IN4320 Machine Learning
Computational Learning Theory: boosting

10 March 2019

TU Delft
Coordinated by Prof Marco Loog

Matthijs Bekendam 4725751 J.M.Bekendam@student.tudelft.nl

Contents

1 Question a

In this section the equivalence between the formulation of AdaBoost as it is given in the slides of the course, and the formulation as it is given in the paper will be discussed.
The formulation from the paper works with a binary $\{0,1\}$ classification. The operation of this classification becomes more evident when the formulation is rewritten.

$$\sum_{t=1}^{T}(log(\frac{1}{\beta_t})h_t(x)) \geq \frac{1}{2}\sum_{t=1}^{T}log(\frac{1}{\beta_t})$$
$$= \sum_{t=1}^{T}log(\frac{1}{\beta_t})h_t(x) - \frac{1}{2}log(\frac{1}{\beta_t}) \geq 0$$
$$= \sum_{t=1}^{T}log(\frac{1}{\beta_t^{h_t(x)}}) - log(\frac{1}{\beta_t^{\frac{1}{2}}}) \geq 0$$
$$= \sum_{t=1}^{T}log(\frac{\beta_t^{\frac{1}{2}}}{\beta_t^{h_t(x)}}) \geq 0 \tag{1.1}$$

$h_t(x)$ is either 0 or 1. An $h_t(x)$ of 0 will yield a negative scalar from the logarithm. Hence if there are more 0 hypothesis than 1 hypothesis, the final sum will be negative (assuming equal weights) and classifies a 0. If the summation is positive, the final output will be a 1.

In case $h_t(x) = 1$, a single iteration of the formulation becomes:

$$log(\beta_t^{-\frac{1}{2}}) \tag{1.2}$$

In case $h_t(x) = 0$, a single iteration of the formulation becomes:

$$log(\beta_t^{\frac{1}{2}}) \tag{1.3}$$

The formulation from the slides has a different approach but yields the same result. Instead of the binary $\{0,1\}$ classification it uses a $\{-1,1\}$ classification.

$$F_K(x) = \sum_{k=1}^{K}\alpha_k f_k(x)$$
$$= \sum_{k=1}^{K}\frac{1}{2}log(\frac{\sum_i w_i}{\epsilon_K} - 1)f_k(x)$$
$$= \sum_{k=1}^{K}\frac{1}{2}log(\frac{1-\epsilon_K}{\epsilon_K})f_k(x)$$
$$= \sum_{k=1}^{K}\frac{1}{2}log(\frac{1}{\beta_K})f_k(x) \tag{1.4}$$

If we set $f_k(x)$ to 1 as done for equation 1.2 the equivalence between the two formulations become evident.

$$\frac{1}{2}log(\frac{1}{\beta_K})$$
$$= log(\beta_K^{-\frac{1}{2}}) \tag{1.5}$$

In case $f_k(x)$ = -1, a single iteration of the formulation becomes:

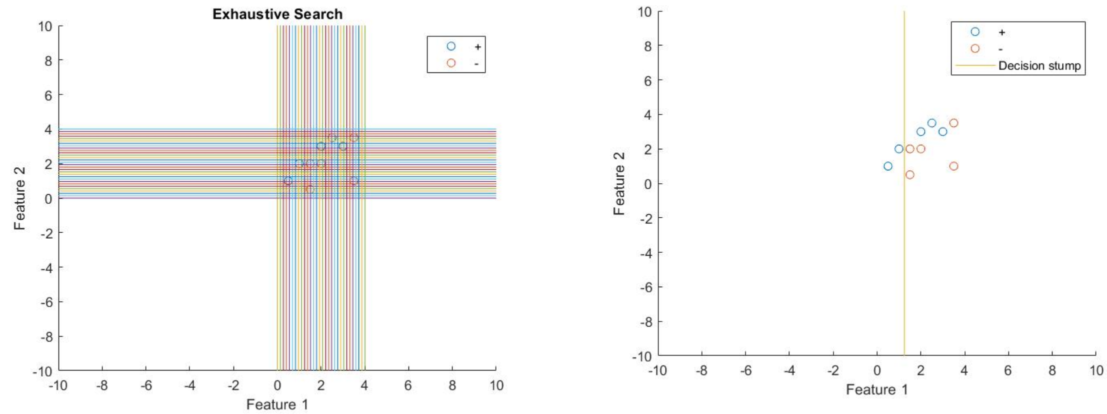
$$-log(\beta_K^{-\frac{1}{2}})$$

$$= log(\beta_K^{\frac{1}{2}}) \tag{1.6}$$

If equation 1.4 sums up to a positive number the final hypothesis will output a 1, if the summation is negative, the output will be a -1.

Equations 1.2 and equation 1.5 are equivalent. Equations 1.3 and equation 1.6 are equivalent. Hence equation 1.1 and 1.4 are equivalent.

2 Question b



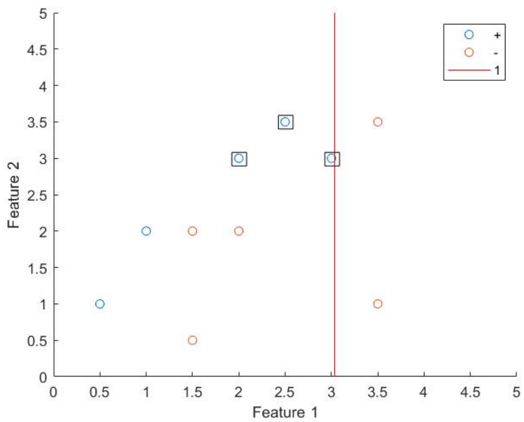
(a) Set of possible thresholds/stumps for a small generic data set.

(b) Stump candidate (among others!) for optimal threshold/feature which minimizes the classification error

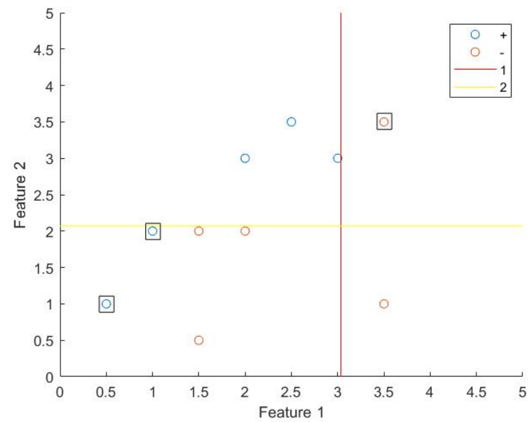
Figure 1: Exhaustive search created multiple stumps along a feature and searches which one minimizes the classification error.

With the addition of a weight vector, the WeakLearn algorithm gives misclassified points a higher weight than correct classified points. Hereby highlighting the incorrect points such that in a next iteration, these points have a higher chance of being correctly classified. For each iteration, the weights are updated accordingly.

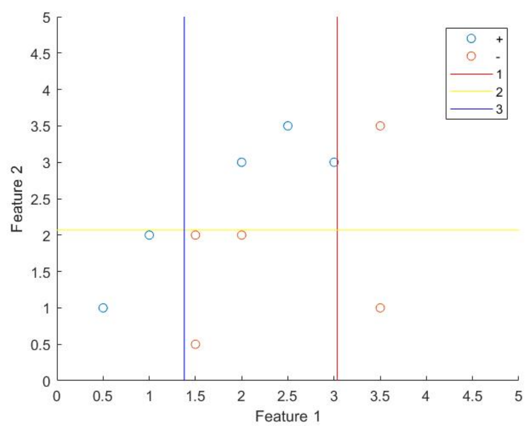
The effect of the weights on the optimal- threshold and feature, after three iterations of the WeakLearn algorithm, is shown in Figure 2.



(a) Iteration one



(b) Iteration two



(c) Iteration three

Figure 2: Three consecutive iterations of the WeakLearn algorithm. The squares around the data points indicate an incorrect classification and are given a higher weight for the next iteration. As shown in the figures, the incorrect classifications are corrected for in the next iteration.

2.1 Code WeakLearn

The WeakLearn algorithm outputs an optimal- feature, threshold, scenario (which side of stump is which class), error and a 'win vector' which is the trained vector belonging to the smallest error/feature/scenario. This vector is used to update the weights.

```
function [optimal_feature,treshold_opt_feature,win_vec,scenario_opt_feature,low_error_feat] =
    WeakLearn(c1,c2,wi,amount_stumps,list_feat,list_thres)
     [L1,B1] = size(c1); %class one
     [L2,B2] = size(c2); %class two

     %Define variables
     min_error_features = zeros(1,L1);
     thresholds_features = zeros(1,L1);
     scenarios_features =zeros(1,L1);
     train_1_all = zeros(amount_stumps,B1+B2,L1);
     train_2_all = zeros(amount_stumps,B1+B2,L1);
     y = [zeros(1,B1) ones(1,B2)];
     stumps_all = zeros(L1,amount_stumps);


     %Create p-matrix with input weights.
     p = wi/sum(wi);

 %% DETERMINING OPTIMAL FEATURE, THRESHOLD

for i_f = 1:L1 %Loop over all features

     c = [ c1(i_f,:) c2(i_f,:)] ;

     min_iter = floor(min(c(1,:)));
     max_iter = ceil(max(c(1,:)));
     stumps = linspace(min_iter,max_iter,amount_stumps);

     train_1 = zeros(amount_stumps,B1+B2);
     train_2 = zeros(amount_stumps,B1+B2);


     for i_s = 1 : length(stumps) %FOR-loop over all stumps

         for i_d = 1 : length(c) %FOR-loop over all data
             %Two scenarios are defined such that the optimal side of the
             %stump is utilized. (I.E the question is awnsered: Points left
             %of stump are +-class or right to the right +-class?)

            %scenario 1
             if c(1,i_d) >= stumps(i_s)
                 train_1(i_s,i_d) = 1 ;
             else
                 train_1(i_s,i_d) = 0;
             end
            %scenario 2

            if c(1,i_d) >= stumps(i_s)
                 train_2(i_s,i_d) = 0 ;
             else
                 train_2(i_s,i_d) = 1;
            end

```

```
        end
      train_1_all(i_s,:,i_f) = train_1(i_s,:);
      train_2_all(i_s,:,i_f) = train_2(i_s,:);
    end

    % the function minimum_er outputs the minimum error from current feature/stumps/scenario with
        threshold.
    [min_error,scenario,threshold,xi] = minimum_er(train_1, train_2, p,
        amount_stumps,stumps,B1,B2);
    %This is done for each feature and saved in the vectors below from
    %which the optimal feature with parameters is chosen.
     stumps_all(i_f,1)=xi;
     min_error_features(1,i_f) = min_error;
     thresholds_features(1,i_f) = threshold;
     scenarios_features(1,i_f) = scenario;


end
%Feature-loop ended; start exhaustive search


%Seach lowest error from all features.
low_error_feat = min(min_error_features,[],'all'); %search lowest value
[c_e,optimal_feature]=find(low_error_feat == min_error_features); %search index lowest value

%In case two features have equal error, the function preference_feat
%prioritizes those features which have not been used alot for diversity.
if length(optimal_feature) > 1
    [optimal_feature] = preference_feat(optimal_feature,list_feat) ;
end

scenario_opt_feature = scenarios_features(1,optimal_feature);
treshold_opt_feature = thresholds_features(1,optimal_feature);

%Receive 'winning' vector from all trained vectors which belongs to the
%optimal feature/stump, which is given as an output for the error/beta calculation.
     if scenario_opt_feature == 1
       win_vec = train_1_all(stumps_all(optimal_feature,1),:,optimal_feature);
    else
       win_vec = train_2_all(stumps_all(optimal_feature,1),:,optimal_feature);

    end

end
```

2.1.1 Code minimum er

This function is used inside the WeakLearn algorithm to calculated the minimum error for both scenarios from a single feature.

```
function [min_error,scenario,threshold,xi,errors] = minimum_er(train_1, train_2, p,
    amount_stumps,stumps,N1,N2)
errors = zeros(amount_stumps,2);
y = [zeros(1,N1) ones(1,N2)];

for i_s = 1:amount_stumps

  errors(i_s,1) = sum(p.*abs(train_1(i_s,:)-y));
```

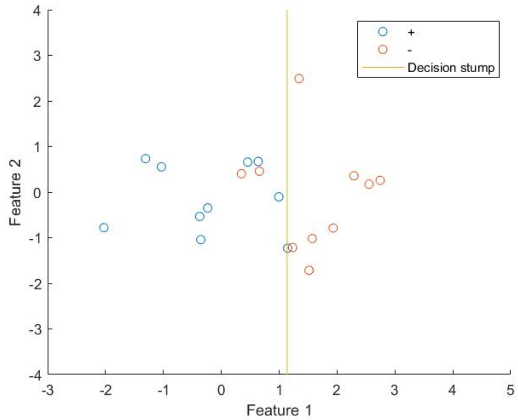
```
  errors(i_s,2) = sum(p.*abs(train_2(i_s,:)-y));

end

 min_error = min(errors,[],'all'); %search lowest value
 [xi,yi]=find(errors == min_error); %search index lowest value


 len_xiyi = length(xi);
 %ran_ind = randsample(len_xiyi,1)
 ran_ind = randi([1 len_xiyi],1,1);
 scenario = yi(ran_ind,1);
 threshold = stumps(xi(ran_ind,1));
 xi = xi(ran_ind,1);

end
```

2.1.2 Code preference feat

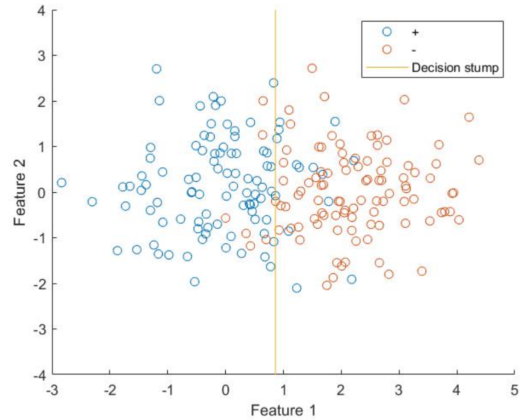
In the case that multiple features have the lowest error, this function prioritizes those which have been used the least amount of times, for diversity.

```
function [feature] = preference_feat(optimal_feature,list_feat)
xa = zeros(length(optimal_feature),1);
xb = zeros(length(optimal_feature),1);
max_feat = zeros(length(optimal_feature),2);

    for i_l = 1:length(optimal_feature);
        [xa,xb] = find(optimal_feature(1,i_l) == list_feat);
         max_feat(i_l,:) = [length(xb),optimal_feature(1,i_l)];
    end


min_max_feat = min(max_feat(:,1));
[x,y] = find(max_feat(:,1) == min(max_feat(:,1)));
feature = max_feat(x,y+1);
if length(feature) > 1
    feature = feature(1,1);
end
end
```

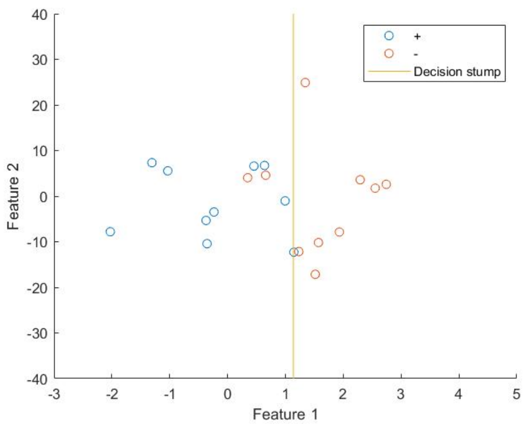
3 Question c



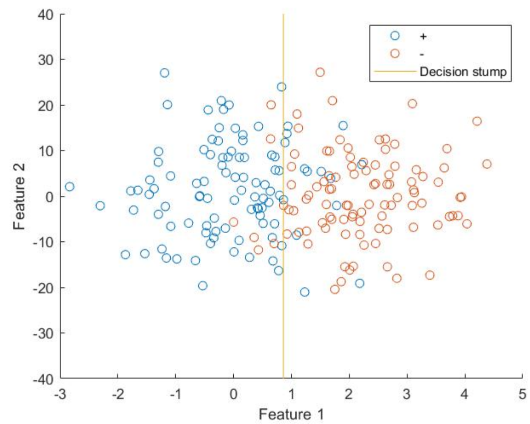
(a) 20 points, optimal threshold = 1.137, optimal feature = 1.



(b) 200 points, optimal feature = 1., optimal threshold = 0.8620



(c) 20 points, optimal feature = 1, optimal threshold = 1.137, feature 2 scaled by a factor 10.



(d) 200 points, optimal feature = 1, optimal threshold = 0.8620, feature 2 scaled by a factor 10.

Figure 3: (a) and (b) show the non-scaled Gaussian distributions. (c) and (d) show the scaled Gaussian distributions with feature 2 multiplied by a factor of 10.

Figure 3 shows that in all four cases feature 1 is the optimal feature to use, which makes sense since the mean of the classes are unequal. Objects from different classes which lie further away from one another on average, are easier to classify. Resulting in a lower error. Scaling feature 2 by a factor of 10 will not yield a better classification error compared to feature 1, since the mean of the classes remain equal. Changing feature 2 will not change the optimal threshold coming from feature 1.

It also makes sense that the optimal threshold is in the middle between both classes, since this is where the classification error would be lowest when no weights are given to the objects.

4 Question d

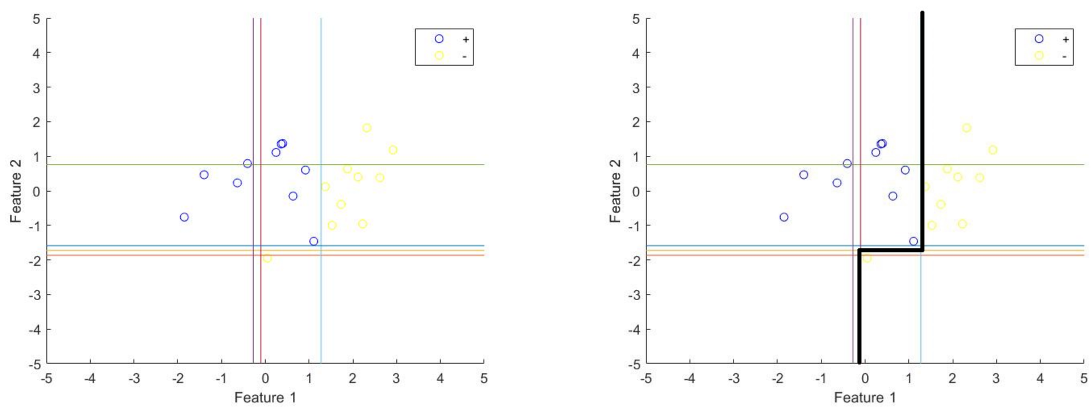
Training the Fashion NIST data set with a decision stump from the WeakLearn algorithm resulted in an apparent error of 20 % and a true error of 34.7500 %. For the training, a single iteration was used and an exhaustive search using 80 stumps. As expected, the apparent error is lower than the true error.

5 Question e

AdaBoost combines a set ($t = (1, 2...., T)$) of weak hypothesis $h_t(x)$ to create one strong, accurate hypothesis $h_f(x)$. The error of e_t of each h_t gives an indication on how much 'right' each h_t has on the voting process when a data point is evaluated. A strong hypothesis will yield (when not over-trained) stronger results than a singular weak hypothesis.

To test the effectiveness of the AdaBoost, the Fashion NIST training set was used again to receive now multiple weak hypothesis (T=100) and the Fashion NIST test set was used to evaluate the hypothesis on. Running AdaBoost on this set gave a apparent error of 0% and a true error of 20.50 %. This is a larger improvement compared to the singular hypothesis from Question 'd'.

Another test is done on the a Gaussian distributed set similar to the one from question c. The decision stumps and classification boundary are illustrated in Figure 4.



(a) Optimal stumps from 100 iterations. (b) Optimal stumps with classification boundary.

Figure 4: (a) and (b) show the effectiveness of the AdaBoost algorithm.

5.1 Code AdaBoost

```
function [output_vector] = AdaBoost(c1_train,c2_train,c1_test,c2_test,iterations,amount_stumps)
%Declaring variables:
[L1,B1] = size(c1_train);
[L2,B2] = size(c2_train);

wi = ones(1,B1+B2)/(B1+B2);
opt_features = zeros(iterations,1);
opt_tresholds = zeros(iterations,1);
beta_all = zeros(iterations,1);
scenarios_all=zeros(iterations,1);
y = [zeros(1,B1) ones(1,B2)];

```

```
% For-loop T-iterations training classifier with WeakLearn:
for T = 1:iterations

 %WeakLearn algorithm:
[feature,treshold,win_vec,scenario,et] =
    WeakLearn(c1_train,c2_train,wi,amount_stumps,opt_features,opt_tresholds);

%Saving output WeakLearn (Optimal- feature, threshold scenario).
opt_features(T,1) = feature;
opt_tresholds(T,1) = treshold;
scenarios_all(T,1) = scenario;

%Calculating Beta and recalculating weights
    beta = et/(1-et);

if et~=0
    wi = wi.*beta.^(1-abs(win_vec-y));
end
%Saving all beta's
beta_all(T,1)=beta;

end

%Test data:
c = [c1_test c2_test] ;
[L3,B3] = size(c) ;
[L1t,B1t] = size(c1_test);
[L2t,B2t] = size(c2_test);

%Declaring train vector
train = zeros(length(opt_features),B3);


for i_of = 1:length(opt_features) %amount of hypothesis to compare

    cf = opt_features(i_of); %current feature/hypothesis

   for i_d = 1 : B3 %Loop over data corresponding to an optimal feature

    %Determine what scenario the feature belongs to:

    if scenarios_all(i_of,1)== 1 %scenario 1


        if c(cf,i_d) >= opt_tresholds(i_of)


                train(i_of,i_d) = 1 ;
        else
                train(i_of,i_d) = 0;
        end

    else %scenario 2
        if c(cf,i_d) >= opt_tresholds(i_of)
                train(i_of,i_d) = 0 ;
        else
                train(i_of,i_d) = 1;
        end

```

```
    end

  end

end

%% OUTPUT HYPOTHOSIS PART TWO (AdaBoost)
%Declaring final output vector
output_vector=zeros(1,B3);

for i_d = 1:(B3) %loop for data amount
    ht_sum = 0;
    condition = 0;
    for i_h = 1 : length(opt_features) %loop for voting
        %bottom inequality equation page 12 (Freund and Shapire)
        ht_sum = ht_sum+log(1/beta_all(i_h))*train(i_h,i_d); %
        condition = condition+log(1/beta_all(i_h));

    end
    condition = 0.5*condition;

    if ht_sum >= condition
        output_vector(1,i_d) = 1;
    else
        output_vector(1,i_d) = 0;
    end



end


end
```

6 Question f

Testing the AdaBoost on the Gaussian distributed set (2000 points), resulted in a true error of 16.15 % and an apparent error of 15.55 %. Difficult objects are those with a larger chance of being miss classified, in a Gaussian distributed data set of two classes, these are the points which have a relatively smaller distance to the other class mean. This is illustrated in Figure 5.

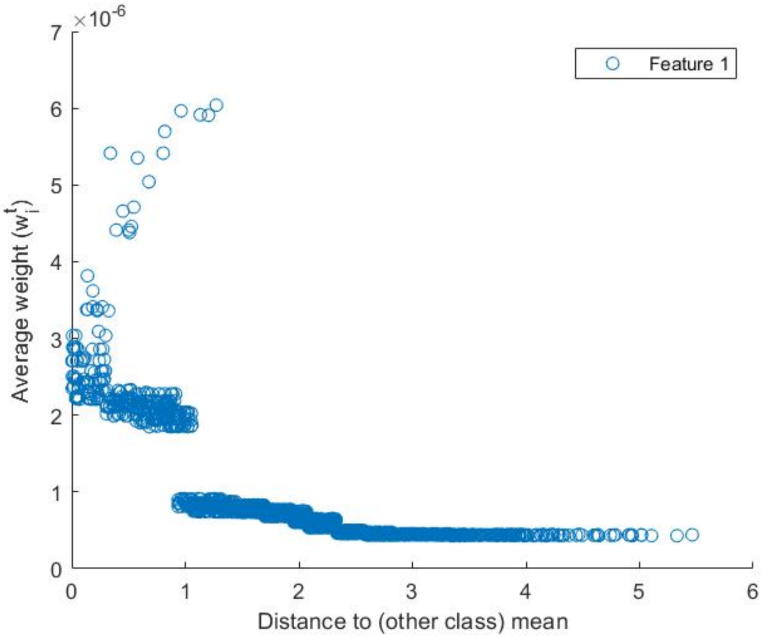


Figure 5: Average weights vs. distance to other class mean after 100 iterations using feature 1 (μ_1 = 0, μ_2 = 2). 2000 objects used (1000 each class).

As shown in Figure 5 objects with a smaller distance to the other class mean (i.e difficult objects), receive on average a higher weight.

7 Question g

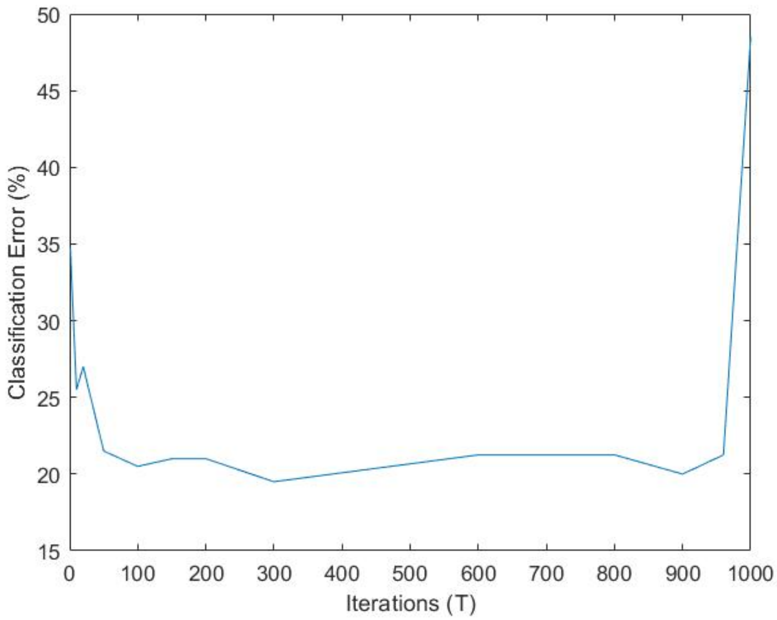


Figure 6: Classification error vs. Iterations.

Figure 6 shows that too few or too many iterations both result in an increase of classification error. Using too few iterations will cause an under fit; the classifier has too few stumps to be accurate. Using to many iterations (stumps) will cause a classic overfit; overgeneralizing the model.

Using an optimal T of 300 iterations, the weights were averaged for each object and plotted in Figure 7.

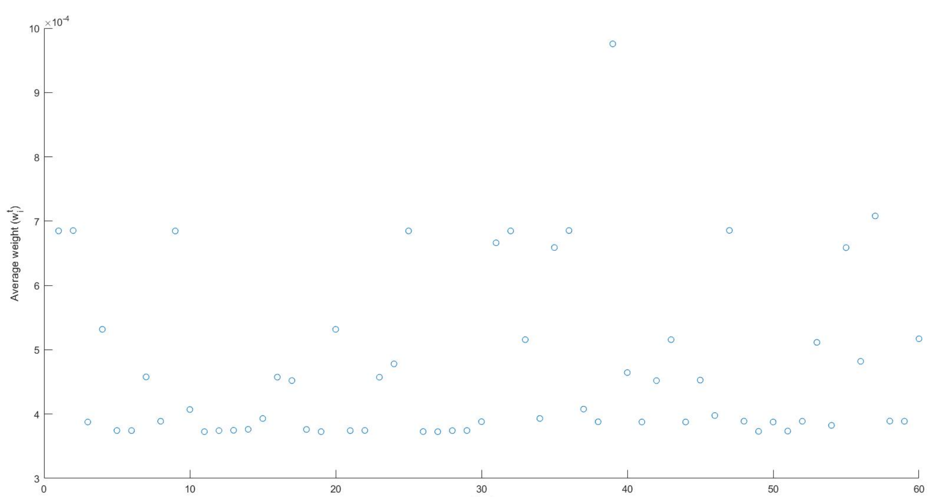


Figure 7: Average weights vs. objects. The objects with a larger average weight are the 'difficult' objects.

8 Question h

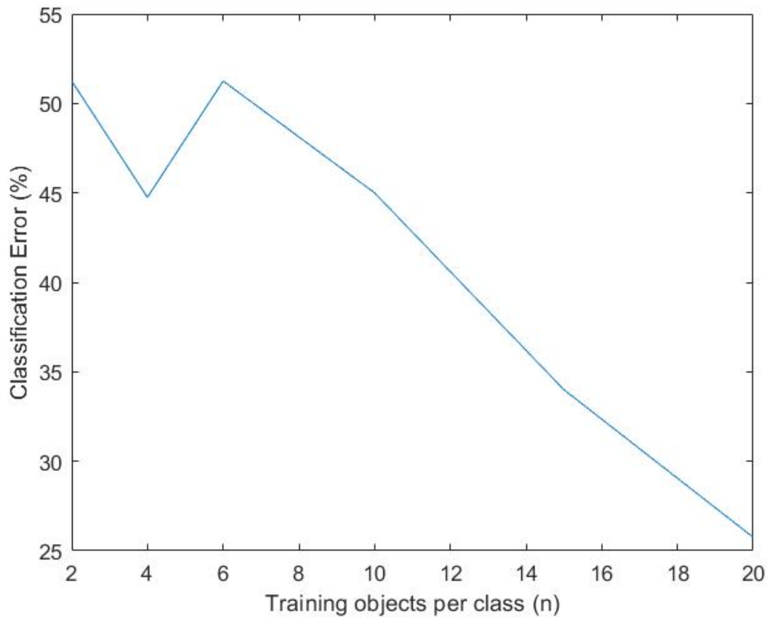


Figure 8: Amount of training objects per class (n) vs. Classification error. As the number of objects increase, the error decreases.

Figure 8 clearly shows the dependence of the AdaBoost algorithm on at least 20+ objects per class when the objects have large amount of features. The error is large when only a few objects are available since most features will not have multiple stumps. When this happens the accuracy per optimal feature decreases and hence the error increases.