ECE661 Computer Vision: HW 9
Object Recognition

By Sirui Hu, 12-16-2012

Problem 1: Face Recognition using PCA and LDA

PCA Object Recognition (Eigen Faces)

PCA is a simple, non-parametric method extracting relevant information from confused data sets. Application of PCA in face recognition linearly projects the image space to a low dimensional feature space, which gives projection directions that maximize the total scatter across all classes, i.e. across all images of all faces.

With X_i representing the image vector of the i-th image and $m = \frac{1}{N}\sum_{i=1}^{N} X_i$, we have

$$X = [X_1 - m, X_2 - m, ..., X_N - m]$$

The total variance matrix is defined as

$$C = XX^T = \frac{1}{N}\sum_i \sum_j \left(\overline{x_{ij}} - \overline{m}\right)\left(\overline{x_{ij}} - \overline{m}\right)^T$$

The goal is to obtain N eigenvectors of C: $C\vec{w} = \lambda\vec{w}$.

Directly do eigen-decomposition on C is too much computation. As C is highly over-determined, we do eigen-decomposition on $X^T X$ instead of XX^T.

$$X^T X\vec{u} = \lambda\vec{u}$$

$$CX\vec{u} = \lambda X\vec{u}$$

$$\therefore \vec{w} = X\vec{u}$$

The new feature vectors $y_k \in \Re^m$ are defined by the following linear transformation:

$$y_k = W^T x_k \quad k = 1, 2, ..., N \ ,$$

where $W \in \Re^{n \times m}$ is a matrix with orthonormal columns.

After applying the linear transformation W^T, the variance of the transformed feature vector $\{y_1, y_2, ..., y_N\}$ is $W^T C W$.

The set of eigenvector with eigenvalue greater than 1 are retained, and form the matrix W_p. The projection of all training images on W_p is

$$y_i = W_p^T (X_i - m)$$

The projection of each test image onto the PCA basis:

$$y_i' = W_p^T (X_i' - m)$$

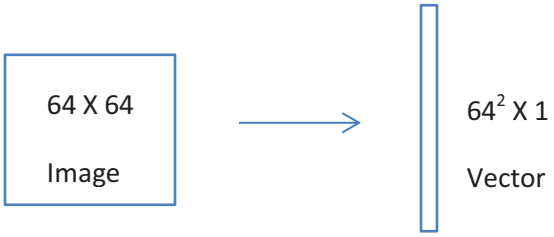
The Euclidean distance between two projection is calculated. The nearest neighbor in the distance space will be recognized as the matched image.

The accuracy of classification is $$\text{accuracy(p)} = \frac{Number\ of\ images\ correctly\ recognized}{Total\ number\ of\ images}$$

where p is the number of eigenvectors used in descripting the feature space.

PCA Procedure:

Step 0: Vectorize face images in training and testing database (make column vectors out of each of them), and subtract the mean value of the image from each image vector.



Normalizing the vector to make the norm of the image vector equal to 1, which add tolerance of difference illumination among images.

Training Steps

1. Calculate the mean of the input face images
2. Subtract the mean from the input images to obtain the mean-shifted images

3. Compute covariance matrix for the image vectors. Calculate the eigenvectors and eigenvalues of the mean-shifted images
4. Order the eigenvectors by their corresponding eigenvalues, in decreasing order
5. Retain only the eigenvectors with the largest eigenvalues. (the principal components)
    a. The remaining feature set could be with eignevalues greater than 1. If a given eigenvalue is greater than 1, the vectors are stretched in the direction of the corresponding eigenvector.
6. Project the mean-shifted images into the eigenspace using the retained eigenvectors

Classification (Testing) Steps

After the face images have been projected into the eigenspace, the similarity between any pair of face images can be calculated by finding the Euclidean distance between their corresponding feature vectors

Classification is performed using a nearest neighbor classifier in the reduced feature space. The similarity score is calculated between an input face image and each of the training images.

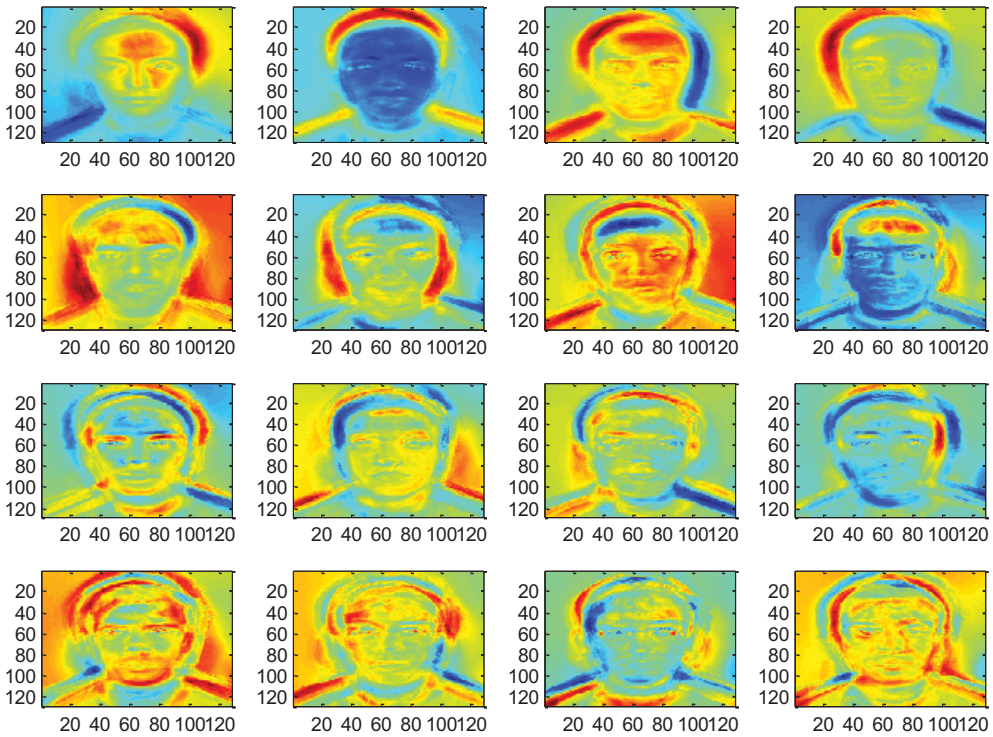


Figure 1. The top 16 eigenfaces. The top-left one is the most principal one.

Classification Result Demo

Test Input Image

Test Image: ./test/06_15.png

Recognized Output Image

Recognized Image: ./train/06_05.png
PCA with 1 eigenvectors

Distance Score (Similarity) = 5.357×10^{-6}

Test Input Image

Test Image: ./test/01_01.png

Recognized Output Image

Recognized Image: ./train/14_11.png
PCA with 3 eigenvectors

Distance Score (Similarity) = 0.0017

Recognized Image: ./train/01_17.png
PCA with 4 eigenvectors

Distance Score (Similarity) = 0.0091

This is the minimum score among other train images, using given feature space.

LDA Object Recognition Process (Fisher Faces)

This method is an enhancement of PCA method, that it uses Fisher's Linear Discriminant Analysis (LDA) for the dimensionality reduction.

Between-class variance is defined by:

$$S_B = \frac{1}{N_C}\sum_{i=1}^{N_C}\left(\overline{m_i} - \overline{m}\right)\left(\overline{m_i} - \overline{m}\right)^T$$

Within-class variance is defined by:

$$S_W = \frac{1}{N_C}\sum_{i=1}^{N_C}\left(\frac{1}{N}\sum_{j=1}^{N_C}\left(\overline{x_{ij}} - \overline{m}\right)\left(\overline{x_{ij}} - \overline{m}\right)^T\right)$$

where N_C is the number of class and $\overline{m_i}$ is the mean vector of the ith class.

The GOAL of LDA method is to maximize the ratio of between-class variance and within-class variance with respective to eigenvector $\vec{w}$

$$J(\vec{w}) = \frac{\vec{w}^T S_B \vec{w}}{\vec{w}^T S_W \vec{w}}$$

Since the global mean is from by summing all class mean, only $N_C - 1$ of the N_C matrices that go into S_B are linearly independent. The rank of S_B is at most $N_C - 1$.

The following procedure described by Yu and Yang is to avoid the problem that S_W may be singular.

1. Use regular eigen-decomposition to diagonalize S_B. This will yield a matrix V of eigenvectors such that $V^T S_B V = \Lambda$.
   where $V^T V = I$ and Λ is a diagonal matrix of eigenvalues in a descending order.
2. Discard eigenvalues that are close to 0 in Λ (no inter-class discriminations), and retain only M eigenvalues. Let Y be the matrix formed by the first M eigenvectors in V.
   $$Y^T S_B Y = D_B$$
   where D_B is the upper-left M*M sub-matrix of Λ.
3. Construct a matrix Z as follows: $Z = YD_B^{-1/2}$

4. Now diagonalize $Z^T S_W Z$ by regular eigendecomposition. This will yield a matrix U of eigenvectors such that

   $U^T Z^T S_W Z U = D_W$, where $U^T U = I$.

5. Since the goal is to maximize the ratio of between-class scatter to within-class scatter and since much inter-class discriminatory information is contained in the smallest eigenvectors of S_W, the largest eigenvalues of D_W and the corresponding eigenvectors are discarded.
6. Let $\hat{U}$ denotes the matrix consists of p (p<M) eigenvectors retained from U. Then matrix of the LDA eigenvectors that maximize the fisher equation is given by

$$W = \hat{U}^T Z^T$$

Comparison of PCA and LDA

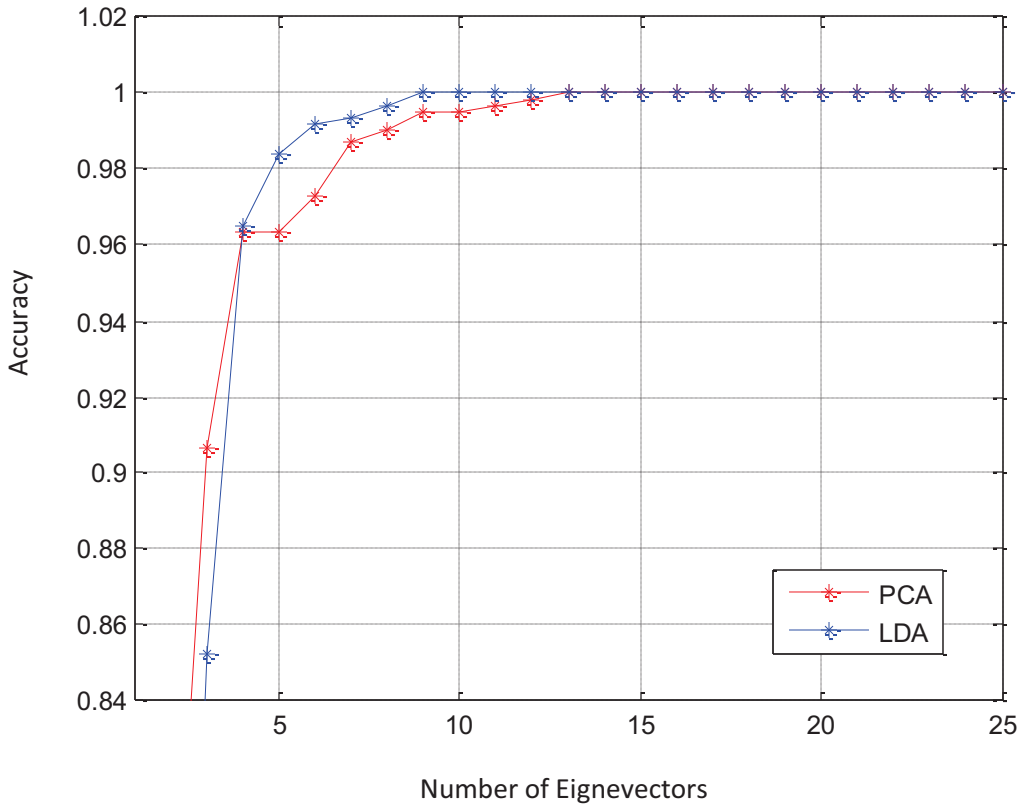
The following figure shows a comparison of PCA and LDA results.

LDA achieve 100% accuracy rate after using 9 eigenvectors as feature space. PCA achieve 100% accuracy after using 13 eigenvectors.

The PCA projections are optimal for representation in a low dimensional basis, but they may not be optional from a discrimination standpoint.

The LDA maximizes the ratio of between-class variance to that of within-class variance. It works better than PCA for purpose of discrimination.

In sum, LDA outperformed the PCA method. Both has good performance under varying illumination.



Matlab Code

Load Database Images Function

```
% Function: Load Training Database Images
% Return vectorized images subtracting the mean and being normalized
% to deal with images with different luminance

function [normImgVec,imgVec,meanFace] =
loadImageDB(path,noPerson,noFaces,sizeImg)

    noTrain = noPerson*noFaces;
    imgVec = zeros(sizeImg,noTrain);  % 16384 x 630

    count=1;
    for i=1:noPerson
    for j=1:noFaces

        stri=num2str(i);
        strj=num2str(j);
```

```
        if i<10
            stri=['0',num2str(i)];  end
        if j<10
            strj=['0',num2str(j)];  end
        img = imread([path,stri,'_',strj,'.png']);
        img = rgb2gray(img);

        temp = reshape(img',sizeImg,1);   % Reshaping 2D images into 1D image
vectors
        imgVec(:,count) = temp;

        count=count+1;
    end
    end

    meanFace = mean(imgVec,2); % the average of face images

    normImgVec = zeros(sizeImg,noTrain);
    for i = 1 : noTrain
        imgVecCenter = double(imgVec(:,i)) - meanFace; % subtract mean
        % normalized
        normImgVec(:,i) = imgVecCenter/norm(imgVecCenter);
    end
```

Perform PCA Function

```
% Perform PCA

function [normEigenFaces, noStretchEig] = performPCA(normTrainVec)

    A = normTrainVec;
    dim = size(A,2);

    [V D] = eig(A'*A);  % computation trick of covariance matrix C=A*A'

    % sort the eigenvalues in descending order
    eigValue = diag(D);
    [temp sortind] = sort(-1.*eigValue);
    eigValue = eigValue(sortind);
    V = V(:,sortind);

    % get number of eigenvector that has eigenvalue greater than 1
    % to form the feature classification matrix
    % Notes: If a given eigenvalue is greater than 1, the vectors are
    % stretched in the direction of the corresponding eigenvector
    noStretchEig = 0;
    for i = 1:dim
        if eigValue(i)>1
            noStretchEig = noStretchEig+1;
```

```
        end
    end

    % use all eigenvectors in this subroutine
    eigVec = [];
    for i = 1:dim
        eigVec = [eigVec V(:,i)];
    end

    % project get full set of eigenfaces
    eigenFaces = A * eigVec;

    % plot the first 16 eigen faces
    figure(4);
    for i=1:16
         img=reshape(eigenFaces(:,i)',128,128);
         subplot(4,4,i)
         imagesc(img');
    end

    for i=1:dim
        normEigenFaces(:,i) = eigenFaces(:,i)/norm(eigenFaces(:,i));
    end

end
```

Training Main Function using PCA Method

```
% Face Recoginition (training)
% Pre-compute EigenFaces
% Sirui HU
% 2012-12-07

% see also: loadImageDB, performPCA

clc
clear all
close all

% Initial training and testing database dimensions
noPerson = 30;
noFaces = 21;
noTrain = noPerson*noFaces;

sizeImg = 128*128;


% Load Training Image Vector subtracting the mean and being normalized
TrainPath = './train/';
[normTrainVec,~,~] = loadImageDB(TrainPath,noPerson,noFaces,sizeImg);
```

```
% Perform PCA, get number of eigen values greater than 1
[normEigenFaces, noStretchEig] = performPCA(normTrainVec);

PCATraining.normEigenFaces = normEigenFaces;
PCATraining.noStretchEig = noStretchEig;

save('PCATraining.mat','PCATraining','-mat','-v7.3');
```

Testing Main Function using PCA Method

```
% Face Recoginition Pre-compute EigenFaces
% Sirui HU
% 2012-12-07

% see also: loadImageDB

clc
clear all
close all

% Initial training and testing database dimensions
noPerson = 30;
noFaces = 21;
noTest = noPerson*noFaces;
noTrain = noTest;
sizeImg = 128*128;


% Load Training Image Vector subtracting the mean and being normalized
TrainPath = './train/';
[normTrainVec,~,~] = loadImageDB(TrainPath,noPerson,noFaces,sizeImg);

% Load Testing Image Vector subtracting the mean and being normalized
TestPath = './test/';
[normTestVec,~,~] = loadImageDB(TestPath,noPerson,noFaces,sizeImg);

% Load precompute PCA components
file = load('PCATraining.mat');
normEigenFaces = file.PCATraining.normEigenFaces;
noStretchEig = file.PCATraining.noStretchEig;

% Set the number of eigenvectos
noEig = noStretchEig;

% Test Accuracy using increasing number of eigenvectors
accuracyPCA = zeros(noTest,noEig);
```

```
for k=1:noEig

    % Select the eigenvector with largest eigen values
    eigenFacesPart = normEigenFaces(:,1:k);

    % Projection of centered train images into feature space
    ProjectedImages = zeros(k,noTrain);
    for i = 1 : noTrain
        ProjectedImages(:,i) = eigenFacesPart'*normTrainVec(:,i);
    end

    % Projection of centered test images into feature space
    ProjectedTestImages = zeros(k,noTest);
    for i = 1 : noTrain
        ProjectedTestImages(:,i) = eigenFacesPart'*normTestVec(:,i);
    end

    errDistance = zeros(noTest,1);

    % input one test image
    for testInd = 1:noTest

        testI = floor(testInd/noFaces)+1;
        testJ = mod(testInd,noFaces);
        if testJ==0
            testI = testI-1;
            testJ = testJ+1;
        end

        % show the test image
        stri=num2str(testI);
        strj=num2str(testJ);
        if testI<10
            stri=['0',num2str(testI)];  end
        if testJ<10
            strj=['0',num2str(testJ)];  end
        testimg_name = ['./test/',stri,'_',strj,'.png'];
        testimg = imread(testimg_name);
        figure(testInd*2-1)
        imshow(testimg)
        title(['Test Image: ',underline(testimg_name)])

    % calculate the distance between projected test image and projected
    % train image database
    for trainInd = 1 : noTrain
        errDistance(trainInd) = (norm(ProjectedTestImages(:,testInd)-
ProjectedImages(:,trainInd)))^2;
    end

    % Get the nearest neighbor as output
    [minDist, recogInd] = min(errDistance);
    [sortDist, sortInd] = sort(errDistance);

        recogI = floor(recogInd/21)+1;
```

```
        recogJ = mod(recogInd,21);
        if recogJ==0
            recogI = recogI-1;
            recogJ = recogJ+1;
        end

        % show the recognized image
        stri=num2str(recogI);
        strj=num2str(recogJ);
        if recogI<10
            stri=['0',num2str(recogI)];  end
        if recogJ<10
            strj=['0',num2str(recogJ)];  end
        train_name = ['./train/',stri,'_',strj,'.png'];
        trainimg = imread(train_name);
        figure(testInd*2)
        imshow(trainimg)
        title({['Recognized Image: ',underline(train_name)],['PCA with
',num2str(noEig),' eigenvectors']})

    % record classification accuracy using k eigenvectors
    if testI==recogI
        accuracyPCA(testInd,k)=1;
    end

    end

end


% save recognization accuracy rate
accuracyPCARate = sum(accuracyPCA,1)/noTest;
save ('PCA_accuracyRate.dat', 'accuracyPCARate', '-ASCII');

%accuracyPCARate = load('PCA_accuracyRate.dat');
figure(testInd*2+1)
ind=1:noEig;
plot(ind(1:25),accuracyPCARate(1:25),'r*-');
axis([1 25 0.84 1])
```

Main Function using LDA Method

```
% Face Recoginition using LDA
% Sirui HU
% 2012-12-07

% see also: loadImageDB

% Initial training and testing database dimensions
noPerson = 30;
```

```
noFaces = 21;
noTrain = noPerson*noFaces;
noTest = noTrain;
sizeImg = 128*128;


% Load Raw Training Image Vector
TrainPath = './train/';
[~,imgTrainVec,meanTrainFace] =
loadImageDB(TrainPath,noPerson,noFaces,sizeImg);

% Load Raw Test Image Vector
TestPath = './test/';
[~,imgTestVec,meanTestFace] = loadImageDB(TestPath,noPerson,noFaces,sizeImg);


% Calculate the average of each class(person)
noImg = noPerson*noFaces;
meanClass = zeros(sizeImg,noPerson);
for i=1:noImg
    imgI = floor(i/noFaces)+1;
    imgJ = mod(i,noFaces);
    if imgJ==0
        imgI = imgI-1;
    end
    meanClass(:,imgI) = meanClass(:,imgI) + imgTrainVec(:,i);
end
meanClass = meanClass/noFaces;


    % SB=XB*XB'
    XB = zeros(sizeImg,noPerson);
    for i=1:noPerson
        XB(:,i)=meanClass(:,i)-meanTrainFace;
    end
    % SW=XW*XW'
    XW = zeros(sizeImg,noImg);
    for i=1:noImg
        XW(:,i)=imgTrainVec(:,i)-meanClass(:,floor((i-1)/noFaces)+1);
    end

    % Yu and Yang's method to avoid SB being singular

    [vecB,valB] = eig(XB'*XB);
    D = diag(valB);
    [temp sortind] = sort(-1.*D);
    vecB = vecB(:,sortind);
    D = D(sortind);
    V = XB*vecB;

    maxnoFeature = 30;
    Y = V(:,1:maxnoFeature);
    DB = Y'*XB*XB'*Y;
    Z = Y*DB^(-0.5);
    H = Z'*XW;%M*630
```

```
    [vecU,valU]=eig(H*H');
    diagvalU=diag(valU);

    accuracyLDA = zeros(noTest,maxnoEig-1);

    for L = 1:maxnoEig-1

        vecUPart = vecU(:,1:L);
        W = Z*vecUPart;
        for i=1:L
            W(:,i) = W(:,i)/norm(W(:,i));
        end

        ProjectedImages = zeros(L,noImg);
        ProjectedTestImages = zeros(L,noImg);
        for i = 1:noImg
            ProjectedImages(:,i) = W'*(imgTrainVec(:,i)-meanTrainFace);
        end
        for i = 1:noImg
            ProjectedTestImages(:,i) = W'*(imgTestVec(:,i)-meanTestFace);
        end
        errDistance = zeros(noTest,1);

        for testInd = 1:noTest

            testI = floor(testInd/21)+1;
            testJ = mod(testInd,21);
            if testJ==0
                testI = testI-1;
                testJ = testJ+1;
            end
            % show the test image
            %stri=num2str(testI);
            %strj=num2str(testJ);
            %if testI<10
            %    stri=['0',num2str(testI)];  end
            %if testJ<10
            %    strj=['0',num2str(testJ)];  end
            %testimg = imread(['./test/',stri,'_',strj,'.png']);
            %figure(1)
            %imshow(testimg)

            for trainInd = 1 : noTrain
                errDistance(trainInd) = (norm(ProjectedTestImages(:,testInd)-
ProjectedImages(:,trainInd)))^2;
            end
            [minDist, recogInd] = min(errDistance);

            recogI = floor(recogInd/21)+1;
            recogJ = mod(recogInd,21);
            if recogJ==0
                recogI = recogI-1;
                recogJ = recogJ+1;
            end
            % show the recognized image
```

```
        %stri=num2str(recogI);
        %strj=num2str(recogJ);
        %if recogI<10
        %    stri=['0',num2str(recogI)];  end
        %if recogJ<10
        %    strj=['0',num2str(recogJ)];  end
        %trainimg = imread(['./train/',stri,'_',strj,'.png']);
        %figure(2)
        %imshow(trainimg)

        if testI==recogI
            accuracyLDA(testInd,L)=1;
        end
    end
end

% save recognization accuracy rate
accuracyLDARate = sum(accuracyLDA,1)/noTest;
save ('LDA_accuracyRate.dat', 'accuracyLDARate', '-ASCII');


%accuracyLDARate = load('LDA_accuracyRate.dat');
figure(3)
hold on
ind=1:maxnoEig-1;
plot(ind(1:25),accuracyLDARate(1:25),'b*-');
axis([1 25 0.84 1.02])
legend ('PCA','LDA')
```

HW9 Part 2:

Object Recognition using Viola-Jones Ada Boost Algorithm

This homework implements an Adaboost classifier with Haar-like features, to detect cars, following Viola and Jones's paper. It is a two-class classification for each sliding window that scans through the image, e.g. cars vs. non-cars. Such classifier requires a very low false positive rate, e.g. detecting non-car as car.

The given test and train database contain both negative and positive images. The image size is of 20 by 40 pixels.

Steps:

1. Compute the integral image for rapid feature calculation.

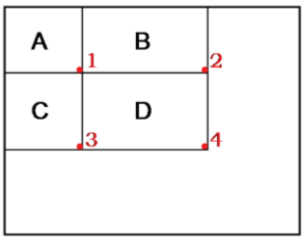
Integral image of a location (x,y) is the sum of the pixel values above and to the left of (x,y).

$$ii(x,y) = \sum_{x' \le x, y' \le y} i(x', y'),$$

where is the integral image and is the original image.
In Matlab implementation, the integral version of an image could be obtained using

```
ii = cumsum(cumsum(double(img)),2);
```

After integrating, we could calculate the sum of all pixels in one rectangle easily, with the location of four corner points.



The sum of pixel values in rectangle D is ii(x4, y4) - ii(x2, y2) - ii(x3, y3) + ii(x1, y1).

2. Compute four types of Haar-like features.

Each feature is obtained by subtracting white areas from the black areas.

(a) Edge features (b) Line features (c) four-rectangular feature

For a 1x2 window sliding through image of size 20x40, there are in total 20x39 features.

For type one (1:2 two rectangular windows), the size could be extended vertically as 1x2, 1x4, 1x6, 1x8 …; horizontally it could be extended as 1x2, 2x2, 3x2 …

The following code counts the number of features (1x2 window):

```
% Count the number of feature for type I
imgW = 20;
imgH = 40;
w = 2;  h = 1; % initial base window size
count = 0;
for i = 1:imgW/w   % extend the size of window
for j = 1:imgH/h
    for x = 1:imgW-i*w+1   % slide the window through the image
    for y = 1:imgH-j*h+1
        count = count+1;
    end
    end
end
end
```

There are in total 166,000 edge features (1:2 and 2:1 windows), 54,600 type b line features and 40,000 type c features. The whole feature set is too large. To reduce computation time, I only use the following two types (136,600 features in total).

Some features could not contribute for good classification. In my implementation, feature windows that have base size (2x1 and 1x3) are ignored, as their distribution is very random. The number of features is reduced by 31,240 in this step. Also, feature windows that contain the boundary pixel are ignored, as the main information usually do not appear at boundary. Windows that are almost as large as the whole images are casted away under this constrain. After this reduction, only 75,286 features are left per image.

We will use Adaptive Boost method to select good features for efficiency. The next chance to dynamic reduce computation is at training classifier step.

3. Adaboost Machine Learning

1) Weak Classifier (boosting for feature selection)

Define a weak classifier of image x based on rectangular feature value f(x):

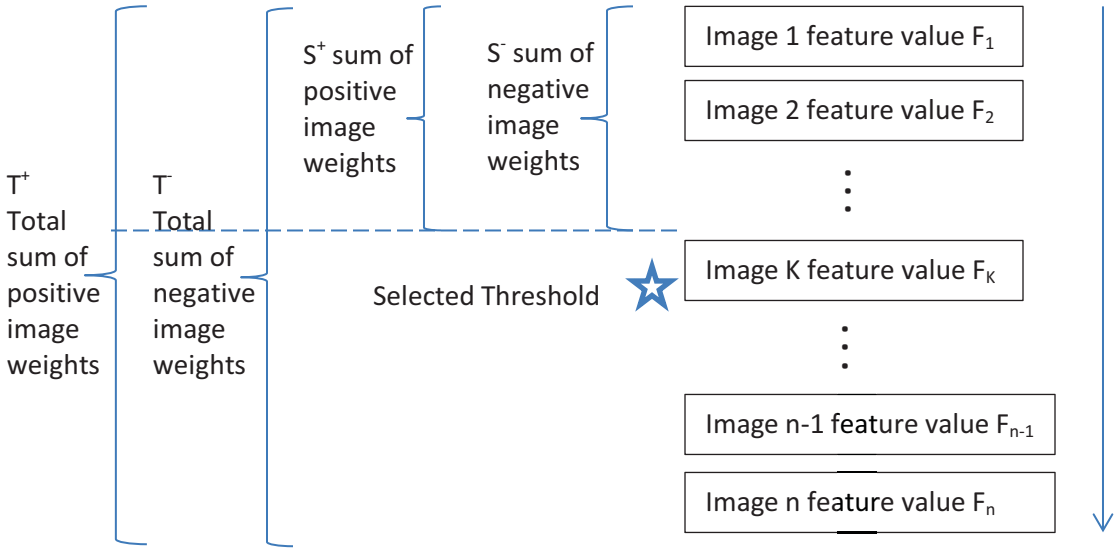
$$h(x, f, p, \theta) = \begin{cases} 1 & if\ pf(x) < p\theta \\ 0 & otherwise \end{cases},$$

where θ is a threshold and p is a polarity indicating sign.

The weak classifier tries to find the best threshold in one feature dimensions to separate the training data into two classes. To train a weak classifier (for one feature) is to determine the best threshold value that minimizes the classification error, under current weights of training images.

For each feature, we sort the images according to their feature values in ascending order. The threshold could be set between any feature values. The classification error of one threshold is

$$e = \min\left(S^+ + (T^- - S^-), S^- + (T^+ - S^+)\right)$$



Initially, we weight each training image equally. The boosting part calls the classifier iteratively. After every classification round, it changes the weights of miss-classified examples.

The algorithm is described as following:

T hypotheses are constructed each using a single feature. The final hypothesis is a weighted linear combination of the T hypotheses where the weights are inversely proportional to the training errors

- Given example images $(x_1, y_1), ..., (x_n, y_n)$ where $y_i = 0,1$ for negative and positive examples respectively. n are the total training set number.
- Initialize weights $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ for $y_i = 0,1$ respectively, where m and l are the number of negatives and positives respectively.
- For $t = 1, ...T$:
  1. Normalize the weights, $\varpi_{t,i} \leftarrow \frac{\varpi_{t,i}}{\sum_{j=1}^{n} \varpi_{t,i}}$
  2. Select the best weak classifier with respect to the weighted error $\varepsilon_t = \min_{f,p,\theta} \sum_i \varpi_i \left| h(x_i, f, p, \theta) - y_i \right|$
  3. Define $h_t(x) = h(x, f_t, p_t, \theta_t)$, where f_t , p_t , and θ_t are the minimizers of ε_t
  4. Update the weights:

     $$w_{t+1,i} = w_{t,i} \beta_t^{1-e_i}$$

     where $e_i = 0$ if example x_i is classified correctly, $e_i = 1$ otherwise, and $\beta_t = \frac{\varepsilon_t}{1 - \varepsilon_t}$.

2) Strong Classifier

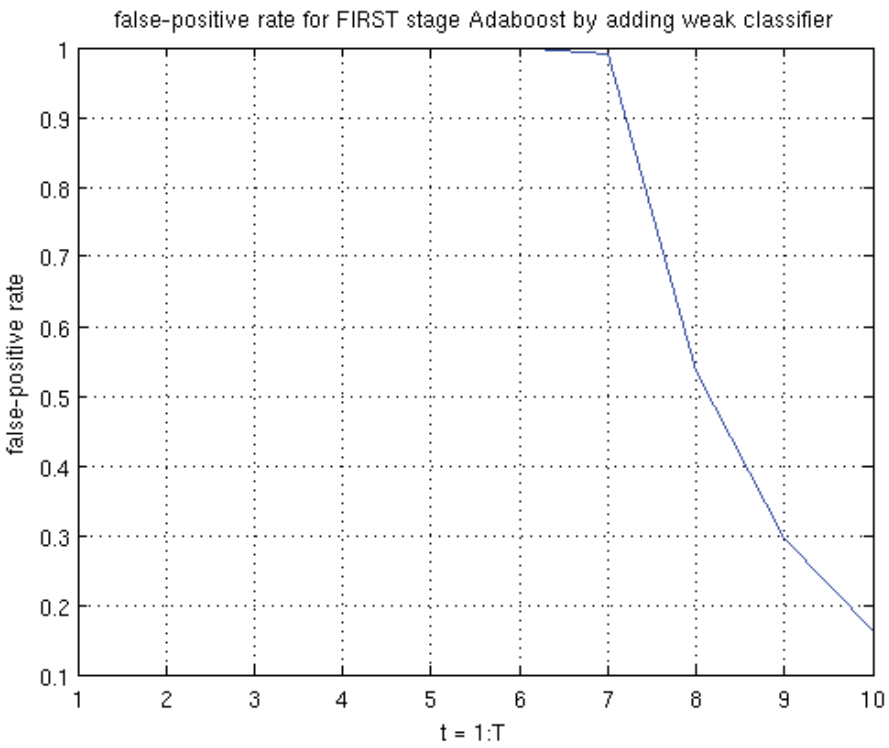
This creates a cascade of "weak classifiers" which behaves like a "strong classifier".

The final strong classifier is:

$$C(x) = \begin{cases} 1 & \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0 & otherwise \end{cases}$$, where $\alpha_t = \log \frac{1}{\beta_t}$

To make sure all the positive training images are recognized as positive (100% classification correctness), we set the strong classifier threshold to be the minimum value of $\sum_{t=1}^{T} \alpha_t h_t(x)$ among positive images.

For each boosting iteration, we go through all features to find the weak classifier that achieves the lowest weighted training classification error. We will keep adding weak classifier (one feature dimension) until the classification error rate of the overall strong classifier is lower than 50%.



4. Attentional Adaboost stage cascade for fast rejection

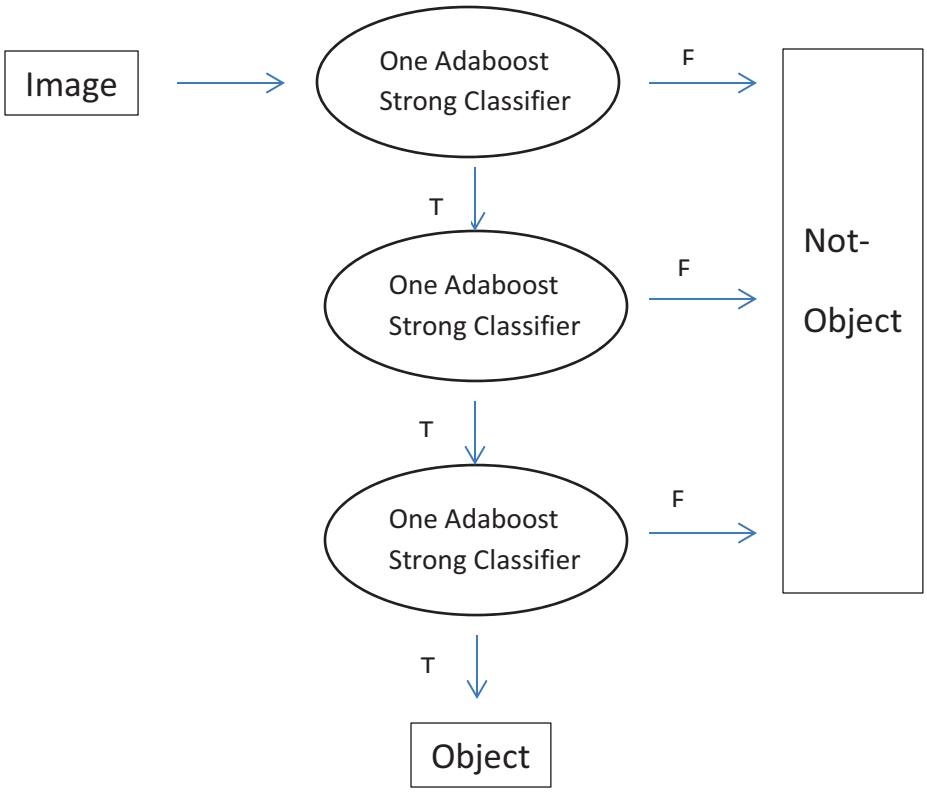
A successful classifier need to achieve a very low false positive rate and high correctness.

Viola and Jones combine several stages of Adaboost classifier as a filter chain. Each filter is a separate Adaboost classifier with several weak classifiers. The filters at each level are trained to classify training images that passed all previous stages. If any one of these filters fails to pass an image, the region is immediately rejected. Only images that pass through all the filters in the chain are classified as positive.

The following terms at each stage are recorded in training process:

1) The features (index and total number) selected to form the strong classifier
2) Threshold
3) Recognized as positive results (false positive and true positive), which will be forward to next stage filter

$$\text{false positive rate} = \frac{\#\text{ of misclassified negative test images}}{\#\text{ of negative test images}}$$



Here are some parameters in my implementation:

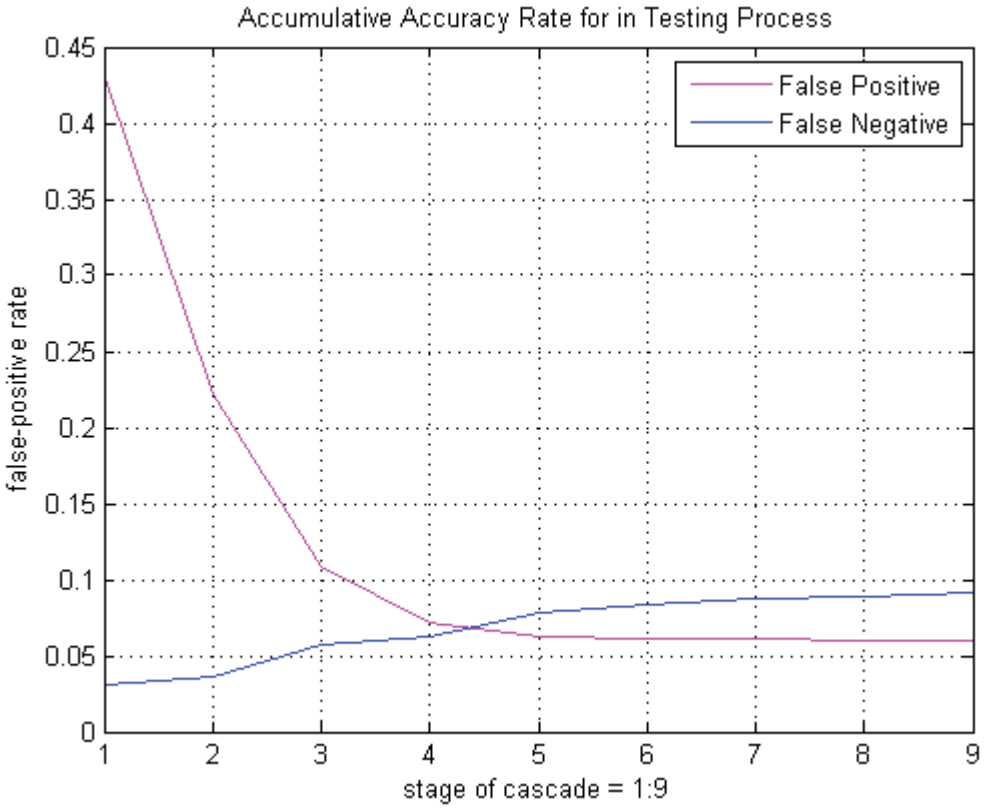
Stage No.	Number of features (weak classifier)	Number of negative training samples for next iteration
1	11	647
2	16	323
3	22	128
4	13	57
5	8	23
6	7	9
7	5	5
8	3	4
9	3	1

5. Testing

Pass the image to the cascaded strong classifiers for detection, report as object if the output is positive.

$$\text{false negative rate} = \frac{\#\text{ of misclassified positive test images}}{\#\text{ of positive test images}}$$


Accumulative false-positive rate for in Training Process
false-positive rate
stage of cascade = 1:9
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
2
3
4
5
6
7
8
9


Accumulative Accuracy Rate for in Testing Process
False Positive
False Negative
false-positive rate
stage of cascade = 1:9
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
2
3
4
5
6
7
8
9

6. Matlab Code

1) Pre-compute Feature Set of Training and Testing Database

```
%<<<<< pre-compute haar features for all training images>>>>>%

clc
clear all
close all

posTrainSize = 710;
negTrainSize = 1758;
noTrain = posTrainSize+negTrainSize;
posTestSize = 178;
negTestSize = 440;
noTest = posTestSize+negTestSize;
imgW = 40;
imgH = 20;


% 1) load the training/testing database
%posTrainImages =
readImageDB('./train/positive/',imgH,imgW,posTrainSize,0,'000');
%negTrainImages =
readImageDB('./train/negative/',imgH,imgW,negTrainSize,0,'00');

posTestImages =
readImageDB('./test/positive/',imgH,imgW,posTestSize,posTrainSize,'000');
negTestImages =
readImageDB('./test/negative/',imgH,imgW,negTestSize,negTrainSize,'00');

noFeatures = 75286;
features = zeros(noFeatures, noTest);
for i=1:noTest
    if i<=posTestSize
        % 2) Get integral image in training DB
        intgImg = integralImg(posTestImages(:,:,i));
        % 3) Calculate all features
        features(:,i) = getHaarFeaturesSub(intgImg);
    else
        intgImg = integralImg(negTestImages(:,:,i-posTestSize));
        features(:,i) = getHaarFeaturesSub(intgImg);
    end
end

% 4) save file for adaboost classifier training
save('features_test.mat','features','-mat','-v7.3');
```

```
%<<<< Harr-like Feature Extraction for One Image >>>>%

function imgfeatures = getHaarFeaturesSub(intgImg)

imgfeatures = zeros(75286, 1);
imgW = 40;
imgH = 20;

% Notes: 1) the base size windows are ignored (2x1 and 1x3)
% 2) windows that contain the boundary pixel are ignored

w = 1;  h = 2; % initial base window size
count = 0;
for i = 2:imgW/w   % extend the size of one rectangular
for j = 2:imgH/h
    for x = 2:imgW-i*w   % slide the window through the image
    for y = 2:imgH-j*h
        firstRect = [y, x, i, j];
        secondRect = [y+j, x, i, j];
        count = count+1;
        imgfeatures(count) = -
sumRect(intgImg,firstRect)+sumRect(intgImg,secondRect);
    end
    end
end
end

w = 3;  h = 1;
for i = 2:imgW/w   % extend the size of one rectangular
for j = 2:imgH/h
    for x = 2:imgW-i*w   % slide the window through the image
    for y = 2:imgH-j*h
        firstRect = [y, x, i, j];
        secondRect = [y, x+i, i, j];
        thirdRect = [y, x+2*i, i, j];
        count = count+1;
        imgfeatures(count) = -
sumRect(intgImg,firstRect)+sumRect(intgImg,secondRect)-
sumRect(intgImg,thirdRect);
    end
    end
end
end



%<<<< Load Image Database >>>>%


function [Images] = readImageDB(dbPath, height, width, noImage, indOffset,
zeros)

Images = ones(height,width,noImage);

for ind = indOffset+1 : indOffset+noImage
```

```
    indstr = num2str(ind);
    if noImage<1000
        if ind<10
            indstr = ['00',num2str(ind)]; end
        if ind<100 && ind>9
            indstr = ['0',num2str(ind)]; end
    end
    if noImage>=1000
        if ind<10
            indstr = ['000',num2str(ind)]; end
        if ind<100 && ind>9
            indstr = ['00',num2str(ind)]; end
        if ind<1000 && ind>99
            indstr = ['0',num2str(ind)]; end
    end

    filename = [dbPath,zeros,indstr,'.png'];
    img = imread(filename);

    grayimg = double(rgb2gray(img));
    Images(:,:,ind-indOffset) = grayimg;

end



function [rectsum] = sumRect(intgImg, fourCorners)

% given four corner points in the integral image
% to calculate the sum of pixels inside the rectangular.

row_start = fourCorners(1);
col_start = fourCorners(2);
width = fourCorners(3);
height = fourCorners(4);

one = intgImg(row_start, col_start);
two = intgImg(row_start, col_start+width);
three = intgImg(row_start+height, col_start);
four =  intgImg(row_start+height, col_start+width);

rectsum = four + one - (two + three);



function [I] = integralImg(img)

[M,N] = size(img);
I = zeros(M+1,N+1);

% calculate the integral image of a window
I(2:M+1,2:N+1) = cumsum(cumsum(double(img)),2);
```

2) Training Process

```
% <<<<< Training Main Function >>>>> %

clc
clear all
close all

posTrainSize = 710;
negTrainSize = 1758;
noTrain = posTrainSize+negTrainSize;

% load pre-compute features
file = load('features_train.mat');
featuresTrain = file.features;
noFeatures = 75286;
clear file


S = 20;
nextInd = 1:noTrain;

for stage = 1:S

    nextInd = adaboost (featuresTrain, nextInd, stage);

    L = length(nextInd);
    if L==posTrainSize;
        break
    end

end
```

One stage Adaboost strong classifier

```
% <<<<< One stage adaboost strong classifier >>>>> %

function nextIndAll = adaboost (featuresAll, indAll, stage)

    fprintf('Stage = %d\n',stage);

    % update negative samples size
    posTrainSize = 710;
    negTrainSize = length(indAll)-710;
    noTrain = posTrainSize+negTrainSize;

    % update feature set throwing away recognized negative samples
```

```
    noFeatures = 75286;
    features = featuresAll(:,indAll);


    % initial weights equally among samples
    weights = zeros(noTrain,1);
    labels = zeros(noTrain,1);
    for i=1:noTrain
        if i<=posTrainSize
            weights(i)=0.5/posTrainSize;
            labels(i)=1;
        else
            weights(i)=0.5/negTrainSize;
        end
    end


    T = 40;    % cascade iteration times
    truepos_rate = zeros(T,1);
    falsepos_rate = zeros(T,1);


    % parameters of weak classifiers: minerror, threshold, polarity, feature
index
    ht = zeros(4,T);
    % classification result of t th weak classifier: 1 for postive, 0 for
negative
    ht_result = zeros(noTrain,T);
    % weight factor of weak classifiers
    alpha = zeros(T,1);
    % cascaded strong classifier results
    strLearner_result = zeros(noTrain,1);

    for t=1:T

        % normalized weights (weights updated on each iteration)
        weights = weights./sum(weights);

        % select a weak classifier given current weights
        [minerr,polarity,threshold,featInd,classifyResults] =
trainWeakLearner(features,weights,labels,noFeatures,posTrainSize,noTrain);
        ht(1,t) = minerr;
        ht(2,t) = polarity;
        ht(3,t) = threshold;
        ht(4,t) = featInd;
        ht_result(:,t) = classifyResults;

        % updates weights of each image
        beta = minerr /(1-minerr);
        weights = weights.*beta.^(1-xor(labels,classifyResults));

        % strong classifier decision value by cascading several weak
        % classifier
        alpha(t) = log(1/beta);
        strLearner = ht_result(:,1:t)*alpha(1:t,:);
```

```
        % set the strong classifier threshold to make sure all the positive
        % image recoganized as positive
        threshold_s = min(strLearner(1:posTrainSize));

        % obtain strong classifier results
        for i = 1:noTrain
            if strLearner(i) >= threshold_s
                strLearner_result(i) = 1;
            else
                strLearner_result(i) = 0;
            end
        end

        % compute true-positive & false-positive rate
        truepos_rate(t)  =  sum(strLearner_result(1:posTrainSize))/posTrainSize;
        falsepos_rate(t) =
sum(strLearner_result(posTrainSize+1:end))/negTrainSize;

        % condition that stops adding weak classifier
        if truepos_rate(t)==1 && falsepos_rate(t)<=0.5
            break;
        end

        fprintf('t = %d\n',t);

    end



    % all positive samples and negative samples recognized as positive
    % will be used in the next iteration
    [temp nextNegInd]=sort(strLearner_result(posTrainSize+1:end));
    for i=1:negTrainSize
        if temp(i)>0  % negative sample missclassified as 1
            nextNegInd = nextNegInd(i:end);
        break;
        end
    end
    nextIndAll = [1:posTrainSize,posTrainSize+nextNegInd];

    save(['ht_',num2str(stage),'.mat'],'ht','-mat','-v7.3');
    save(['strLearner_',num2str(stage),'.mat'],'strLearner','-mat','-v7.3');
    save(['threshold_s_',num2str(stage),'.mat'],'threshold_s','-mat','-v7.3');
    save(['falsepos_rate_',num2str(stage),'.mat'],'falsepos_rate','-mat','-
v7.3');
    save(['nextIndAll_',num2str(stage),'.mat'],'nextIndAll','-mat','-v7.3');


    % plot false positive rate of one stage training
    noCasc = 11;
    fp_rate = zeros(noCasc,1);
    for i=1:noCasc
        fp_rate = falsepos_rate(1:i);
        for j=2:i
            if fp_rate(j)==0
```

```
                fp_rate(j)=fp_rate(j-1);
                break;
            end
        end
        falsepos_acc = cumprod(fp_rate);
    end

    ind = 1:noCasc;
    figure (stage)
    plot(ind,falsepos_acc);
    title(['False-positive rate for ',num2str(stage),' stage Adaboost by
adding weak classifier'])
    xlabel(['t = 1:',num2str(noCasc)]);
    ylabel('false-positive rate');
    grid;


end
```

Select one weak classifier for Adaboost cascading

```
% <<<<< Select best weak classifer for one feature over all images >>>>> %
% given the current weight of training data,
% select the best weak classifer for each iteration

function [minerr,polarity,threshold,featInd,bestresult] =
trainWeakLearner(features,weights,labels,noFeatures,posTrainSize,noTrain)

    minerr = inf;
    polarity = 0;
    threshold = 0;
    featInd = 0;
    bestresult = zeros(noTrain,1);


    T_plus = repmat(sum(weights(1:posTrainSize,1)),noTrain,1);
    T_minu = repmat(sum(weights(posTrainSize+1:noTrain,1)),noTrain,1);

    % <<<< search for the best weak classifer
    for i = 1:noFeatures

        % <<<< select the best threshold for one weak classifer

        % sort one features among all images
        onefeat = features(i,:);
        [sortonefeat, sortind] = sort(onefeat,'ascend');
        sortweights = weights(sortind);
        sortlabels = labels(sortind);
        S_plus = cumsum(sortweights.*sortlabels);
        S_minu = cumsum(sortweights) - S_plus;

        err_plus = S_plus+(T_minu-S_minu);
```

```
        err_minu = S_minu+(T_plus-S_plus);

        % select best threshold
        oneFeatErr = min(err_plus, err_minu);
        [onefeaterr, ind] = min(oneFeatErr);

        result = zeros(noTrain,1);

        % classification result under best threshold
        if err_plus(ind)<=err_minu(ind)
            polar = -1;
            result(ind+1:end) = 1;
            result(sortind) = result;
        else
            polar = 1;
            result(1:ind) = 1;
            result(sortind) = result;
        end

        % classification error for this feature
        % classifyErr = 1- sum(weights(labels==result));

        % get the parameters that minimize the classification error
        if onefeaterr < minerr
            minerr = onefeaterr;
            if(ind==1)
                threshold = sortonefeat(1)-0.5;
            elseif(ind==noTrain)
                threshold = sortonefeat(noTrain)+0.5;
            else
                threshold = (sortonefeat(ind-1)+sortonefeat(ind))/2;
            end
            polarity = polar;
            featInd = i;
            bestresult = result;
        end


    end
```

3) Testing Process

```
clc
clear all
close all


posTestSize = 178;
negTestSize = 440;
noTest = posTestSize+negTestSize;
```

```
% load pre-compute features
file = load('features_test.mat');
featuresTest = file.features;
noFeatures = 75286;
clear file


S = 10;
falsepos_rate = zeros(S,1);
falseneg_rate = zeros(S,1);


for stage = 1:S

    fprintf('Test Stage = %d\n',stage);

    stage =
    file = load(['ht_',num2str(stage),'.mat']);
    ht = file.ht;
    [m,n]=size(ht);
    count=0;
    for i=1:n
        if ht(1,i)==0;
            break;
        end
        count=count+1;
    end
    count
    alpha = ht(1,1:count);
    polarity = ht(2,1:count);
    threshold = ht(3,1:count);
    featInd = ht(4,1:count);

    result = classifier (featuresTest, alpha, polarity, threshold, featInd,
count);

    falseneg_rate(stage) = (posTestSize-
sum(result(1:posTestSize)))/posTestSize;
    falsepos_rate(stage) = sum(result(posTestSize+1:end))/negTestSize;

end


    % plot result of training
    noStage = 11;
    fp_rate = zeros(noStage,1);
    for i=1:noStage
        fp_rate = falsepos_rate(1:i);
        for j=2:i
            if fp_rate(j)==0
                fp_rate(j)=fp_rate(j-1);
                break;
            end
```

```
        end
        falsepos_acc = cumprod(fp_rate);
    end
    fn_rate = zeros(noStage,1);
    for i=1:noStage
        fn_rate = falseneg_rate(1:i);
        for j=2:i
            if fn_rate(j)==0
                fn_rate(j)=fn_rate(j-1);
                break;
            end
        end
        falseneg_acc = cumprod(fn_rate);
    end

    ind = 1:noStage;
    figure (1)
    plot(ind, falsepos_acc);
    title(['False-positive rate for ',num2str(stage),' Adaboost stages'])
    xlabel(['stage = 1:',num2str(noStage)]);
    ylabel('false-positive rate');
    grid;
    figure (2)
    plot(ind, falseneg_acc);
    title(['False-negative rate for ',num2str(stage),' Adaboost stages'])
    xlabel(['stage = 1:',num2str(noStage)]);
    ylabel('false-negative rate');
    grid;



% <<<<< Perform Classifier on Test Images >>>>>%

function strLearner_result = classifier (featuresTest, alpha, polarity,
threshold, featInd, T)


    posTestSize = 178;
    negTestSize = 440;
    noTest = posTestSize+negTestSize;

    % classification result of classifier
    ht_result = zeros(noTest,T);
    strLearner_result = zeros(noTrain,1);

    for t=1:T

        featureValue = featuresTest(featInd(t));

        for j=1:noTest % for each image, compute decision of classifier
            if polarity*featureValue(t)<=polarity*threshold(t)
                ht_result(j,t)=1;
            end
        end
```

```
        strLearner = ht_result(:,1:t)*alpha(1:t,:);

        threshold_s = min(strLearner(1:posTrainSize));

        for i = 1:noTrain
            if strLearner(i) >= threshold_s
                strLearner_result(i) = 1;
            else
                strLearner_result(i) = 0;
            end
        end

        fprintf('t = %d\n',t);
    end

end
```