IN4320 Machine Learning — Assignment 2

Francisca Mestre - 4930142

March 2019

a.

The formulation used in the slides of the class computes the error of each weak classifier in a slightly different way it is computed in the formulation on the paper. The difference is in the fact that the paper uses normalized sample weights as opposed to the slides. This will later be compensated by using the summation of all sample weights divided by the error (sum of weights of misclassified samples) when computing the classifier weight in the slides.
The formulation in the slides computes directly the weak classifier weights and uses them later on to update the sample weights. In the paper the sample weights are updated in a slightly different way using a function of the classifier error. These are, however, equivalent as the formulation in the paper can be approximated by the exponential function used in the slides for large T.
Finally, in the slides the final hypothesis is computed by directly applying a weighted average of each classifier's hypothesis while in the paper the classifier weight is used in an `if` statement to assign the final labels.
Despite these differences, the two formulations are equivalent as the differences that occur are eventually "compensated" by another operation.

b.

The code used for the implementation of the weak learner is the following:

```
function [f_optimal, theta_optimal, min_error, side] =
   training_decision_tree(dataset, examples_plus,
   examples_minus, w_plus, w_minus)
% This function is used for training the decision stumps
% INPUTS:   dataset - all examples
%           examples_plus - examples from class 1 of the
   dataset
%           examples_minus - examples from class 2 of the
   dataset
%           w_plus - weights of the examples from class 1
%           w_minus - weights of the examples from class 2
% OUTPUTS:  f_optimal - optimal feature
```

```
%           theta_optimal - optimal threshold
%           min_error - training error
%           side - most accurate partition of the classes

N_features = size(dataset, 2); N_examples = size(dataset, 1);
theta = zeros(N_examples, N_features); error = zeros(N_examples
   , N_features);
side_m = zeros(N_examples, N_features);
aux = zeros(N_examples, N_features); aux_i = zeros(N_examples,
   N_features);

for f = 1:N_features
    [aux(:,f), aux_i(:,f)] = sort(dataset(:,f));
    for ex = 1:N_examples
        if ex == 1
            theta(ex, f) = -10 + (aux(ex, f)+10)*rand;
        elseif ex ~= 1
            theta(ex, f) = aux(ex-1, f) + (aux(ex, f)-aux(ex-1,
                f))*rand;
        end
        R_plus = []; L_plus = []; R_minus = []; L_minus = [];
        rp = 0; rm = 0; lp = 0; lm = 0; wR_plus = []; wL_plus =
            []; wR_minus = []; wL_minus = [];
        for p = 1:size(examples_plus, 1)
            if examples_plus(p, f) >= theta(ex, f)
                rp = rp+1;
                R_plus(rp,:) = examples_plus(p,:);
                wR_plus(rp,:) = w_plus(p);
            else
                lp = lp+1;
                L_plus(lp,:) = examples_plus(p,:);
                wL_plus(lp,:) = w_plus(p);
            end
        end
        for m = 1:size(examples_minus, 1)
            if examples_minus(m, f) >= theta(ex, f)
                rm = rm+1;
                R_minus(rm,:) = examples_minus(m,:);
                wR_minus(rm,:) = w_minus(m);
            else
                lm = lm+1;
                L_minus(lm,:) = examples_minus(m, :);
                wL_minus(lm,:) = w_minus(m);
            end
        end
```

```
            error1 = sum(wL_plus)+sum(wR_minus); %if Right is class
                p and Left is class m
            error2 = sum(wR_plus)+sum(wL_minus); %if Right is class
                m and Left is class p
            [error(ex, f), side_m(ex, f)] = min([error1, error2]);
        end
    end

    min_error = min(min(error));
    [line_m, f_optimal_m] = find(error == min_error);
    f_optimal = f_optimal_m(1);
    example = aux_i(line_m(1));
    side = side_m(example, f_optimal);
    theta_optimal_m = theta(line_m, f_optimal_m);
    theta_optimal = theta_optimal_m(1, 1);
    end
```

For each feature the program generates a line which is orthogonal to the feature axis between every two samples and it splits the data accordingly. Then it computes one error assuming that if feature value $\geq \theta$ is classified as Class 1 and another error assuming the same example is classified as Class 2. Taking the minimum of these two errors we obtain our apparent error and the correct way to partition the data in the decision stump.

c.

The simple data set generated by the two Gaussian distributions is shown in Figure 1 where the horizontal axis corresponds to feature 1 and the vertical axis corresponds to feature 2.

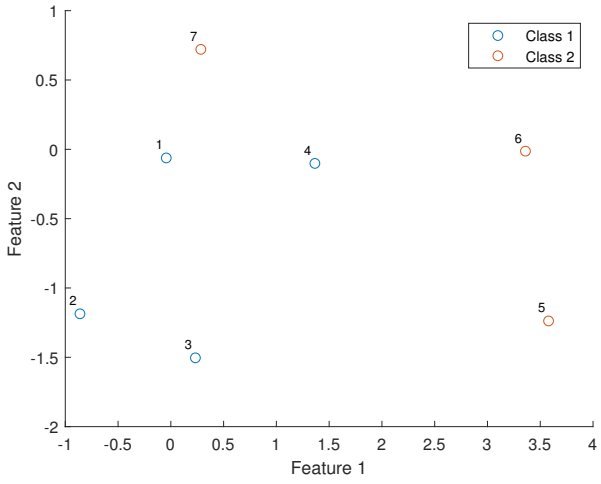

Figure 1: Simple dataset

The function `training_decision_tree` outputs that there are three different ways to

divide the seven data points that will generate the same error. We can split the data from feature one with the threshold θ at 0.2544 (a vertical line between samples number 3 and 7). This results in sample number 4 being misclassified. Another way to split the data is again from feature 1 with θ at 3.0388 with sample number 7 ending up misclassified. The third way is to split from feature 2 with θ at -0.0416, i.e. a horizontal line between samples 1 and 6. Now the misclassified sample is sample number 5. The error generating by these three equivalent decision stumps is of 14.29%.

Scaling feature 2 by a factor of 10 gives the dataset in Figure 2. The results now returned

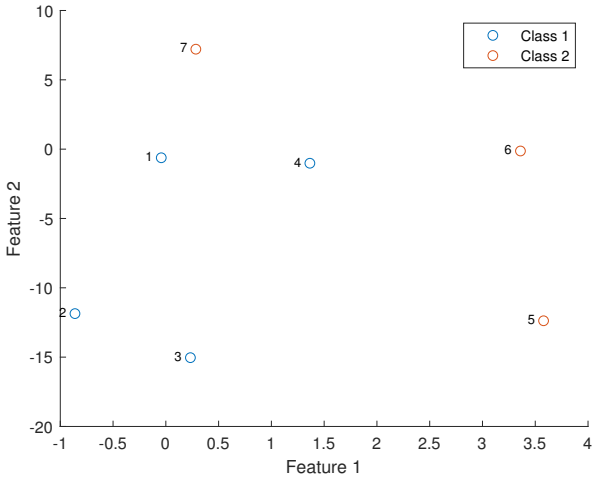

Figure 2: Simple dataset - feature 2 scaled by a factor of 10

by the function are equivalent to the previous ones as the three ways to split the data are still valid producing the same classification error.

d.

Now we apply the decision stump implementation on the Fashion NIST dataset. The optimal feature returned by the weak learner is feature 107 with a threshold $\theta = 0.0629$. The apparent error and test error are, respectively, 18.33% and 50.25%.

e.

Two MATLAB functions were created: one to train the AdaBoost algorithm and another one to use it to classify new data called `training_AdaBoost` and `prediction_AdaBoost` respectively. The codes are presented next.
Training the AdaBoost algorithm:

```
function [hf, weights_f, weak_h] = training_AdaBoost(dataset_t,
    e_plus_t, e_minus_t, w_plus, w_minus, y, T)
% This function is used for training the AdaBoost algorithm, i.
   e., finding the hypothesis weights
```

```
% INPUTS:   dataset_t - complete training set of examples
%           e_plus_t - set of training examples from Class 1
%           e_minus_t - set of training examples from Class 2
%           w_plus - weights of the examples from Class 1
%           e_minus - weights of the examples from Class 2
%           y - known lables of the training set
%           T - number of iterations
% OUTPUTS:  hf - final hypothesis
%           weights_f - final sample weights
%           weak_h - structure containing optimal feature,
   threshold, side
%           and weights of all T weak hypothesis

B = zeros(1, T);
h = zeros(size(dataset_t,1), T);
p = zeros(size(dataset_t,1), T);
%training the decision tree
w = [w_plus; w_minus];
weak_h.f_optimal = zeros(1, T); weak_h.theta_optimal = zeros(1,
    T);
weak_h.a = zeros(1,T); weak_h.side = zeros(1, T);
for t = 1:T
    p(:, t) = w/(sum(w));
    p_plus = p(1:size(e_plus_t, 1),t);
    p_minus = p(size(e_plus_t, 1)+1:end,t);
    %call WeakLearn
    [f, theta, ~, side] = training_decision_tree(dataset_t,
       e_plus_t, e_minus_t, p_plus, p_minus);
    [h(:,t)] = decision_tree(dataset_t, f, theta, w, side);
    error = sum(p(:,t).*abs(h(:,t)-y));
    B(t) = error/(1-error);
    w = w.*(B(t).^(1-abs(h(:,t)-y)));
    weak_h.f_optimal(t) = f; weak_h.theta_optimal(t) = theta;
    weak_h.a(t) = (1/2)*log(1/B(t));  weak_h.side(t) = side;
end

hf = zeros(size(dataset_t,1), 1);
for e = 1:size(dataset_t,1)
    if sum((log10((1./B)).*h(e,:))) >= (1/2)*sum(log10(1./B))
        hf(e) = 1;
    else
        hf(e) = 0;
    end
end
weights_f = p(:,T);
```

```
end
```

Using the AdaBoost algorithm to predict labels:

```
function [label] = prediction_AdaBoost(dataset, weak_h)
% This function is used to predict labels with AdaBoost
% INPUT:    dataset - complete set of examples to predict
%           weak_h - structure containing optimal feature,
   threshold, side
%           and weights of all T weak hypothesis
% OUTPUT:   label - assigned labels of all examples in the
   dataset

a = weak_h.a/sum(weak_h.a);
T = length(a);
hypothesis = zeros(size(dataset,1), T);
w = 1/size(dataset,1)*ones(1, size(dataset,1));
for t = 1:T
    f = weak_h.f_optimal(t); theta = weak_h.theta_optimal(t);
       side = weak_h.side(t);
    [hypothesis(:, t)] = decision_tree(dataset, f, theta, w,
       side);
end
label = zeros(size(dataset,1), 1);
for e = 1:size(dataset,1)
    aux_h = hypothesis(e, :);
    aux_h(aux_h==0) = -1;
    aux = a*aux_h';
    if aux >= 0
        label(e) = 1;
    elseif aux < 0
        label(e) = 0;
    end
end
end
```

f.

Now we apply our implementation of the AdaBoost algorithm to the simple dataset generated for question **c.**. After the last iteration the examples assigned the higher weights are examples number 4, 5 and 7 weighing 0.191, 0.5, 0.309 respectively. The weights of the other examples are of order 10^{-17} or superior. From part **c.** we know that the hardest samples to classify are samples number 4, 5 and 7. Thus the weights assigned to these examples after the T iterations are quite reasonable. Furthermore, the labels vector of the dataset **y** and the final

hypothesis returned by AdaBoost $\mathbf{h}_f$ are

$$\mathbf{y} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{h}_f = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

We can verify, then, that for the simple dataset the AdaBoost results in zero apparent error.

g.

Here we show the results of the application of the AdaBoost algorithm to the Fashion NIST dataset. Tests were run for $\mathrm{T} = \begin{bmatrix} 10 & 20 & 30 & 40 & 50 & 60 \end{bmatrix}$. The errors on the test set obtained are shown in Figure 3. From the tested values of T the one that yields the lowest error rate is $\mathrm{T} = 40$. For $\mathrm{T} = 40$ iterations the weights assigned to the training objects after

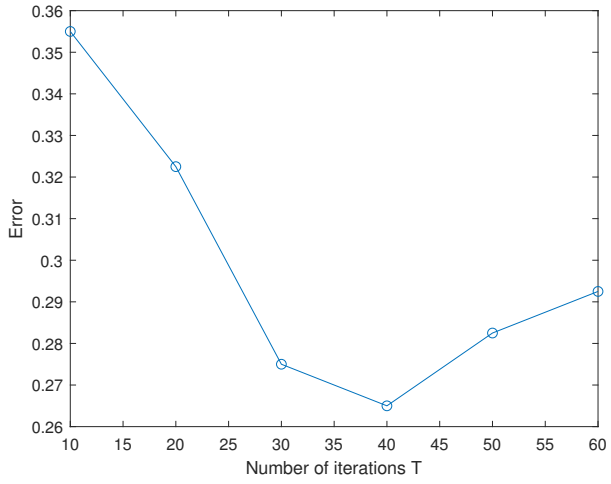

Figure 3: Classification errors vs. number of iterations

the 40th iteration are as shown in Figure 4. The examples assigned with higher weights are examples number 25 and 36, with weights 0.0963 and 0.0857 respectively, followed by example 1 with weight 0.0581 which is considerably lower.

h.

Figure 5 shows the learning curve for $\mathrm{n} = \begin{bmatrix} 2 & 4 & 6 & 10 & 15 & 20 \end{bmatrix}$ training examples per class.

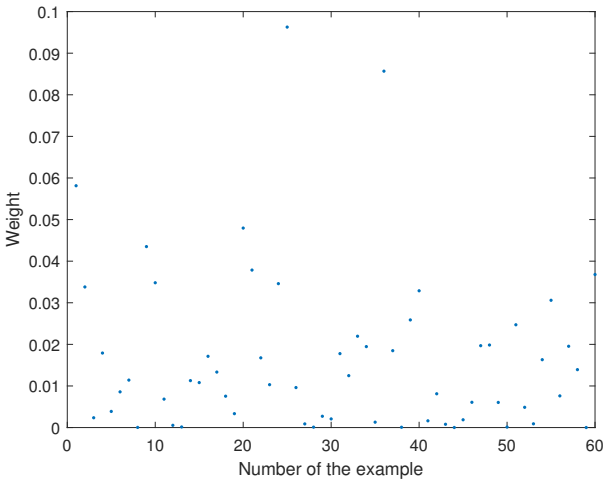

Figure 4: Weights of the training examples

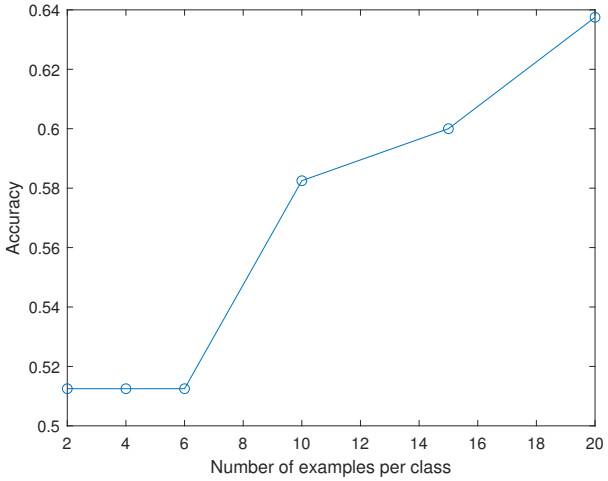

Figure 5: Learning curve