=1:int32(TEST\_NUM/52)

targetCl((i-1)\*int32(TEST\_NUM/52)+j)=i;

end

end

% Plot confusion matrix

confusion=confusionmat(targetCl,overall\_predict);

imagesc(confusion);

colormap cool;

title('Confusion Matrix');

xlabel('Output Class');

ylabel('Target Class');

% Calculate on accuracy

match=0;

for i=1:TEST\_NUM

if overall\_predict(i)==targetCl(i)

match=match+1;

end

end

accuracy=match/TEST\_NUM

% To calculate margin

margin=zeros(52,2);

for i=1:52

margin(i,1)=1/norm((model(i).sv\_coef(1:model(i).nSV(1)).')\*model(i).SVs(1:model(i).nSV(1),:));

margin(i,2)=1/norm((model(i).sv\_coef(model(i).nSV(1):model(i).totalSV).')\*model(i).SVs(model(i).nSV(1):model(i).totalSV,:));

end

minMargin=min(min(margin))

%{

% Plot some specific faces

load('face.mat');

wrong=zeros(56,46);

test=zeros(56,46);

svPlot=zeros(56,46);

for i=1:46

wrong(:,i)=X(1+(i-1)\*56:i\*56,379); %342

test(:,i)=X(1+(i-1)\*56:i\*56,380); %340

svPlot(:,i)=model(38).SVs(6,1+(i-1)\*56:i\*56).';

end

I(:,:) = mat2gray(test, [0 256]);

subplot(1,3,1);

imshow(I(:,:));

title('Test face');

I(:,:) = mat2gray(wrong, [0 256]);

subplot(1,3,2);

imshow(I(:,:));

title('..best matches to..');

subplot(1,3,3);

I(:,:) = mat2gray(svPlot, [-1 1]);

imshow(I(:,:));

title({'Most Sifnificant SV', ' of output class'});

%}

1. Ge Gao

   Kezhen Liu, with CID 00811737, email Kezhen.liu13@imperial.ac.uk [↑](#footnote-ref-1)