AdaBoost + RealBoost

Project -2

BOOSTING TECHNIQUES

Vaishnavi Ravindran | UID: 805227216 | 11/18/2018

Part 1: Constructing Weak Classifiers:

Objective:

1. Compute the features $\{f_j\}$ by applying each Haar filter to the integral images of the positive and negative populations.
2. Find the polarity and threshold of each weak classifier.

Observations:

1. After applying the Haar Filters to the integral images of the positive and negative populations we get "activations" that is of the following size

   (10032, 37194)
2. Calculating threshold and polarity of each weak classifier: These values were calculated using the procedure mentioned in the Viola and Jones paper [1].

   Threshold:

   a. Sort all the features (activations) in increasing order.
   b. At each sample, compute the error given by:
      $e = \min (S^+ + (T^- - S^-), S^- + (T^+ - S^+))$ -----eqn(1)

      where, T+ is the total sum of positive example weights, T- is the total sum of negative example weights, S+ is the sum of positive weights below the current example and S- is the sum of negative weights below the current example S–.
   c. Find the minimum value of e and use the feature value of the corresponding sample as the threshold.

   Polarity:

   a. Find the polarity at each sample by using the procedure:
      - If $S^+ + (T^- - S^-), > S^- + (T^+ - S^+)$, then set polarity to 1
      - Else set to 0
   b. Pick the polarity of the sample which has been chosen for the best threshold.

   It is clear from the above procedure that as the weights of the sample change, the threshold and polarity of the filters will change accordingly.

Function for Threshold and Polarity calculation:

```
def calculate_threshold_and_polarity(self, data, labels, weights_data_points, wc_activations_entire, num_cores,
                                     hnm=False):
    wc_activations = wc_activations_entire[self.id, :]
    if (num_cores == 1):
        save_dir_indices = './sorted_filters/wc_sortedindex' + str(self.id) + 'subset.npy'
        save_dir_features = './sorted_filters/wc_sortedfeatures' + str(self.id) + 'subset.npy'
```

```
else:
        save_dir_indices = './sorted_filters/wc_sortedindex' + str(self.id) + '.npy'
        save_dir_features = './sorted_filters/wc_sortedfeatures' + str(self.id) + '.npy'
    if save_dir_indices is not None and os.path.exists(save_dir_indices):
        sorted_indicies = np.load(save_dir_indices)
    else:
        sorted_indicies = np.argsort(wc_activations)
        if save_dir_indices is not None:
            np.save(save_dir_indices, sorted_indicies)
    if save_dir_features is not None and os.path.exists(save_dir_features):
        sorted_features = np.load(save_dir_features)
    else:
        sorted_features = np.sort(wc_activations)
        if save_dir_features is not None:
            np.save(save_dir_features, sorted_features)
    searchval = 1
    indicesOfFaces = np.where(labels == searchval)[0]
    searchval = -1
    indicesOfNonFaces = np.where(labels == searchval)[0]
    weightsOfFaces = np.array(list(weights_data_points[indicesOfFaces]))
    weightsOfNonFaces = np.array(list(weights_data_points[indicesOfNonFaces]))
    afs = np.sum(np.array(weightsOfFaces))
    abg = np.sum(np.array(weightsOfNonFaces))
    fs = 0
    bg = 0
    besterror = 99999999999999999999
    for i in range(0, data.shape[0]):
        idx = sorted_indicies[i]
        curent_label = labels[idx]
        if (curent_label == 1):
            fs = fs+ weights_data_points[idx]
```

```
        else:
            bg = bg + weights_data_points[idx]
        left = bg + (afs - fs)
        right = fs + (abg - bg)
        error = min(left, right)
        if (left < right):
            polarity1 = -1
        else:
            polarity1 = 1
        if (error < besterror):
            besterror = error
            bestPolarity = polarity1
            bestThreshold = sorted_features[i]
    self.polarity = bestPolarity
    self.threshold = bestThreshold
    polarity_threshold = np.empty(shape=2)
    polarity_threshold[0] = (bestPolarity)
    polarity_threshold[1] = bestThreshold
    return polarity_threshold
```

ADABOOST

Objective:

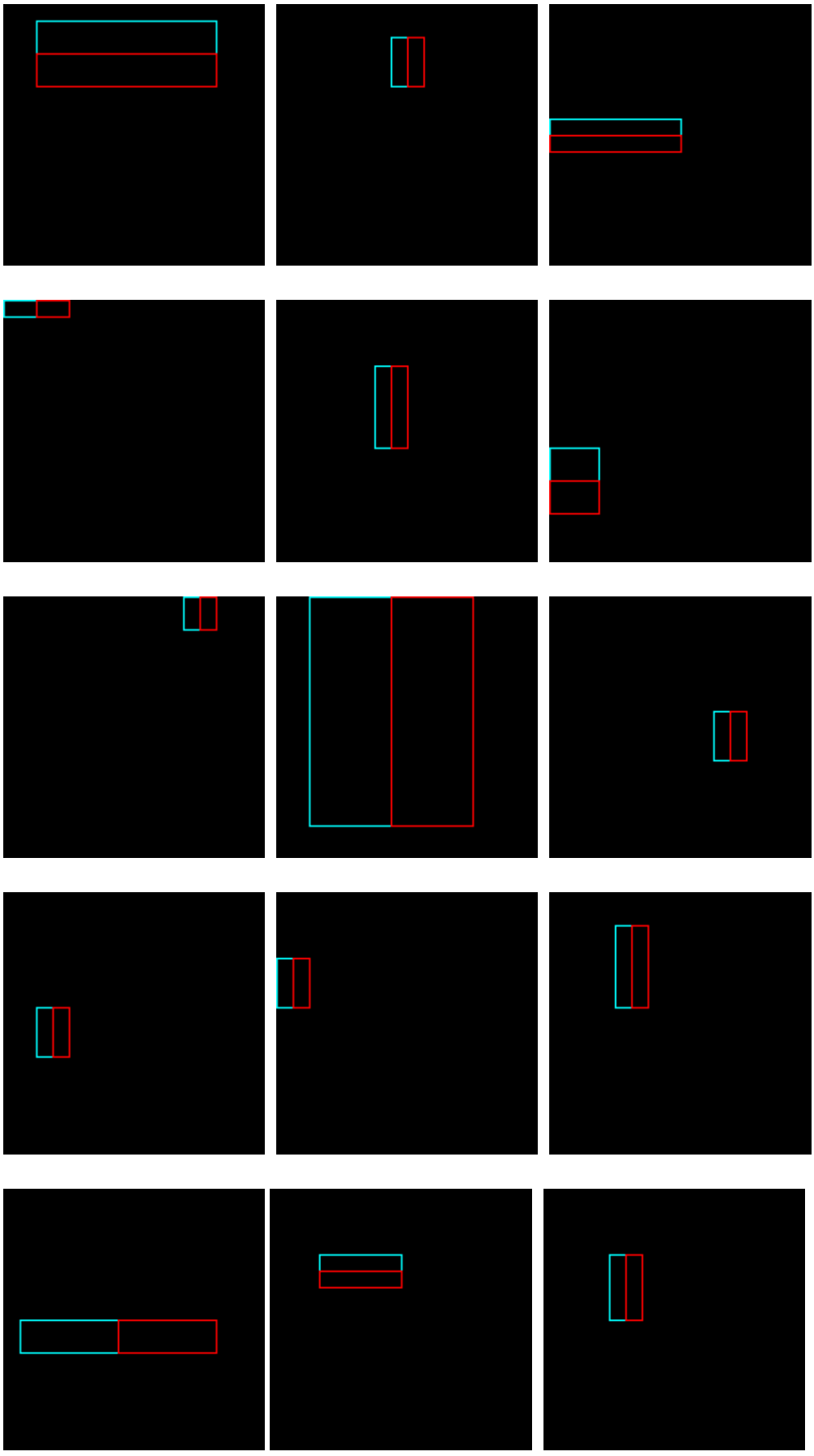
Results:

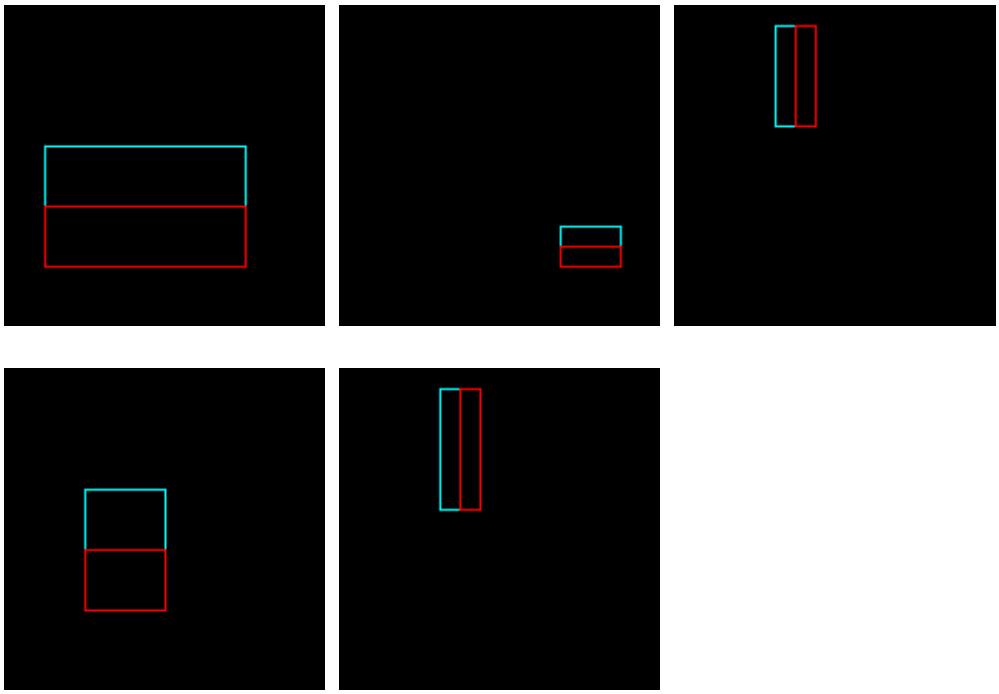
The experiment was performed on two different training data(due to the unzip issue for windows users) and the results from both are reported and compared below.

- The first set of training data "**Training Data 37k**" consisted of 37194 images, 11838 faces, 25356 non-faces.
- The second set of training data "**Training Data 23k**" consisted of 23838 images, 11838 faces, 12000 non-faces

a. The top 20 Haar filters after performing ada boosting for T= 100:
(Blue rectangle indicates positive and red indicates negative)

For Training Data 37k:





The corresponding voting weights α_t :

alpha(voting weight) of top Haar Filter Number 0 = 0.476997

alpha(voting weight) of top Haar Filter Number 1 = 0.388613

alpha(voting weight) of top Haar Filter Number 2 = 0.307811

alpha(voting weight) of top Haar Filter Number 3 = 0.261749

alpha(voting weight) of top Haar Filter Number 4 = 0.244840

alpha(voting weight) of top Haar Filter Number 5 = 0.276738

alpha(voting weight) of top Haar Filter Number 6 = 0.217962

alpha(voting weight) of top Haar Filter Number 7 = 0.194507

alpha(voting weight) of top Haar Filter Number 8 = 0.217410

alpha(voting weight) of top Haar Filter Number 9 = 0.195004

alpha(voting weight) of top Haar Filter Number 10 = 0.170178

alpha(voting weight) of top Haar Filter Number 11 = 0.166878

alpha(voting weight) of top Haar Filter Number 12 = 0.235881

alpha(voting weight) of top Haar Filter Number 13 = 0.165403

alpha(voting weight) of top Haar Filter Number 14 = 0.183394

alpha(voting weight) of top Haar Filter Number 15 = 0.209597

alpha(voting weight) of top Haar Filter Number 16 = 0.157705

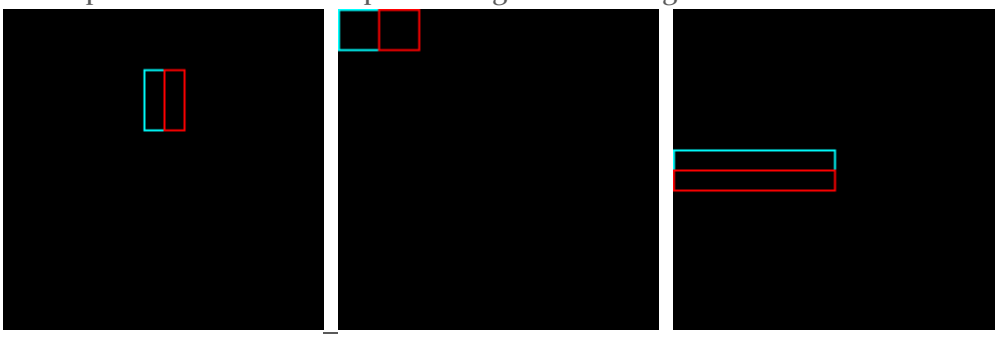
alpha(voting weight) of top Haar Filter Number 17 = 0.168105

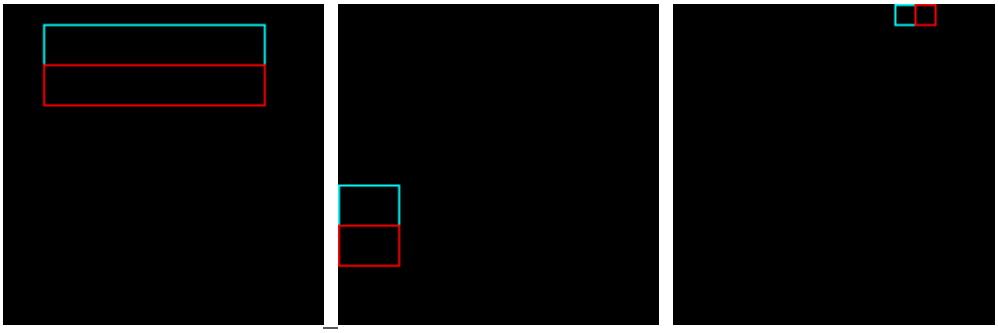
alpha(voting weight) of top Haar Filter Number 18 = 0.215597

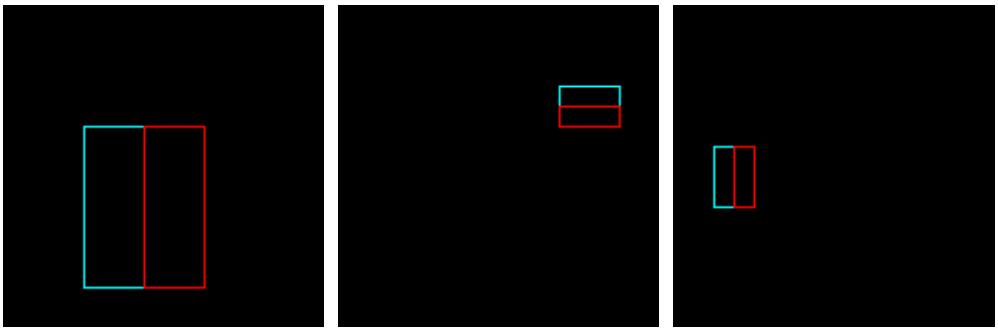
alpha(voting weight) of top Haar Filter Number 19 = 0.134959

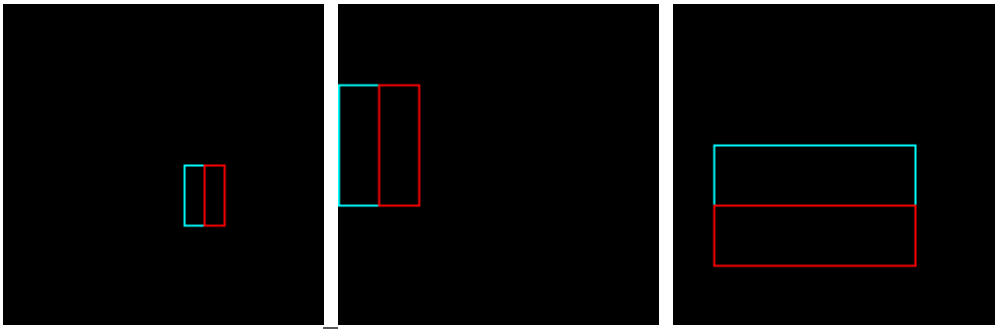
For Training Data 23k:

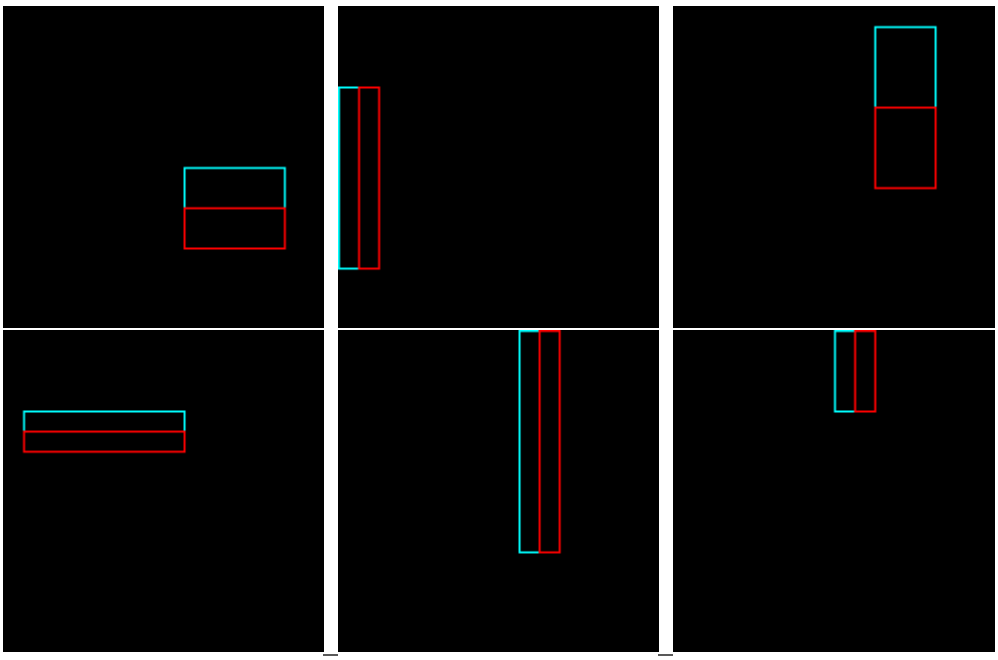
The top 20 Haar filters after performing ada boosting for T= 100:

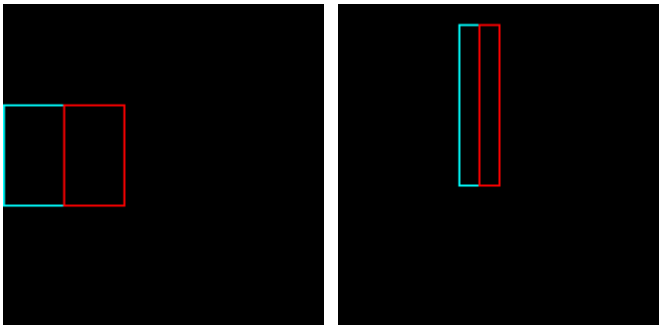












The corresponding voting weights α_t :

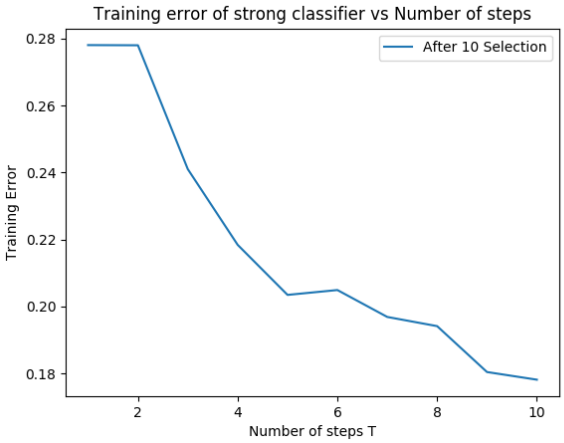
alpha(voting weight) of top Haar Filter Number 0 = 0.443423
alpha(voting weight) of top Haar Filter Number 1 = 0.341916
alpha(voting weight) of top Haar Filter Number 2 = 0.284060
alpha(voting weight) of top Haar Filter Number 3 = 0.286807
alpha(voting weight) of top Haar Filter Number 4 = 0.268495
alpha(voting weight) of top Haar Filter Number 5 = 0.223424
alpha(voting weight) of top Haar Filter Number 6 = 0.199203
alpha(voting weight) of top Haar Filter Number 7 = 0.216084
alpha(voting weight) of top Haar Filter Number 8 = 0.183007
alpha(voting weight) of top Haar Filter Number 9 = 0.180883
alpha(voting weight) of top Haar Filter Number 10 = 0.165899
alpha(voting weight) of top Haar Filter Number 11 = 0.233175
alpha(voting weight) of top Haar Filter Number 12 = 0.175730
alpha(voting weight) of top Haar Filter Number 13 = 0.214646
alpha(voting weight) of top Haar Filter Number 14 = 0.207527
alpha(voting weight) of top Haar Filter Number 15 = 0.176376
alpha(voting weight) of top Haar Filter Number 16 = 0.151788
alpha(voting weight) of top Haar Filter Number 17 = 0.202267
alpha(voting weight) of top Haar Filter Number 18 = 0.151265
alpha(voting weight) of top Haar Filter Number 19 = 0.173517

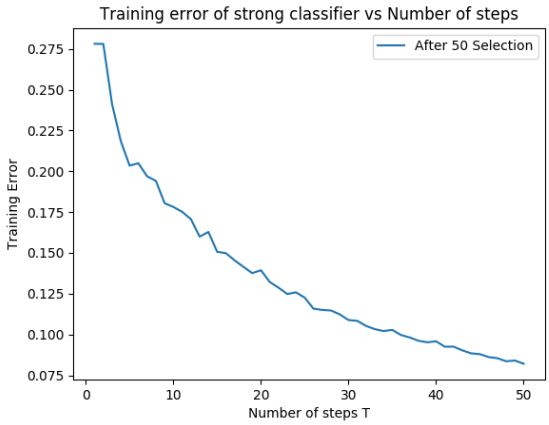
Observations:

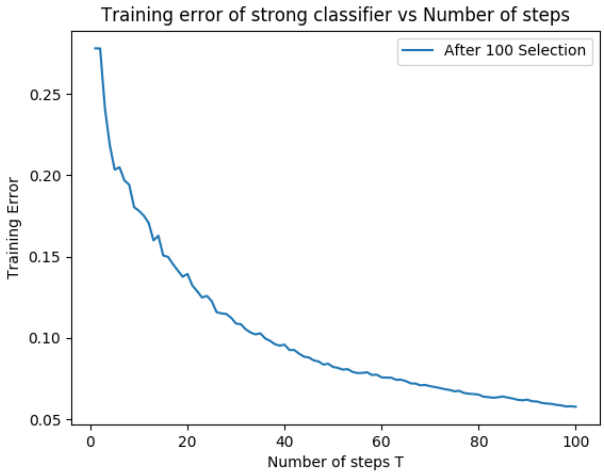
- It can be seen that the value of α_t goes on decreasing as t increases.
- Between Training Data 37k and Training Data 23k, the voting weights of the former are higher than the latter, indicating that the weak classifiers in the former will perform much better than the weak classifiers in the latter model.

b. Plots of the training error of the strong classifier over the number of steps T:

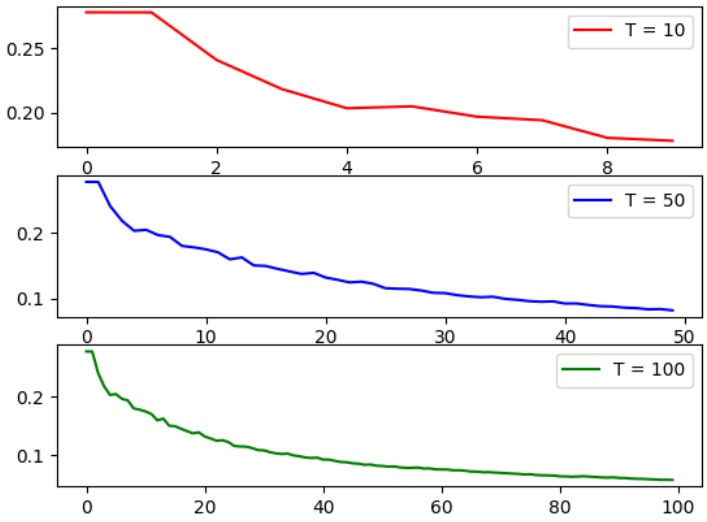
For Training Data 37k:







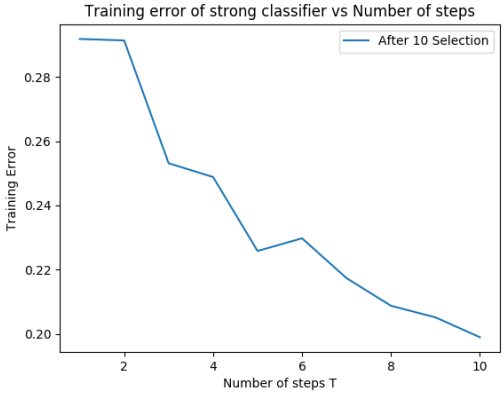
The above three graphs have also been plotted as sub plots in a single graph (as shown below)for the sake of better comparison:

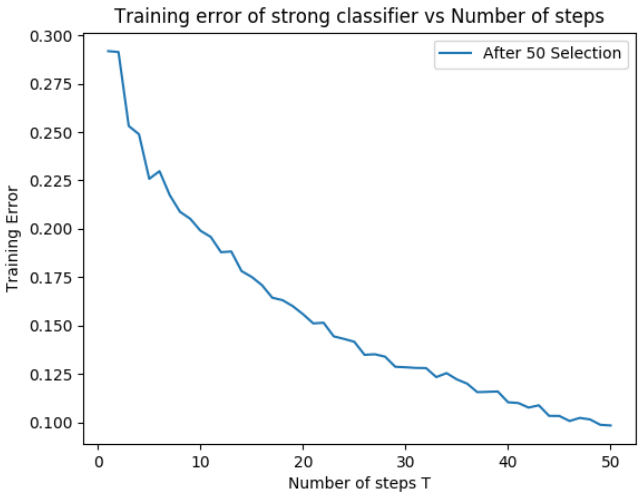


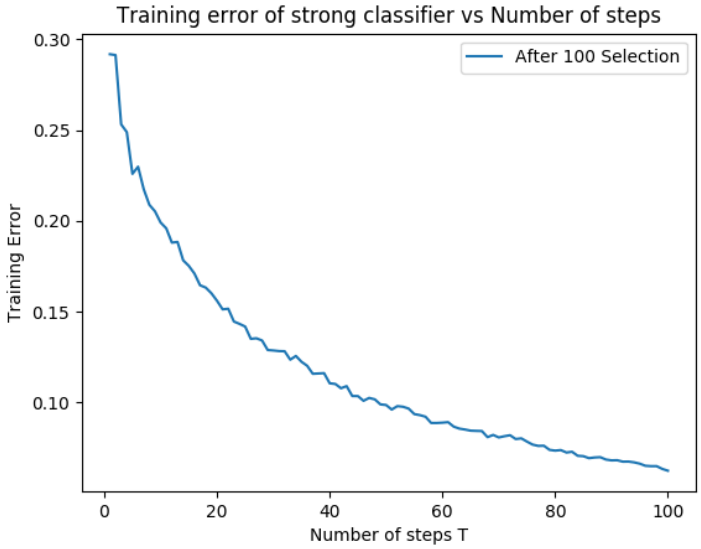
Training accuracy of strong classifier vs number of steps T(for For Training Data 37k):

T	*Training accuracy of strong classifier*
1	0.7219175135774587
10	0.8218529870409206
50	0.9179168683120934
100	0.9423831800828091

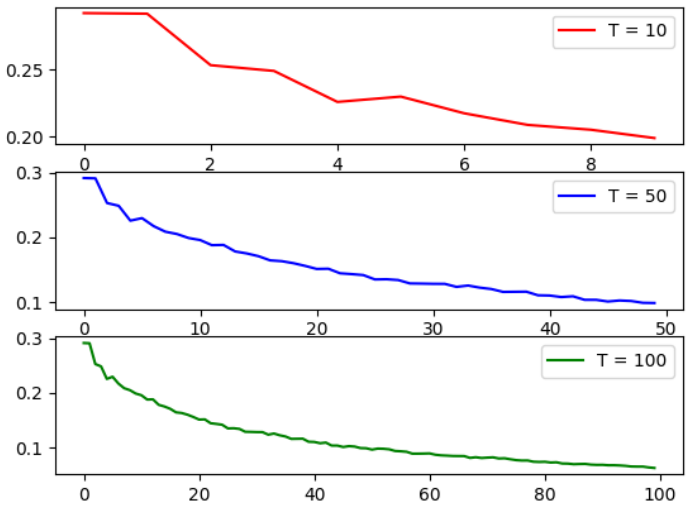
For Training Data 23k:







The above three graphs have also been plotted as sub plots in a single graph (as shown below)for the sake of better comparison:



Training accuarcy of strong classifier vs number of steps T(for For Training Data 23k):

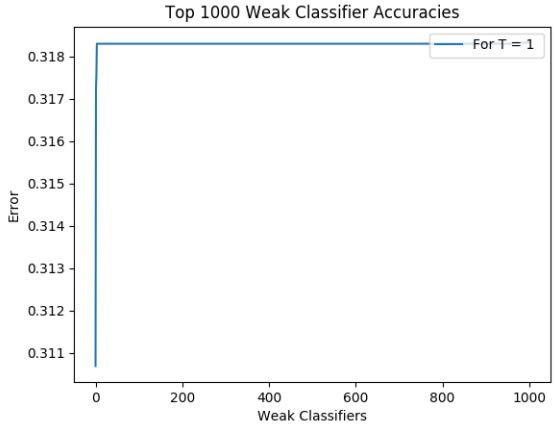
T	*Training accuracy of strong classifier*
1	0.7082389462203205
10	0.8009480661129289
50	0.9014598540145985
100	0.9376625555835221

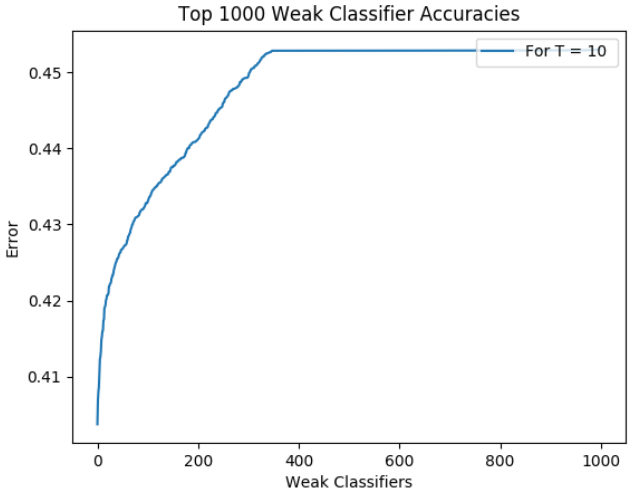
Observations:

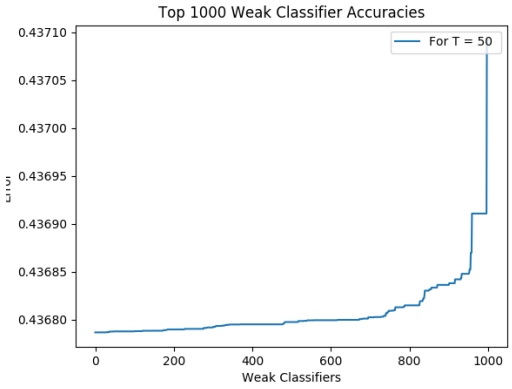
- The training errors of the strong classifier keeps on decreasing as T increases. That is, $TrainingError_{t=10} > TrainingError_{t=50} > TrainingError_{t=100}$
- Between Training Data 37k and Training Data 23k, the training error of the strong classifier at each T is lesser for Training Data 37k. That is ,
  $TrainingError23k_{t=10} > TrainingError37k_{t=10}$
  $TrainingError23k_{t=50} > TrainingError37k_{t=50}$
  $TrainingError23k_{t=100} > TrainingError37k_{t=100}$

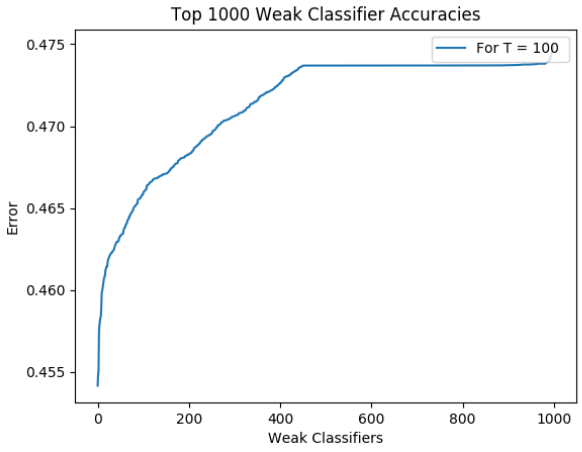
c. Training errors of the top 1000 weak classifiers at T=0,10,50,100

For Training Data 37k:

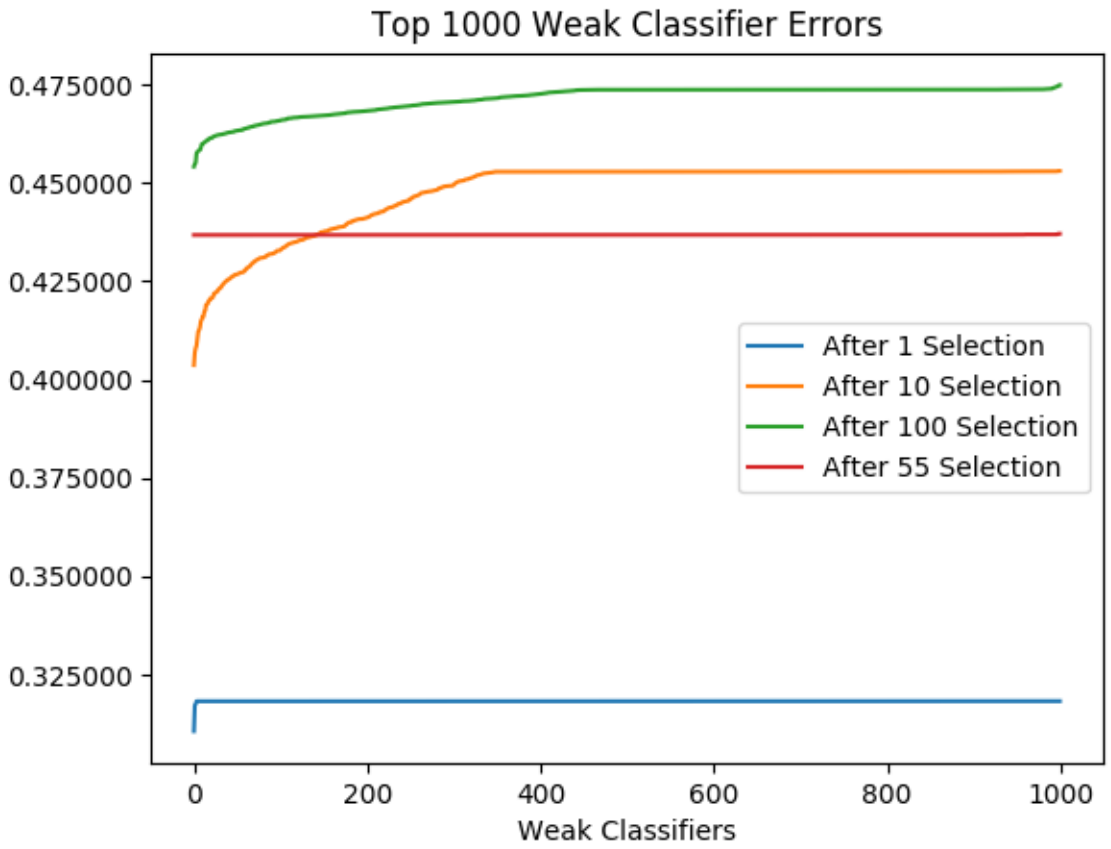




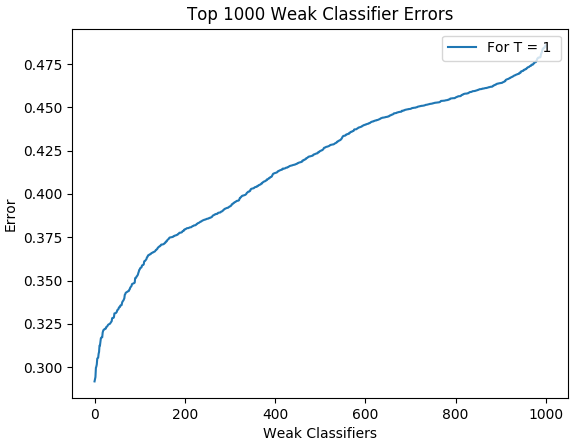


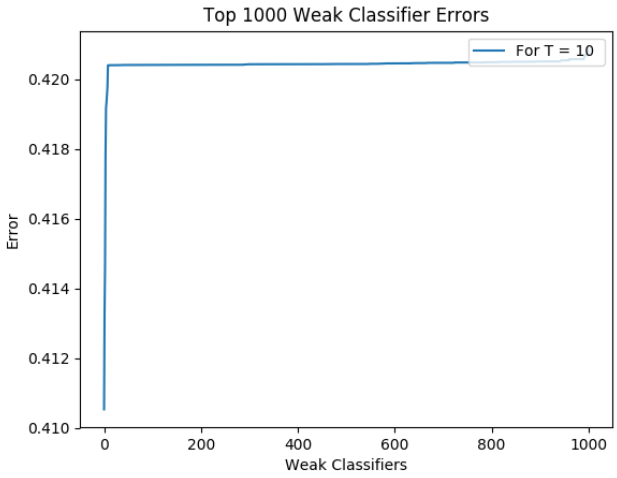


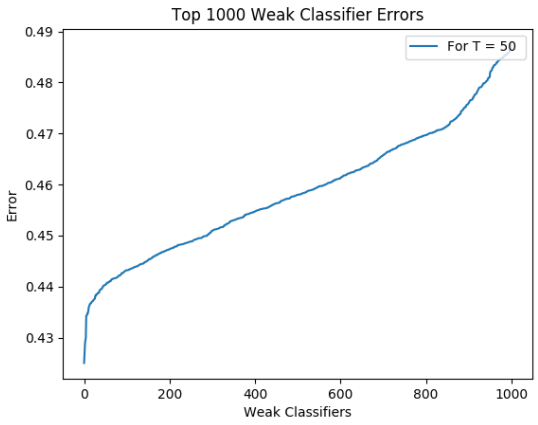
Combined in one graph:

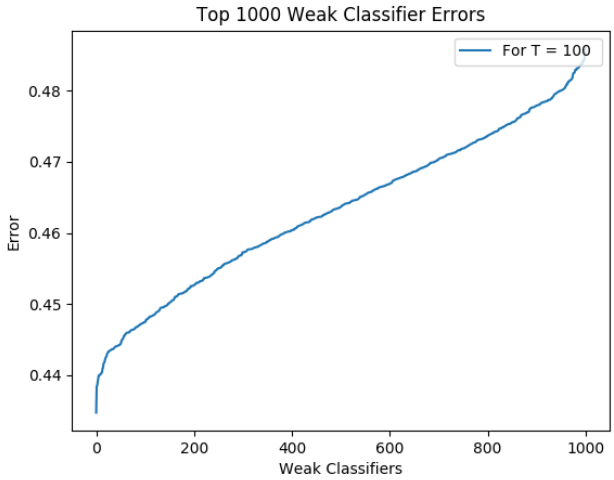


For Training Data 23k:

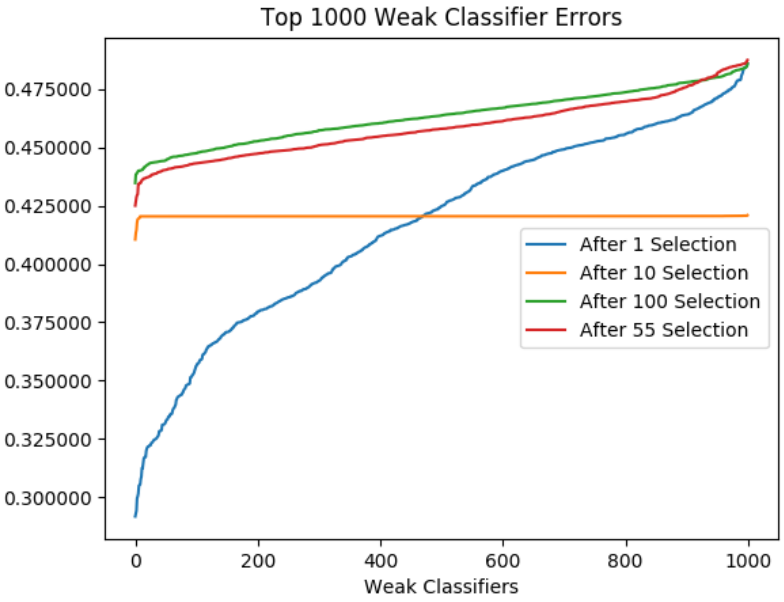








Combined in one graph:

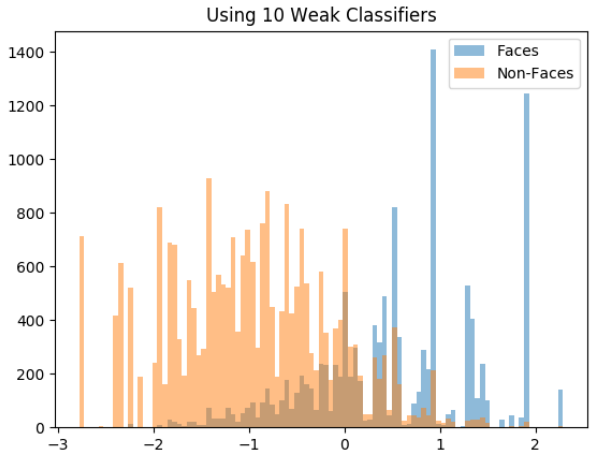


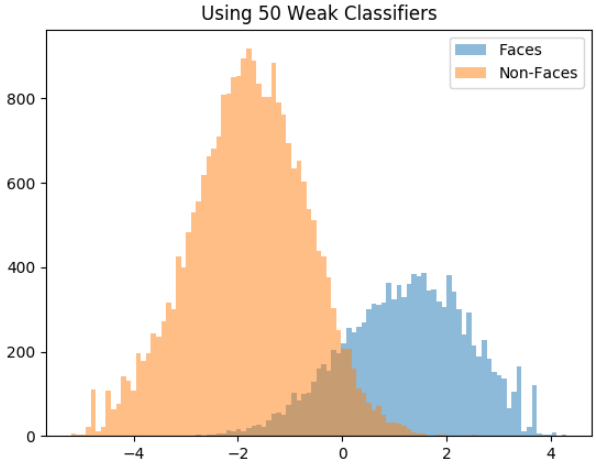
Observations:

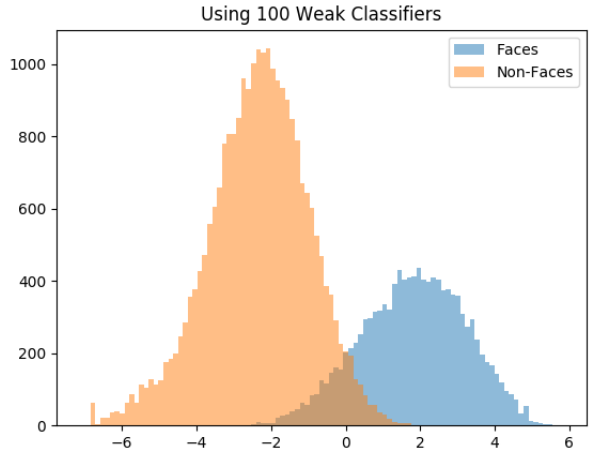
- From the graphs above, as T increases, the weak classifiers get more weaker, that is, their errors keep on increasing.
- Between Training Data 37k and Training Data 23k, the training error of the top 1000 weak classifier at each T increases at a much faster rate for Training Data 23k. This also results in the training error of the weak classifiers plateauing off much sooner than those in Training Data 37k, which will result in overall lesser accuracy of the strong classifier for Training Data 23k.

d. The histograms of the positive and negative populations over F(x), for T = 10; 50; 100.

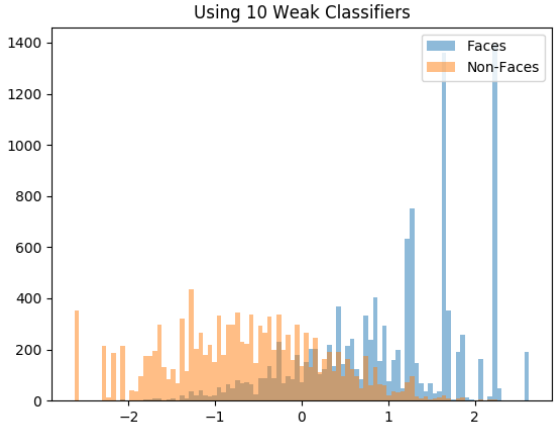
For Training Data 37k:

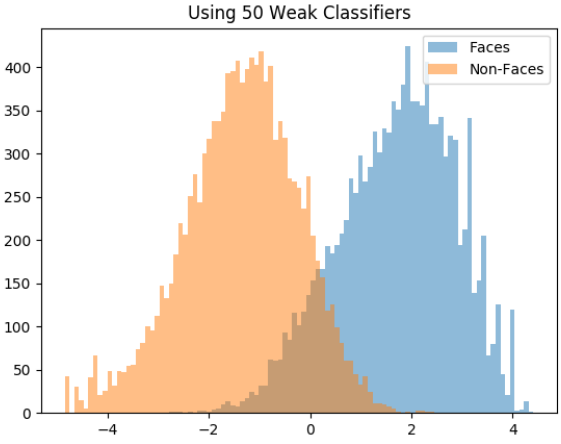


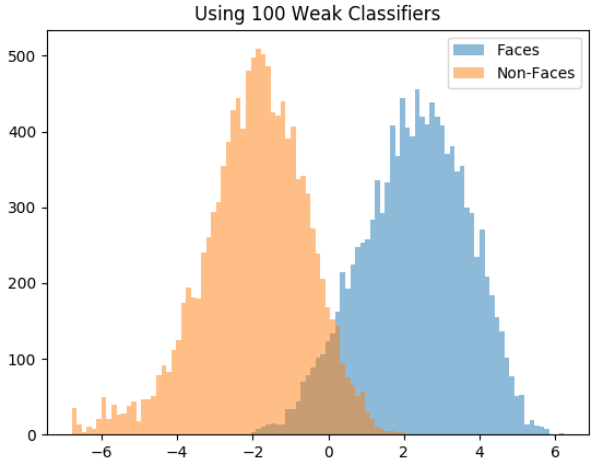




For Training Data 23k:





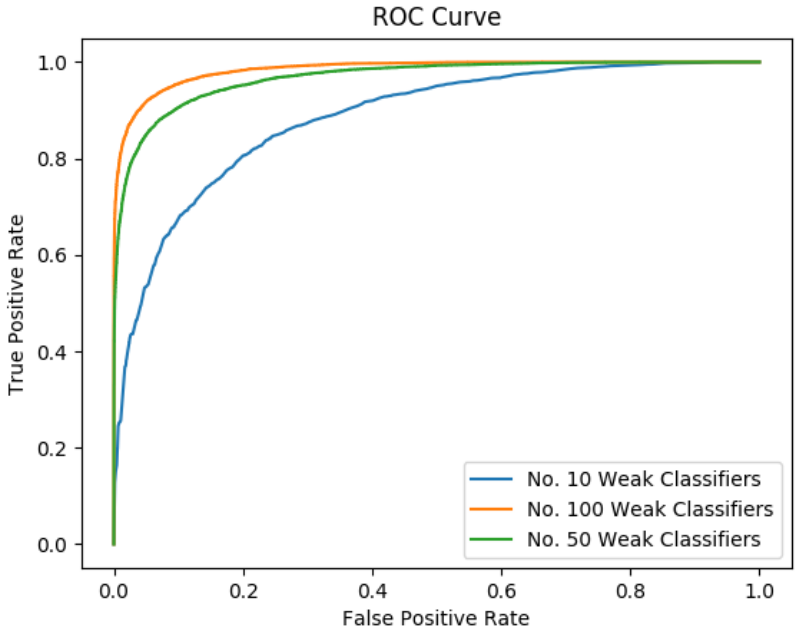


Observations:

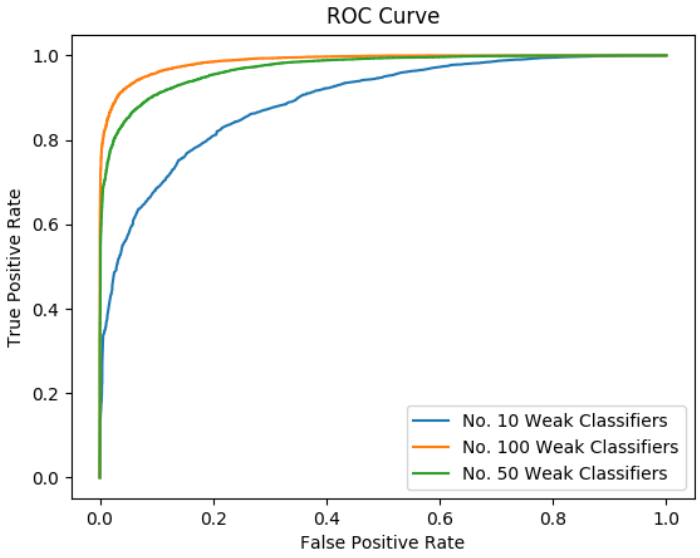
- From the graphs above, as T increases, the overlap between the positive and negative populations in the histogram keeps on decreasing. (The distance between the orange and the negative populations increases).
- Between Training Data 37k and Training Data 23k, at each T, the overlap between the positive and negative populations in the histogram is lesser for Training Data 37k than Training Data 23k, implying that Training Data 37k model is doing much better.

e. ROC curves for T = 10; 50; 100

For Training Data 37k:



For Training Data 23k:



Training accuracy of strong classifier vs number of steps T

T	*Training accuracy of strong classifier for For Training Data 37k:*	*Training accuracy of strong classifier for For Training Data 23k:*
1	0.7219175135774587	0.7082389462203205
10	0.8218529870409206	0.8009480661129289
50	0.9179168683120934	0.9014598540145985
100	0.9423831800828091	0.9376625555835221

Observations:

- The area under the ROC curve, which is the accuracy of the strong classifier, increases at T increases, with T=100 reporting the maximum area.
- From the table above, we can see that Between Training Data 37k and Training Data 23k, at each T, the Training accuracy of the strong classifier is better for Training Data 37k.

f and g. The detected faces in three of the provided test images with and without hard negative mining

For Training Data 37k:

Test image 1 without hard negative mining

Test image 1 with hard negative mining

Test image 2 without hard negative mining

Test image 2 with hard negative mining

Test image 3 without hard negative mining

Test image 3 with hard negative mining

Training accuracy of strong classifier vs number of steps T compared for with and without hard negative mining:

T	*Training accuracy of strong classifier for without hard negative mining:*	*Training accuracy of strong classifier with hard negative mining:*
1	0.7219175135774587	0.7219175135774587
10	0.8218529870409206	0.8138947142012153
50	0.9179168683120934	0.9167607678657848
100	0.9423831800828091	0.94275958487928l6

For Training Data 23k:

The detected faces in three of the provided test images with and without hard negative mining Test image 1 without hard negative mining

Test image 1 without hard negative mining

Test image 2 without hard negative mining

Test image 3 without hard negative mining

Observations:

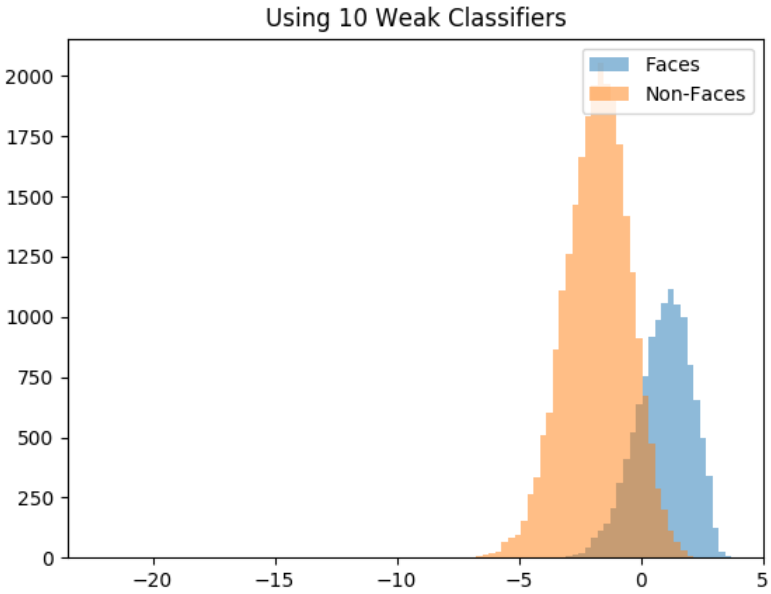
- From the table above, at each T, the training accuracy of the strong classifier is slightly better with hard negative mining than without. This can also be observed in the results of the face detection, where there are fewer false positives after hard negative mining.
- Between Training Data 37k and Training Data 23k, there are more false positive face detections for Training Data 23k.

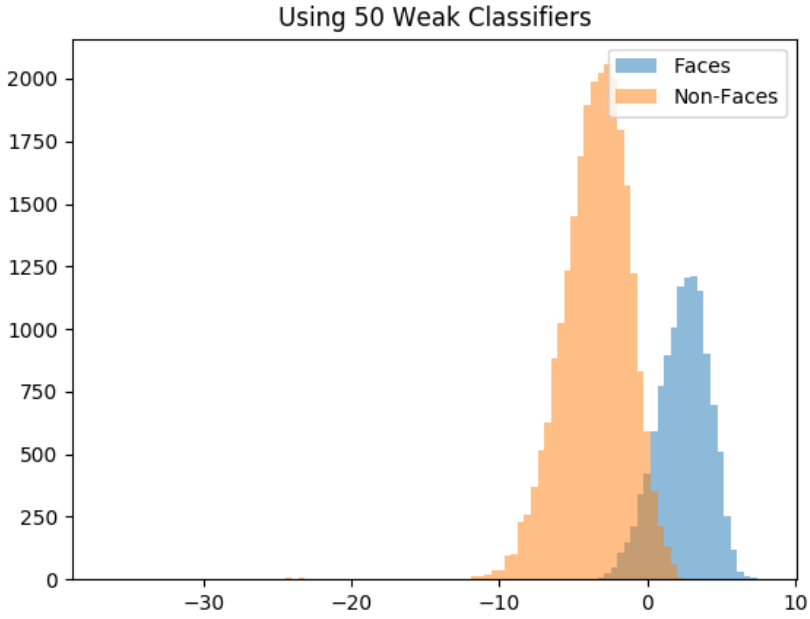
REAL BOOST

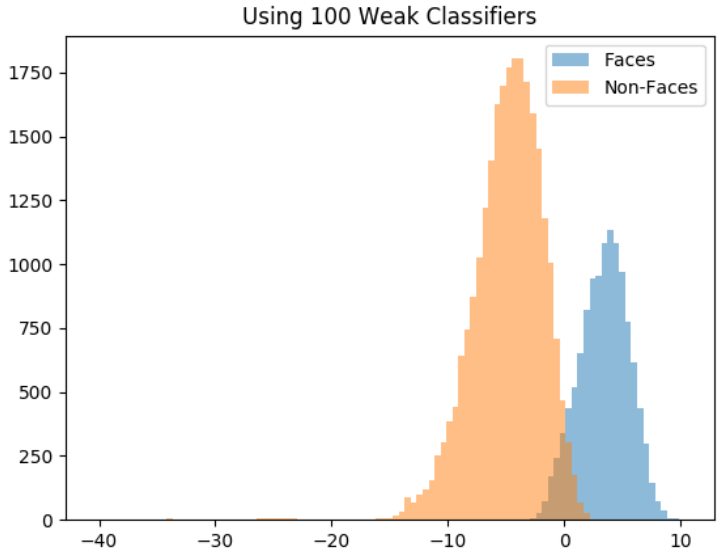
Objective: Implement the RealBoost algorithm

Results: The top T = 10; 50; 100 classifiers got from AdaBoost were chosen to perform real Boosting.

h. The histograms of the positive and negative populations over F(x), for T = 10; 50; 100.



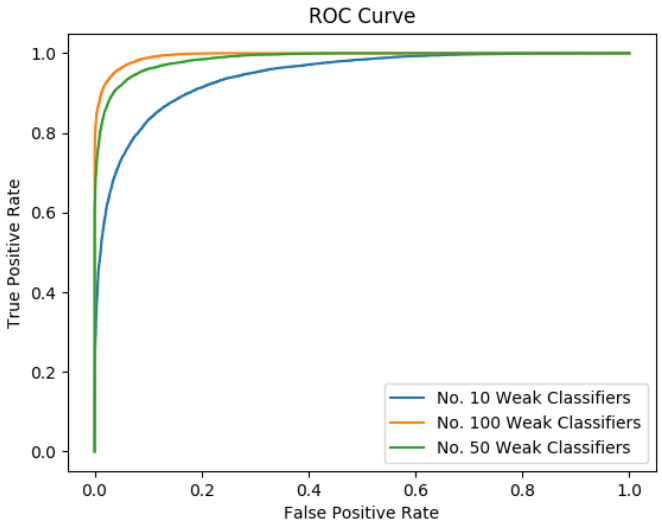
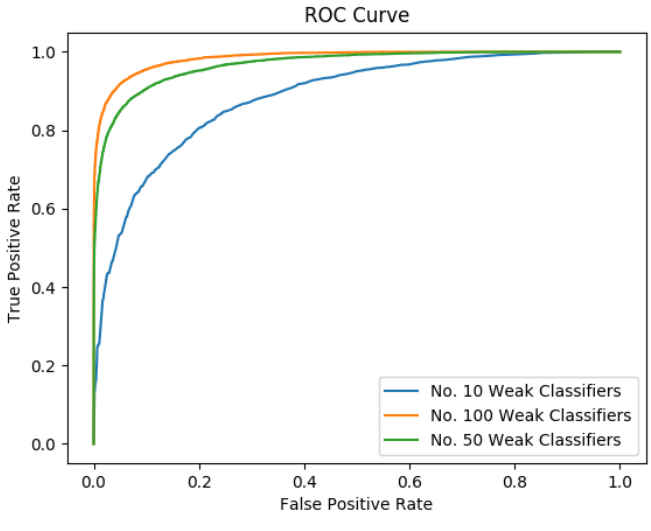




Observations:

- From the graphs above, as T(number of weak classifiers chosen) increases, the overlap between the positive and negative populations in the histogram keeps on decreasing. (The distance between the orange and the negative populations increases).
- Between AdaBoost and RealBoost, at each T, the overlap between the positive and negative populations in the histogram is slightly lesser for RealBoost than AdaBoost, implying that RealBoost model is doing slightly better.

(i) Comparing the ROC curves of RealBoost and AdaBoost:

RealBoost	AdaBoost
	

Observations:

- The area under the ROC curve, which is the accuracy of the strong classifier, at every T, is slightly greater for RealBoost than for AdaBoost.

References

[1] Viola, P., & Jones, M. (2004). Robust Real-Time Face Detection. *International Journal Of Computer Vision*, *57*(2), 137-154. doi: 10.1023/b:visi.0000013087.49260.fb