STAT 542, homework 4

Shuhui Guo

Question 1

a)

Consider the Gaussian kernel function:

$$K(u) = 1/\sqrt{2\pi}exp(-u^2/2)$$

Let $u = |x - x_i|/\lambda$, the Nadaraya-Watson kernel regression estimator can be written as:

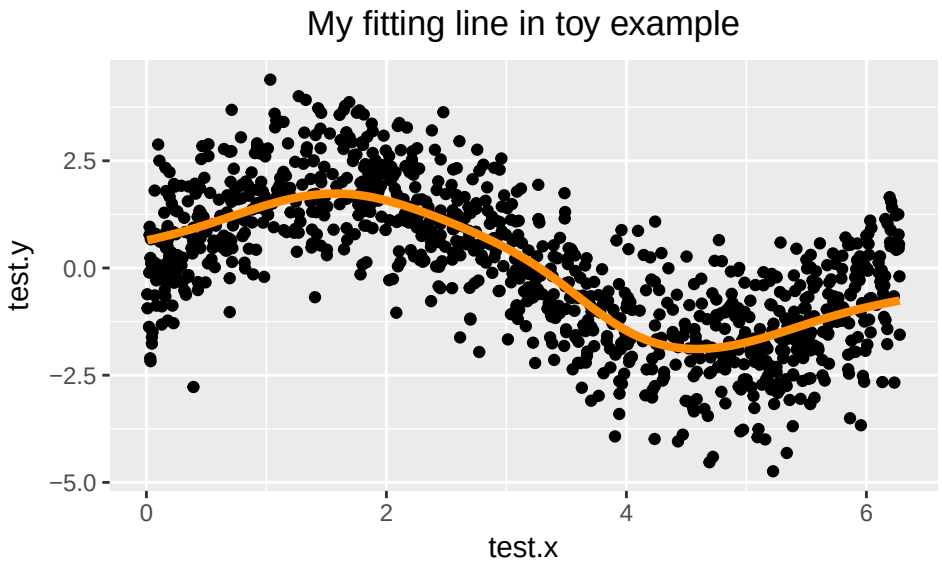
$$\hat{f(x)} = \frac{\sum_i K_\lambda(x, x_i)y_i}{\sum_i K_\lambda(x, x_i)} = \frac{\sum_i K(|x - x_i|/\lambda)y_i}{\sum_i K(|x - x_i|/\lambda)} = \frac{\sum_i exp[-(x - x_i)^2/(2\lambda^2)]y_i}{\sum_i exp[-(x - x_i)^2/(2\lambda^2)]}$$

I wrote my own code to fit the Nadaraya-Watson kernel regression estimator using Gaussian kernel for one dimensional problem. To apply my code, I generated a toy data example which contains the training data and testing data. In the training data, there are 1000 points of x randomly in $[0, 2\pi]$ and 1000 points of y generated by

$$y = 2sin(x) + \epsilon, \quad \epsilon \sim N(0, 1)$$

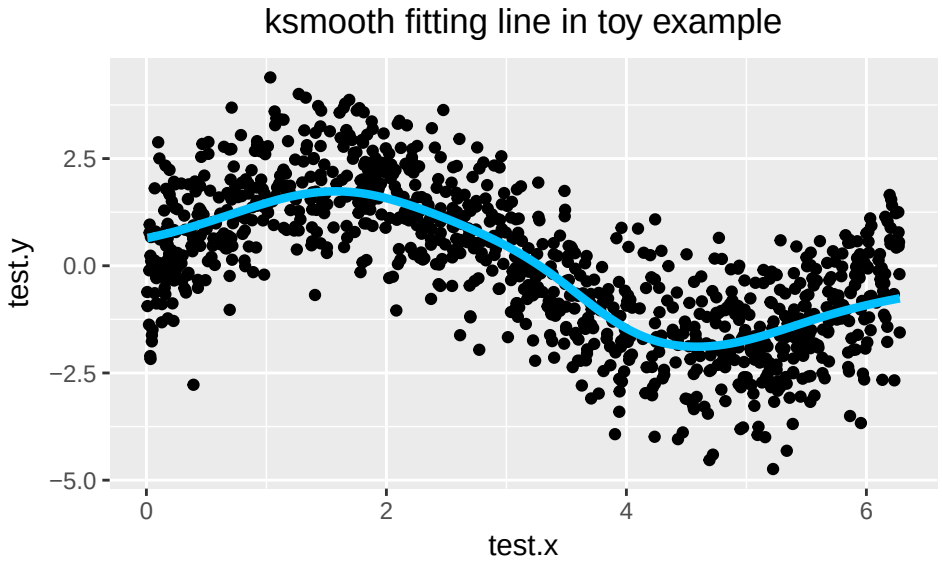
For the testing data, I regenerated a new dataset of x and y using the above methods.

Then I used my code to fit the Nadaraya-Watson kernel regression estimator for my toy data example. Set the bandwidth as 0.5, and the fitting line is plotted as below:

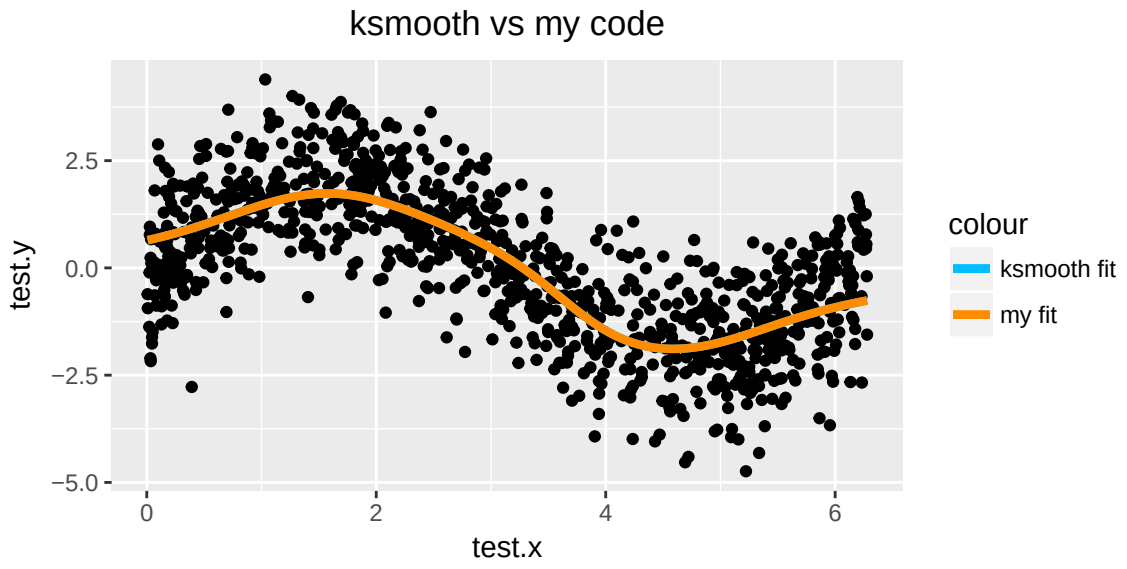


To validate my code, I used the ksmooth function to fit the Nadaraya-Watson kernel regression estimator for this toy data example. While using the ksmooth function, we noticed that in the source code of this function, the bandwidth is multiplied by 0.3706506. Therefore, to accurately validate my code, I divided the bandwidth by 0.3706506. So I set the parameter as 0.5/0.3706506 in ksmooth function.

The fitting line derived by ksmooth function is plotted as below:



Based on the two plots, we can see the fitting lines derived by my code and the ksmooth function are quite similar. To observe them more visually, I plot the two fitting lines in the same plot:

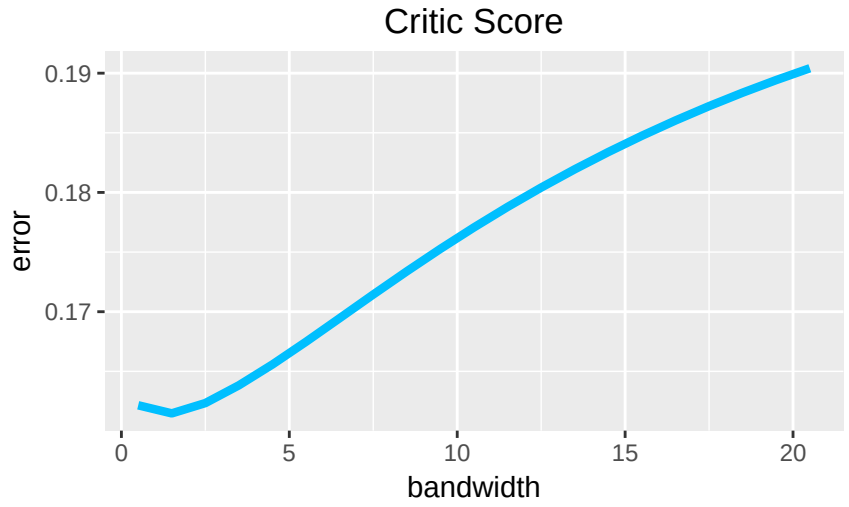


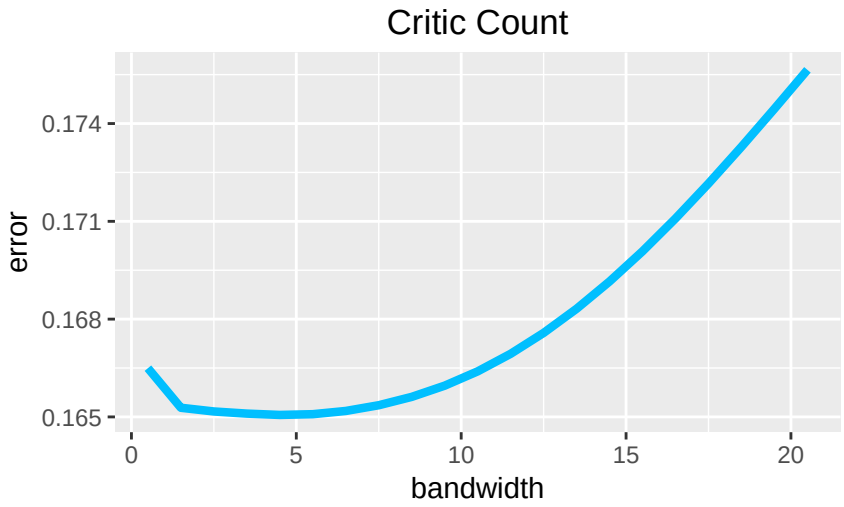
In the above plot, the darkorange line is the fitting line derived by my code. It covers the deepskyblue line, which is the fitting line derived by the ksmooth function. Therefore, we can say the two lines are the same, which validates my code is correct.

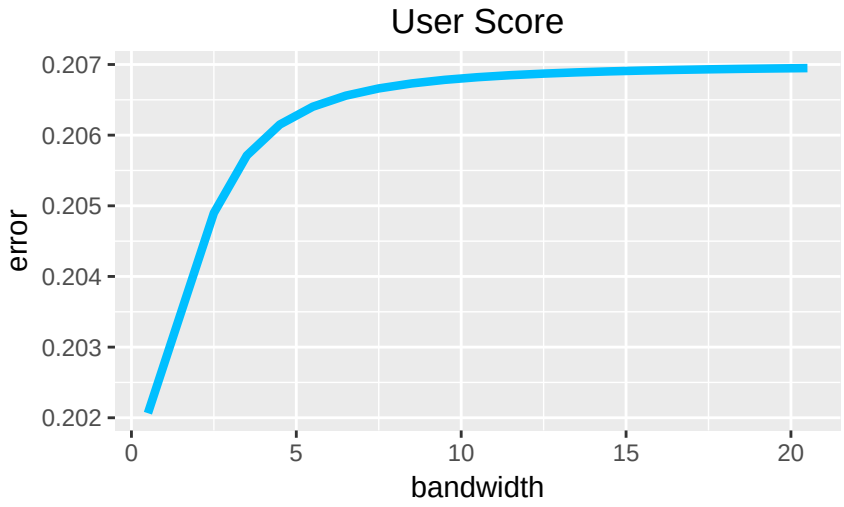
b)

Since there are many missing value and text value in the original dataset, I first did data cleaning and got a new dataset with 6947 observations of each variable to analyze. Then I shuffled the new dataset, and used my code to do 5-fold cross validation with different bandwidth.

To find the best bandwidth of each model, I first tried the bandwidth in a larger interval $[0.5, 20.5]$ by 1 and report the MSE to compare. The trends of MSE with changes of bandwidth are plotted as below:

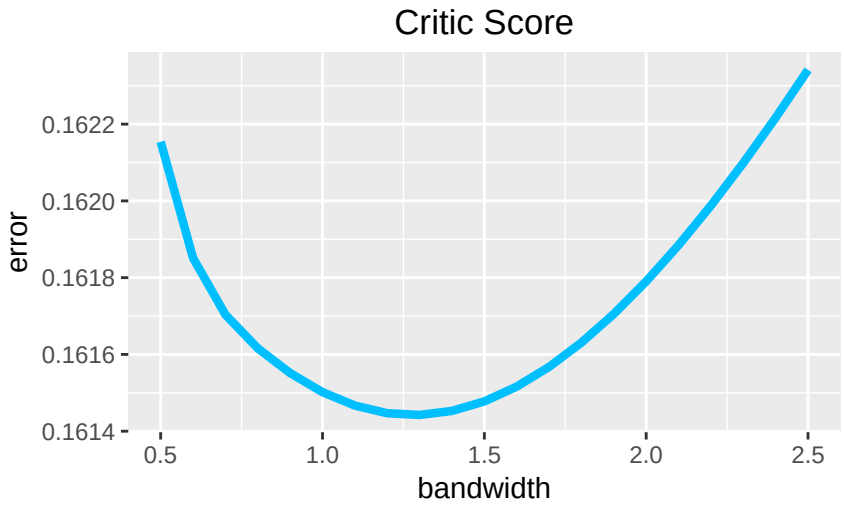


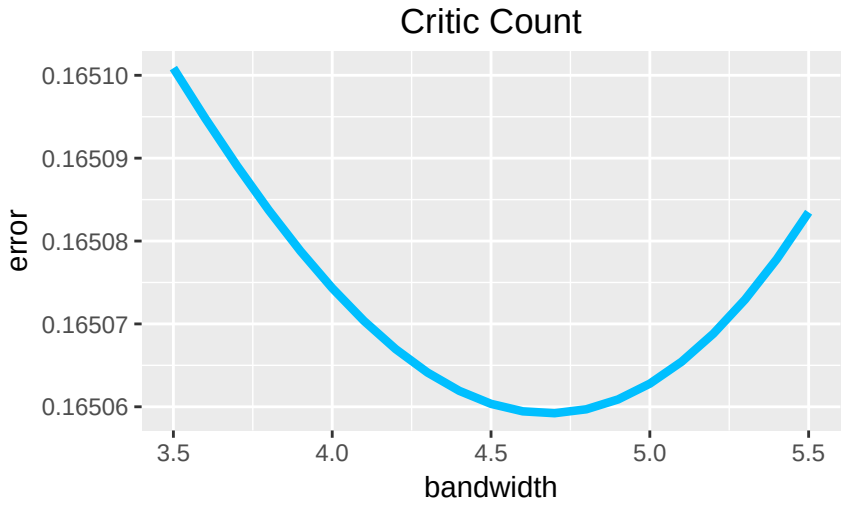


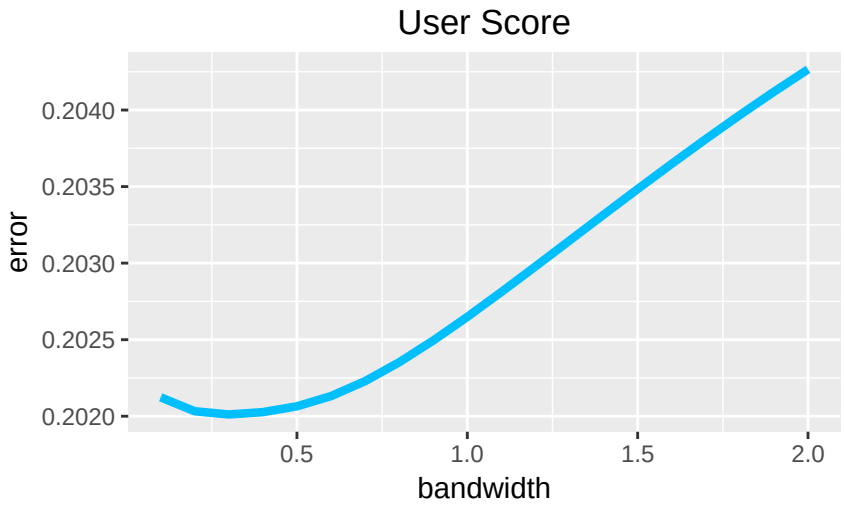


In each of the above plot, we can find that there is a minimum MSE in the larger interval. And the bandwidth with the minimum MSE can be roughly spotted. For Critic_Score, it is 1.5. For Critic_Count, it is 4.5. For User_Score, it is 0.5.

To get the best bandwidth exactly, I tried a smaller interval for each variable. For Critic_Score, the new smaller interval is $[0.5, 2.5]$ by 0.1. For Critic_Count, the new smaller interval is $[3.5, 5.5]$ by 0.1. For User_Score, the smaller interval is $[0.1, 2.0]$ by 0.1. The trends of MSE with changes of bandwidth are plotted as below:







In each of the above plot, we can find that there is a minimum MSE in the smaller interval. The bandwidth with the minimum MSE can be exactly spotted. For Critic_Score, the best

bandwidth is 1.3, the minimum MSE is 0.1614. For Critic_Count, the best bandwidth is 4.7, the minimum MSE is 0.1651. For User_Score, the best bandwidth is 0.3, the minimum MSE is 0.202.

Since the minumum MSE of Critic_Score is the smallest, Critic_Score gives the best model.

Question 2

a)

Generate X from independent standard normal distribution with $n = 200$ and $p = 20$. The true model is

$$f(X) = 1 + 0.5 \sum_{j=1}^{4} X_j + \epsilon, \quad \epsilon \sim N(0, 1)$$

Define the degrees of freedom of a fit as $\sum_{i=1}^{n} Cov(\hat{y}_i, y_i)/\sigma^2$. And choose $mtry = 1, 5, 15$, $nodesize = 5, 15, 25$ to estimate the degrees of freedom. To get better estimations, I performed this experiment for 20 times and calculated a sample covariance. Then I repeated the previous process 20 times to calculate the estimation and got the results below:

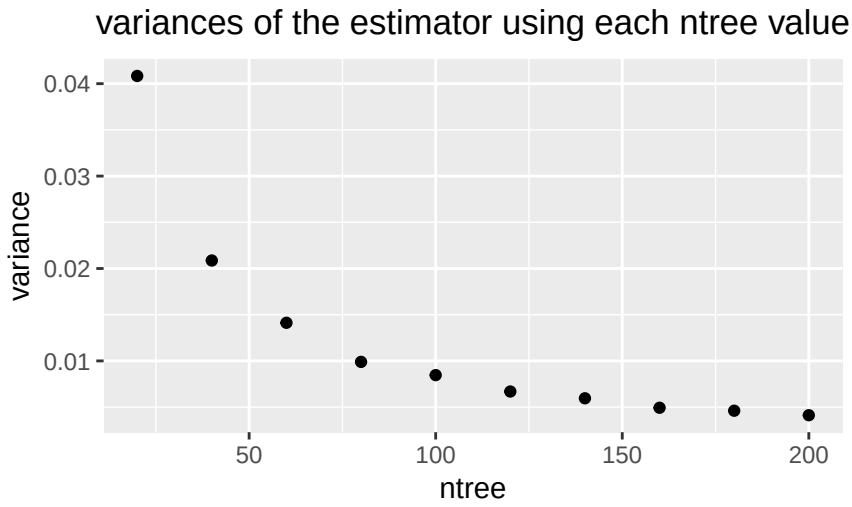
Table 1: degree of freedom

	nodesize=5	nodesize=15	nodesize=25
mtry=1	105.4777	56.1262	35.5031
mtry=5	118.4000	75.0626	48.9317
mtry=15	121.2106	82.3764	56.3597

Based on the results, we can find that the degree of freedom is correlated with mtry and nodesize. The degree of freedom increases as mtry increases and nodesize decreases. Mtry means the number of variables selected at each split. When mtry increases, there are more variables at each split and thus the degree of freedom increases. On the other hand, nodesize means the terminal node size where splitting stops. So when nodesize decreases, the degree of freedom increases. The above results make sense.

b)

In this question, I fixed mtry and nodesize as the default value and chose ntree from 20 to 200 by 20. After repeating the estimation process for 20 times, I calculated the variances of this estimator using each value of ntree. The results are plotted as below:



Based on the results, we can find that the variance is correlated with ntree. The variance decreases as ntree increases. Ntree means the number of trees builded to make estimation. When ntree increases, the estimation is made for more times, which increases the stability of the estimator and thus decreases the variance of the estimator. The above results make sense.

Question 3

a)

I wrote the function "stump" to fit the stump model with subject weights:

```
stump <- function(x, y, w){
  n <- length(x)
  xnew <- sort(x)
  ynew <- y[order(x)]
  wnew <- w[order(x)]

  score <- c()
  fleft <- c()
  fright <- c()
  for (i in 1:n){
    # left
    xleft <- xnew[1:i]
    yleft <- ynew[1:i]
    wleft <- wnew[1:i]
    pleft = sum(wleft[yleft == 1])/sum(wleft)
    Gleft = pleft*(1-pleft)

    # right
```

```
    xright <- xnew[(i+1):n]
    yright <- ynew[(i+1):n]
    wright <- wnew[(i+1):n]
    pright = sum(wright[yright ==1])/sum(wright)
    Gright = pright*(1-pright)

    # score
    score[i] = -(sum(wleft)/sum(w))*Gleft-(sum(wright)/sum(w))*Gright

    # predictions
    fleft[i] <- (pleft >= 0.5)-(pleft < 0.5)
    fright[i] <- (pright >= 0.5)-(pright < 0.5)
  }
  # find the output
  c <- xnew[which.max(score)]
  fl <- fleft[which.max(score)]
  fr <- fright[which.max(score)]

  return(c(c, fl, fr))
}
```

b)

Part I

Following the lecture slides page 6 in "Boosting.pdf", I first designed a model to train data and gave the outputs including α, node predictions f_L, f_R and the cutting point c. Then, I applied the outputs to fit the final model and give the output as:

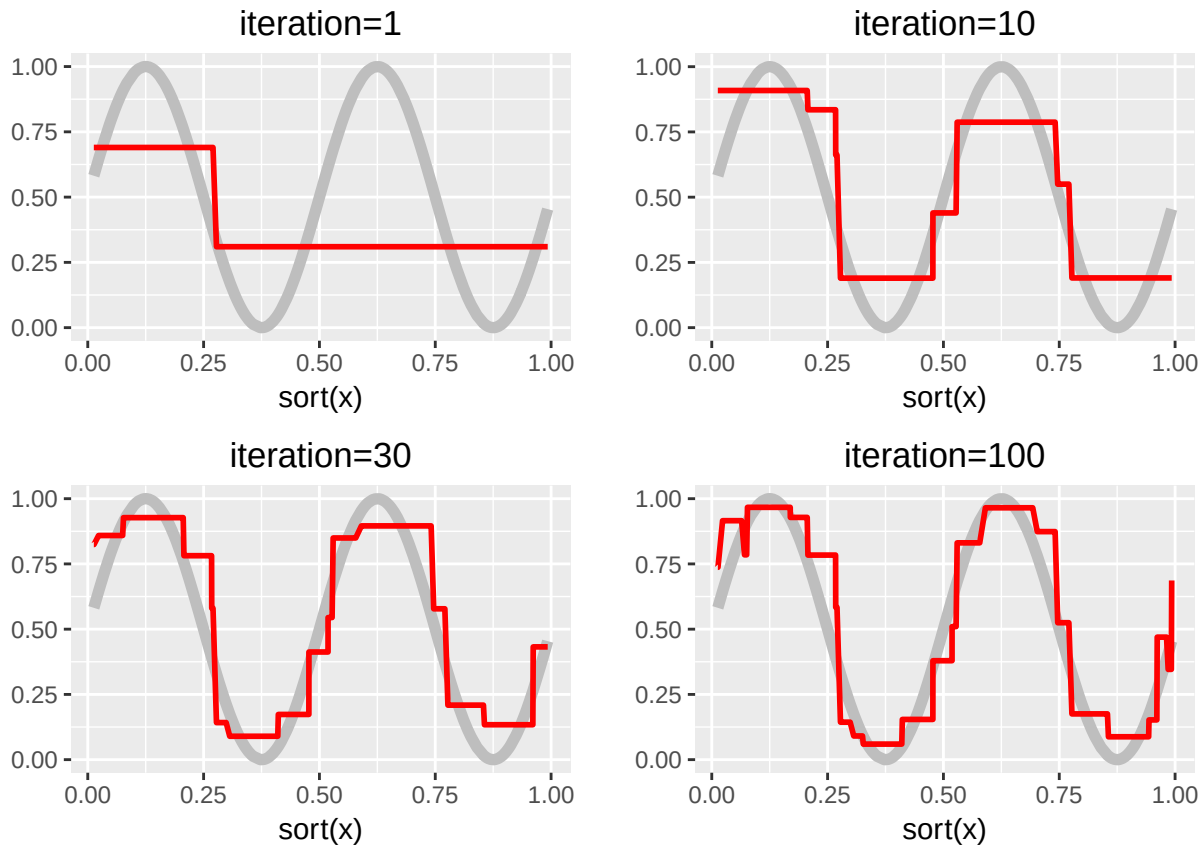
$$F_T(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$$

Since I would first use this output instead of $sign(F_T(x))$ to test my code.

Then I generated x and y by the given code. I chose the iteration as $1, 10, 30, 100$ and got the final models. The results are plotted below.

In these plots, the grey line is the fitting line of $(sin(4\pi x) + 1)/2$, where x is from the data I genereated above. The red line is the estimated probability given as:

$$P(y = 1|x) = \frac{e^{2F(x)}}{1 + e^{2F(x)}}$$



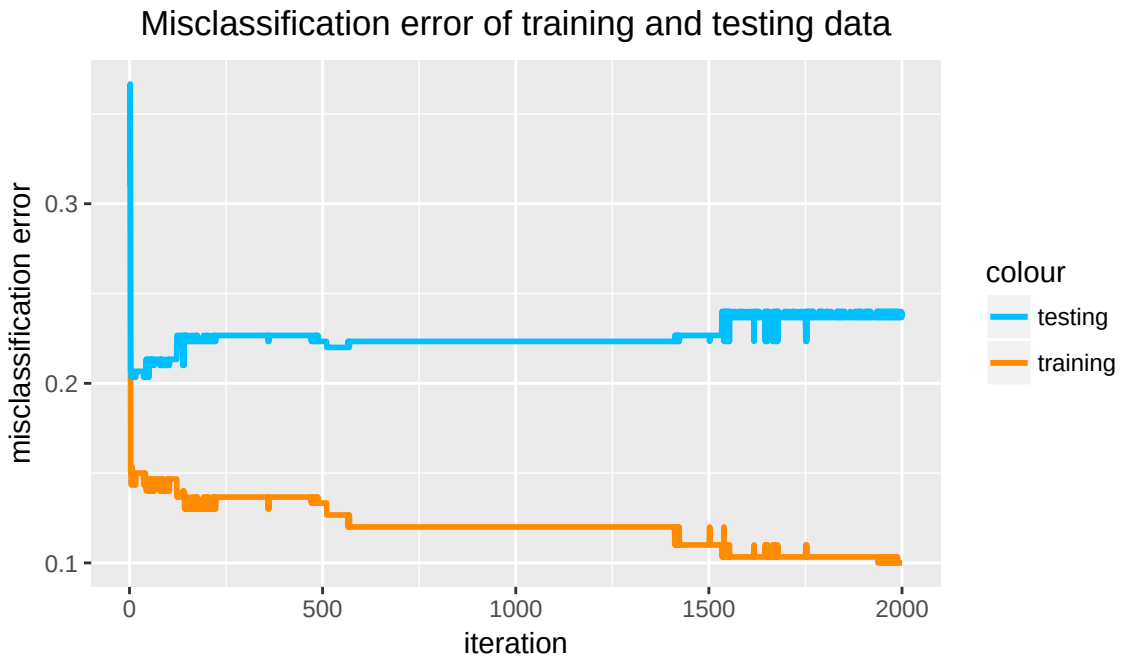
Based on the above plots, as the iteration increases, the red line can better fit the grey line, which validates that my code is correct.

Part II

In this part, I set the output as the classification rule:

$$sign(F_T(x))$$

Then I generated another dataset of x and y as testing data. Set the iteration as 2000, and calculate the misclassification error of training and testing data. The results are plotted as below:



Based on the above plot, during the 2000 iterations, the misclassification error of training data keeps decreasing. However, the misclassification error of testing data first decreases then begins to increase after about 100 iterations, which shows there is overfitting after certain value of iteration.

In conclusion, adaboost model can significantly decrease the misclassification error after a long iteration. But for testing error, the overfitting problem could appear after a certain value of iteration. Therefore, the model would be much more better if the iteration value had been tuned.