IN4320 Machine Learning: Assignment 5
Computational Learning Theory: Boosting

Georgios Dimitropoulos: 4727657

May 31, 2018

Exercise A

We have to prove that: $e^{-x} \geq (1-x)$. In order to be able to prove this inequality let $f(x) = e^{-x}-1+x$. Thus, it suffices to show that $f(x) \geq 0$ $\forall x \in$ domain of $f(x)$. We are going to find the monotonicity and the critical points of this function. The domain D of $f(x)$ is $D = \mathbb{R}$, all the real numbers, as $f(x)$ consists of an exponential, a polynomial and a constant function. The function $f(x)$ is continuous and differentiable in $\mathbb{R}$. Thus we are able to find its derivative $f'(x) = -e^{-x} + 1$. We set $f'(x) = 0$ and we have: $f'(x) = 0 \Leftrightarrow -e^{-x} + 1 = 0 \Leftrightarrow e^{-x} = 1 \Leftrightarrow \ln(e^{-x}) = \ln 1 \Leftrightarrow \boxed{x = 0}$. Thus the derivative $f'(x)$ has only one root, namely $\boxed{x = 0}$. For this reason it has a constant sign in the ranges $(-\infty, 0]$ and $[0, \infty)$ respectively. In order to find these signs, we calculate the value of the function for one number in these two ranges respectively. Namely, we have that $f'(1) = -e^{-1} + 1 = \frac{-1}{e} + 1 > 0$. Thus the derivative $f'(x)$ is positive in the range $[0, \infty)$ and for this reason the function $f(x)$ is monotonically increasing in this range. Similarly, we have that $f'(-1) = -e^{1} + 1 < 0$. Thus the derivative $f'(x)$ is negative in the range $(-\infty, 0]$ and for this reason the function $f(x)$ is monotonically decreasing in this range. Based on these, we could say that at $x = 0$, the function $f(x)$ has its global minimum with a value of $f(0) = e^0 - 1 = 0$. Hence, the minimum value of $f(x)$ is 0 and for this reason we have that $f(x) \geq 0 \Leftrightarrow e^{-x} \geq (1-x)$ $\forall$ $x \in \mathbb{R}$.
A more intuitively description of our explanation can be depicted in table 1, where the Table of Monotonicity of the function $f(x)$ is given. Furthermore, if we wanted to compute the full range of the function $f(x)$, then we should calculate two limits. Namely we have that, $\lim_{x\to\infty} f(x) = \lim_{x\to\infty} e^{-x} - 1 + x = 0 + \infty - 1 = +\infty$. Similarly, $\lim_{x\to-\infty} f(x) = \lim_{x\to-\infty} e^{-x} - 1 + x = \lim_{x\to-\infty} e^{-x}(1 + \frac{x}{e^{-x}} - \frac{1}{e^{-x}}) = \lim_{x\to-\infty} e^{-x} = +\infty$, (since the terms $\frac{x}{e^{-x}} - \frac{1}{e^{-x}}$ go to zero as x tends to $-\infty$ as we have limits of form $\frac{c\in\mathbb{R}}{\infty}$, (first we used the De l' Hospital rule for the limit $\lim_{x\to-\infty} \frac{x}{e^{-x}}$, since it is in form $\frac{-\infty}{+\infty}$)). Thus the range of function $f(x)$ is $[0, +\infty)$.

Table 1: Table of Monotonicity of the function $f(x)$

x	$(-\infty, 0]$	$[0, \infty)$
$f'(x)$	-	+
$f(x)$	↘	↗

Finally, a more geometric intuition of our proof can be depicted in figure 1, where we can see see that $e^{-x} \geq (1-x)$ $\forall$ $x \in \mathbb{R}$ as the function e^{-x} is always at least above of the function $(1-x)$.

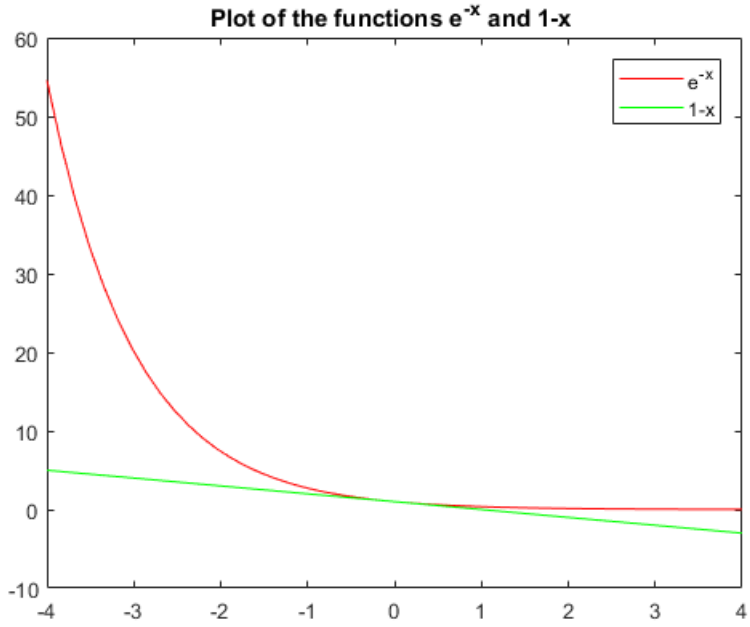

Figure 1: Plot of the functions e^{-x} and $1-x$

Exercise B

In this question we implemented a "weak learner", the decision stump. This classifier accepts as an input a dataset with its corresponding labels and its outputs the optimal f^*, θ^* and y^* for which the classification error on the training set is minimum . The f^* corresponds to the optimal feature of the dataset, the θ^* corresponds to the optimal threshold and the y^* corresponds to the optimal sign ($<$ or $>$). Our MATLAB code is as follows:

```
function [feature,theta,y] = weak_learner(dataset,labels)
%Input: Dataset and corresponding labels
%Output: Optimal feature, theta, and sign.
[n,d] = size(dataset);%n=number of points and d=number of features
minimum_score = 9999999;
for i=1:d
    for j = 1:n
        sign = 0; %sign 0 corresponds to < and sign 1 corresponds to >
        Thetas = ones(n,1)*dataset(j,i);%Compute predictions
        predict0 = dataset(:,i)-Thetas<0;
        predict1 = dataset(:,i)-Thetas>=0;
        predict1 = predict1+1;% Tranform labels to 1 and 2 respectively
        predict0 = predict0+1;
        score0 =sum(abs(predict0-labels));
        score1 = sum(abs(predict1-labels));
        if score0>score1%Find optimal sign (< or >)
            score0 = score1;
            sign =1;
        end
        if score0<minimum_score%Find optimal feature,theta and sign for
            which the classification error on the training set is
           minimum
            minimum_score = score0;
            y = sign;
            feature = i;
            theta = dataset(j,i);
```

```
            end
        end
    end
end
```

Exercise C

In order to be able to test our implementation, we performed two different tests. The first one was performed on the dataset generated by gendats from Prtools, where we have 2 classes with 200 samples in total (97 points belong in class 1 and the remaining 103 belong in class 2). The second test was performed in a dataset which generated two classes from two Gaussian distributions, where the means were $\mu_1 = [0,0]^T$, $\mu_2 = [2,0]^T$ and the covariance matrices were identity matrices, where we have 2 classes with 200 samples in total (100 points belong in class 1 and the remaining 100 belong in class 2). The scatterplot of the first dataset can be depicted in figure 2.

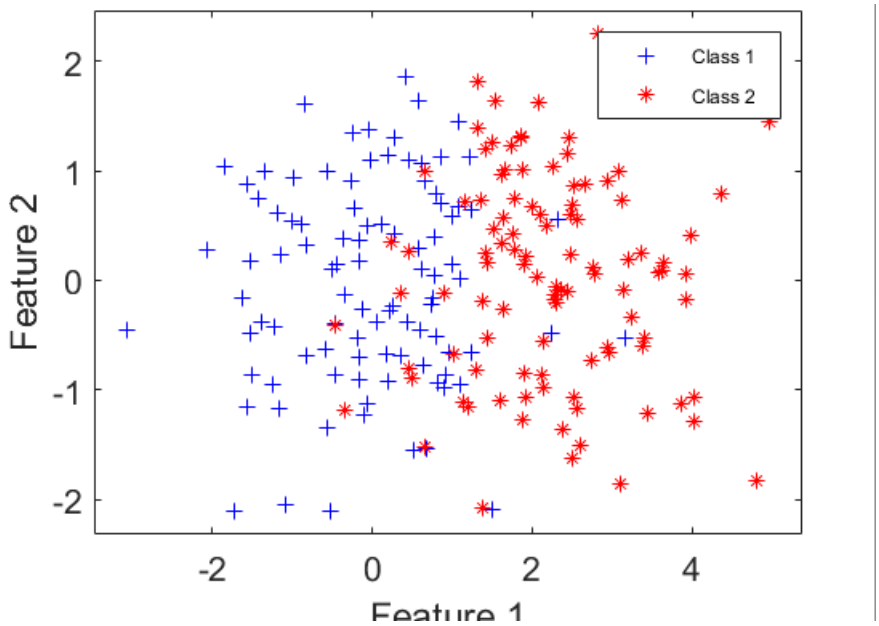

Figure 2: Scatterplot of the first dataset

The optimal parameters which were obtained by our decision stump in this case were: $f^* = 1, \theta^* = 1.1475$ and $y^* = 1$. This means that the most indicative feature was the feature 1 and the $y^* = 1$ means that if $x_f < \theta^*$ then assign to class 1.
In case which we rescale one of the features, the decision stump might change or might not. To be more specific, in case that we multiplied feature 2 by a factor of 10, then the optimal parameters f^*, θ^* and y^* remained exactly the same, as the indicative(optimal) feature (feature 1) stayed the same. Nevertheless, in case that we multiplied feature 1 by a factor of 10, then the optimal parameters f^* and y^* remained exactly the same, but θ^* was multiplied also by a factor of 10 as well and it became as we expected $\theta^* = 11.4747$.
In a similar manner, the scatterplot of the second dataset can be depicted in figure 3.

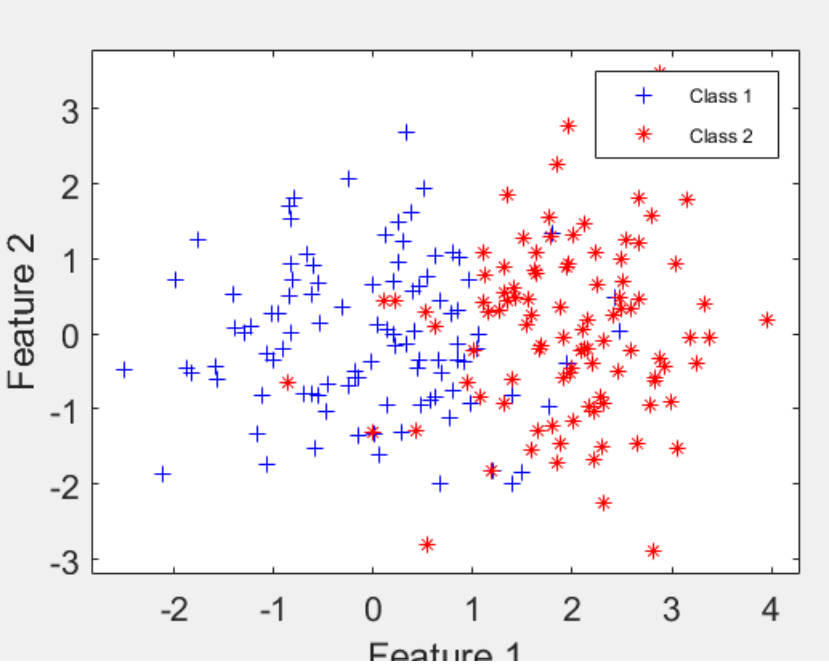

Figure 3: Scatterplot of the second dataset

The optimal parameters which were obtained by our decision stump in this case were: $f^* = 1, \theta^* = 1.0774$ and $y^* = 1$. Again, this means that the most indicative feature was the feature 1 and the $y^* = 1$ means that if $x_f < \theta^*$ then assign to class 1.
Similarly, in case which we rescale one of the features, the decision stump might change or might not. To be more specific, in case that we multiplied feature 2 by a factor of 10, then the optimal parameters f^*, θ^* and y^* remained exactly the same, as the indicative(optimal) feature (feature 1) stayed the same. Nevertheless, in case that we multiplied feature 1 by a factor of 10, then the optimal parameters f^* and y^* remained exactly the same, but θ^* was multiplied also by a factor of 10 as well and it became as we expected $\theta^* = 10.7736$.

Exercise D

In this question we tested our implementation on the optdigitsubset dataset. We used the first 50 objects for each class for training and the rest for testing. The classification error which we calculated on the test objects was 1.7561%. In case that we take other random subsets of 50 for training, the testing error did not vary so much. In order to be able to give an intuition about this, we performed 100 iterations of random subsets of 50 for training and the corresponding test error can be depicted in figure 4. The mean of the error was found to be 2.2142% and the standard deviation of the error 1.1897%. Finally, we could say that the performance with training on the first 50 object seems to be unexpectedly good as it achieves a very small rate of test error.

Figure 4: Test Error rate vs Number of Iterations

Exercise E

In this question we extended the implementation of the weak learner in a way that it accepts a weight per object and minimizes the weighted classification error. In order to be able to test our code we used a simple classification problem. The dataset which we used can be depicted in figure 5. It consists of 2 classes where the first class has 6 observations and the second class has 4 observations. Namely the first class consists of the points A(-0.3421, -0.3343), B(-0.6724, 0.9848), C(-0.1841, -0.9958), D(-0.3892, 1.2271), E(0.4539, 0.3243) and F(1.7156, 0.0377) and the second class consists of the points A(0.6661, 0.3584), B(3.1638, 1.3126), C(2.3551, 0.6523) and D(1.9052, 2.5487).

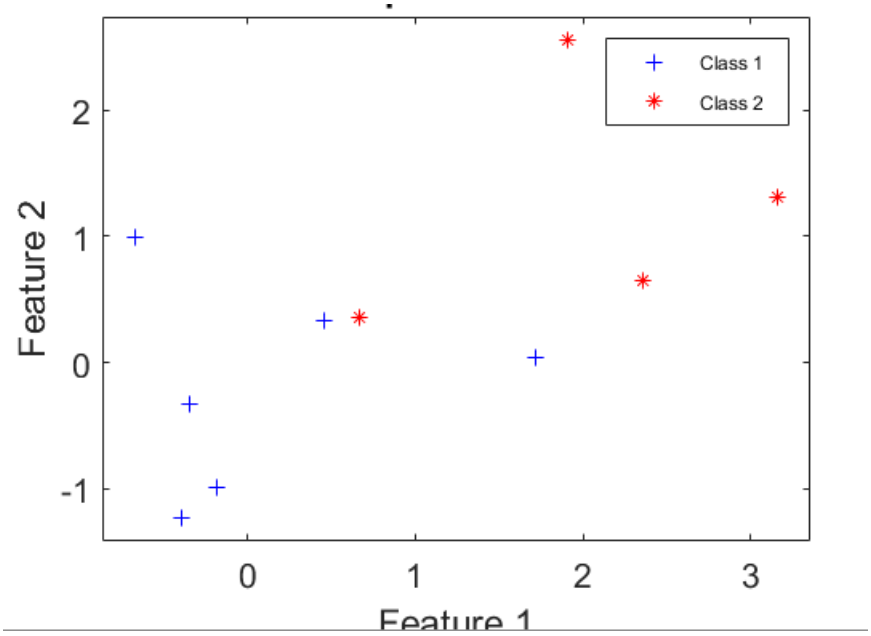

Figure 5: Scatterplot of the Dataset

Initially, we used the extension of the implementation of the weak learner and we set all the

weights for all the 10 samples to 1. The optimal parameters which were obtained in this case were: $f^* = 1, \theta^* = 0.4539$ and $y^* = 1$. The scatterplot of our dataset in combination with the optimal threshold for the case that all the weights were equal to 1 can be depicted in figure 6.

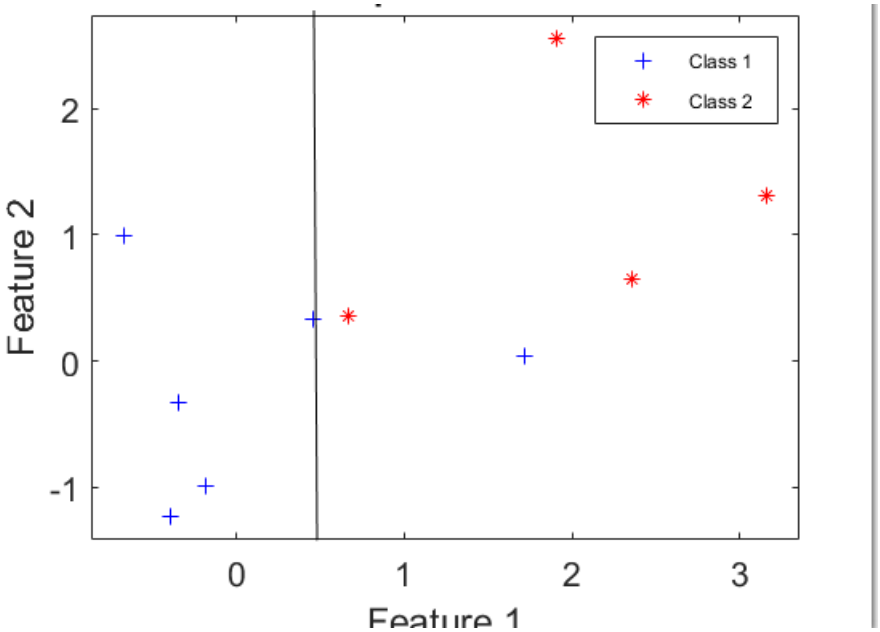

Figure 6: Scatterplot of our dataset in combination with the optimal threshold for the case that all the weights were equal to 1

We can observe from figure 6 that one object is misclassified. Namely, we have that the object F(1.7156, 0.0377) of the first class is misclassified. For the reason, we assign a very large weight to this point. After that, the the optimal parameters which were obtained in this case were: $f^* = 1, \theta^* = 1.7156$ and $y^* = 1$. The scatterplot of our dataset in combination with the optimal threshold for the case that the object F(1.7156, 0.0377) has a very large weight can be depicted in figure 7, where we can observe that this point is now classified correctly as the θ^* moves towards this point.

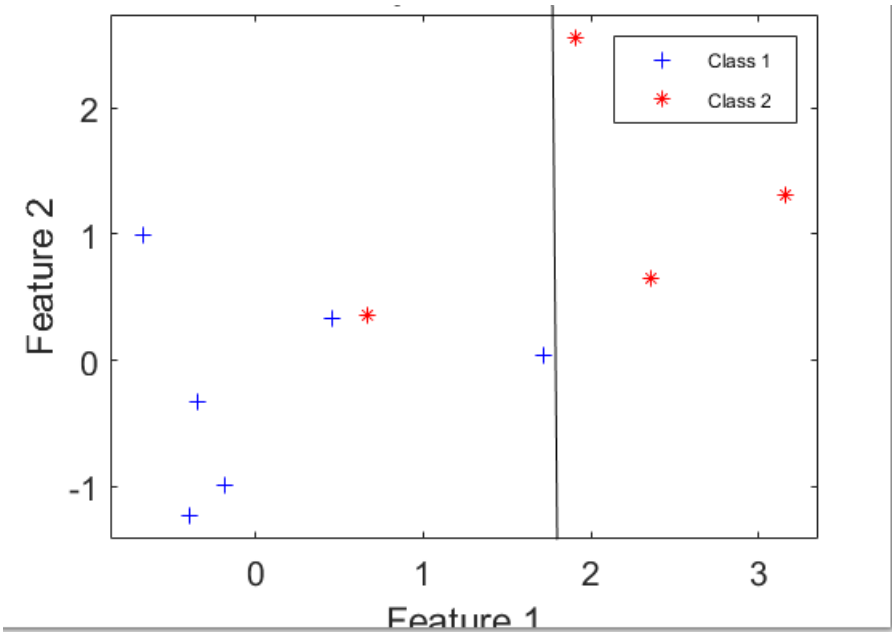

Figure 7: Scatterplot of our dataset in combination with the optimal threshold for the case that the object F(1.7156, 0.0377) has a very large weight

Hence, form figure 7 we could infer that objects which have a higher weight have a larger in-

fluence on the solution.

Exercise F

In this question we implemented the AdaBoost algorithm that is described in [1]. This algorithm accepts as an input a dataset with its corresponding labels and the number of iterations and its outputs are the predicted labels for the data, the beta and parameters for all the base classifiers and the Weight matrix which contains the weight for each point for each iteration. AdaBoost initially finds the weak learner which has the lowest error rate on the data. After that, the data which were classified correctly are given a lower weight in comparison to them that were misclassified in which are given a higher weight. In the sequel, AdaBoost starts a new iteration and selects the best weak learner based on these weighted data. Thus, based on this procedure finds a new weak learner that is better in classifying the samples that were misclassified by the previous weak learners and finally the learned classifier based on a majority vote of each weak classifier Our MATLAB code for the AdaBoost is as follows:

```
1  function [predictedLabels,beta,parameter,W] = adaBoost(dataset,
       iterations,labels)
2  % Input: Dataset, corresponding labels and number of iterations
3  % Output: Predicted labels, beta and parameters of base classifiers and
4  % weight matrix W
5  n = size(dataset,1);%number of samples
6  weight = ones(n,1);%Initialization
7  W = ones(n,iterations);
8  beta = zeros(iterations,1);
9  parameter = zeros(iterations,3);
10 for t = 1:iterations
11     p = weight./sum(weight);
12     W(:,t) = p;
13     [feature,theta,y] = weightedWeakLearner(dataset,p,labels);
14     parameter(t,:) = [feature,theta,y];
15     [error,prediction] = ErrorCalculation(feature,theta,y,dataset,p,
           labels);%A function that returns the error
16     if error==0
17         error=0.00001;%avoid division by 0
18     end
19     beta(t) = error/(1-error);%update of our parameters
20     weight = weight.*(beta(t).^(1-abs(prediction-labels)));
21 end
22 predictedLabels = adaPrediction(beta,parameter,dataset);%Function that
       classify new and unseen data
23 end
```

Inside this code, there are other two functions which we created. Firstly, in line 13, we use the function weightedWeakLearner which is simply a modification (was implemented in Question E) of the weaklearner function which we used in exercise B in the way that the error is calculated (we now have instead: $score = weight' * abs(predict - labels)$). Furthermore, the second function which we used in line 15 is the function ErrorCalculation that takes as input the optimal feature, theta and sign which are returned by the weightedWeakLearner, the corresponding samples with their labels and the distribution p and returns the predicted labels and the error. Our MATLAB code for the ErrorCalculation is as follows:

```
1  function [error,prediction] = ErrorCalculation( feature,theta,y,X_test,
       weight,lab_test)
2  weight = weight/sum(weight);%initialization of the weight
3  n = size(X_test,1);%number of points
4  Theta = ones(n,1)*theta;
5  if y==0
```

```
6       prediction = X_test(:,feature)-Theta<0;
7       prediction = prediction+1;
8       score = weight'*abs(prediction-lab_test);
9
10  else
11      prediction = X_test(:,feature)-Theta>=0;
12      prediction = prediction+1;
13      score = weight'*abs(prediction-lab_test);
14
15  end
16  error = sum(score)/n*100;
17  end
```

Finally, we implemented another function (which is used in line 22 in the AdaBoost code) with the name adaPrediction which is used to classify new and unseen data and takes as an input the beta and parameters for all the base classifiers and our data and returns their predicted labels. Our MATLAB code for the adaPrediction is as follows:

```
1  function [predictedLabels] = adaPrediction(beta,parameter,dataset)
2  % Input: beta, parameter=[feat,theta,y], dataset
3  % Output: predictedLabels
4  N = size(dataset,1);%number of data
5  b = size(beta,1);
6  baseScore = 0.5*sum(log(1./beta));
7  scores = zeros(N,b);
8  for t=1:b
9      feature = parameter(t,1);
10     theta = parameter(t,2);
11     y = parameter(t,3);
12     Theta = ones(N,1)*theta;
13     if y==0
14         scores(:,t) =dataset(:,feature)-Theta<0;
15     else
16         scores(:,t) =  dataset(:,feature)-Theta>=0;
17     end
18     scores(:,t) = scores(:,t)*log(1/beta(t));
19 end
20 predictedLabels = sum(scores,2)>=baseScore;
21 predictedLabels = predictedLabels+1;
22 end
```

Exercise G

In this question, we tested our implementation of the AdaBoost algorithm for 100 iterations. We tested on two cases. The first time we tested it on a simple dataset like gendats, where we have 2 classes with 10 samples in total (7 points belong in class 1 and the remaining 3 belong in class 2). The scatterplot of the this dataset can be depicted in figure 8. This dataset contains a point that is difficult to be classified in class 2, namely the 10th point of our dataset. This point can be depicted in figure 9 where we marked it. Hence, we trained the AdaBoost classifier in this dataset, and we visualized the weights for each point at each iteration. A colormap of the weights for each point at each iteration can be observed in figure 10. As we expected, this data point has a larger weight in comparison with the weights of the other points. Furthermore, we could also mention here, that in this case (simple dataset) only one point has a large weight and all the other points have relatively smaller weights. Thus we could say that the large weights are more sparse in this case (only very few points have large weights in the simple dataset).

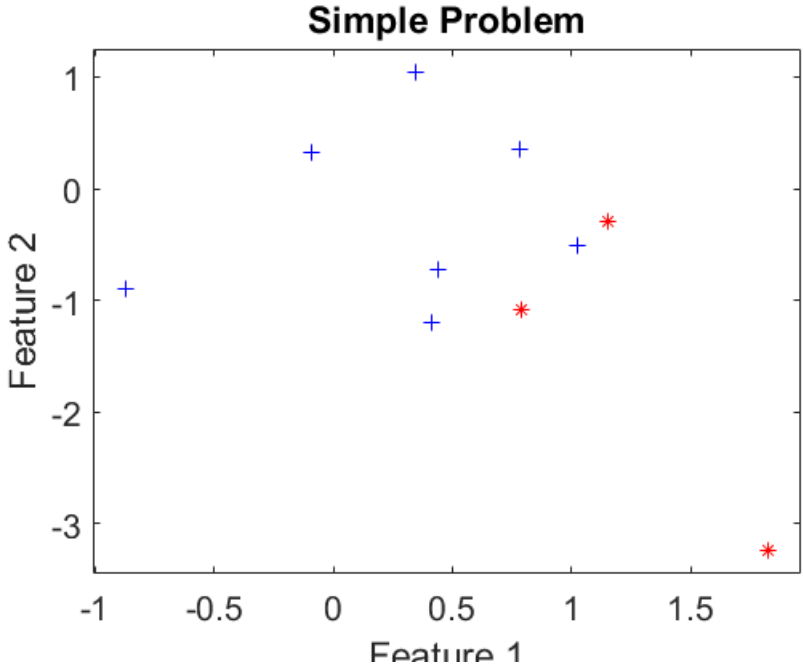

Figure 8: Scatterplot of the dataset

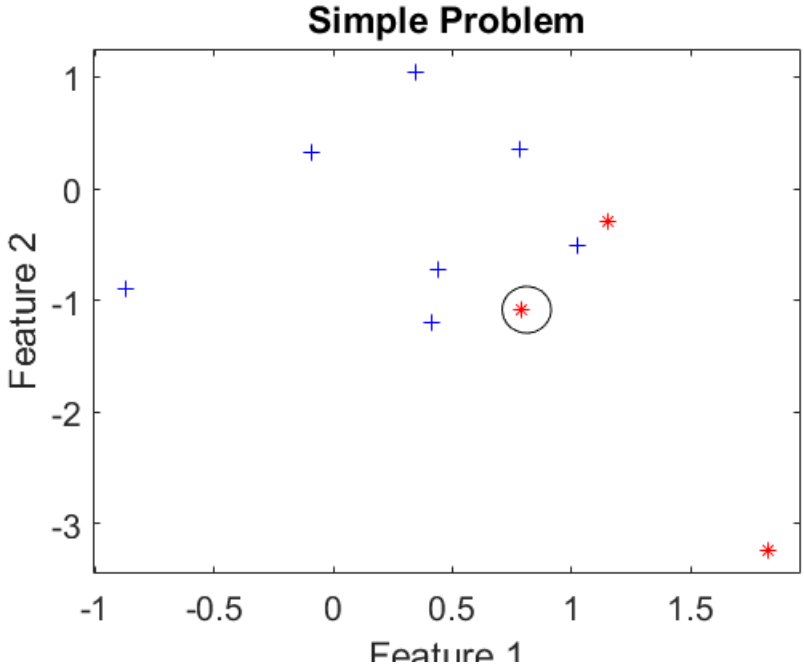

Figure 9: Scatterplot of the dataset-10th point of the dataset (marked)

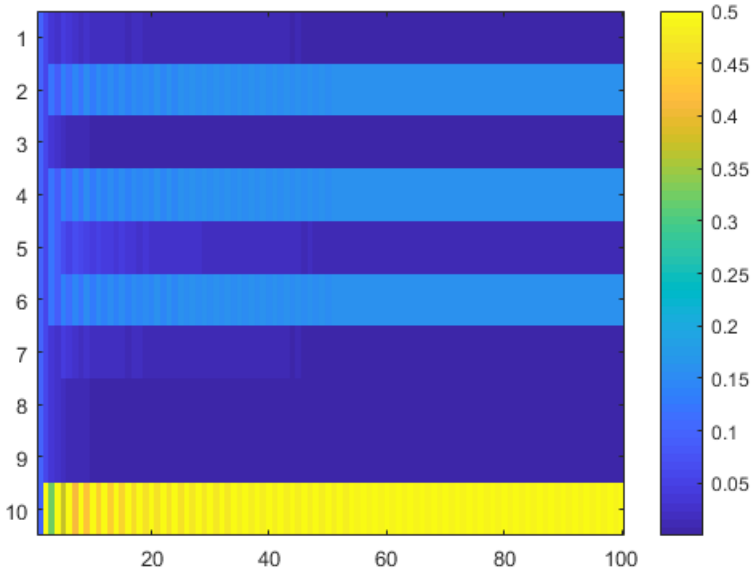

Figure 10: Colormap of the weights for each point at each iteration

Furthermore we tested we tested our implementation on a more complicated dataset like gendatb (Banana Dataset), where we have 2 classes with 50 samples in total (25 points belong in class 1 and the remaining 25 belong in class 2). The scatterplot of the this dataset can be depicted in figure 11. This dataset contains more that one points that are difficult to be classified. For example the 15th and the 32th point of our dataset. These points can be depicted in figure 12 where we marked them. Hence, we trained the AdaBoost classifier in this dataset, and we visualized the weights for each point at each iteration. A colormap of the weights for each point at each iteration can be observed in figure 13. As we expected again, these data points have a larger weight in comparison with the weights of the other points. Also there are some other points which are difficult to be classified and have a large weight too. Furthermore, we could also mention here, that in this case (complicated dataset, there is a larger class overlap) there are many points that have a large weight. Thus we could say that the large weights are more diffuse in this case (many points have a large weight (maybe not so large as in the easy case) in the more complicated dataset).

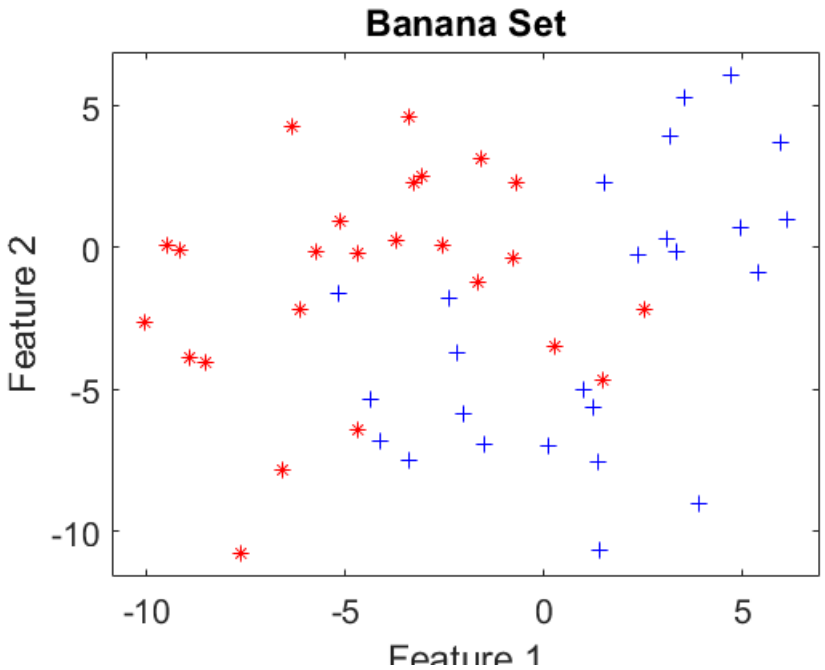

Figure 11: Scatterplot of the dataset

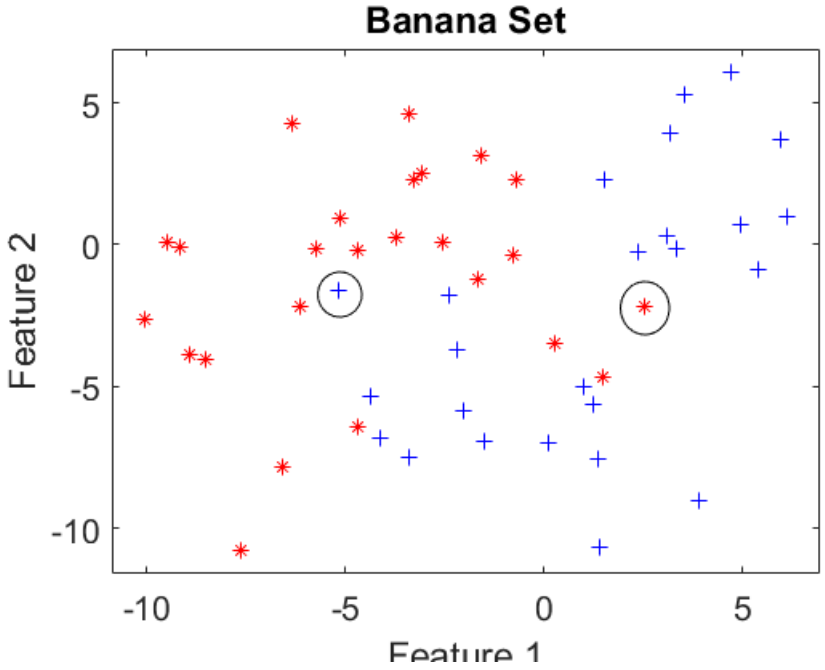

Figure 12: Scatterplot of the dataset-15th and the 32th point of our dataset (marked)

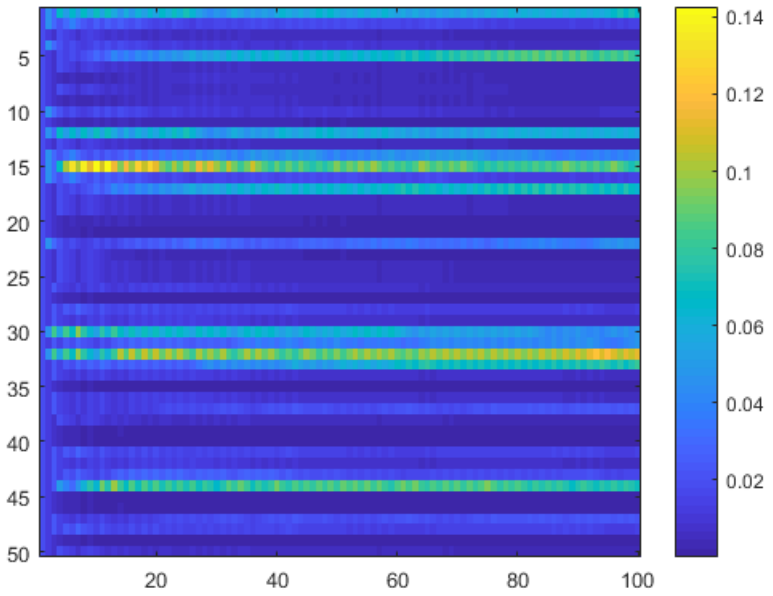

Figure 13: Colormap of the weights for each point at each iteration

Exercise H

In this question, we tested the implementation of the AdaBoost algorithm on the optdigitsubset dataset. We used the first 50 objects for each class for training and the rest for testing. The train and test error for each iteration can be depicted in figure 14.The mean classification error after 100 iterations on the test objects was 1.2410% and the standard deviation was 0.1249. We can observe from figure 14 that the error on the test objects reaches its minimum value on iteration 17 and is equal to 0.7805%. Furthermore, it can be depicted from figure 14 that as the number of iterations is increased, then the train error drops dramatically and remains stable at zero. In the contrary, the testing error, as the number of iterations is increased, initially it is also decreased but in the sequel fluctuates around a certain range.

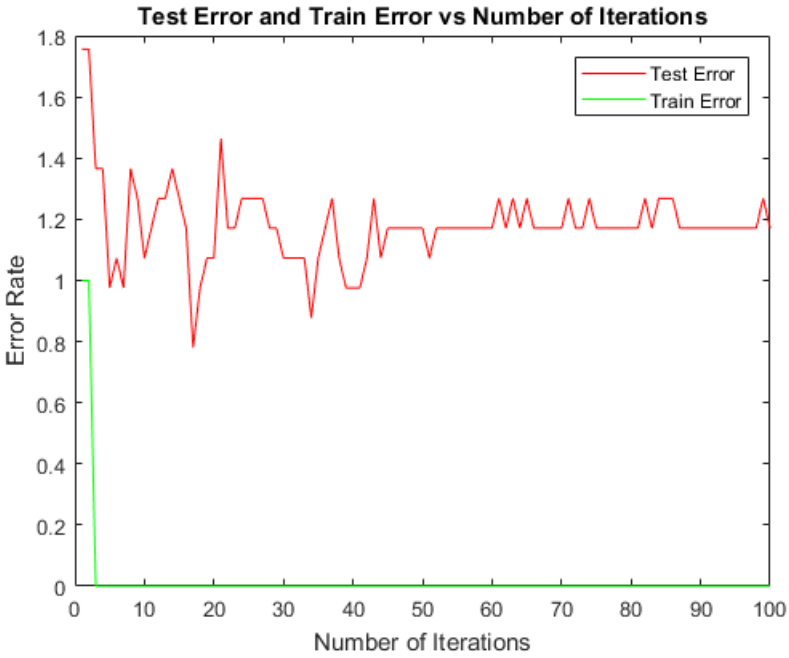

Figure 14: Test Error and Train Error vs Number of Iterations

In the sequel, we took the AdaBoost classifier with this optimal number of iterations (17) and trained it again. Thus, we obtained the weights for each object during each iteration. A colormap of the weights for each point at each iteration can be observed in figure 15.

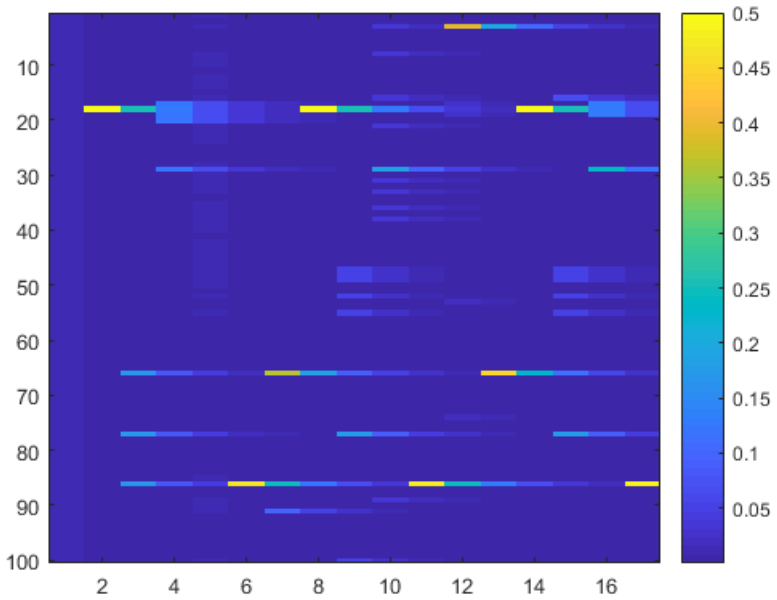

Figure 15: Colormap of the weights for each point at each iteration

It can be depicted from figure 15 that objects 18, 66 and 86 have a high weight during the iterations. Finally, we tried to investigate the reasons that these objects have a large weight. Thus we visualized these images and can be depicted in figures 16-18

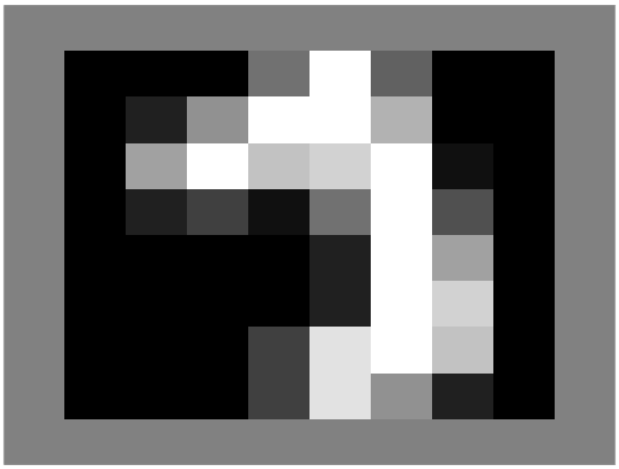

Figure 16: Object 86

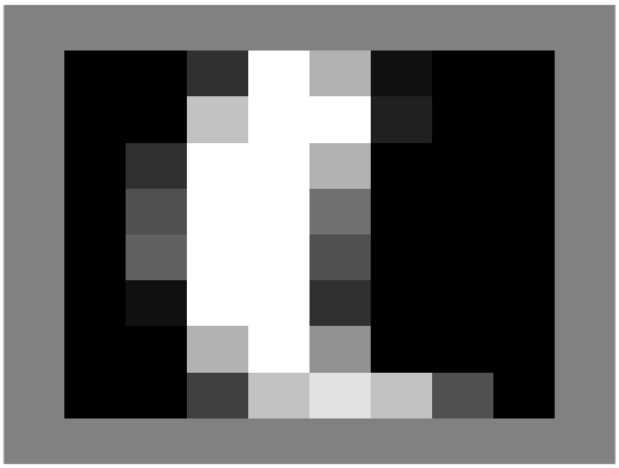

Figure 17: Object 66

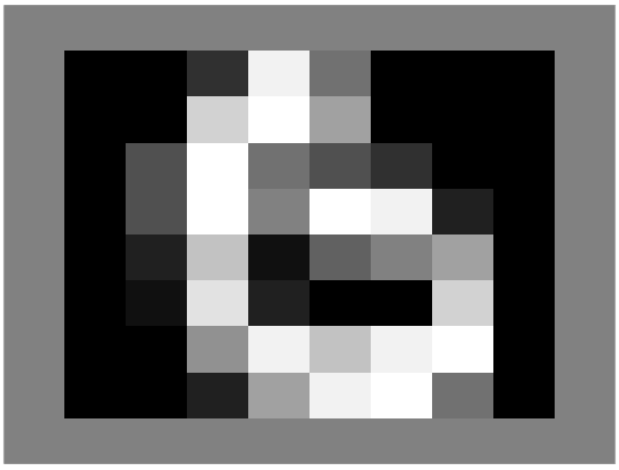

Figure 18: Object 18

Observing figures 16-18 we could depict that these images have some distortions and strange shapes (too noisy). Thus, for these reasons it seems that they are difficult enough objects to be classified correctly and due to these facts they have larger weights during the iterations.

Finally, it can be depicted from figure 15 that objects 01, 02 and 03 have a low weight during the iterations. Hence, we tried to investigate the reasons that these objects have a low weight. Thus we visualized these images and can be depicted in figures 19-21

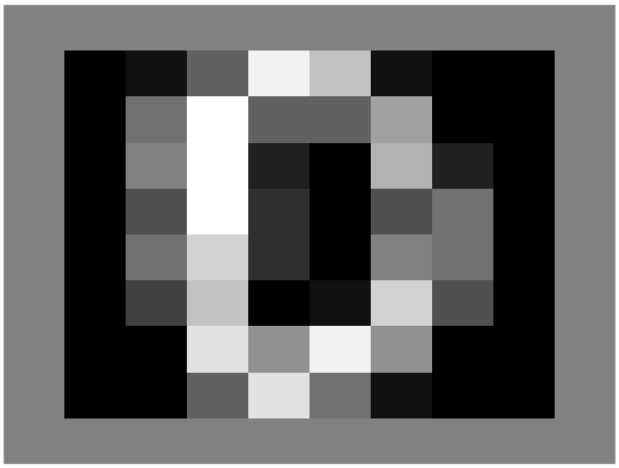

Figure 19: Object 01

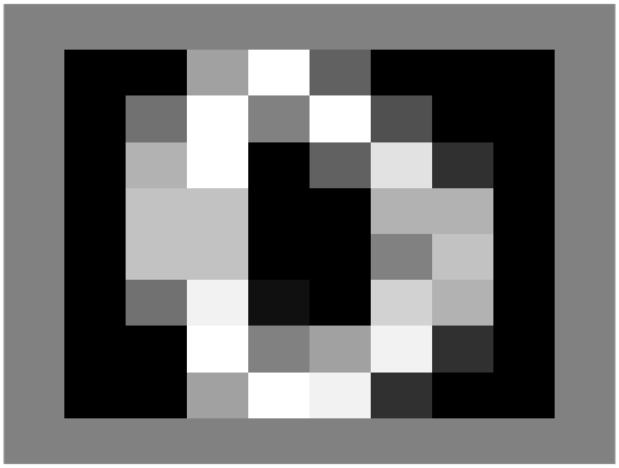

Figure 20: Object 02

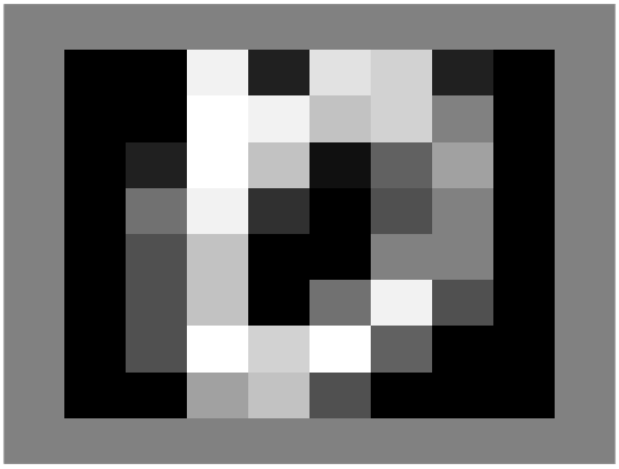

Figure 21: Object 03

Observing figures 19-21 we could depict that these images are less noisy. Thus, for this reason it seems that they are easier objects to be classified correctly and due to this fact they have smaller weights during the iterations.

References

[1] *Yoav Freund , Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In Journal of Computer and System Sciences, v.55 n.1, p.119-139, Aug. 1997*