Homework Three

Yibo Yang

Department of Statistics, University of Chicago, IL 60637, USA.

Abstract

In this Latex are the answers regarding to Homework Three of the class STAT 37710 Machine Learning.

Keywords: Adaboosting, Facial detection, Classification

1. Choice of feature

Due to the computational speed of this program, I cannot use all the features around 4000000 in this program. For testing the code, I use 14848 features to get a reasonable result where the feature are 4×8 for smallest and 8×16 for largest. After that, I increase the feature to 118272. Each footstep of constructing the feature is 2, which means I reduce the 64×64 picture into a 32×32 picture that is a change that maybe acceptable.
To sum up, I computed 14848 Haar-like features for all 4000 images and used them as basis of features.

2. Approach for computing

As the training pictures are all of the size 64×64, I use the 65×65 frame to mark the boundary. This would help me to specify the computation of the value of the features.
Instead of using the trick professor gives, I use the *numpy.mean* function to compute the value and speed up the computational process greatly.
As I find there are many cases that two faces maybe very close to each other, I consider to set some tolerance for the overlap that should help us to find the faces more precisely. So, I set another matrix called $Indicator$ has the same size with this final testing picture that is 1280×1600 points and each of the entry I set its value as 1. If I find it is a face, I would set all value

of *Indicator* at this area to be 0. And every time I want to detect a new area, I would first calculate the number of 1 in this area, if the sum is over Tolerance $\times$ 64 $\times$ 64, I would test whether it is a face or not. Otherwise, I would skip this area and continue with the next one. Typically, I accept that the face could 20% overlap area, but this coefficient could be also changed to 10% and 5% or some number else.
Once I found the face, I would change the value of the frame of the face's grayscale to 255 which is white. So I could mark the boundary of the face in the original picture.
As the domain near a face could be also tested as a face and the way I am doing the test is from top to bottom and from left to right, I would move 2 step to mark the face each time I find a face. This would help me the mark the more precisely.

3. Training process

Apply AdaBoost Algorithm to select a small set of features and train the classifier. The codes are attached in the appendix.
I run the code and for each strong classifier, each time I get a weak classifier, I would pause and let the weak learners I got together to be a strong classifier and calculate its Θ.
This strong classifier is as:

$$H(x) = sgn(\sum_{t=1}^{T} \alpha_t h_t(x) - \Theta) \tag{1}$$

Then I use all the weak learners I got in this new strong classifier to calculate the false positive rate and compare it with 30%. If it is smaller than 30%, the round of this strong classifier would stop and calculate the Θ for this classifier. Then I re-manage the training data by removing the background that are correctly classified and go on to calculate the next strong classifier. For this problem I set $N = 4$ as the number of the strong classifiers to do the cascade.
After all, I derive the final classifier:

$$H_i(x) = sgn(\sum_{t=1}^{T} \alpha_t h_t(x) - \Theta) \quad i = 1, ..., N \tag{2}$$

I use the $H_i(x)$ together to test the final testing figure.

3.1. First Trail

I just use one strong classifier(without cascade) to have a try on the testing figure to see the performance of the algorithm. The result is in fig.1:
The false positive rate vs the number of weak classifier is in fig.2. We can

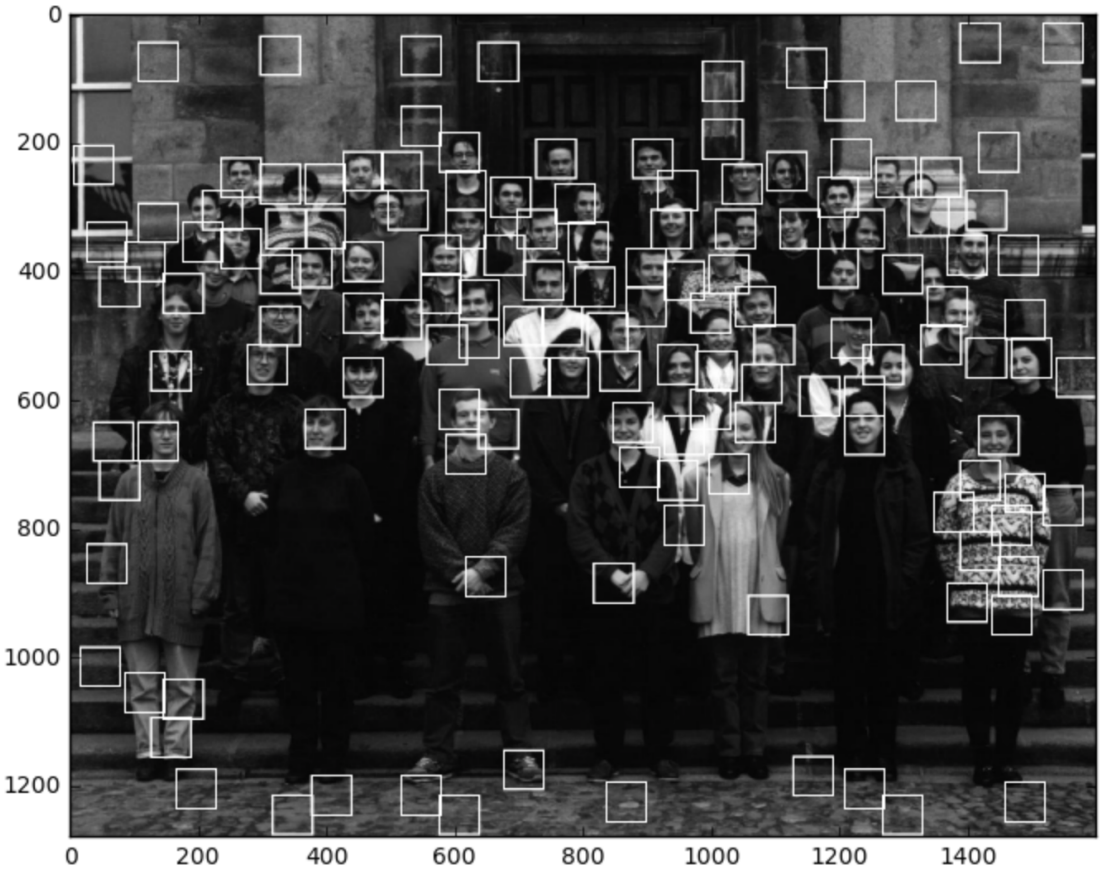

Figure 1: First Trail of the facial detection

see the false positive rate would be under 10% as the number of weak learner larger than 16. (Here I should point out that I do not calculate the false positive rate until the number of weak learners is over 3 because the false positive rate should be large than 30% due to the small number of the weak

learners. That is why the false positive rate are all equal to 1 for the first weak learners)

The computational time cost vs the number of weak classifier is in fig.3:

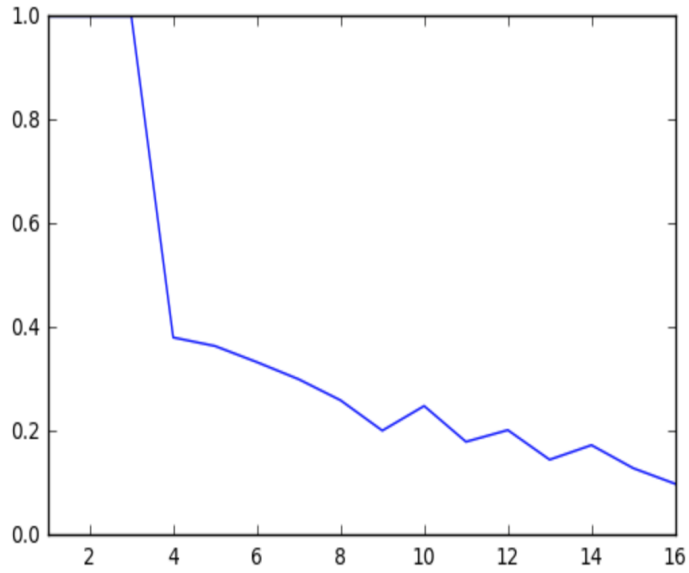

Figure 2: False positive rate vs the number of weak classifier

We can see from fig.2, almost all the faces are standing out in this algorithm. But there still are many back grounds counted as faces. In the next section, we can see that this is improved by adding in more strong classifiers.

3.2. Final test

Run the final version of code in the appendix. I can derive 3 strong classifiers and each of them has the number of weak classifiers of 9 for the first one, 19 for the second one and 30 for the third one and 23 for the last one. The coefficients I derive is as following in table.1-table.4. The whole procedure takes about 40 mins for calculating 14000 features as total.

For the first strong classifier:

We get the $\Theta_1 = -2.32$

For the second strong classifier:

We get the $\Theta_2 = -1.903$

Table 1: First strong classifier

θ	p	α	position
6.775	1	1.146	[16,2,20,2,20,62,24,62]
5.925	1	0.695	[0,24,0, 28, 20, 28, 20, 32]
6.270	1	0.588	[0, 24, 0, 28, 20, 28, 20, 32]
4.063	1	0.563	[0, 0, 0, 4, 4, 4, 4, 8]
9.742	1	0.465	[12, 40, 16, 40, 16, 56, 20, 56]
-0.452	-1	0.420	[30, 12, 34, 12, 34, 64, 38, 64]
3.563	1	0.436	[0, 12, 0, 16, 4, 16, 4, 20]
3.5	1	0.414	[30, 40, 30, 44, 42, 44, 42, 48]
2.323	1	0.388	[0, 18, 0, 22, 12, 22, 12, 26]

Table 2: Second strong classifier

θ	p	α	position
12.067	1	0.728	[16., 10., 20., 10., 20., 62., 24., 62.]
6.477	1	0.540	[2., 22., 2., 26., 18., 26., 18., 30.]
3.675	1	0.487	[0, 24, 0, 28, 20, 28, 20, 32]
10.641	1	0.501	[20., 36., 24., 36., 24., 56., 28., 56.]
9.578	1	0.469	[12., 26., 12., 30., 20., 30., 20., 34.]
-0.218	-1	0.436	[12., 44., 16., 44., 16., 52., 20., 52.]
5.459	1	0.394	[32., 2., 36., 2., 36., 18., 40., 18.]
3.968	1	0.345	[0., 0., 0., 4., 40., 4., 40., 8.]
-0.762	-1	0.375	[26., 40., 26., 44., 38., 44., 38., 48.]
14.515	1	0.383	[30., 0., 34., 0., 34., 64., 38., 64.]
6.375	1	0.345	[14., 8., 18., 8., 18., 16., 22., 16.]
3.802	1	0.392	[0., 8., 0., 12., 4., 12., 4., 16.]
5.198	1	0.360	[18., 18., 22., 18., 22., 54., 26., 54.]
8.031	1	0.354	[54., 32., 58., 32., 58., 44., 62., 44.]
-1.650	-1	0.386	[50., 28., 54., 28., 54., 32., 58., 32.]
2.843	1	0.349	[2., 20., 6., 20., 6., 40., 10., 40.]
-0.187	-1	0.373	[24., 14., 28., 14., 28., 18., 32., 18.]
20.078	1	0.350	[36., 46., 40., 46., 40., 62., 44., 62.]
-1.288	-1	0.366	[10., 10., 14., 10., 14., 18., 18., 18.]

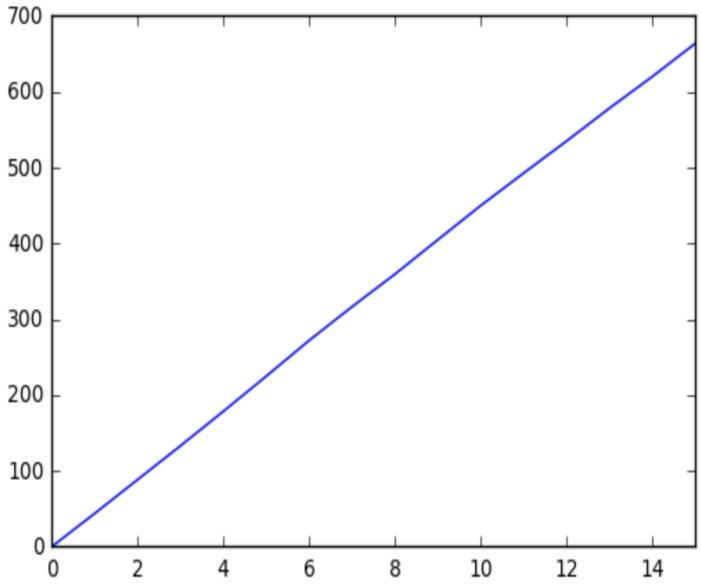

Figure 3: Computational time cost vs the number of weak classifier

For the third strong classifier:
We get the $\Theta_3 = -1.796$
For the forth strong classifier:

We get the $\Theta_4 = -2.106$
Here we should point out as we do not have the fifth strong classifier, we consider set $\Theta_4 = 0$ to do the testing.
The final version of this project is in fig.4 where I choose the tolerance of overlapping is 80%. We can see it performs better than the previous figures.

4. For the other types of features

4.1. Use two-rectangle and three-rectangle features

I try to use more features into this model that are the other two of the three distinct types of Haarlike features. The three rectangle feature is added and the result is shown in fig.5. Here I do not use equal area to calculate the feature. For simplicity, I just equally separate the rectangular into four parts and the middle two combine to be black and the outer together to be

Table 3: Third strong classifier

θ	p	α	position
10.796	1	0.599	[18., 4., 22., 4., 22., 60., 26., 60.]
7.593	1	0.548	[2., 24., 2., 28., 18., 28., 18., 32.]
5.868	1	0.467	[14., 34., 18., 34., 18., 54., 22., 54.]
22.697	1	0.452	[14., 14., 18., 14., 18., 26., 22., 26.]
-0.170	-1	0.448	[30., 2., 34., 2., 34., 62., 38., 62.]
3.031	1	0.438	[0., 16., 0., 20., 12., 20., 12., 24.]
3.500	1	0.424	[0., 4., 0., 8., 4., 8., 4., 12.]
2.908	1	0.386	[22., 2., 26., 2., 26., 58., 30., 58.]
-0.895	-1	0.371	[34., 0., 38., 0., 38., 12., 42., 12.]
3.593	1	0.385	[34., 38., 34., 42., 42., 42., 42., 46.]
9.656	1	0.355	[48., 38., 52., 38., 52., 42., 56., 42.]
0.083	-1	0.437	[0., 16., 4., 16., 4., 40., 8., 40.]
9.812	1	0.336	[8., 28., 8., 32., 12., 32., 12., 36.]
-1.437	-1	0.335	[0., 32., 0., 36., 64., 36., 64., 40.]
-0.75	-1	0.344	[30., 54., 30., 58., 34., 58., 34., 62.]
4.640	1	0.321	[12., 48., 12., 52., 20., 52., 20., 56.]
2.755	1	0.336	[14., 14., 18., 14., 18., 62., 22., 62.]
6.710	1	0.364	[48., 8., 48., 12., 64., 12., 64., 16.]
3.773	1	0.332	[20., 46., 24., 46., 24., 62., 28., 62.]
1.993	1	0.336	[0., 18., 0., 22., 20., 22., 20., 26.]
8.093	1	0.307	[52., 22., 56., 22., 56., 26., 60., 26.]
-13.343	1	0.337	[0., 12., 0., 16., 4., 16., 4., 20.]
9.875	1	0.309	[16., 28., 16., 32., 32., 32., 32., 36.]
-1.645	-1	0.330	[22., 52., 22., 56., 34., 56., 34., 60.]
2.0468	1	0.323	[2., 22., 2., 26., 18., 26., 18., 30.]
0.052	-1	0.277	[0., 2., 4., 2., 4., 62., 8., 62.]
17.020	1	0.301	[38., 28., 42., 28., 42., 40., 46., 40.]
0.198	-1	0.346	[0., 56., 0., 60., 56., 60., 56., 64.]
-1.687	-1	0.292	[58., 48., 58., 52., 62., 52., 62., 56.]
0.773	1	0.298	[18., 30., 22., 30., 22., 46., 26., 46.]

Table 4: Forth strong classifier

θ	p	α	position
6.870	1	0.728	[16., 0., 20., 0., 20., 28., 24., 28.]
19.570	1	0.540	[6., 24., 6., 28., 22., 28., 22., 32.]
-0.406	1	0.487	[22., 46., 26., 46., 26., 50., 30., 50.]
-0.117	1	0.501	[0., 16., 0., 20., 16., 20., 16., 24.]
0.177	-1	0.469	[30., 40., 34., 40., 34., 64., 38., 64.]
1.911	1	0.436	[22., 0., 22., 4., 46., 4., 46., 8.]
6.945	1	0.394	[4., 22., 4., 26., 20., 26., 20., 30.]
12.901	1	0.345	[4., 26., 4., 30., 32., 30., 32., 34.]
-5.848	1	0.375	[20., 36., 24., 36., 24., 60., 28., 60.]
26.437	1	0.383	[12., 46., 16., 46., 16., 50., 20., 50.]
-5.937	1	0.345	[14., 38., 18., 38., 18., 42., 22., 42.]
-0.831	-1	0.392	[54., 4., 58., 4., 58., 24., 62., 24.]
-0.822	1	0.360	[56., 26., 60., 26., 60., 38., 64., 38.]
-0.015	-1	0.354	[2., 42., 6., 42., 6., 50., 10., 50.]
0.338	1	0.386	[22., 28., 22., 32., 46., 32., 46., 36.]
-9.906	1	0.349	[40., 60., 44., 60., 44., 64., 48., 64.]
0.726	-1	0.373	[36., 46., 40., 46., 40., 62., 44., 62.]
6.218	-1	0.350	[2., 48., 2., 52., 6., 52., 6., 56.]
7.515	1	0.366	[8., 16., 8., 20., 16., 20., 16., 24.]
-6.031	1	0.366	[14., 24., 18., 24., 18., 28., 22., 28.]
10.878	1	0.366	[18., 6., 22., 6., 22., 42., 26., 42.]
-0.656	-1	0.366	[6., 22., 10., 22., 10., 26., 14., 26.]
0.928	-1	0.366	[8., 12., 8., 16., 52., 16., 52., 20.]

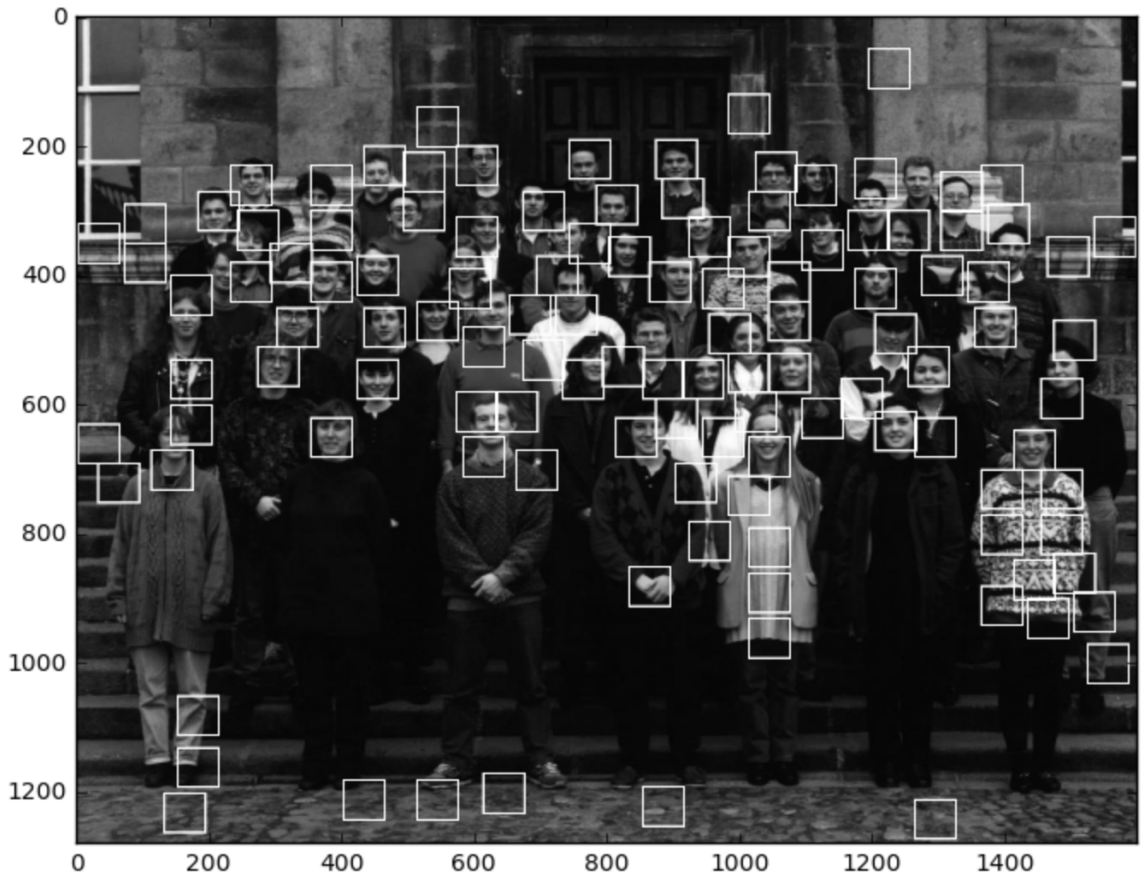

Figure 4: Final version of the facial detection using only simplest two-rectangle features

white. We can take a look at the white dress that the girl is wearing in the first line. There are four "faces" on the dress. So, we may conclude that this method catches the most symmetric parts of the pictures. This features I guess is more powerful in catching the patterns like two-eye pattern and nose pattern.

4.2. Use all the features

I add all the features into the model and get the result in fig.6. To calculate the four-rectangle feature, we only need to store the location of three points with 6 values. We can see from this picture that all the faces are nearly found and the backgrounds which are misclassified as faces is declined.

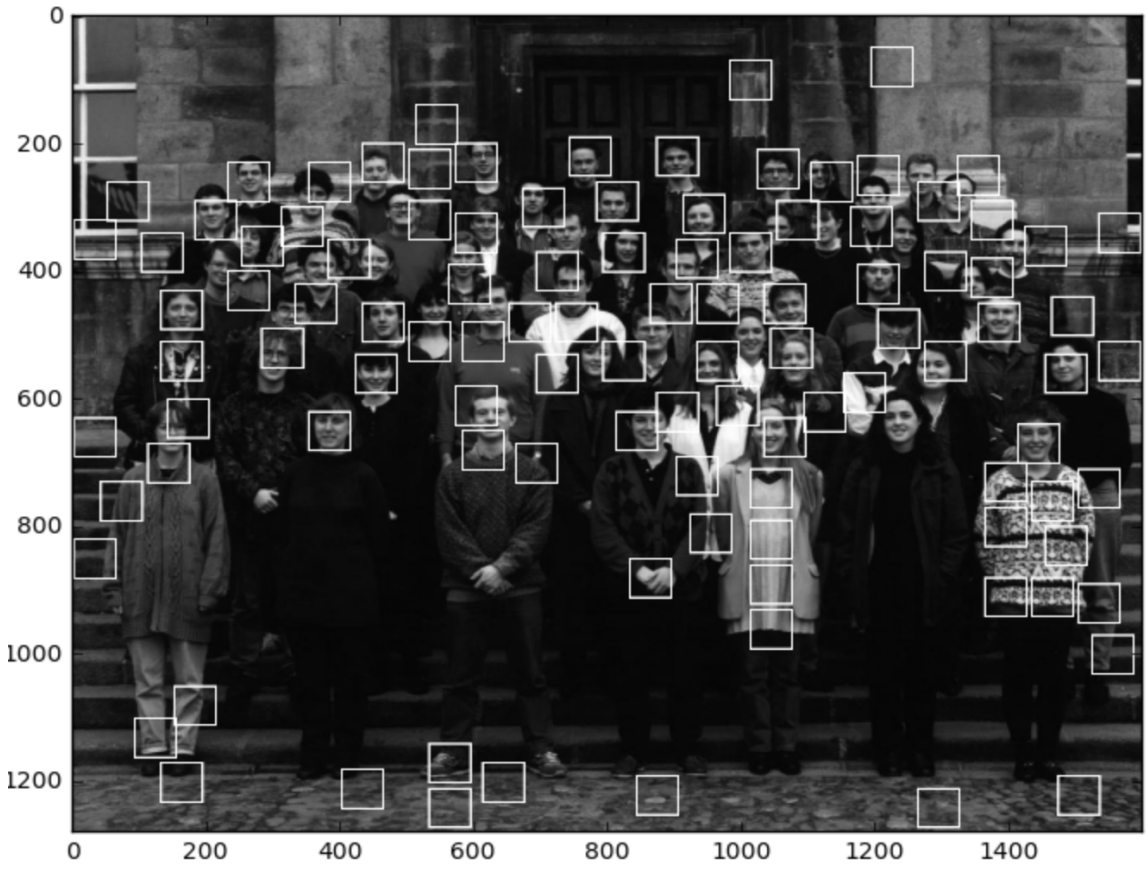

Figure 5: Final version of the facial detection using two-rectangle and three-rectangle features

So, we see the performance of using all the features.

5. Appendix

Here are the codes I use:
Read the training data:

```
import random
import pandas as pd
import numpy as np
```

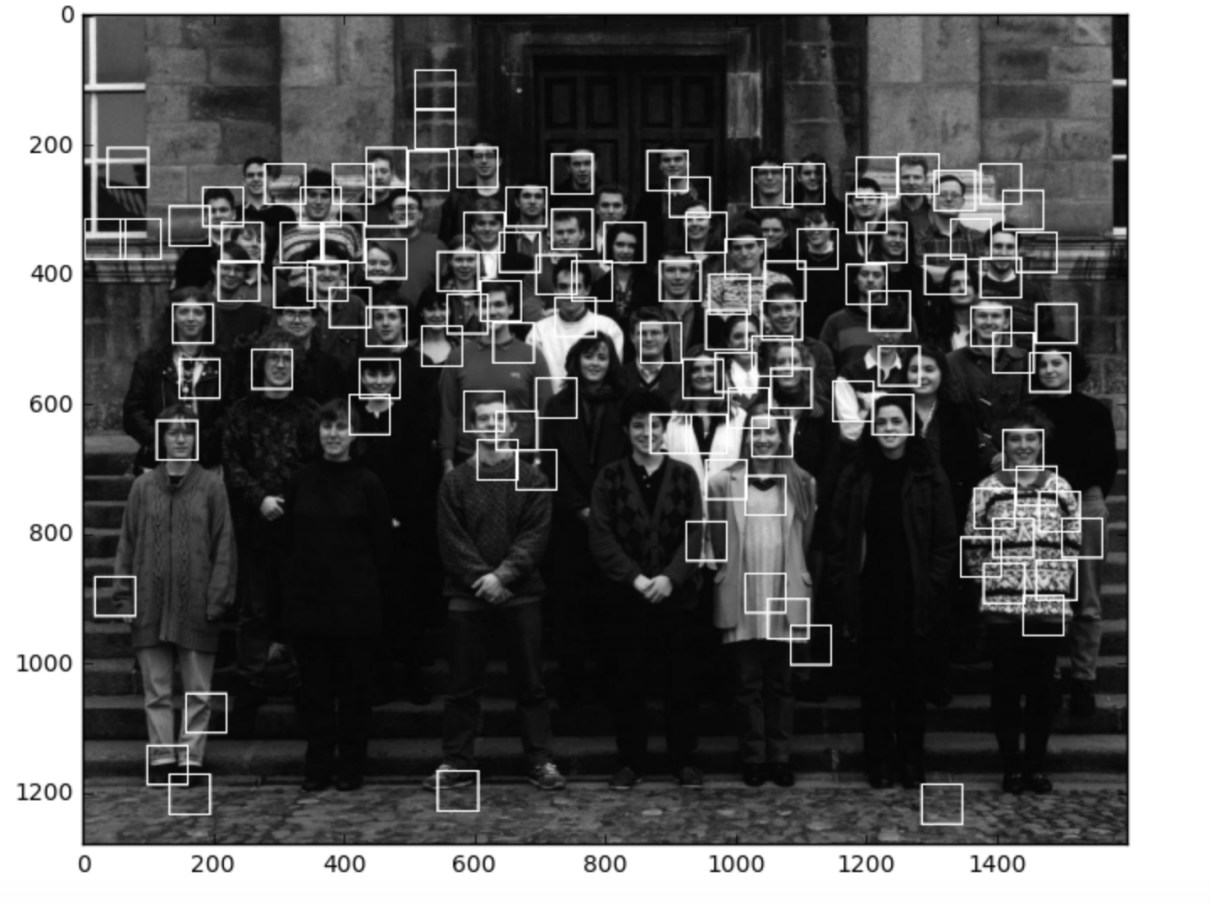

Figure 6: Final version of the facial detection using all Haarlike features

```
import matplotlib.pyplot as plt
from scipy import sparse
from PIL import Image
import pickle as pk

datatrain=np.zeros((4000,64,64))
for p in range(2000):
    newname = "/Users/yangyibo/Desktop/MachineLearning/hw3/faces/face"+s
    pil_im =Image.open(newname).convert('L')
    for m in range(64):
        for n in range(64):
```

```
                datatrain[p,n,m]= pil_im.getpixel((m,n))

for p in range(2000,4000):
    newname = "/Users/yangyibo/Desktop/MachineLearning/hw3/background/"-
    pil_im =Image.open(newname).convert('L')
    for m in range(64):
        for n in range(64):
            datatrain[p,n,m]= pil_im.getpixel((m,n))
```

Generate the features:

```
def FeatureTBL(minbound1,minbound2,maxb1,maxb2,step1,step2,size,n):
    x1 = [] ; y1 = []
    x2 = [] ; y2 = []
    x3 = [] ; y3 = []
    x4 = [] ; y4 = []

    for i in step1 * np.array(range((n+1)/step1)):
        tail = i
        for j in range((n-i)/(2*size)):
            tail = tail + 2*size
            if (tail-i) >= minbound1 and tail<=n and (tail-i) <= maxb1:
                x1.append(i)
                x4.append(tail)
                x3.append((i+tail)/2)
                x2.append((i+tail)/2)

    for k in step2 * np.array(range((n+1)/step2)):
        tail = k
        for l in range((n-k)/size):
            tail = tail + size
            if (tail-k) >= minbound2 and tail<=n and (tail-i) <= maxb2:
                y1.append(k)
                y4.append(tail)
    y3=y4;y2=y1

    print(len(x1))
    print(len(y1))
    print(2*len(x1)*len(y1))
```

```
    ans = []
    for i in range(len(x1)):
        for j in range(len(y1)):
            l1 = [x1[i],y1[j],x2[i],y2[j],x3[i],y3[j],x4[i],y4[j]]
            l2 = [y1[j],x1[i],y2[j],x2[i],y3[j],x3[i],y4[j],x4[i]]
            #l3=[x1[i],y1[j],x2[i],y2[j],x1[i],(y1[j]+y2[j])/4,x2[i],(y
            #l4=[x1[j],y1[i],x2[j],y2[j],x1[j],(y1[i]+y2[i])/4,x2[j],(y
            #l5=[x1[i],y1[j],x2[i],y2[j],(x1[i]+x2[i])/2,(y1[j]+y2[j])/
            ans.append(l1)
            ans.append(l2)
    return(ans)
```

Calculate the value of an area:

```
def getvolume(ft):
    black = np.mean(datatrain[:, ft[0]:ft[4], ft[1]:ft[5]], axis=(1,2))
    white = np.mean(datatrain[:, ft[2]:ft[6], ft[3]:ft[7]], axis=(1,2))
    return -black + white
#def getvolumeTian(ft):
#    black = np.mean(datatrain[:,ft[0]:ft[4],ft[1]:ft[5]], axis=(1,2))+
#    white = np.mean(datatrain[:,ft[0]:ft[4],ft[5]:ft[3]], axis=(1,2))+
#    return -black + white
```

Compute the value of all features and save as FFunct:

```
FFunct=np.zeros((FeatN,4000))
for m in range(FeatN):
    FFunct[m,:]=getvolume(featuretbl[m,:])
```

Get one weak classifier's p, θ and $Error$:

```
def getPtheta(FFunctKK,Tweight,reference):
    Ind=np.argsort(FFunctKK)
    orderweight=Tweight[Ind]
    reference=reference*2-1
    orderlabel=reference[Ind]
    FFunctorder=np.sort(FFunctKK)
    Tzheng=np.sum((orderlabel>0)*orderweight)
```

```
    Tfu=np.sum((orderlabel<0)*orderweight)
    Szheng=np.cumsum((orderlabel>0)*orderweight)
    Sfu=np.cumsum((orderlabel<0)*orderweight)
    signalorder=((Szheng+(Tfu-Sfu)<(Sfu+(Tzheng-Szheng)))*1-0.5)*2
    tempindex=np.argmin(np.amin(np.array([Szheng+(Tfu-Sfu),Sfu+(Tzheng-S
    tempP=signalorder[tempindex]
    if (tempindex<(len(reference)-1)):
        Temptheta=(FFunctorder[tempindex]+FFunctorder[tempindex+1])/2
    else:
        Temptheta=FFunctorder[tempindex]
    esty=np.array(np.sign(tempP*FFunctKK-Temptheta))
    error=np.sum((esty!=reference)*Tweight)
    return Temptheta,tempP,error
```

Get one weak classifier:

```
def getWeakClassifier(weight,tempdataId):
    NL=tempdataId.shape[0]
    referencelabel=np.append(np.ones(2000),np.zeros(NL-2000))
    p=np.zeros(FeatN)
    theta=np.zeros(FeatN)
    epsilon=np.zeros(FeatN)
    for m in range(FeatN):
        #if m%1000==0:
        #    print m
        theta[m],p[m],epsilon[m]=getPtheta(FFunct[m,tempdataId],weight,
    J=np.argmin(epsilon)
    thetareturn=theta[J]
    preturn=p[J]
    currentMin=epsilon[J]
    featureIdxreturn=J

    return currentMin,preturn,thetareturn,featureIdxreturn
```

Codes for training the whole classifiers (All the strong classifiers and all the weak classifier within each strong one):

```
T=50
```

N=4

```
def newgetvolume(ff,ttimage):
    black = np.mean(ttimage[ff[0]:ff[4], ff[1]:ff[5]], axis=(0,1))
    white = np.mean(ttimage[ff[2]:ff[6], ff[3]:ff[7]], axis=(0,1))
    return -black + white


def ThetaFinder(idiid,inalphaFinal,inpFinal,infeatureIdxFinal,inthetaFi
    TempTheta=100
    Tt=len(inpFinal)
    Fre=np.zeros(Tt)
    for i in range(2000):
        for j in range(Tt):
            Fre[j]=newgetvolume(featuretbl[infeatureIdxFinal[j],:],data
        TempTheta=np.minimum(TempTheta,tempStrongClassify(Fre,inalphaFi
    return TempTheta

def tempStrongClassify(F_input,iinalphaFinal,iinpFinal,iinthetaFinal):
    h_tt=np.sign(F_input-iinthetaFinal)*iinpFinal
    temp1=np.sum(iinalphaFinal*h_tt)
    return temp1


def tempJudge(Fig_in,iiinalphaFinal,iiinpFinal,iiinthetaFinal,iiinfeatur
    Ttt=len(iiinpFinal)
    Fre=np.zeros(Ttt)
    for i in range(Ttt):
        Fre[i]=newgetvolume(featuretbl[iiinfeatureIdxFinal[i],:],Fig_in
    return np.sign(tempStrongClassify(Fre,iiinalphaFinal,iiinpFinal,iiin


ThetaFinal=np.zeros(N)
thetaFinal=[]#np.zeros((N,T))
alphaFinal=[]#np.zeros((N,T))
pFinal=[]#np.zeros((N,T))
featureIdxFinal=[]#np.zeros((N,T))
```

```
Xr=1600
Yr=1280
Indicator=np.ones((Xr,Yr))
Idremain=np.array(range(4000))
for n in range(N):
    FN=1
    ttttthetaFinal=[]
    tttalphaFinal=[]
    tttpFinal=[]
    tttfeatureIdxFinal=[]

    Long=Idremain.shape[0]
    print Long
    pos=2000
    neg=Long-2000
    label=np.append(np.ones(pos),np.zeros(Long-2000))
    label=label*2-1
    w=np.zeros((pos+neg))
    w[range(0,pos)]=1.0/pos
    w[range(pos,pos+neg)]=1.0/neg
    for t in range(T):
        print t
        w=w/np.sum(w)
        npFinal=0
        nthetaFinal=0
        nfeatureIdxFinal=0
        hcurrentMin,npFinal,nthetaFinal,nfeatureIdxFinal=getWeakClassif
        print hcurrentMin,npFinal,nthetaFinal,nfeatureIdxFinal
        ttttthetaFinal.append(nthetaFinal)
        tttpFinal.append(npFinal)
        tttfeatureIdxFinal.append(nfeatureIdxFinal)

        FFunctTempora=FFunct[tttfeatureIdxFinal[t],Idremain]
        httt=np.sign(tttpFinal[t]*(FFunctTempora-ttttthetaFinal[t]))
        nalphaFinal=0.5*np.log((1-hcurrentMin)/hcurrentMin)
        tttalphaFinal.append(nalphaFinal)
        z=2*np.sqrt((1-hcurrentMin)*hcurrentMin)
        w=np.exp(-nalphaFinal*label*httt)*w/z
```

```
    if t>3:
        ThetaFinal[n]=ThetaFinder(Idremain,tttalphaFinal,tttpFinal,
        round1=np.array(list(range(Long)))
        tempGroup=[]
        for i in round1:
            if i<=2000 or (i>2000 and tempJudge(datatrain[Idremain[
                tempGroup=np.append(tempGroup,i)
        #print tempGroup.shape[0]
        if (tempGroup.shape[0]-2000.1)/(Long-2000)<0.3:
            ThetaFinal[n]=ThetaFinder(Idremain,tttalphaFinal,tttpFi
            print ThetaFinal[n]
            print (tempGroup.shape[0]-2000.1)/(Long-2000)
            Idremain=Idremain[tempGroup.astype(int)]
            thetaFinal.append(tttthetaFinal)
            alphaFinal.append(tttalphaFinal)
            pFinal.append(tttpFinal)
            featureIdxFinal.append(tttfeatureIdxFinal)
            break
```

Function for doing the strong classification and codes for printing out the faces:

```
def StrongClassify(F_input,tttn):
    h_tt=np.sign(F_input-thetaFinal[tttn])*pFinal[tttn]
    temp1=np.sum(alphaFinal[tttn]*h_tt)
    return temp1
def printBound(Finalfig_in,n1,m1):
    Finalfig_in[n1:n1+3,m1:m1+64]=255
    Finalfig_in[n1+61:n1+64,m1:m1+64]=255
    Finalfig_in[n1:n1+64,m1:m1+3]=255
    Finalfig_in[n1:n1+64,m1+62:m1+65]=255
    Indicator[n1:n1+64,m1:m1+64]=0
    return Finalfig_in
```

Codes for storing all the selected features:

```
Newfeaturetbl=[]#np.zeros((N,T,8))
```

```
for q in range(N):
    TTtemp=len(pFinal[q])
    Newfeaturetbltemp=np.zeros((TTtemp,8))
    for p in range(TTtemp):
        Newfeaturetbltemp[p,:]=featuretbl[featureIdxFinal[q][p],:]
    Newfeaturetbl.append(Newfeaturetbltemp)
```

Codes for Judging whether a picture is a face or not and whether we should skip to the next position or not:

```
def Judge(Fig_in,xlabel,ylabel):
    if Doornot(xlabel,ylabel)==1:
        return 2
    for n in range(N):
        TTtemp=len(pFinal[n])
        Fre=np.zeros(TTtemp)
        for i in range(TTtemp):
            Fre[i]=newgetvolume(Newfeaturetbl[n][i,:],Fig_in)
        if np.sign(StrongClassify(Fre,n)-ThetaFinal[n])==-1:
            return -1
    return 1
def Doornot(n1,m1):
    Tempmatrix=Indicator[n1:n1+64,m1:m1+64]
    if np.sum(Tempmatrix)<0.8*64**2:
        return 1
    else:
        return 0
```

Codes for loading the final testing figure:

```
Finalpicture=np.zeros((1280,1600))
newname = "/Users/yangyibo/Desktop/MachineLearning/hw3/class.jpg"
pil_im =Image.open(newname).convert('L')
for m in range(1600):
    for n in range(1280):
        Finalpicture[n,m]= pil_im.getpixel((m,n))
arr=Finalpicture
plt.imshow(arr,cmap='gray')
```

```
plt.show()
```

Codes for doing the final test:

```
Xr=1280
Yr=1600
Indicator=np.ones((Xr,Yr))
Detect=np.zeros((64,64))
for m in range(0,Xr-64,5):
    for n in range(0,Yr-64,5):
        Detect=Finalpicture[m:m+64,n:n+64]
        if Judge(Detect,m,n)==1:                Finalpicture=printBound(Fin
```