Homework 1: Face Detection Report

0711257葉長瀚

Part I. Implementation:

```
# Begin your code (Part 1)
'''
Open the directory of the testing data, and read the grayscale version
of face and non-face images in with a for loop.
'''
dataset=[]
path= dataPath+"/face"
fptr= os.listdir(path)
for i in fptr:
  img= cv2.imread(path+"/"+i, cv2.IMREAD_GRAYSCALE)
  dataset.append((img,1))
path=dataPath+"/non-face"
fptr= os.listdir(path)
for i in fptr:
  img=cv2.imread(path+"/"+i, cv2.IMREAD_GRAYSCALE)
  dataset.append((img,0))
# raise NotImplementedError("To be implemented")
# End your code (Part 1)
return dataset
```

```
# Begin your code (Part 2)
'''
Create an numpy array h to store the value after classified, and calculate
the error of each feature. Find out which feature has the minimum error,
return it as the best weak classifier and the best error.
'''
h=np.zeros(featureVals.shape)
error=[]
for i in range(len(features)):
  for j in range(len(labels)):
    h[i][j]=int(featureVals[i][j]<0)
for i in range(len(features)):
  error.append(sum(weights[j]*abs(h[i][j]-labels[j]) for j in range(len(labels)) ))
bestError=error[0]
index=0
for i in range(len(features)):
  if bestError>error[i]:
    bestError=error[i]
    index=i
bestClf=WeakClassifier(features[index])
# raise NotImplementedError("To be implemented")
# End your code (Part 2)
return bestClf, bestError
```

```
# Begin your code (Part 4)
'''
Read the information of detection data in, modify the image to the form that can be
classified, and draw a rectangle on the original image depending on the classified
result.
'''
path="./"+dataPath
fptr=open(path, 'r')
while 1:
  temp=fptr.readline()
  if not temp:
    break
  temp=temp.split()
  img=cv2.imread("./data/detect/"+temp[0])
  imgGray=cv2.imread("./data/detect/"+temp[0], cv2.IMREAD_GRAYSCALE)
  for i in range(int(temp[1])):
    (x,y,w,h)=[int(x) for x in fptr.readline().split()]
    face=imgGray[y:y+h, x:x+w]
    face=cv2.resize(face, (19,19))
    res=clf.classify(face)
    if res:
      cv2.rectangle(img, (x,y), (x+w, y+h), (0, 255, 0), 1)
    else:
      cv2.rectangle(img, (x,y), (x+w, y+h), (0, 0, 255), 1)
  cv2.imshow("Result", img)
  cv2.waitKey(0)
fptr.close()
# raise NotImplementedError("To be implemented")
# End your code (Part 4)
```

Part II. Results & Analysis:

Training and testing accuracy of method 1 when T=10:

```
Run No. of Iteration: 10
Chose classifier: Weak Clf (threshold=0, polarity=1, Haar feature (positive regions=[RectangleRegion(4,
tangleRegion(2, 9, 2, 2), RectangleRegion(4, 11, 2, 2)]) with accuracy: 137.000000 and alpha: 0.811201

Evaluate your classifier with training dataset
False Positive Rate: 17/100 (0.170000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 183/200 (0.915000)

Evaluate your classifier with test dataset
False Positive Rate: 45/100 (0.450000)
False Negative Rate: 36/100 (0.360000)
Accuracy: 119/200 (0.595000)
```

Implementing detail of part 6(method 2):

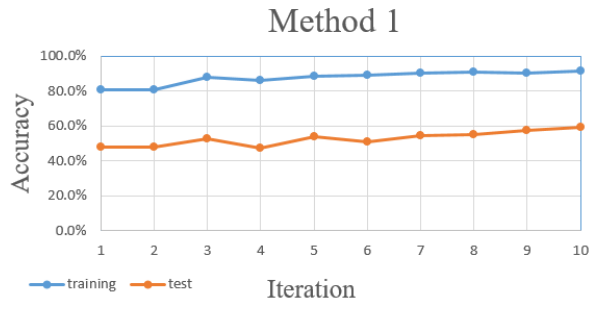
```
'''
Part 6
I think that the threshold and the polarity can be adjusted , so i try to calculate
the average feature value of the face data, and set the threshold to be the integer
closest to the average, which us 18 here. For the polarity part, i calculate the
error rate for each feature. If the error rate is larger than 0.5, indicating that
there's over half of the data are wrongly detected, so to reverse the result, i will
set the polarity to be -1 in this case.
'''
h=np.zeros(featureVals.shape)
clf=[]
error=[]
'''
s, cnt= 0,0
for i in range(len(features)):
  for j in range(len(labels)):
    h[i][j]=int(featureVals[i][j]<0)
    if labels[j]:
      s+=featureVals[i][j]
      cnt+=1
print("avg: %f " % (s/cnt))
'''
for i in range(len(features)):
  cnt=0
  for j in range(len(labels)):
    if (featureVals[i][j]<0 and not labels[j]) or (featureVals[i][j]>0 and labels[j]):
      cnt+=1
  if(cnt<=0.5*len(labels)):
    clf.append(WeakClassifier(features[i], 18,1))
  else:
    clf.append(WeakClassifier(features[i], 18, -1))
for i in range(len(features)):
  clf.append(WeakClassifier(features[i]))
for i in range(len(features)):
  for j in range(len(labels)):
    h[i][j]=clf[i].classify(iis[j])
```

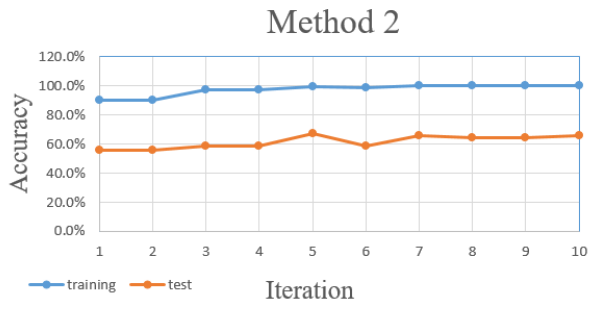
Training and testing accuracy of method 2 when T=10:

```
Evaluate your classifier with training dataset
False Positive Rate: 0/100 (0.000000)
False Negative Rate: 0/100 (0.000000)
Accuracy: 200/200 (1.000000)

Evaluate your classifier with test dataset
False Positive Rate: 7/100 (0.070000)
False Negative Rate: 62/100 (0.620000)
Accuracy: 131/200 (0.655000)

Detect faces at the assigned location using your classifier

Detect faces on your own images
```

Comparison and graphs of accuracy between 2 methods from T=1-10:

traning data accuracy	testing data accuracy	200 samples	training data accuracy	testing data accuracy
81.0%	48.0%	method 2, T=1	90.0%	56.0%
81.0%	48.0%	method 2, T=2	90.0%	56.0%
88.0%	53.0%	method 2, T=3	97.0%	58.5%
86.0%	47.5%	method 2, T=4	97.0%	58.5%
88.5%	54.0%	method 2, T=5	99.5%	67.5%
89.0%	51.0%	method 2, T=6	98.5%	58.5%
90.0%	54.5%	method 2, T=7	100.0%	65.5%
91.0%	55.0%	method 2, T=8	100.0%	64.5%
90.0%	57.5%	method 2, T=9	100.0%	64.0%
91.5%	59.5%	method 2, T=10	100.0%	65.5%





From the chart, we can easily find that both training and testing accuracy are improved.

Detecting result comparison between method 1(first) and method 2(2nd):

For the first and second image, as expected, the detecting result did get improved. However, for the third one, the result got worse. I think it's because of the overall low feature value of this image.

Part III. Answer the questions:

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

```
for t in range(self.T):
    print("Run No. of Iteration: %d" % (t+1))
    # Normalize weights
    # weights = weights / np.linalg.norm(weights)
    weights = weights / np.sum(weights)
```

I faced this problem when I just finish the implementation of Part 2 but got an error message about variable in the train function, and I found that the weight part of the code is quite different from the algorithm, so I modified it as the picture. It finally works well.

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

- Training time is slow
- Restricted to mask, sunglass, items that will cover the face
- Mostly effective when face is in frontal view
- May be sensitive to very high/low exposure (brightness)

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

Beside modifying parameters in the algorithm itself like threshold and polarity, I think we can also find out values of image properties (brightness, saturation, contrast, etc.) conducive for the detection, and preprocess the image to be detected to that condition.

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.

Histogram of Oriented Gradients (HOG)
HOGs are a feature descriptor that has been successfully used for object and pedestrian detection, represented an object as a single value vector as opposed to a set of feature vectors where each represents a region of the image, computed by sliding window detector over an image. HOG descriptor is computed for each position, while the scale of image adjusted to get a HOGs feature. The method used for training following: Sampling P (positive samples) from our training data of the face that we want to detect and extract the HOG descriptors from these samples, then we sampling N (negative samples) from a negative training set that doesn't have or contain any face we want to detect, then extracting HOG descriptors from these sample, which number of dataset N more than P, train a linear Support vector Machine (SVM) on our positive and negative sample. The next step is applying hard-negative mining, for each image after the scaling process in our negative training set, sliding window technique is used across the image, each window calculated the HOG descriptor and the classifier applied, if the classifier showing the wrong classification results in a given window as a face, record the feature vector associated with the false-positive result along with the probability of classification. Next step is sorting the false-positive samples that we found during the stage of hardnegative mining by their confidence of probability, re-retraining the classifier using these hard-negative samples by 3-5 steps and the classifier window is trained and it is ready to be used for the test dataset.

The HOGs can detect the face who have occlusions like using helm, eyeglass, and mask. Adaboost algorithm can perform in real-time on many applications. The HOGs is overall more accurate than V-J for face detection.