NORTH CAROLINA STATE UNIVERSITY
DEPARTMENT OF ELECTRICAL ENGINEERING

ECE 763: COMPUTER VISION

Spring 2020 **Project 2** Dr. Tianfu Wu

ECE 763: Computer Vision: Models, Learning, and Inference

Adaptive Boosting for Face Detection Model

John McDonald

March 16th, 2020

Abstract

In this paper we expand upon the famous Viola-Jones object detection framework. Through creating haar features and testing them against a training set, we can find some simple geometric shapes that are extremely effective at image classification problems. In this paper, we specifically implement a face detection classification model. After testing the haar features against a training set of face images, the haar features with the best classification scores are selected to be weak classifiers. These weak classifiers are chained together using the AdaBoost algorithm to develop a single effective strong classifier. In this paper, the model build is effective at classifying a testing set with only 8.8% classification error. This error is increased by a high rate of false negatives in the testing set. Furthermore, this model is potentially subject to overfitting, as the strong classifier when testing against the training set performed at significantly less error, performing with an F_i score under .01. Evaluation of the final model and adaboost algorithm steps are included.

Objective

The objective of the project is to build an AdaBoost model for 2 class face detection. This includes:

- Data Preparation, using 2 classes of positive and negative faces. Recommended 20x20 grey images.
- Development of feature vector comprised of Haar features.
- Implementation of AdaBoost algorithm, including initialization of data weights, the calculated weighted error for features, selection of weak classifiers, derivation of classifier weight, and updated data weights
- A finalized strong classifier, comprised of boosted stumps (weak classifiers)
- Evaluation of all steps of the AdaBoost algorithm, and the finalized strong classifier

Data Preparation

The dataset and preparation used for this model are like that of the implementation in project 1.

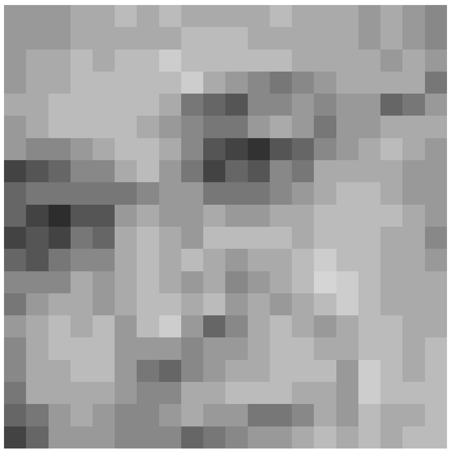
The dataset used for this projected as pulled from github page for face resources, https://github.com/betars/Face-Resources [1]. The 7th dataset listed contains a Face Detection and Data Set Benchmark, of 5k images which is sufficient for our project, found at http://vis-www.cs.umass.edu/fddb/ [2]. This dataset was chosen is it contains the annotations for images, and a helpful Readme with instructions, to get started in cropping and creating the needed face and non-face image sets.

Once the dataset was selected, it was used to create two folders each of 1100 images, one for faces and another for non-faces, both at 20 x 20 resolution, as RGB images. All training face images, and test images were separated within the python program to quickly and easily change training testing split.

To double the number of negative images for training and testing, the non-face crops were rotated by 90 degrees and added to the total training and testing arrays. When creating the image array in the python file, all images were read to the array as greyscale, in order to keep processing power low, as instructed by the project prompt.

Example:

a)

b) 

c)

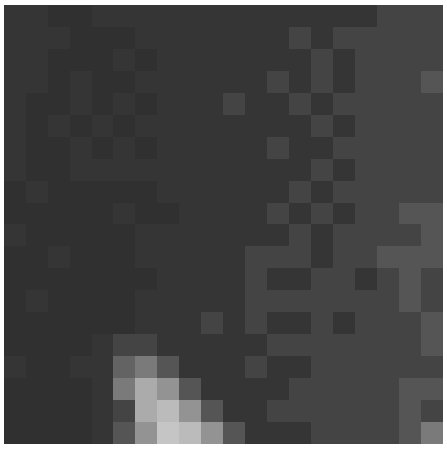
d) 

Figure 1. a) Original image from FDDB dataset b) used annotation from FDDB dataset to crop face image
c) Original image from FDDB dataset d) randomly selected negative image that doesn't overlap face

Creating Haar Features

The haar feature is a simple geometric shape used to overlap a image and find patterns. In the referenced ViolaJones paper in the prompt, we see that certain geometric shapes were chosen as highly effective in classifying faces. Seven template haar types were created based on these previous results. Their shapes were created using NumPy array and visualized using matplotlib plots Their shapes are displayed in Figure 2 below.

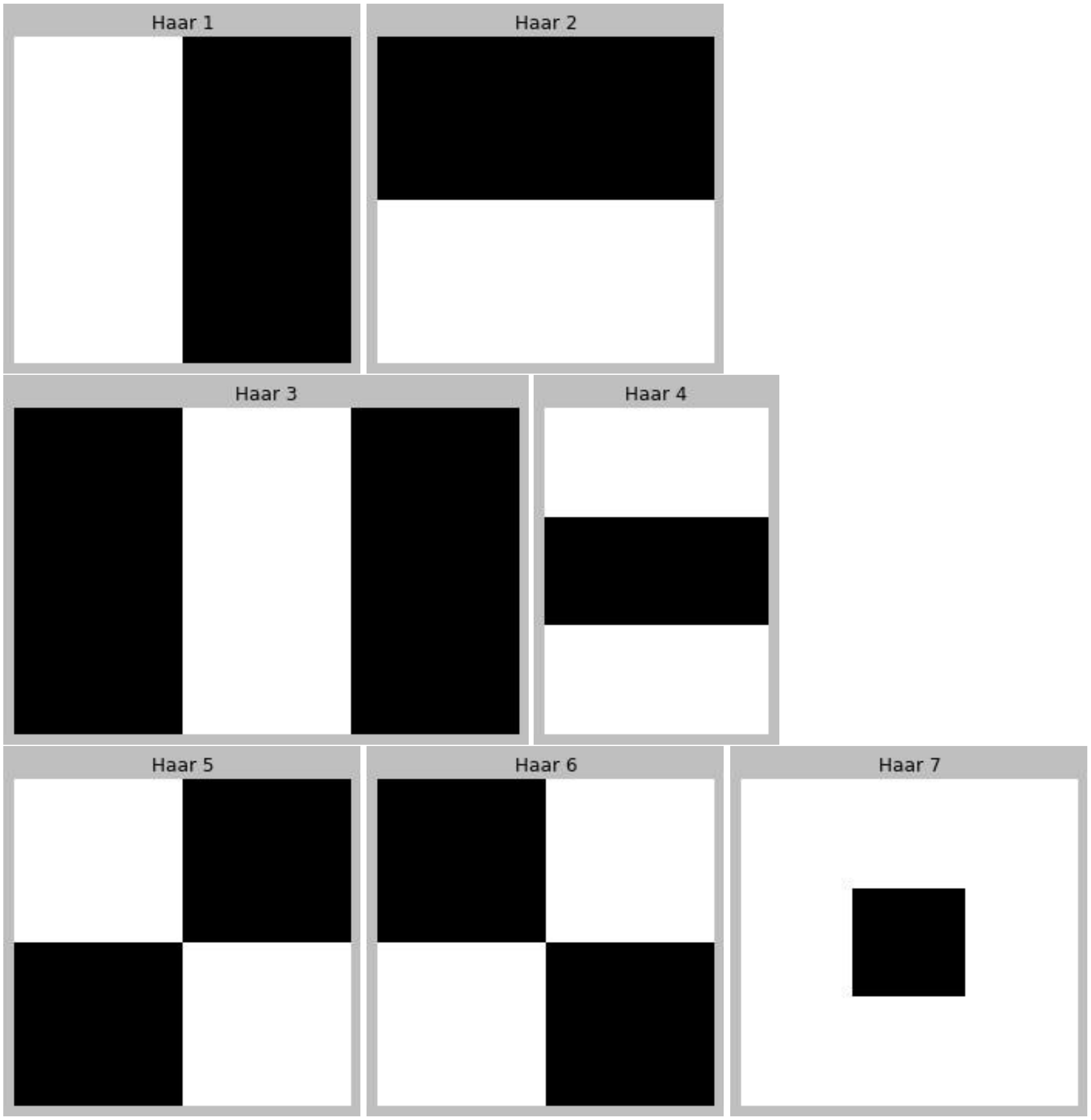


Figure 2. Seven haar feature types manually created using NumPy array. Varying in pixel dimensions between 2x2, 2x3, 3x2, and 3x3. Each is scaled larger, up to a factor of 7, to create feature vector for 20x20 pixel images.

To comprehensively create our haar features vector, these seven haar types are scaled larger, up to 7 times as large. To operate within the dimensions of our 20x20 pixel window size, this creates a total of 7742 features for our vector. Each feature is then scored on the training set and characterized by a polarity and optimal threshold.

The computed haar score for each feature is stored in the scores array, but not displayed in this report, as there are scores for 7742 features x 2970 samples. Too many to include in the report. Albeit, these scores are used to find polarity and threshold for each feature. And ultimately build the cascade.

Creating the Haar Cascade

The haar cascade is built using the algorithm in the project prompt. This includes multiple steps [3]:

- Initializing equal weights for all training images
- Looping the cascade until the F_i score is below a target score (.05 for this model)
- Computing the weighted error at each iteration for each feature
- Adding the best feature to the cascade
- Updating weights for training images

The error for each feature at each iteration is calculated and selected as the next best weak classifier as shown:

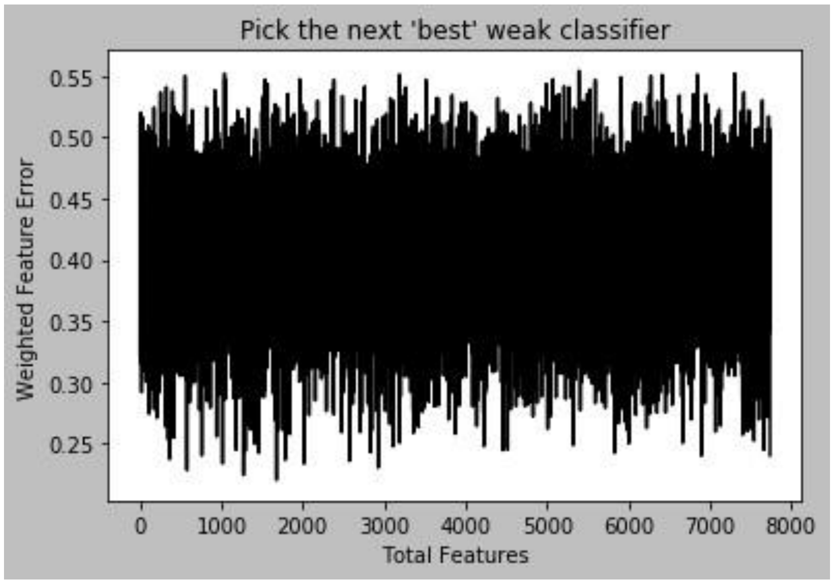
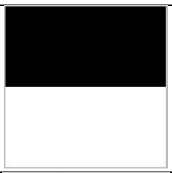
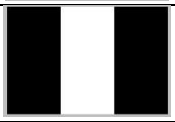


Figure 3. Weighted error for each feature is computed. Adding the feature with lowest weighted error as the next weak classifier in the cascade, during adaboost algorithm.

The model took 51 minutes to train and achieve a target F_i of below .05 score. The final cascade size was 112. Detailed explanation of the 10 best weak classifiers in the cascade are shown below. After making minor changes to the algorithm throughout testing, the feature indexes may change. But these feature shapes tend to consistently be the most effective at classifying faces and are often chosen as the first classifiers when building the cascade.

Top 10 Weak Classifiers

No.	Index	f_i	F_i	TP	TN	FP	FN	Threshold	Classif. Error	Feature Size	Feature Shape
1	$\frac{3376}{7742}$	.23	.23	.71	.84	.16	.29	-210.12	.20	6x6	
2	$\frac{6117}{7742}$	.28	.23	.65	.78	.22	.35	142.68	.26	6x4	

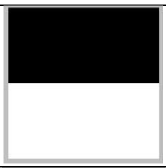
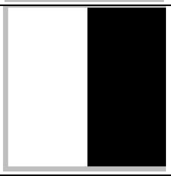
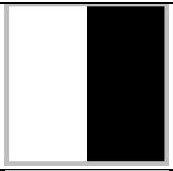
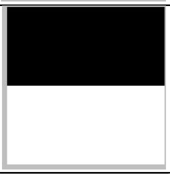
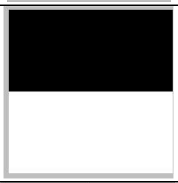
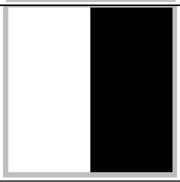
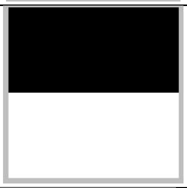
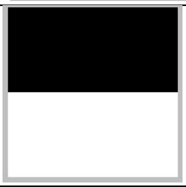
3	$\frac{5777}{7742}$	.25	.17	.66	.84	.16	.34	-75.28	.22	4x4	
4	$\frac{6556}{7742}$	.28	.17	.65	.78	.21	.35	139.11	.26	6x6	
5	$\frac{6643}{7742}$	.37	.16	.72	.54	.45	.27	144.69	.40	2x2	
6	$\frac{6061}{7742}$	.31	.15	.64	.74	.26	.36	-243.38	.29	8x8	
7	$\frac{972}{7742}$	.35	.15	.58	.72	.28	.42	32.83	.33	4x4	
8	$\frac{1705}{7742}$	.36	.14	.73	.55	.47	.27	146.27	.39	2x2	
9	$\frac{3650}{7742}$	.42	.14	.73	.44	.56	.26	93.04	.46	2x2	
10	$\frac{181}{7742}$	.33	.13	.58	.75	.25	.41	-39.68	.30	4x4	

Table 1. Detailed evaluation of first 10 selected Weak Classifiers in cascade

As you can tell, the best classifiers share for the most part the same feature type. As the cascade begins to increase in size, more feature types are explored and added to the cascade.

Next, the cascade performance is compared for three different sizes of the cascade. The classification performance for next weak classifier is compared at each step. Also the performance of the entire cascade is compared, using the strong score as an evaluation. The F_i score for the cascade improves from .17 at step 2, to 0816 at step 56, and ultimately .0497 at step 112. Graphs showing their performance and summarizing confusion matrices are displayed below.

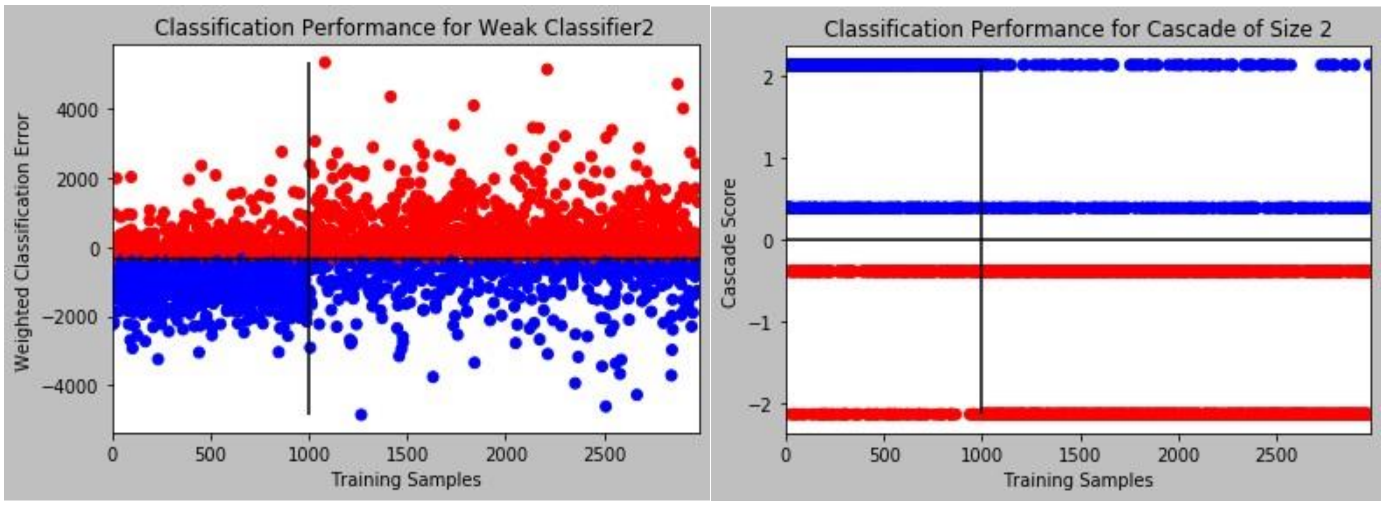


Figure 4. Details of adaboost algorithm for cascade step 2 in iteration. The verticle line at 990 splits positive samples on the left with and negative samples on the right. Blue represents what was classified as positive, and red as negative.

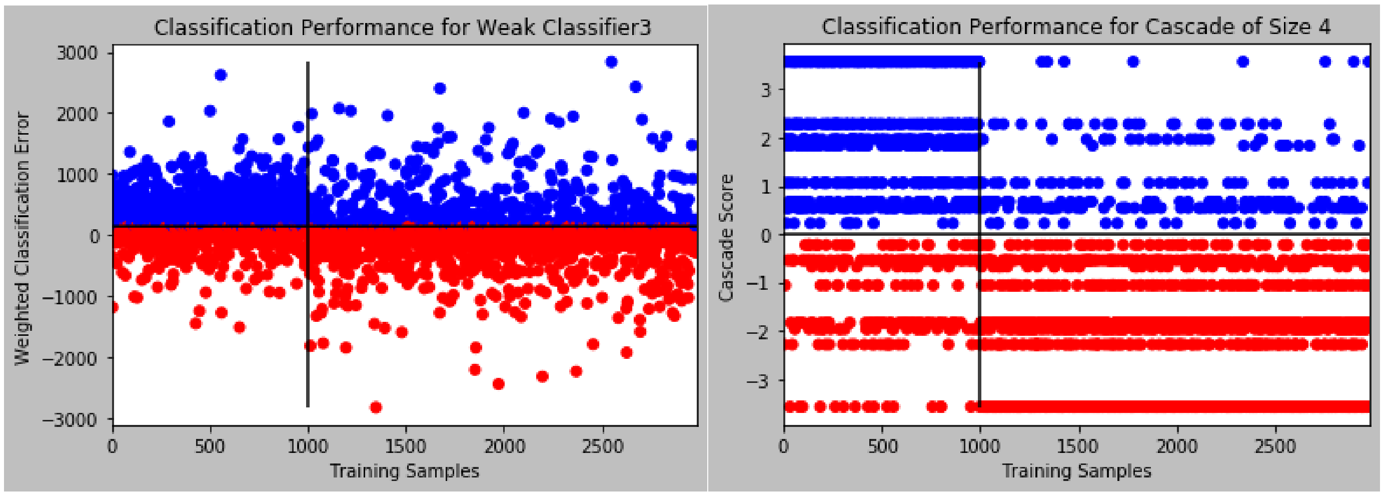


Figure 5. Details of adaboost algorithm for cascade step 4 in iteration. The verticle line at 990 splits positive samples on the left with and negative samples on the right. Blue represents what was classified as positive, and red as negative.

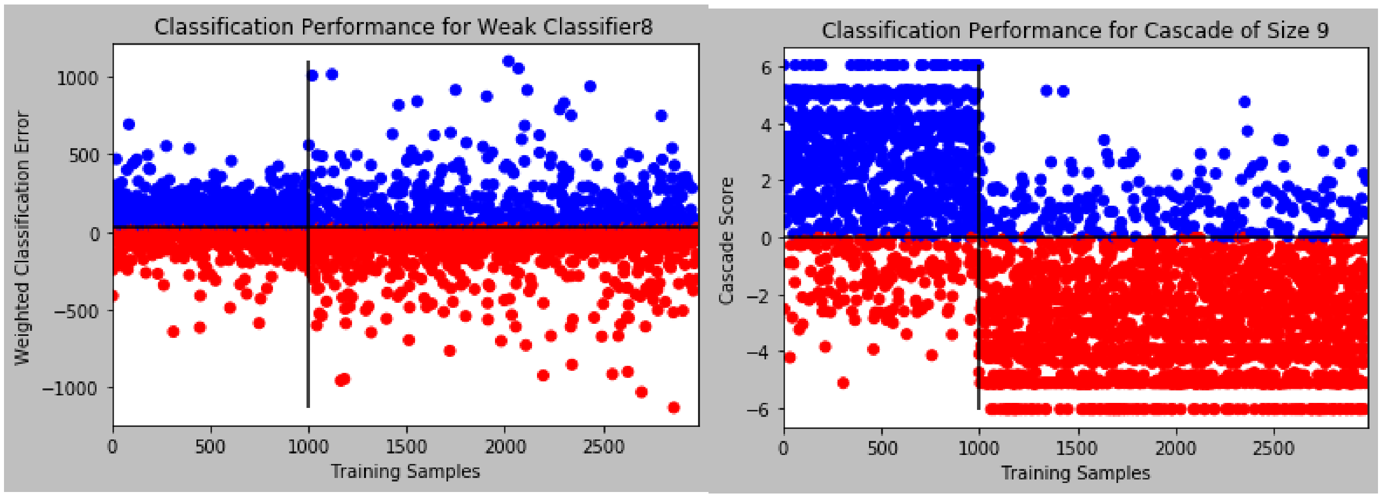


Figure 6. Details of adaboost algorithm for cascade step 9 in iteration. The verticle line at 990 splits positive samples on the left with and negative samples on the right. Blue represents what was classified as positive, and red as negative.

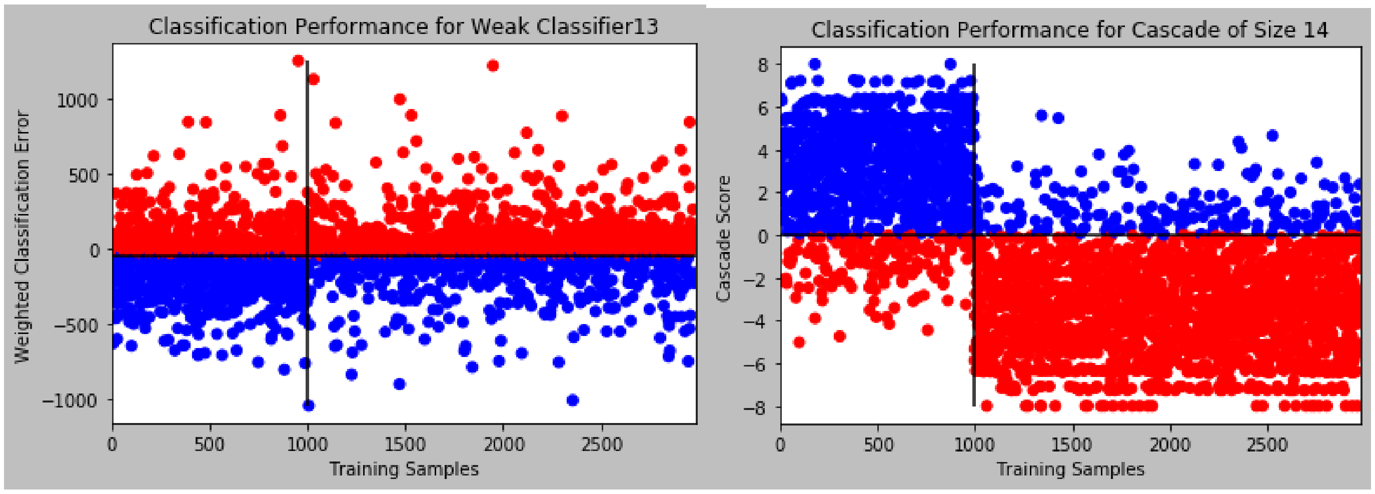


Figure 7. Details of adaboost algorithm for cascade step 14 in iteration. The verticle line at 990 splits positive samples on the left with and negative samples on the right. Blue represents what was classified as positive, and red as negative.

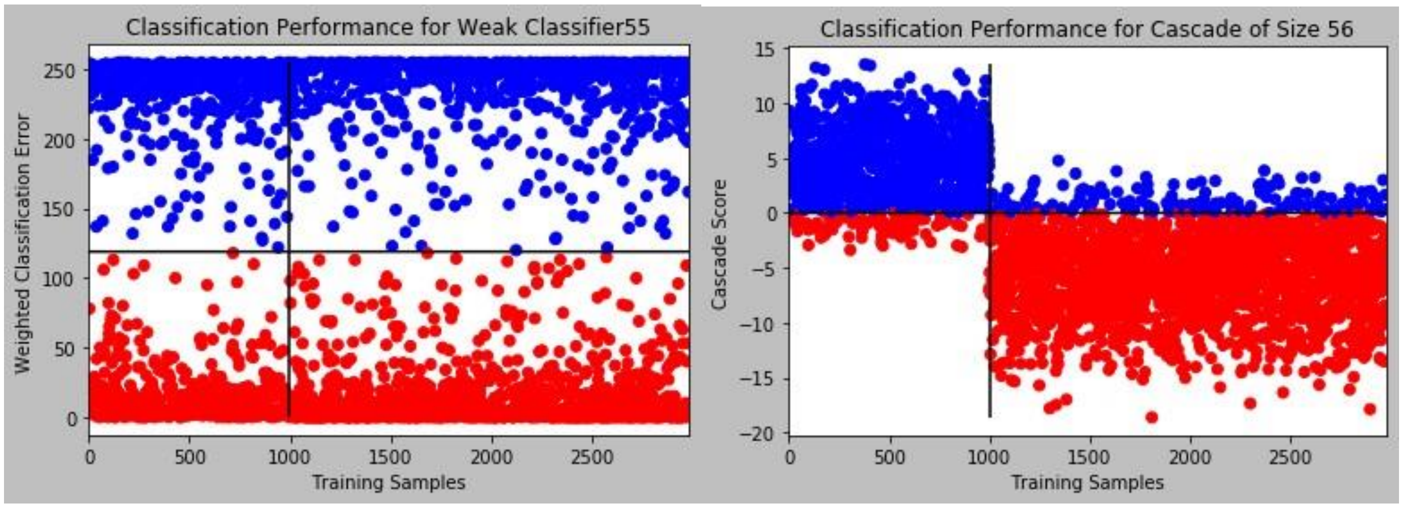


Figure 8. Details of adaboost algorithm for cascade step 56 in iteration. The verticle line at 990 splits positive samples on the left with and negative samples on the right. Blue represents what was classified as positive, and red as negative.

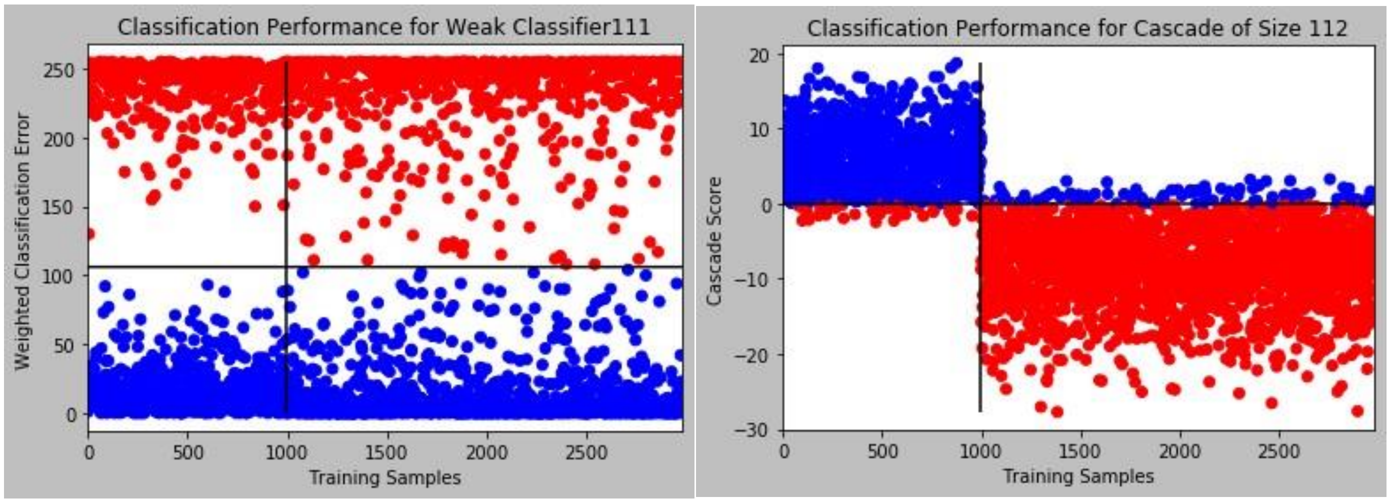


Figure 9. Details of adaboost algorithm for cascade step 294 in iteration. The verticle line at 990 splits positive samples on the left with and negative samples on the right. Blue represents what was classified as positive, and red as negative.

Note that the weighted classification error is much higher for weak classifiers at the beginning of the cascade, compared to later such as at step 112. Also, as the strong classifier is developed, the cascade score range increases for the sample space.

Testing Cascade

After the strong classifier is fully build and the adaboost algorithm is completed, the finalized strong classifier is run against a testing set. The performance is shown in the below graph.

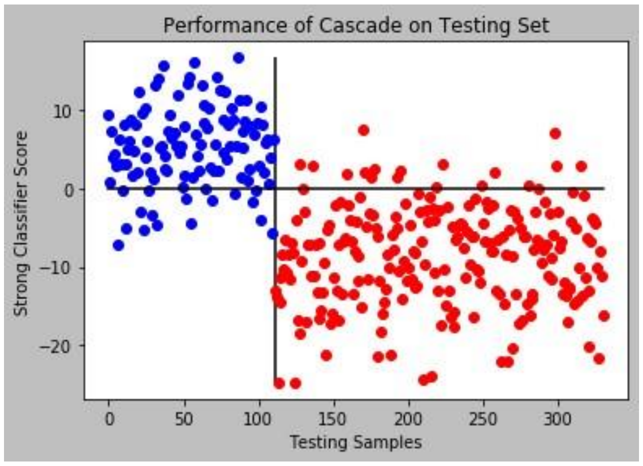


Figure 10. Performance of finalized strong classifier on testing set. The y axis representing strong scores and x axis representing the testing images. The blue dots represent positive images and red dots represent negative images. As we can see, blue dots falling below the origin represent false negatives, and red dots falling above the origin represent false positives. Representing thus the confusion matrix of the performance of our model.

Additional metrics are listed below:

Test F_i	**Total FP**	**Total FN**	**TP Rate**	**TN Rate**	**FP Rate**	**FN Rate**
.1198	19	17	.8468	.9136	.0863	.1531

Table 2. Evaluation metrics for strong classifier when run against the testing set.

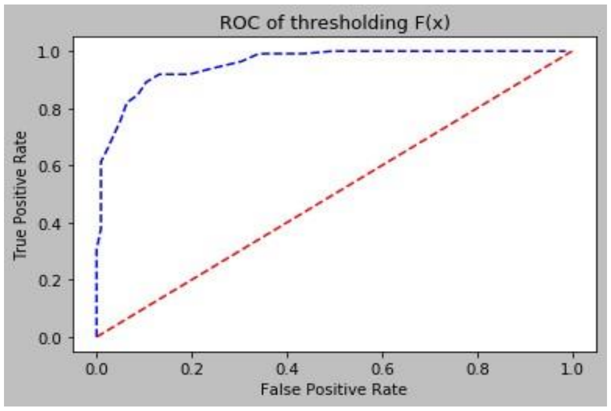


Figure 11. ROC Graph for thresholding F(x) in strong classifier, from minimum of strong classifier scores to maximum of strong classifier scores.

Conclusion

All objectives of the project prompt were successfully met. Basic haar features were manually created and used to populate a full feature vector. The feature vector was successfully scored against the training set, and thresholds and polarities for the haar features were stored. The adaboost algorithm managed to chain and select weak classifiers based on lowest feature errors, thus incrementally reducing the F_i score with each cascade iteration. The finalized strong classifier was run against the testing set, and while there were signs of some overfitting, the classifier did manage to quickly classify each test sample with moderately high accuracy. The ROC curve was created based on changing the threshold for the strong classification score.

My main critic of the project is the sign of overfitting test errors at an F_i of about .12, with lower training error at below F_i of .05. If I had more time, I would like to store the training cascade error and test error over time, and find an optimal fit and cascade size for this model.

References

[1] Betars, "betars/Face-Resources," *GitHub*, 27-Apr-2017. [Online]. Available: https://github.com/betars/Face-Resources. [Accessed: 27-Feb-2020].

[2] "Face Detection Data Set and Benchmark Home," *FDDB : Main*. [Online]. Available: http://viswww.cs.umass.edu/fddb/. [Accessed: 27-Feb-2020].

[3] Singh, H., 2020. *Object Detection Using Haar-Like Features.* [Online] Cs.utexas.edu. Available at: <http://www.cs.utexas.edu/~grauman/courses/spring2008/slides/Faces_demo.pdf> [Accessed 13 March 2020].