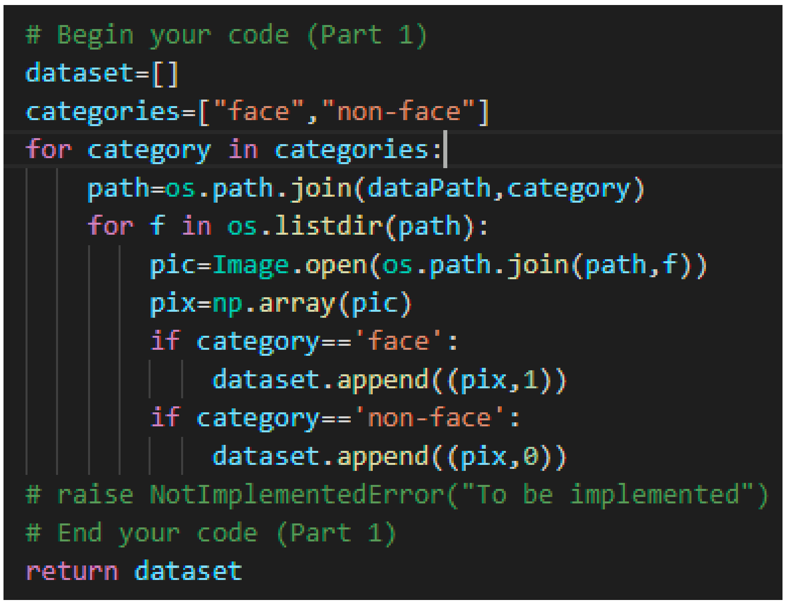
Homework 1: Face Detection

李耀凱 109611092

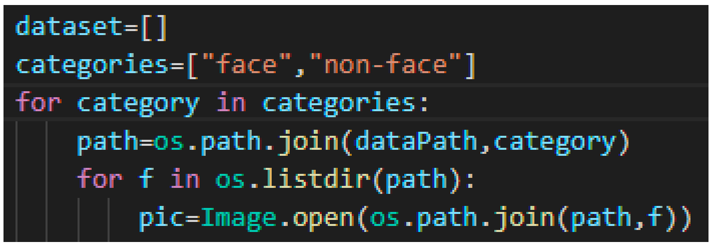
Part I . implementation

part 1. dataset



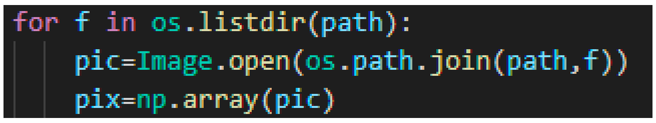
```python
# Begin your code (Part 1)
dataset=[]
categories=["face","non-face"]
for category in categories:
    path=os.path.join(dataPath,category)
    for f in os.listdir(path):
        pic=Image.open(os.path.join(path,f))
        pix=np.array(pic)
        if category=='face':
            dataset.append((pix,1))
        if category=='non-face':
            dataset.append((pix,0))
# raise NotImplementedError("To be implemented")
# End your code (Part 1)
return dataset
```

In this part, first, we have to get the path of the folder where the images are . Thus, by importing os.path.join functions, we can list and get the paths of folders and images we want.



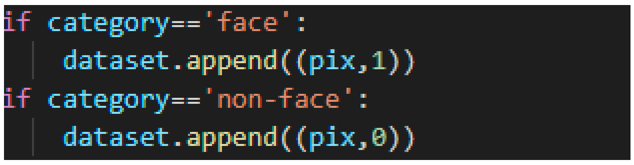
```python
dataset=[]
categories=["face","non-face"]
for category in categories:
    path=os.path.join(dataPath,category)
    for f in os.listdir(path):
        pic=Image.open(os.path.join(path,f))
```

Then, we load the images by importing Image.open() from Pillow, and turn them into numpy arrays.



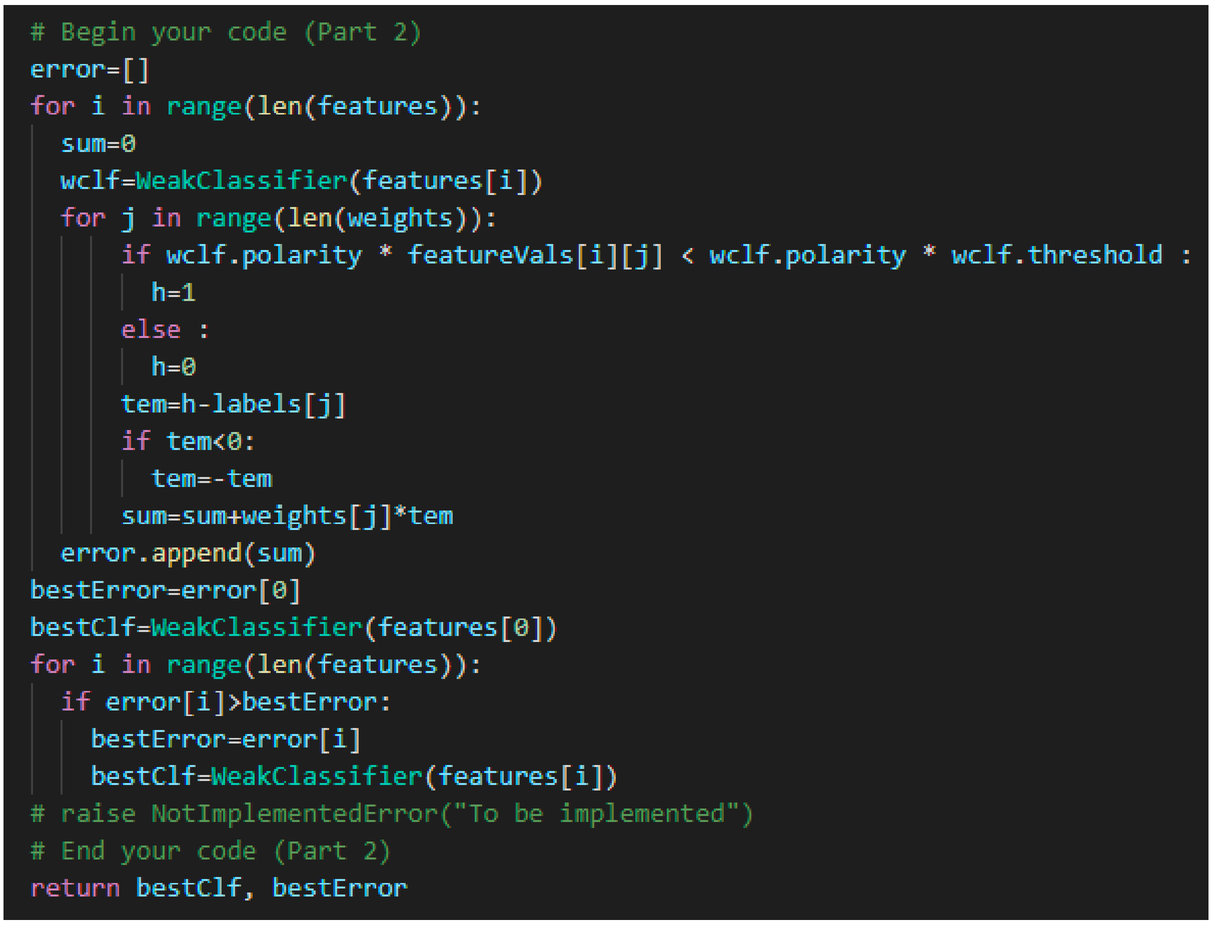
```python
for f in os.listdir(path):
    pic=Image.open(os.path.join(path,f))
    pix=np.array(pic)
```

Finally, we put the classification of each images into dataset[] by checking whether the image is face, if it is, labels 1 into numpy array, else labels 0.



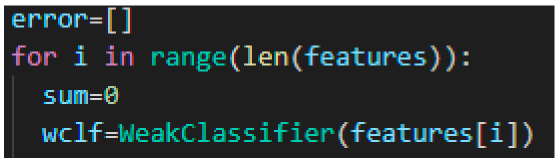
```
if category=='face':
    dataset.append((pix,1))
if category=='non-face':
    dataset.append((pix,0))
```

part 2. Adaboost



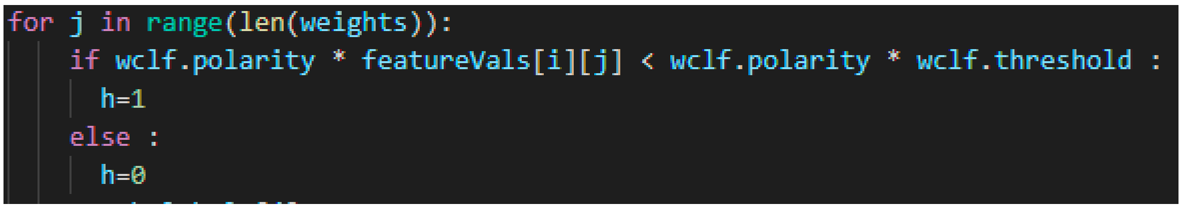
```
# Begin your code (Part 2)
error=[]
for i in range(len(features)):
  sum=0
  wclf=WeakClassifier(features[i])
  for j in range(len(weights)):
      if wclf.polarity * featureVals[i][j] < wclf.polarity * wclf.threshold :
        h=1
      else :
        h=0
      tem=h-labels[j]
      if tem<0:
        tem=-tem
      sum=sum+weights[j]*tem
  error.append(sum)
bestError=error[0]
bestClf=WeakClassifier(features[0])
for i in range(len(features)):
  if error[i]>bestError:
    bestError=error[i]
    bestClf=WeakClassifier(features[i])
# raise NotImplementedError("To be implemented")
# End your code (Part 2)
return bestClf, bestError
```

Because our goal is to find the minimum error among all classifiers, we build a list of errors first. Then in the first for-loop, we declare a value sum in order to calculate the error later, also, we apply the WeakClassifier function with input features, which is a numpy array, to calculate in the second for-loop.



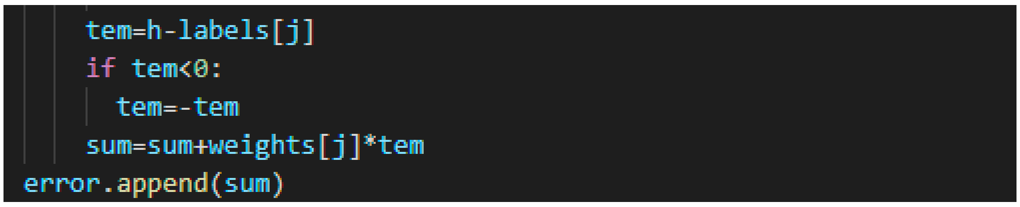
```
error=[]
for i in range(len(features)):
  sum=0
  wclf=WeakClassifier(features[i])
```

In the second for-loop, I choose to write the formula in the last line(line27) in classifier.py for easier reading and understanding, and after applying the formula, we can get the value of all classifiers with 1 or 0.



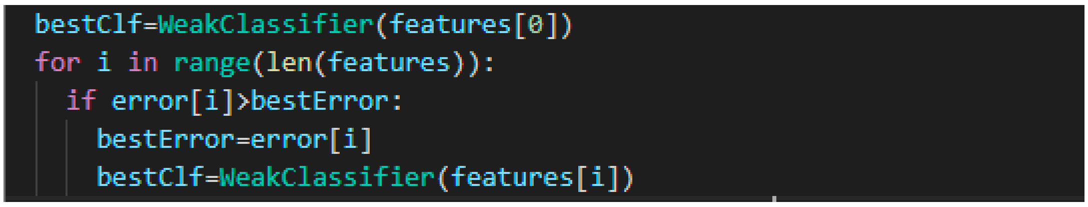
```
for j in range(len(weights)):
    if wclf.polarity * featureVals[i][j] < wclf.polarity * wclf.threshold :
        h=1
    else :
        h=0
```

then, use the formula to calculate the error of each classifiers, and put them into a list (error[]) in order to find the minimum later.

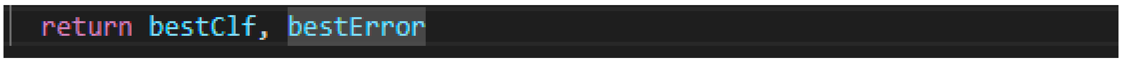


```
        tem=h-labels[j]
        if tem<0:
            tem=-tem
        sum=sum+weights[j]*tem
    error.append(sum)
```

Compare the errors and return the minimum error, also,return the classifier corresponding to the minimum error.

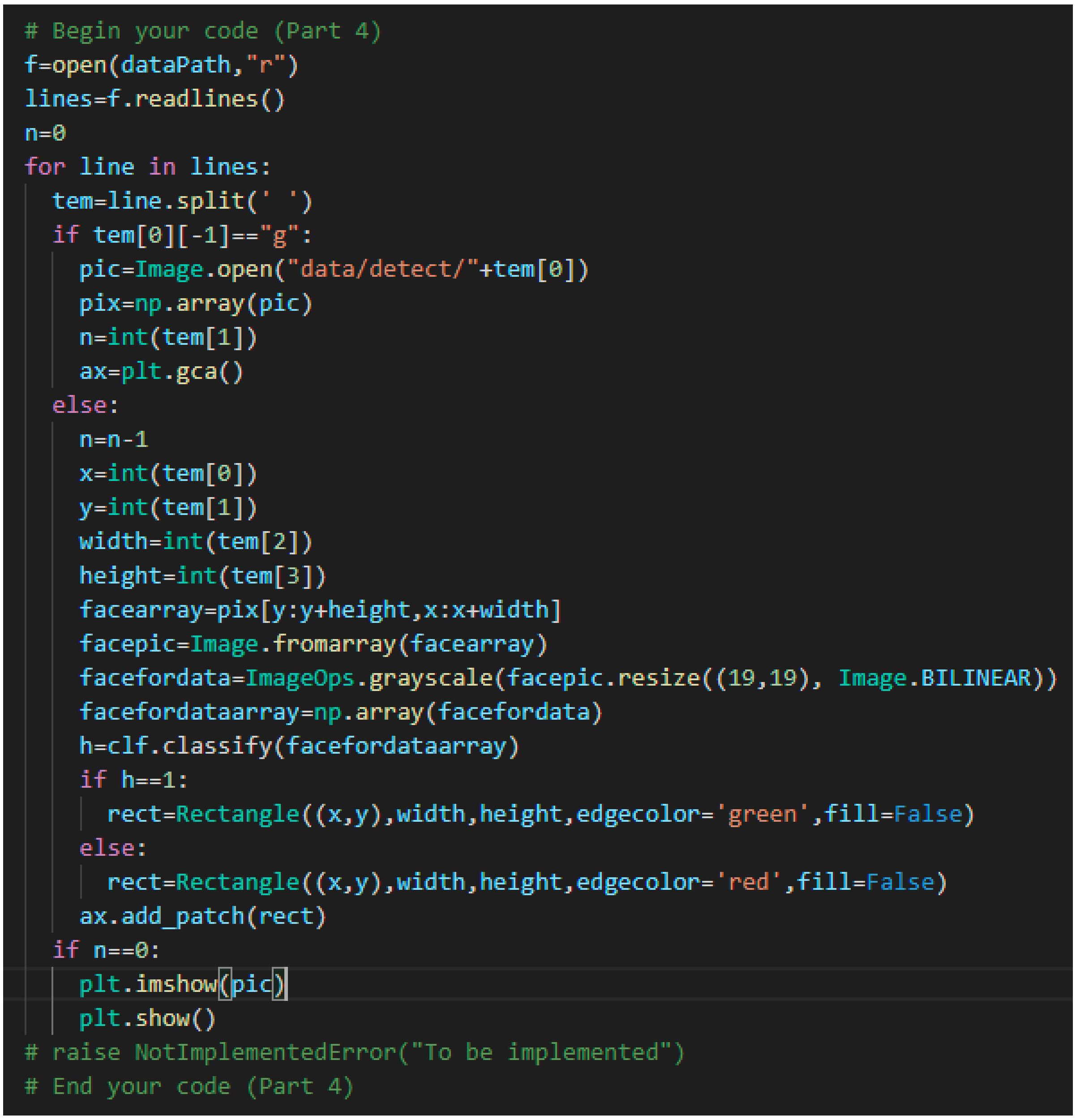


```
bestClf=WeakClassifier(features[0])
for i in range(len(features)):
    if error[i]>bestError:
        bestError=error[i]
        bestClf=WeakClassifier(features[i])
```



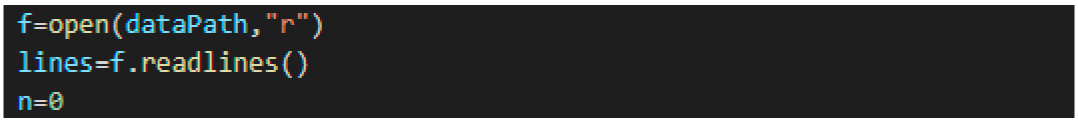
```
return bestClf, bestError
```

part4. detection



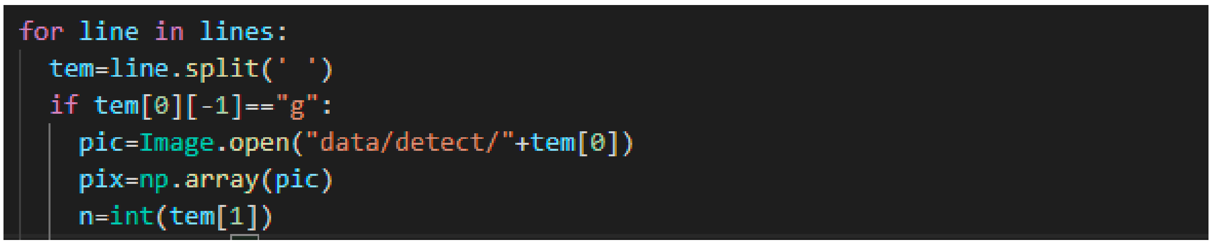
```python
# Begin your code (Part 4)
f=open(dataPath,"r")
lines=f.readlines()
n=0
for line in lines:
    tem=line.split(' ')
    if tem[0][-1]=="g":
        pic=Image.open("data/detect/"+tem[0])
        pix=np.array(pic)
        n=int(tem[1])
        ax=plt.gca()
    else:
        n=n-1
        x=int(tem[0])
        y=int(tem[1])
        width=int(tem[2])
        height=int(tem[3])
        facearray=pix[y:y+height,x:x+width]
        facepic=Image.fromarray(facearray)
        facefordata=ImageOps.grayscale(facepic.resize((19,19), Image.BILINEAR))
        facefordataarray=np.array(facefordata)
        h=clf.classify(facefordataarray)
        if h==1:
            rect=Rectangle((x,y),width,height,edgecolor='green',fill=False)
        else:
            rect=Rectangle((x,y),width,height,edgecolor='red',fill=False)
        ax.add_patch(rect)
    if n==0:
        plt.imshow(pic)
        plt.show()
# raise NotImplementedError("To be implemented")
# End your code (Part 4)
```

First, find the path of the detectData.txt and open it with mode:read. And I declare a variable, n, to stand for how many lines (faces) under a .jpg file (picture) later.



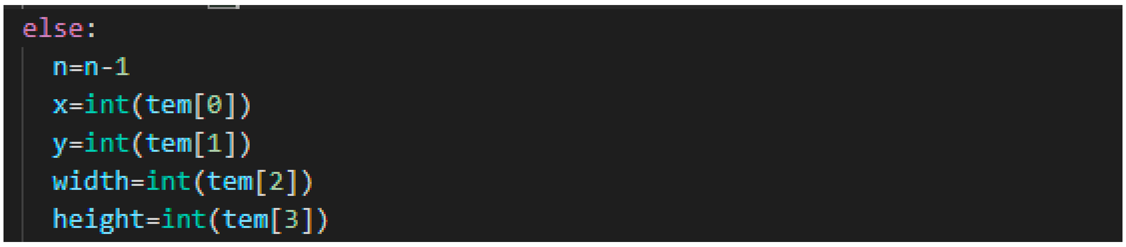
```python
f=open(dataPath,"r")
lines=f.readlines()
n=0
```

Secondly, read the detectData file line by line , in each line, we cut the string into several strings when there is a space, if the final letter of the first string is g, then we know it is a jpg file. What we gonna do is open it and turn the picture into a numpy array, and record how many lines that will appear afterward into n.



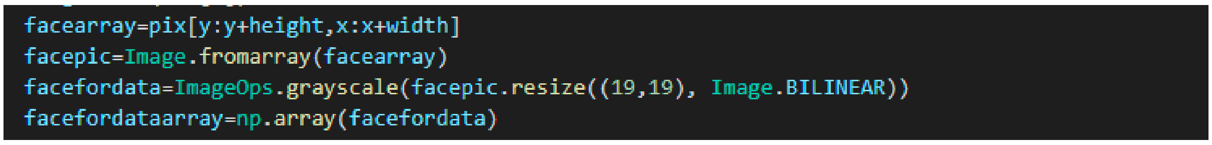
```
for line in lines:
    tem=line.split(' ')
    if tem[0][-1]=="g":
        pic=Image.open("data/detect/"+tem[0])
        pix=np.array(pic)
        n=int(tem[1])
```

If it isn't a jpg file, we know the string consists of 4 elements, which are the position, height and width of the subgraphs we want to classify.



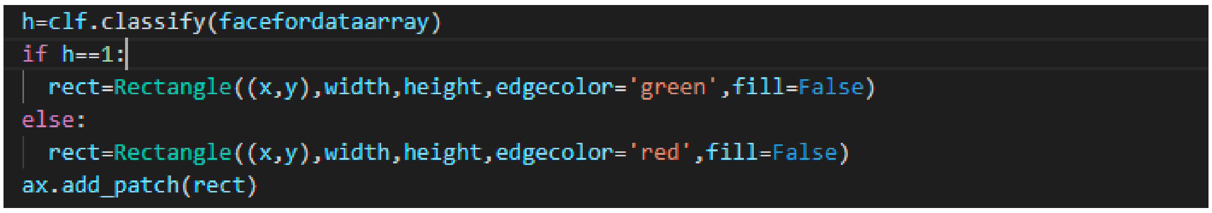
```
else:
    n=n-1
    x=int(tem[0])
    y=int(tem[1])
    width=int(tem[2])
    height=int(tem[3])
```

We have to resize and turn the subgraphs into grayscale, so we need to transfer them from the numpy array into images first. However, because the input form of clf.classify.function should be a numpy array, we have to transfer it again.



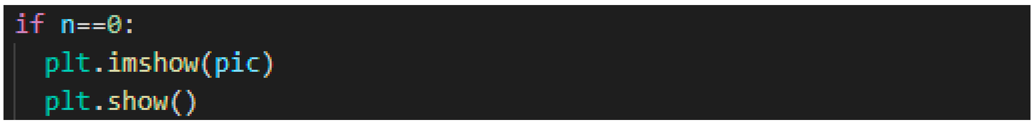
```
facearray=pix[y:y+height,x:x+width]
facepic=Image.fromarray(facearray)
facefordata=ImageOps.grayscale(facepic.resize((19,19), Image.BILINEAR))
facefordataarray=np.array(facefordata)
```

Finally, put all the data of subgraphs into classify, judge if it is a face. if it is, draw a green rectangle on the edge, else draw a red one. Here i imported matplotlib.pyplot and Rectangle from matplotlib.patches as tools to draw.



```
h=clf.classify(facefordataarray)
if h==1:
    rect=Rectangle((x,y),width,height,edgecolor='green',fill=False)
else:
    rect=Rectangle((x,y),width,height,edgecolor='red',fill=False)
ax.add_patch(rect)
```

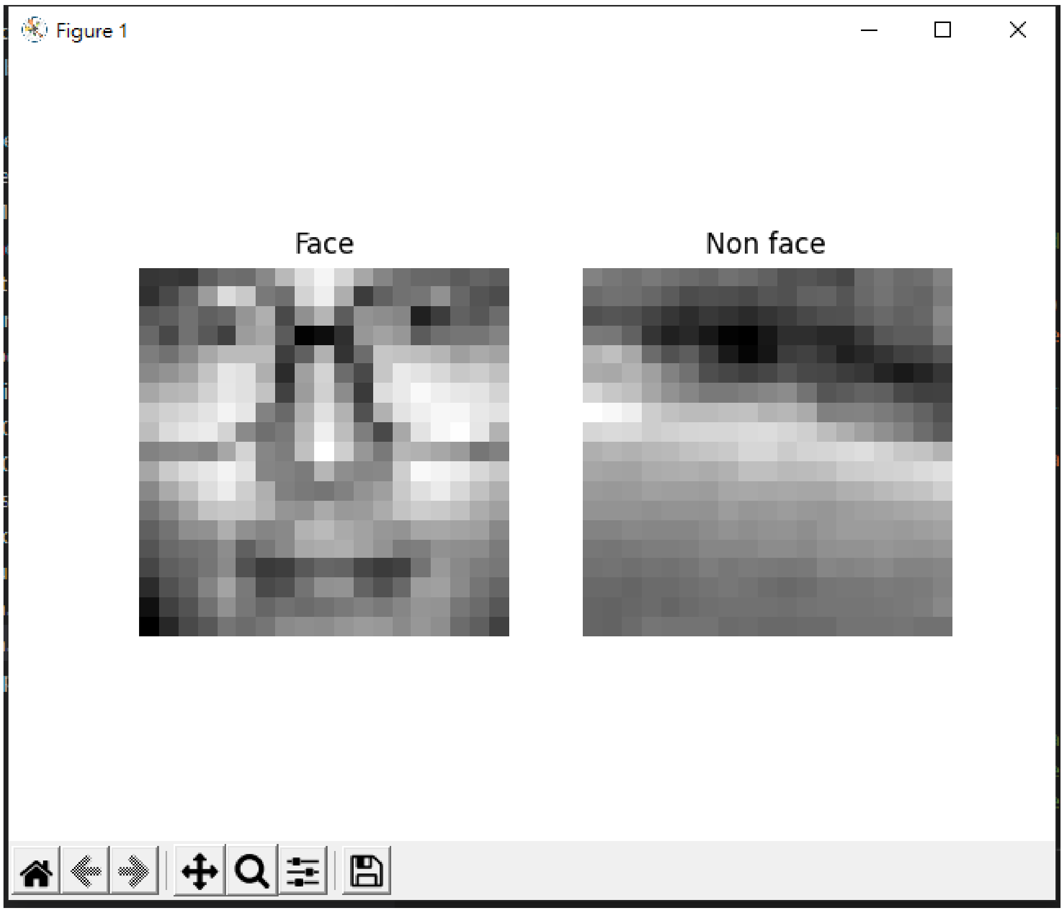
Show the result



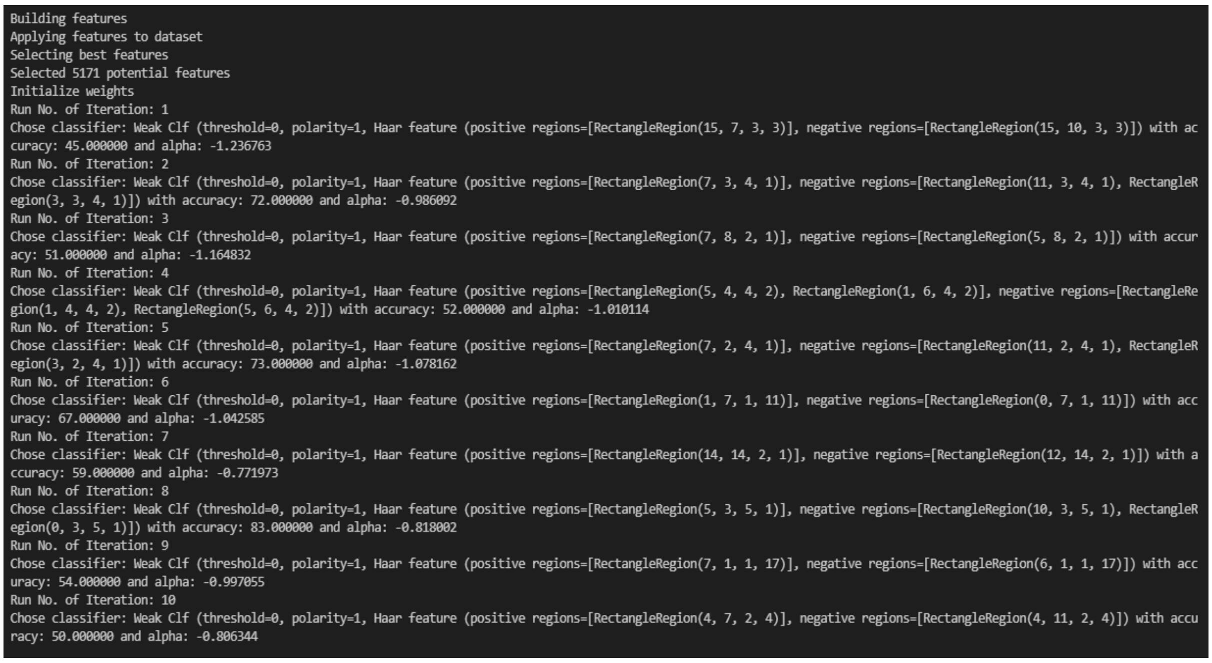
```
if n==0:
    plt.imshow(pic)
    plt.show()
```

Part II. Result and Analysis

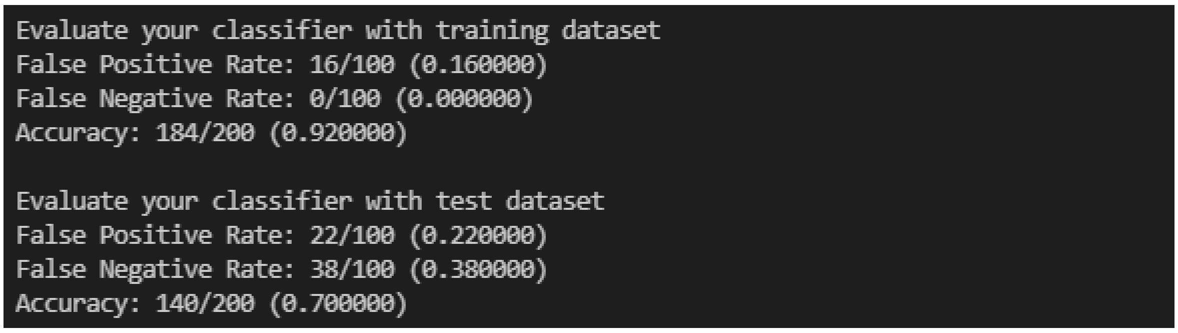
Part1.



part2.



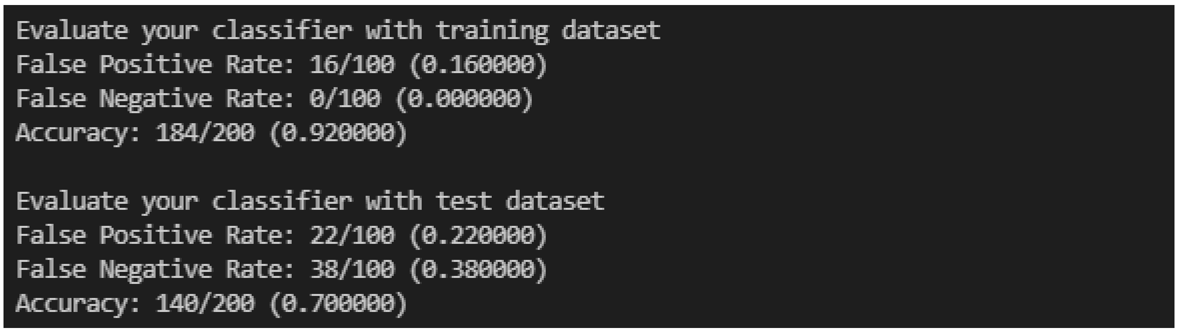
```
Building features
Applying features to dataset
Selecting best features
Selected 5171 potential features
Initialize weights
Run No. of Iteration: 1
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(15, 7, 3, 3)], negative regions=[RectangleRegion(15, 10, 3, 3)]) with ac
curacy: 45.000000 and alpha: -1.236763
Run No. of Iteration: 2
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(7, 3, 4, 1)], negative regions=[RectangleRegion(11, 3, 4, 1), RectangleR
egion(3, 3, 4, 1)]) with accuracy: 72.000000 and alpha: -0.986092
Run No. of Iteration: 3
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(7, 8, 2, 1)], negative regions=[RectangleRegion(5, 8, 2, 1)]) with accur
acy: 51.000000 and alpha: -1.164832
Run No. of Iteration: 4
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 4, 4, 2), RectangleRegion(1, 6, 4, 2)], negative regions=[RectangleRe
gion(1, 4, 4, 2), RectangleRegion(5, 6, 4, 2)]) with accuracy: 52.000000 and alpha: -1.010114
Run No. of Iteration: 5
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(7, 2, 4, 1)], negative regions=[RectangleRegion(11, 2, 4, 1), RectangleR
egion(3, 2, 4, 1)]) with accuracy: 73.000000 and alpha: -1.078162
Run No. of Iteration: 6
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(1, 7, 1, 11)], negative regions=[RectangleRegion(0, 7, 1, 11)]) with acc
uracy: 67.000000 and alpha: -1.042585
Run No. of Iteration: 7
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(14, 14, 2, 1)], negative regions=[RectangleRegion(12, 14, 2, 1)]) with a
ccuracy: 59.000000 and alpha: -0.771973
Run No. of Iteration: 8
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(5, 3, 5, 1)], negative regions=[RectangleRegion(10, 3, 5, 1), RectangleR
egion(0, 3, 5, 1)]) with accuracy: 83.000000 and alpha: -0.818002
Run No. of Iteration: 9
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(7, 1, 1, 17)], negative regions=[RectangleRegion(6, 1, 1, 17)]) with acc
uracy: 54.000000 and alpha: -0.997055
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(4, 7, 2, 4)], negative regions=[RectangleRegion(4, 11, 2, 4)]) with accu
racy: 50.000000 and alpha: -0.806344
```



```
Evaluate your classifier with training dataset
False Positive Rate: 16/100 (0.160000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 184/200 (0.920000)

Evaluate your classifier with test dataset
False Positive Rate: 22/100 (0.220000)
False Negative Rate: 38/100 (0.380000)
Accuracy: 140/200 (0.700000)
```

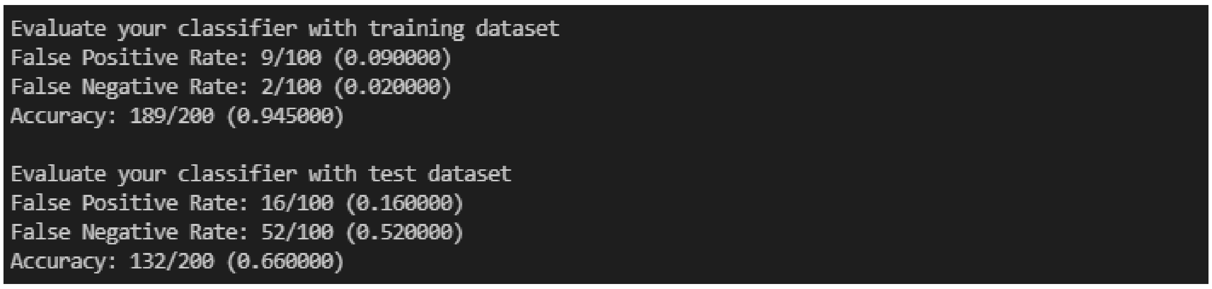
part3 change T from 1 to 10

T=10



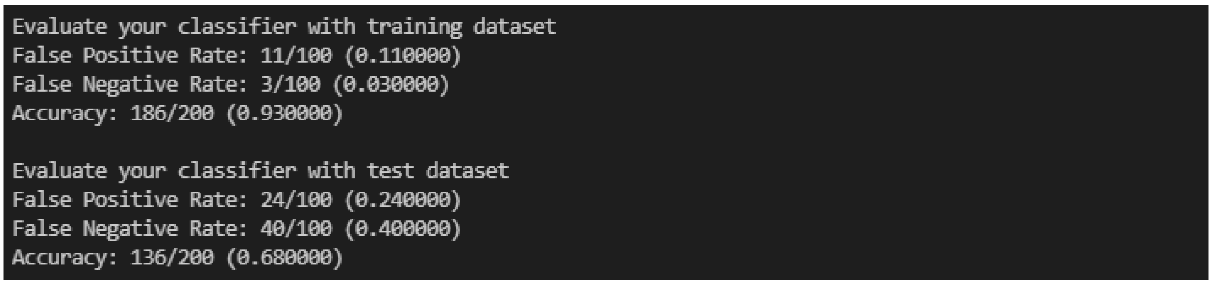
```
Evaluate your classifier with training dataset
False Positive Rate: 16/100 (0.160000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 184/200 (0.920000)

Evaluate your classifier with test dataset
False Positive Rate: 22/100 (0.220000)
False Negative Rate: 38/100 (0.380000)
Accuracy: 140/200 (0.700000)
```

T=9



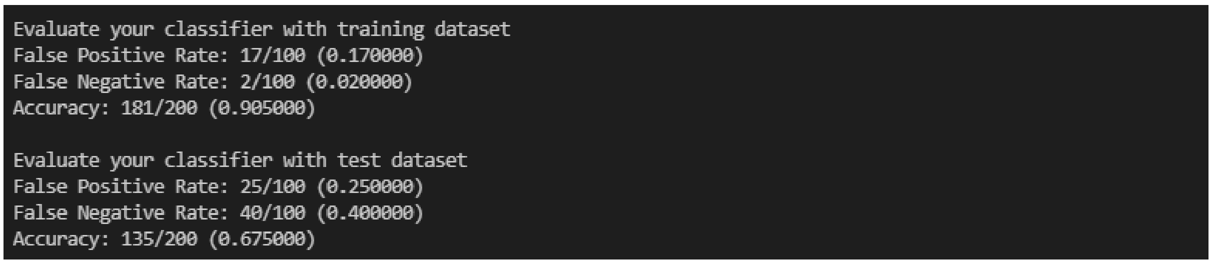
```
Evaluate your classifier with training dataset
False Positive Rate: 9/100 (0.090000)
False Negative Rate: 2/100 (0.020000)
Accuracy: 189/200 (0.945000)

Evaluate your classifier with test dataset
False Positive Rate: 16/100 (0.160000)
False Negative Rate: 52/100 (0.520000)
Accuracy: 132/200 (0.660000)
```

T=8

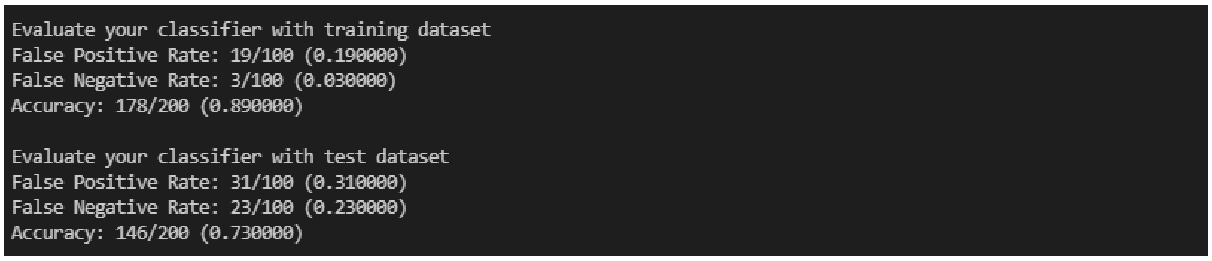


```
Evaluate your classifier with training dataset
False Positive Rate: 11/100 (0.110000)
False Negative Rate: 3/100 (0.030000)
Accuracy: 186/200 (0.930000)

Evaluate your classifier with test dataset
False Positive Rate: 24/100 (0.240000)
False Negative Rate: 40/100 (0.400000)
Accuracy: 136/200 (0.680000)
```

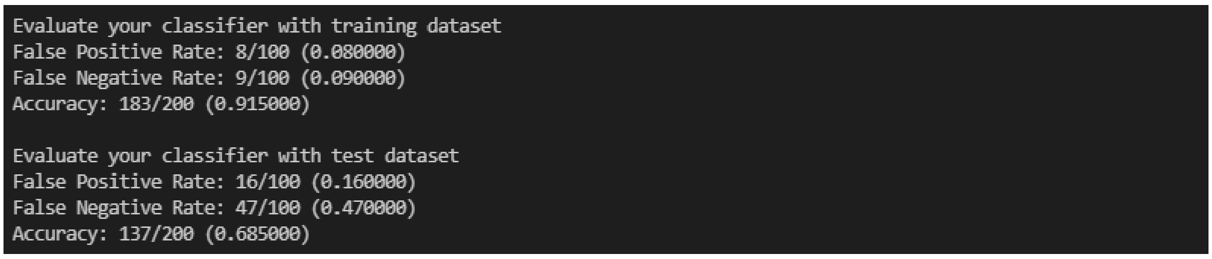
T=7


```
Evaluate your classifier with training dataset
False Positive Rate: 17/100 (0.170000)
False Negative Rate: 2/100 (0.020000)
Accuracy: 181/200 (0.905000)

Evaluate your classifier with test dataset
False Positive Rate: 25/100 (0.250000)
False Negative Rate: 40/100 (0.400000)
Accuracy: 135/200 (0.675000)
```

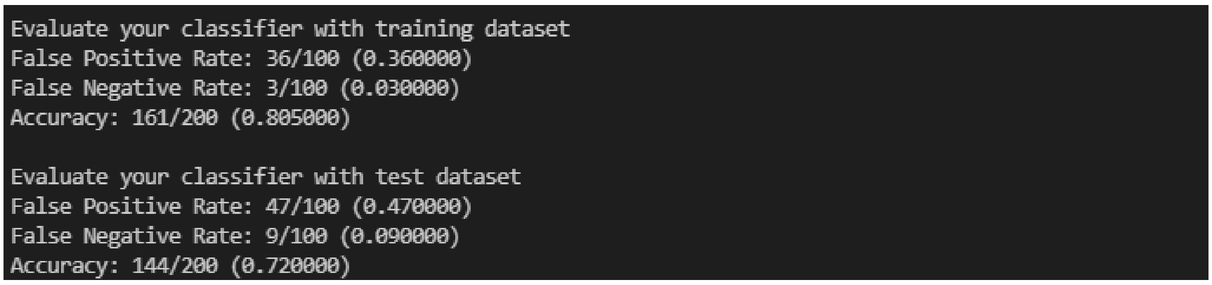
T=6


```
Evaluate your classifier with training dataset
False Positive Rate: 19/100 (0.190000)
False Negative Rate: 3/100 (0.030000)
Accuracy: 178/200 (0.890000)

Evaluate your classifier with test dataset
False Positive Rate: 31/100 (0.310000)
False Negative Rate: 23/100 (0.230000)
Accuracy: 146/200 (0.730000)
```

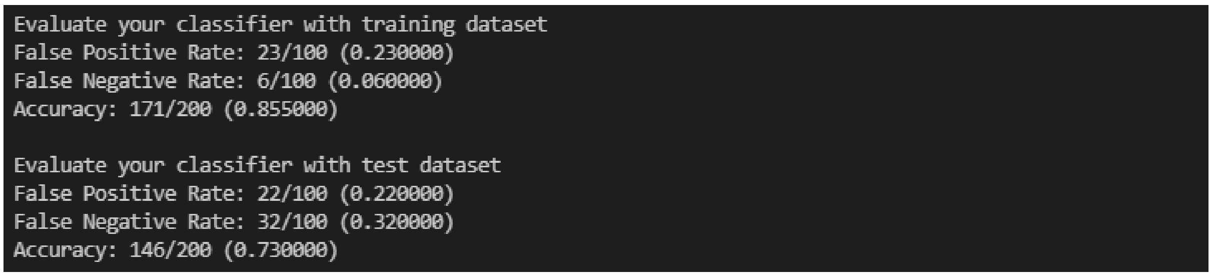
T=5


```
Evaluate your classifier with training dataset
False Positive Rate: 8/100 (0.080000)
False Negative Rate: 9/100 (0.090000)
Accuracy: 183/200 (0.915000)

Evaluate your classifier with test dataset
False Positive Rate: 16/100 (0.160000)
False Negative Rate: 47/100 (0.470000)
Accuracy: 137/200 (0.685000)
```

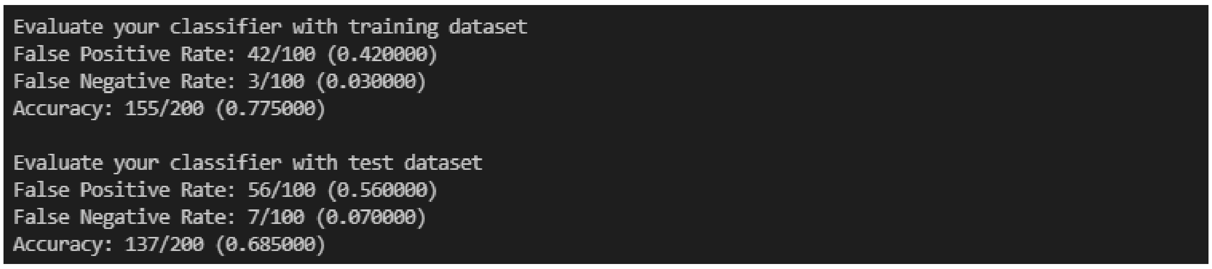
T=4


```
Evaluate your classifier with training dataset
False Positive Rate: 36/100 (0.360000)
False Negative Rate: 3/100 (0.030000)
Accuracy: 161/200 (0.805000)

Evaluate your classifier with test dataset
False Positive Rate: 47/100 (0.470000)
False Negative Rate: 9/100 (0.090000)
Accuracy: 144/200 (0.720000)
```

T=3

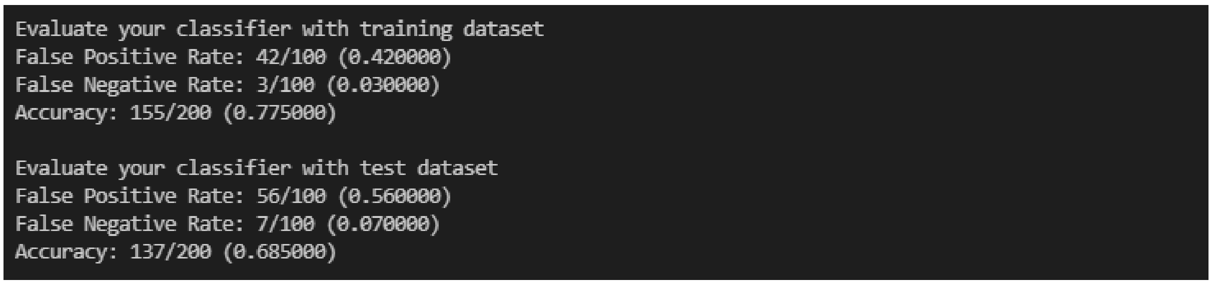

```
Evaluate your classifier with training dataset
False Positive Rate: 23/100 (0.230000)
False Negative Rate: 6/100 (0.060000)
Accuracy: 171/200 (0.855000)

Evaluate your classifier with test dataset
False Positive Rate: 22/100 (0.220000)
False Negative Rate: 32/100 (0.320000)
Accuracy: 146/200 (0.730000)
```

T=2



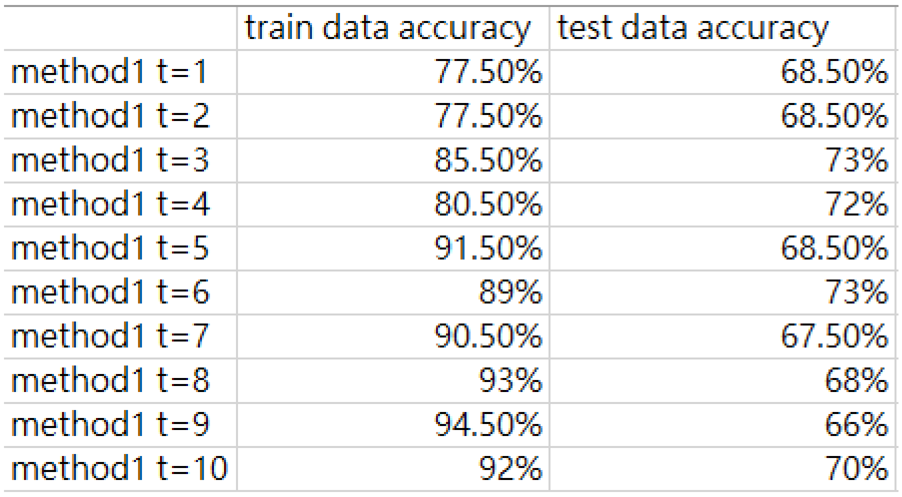
```
Evaluate your classifier with training dataset
False Positive Rate: 42/100 (0.420000)
False Negative Rate: 3/100 (0.030000)
Accuracy: 155/200 (0.775000)

Evaluate your classifier with test dataset
False Positive Rate: 56/100 (0.560000)
False Negative Rate: 7/100 (0.070000)
Accuracy: 137/200 (0.685000)
```

T=1



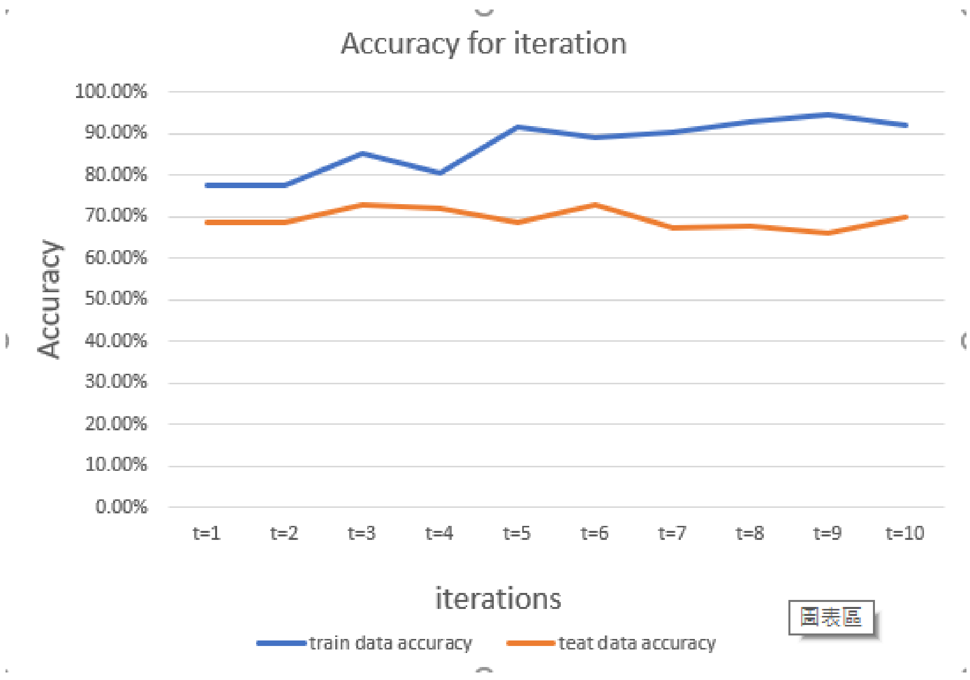
```
Evaluate your classifier with training dataset
False Positive Rate: 42/100 (0.420000)
False Negative Rate: 3/100 (0.030000)
Accuracy: 155/200 (0.775000)

Evaluate your classifier with test dataset
False Positive Rate: 56/100 (0.560000)
False Negative Rate: 7/100 (0.070000)
Accuracy: 137/200 (0.685000)
```

Show the accuracy in table and chart



	train data accuracy	test data accuracy
method1 t=1	77.50%	68.50%
method1 t=2	77.50%	68.50%
method1 t=3	85.50%	73%
method1 t=4	80.50%	72%
method1 t=5	91.50%	68.50%
method1 t=6	89%	73%
method1 t=7	90.50%	67.50%
method1 t=8	93%	68%
method1 t=9	94.50%	66%
method1 t=10	92%	70%

When we have higher parameters, the accuracy of train data is higher, but the test data accuracy doesn't have manifest change, I think the reason is the dataset is too small.


Accuracy for iteration
Accuracy
100.00%
90.00%
80.00%
70.00%
60.00%
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%
t=1
t=2
t=3
t=4
t=5
t=6
t=7
t=8
t=9
t=10
iterations
train data accuracy
teat data accuracy
圖表區

part4

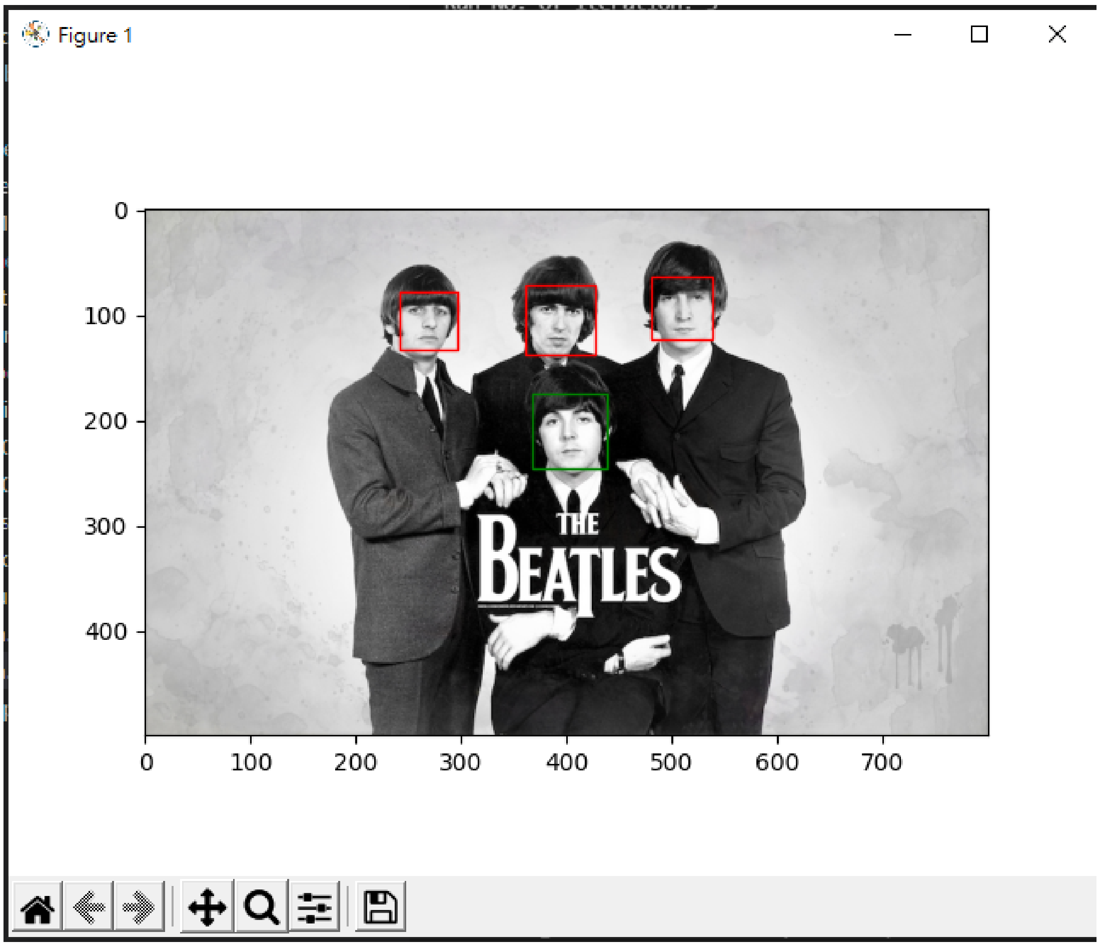


Figure 1
0
100
200
300
400
0
100
200
300
400
500
600
700
THE BEATLES

t

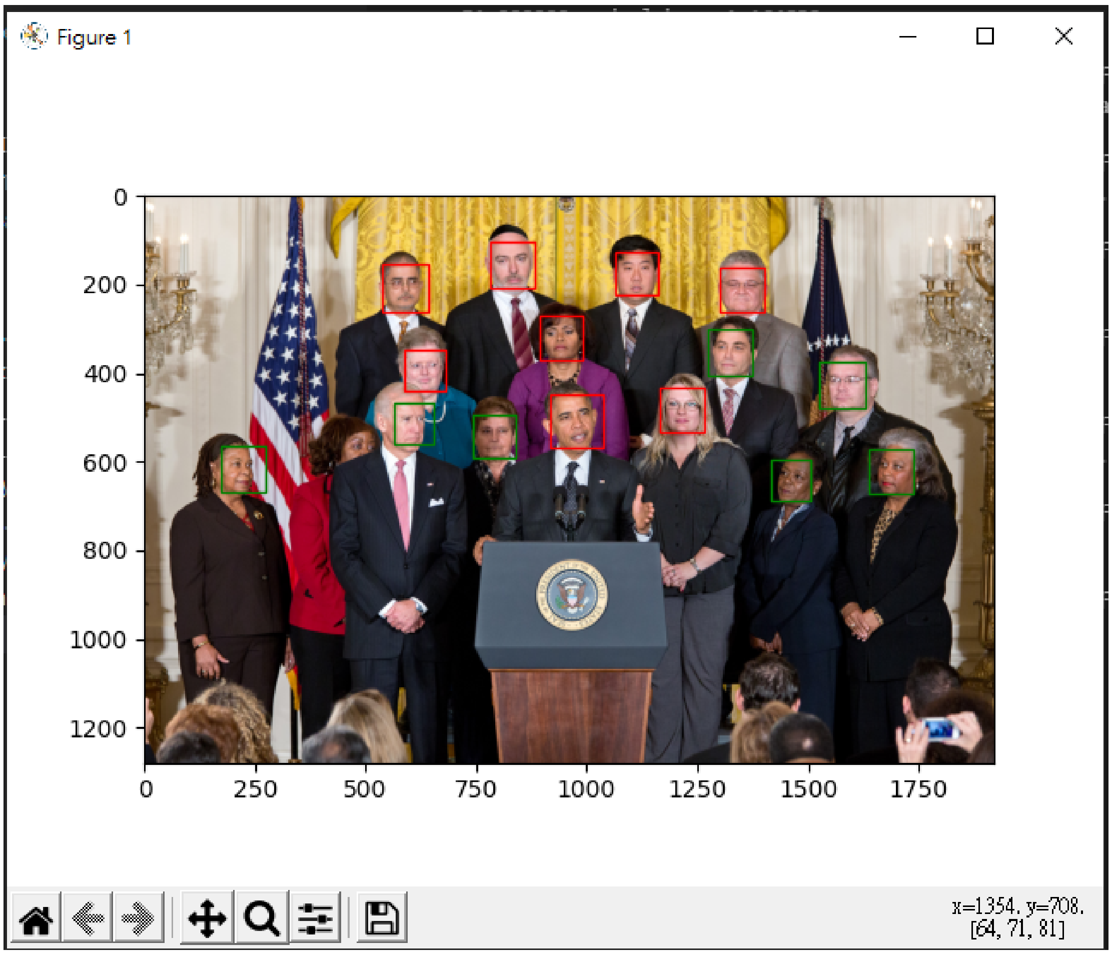


Figure 1
0
200
400
600
800
1000
1200
0
250
500
750
1000
1250
1500
1750
x=1354, y=708.
[64, 71, 81]

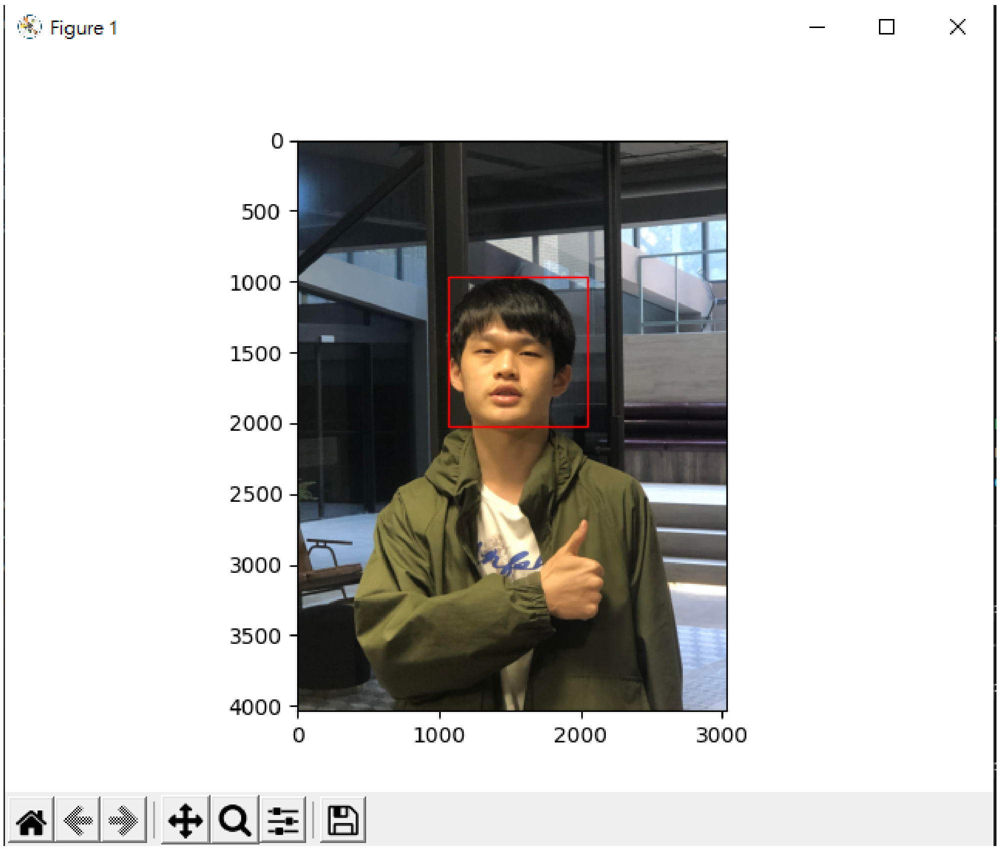
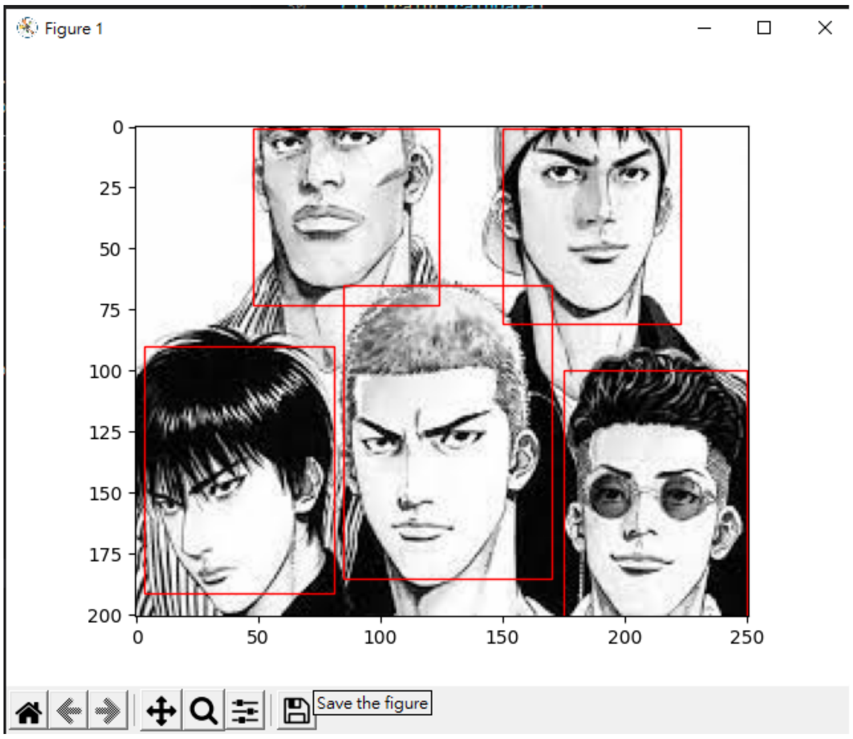
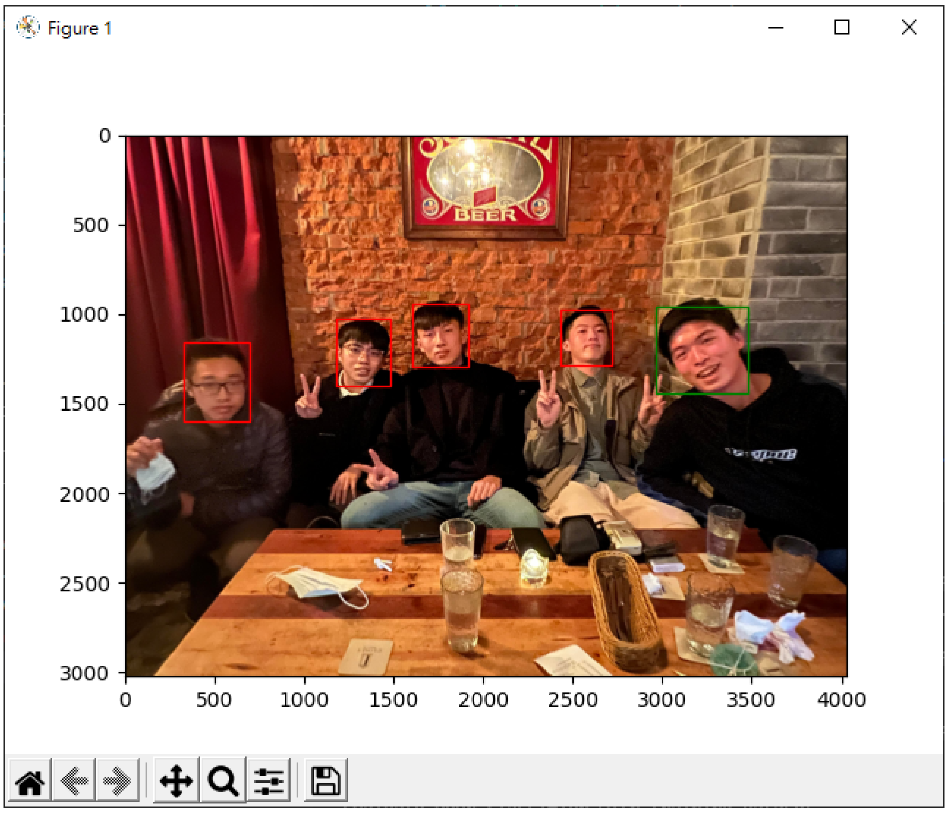


Figure 1
0
500
1000
1500
2000
2500
3000
3500
4000
0
1000
2000
3000

Because I want to know whether it can work on faces drew by hand, which have a clear counter, I put the comic picture as a experiment, however,it seems that all of the faces can't be detected, and I guess the reason is that 2D pictures have difference in considering the shadow on face like 3D pictures.





Part III. Questions

1. Please describe a problem you encountered and how you solved it.

   Because I'm not familiar with python, I spend a lot of time learning the grammar. Also, understanding the relationship among features, classifiers, weight and labels took me a lot of time, I spent lots of time on searching the information.

2. What are the limitations of the Viola-Jones algorithm?

   A: If people's faces are not in front of the lens or covered by a mask…etc, the accuracy of Viola-Jones algorithm might decrease a lot.

3. Based on Viola-Jones' algorithm, how to improve the accuracy except increasing the training dataset and changing the parameter T?

   A: I think we can try to create a function to harmonize the gray scale or have more detailed parameters with haar-like features to avoid the situation of faces under different light sources.

4. Please propose another possible face detection method (no matter how good or bad, please come up with an idea). Please discuss the pros and cons of the idea you proposed, compared to the Adaboost algorithm.

   A:Neural network based. Although the performance of Adaboost is better, the Neural network based method has very high precision with lower recall.