ECE661: Computer Vision (Fall 2014) Face Recognition and Object Detection with Cascaded AdaBoost Classification

Shaobo Fang: s-fang@purdue

December 16, 2014

Contents

1 Overview

This assignment consists of two main parts. The first part is PCA or LDA based face recognition. In the first part, we will perform dimension reduction to classify faces conveniently based on both features obtained using PCA and LDA. The results generated by PCA and LDA will be compared.

Secondly, we will implement Cascaded AdaBoost algorithm on car detection. In the car detection assignment, we will implement Haar filters to extract numerous weak classifiers, and then we will form stronger classifier in a cascaded pattern to eventually form a good classification scheme.

2 PCA: Principal Component Analysis

2.1 PCA Overview

The first important concept is the vectorized image. Assume we have an image window of 64 by 64, then the vectorized image vector for each image would have the size of 64 * 64. Usually, we consider the vectorized image to be a column vector $\vec{x}_i$. The mean image vector then could be defined accordingly:

$$\vec{m} = \frac{1}{N}\sum_{i=0}^{M-1}\vec{x}_i$$

And covariance matrix for all image vectors could then be defined as:

$$C = \frac{1}{N}\sum_{i=0}^{N-1}\{(\vec{x}_i - \vec{m})(\vec{x}_i - \vec{m})^T\}$$

Note the important property for the covariance matrix is that the diagonal elements of the covariance matrix C will be the averages of the squares of the individual pixel values and the off-diagonal elements will be the averages of the pairwise products of different pixels. Before we proceed, we first normalize the image vectors so that:

$$\vec{x}_i^T\vec{x}_i = 1$$

The K dimensional feature space is then defined as the eigenvectors corresponding to the largest K eigenvalues of the covariance matrix. After the feature space has been defined, we simply project the vectorized testing images onto the feature space. Note that, the dimension of features after projection will be hugely reduced. Assume the K dimension feature space is W_K:

$$W_K = [\vec{w}_0, \vec{w}_1, ..., \vec{w}_{K-1}]$$

The feature values then determined as:

$$\vec{y} = W_K^T(\vec{x} - \vec{m})$$

2.2 Computation of the Eigenvectors for PCA

Now, let us focus on the covariance matrix of the vectorized image.

$$C = \frac{1}{N}\sum_{i=0}^{N-1}\{(\vec{x}_i - \vec{m})(\vec{x}_i - \vec{m})^T\}$$

According to the equation, the size of the covariance matrix would be the defined as $m * n \ by \ m * n$, where m and n are the size of the size of a single image (image size m by n). For example if the image size is 64 by 64 then the size of covariance matrix would be 4096 by 4096. **This is computationally inefficiency.** To solve this problem, we propose a computational trick.

Let us reconsider, if for example we have N vectorized images:

$$\mathbf{x} = [\vec{x}_0 - \vec{m}, \vec{x}_1 - \vec{m}, ..., \vec{x}_{N-1} - \vec{m}]$$

Where $\vec{m}$ is the mean vector of all images, the $\mathbf{x}$ will have the size of m*n by N. Now, if based on previous section that $\vec{w}$ represents an eigenvector of $\mathbf{C}$, it must satisfy:

$$\mathbf{X}\mathbf{X}^T\vec{w} = \lambda\vec{w}$$

The trick here is instead of finding the eigenvector of $\vec{w}$ as defined in previous part, we find the eigenvectors $\vec{u}$ of matrix X^TX:

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

Note that X^TX will only have the size of N by N. **This is a great advantage for computation efficiency improvement as in general N by N matrix would be a lot easier to do eigenvalues decomposition compared to a covariance matrix of 4096*4096 or higher dimensions**. With our computational trick, the time training the data will be reduced significantly:

$$XX^TX\vec{u} = \lambda X\vec{u}$$
$$(XX^T)X\vec{u} = \lambda X\vec{u}$$
$$C(X\vec{u}) = \lambda(X\vec{u})$$
$$\vec{w} = X\vec{u}$$

Note that: The number of eigenvectors in feature WILL NEVER exceed the number of the images. To have better classification result, we always want to normalize image vectors and the features spaces before we conduct projection and classification based on euclidean distance.

At the very last, we will perform classification based on euclidean distance between each feature vectors. Assume we have N images in the training data then the features vectors obtained from training data is $v^{(i)}_{train}$. And M images in the testing data the feature vectors are: $v^{(j)}_{test}$

$$\hat{i} = \arg\min_{i\in\Omega}\{||v^{(i)}_{train} - v^{(j)}_{test}||\}$$

All above information have concluded PCA. ||

3 LDA: Linear Discriminant Analysis

3.1 LDA Overview

LDA is another classification method based on vectorized images. Unlike PCA, the goal of LDA is to maximally discriminating between the classes. Hence, we need to perform both between-class scatter and within-class scatter. Define $\vec{m}_i$ as class mean and $\vec{m}$ as global mean, we have:

$$S_B = \frac{1}{|\mathcal{C}|}\sum_{i=1}^{\mathcal{C}}(\vec{m}_i - \vec{m})(\vec{m}_i - \vec{m})^T$$

And:

$$S_W = \frac{1}{|\mathcal{C}|}\sum_{i=1}^{|\mathcal{C}|}\frac{1}{|\mathcal{C}_i|}\sum_{k=1}^{|\mathcal{C}_i|}(\vec{x}_k - \vec{m}_i)(\vec{x}_k - \vec{m}_i)^T$$

Now, assume that the feature space is defined as $\vec{w}$, then the **Fisher Discriminant Function** is given by:

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

It can be shown that a vector $\vec{w}$ that maximize the Fisher Discriminant Function must satisfy:

$$S_B\vec{w} = \lambda S_W \vec{w}$$

If S_W is non-singular, then the above problem can be translated into the following form:

$$S_W^{-1} S_B \vec{w} = \lambda \vec{w}$$

Please note that, as:

$$S_B = \frac{1}{|\mathcal{C}|}\sum_{i=1}^{\mathcal{C}}(\vec{m}_i - \vec{m})(\vec{m}_i - \vec{m})^T$$

Only $|\mathcal{C}| - 1$ in S_B is linearly independent.

3.2 Computational Issues Related to LDA

As we mentioned earlier: S_W **has to be non-singular** for the following equation to hold

$$S_W^{-1} S_B \vec{w} = \lambda \vec{w}$$

Now, in order to get rid of the issues it might have with singular matrix, we propose a more direct approach. For the more direct approach, in the Fisher Discriminant Function:

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

We want to:

$$\vec{w} = \arg\max_{\vec{w}} J(\vec{w})$$

The above equation could be solved by Yu and Yang's algorithm.

1. Use regular eigen decomposition to diagonalized S_B:

$$V^T S_B V = \Lambda \quad wheere \quad V^T V = I$$

2. Discard those eigen values in Λ that are close to 0.

3. Now, define D_B as the upper-left $M \times M$ sub-matrix of Λ:

$$Y^T S_B Y = D_B$$

4. Construct Matrix Z accordingly:

$$Z = Y D_B^{-0.5}$$

5. Diagonalized $Z^T S_W Z$ as:

$$U^T Z^T S_W Z U = D_W \quad where \quad U^T U = I$$

6. **Discard the largest eigenvalues of D_W and drop the eigenvectors associated with those eigen values**

7. At the very last, the feature space is defined as:

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

After we obtain the feature space the rest is similar as we have done in PCA. Classification again is based on euclidean distances of feature vectors and is defined as: $v_{test}^{(j)}$

$$\hat{i} = \arg\min_{i \in \Omega} \{||v_{train}^{(i)} - v_{test}^{(j)}||\}$$

4 PCA and LDA Result Comparison

We will plot the accuracy of both PCA and LDA with regard to the the dimensionality of feature space p.

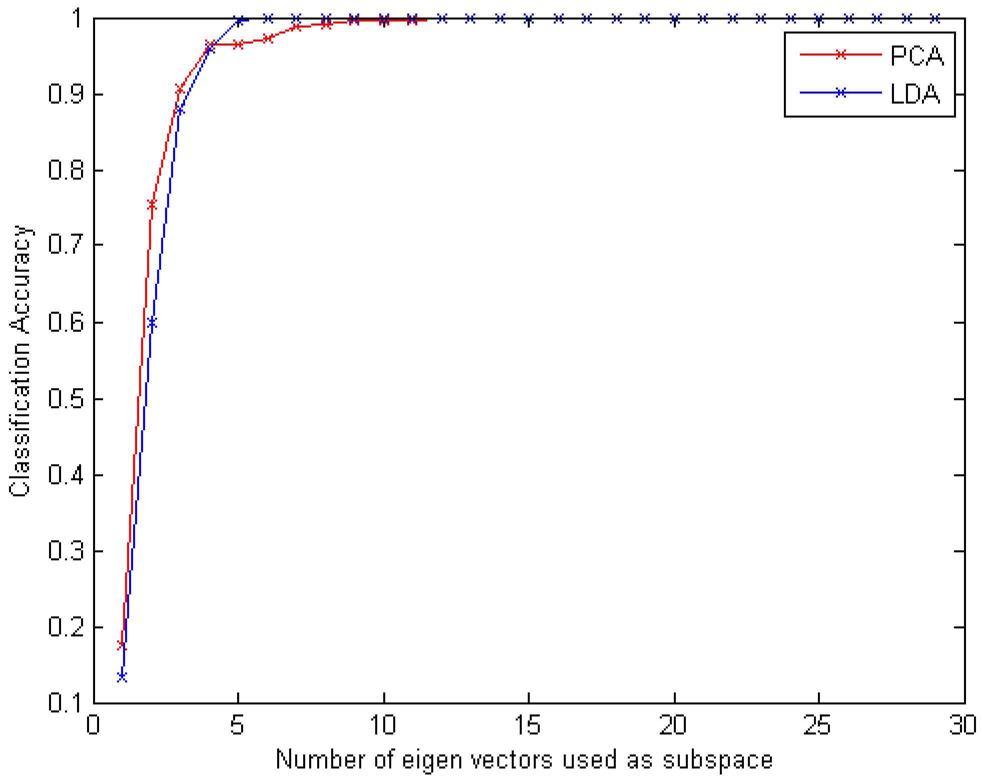

Fig 1. The accuracy (y-axis) Comparison of PCA and LDA with different dimensionality p (x-axis).

$$accuracy = \frac{number\ of\ test\ images\ correctly\ classified}{total\ number\ of\ test\ images}$$

Conclusions of comparison of PCA and LDA:

1. Both PCA and LDA have very good performance. Accuracy for PCA converge to 1 in 13 iterations while accuracy for LDA converge to 1 in 5 iterations
2. LDA converge to 100% in a faster rate
3. Computationally efficiency wise, we will use LDA as it requires less iterations to converge to 100% accurate.

`Matlab Code used in this section please refer to the .zip file`

5 Object Detection with Cascaded AdaBoost Classification

AdaBoost is an iterative method that form strong classifiers based on enormous weak classifiers. The real challenge of implementing AdaBoost would be the computation efficiency. To extract those weak features from the images, we will implement Haar-like filters (of various size) and sweeping through the entire image. Thus, we will obtain 10240 features from each image and the features will be store as a vector. The Haar-like filters we used are the two most fundamental ones, as illustrated as below:

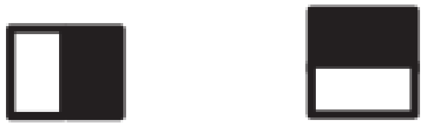

Fig 2. The Haar-like filters used to extract enormous features. (Feature is a value that is obtained by subtracting the white areas from black areas)

One may argue that the features extracted using Haar-like filters does not make any sense (too weak). However that is the core of AdaBoost Algorithm: To build a strong classifier based on features that does not make a lot of sense if on their own.

Now, let us define the weak classifier:

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

A weak classifier consists of a feature f, a threshold value θ and a polarity p. Very likely, the best weak classifier that is still not good enough on the training data. Thus, let us consider the T hypothesis boosting algorithm.

The boosting algorithm has an iterative structure and thus we need to update the weighted factor $w_{t,i}$ for each image i at each iteration t. Assume a strong classifier would be constructed using T stages. (T weak classifiers). Based on the `Viola and Jones Paper`, the implementation of boosting algorithm is as followed:
First of all, initialize the weighted factor. Assume in the training data there is m number of negative images and there is l number of positive images. Then $w_{1,i} = \frac{1}{2m} \ or \ \frac{1}{2l}$.

Then $for \ t = 2, ..., T$:

1. Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$

2. Select the best weak classifier with respect to the weighted error
$$\epsilon_t = \min_{f,p,\theta} \sum_i w_i |h(x+i,f,p)|$$
3. Define $h_i(x) = h(x, f_t, p_t, \theta_t)$ where f_t, p_t and θ_t are the minimizer of ϵ_t
4. **VIP** Update the weights:
$$w_{1+i,i} = w_{t,i}\beta_t^{1-\epsilon_i}$$
   Where $e_i = 0$ if example x_i is classified correctly, $e_t = 1$ otherwise, and $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

After the iterations have been done, the strong classifier observed is defined as:

$$C(x) = \begin{cases} 1 & \text{if} \quad \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2}\sum_{t=1}^{T}\alpha_t \\ 0 & \text{otherwise} \end{cases}$$

Where $\alpha_t = \log\frac{1}{\beta_t}$.
According to `Viola and Jones` paper, the AdaBoost optimal threshold can be computed in a single pass over this sorted list. For each element in the sorted list, four sums are maintained and evaluated: the total sum of positive example weights T^+, the total sum of negative example weights T^-, the sum of positive weights below the current examples S^+ and the sum of the negative weights below the current examples S^-.

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

False Positive Rate

$$f_p = \frac{number\ of\ misclassified\ negative\ test\ images}{number\ of\ negative\ test\ images}$$

False Negative Rate

$$f_n = \frac{number\ of\ misclassified\ positive\ test\ images}{number\ of\ positive\ test\ images}$$

Finally, after obtaining several strong classifier, the attentional cascade method is defined as below:

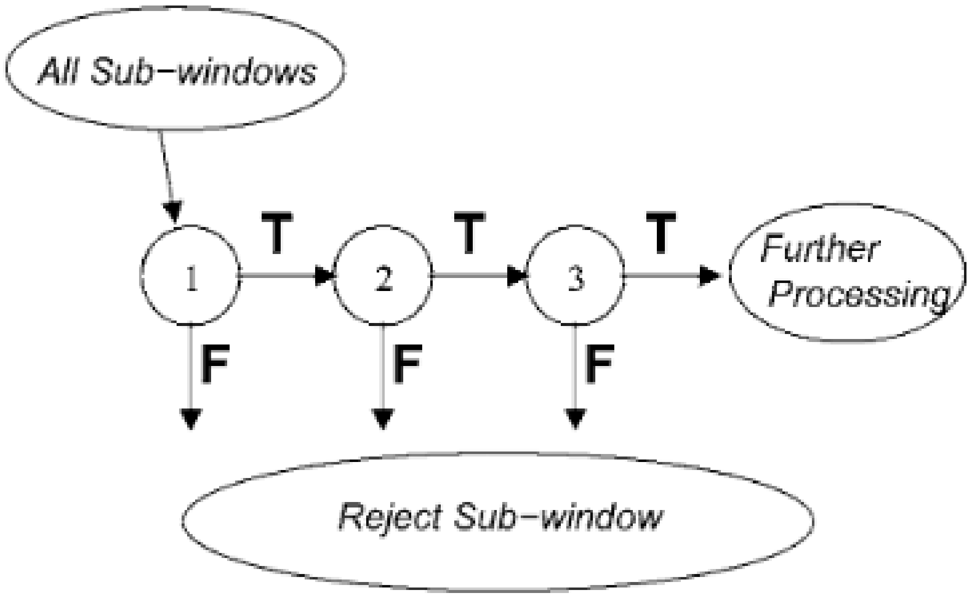

Fig 3. Illustration of Cascade Method (From `Viola and Jones`)

Results for AdaBoost as example of 1 strong classifier.

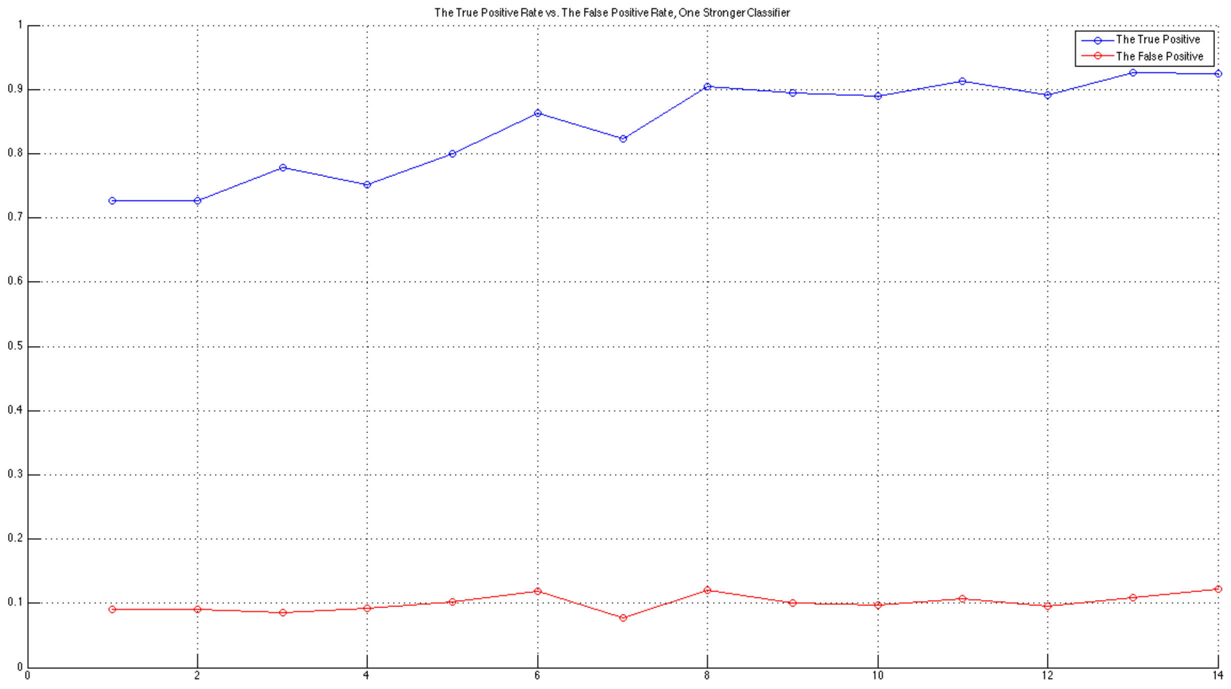

Fig 4. The True Positive rate and the False Positive Rate For Training Data

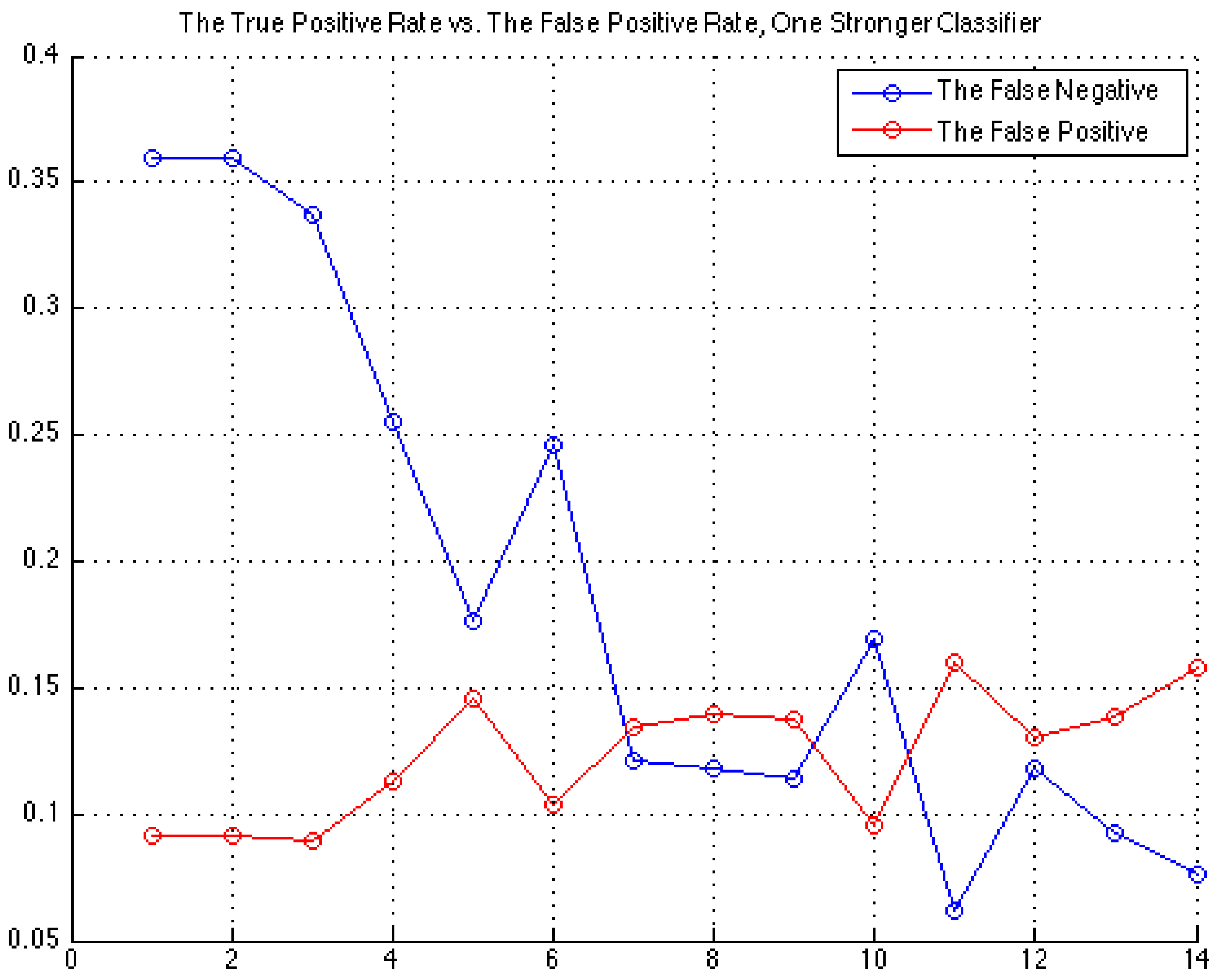

Fig 5. The False Negative Rate and False Positive Rate on Testing Data

The True Difficult Part for AdaBoost is iterative and EFFICIENT implementation of the algorithm. We have a huge issue in the implementation even for 10240 features. We need to update the weighted factor and that process is really lengthy. In the future a better algorithm should have been designed.

`Matlab Code Used in This Section Please Refer To .zip file`