ECEN 649 : pattern recognition

HW5 Report

Hsing-Yu Chen

ECEN, Texas A&M University

UIN 829009530

Fall, 2020

▪ Project Description

In this project, I programmed Adaboost to train / test classifiers using given data. The code is written in Python 3.8 on Anaconda environment. My project can be accessed at: https://github.com/peterchen3301/Viola-Jones-Adaboost.git

▪ Imported tools:

Following packages are imported in this project:

Numpy : for faster image / array manipulation

OpenCV : only for reading and showing images, without using any built-in classifier functions.

Matplotlib : for demonstration of performance metrics

JSON : for recording the training results

▪ Training / Testing datasets:

In this project, I use CMU face recognition dataset that consists 500 positive (face) images and 2000 negative (non-face) samples in training set; 472 positive, 2000 negative in test set. All images are grayscale, 19 * 19 in size.

▪ Extracting Haar features:

I defined 9 feature patterns that maps along the sample image with various sizes and positions. 8055 Haar features are generated in total.

haar0 = [[1, -1], [1, -1]]

haar1 = [[1, 1], [-1, -1]]

haar2 = [[1, -1, 1], [1, -1, 1]]

haar3 = [[1, 1], [-1, -1], [1, 1]]

haar4 = [[1, -1], [-1, 1]]

haar5 = [[1, 1, 1], [1, -1, 1], [1, 1, 1]]

haar6 = [[1, 1, 1], [-1, -1, -1], [1, 1, 1]]

haar7 = [[1, -1, 1], [1, -1, 1], [1, -1, 1]]

haar8 = [[1, 1, 1], [-1, -1, -1]]

These Haar features define feature values from given sample images, which are transferred to integral images in order to speed up the calculations of rectangular sums.

- **Implementing ERM for decision stumps**

In this program, we train each weak classifier (corresponding to each of 8055 Haar features) by looping every training samples and get the best threshold (decision stump), which is the feature value at where the minimal sample error occurs. The definition of sample error is:

$$min(S^+ + T^- - S^- , S^- + T^+ - S^+)$$

Where T^+, T^- note for the total weight of positive / negative samples, while S^+, S^- note for the accumulated positive / negative weight till current iteration.

The best threshold for each weak classifier is used to determine whether it classifies a sample correctly if following equation holds and *vice versa*.

$$polarity * feature\ value < polarity * best\ threshold$$

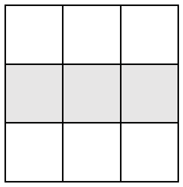
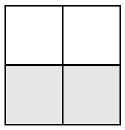
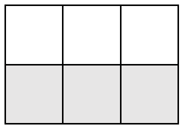
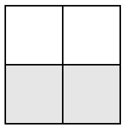
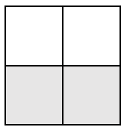
- **Implementing Adaboost predictor**

In each round of training, all 8055 weak classifiers loop over training samples to solve for total weighted error. After completion, we select the one with lowest error and append it as one of the cascades in the strong classifier.

In this project, we are required to train 10 rounds and get the cascades of size 1, 3, 5 and 10 within the strong classifier.

- **Answers to given problems**

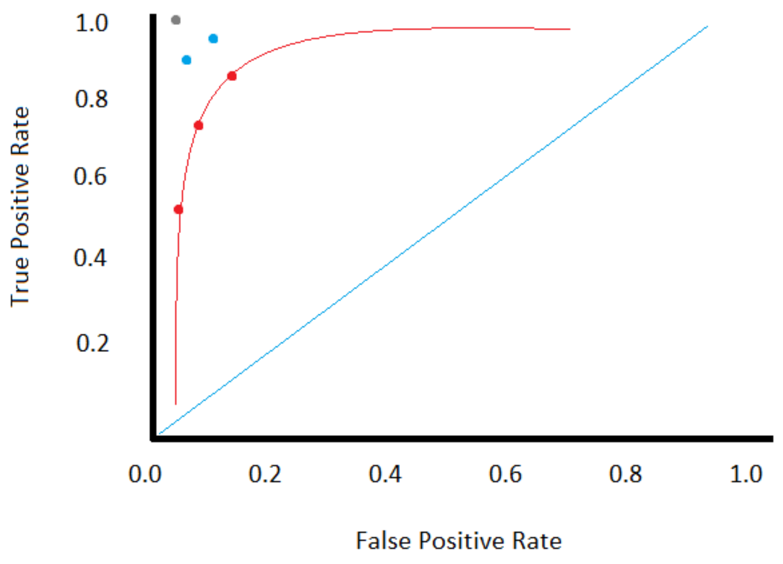
1. The 10 best weak classifiers obtained from this program:

# of cascade	Pattern	Size (in pixels)	Position (upper left)	Training error
1st		3 * 3	(10, 3)	0.18
2nd		2 * 2	(1, 17)	0.26
3rd		2 * 3	(0, 15)	0.36
4th		2 * 2	(2, 15)	0.48
5th ~ 10th		2 * 2	(2, 15)	0.50

2. The combined strong classifier is the cascade of 1, 3, 5 and 10 weak classifiers in this project:

Cascade size	Accuracy	True positive	False positive	True negative	False negative	Avg. drops per sample
1	0.89	0.67	0.03	0.97	0.33	0.11
2	0.99	0.98	0.05	0.95	0.02	0.12
3	0.99	0.99	0.01	0.99	0.01	0.12
5	0.99	0.99	0.01	0.99	0.01	0.12
10	0.99	0.99	0.01	0.99	0.01	0.12

3. The ROC curve of the combined classifiers after running 1, 3, 5, and 10 boosting rounds when applied to the test set:



Where the red points note for the size # 1 classifier, blue for size # 2 and gray for size #3 and more. By manually adjusting the training threshold in each decision stump, the testing result moves along a curve as a tradeoff between high detection rate (high TP, high FP) and low sensitivity (low TP, low FP). Typically, the result obtained by applying ERM strategy that chooses the best threshold within every weak classifier will be the optimal on the curve, which is closest to the (1.0, 1.0) position.

4. Space for improvement

I found some potential spaces of improvement for this project. First, the accuracy rate is much be higher than I expected. This may attribute to the bug at somewhere within my coding ,even though I failed to find such in my program after carefully check out image loading / feature value obtaining parts of coding. It may also attribute to the insufficient amount of testing samples. Also, it may lead to abnormally high testing accuracy if the test samples are too similar with training samples or themselves.

Generally, there are several method to raise the classifying accuracy. The first is to improve the amount and variance of samples within training set, and the reason is self-explainable. However, if for some reason one cannot simply collect more training data, then data augmentation may be helpful, in which we apply several transformations to original training images like distortion, rotation, contrast stretching, dilation / erosion… and so on to generate slightly different set of images. By doing this, we can augment the training dataset only by using current samples.

Furthermore, we can also augment the set of weak classifiers (in which we use Haar features) that includes more patterns, sizes and positions. Note that this will be a tradeoff between training speed.

There are also some methods to increase the performance speed. The most typical one is to randomly select a portion of images from training dataset (instead of looping all of them) while training each weak classifier, which is a method that modern ML/DL frame adopts. Moreover, particularly in my program, I use list type instead of numpy arrays as container of datasets and their variables, which is an easy-to-use data structure with tradeoff with performance speed. This is a point where we can work on to optimize this project.