HW1 0616086邱彥慈

|  |
| --- |
| code |
| 1. os.listdir() returns a list of image names in the face/non-face directories. 2. Use regular expression to parse pgm images based on pgm format, and store the gray values in numpy array. 3. The format of output is (np,label).   Reference : https://stackoverflow.com/questions/7368739/numpy-and-16-bit-pgm |
| result |
|  |

Part 1: Load and prepare your dataset (10%)

Part 2: Implement Adaboost algorithm (30%)

|  |
| --- |
| code |
| 1. For each feature indexed h, build 3 default classifier with respect to 3 kinds of threshold   : zero, mean(featureVals[h]), and midian( featureVals[h]).   1. The classifier will invert the polarity if (threshold!=0) and (featureVals <0).   (I know this is non-sense and would make the classifier unstable, but I still applied this since I the model would thereby perform better. QQ???)   1. Under the feature[h], calculate the error of each data and sum them up.   That is, error\_ h =   1. If (error\_h < bestError), set the current classifier as bestClf. 2. Looping through all features, we will get a bestClf with bestError. |
| result |
|  |

Part 3: Additional experiments (20%)

|  |
| --- |
| code |
| 1. Test by changing T from 1 to 10 2. In the meantime, record the results of utils.evaluate() in train\_log and test\_log.      1. Plot the line chart of the false positive/negative rate of train and test data.      1. Plot the line chart of the accuracy of train and test data. |
| result |
| 1. The accuracy of train data is maximized on T = 5, from which the accuracy started to decreased. 2. The growth rate of test data became slow as T increased after T = 6, which means that doing more iterations would become less effective on improving accuracy. 3. The more iterations we went through (specifically from 1 to 10), the better this model perform on test data. |

Part 4: Detect face (15%)

|  |
| --- |
| code |
| 1. After parsing the .txt file, read the location (x, y), width, and height of each face. 2. Upon all face information is read, use cv.imread() to read image. 3. Keep the original image and create a grayscale one (🡪image\_gray). 4. Cut off faces from grayscale images (🡪face\_image). 5. Resized the face\_images to 19 x 19 (🡪resized\_image). 6. Increase the contrast of resized\_imageas (🡪inhanced\_images). 7. Green rectangles appear if either resized\_images or inhanced\_images are classified as faces; otherwise there will be red rectangles. |
| result (T=1) |
|  |
| result (T=10) |
| 1. I don’t know why the right picture failed to classify properly, but I guess the reason is that the model had become more sensitive as T grew (overfitting). 2. Or maybe the images processed after cv.cvtColor() still show different pattern from grayscale pictures. |

Part 5: Test classifier on your own images (5%)

|  |
| --- |
| explanaion |
| 1. Select 3 images and write their information to 'data/detect/MyOwnImages.txt'. 2. The first picture didn’t perform well. I guess it’s because that the image size is quite larger than the training data, or people with glasses are harder to be identified. 3. The model performed better on identifying little橋本環奈. I think it might because of the smaller picture size as well as her clear facial features and fair skin. |
| result (T=1) |
|  |
| result (T=10) |
|  |

Part 6: Implement another classifier

I only do modifications on threshold as part 2 show.

Part 7: Problems you meet and how you solve them

1. Failed to import cv2 on VScode > use” import cv2, from cv2 import cv2 as cv” instead.
2. Failed to call a null classifier bestClf = WeakClassifier() > passed none to the parameter, that is, WeakClassifier(…, feature = none, …)
3. To store the accuracy of each iteration, the utils.py is also modified as mentioned in Part 3.
4. I did data augmentation with flip() and rotate(), but nothing is improved.
5. Locating face from scratch is too difficult, as a result I used face detector found on GitHub to automatically located the faces first, then modified them in details.

