**Pattern Recognition**

**Q1) Eigenfaces**   
  
part a ) Calculating PCA using computationally intensive method for covariance matrix.  
**Algorithm** -

1. Parititon sample dataset into training and validation set.
2. Calculate mean image.
3. Calculate mean centred data.
4. Calculate covariance matrix S.
5. Use Matlab svd(S) to get all eigenvectors and eigenvalues.
6. Filter non-zero eigenvalues and corresponding eigenvectors.
7. Choose new reduced dimension (k) based on a heuristic.
8. Finally calculate “W” matrix of containing the chosen eigenvectors.
9. To confirm results, reconstruction error is calculated using 2 methods using -
   1. Summation of eigenvalues.
   2. Euclidean distance between reconstructed training samples and   
      original training samples

**Note**

* The eigenvalues returned by Matlab svd() function are already in descending order by value.
* The eigenvectors returned by matlab are already normalized.

1. Procedure for partitioning dataset  
    <https://www.researchgate.net/post/What_is_the_best_way_to_divide_a_dataset_into_training_and_test_sets>

Used Matlab randperm( ) command to generate random, non-overlapping indices of samples to choose training and testing set. It is made sure that the training set contains sample of 52 classes.

Using matlab command mean() *along the rows ( since each coloumn is a training sample )*-   
mu = mean(train\_set,2);



Figure 2 : mean\_images of training\_set

Using Matlab svd() command we obtain all the eigenvalues and eigenvectors -

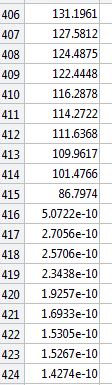


Figure : all eigenvalues. Notice how they shrink to a very small value after total\_samples(416)-1 = 415

Clearly the eigenvectors form an orthonormal basis. This is verified by the following Matlab commands –

a) **Orthogonality** of eigenvectors ( can pick any 2 eigenvectors, here we pick 300th and 104th eigenvector)   
 sum(all\_eigenvectors(:,300).\*all\_eigenvectors(:,104)) = -7.3509e-17

b) **Orthonormality** of eigenvectors ( can pick any eigenvector, here we pick 211th eigenvector )   
sum(all\_eigenvectors(:,211).\*all\_eigenvectors(:,211)) = 1.0000

This means that each eigenvector forms an axis/basis vector for a new coordinate system.

As seen from Figure 1 above, all eigenvalues after N-1 **WHY** ( where N = total samples ) are very small in value. Hence these and the corresponding eigenvectors are filtered using simple Matlab commands -

N = size(X\_centred,2);

nz\_eigenvalues = all\_eigenvalues(1:N-1);

nz\_eigenvectors = all\_eigenvectors(:,1:N-1);

**In the remaining discussion, eigenvalues refers to the non-zero eigenvalues as shown above.**

Now we choose to reduce the dimension of this new coordinate system by reducing the number of eigenvectors (using a heursitic explained below). This reduced number of eigenvectors is the important result of PCA.

Heuristic for choosing number (“k”) of eigenvectors –

The value of “k” can range from 1 to N-1, where choosing k = N-1 would result in no dimensionality reduction.

Hence a ratio is defined : reconstruction\_error/variance

Where reconstruction error = sum(eigvalues(k+1:end));  
and variance = sum(eigenvalues); **WHY**   
  
It can be observed that if “k” increases, the ratio decreases ( since numerator reconstruction error decreases and denominator variance increases ).

Based on this observation, a loop is setup with a loop index “k”.

**Termination of loop/finding k** -  
We define a threshold such that if the ratio <= threshold in a given loop iteration k\*, the loop is terminated and the value of k\* is chosen as the reduced dimension.

For example – if threshold = 0.01. This means that the reconstruction error to variance ratio is 1/100 and hence a common method of interpreting this is to consider it as “99% variance is retained”.

**Confirming validity of results :**Finding Reconstruction error using 2 methods –

Method 1) - Using eigenvalues  
 Reconstruction error = summation of last N-k eigenvalues of covariance matrix.

Method 2) - Squared Euclidean distance between original training samples and reconstructed samples.

Reconstructed training samples = PCA \* W’ + mean\_image

Clearly, both methods give the same reconstruction error for a given “k” -   
  
Reconstruction error found using eigenvalues method = 44046.098979

The reconstruction error found using squared euclidean distance method = 44046.098979

**part b )** Calculating PCA using computationally fast method for covariance matrix.  
  
  
Let eigenvectors of S calculated through previous  
  
Let us consider eigenvectors (v\_i) of A’A.  
(A’A)v\_i = lambda \* v\_i

Multiplying by A on both sides and using associativity of Matrix multiplication ,   
  
(AA’)Av\_i = lambda \* Av\_i  
SAv\_i = lambda \* Av\_i

Hence Av\_i is eigenvector of S.   
Hence normalized(u\_i) = A v\_i   
  
u\_i or A\*v\_i needs to be normalized since eigenvectors must always represent a set of orthonormal basis vectors.

**Q1) Application Eigenfaces**

**Part a)** Reconstructing face images

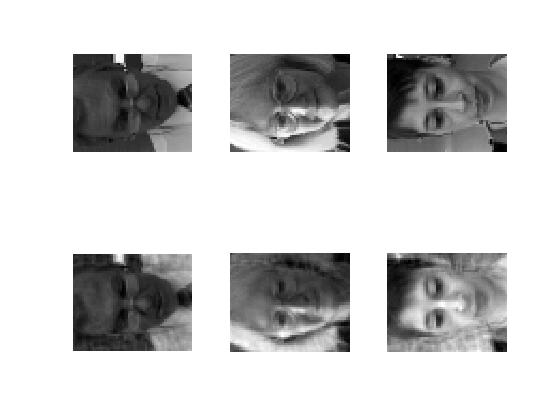


Figure : Comparison of 3 images from original training set vs the reconstructed images.  
The upper row contains face images from training set and lower row contains reconstructed images with **k = 124** ( threshold = 0.05 )

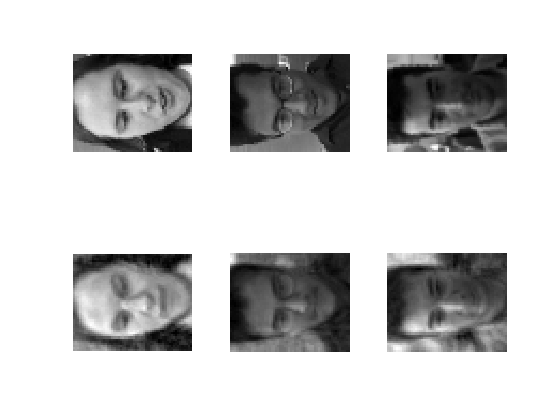
****

Figure : Comparison of 3 images from test set and reconstructed test set.The upper row contains face images from test set and lower row contains reconstructed images with **k = 124** ( threshold =0.05)

**Part b)**

Recognition Accuracy -

Example success and failure cases

Confusion matrix

time/memory

**Q2) Multi-class SVM  
  
Introduction**

SVM is inherently a binary classifier. In order to do multi-class classification, 2 of either approaches can be taken – one-vs-one or one-vs-all. Both approaches are methods of reducing a multi-class classification problem to a smaller binary classification problem.  
  
Consider an example scenario of training labels consisting of 52 classes.  
The **one-vs-all** approach would train 52 classifiers ( one for each class ) ***for the entire training set***,   
  
The **one-vs-one** classifier would train 52C2 = 1326 classifiers, but the training set for each classifier only consists of a subset of the training data. The subset depends on which pair of classes the current classifier is being trained for. **This is better understood by looking at the Matlab code.**

**Setting of parameters –**

Best params found for one-vs-all: C = 32, sigma = 128. Accuracy = 86.5385%.  
  
Best params found for one-vs-one : C = 32, sigma = 128. Accuracy = 86.5385

**Recognition Accuracy**

One-vs-all:

One-vs-one:

**Confusion Matrix –**

**Appendix**

Complete Matlab Code for Multi-class SVM training and testing.

function []= multiClassSVMTrainer(training\_samples,training\_labels,test\_samples,test\_labels)

t = unique(training\_labels);

total\_classes = size(t,1); % <=52(since depends on partition) in our case.

% ======= One-vs-all (Training Phase)=============

SVMModels = cell(total\_classes,1); % some value <=52 depending on training set partition.

% training "t" classifiers

for index = 1: total\_classes

% imp, cannot assume input set has all classes in a given range.

% Hence safer to use an index to find out class.

currentClass = t(index);

% for each class, convert all y labels of current class

% to +1, and remaining labels to -1

X = training\_samples ;

Y = training\_labels;

Y(training\_labels == currentClass)= 1;

Y(training\_labels ~= currentClass)= 0;

SVMModels{index} = fitcsvm(X,Y,'ClassNames',[false true],'Standardize',true,...

'KernelFunction','rbf','BoxConstraint',1);

end

% ======= One-vs-all (Testing Phase)============

myTestingData = training\_samples;

myTestingLabels = training\_labels;

% Matrix storing scores (for positive class) by all classifiers for

% each test sample.

Scores = zeros(size(myTestingData,1),total\_classes);

% For each trained SVM, get the score for the "positive" class.

for index = 1: total\_classes

[~, score] = predict(SVMModels{index},myTestingData);

Scores(:,index) = score(:,2); % Second column contains positive-class scores

end

% Most imp, use the max score (max of all classifiers)

% to obtain indices of the predicted classes.

[~, maxScoreIndices] = max (Scores, [],2);

predictorLabels = t(maxScoreIndices);

acc = findAccuracy (predictorLabels, myTestingLabels);

fprintf ('Accuracy = %f %% \n', acc);

%%

% ======= One-vs-one (Training Phase)===========

% 1) For each pair of classes, first extract labels

% and corresponding samples.

% 2) Then convert all y labels of first class

% to +1, and labels of other class to -1

% The two loops below iterate over all possible combinations

% ( in an efficient manner, similar to bubble sort )

% and training 52C2 = 1326 SVM classifiers ( which is a lot )

for i = 1: total\_classes

for j = i+1:total\_classes

% unlike OvA method, we need to filter the training labels

% and samples based on class pair being considered.

indices = training\_labels == i | training\_labels == j;

training\_samples12 = training\_samples (indices,:) ;

training\_labels12 = training\_labels(indices) ;

% Use cross validation here to choose bestC and bestSigma

bestC = 1;

SVMModels{index} = fitcsvm (training\_samples12, training\_labels12,'ClassNames’, [false true],'Standardize',true,...

'KernelFunction','rbf','BoxConstraint’, bestC);

end

end

end

% Compares prediction labels with given labels for a test dataset

% and calculates accuracy as a percentage value.

function [acc] = findAccuracy (predictorLabels, myTestingLabels)

sum=0;

% Find accuracy of predictor against test labels

for i = 1:size(myTestingLabels,1)

if(predictorLabels(i) == myTestingLabels(i))

sum = sum + 1;

end

end

acc = sum/size(myTestingLabels,1)\*100;

end

**References**

1. <https://www.cs.utah.edu/~lifeifei/cis5930/lecture12-a.pdf>
2. <https://www.coursera.org/learn/machine-learning/lecture/S1bq1/choosing-the-number-of-principal-components>
3. <https://stats.stackexchange.com/questions/229092/how-to-reverse-pca-and-reconstruct-original-variables-from-several-principal-com>