Eda Gür

50488

Comp 421 HW5 Report

I started by dividing the data to training and test with code snippets such as as.matrix(data\_set$x[1:100]) for X\_train and as.matrix(data\_set$x[-1:100]) for the X\_test.

After that I wrote a decision tree function. This function receives a parameter called P, which is the prepruning factor. It finds the node\_splits, node\_means and is\_terminal values for nodes.

In the DecisionTree function, I start by declaring node\_indices, is\_terminal, need\_split, node\_splits, node\_means. If a node is terminal, we don’t need to split any further. We find the nodes that we will be splitting with the need\_split list and assign them to split nodes. Node\_splits is the resulting nodes from a split and node\_means will give us the prediction values. Node\_indices is just the indices of the training points. Now that I have defined the main points, I can move onto the algorithm. If there are split nodes available, we might want to divide them further. If length of split nodes is 0, we can break out of the while loop. Similarly, we preprune. So, if we come across a node that has <= P training points, we declare it as a terminal point and give the mean value of its data points as its regression value. We do the same, if all the data points are the same in a given node (length(unique\_values) == 1), because we don’t need to split a pure node any further. For each split node, we want to find the best position to split. So, we evaluate each split position and give them a score:

right\_score = sum((y\_train[left\_indices] - mean(y\_train[left\_indices], na.rm = TRUE)) ^ 2)

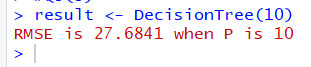
left\_score = sum((y\_train[right\_indices] - mean(y\_train[right\_indices], na.rm = TRUE)) ^ 2)

split\_scores[s] <- (right\_score + left\_score) / length(data\_indices)

We pick the split position with the minimum score (lowest impurity) and assign it to the node\_split of the split\_node. After finding out where to split, we can create the left node and right node of the split. For this, we use a data structure s.t. left\_node = 2\*original\_node, right\_node = 2\*original\_node+1. We assign the is\_terminal, node\_indices, and need\_split values accordingly.

After learning these values from the training data, we are ready to find the predictions for the test data by traversing the nodes for each prediction. The code for this part is the same as the lab. Now, I have a y\_predicted list. I can find the RMSE value by assigning sqrt(mean((y\_test - y\_predicted) ^ 2)) to it. I output it to the console with the message(sprint(…)) functions. Finally, this function returns a list of four consisting of node\_splits, node\_means, is\_terminal lists and the rmse value.

My function correctly gives the RMSE for P=10:



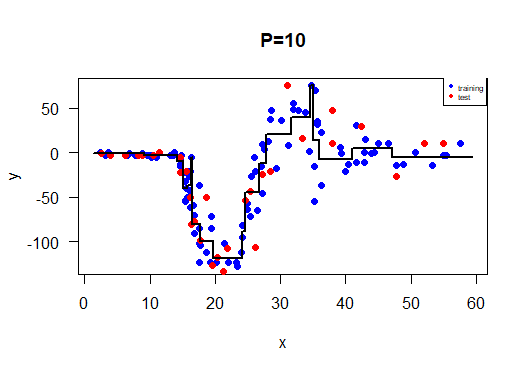
Now, it is time to draw my graph. I call my function DecisionTree(10) and I get a result which is a list of four values. I retrieve the three values I will need by doing:

is\_terminal = unlist(result[3])

node\_means = unlist(result[2])

node\_splits = unlist(result[1])

After this, I define a data interval that spans roughly from the minimum x point to the maximum x point. For each interval in this data interval, I have a line which needs four argument: two x values and two y values. I find the x values with line\_x <- c(data\_interval[b], data\_interval[b+1]) and for the y values, I make predictions similar to the predictions I made for the test points in the decision tree function. Now I can draw the lines. I draw the horizontal lines with lines(c(line\_x[1], line\_x[2]), c(line\_y\_predicted[1], line\_y\_predicted[1]), lwd = 2, col = "black") and the horizontal lines with lines(c(line\_x[2], line\_x[2]), c(line\_y\_predicted[1], line\_y\_predicted[2]), lwd = 2, col = "black"). My resulting graph is below:



Now, it is time to find the RMSE values for each P value from 1 to 20. For this, I simply call the DecisionTree function for 20 times with the lapply function and unlist the fourth value from the returned list of the DecisionTree function. I give the resulting vector as the y argument to the plot function. Below is the resulting graph:

