ECE661 Homework 8

Lin Yuan

Section 1 Problem Description

This homework addresses the problem of object detection using AdaBoost. The problem of object detection is formulated into a binary classification problem, solved with AdaBoost. AdaBoost is considered as one of the best off-the-shelf binary classification algorithm which carries several merits: realtime online testing, consistent generalization error, and high classification rate. On the commonly used datasets like UCI dataset, AdaBoost together with weak classifier Decision Tree is reported to perform very well, along with other competitors such as SVM. After AdaBoost is proposed by [1], it has been proven to be convergent in [2], and universally consistent in [3], which explains why AdaBoost performs well in avoiding the overfitting problem. In this homework, we are going to implement both the original (monolithic) AdaBoost algorithm, together with the Viola-Jones variant – cascaded AdaBoost algorithm.

Section 2 Solution

All major algorithms are explained quite well in the paper of [4], but a few keypoints need to be further explained for successful implementation.

(1) AdaBoost

The AdaBoost algorithm is described in Table 1 of [4], and is briefly recapitalized here with special directions for implementation.

- Initialize positive and negative weights. Note here that the weights are different depending on positive and negative sample counts.
- For t=1 to T, do the following steps. Note that T is the number of weak features you want to select, and this procedure can be done online, by storing the weights from the last iteration t. Which can save computation time in the cascaded version.
  - Normalize the weights to sum to 1.
  - Find best weak classifier using the procedure in Section 3.1 of [4]. Due to the nature of decision stump, we can have such a efficient implementation.

  - Store the best weak feature and classifier, and the weak classification error, for time t.
  - Update the weights using $w_i^t = w_i^{t-1} \times e^{\frac{\epsilon_t}{1-\epsilon_t}}$.

- Output the final strong classifier. Note that for the calculation of α_t, if the weak classification error ϵ_t is 0, α_t may be infinite. (Which happens at the case where there is only one negative / positive sample.) In this case, special consideration needs to be taken.

(2) Cascaded AdaBoost

The variant cascaded AdaBoost is described in Table 2 of [4]. Below are a few clarifications for implementation of the cascaded AdaBoost algorithm.

- f needs to be lower than 0.5, while d can be always kept at 1, however, if one allow d to be a little bit lower than 1, it may help to avoid overfitting, but may be computational costly to find a suitable threshold. In the case of using $d = 1$, one can simply keep the threshold for the strong classifier of each stage as the lowest score for positive samples.

- Each iteration is about training a monolithic AdaBoost classifier on the specific set of positive and negative training samples, until this monolithic classifier, on this specific training set, achieves the false positive rate lower than f, and the detection rate higher than d.

- When evaluating for F_i and D_i, this evaluation is done using the cascades classifier of all stages until now, on the set of all positive and negative samples.

- For the inner loop, the training of the monolithic AdaBoost can be done online by storing the previous sample weights.

- For the outer loop, every time the size of the negative samples is shrank to the false positives of the previous stage, and the positive samples are kept pretty much the same.

(3) Decision Stump

Section 3.1 of [4] details how to find the weak classifier, i.e. a decision stump for a feature. It can be described as follows:

- Given the vector $\mathbf{X}_i$ of feature i of all images.

- Sort $\mathbf{X}_i$ in decreasing order.

- Loop from the top to the bottom, and find the total positive weights above the current sample's ith feature value (inclusive) and the total positive weights below the current samples's ith feature value. Same for the negative weights. Note that in order to keep the information of "inclusive", the found threshold θ, should better be stored with double type, and minus a small fraction number such as "0.1" to the end.
- Store the minimum weight error found by splitting the dataset by the threshold θ.

(4) Finding Haar Features

The Haar features can be easy to describe, but when actually coding for it, it might take a little bit effort. Here is the algorithm for finding the vertical and horizontal Haar features.

- Each horizontal or vertical feature is determined by 6 points.
- These 6 points are determined by the length and height of the rectangle, and the position of the upper-left corner of the rectangle.
- Loop through the image pixel positions to fix a upper-left corner.
- Inner loop, through the feasible length (from pixel position to the border), and an even number
- Inner inner loop through the feasible height(from pixel position to the border), even or odd number
- Now depending on the vertical or horizontal configuration, one can determine $x_1, ..x_6$.

(5) Calculating Integral Image

This is well elaborated in [4].

Section 3 Discussion

Beyond the above keypoints for successful implementation for AdaBoost, here are a few more tips for debugging and memory allocation.

- Because haar features are memory-expensive, we need to store them to files first. When storing and reading the haar features, using binary mode can be much faster than storing it using plain text.

- Debugging AdaBoost is not an easy task for C++. Because usually it might be the case that you won't know where the bug is only after a long time of computation. Always store the cascades stages into files first, and then read in the file directly to debug for the specific stage that the program has bug with.
- Build with release option and run it on a fast computer.
- Avoid memory leak by deleting allocated array in time.

Section 4 Results

The results shown in Fig.1 and Fig.2 are obtained with a cascaded AdaBoost with $f = 0.3$ and $d = 1$ for each cascade. The obtained cascade classifier is listed in Table1. The first two weak features selected for car detection on the 20 by 40 image. The result of monolithic AdaBoost with 200 features is also shown in Table2. Note that the 200 features monolithic AdaBoost can also achieve 0 false positive rate on the training set.

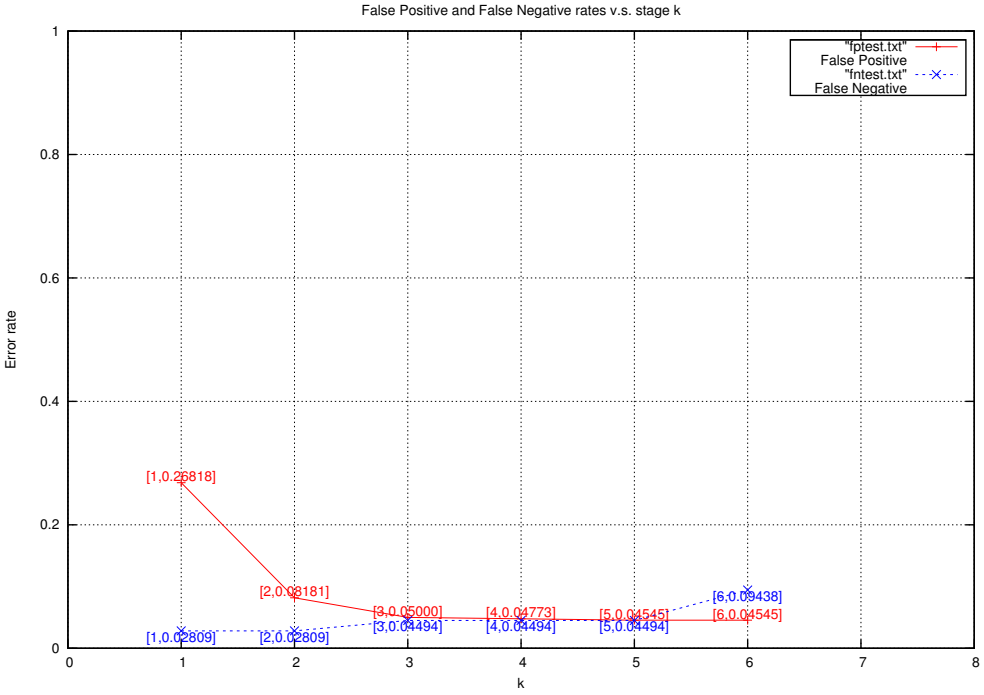

Figure 1: False positive rate and false negative rate v.s. the number of stages k, for test images

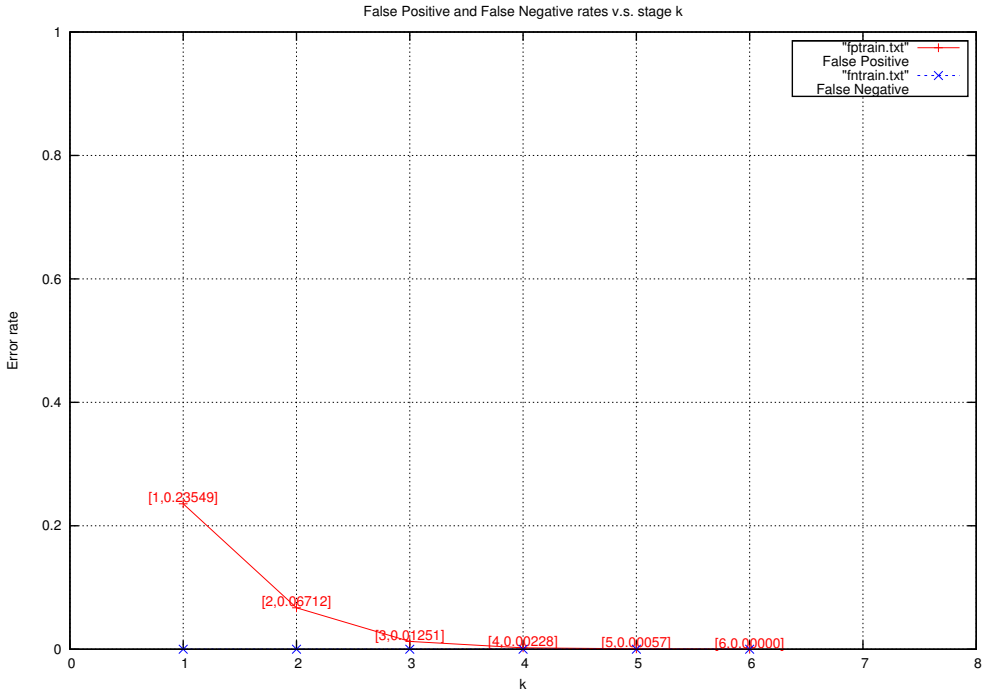

Figure 2: False positive rate and false negative rate v.s. the number of stages k, for training images

Stage	Haar Index	ϵ_t	polarity	θ
1	23378	0.14997	1	-1101.1
1	72373	0.25117	1	-56.1
1	123492	0.27796	-1	17.9
1	158759	0.30367	-1	97.9
5	2700	0.06690	1	4.9
5	120047	0.07818	1	465.9
5	92760	0.06835	-1	-115.1
5	31300	0.08337	-1	-87.1
6	15519	0.0	1	3732.9

Table 1: The cascaded classifier at some stages

	False positive	False negative
Monolithic(200 features)	0.011	0.073
Cascaded(6 stages)	0.045	0.056

Table 2: Comparison of monolithic and cascaded AdaBoost applied on the test dataset

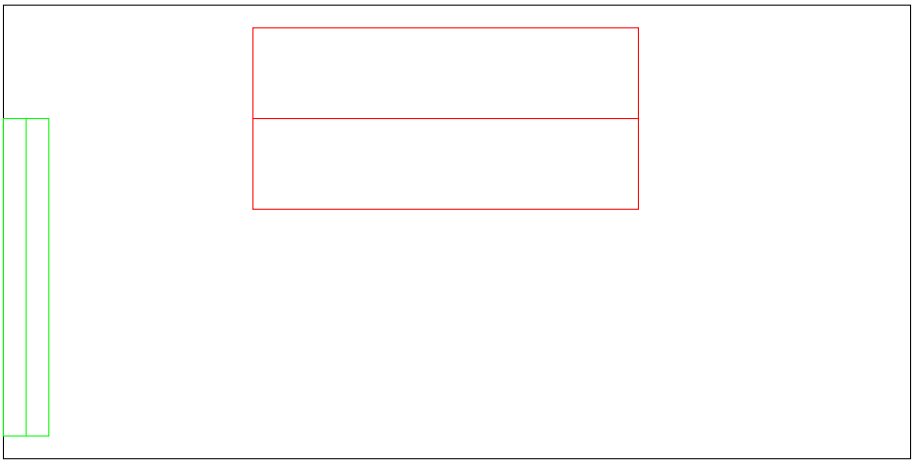

Figure 3: First two weak features identified for the 20 by 40 car image. Red is feature No. 23378, green is feature No. 72373.

References

[1] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," 1997.

[2] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Annals of Statistics*, vol. 28, p. 2000, 1998.

[3] P. L. Bartlett and M. Traskin, "Adaboost is consistent," *J. Mach. Learn. Res.*, vol. 8, pp. 2347–2368, December 2007.

[4] P. A. Viola and M. J. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004.