IN4320 Machine Learning Exercise 2

Student number:4795245

a. Show 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:
The calculation of Adaboost can be divided into five sequential parts:
1. weight initialization.
2. Selecting a weak classifier
3. calculating error.
4. Computing the weight of classifier
5. Recalculating weight vector

To prove the equality of the formula in slides –

$$F_{K(x)} = \sum_{k=1}^{K} a_k * f_k(x)$$

and the equation in paper –

$$h_f(x) = \begin{cases} 1 \quad if \sum_{t=1}^{T} \left(log\frac{1}{\beta_t}\right) * h_t(x) \geq 1/2 \sum_{t=1}^{T} log\frac{1}{\beta_t} \\ 0 \text{ otherwise} \end{cases}$$

I'll go through all five steps.

1. In weight initialization part, we can see that an additional process is done in paper – weight normalization.
2. Section two is the same for both methods.
3. Formula in the paper is:
$$\epsilon_t = \sum_{i=1}^{N} p_i^t * |h_t(x_i) - y_i|$$
While in slides, following formula is used:
$$\epsilon_K = \sum_{i=1}^{N} \omega_i L(f_K(x_i) \neq y_i\ )$$
In general, these two formula are not different. Both use weighted sum of error as their error function. We have:
$$\epsilon_t = \frac{\epsilon_K}{\sum_{i=1}^{N} \omega_i^t}$$
4. $a_K = 1/2log\left(\frac{\sum_i w_i}{\epsilon_K} - 1\right)$(slides)$= 1/2log\left(\frac{\sum_i w_i(1-\epsilon_t)}{\epsilon_t \sum_i w_i}\right) = 1/2log\left(\frac{1-\epsilon_t}{\epsilon}\right)$=1/2log($\frac{1}{\beta_t}$) (paper)
5. In this last step, from the equation in the paper:
$$\omega_i^{t+1} = \omega_{\mathrm{i}}^t \beta_t^{1-|h_t(x_i)-y_i|}$$

We can see that if the hypothesis is correct, the exponential part of β will become 1. Thus the weight of object i will decrease by multiplying a numeric number between 0 and 1. If the hypothesis is not correct, then the weight will remain unchanged. We can reformulate the equation as:

$$\omega_i^{t+1} = \begin{cases} \omega_i^t & if\, h_t(x_i) \neq y_i \\ \omega_i^t \beta_t & if\, h_t(x_i) = y_i \end{cases}$$

For equation in slides:

$$F_K(x) = exp\left(-y_{i\sum_{k=1}^{K} a_k} f_k(x_i)\right)$$

$$= \begin{cases} exp\left(-\sum_{k=1}^{K} a_k\right) \; if\; y_i = f_k(x_i) \\ \exp(\sum_{k=1}^{K} a_k) \qquad \text{otherwise} \end{cases}$$

$$= \begin{cases} exp(-\mathrm{x}) & if\; y_i = f_k(x_i) \\ \exp(\mathrm{x}) & \text{otherwise} \end{cases}$$

We can see that the prediction weight decreases when the hypothesis is well classified. When the hypothesis is wrong, the weight will increase. Although the two formula are not the same, their essence are similar – assign more weight on objects that are difficult to classify.

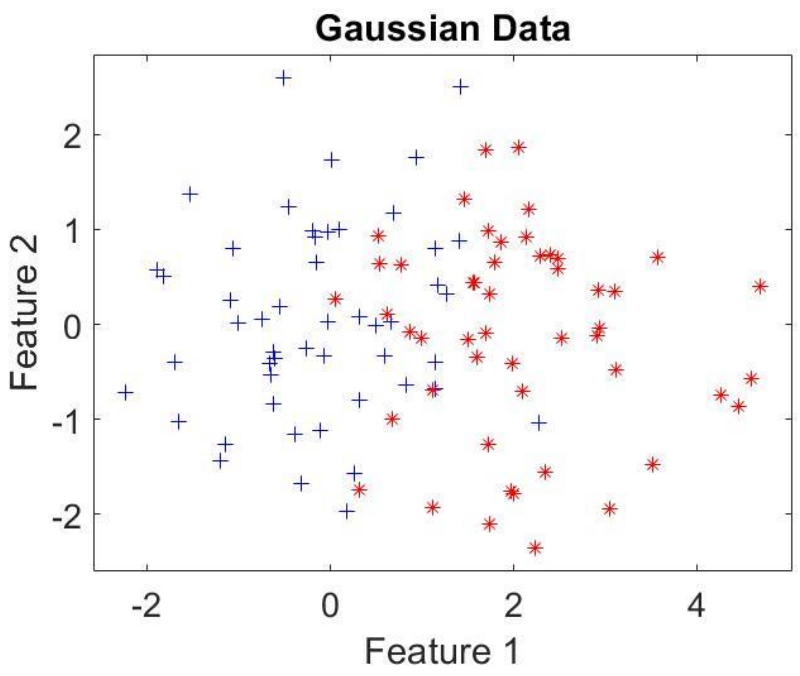
b. The decision stump is a simple learner. Its concept is using a threshold to classify objects. For example, if data is larger than threshold it will be assigned to +1 class and otherwise -1. Threshold is not known to the classifier at the beginning. Take 1D dataset as an example: using exhaustive search, for each iteration the predicted labels are generated using a hypothetical threshold. Then calculate the error rate between predicted label and true label. After all iterations, the threshold with least error rate will be chosen as optimal threshold. Also, the sign of label under same threshold should be taken into consideration (+1 if larger than threshold or -1 if larger than threshold). Based on above illustration, this function should take a labeled dataset as input and outputs the optimal feature f, threshold theta and classification method y. To deal with weight of different object, I use the weighted sum of error. MATLAB and Prtools are used in this experiment. Code is listed below.

```
function [f,t,y] = decisionStu(data,weight)
%decisionStu: weak learner using decision stump method.
%input: x,y: dataset with label
%output: f,t,y

%read data.
lab=getlab(data);
if(ischar(lab))
  lab=str2num(lab);
end
x=getdata(data);
```

```
weight=weight/sum(weight);
%r objects with c dimensions.
[r,c]=size(x);
minError=inf;
for i=1:c
   for j=1:r
      %exhausive search on theta
      theta=ones(r,1)*x(j,i);
      %prediction based on threshold,method2 is to classify
      %objects to class 2 if its smaller than threshold. method2
      %classifies objects smaller than threshold to class 1
      method1= x(:,i)<=theta;
      method2= x(:,i)>=theta;
      %error rate of method1 and method2
      err1=method1+1~=lab;
      err1=sum(weight'*err1);
      err2=method2+1~=lab;
      err2=sum(weight'*err2);
      %judge classification method
      if err1>err2
         err1=err2;
         sign=1;
      else
         sign=0;
      end
      %updata the optimal
      if err1<=minError
         y=sign;
         minError=err1;
         f=i;
         t=x(j);
      end
   end
end
disp(minError);
end
```

c.



·Figure 1: Scatterplot for two classes following guassian distributions

Two classes from two gaussian distributions are generated to test the decision stump, where the means are $u_1 = [0; 0] u_2 = [2; 0]$. Covariance matrices are identity matrices. For each class, 50 objects are generated.

The optimal obtained by my decision stump is:

F=1(means feature 1)

θ= 1.4713

y =1(means classifies objects smaller than threshold to class 1)

From the result above we can know that the optimal feature is feature 1. So, multiplying feature 2 by a factor 10 will not change the optimal parameters. This is proved by experiment. But rescaling the optimal feature will make θ rescale at the same pace. While the rest parameter will remain the same. I multiply feature 1 with 10, then the optimal features are

F=1(means feature 1)

θ= 14.713

y =1(means classifies objects smaller than threshold to class 1)

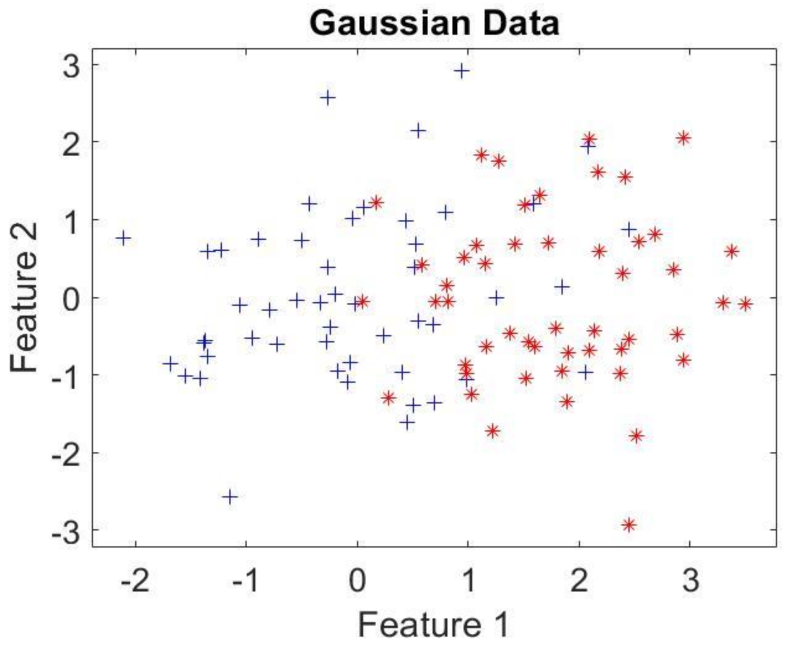
d. Using Fashion NIST dataset as train and test set, the decision stump algorithm evaluate apparent error and test error, which are 18.33% and 51.75% respectively.

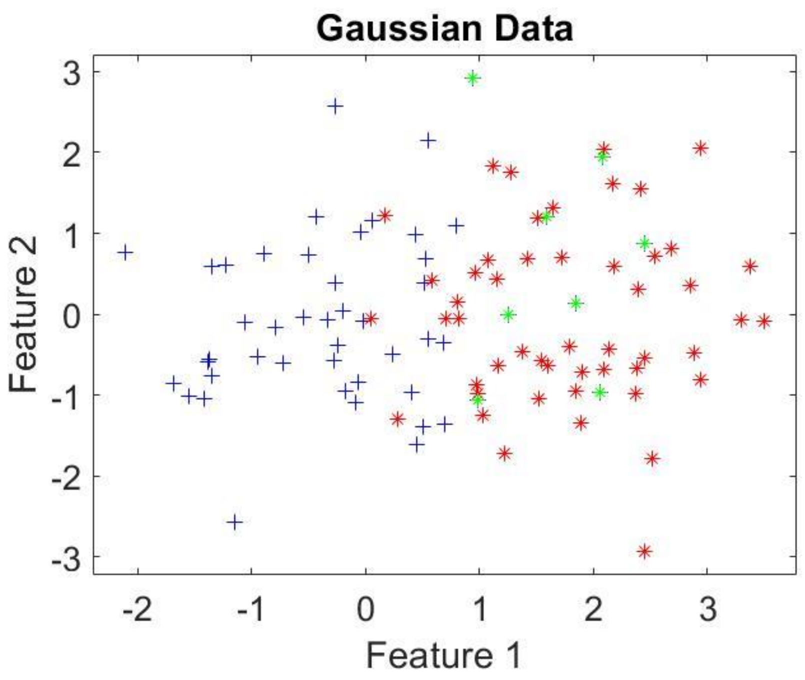
e. The coda of AdaBoost is below:

```
function [error, beta,H, ret] = adaBoost(dataset,D,T)
%adaBoost
%input: dataset: labeled data; D: distribution over dataset;
%T: number of iterations
data=getdata(dataset);
label=getlab(dataset);
[r c]=size(data);
weight=zeros(r,1);
```

```
%initialize weight
weight=D;
H=zeros(r,T);
%error for each iteration
error=zeros(T,1);
beta=zeros(T,1);
%returned values from weakLearner
ret=zeros(T,3);
for i=1:T
   p=weight/sum(weight);
   H(:,i)=p;
   ret(i,:)=decisionStu(dataset,p);
   [error(i) predict]=testerr(dataset,weight,ret(i,1),ret(i,2),ret(i,3));
   beta(i)=error(i)/(1-error(i));
   weight=weight.*(beta(i).^(1-abs(predict-label)));
end
end
```

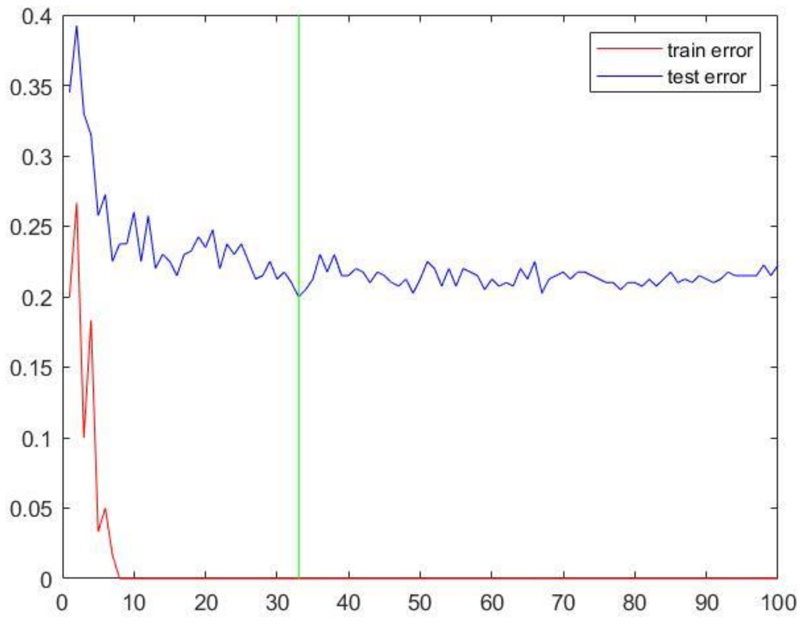
f. Using the simple dataset generated in c, I test the AdaBoost algorithm. The objects with largest weight in this dataset are points: 6 12 23 36 39 41 43 49 when iteration number equals 100. Below is the plot showing all the dataset with large weight. We can see that the points with large weight are blue points mixed in the group of red points. Experiments set other dataset show similar pattern. In conclusion, objects which are hard to classify will be assigned a higher weight. While small weight will be assigned to objects which are easy to classify.



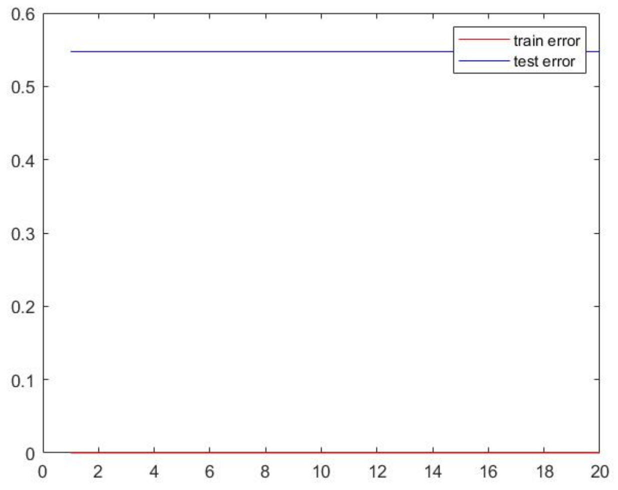


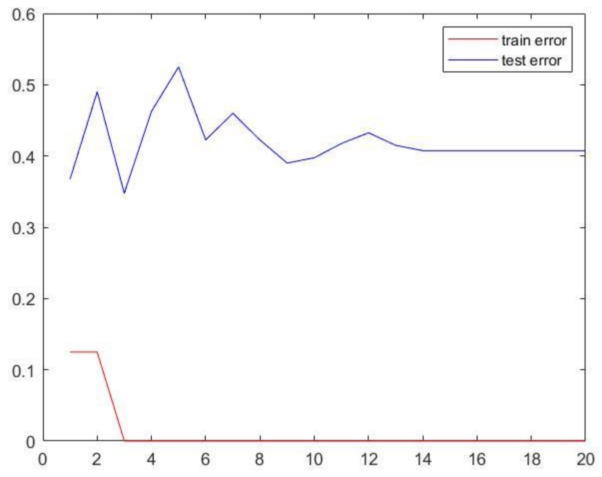
·points with largest weight are marked using green*.

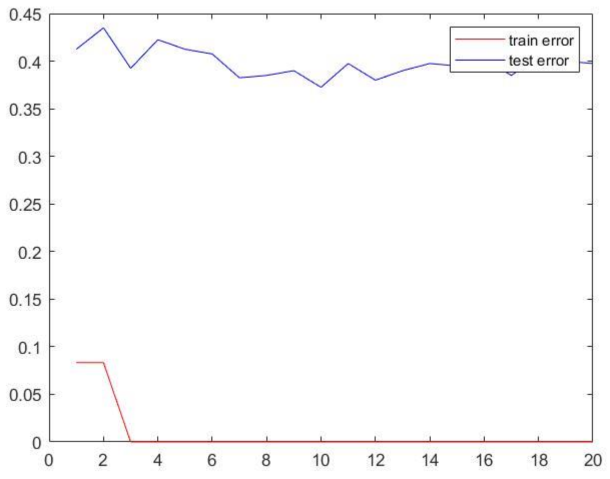
g. Using ***fashion57_train.txt*** as training dataset and ***fashion57_test.txt*** as test set, the apparent error and test error are 0 and 0.2225 respectively when iteration number is 100. To verify the relationship between error and iteration number, I plot following graph. It shows that train error and test error will fluctuate downwards as iteration number grows. However, it's noticeable that train error can drop to 0 while test error will variate around a certain level after several iterations. T=33 is the optimal T with which object 31 get the highest weight.

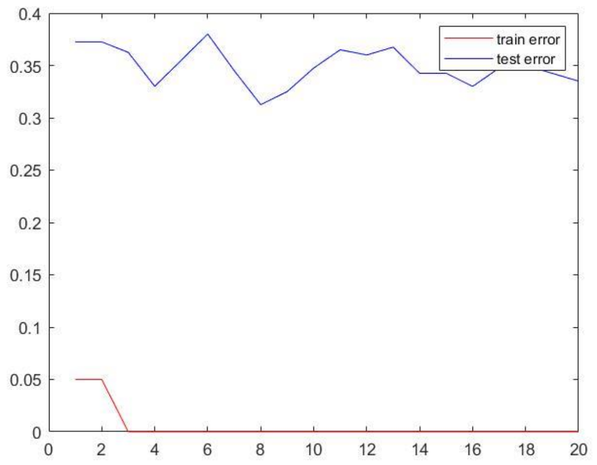


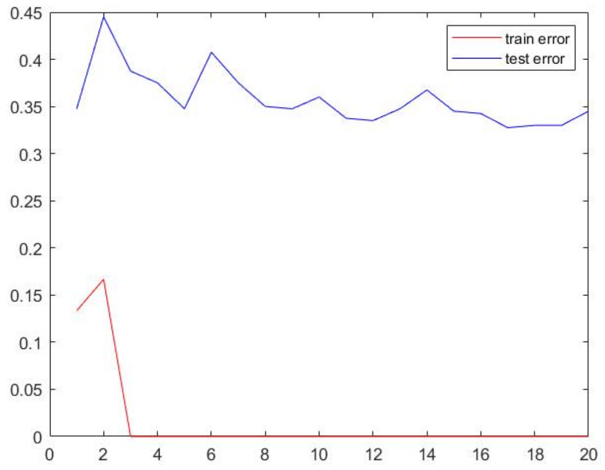
h. The plots below shows the learning curve for n=[2,4,6,10,15,20]. It can be seen that with the increase of n, the learning rate is also increasing. Besides, the optimal test error will be lower.

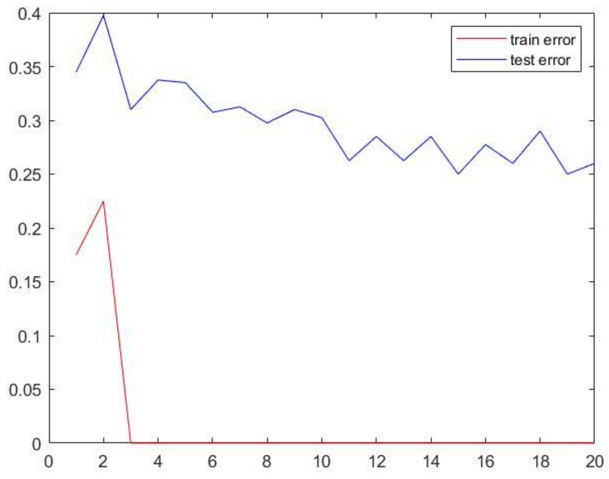












·From top to bottom are plots with n=2,4,6,10,15,20