Ozan, Ran
January 22, 2014

# Applied Mathematics

Home work 4
Face Recognition using PCA

## Chapter 1: Introduction

In this project, we implemented a face recognition system by using principal component analysis, which is known as $PCA$. $PCA$ method provide a mathematical way to reduce the dimension of problem.

Since the most elements of a facial image are highly correlated, it is better to extract a set of interesting and discriminative feature of a facial image. Mathematically speaking, we transform the correlated data to independent data. To implement the transform, we employed some linear algebra method such as SVD ($Chapter3$). The main idea is to obtain $Eigenfaces$ that every face can be regard as a linear combination of these eigenfaces ($Chapter4$). Then the face recognition problem convert to a mathematic problem: what is the linear combination of a face? In other words, it simplify a problem from 2D to 1D.

## Chapter 2: Normalization

### How to normalized images

Normalization actually is a projection process. Because we detected the face images with different angle, we hope that we can use a same coordinate system for different angle faces. This coordinate system is a $64 \times 64$ window which we predetermined five positions as facial features' location. These five locations is $\overline{F}$. Once we have $\overline{F}$, we can obtain affine transformation coefficients for each image as following figure shown.

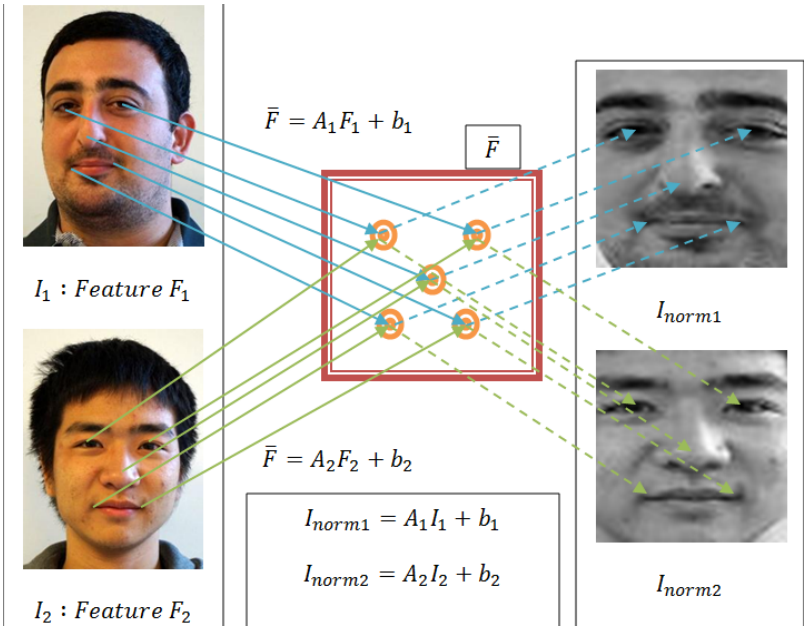


Figure 1:

In $Figure1$, $A_i$ and $b_i$ are affine transformation matrix. Where $A$ and $b$ are given by:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

### How to compute A and b

For each feature, the affine transformation is given by:

$$f_i^P = Af_i + b$$

Where $f_i$ is the feature location in original image and $f_i^P$ is the position in normalization image. Since we already have $f_i$ and $f_i^P$ is also predetermined, let $f = \begin{bmatrix} x \\ y \end{bmatrix}$, then the transformation can be written into:

$$x_i^P = a_{11}x_i + a_{12}y_i + b_1$$
$$y_i^P = a_{21}x_i + a_{22}y_i + b_2$$

We apply it on five features, the transformation is given by:

$$\begin{bmatrix} x1_i & y1_i & 1 \\ x2_i & y2_i & 1 \\ x3_i & y3_i & 1 \\ x4_i & y4_i & 1 \\ x5_i & y5_i & 1 \end{bmatrix} \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ b_1 & b_2 \end{bmatrix} = \begin{bmatrix} x1_i^P & y1_i^P \\ x2_i^P & y2_i^P \\ x3_i^P & y3_i^P \\ x4_i^P & y4_i^P \\ x5_i^P & y5_i^P \end{bmatrix}$$

Until now, the problem convert to solve $Ax = b$ in linear algebra. Since we got 10 equations for 6 unknowns, we employ SVD (*function pinv() in matlab*) to solve it.

### How to find $\overline{F}$

It is very important to get an appropriate $\overline{F}$. We calculate $\overline{F}$ by following steps:

- Step 1: We initialize $\overline{F}$ with first image's feature to find average of all.
- Step 2: Finding transformation that project $\overline{F}$ to predefined position in 64x64 window. Then we can find this affine transformation as A and b matrices that has 6 parameters, and we apply this transformation to $\overline{F}$.
- Step 3: For each image feature $F_i$, we computing the A and b matrices for affine transformation that projecting $\overline{F}$ to $F_i$.
- Step 4: We change $\overline{F}$ to averaged projected feature locations that we calculated previous step.
- Step 5: If error between $\overline{F}_t$ - $\overline{F}_{t-1}$ is less than our threshold, we go back to step 2.

## ⌞ Chapter 3: Principal Component Analysis ⌝

We used PCA to transform our correlated data set into smaller uncorrelated data set. So PCA helped us on reducing the large dimensions of data set into smaller dimensions of it. After finding $\overline{F}$, we followed these steps:

We converted our 64 by 64 normalized images $X_i$ into row vector sized 1x4096 and constructed matrix $D_{pxd}$(p = number of images, d = 64x64) that each row corresponds to $X_i$. Then we defined k between 50 and 100 to $k$th largest eigen values.

Before computing the PCA, we computed the covariance matrix of D as follows:

$$\Sigma = \frac{1}{p-1} D^T D$$

Then we used matlab's "svd()" function to get eigenvectors from $\Sigma$ matrix. This process in our training is taking the most time. We put k eigenvectors corresponding to biggest k eigenvalues to $\Phi$.
We calculated PCA space by;

$$\phi_i = X_i.\Phi$$

PCA space represent the linear combination of eigenfaces of train images. Which means, for any train image, we just need know what is its linear combination of eigenfaces. After all, now we can store all our training images in the PCA space and easily search them to find closest matches.

## ⌞ Chapter 4: Recognition ⌝

In face recognition part we created a function called "RecognizingSystem()"; it has one input: test image name, e.g. 'Ran_1', and return us a test image itself, and related 3 images we found in accuracy order, and accuracy 0 or 1 if first image is correct.

In this function, first we use $\overline{F}$ to normalize our image. Then we use our $\Phi$ to carry it to our PCA space and get its corresponding feature vector $\phi_j$. Then we calculating eclidean distance between $\phi_j$ and all feature vectors $\phi_i$ and order our result ascending according to this distance. We pick the top 3 row, and show the result as our face recognition algorithm. So basically we used the Eigenface idea of set of eigenvectors to detect face that created by Turk and Pentland.

## ⌞ Chapter 5: Experimental Result and Analysis ⌝

We use No.1 and No.5 images of each person as test images and the rest are train images. For each test image, we calculate the distance of feature vector between test image and train images. Then we apply ascending sort on the train images by distance. Since we have three train images for each person, we shown top three of them.

We are unable to get a ideal result which means three of them are matched, however it is acceptable if the first image is matched. We compute the accuracy which is the ratio of all results that first image is matched. The accuracy we got is 69.23%. Result of our matlab code shown by following figure.

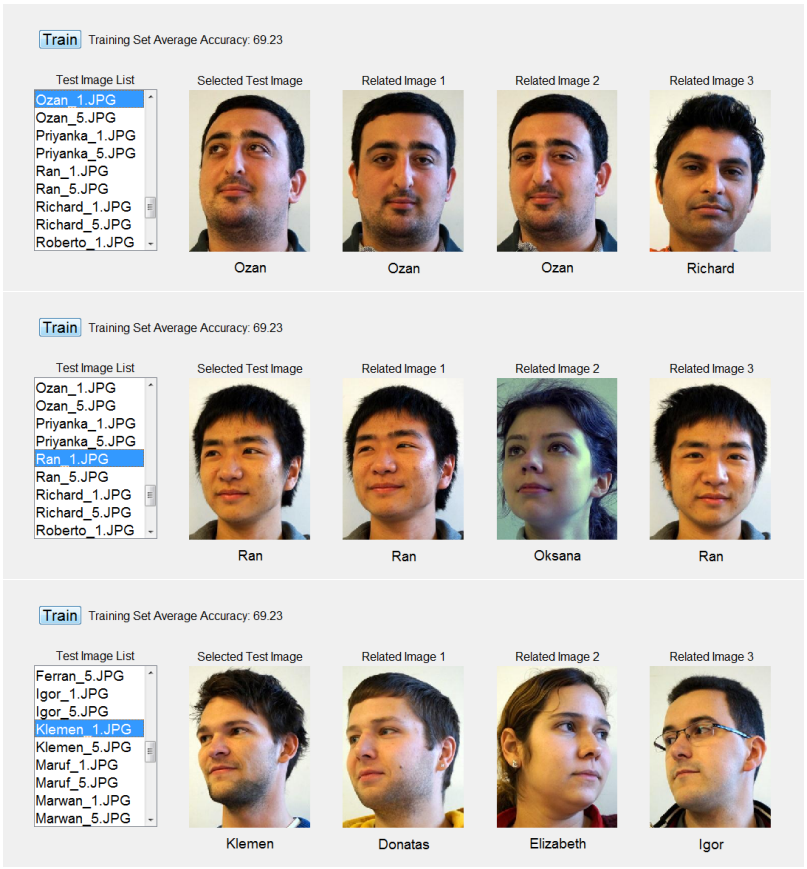


Figure 2:

We also change the number of principal components (*mark as* $k$). When $k < 10$, accuracy decrease effectively when we reduce $k$. When $k > 100$, it has no effect on accuracy. In our finial result, we define $k = 90$.

## Chapter 6: Summary

This project help us understand a great idea that PCA provide: transform the data from correlated to independent. From image processing point of view, PCA is similar with Fourier transform which represent an image as the combination of some principal spectrum, Face recognition based on PCA actually represent a face by the combination of eigenfaces. It is a powerful method to reduce the dimension of problem.

We also got some deeper understanding of the application of some linear algebra tools such as eigenvalue, eigenvector, SVD, etc. Specifically, we got the point that why and how we use SVD.

## References


1. Désiré's Notes - `http://mscvision.u-bourgogne.fr/wp-content/uploads/2011/09/applied_maths1.pdf`
2. `https://en.wikipedia.org/wiki/Principal_Component_Analysis`
3. `https://en.wikipedia.org/wiki/Eigenface`
4. `https://en.wikipedia.org/wiki/Eigenvector`
5. Alper Akcoltekin - Ozan Gonenc (old students)

# Matlab Code

## Get Fbar

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This function is to find the Fbar by iteration
%Note:  Need the extra function to read Fi where Fi is the feature of No.i
%       image. example: Fi = readF(index)
%Fpre is defined on the paper
%input: threshold of error, MaxIterations for limit the interations
%output: Fbar for normalization
%Last modify time: 1/18/2014
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Fbar] = GetFbar(threshold,MaxIterations)

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %check the input
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if (nargin == 2)
        label = ['threshold = ',int2str(threshold)];
        disp(label);
        label = ['MaxIterations = ',int2str(MaxIterations)];
        disp(label);
    end

    if (nargin == 1)
        MaxIterations = 30;
        label = ['threshold = ',int2str(threshold)];
        disp(label);
        disp('MaxIterations = 30');
    end

    if (nargin == 0)%if input is wrong
        MaxIterations = 30;
        threshold = 1;
        disp('threshold = 1');
        disp('MaxIterations = 30');
    end

    pfo = PCAFileOperations; % PCA File Operations

    Fpre = pfo.getPredefinedFeatureLocations();

    [trainingFeatureFiles, trainingFeatureFileCount] = pfo.getTrainingFeatures();

    [allFeatureFiles, allFeatureFileCount] = pfo.getAllFeatures();

    NumberOfTrainFeature = trainingFeatureFileCount;


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Step 1:
    %initialize Fbar = F1
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    F1 = pfo.getFeatureMatrixByIndex(trainingFeatureFiles, 1);
    Fbar = F1;

    IterationCounter = 0;
```

```
    Error = 30;
    Prve_Error = Error;
    while(IterationCounter < MaxIterations)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %Step 2:
        %Get A and b from Fpre = A*Fbar + b
        %Get Fbar = A*Fbar + b
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        IterationCounter = IterationCounter + 1;
        [A,b] = GetAB(Fbar, Fpre);
        Fbar = FeatureTransformation(A, b, Fbar);
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %Step 3:
        %Find Fi_dash of all Fi
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        Fsum = zeros(5,2);
        for i = 1:NumberOfTrainFeature
            Fi = pfo.getFeatureMatrixByIndex(trainingFeatureFiles, i);
            [A,b] = GetAB(Fi, Fbar);
            Fi_dash = FeatureTransformation(A, b, Fi);
            Fsum = Fsum + Fi_dash;
        end
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %Step 4:
        %Update new Fbar by averaging Fi_dash
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        Fbar_t = Fsum / NumberOfTrainFeature;
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %Step 5:
        %error between Fbar_t and Fbar_(t-1)
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        Error = sum(sqrt(sum(((Fbar_t' - Fbar').^2))));
        if(Error < threshold)
            disp('Fbar Found...');
            break;
        end
        %check if it is necessary to continue iteration
        if(Prve_Error <= Error)
            label = ['Minimum Error at iterations = ',int2str(IterationCounter-1),'...Iteration stoped.
            disp(label);
            break;
        else
            label = ['Iterations = ',int2str(IterationCounter),' Error = ',num2str(Error)];
            disp(label);
            Fbar = Fbar_t;
        end
        Prve_Error = Error;
    end
end
```

### Training

```
display('Training is started. This may take several minutes. Please wait...');

pfo = PCAFileOperations;

Fbar = GetFbar();
save('Fbar.mat','Fbar');
```

```
pfo.saveFbar(Fbar);

[trainingFeatureFiles, trainingFeatureFileCount] = pfo.getTrainingFeatures();

%traningSetFolder = pfo.getTrainingSetFolderPath();

%normalizedSetFolder = pfo.getNormalizedSetFolderPath();

%dirOutput = dir(fullfile(traningSetFolder, '*.jpg'));

%fileNames={dirOutput.name}';

fileNames = pfo.getTrainingSetImageNameList();

D_matrix = zeros(length(fileNames),4096);

for i=1:length(fileNames)
    Fimg = pfo.getFeatureMatrixByIndex(trainingFeatureFiles, i);
    Img = pfo.getTrainingImageByName(fileNames{i});
    normImg = ImageNormalization(Img, Fimg, Fbar);
    pfo.saveNormalizedImage(normImg, fileNames{i});
    D_matrix(i, :) = reshape(normImg, 1, 4096);
end



pfo.saveDMatrix(D_matrix);
save('D_matrix.mat','D_matrix');
k = 50;
display('Calculating Sigma...');
Sigma = 1/(length(fileNames)-1) .* D_matrix' * D_matrix;
save('Sigma.mat','Sigma');
display('Starting SVD process. This may take several minutes. Please wait...');
tic;
[U,S,V] = svd(Sigma);
display('Total time for SVD: ');
toc;
Phi = V(:,1:k);
save('Phi.mat','Phi');
PCAspace = D_matrix*Phi;
save('PCAspace.mat','PCAspace');

display('Training is finished.');



accuracy = FindAccuracy();
```

## Recognizing System

```
function [Test_image, Img1, Img2, Img3, Name1, Name2, Name3, Error] = RecognizingSystem(test_image)
    pfo = PCAFileOperations;
    fileNames = pfo.getTrainingSetImageNameList();
    Fbar = importdata('Fbar.mat');
    testImagePath = strcat(test_image, '.JPG');
    Test_image = pfo.getOriginalImageByName(testImagePath);
    F_test_img = pfo.getFeatureMatrixByName(test_image);
```

```
    Index_Distance_Table = FaceRecognization(Test_image,F_test_img,Fbar);
    Name_1 = fileNames{Index_Distance_Table(1,1)};
    Img1 = pfo.getTrainingImageByName(Name_1);
    Name_2 = fileNames{Index_Distance_Table(2,1)};
    Img2 = pfo.getTrainingImageByName(Name_2);
    Name_3 = fileNames{Index_Distance_Table(3,1)};
    Img3 = pfo.getTrainingImageByName(Name_3);

    C = strsplit(test_image,'_');
    TestImageName = C(1);
    C = strsplit(Name_1,'_');
    Name1 = C(1);
    C = strsplit(Name_2,'_');
    Name2 = C(1);
    C = strsplit(Name_3,'_');
    Name3 = C(1);

    if(~strcmp(Name1, TestImageName))
        Error = 1;
    else
        Error = 0;
    end

end
```

### Image Normalization

```
function [normImg] = ImageNormalization(OriginalImg, Fimg, Fbar)

img = rgb2gray(OriginalImg);
%img = histeq(img);

[A,b] = GetAB(Fimg, Fbar);

normImg = uint8(zeros(64, 64));

 for i=1:64
      for j=1:64
          % solve the equation xy64 = A*xy + b to obtain the pixel
          %positions in the bigger image
          xy = (pinv(A)*( [ i; j ] - b ));

          % extract the x and y coordinate
          x240 = int32(xy(1,:));
          y320 = int32(xy(2,:));

          % Although very rare, these values can fall down to negative values. So if it
          % happens, just make it zero. One pixel won't make a huge difference.
          if(x240 <= 0)
              x240 = 1;
          end

          if(y320 <= 0)
              y320 = 1;
          end

          if(x240 > 240)
              x240 = 240;
```

```
            end

            if(y320 >320)
                y320 = 320;
            end
            % copy the value of the pixel in the bigger image to the
            % normalized image
            normImg(i,j) = uint8(img(y320, x240));
        end

 end

 normImg = normImg';

end
```

### Get AB

```
function [A,b] = GetAB(F, Fp)
%     AB = zeros(3,2);
    AB = pinv([F, [1;1;1;1;1]]) * Fp;
    A = AB(1:2,:);
    A = A';
    B = AB(3,:);
    b = B';
end
```

### Feature Transformation

```
function [F_of_norm] = FeatureTransformation(A, b, F_of_img)
    AB = [A'; b'];
    F_of_norm = [F_of_img, [1;1;1;1;1]] * AB;
end
```

### Face Recognization

```
function [Index_Distance] = FaceRecognization(Test_image,F_test_img,Fbar)
    Phi = importdata('Phi.mat');
    PCAspace = importdata('PCAspace.mat');
    img = ImageNormalization(Test_image,F_test_img,Fbar);
    Xj = reshape(img,1,4096);
    oslash_j = double(Xj) * double(Phi);
    [m,n] = size(PCAspace);
    Index_Distance = zeros(m,2);
    for i = 1:m
        Index_Distance(i,1) = i;
        Index_Distance(i,2) = sqrt(sum((PCAspace(i,:) - oslash_j).^2));
    end
    Index_Distance = sortrows(Index_Distance,2);
end
```