IN 4320 Machine Learning Assignment 5

Xiangwei Shi
4614909

a.

In order to prove $e^{-x} \geq (1-x)$, it equals to prove $e^{-x} - (1-x) \geq 0$.
Let us set $f(x) = e^{-x} - (1-x)$. And then we take the first order derivative of $f(x)$, which is $f'(x) = -e^{-x} + 1$. And then let $f'(x) = 0$, which means $e^{-x} = 1$. The result of this equation is $x = 0$, which also means $x = 0$ is either maximum or minimum of $f(x)$.
When $x > 0$, we can find that $f'(x) > 0$. And when $x > 0$, it will be $f'(x) < 0$. Therefore, we can tell that when $x = 0$, $f(x)$ has a minimum, of which value will be $f(0) = 0$. Because the minimum of $f(x)$ is 0, $f(x) = e^{-x} - (1-x) \geq 0$ always stands, which means $e^{-x} \geq (1-x)$ is proven.

b.

In this section, a decision stump is implemented in Matlab code, which is shown below. This decision stump function is a simple classifier that chooses optimal f^*, θ^* and y^*.

```
function [f,theta,y] = weaklearner(pr_dataset)
% This function 'weaklearner' is for IN4320 Computational Learning Theory
% question b: implement a 'weak learner'
% This decision stump is for finding the optimal f, theta and sign y for
% which the classification error on the training set is minimum.
% The input of this function is prdataset, which contains data and label.
% The output of this function are the optimal f, theta and sign y.
data = getdata(pr_dataset);
label = getlab(pr_dataset);
[num, feature] = size(data);
%num represents the number of data sample, feature represents the ...
    number of features
threshold = Inf;
for i = 1:feature
    for j = 1:num
        theta_temp = data(j,i);
        % theta is selected as the value of each feature of each sample,
        sign1 = data(:,i)≥theta_temp; % if data<theta_temp, 0; ...
            otherwise, 1: '<'
        sign2 = data(:,i)≤theta_temp; % if data>theta_temp, 0; ...
            otherwise, 1: '>'
        % since the labels are 1 and 2
        score1 = sum(abs(sign1+1−label));
        score2 = sum(abs(sign2+1−label));
        % To select '<' or '>'
```

```
        if score1<score2
            score = score1;
            y_temp = 0; % '<'
        else
            score = score2;
            y_temp =1; % '>'
        end
        % Comparing the current error with the smallest error from before
        if score<threshold
            threshold = score;
            y = y_temp;
            f = i;
            theta = theta_temp;
            % the optimal theta here is not the actual optimal one, but is
            % closed to it
        end
    end
end
end
```

c.

To test the decision stump implemented in b, two classes from two Gaussian distributions are generated, where the means are $\mu_1 = [0, 0]^T$ and $\mu_2 = [2, 0]^T$, and the covariance matrices are identify matrices. The scatterplot is show below, Figure 1. The optimal parameters obtained by the Matlab code above are

$$f^* = 1, \theta^* = 0.9333, y^* = 0,$$

which means feature 1 is the optimal feature, 0.9333 is the optimal threshold and if $x_f < 0.9333$, the object is assigned to class 1.

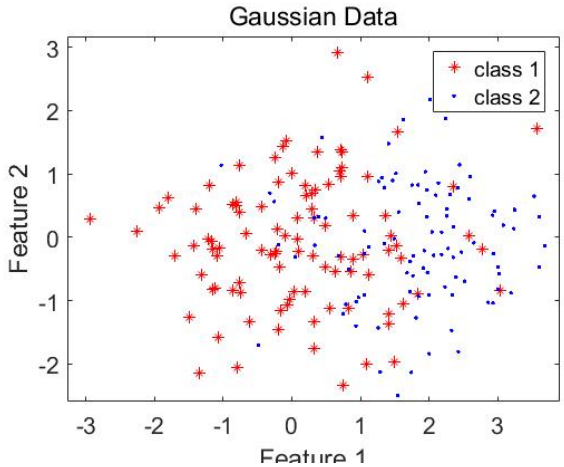

Figure 1: Scatterplot of data

If one of the features is rescaled, the decision stump might change, which depends on which feature will be rescaled. If feature 2 in this care is multiplied by a factor of 10, the optimal parameters will basically remain the same. However, if the feature 1 is rescaled in this case,

the optimal parameter will change, such as the optimal θ^*.

d.

Using the first 50 objects from both classes for training, and the rest for test, I get the classification error, 0.0176(18/1025). As for another condition, using random 50 objects for training and the rest for test, I got the mean and standard deviation of the 50 times errors are 0.0266 and 0.0137. The scatterplot of errors is shown below, Figure 2. Clearly, the performance with the first 50 objects as training dataset is unexpectedly good.

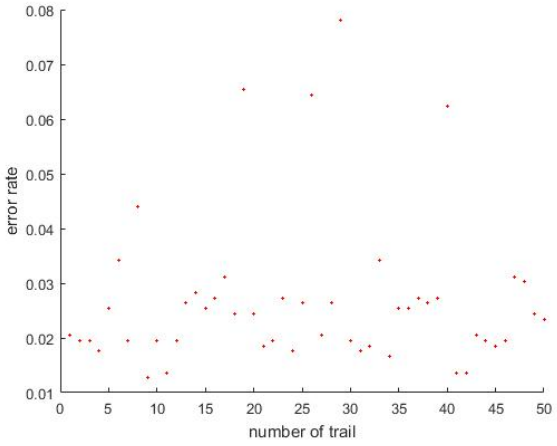

Figure 2: Scatterplot of error(randomly select 50 samples from both classes as training set)

e.

To extend the implementation of the weak learner in part b with weights, the classification error should be weighted. Therefore, in the Matlab code, the classification error becomes below.

```
score1 = weight*abs(sign1+1-label);
score2 = weight*abs(sign2+1-label);
```

To test the code of weighted classification error, a simple classification problem is used. I used 7 objects from both classes generated from part c. The scatterplot is shown below, Figure 3. Firstly, I assume the weights of all 14 objects are the same. Then the optimal parameters are

$$f^* = 1, \theta^* = -0.1924, y^* = 0.$$

And the values of the first feature of these 14 objects are (class 1: 0.7172, 2.5855, -1.1658, 0.0859, -1.4916, -0.4390, 1.1174) and (class 2: -0.1924, 2.3376, 2.5812, 0.4938, 1.8078, 1.9692, 1.0585). And clearly, the first two, the fourth and the seventh objects of class 1 are misclassified. Secondly, I change the weights and increase the weights of those four objects. The optimal parameters change and they are

$$f^* = 1, \theta^* = 1.8078, y^* = 0.$$

This time, only the second object of class 1 and the first and last objects of class 2 are misclassified, where the number of misclassification decreases. The optimal threshold of the second situation is forced to move along the positive direction of feature 1, making the decision boundary closer to the objects of which the weights are increased, which can prove that objects with higher weight have a large influence.

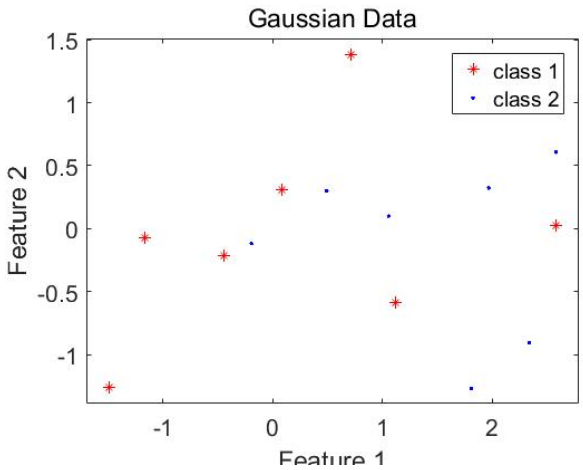

Figure 3: Scatterplot of simple classification

f.

The Matlab code of AdaBoost is shown below.

```
function [predicted,weight] = adaboost(pr_dataset,iteration)
% This function 'adaboost' is for IN4320 Computational Learning Theory
% question f: implement 'AdaBoost'
% The input of this function is prdataset, which contains data and label,
% and the time of iteration.
% The output of this function are the predicted label and the weights of
% different objects.
data = getdata(pr_dataset);
label = getlab(pr_dataset);
[num, feature] = size(data);
w = ones(1,num);
beta = zeros(iteration,1);
parameter = zeros(iteration,3);
% iterating T times and calculating T weighted weaklearner
for i = 1:iteration
    p = w./sum(w);
    weight(:,i) = p;
    [f,theta,y] = weighted_weaklearner(pr_dataset,p);
    parameter(i,:) = [f,theta,y];
    if y==0
        predict = data(:,f)≥theta;% if data<theta, 0; otherwise, 1: '<'
        error = w*abs(predict+1-label);
    else
        predict = data(:,f)≤theta;% if data>theta, 0; otherwise, 1: '>'
```

```
        error = w*abs(predict+1-label);
    end
    if error == 0 % in case weight is zero all the time
        error = 0.001;
    end
    beta(i) = error/(1-error);
    w = w.*(beta(i).^(1-abs(predict-label)))';
end
temp = 1/(2*sum(log(1./beta))); % from the paper hypothesis
scores = zeros(num,iteration);
for i = 1:iteration
    f = parameter(i,1);
    theta = parameter(i,2);
    y = parameter(i,3);
    if y == 0
        scores(:,i) = data(:,f)>=theta;
    else
        scores(:,i) = data(:,f)<=theta;
    end
    scores(:,i) = scores(:,i).*log(1./beta(i));
end
predicted = sum(scores,2)>=temp; % making labels as 1 and 2
predicted = predicted+1;
end
```

g.

To test the implementation of AdaBoost, firstly, I test some simple dataset, the same 14 objects with 2 classes in 2-dimensional space, Figure 3. The weights after AdaBoost become (class 1: 1.1228, 3.1447, 1, 1.1228, 1.0401, 1, 1.1228) and (class 2: 0.3433, 1, 0.9615, 0.3433, 1, 0.9615, 0.3433). Apparently, the weights of the first two, the fouth, the fifth and the last objects increase, which is mainly consistent with the result of part e. The misclassified object without weighted classification error have larger weights. Secondly, I test the code with banana shaped dataset. The scatterplot of banana shaped dataset is shown below, Figure 4. We can find a black square(object 40) and a blakck diamond(object 64) in Figure 4, which are two objects that are easily misclassified without weighted classification error. After AdaBoost, the weights of these two objects increase, shown in Figure 5.

h.

To test the implementation of Adaboost on dataset *optdigitsubset*, using the first 50 objects for training, I plot the relation between test classification error with iteration T, Figure 6. And it is easy to find the smallest classification error, where the optimal iteration is 17. And the weights are plotted in Figure 7. We can get 18th, 67th, and 88th objects have the highest weights, which are shown in 8, 9, 10. From the figures, we can get these three objects are not easy to distinguish. Therefore, they have highest weights.

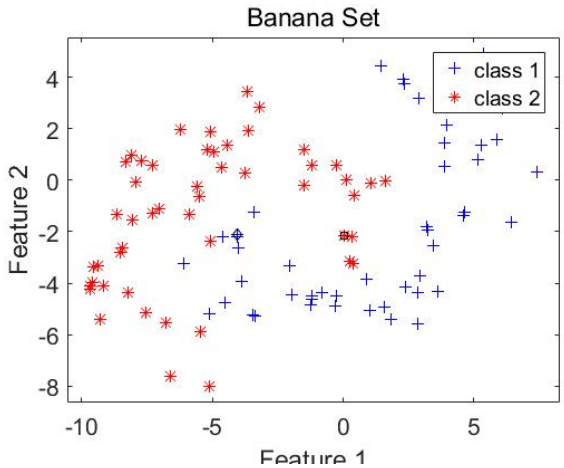

Figure 4: Scatterplot of banana shaped dataset

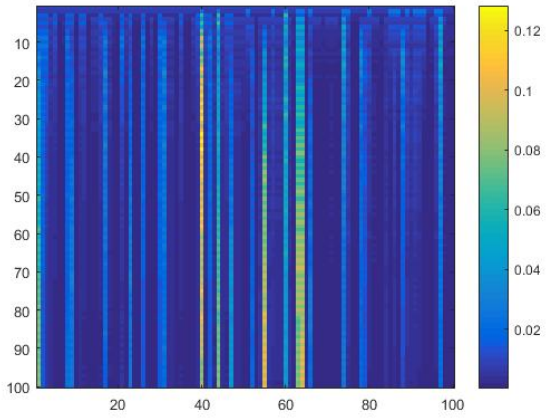

Figure 5: Display weights with scaled colors

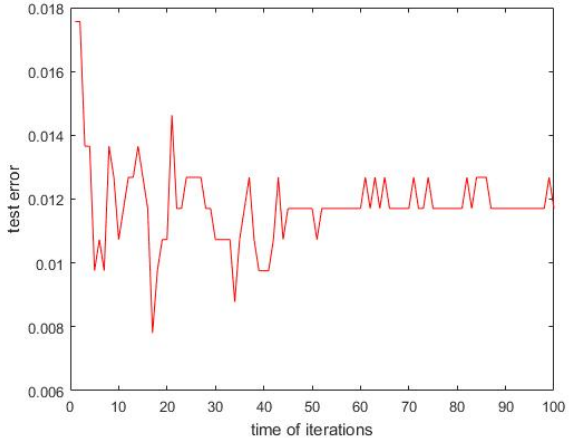

Figure 6: Relation between test classification error with the time of iteration

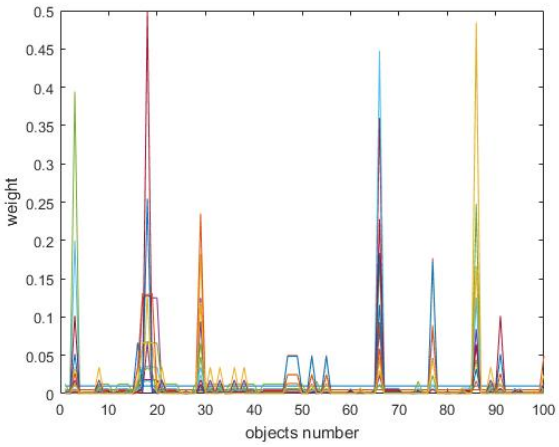

Figure 7: Relation between weight with the time of iteration

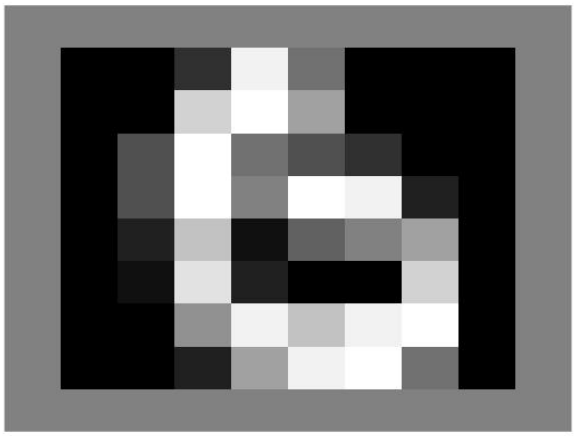

Figure 8: Object 18 with high weights

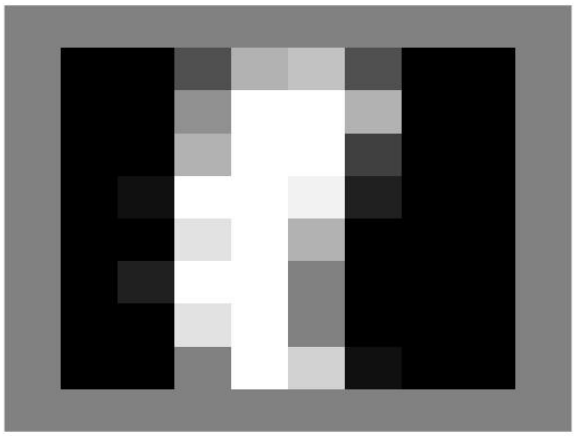

Figure 9: Object 67 with high weights

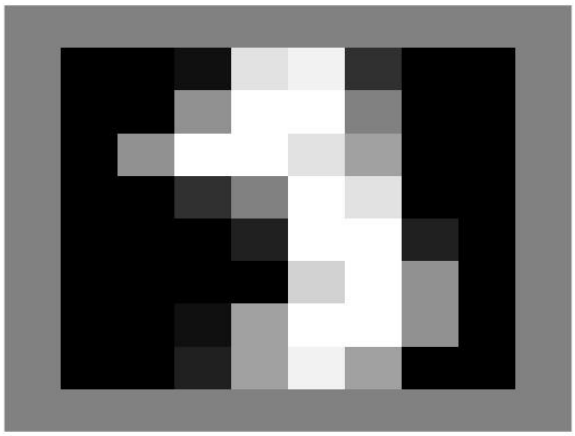

Figure 10: Object 88 with high weights