STAT542 Statistical Learning Homework 4

Huamin Zhang

Nov 14, 2017

Name: Huamin Zhang (huaminz2@illinois.edu)

Question 3

a) [15 points]

Answer:

```
# X: Observation(one dimension)
# Y: Response variable
# W: Weight of each observation
# Output: return a stump.model object with the following value.
# cut_point: The cutting point c of this stump model
# left_sign: Left node predictions(The prediction when x <= cut_point)
# right_sign: Right node predictions(The prediction when x > cut_point)
CART_stump<-function(X,Y,W){
  # Calculate the weighted reduction of Gini impurity
  # x: Observation(one dimension)
  # y: Response variable
  # w: Weight of each observation
  # cut_point: cut point we use in the model
  # Output:
  # score: the weighted reduction of Gini impurity
  # left_sign: Left node predictions
  # right_sign: Right node predictions
  cal_score<-function(x,y,weight,cut_point){
    # split data using cut_point
    left = (x <= cut_point); right = (x > cut_point)
    left_y = y[left]; right_y = y[right]
    left_weight = weight[left]; right_weight = weight[right]
    left_p = weighted.mean((left_y == 1),left_weight)
    right_p = weighted.mean((right_y == 1),right_weight)
    left_gini = left_p * (1-left_p)
    right_gini = right_p * (1-right_p)
    # Calculate score
    score = -(sum(left_weight) * left_gini)/sum(weight) -
      (sum(right_weight) * right_gini)/sum(weight)
    # Calculate the sign in each child node
    # If the number of +1 and -1 are the same, we define this prediction as -1
```

```
    left_sign = ifelse(sum(left_weight*left_y)>0,1,-1)
    right_sign = ifelse(sum(right_weight*right_y)>0,1,-1)
    return(list(score = score,left_sign = left_sign,right_sign = right_sign))
  }
  # Get the cut points sequence
  split_list = unique(X)
  result = matrix(NA,length(split_list),4)
  # Claculte the weighted reduction of Gini impurity of each cut point
  for(i in 1:length(split_list)){
    split_result = cal_score(X,Y,W,split_list[i])
    result[i,1] = split_list[i]; result[i,2] = split_result$score
    result[i,3] = split_result$left_sign; result[i,4] = split_result$right_sign
  }
  # Choose the point with the maxiumum score as the best cut point
  index = which.max(result[,2])
  result = list(cut_point = result[index,1],left_sign = result[index,3],
               right_sign = result[index,4])
  class(result) <- "stump.model"
  return(result)
}
```

Here we create a small sample data to test our function.

```
x <- c(1,2,3,4,5,6,7,8,9,10)
y <- c(1,-1,1,-1,-1,1,1,1,1,1)
w <- rep(1/length(x),length(x))
CART_stump(x,y,w)
```

```
## $cut_point
## [1] 5
##
## $left_sign
## [1] -1
##
## $right_sign
## [1] 1
##
## attr(,"class")
## [1] "stump.model"
```

According to the result, we think our code is correct.

b) [20 points]

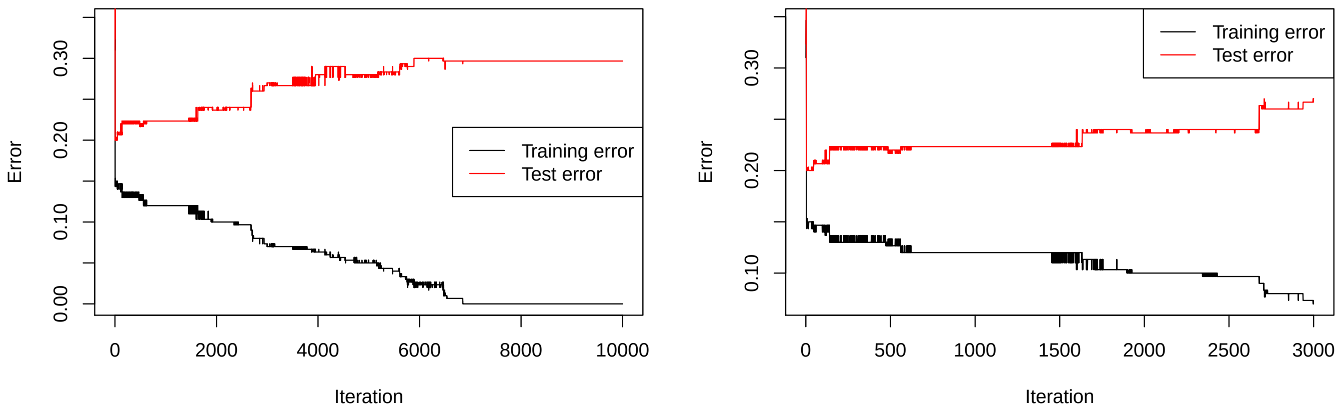
Answer:

```r
# To make prediction on data X with stump model
# X: Obsearvation data
# model: stump.model object, a stump model.
# Output: pred.y: The prediction value
stump.predict<-function(X,model){
  pred.y = rep(NA,length(X))
  pred.y[X <= model$cut_point] = model$left_sign
  pred.y[X > model$cut_point] = model$right_sign
  return(pred.y)
}

# Fit the adaboost model using the stump as the base learner.
# X: The observation data
# Y: The response variable
# iteration: The iteration of Adaboost algoright (the number of base learner)
# Output: A adaboost.stump.model object with the following value
# iteration: The number of base learner in the final adaboost model
# model: A list contain the base learners
# alpha: The weight of base learners in the adaboost model
# epsilon: The error of each base learner
Adaboost_stump<- function(X,Y,iteration = 500){
  weight = rep(1/length(X),length(X))
  epsilon = rep(NA,iteration)
  alpha = rep(NA,iteration)
  model = list()
  # Do the Adaboost
  for(i in 1:iteration){
    # Fit the base learner
    model[[i]] = CART_stump(X,Y,weight)
    pred.y = stump.predict(X,model[[i]])
    epsilon[i] = sum(weight * (Y != pred.y))
    # If error >= 0.5, reverse the model
    if(epsilon[i] >= 0.5){
      model[[i]]$left_sign = model[[i]]$left_sign * -1
      model[[i]]$right_sign = model[[i]]$right_sign * -1
      pred.y = stump.predict(X,model[[i]])
      epsilon[i] = sum(weight * (Y != pred.y))
    }
    # Calculate the alpha
    alpha[i] = 1/2 * log((1-epsilon[i])/max(epsilon[i],1e-10))
    # Update the weight
    w = weight * exp(-alpha[i] * Y * pred.y)
    weight = w / sum(w)
  }
  result = list(iteration = iteration, model = model, alpha = alpha, epsilon = epsilon)
  class(result) <- "adaboost.stump.model"
  return(result)
}
```

```r
# To make prediction on data X with adaboost model
# X: Obsearvation data
# Y: Response variable
# model: A adaboost.stump.model object, a adaboost model.
# Output:
# pred.y: The final prediction value
# pred.error: The error of the final prediction value
# error.list: The error sequence with the iteration increasing
Adaboost.stump.predict<-function(X,Y,model){
  pred.y = rep(0,length(Y))
  error_list = rep(NA,model$iteration)
  for(i in 1:model$iteration){
    yhat = stump.predict(X,model$model[[i]])
    pred.y = pred.y + yhat * model$alpha[i]
    predict = ifelse(pred.y > 0, 1, -1)
    error_list[i] = sum(predict != Y) / length(Y)
  }
  pred.y = ifelse(pred.y > 0, 1, -1)
  error = sum(pred.y != Y) / length(Y)
  return(list(pred.y = pred.y, pred.error = error, error.list = error_list))
}
```

Here we generate a sample data to test our code and the algorithm.

```r
# Generate the data
set.seed(0)
n = 300
x = runif(n)
y = (rbinom(n,1,(sin (4*pi*x)+1)/2)-0.5)*2
test.x = runif(n)
test.y = (rbinom(n,1,(sin (4*pi*test.x)+1)/2)-0.5)*2
w <- rep(1/length(x),length(x))

# Fit a Adaboost model with 10000 base learners.
adaboost.model = Adaboost_stump(x,y,10000)
train_predict = Adaboost.stump.predict(x,y,adaboost.model)
test_predict = Adaboost.stump.predict(test.x,test.y,adaboost.model)
par(mfrow=c(1,2))
plot(train_predict$error.list,type = 'l',xlab="Iteration",ylab = "Error")
lines(test_predict$error.list,col='red')
legend("right", c("Training error","Test error"), col = c("black","red"),
       cex = 1, lty = 1)
plot(train_predict$error.list[1:3000],type = 'l',xlab="Iteration",ylab = "Error")
lines(test_predict$error.list[1:3000],col='red')
legend("topright", c("Training error","Test error"), col = c("black","red"),
       cex = 1, lty = 1)
```



From the left plot, we can see the training error tends to decrease with the increasing of iteration, and if the iteration is large enough, the training error tends to zero. It validates that the training error of AdaBoost decreases the upper bound exponentially. According to the result, we think the code is correct. Moreover, in the left plot we find that with the iteration increasing, the training error decreases, but the test error decreases first and then we observe an increasing trend, that means the model is not improving anymore and it is overfitting.

And in the right plot, we foucus on the iteration between 1 to 2000, and we think the testing error start to go up already after just a few hundred iterations. According to the result, we can say that the Adaboost algorithm will cause overfitting and it is important to choose a reasonable iteration number.