Introduction to Artificial Intelligence

0711239 李勝維

Part 1: Load and prepare your dataset

```
def loadImages(dataPath):
    # Begin your code (Part 1)
    dataset = []
    face_img = [os.path.join(dataPath, "face", f)
                for f in os.listdir(os.path.join(dataPath, "face"))]
    non_face_img = [os.path.join(dataPath, "non-face", f)
                for f in os.listdir(os.path.join(dataPath, "non-face"))]
```

First of all, the program creates two lists, each list stores the names of images in a directory, one list for face images and one list for non-face images.

```
    flag = 1
    for i in face_img:
        img = cv2.imread(i, -1)
        tmp = (img, flag)
        dataset.append(tmp)

    flag = 0
    for i in non_face_img:
        img = cv2.imread(i, -1)
        tmp = (img, flag)
        dataset.append(tmp)

    # End your code (Part 1)
    return dataset
```

Next, the program iterates through the list and load each image as a numpy array, then the program packs the numpy array with its corresponding label into a 2-tuple (The variable “flag” represents the label for the image).

Finally, the program inserts the tuples into “dataset” and return it.

Part 2: Implement Adaboost algorithm

(this part should be commented in the actual code since I've implemented part 6)

```
def selectBest(self, featureVals, iis, labels, features, weights):
    # Begin your code (Part 2)

    bestError = 2**64
```

First, the program initializes the variable "bestError", which stores current lowest (best) error.

```
    for haar in features:
        Clf = WeakClassifier(haar)
        error = 0
        for wt, label, ii in zip(weights, labels, iis):
            error += wt * abs(label - Clf.classify(ii))
```

Second, the program iterates through each Haar-like feature in "features". For each Haar-like feature, the program generates a weak classifier for it and calculates the error of the weak classifier using formula:

$$\sum_i w_i \left| h_j(x_i) - y_i \right|$$

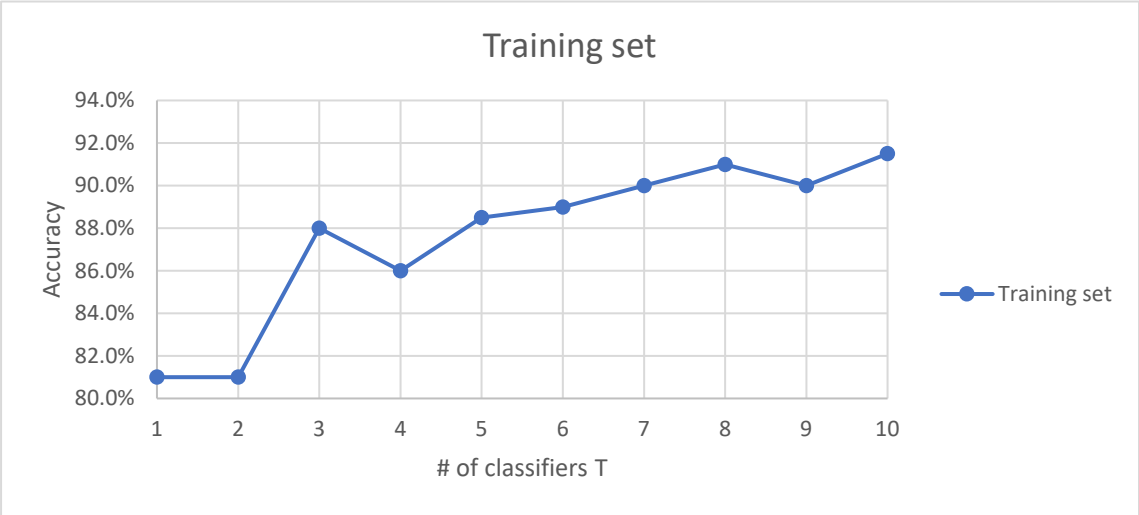
```
        if error <= bestError:
            bestError = error
            bestClf = Clf

    # End your code (Part 2)
    return bestClf, bestError
```

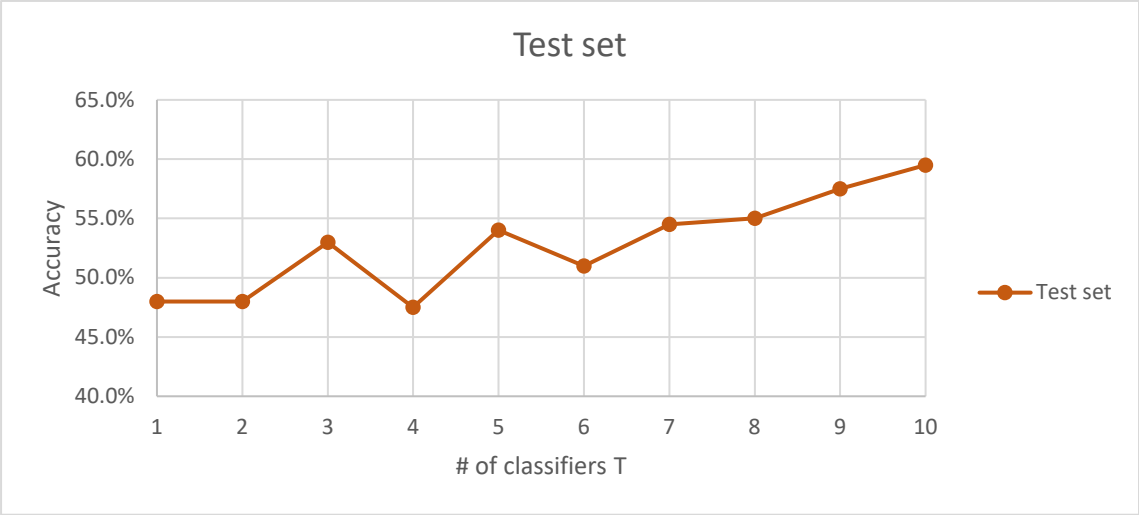
After that, the program updates "bestError" variable if current weak classifier has lower error than "bestError" , then the program stores the weak classifier as well.

Finally, the program returns the best weak classifier, "bestClf", and its error, "bestError".

Part 3: Additional experiments

Accuracy on training set and test set for different parameter T:





As we can see, for both training set and test set, the accuracy tends to increase as T increases. Additionally, overfitting doesn't seem to occur, my guess is that since the design of our weak classifier is very simple, averaging them doesn’t lead to overfitting.

Part 4: Detect face

```
def detect(dataPath, clf):
    # Begin your code (Part 4)

    file = open(dataPath, "r")
    line = file.readline()
```

First of all, the program opens “detectData.txt” and iterates through every line in it

```
    while line:
        line = line.split()
        filename = line[0]
        face_num = int(line[1])
        face_set = []
        img = cv2.imread(os.path.join(dataPath, "..", filename))
```

In this section, the program opens the image and reads how many faces are in the image.

```
        for i in range(face_num):
            line = file.readline()
            line = line.split()
            line = [int(i) for i in line]
            x, y, w, h = line
            # crop, convert and resize image
            cropped_img = img[y:y+h, x:x+w]
            gray_img = cv2.cvtColor(cropped_img, cv2.COLOR_BGR2GRAY)
            resized_img = cv2.resize(
                gray_img, (19, 19), interpolation=cv2.INTER_NEAREST)

            # classify image
            if clf.classify(resized_img):
                color = (0, 255, 0)
            else:
                color = (0, 0, 255)

            cv2.rectangle(img, (x, y),
                          (x+w, y+h), color, 3)
```

Thus, the program reads every line which contains the location of each face, then the program converts each face into a 19 x 19 grayscale image.

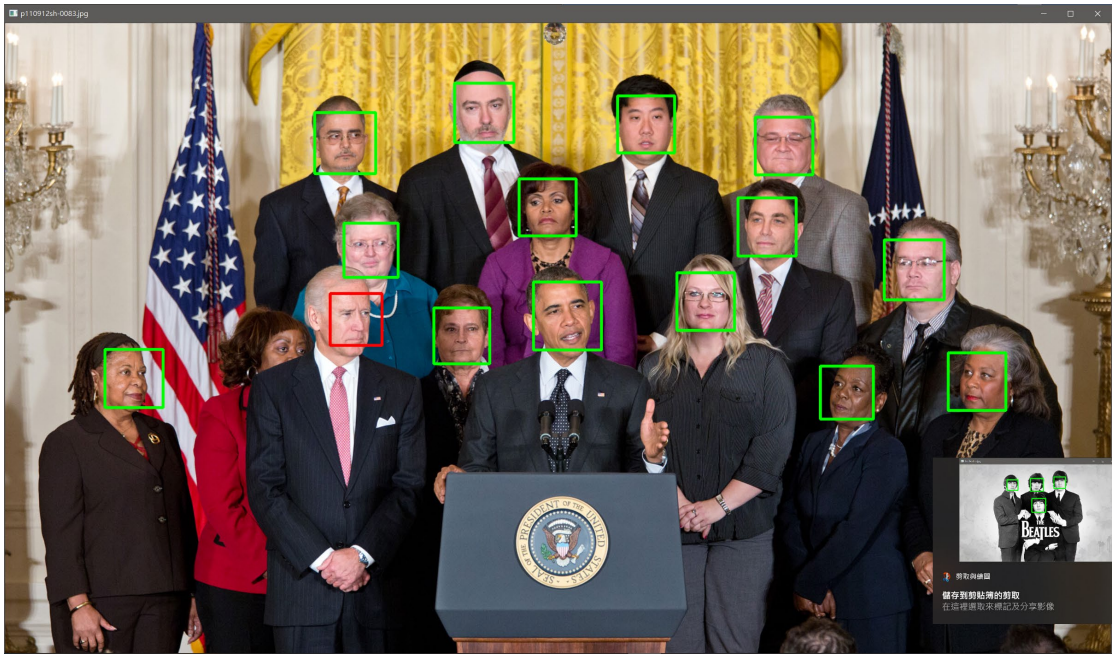
```
            # classify image
            if clf.classify(resized_img):
                color = (0, 255, 0)
            else:
                color = (0, 0, 255)

            cv2.rectangle(img, (x, y),
                          (x+w, y+h), color, 3)
```

Next, the program classifies each face using adaboost classifier, then draws a box according to the result of the classifier.

```
        cv2.imshow(filename, img)
        cv2.waitKey(0)
        cv2.destroyWindow(filename)
        line = file.readline()

    file.close()

    # End your code (Part 4)
```

After the program classifies each face, it shows the result image. The program ends after it reads all images in “detectData.txt”.

Here are the results for given images and my own image:
(Parameter T = 1, Adaboost algorithm = part 2)



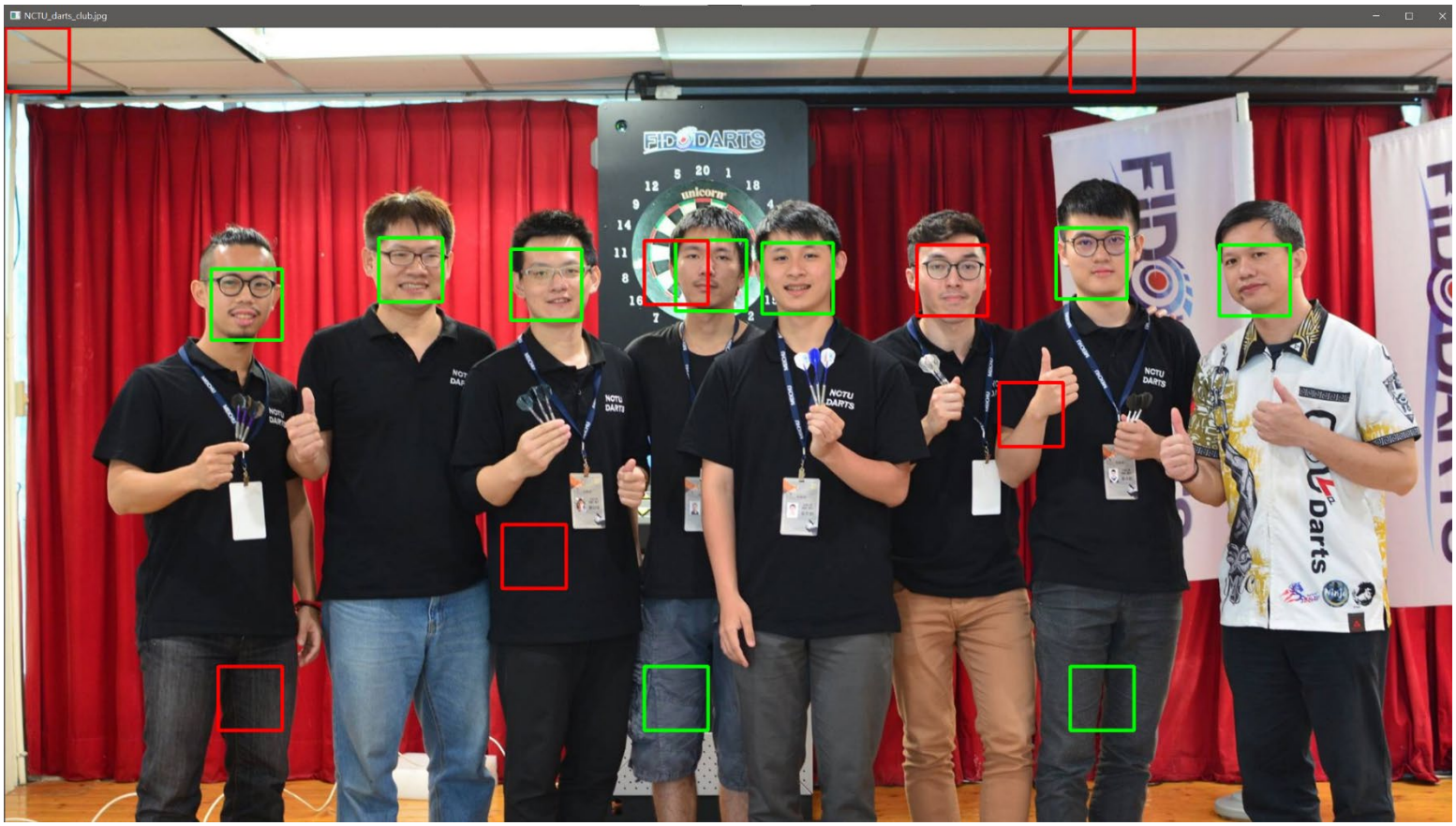
Although the results look amazing, according to the high false positive rate (49%) on test set, I believe that the classifier simply just outputs 1 (True) in most cases.

Part 5:

The source of the image is NCTU darts club @ MeiChu games, and I use MS Paint to locate each box.

I've selected 8 faces and 8 non-faces to test the classifier, here are the results:

(Parameter T = 1, Adaboost algorithm = part 2)



False positive: 2/8

False negative: 1/8

Accuracy: 13/16 (81.25%)

The results are quite impressive!

Part 6: Implement another classifier

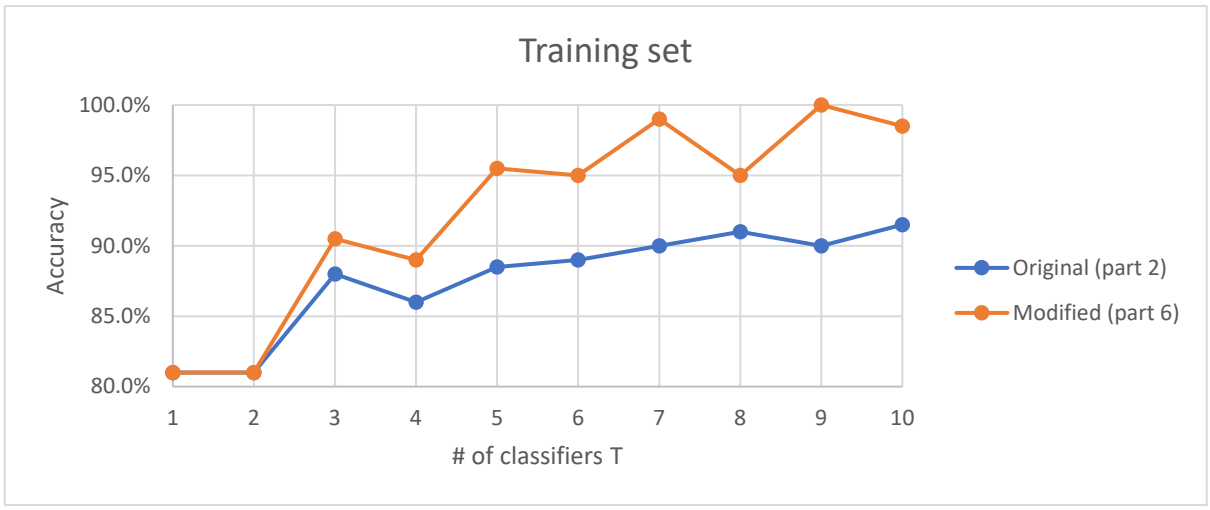
I will first explain the reasons for modification to the adaboost algorithm, then the comparison with part 2

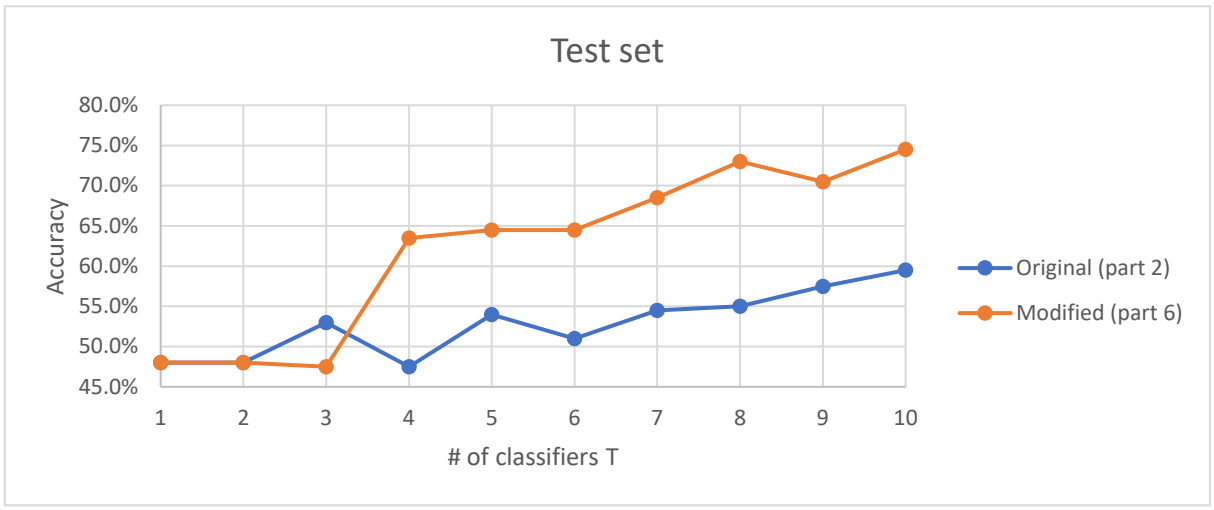
```
def selectBest(self, featureVals, iis, labels, features, weights):
    # Begin your code (Part 6)

    lowestError = 2**64
    highestError = 0
    for haar in features:
        Clf = WeakClassifier(haar)
        error = 0
        for wt, label, ii in zip(weights, labels, iis):
            error += wt * abs(label - Clf.classify(ii))
        if error <= lowestError:  # update lowest error
            lowestError = error
            lowestClf = Clf
        if error >= highestError:  # update highest error
            highestError = error
            highestClf = Clf

    if lowestError < (1 - highestError):
        bestClf = lowestClf
        bestError = lowestError
    else:
        bestClf = highestClf
        bestClf.polarity = (-1) * bestClf.polarity  # inverting output
        bestError = (1 - highestError)

    # End your code (Part 6)
    return bestClf, bestError
```

In my version of Adaboost, instead of selecting the classifier with the lowest error, I also consider classifiers with high errors, since our classifier is doing a binary classification task, the complement of error rate is also a possible error rate (simply by inverting the output). So, in each iteration of Haar-feature, the program will update not only the lowest error, but also the highest error.

In the end, the program returns the best classifier by comparing the lowest error and the complement of highest error

Here are the results:





Both chart shows that the modified version performs better than the original version of adaboost algorithm

Problems I've met and how I solve them

1. Converting faces into 19*19 grayscale image

➔ I thought that y axis was from bottom to top at first, it took me about 20 minutes to find out the problem.

2. Labeling faces (part 5) was a difficult job for me.

➔ I found out that MS paint can location each pixel directly. Thanks a lot to TAs for labeling the data!