CART Coding

Qi Wang

2022/7/18

Here are some coding tricks I learned from the code in this website https://github.com/ebprado/MOTR-BART/blob/master/MOTR%20BART%20paper/MOTR_BART.R .

1."vector" Function

This function is useful when using vector('list', length = 2). This will create a list like vector but we don't need the "$" to subtract the elements in this vector. Instead, we use "[[x]]", this will become more convenient to store some values especially when multiple indexes of an observation is stored. For example:

```
a <- vector('list', 3)
a[[1]] <- c(1,1,2,3,4)
a[[2]] <- matrix(c(4,4,5,6),2,2)
a[[3]] <- array( c(7,7,8,9,0,1), c(2,2,2))
```

This is actually like a vector of all kinds of things including matrices or arrays. When I want to get the first part of information, I may use:

```
a[[1]]
```

```
## [1] 1 1 2 3 4
```

When I want to get the second part of information, I may use:

```
a[[2]]
```

```
##      [,1] [,2]
## [1,]    4    5
## [2,]    4    6
```

However, when I want to get the information of the matrix which is the second element of the whole vector, I will use this:

```
a[[2]][1,1]
```

```
## [1] 4
```

```r
a[[2]][1,]
```

```
## [1] 4 5
```

This is how "vector" function works in this code, which is a very important function.

2.Create a function that stores stump information for each Tree

This step is to keep track of whether this node is a terminal node, it is just a stump model so it only store information for one node. Here, we create a information box for all trees.

```r
num_trees <- 5
x1 <- rnorm(1000, mean = 9, sd = 3)
x2 <- rnorm(1000, mean = 30, sd = 10)
y <- rnorm(1000, mean = 2*x1 + 3*x2, sd = 1)
x <- cbind(x1, x2)
```

```r
create_stump <- function(x, y, num_trees = 1){

all_tree <- vector("list", length = num_trees)

# Make each tree information stored in all_tree[[i]]
# This is like creating a box, but it's empty for further storage

for (i in 1:num_trees) {

  # Make each element has two indexes
  all_tree[[i]] <- vector("list",2)
  names(all_tree[[i]]) <- c("info", "node")

  # Create tree information storage
  all_tree[[i]][[1]] <- matrix(NA, ncol = 8, nrow = 1)

  # Create node indices storage
  all_tree[[i]][[2]] <- rep(1, length(y))

  # Create column names for information matrix
  colnames(all_tree[[i]][[1]]) <- c("terminal","child_L","child_R","parent","split_variable","split_valu

  all_tree[[i]][[1]][1,] <- c(1, rep(NA, 5), 0, length(y))



}

return(all_tree)

}
```

3. Create a function to update trees in the M-H step.

3.1 "Switch" Function

This function is like choosing different directions of the result. For example,

```
switch (1,"A","B","C")
```

```
## [1] "A"
```

```
switch (2,"A","B","C")
```

```
## [1] "B"
```

```
switch (3,"A","B","C")
```

```
## [1] "C"
```

However, in this code, the first argument can also be a string, for example,

```
switch ("A", "A" = 4, "B" = 5, "C" = "Hello World")
```

```
## [1] 4
```

```
switch ("B", "A" = 4, "B" = 5, "C" = "Hello World")
```

```
## [1] 5
```

```
switch ("C", "A" = 4, "B" = 5, "C" = "Hello World")
```

```
## [1] "Hello World"
```

3.2 Updating Tree

Therefore, we can simply use this function to create a tree update function, which depends on the method of the decision. If the method of updating is "grow", then we will grow the tree, and so on. So the update function should have the shape like this:

```
update_tree <- function(x, y, type = c("grow", # For growing the existing tree
                                       "prune", # For merging one pair of the terminal node
                                       "change", # Change the splitting variable and splitting rule
                                       "swap", # Swap the splitting rules for two pairs of terminal nod
                                       ), curr_tree # The current sets of trees
                                        , node_min_size # The minimum size of a node to grow, avoid the

                        ){
```

```r
  new_tree <- switch (type, grow = grow_tree(x, y, curr_tree, node_min_size),
                            prune = prune_tree(x, y, curr_tree),
                            change = change_tree(x, y, curr_tree, node_min_size),
                            swap = swap_tree(x, y, curr_tree, node_min_size))
  return(new_tree)
                                      }
```

Then we have to figure our how to grow, prune, change or swap a tree.

4. Grow Tree Function

4.1 Quick Notes(R trick)

1. We can quickly get the elements corresponding to some categories by using the "==", just a[xx == yy] if a is a vector. No need to use a[which[xx == yy]].

```r
c(1,2,3,4,5,6)[c(T,F,T,F,T,F)]
```

```
## [1] 1 3 5
```

2. We can quickly pick out one column with the column name if the matrix has a column name.

```r
A <- matrix(c(1,2,3,4), 2,2, byrow = TRUE)
colnames(A) <- c("A", "B")
A
```

```
##      A B
## [1,] 1 2
## [2,] 3 4
```

```r
A[,"A"]
```

```
## [1] 1 3
```

```r
A[,"B"]
```

```
## [1] 2 4
```

3. We can use "any" function to judge whether there exists something that satisfies our expectation, for example,

```r
if(any(c(1,2,3)>1)){print("Exist one value bigger than 1")}
```

```
## [1] "Exist one value bigger than 1"
```

```
if(any(c(1,2,3)<=1)){print("Exist one value smaller than or equal to 1")}
```

```
## [1] "Exist one value smaller than or equal to 1"
```

4.2 Grow Tree Function

We first get the available terminal nodes list which can grow, then get the available rules that we can use, then select the spliting variables and rules uniformly.

```
grow_tree <- function(x, y, curr_tree, node_min_size){

  # Give new tree information box
  new_tree <- curr_tree

  # Get the terminal nodes list
  terminal_node <- which(as.numeric(new_tree$info[,"terminal"]) == 1)

  # Get the terminal node size
  terminal_node_size <- new_tree$info[terminal_node,"node_size"]

  # Initialize
  available_values <- NULL
  max_bad_trees <- 10
  count_bad_trees <- 0
  bad_trees = TRUE

  # If it's a bad_tree, for example, the size of one new terminal node is too small, then we need to re

  while (bad_trees) {

    # Information box for new tree
    new_tree <- curr_tree

    # Add two extra rows to the tree information, because we are now having more nodes of the tree.
    new_tree$info <- rbind(new_tree$info, c(1, rep(NA, 7)), c(1, rep(NA, 7)))

    # Choose a random terminal to split, if it has been smaller than the minimum size of our requiremen
    split_node <- sample(terminal_node, 1, prob = as.integer(as.numeric(terminal_node_size))> node_min_

    # Choose a variable to split
    split_var <- sample(1:ncol(x),1)

    # The next step guaranteed that we are having a nice splitting value.
    # Available values are the range of the selected variables belonging to this node. Noting that node
    available_values <- sort(unique(x[new_tree$node == split_node, split_var]))

    # Select an available value to split

    if (length(available_values)==1){
      split_val <- available_values[1]
    }else if(length(available_values)==2){
        split_val <- available_values[2]
```

```
    }else{
      # We cannot select the biggest or smallest value as the splitting rule
        split_val <- gdata::resample(available_values[-c(1, length(available_values))],1)
    }
       # Make sure the current parent exist there. If it's the root node, then it's NA.
      curr_parent <- new_tree$info[split_node, "parent"]
      new_tree$info[split_node, 1:6] <- c(0, # Because now it's not terminal node
                                          nrow(new_tree$info)-1, # The second last row is the left chil
                                          nrow(new_tree$info), # The last row is the right child
                                          curr_parent, # Record the current parent
                                          split_var, split_val)


      # Fill the parent of these two child nodes
      new_tree$info[nrow(new_tree$info)-1, "parent"] <- split_node
      new_tree$info[nrow(new_tree$info), "parent"] <- split_node

      #Fill the details including updating the node indices
      new_tree <- fill_tree_details(new_tree, x)

      # Check for bad trees, if it's a bad tree, then we cannot use this.
      if(any(new_tree$info[,"node_size"]<= node_min_size)){

        # Count how many bad trees that we have generated
        count_bad_trees = count_bad_trees + 1
      }else{
        bad_trees = FALSE
      }

      # If too many bad trees are generated, we return the current tree, which means now the trees has
      if(count_bad_trees == max_bad_trees){return(curr_tree)}


}
        return(new_tree)
}
```

5. Fill Detailed Tree Function

Even if we have updated the tree information, we haven't updated the nodes that the observation belongs to, i.e., new_tree$node. In this section, we create a function to fill the tree details. Also, we update the node size.

```
fill_tree_details <- function(curr_tree, x){

  # Collect old information from the tree
  tree_info <- curr_tree$info

  # Create a new matrix to overwrite the information
  new_tree_info <- tree_info

  # Start with default node
```

```
  node <- rep(1, nrow(x))

  # Get the number of observations falling in each node.
  # But we don't need the first node, because it's the root so all the observations will fall into the

    for (i in 2:nrow(tree_info)) {

    #Get the parent
    curr_parent <- as.numeric(tree_info[i,"parent"])

    #Get the splitting variable and splitting value
    split_var <- as.numeric(tree_info[curr_parent, "split_variable"])
    split_val <- as.numeric(tree_info[curr_parent, "split_value"])
    direction <- ifelse(tree_info[curr_parent, "child_L"] == i, "L", "R")
    if(direction == "L"){
      # if the direction is the left, then it should use "smaller than" to confirm that this observatio
      # x[node == curr_parent,] selects the observations who belongs to the parent node.
      # Be careful that the "node" updated every time in the loop!
      # The node now haven't been updated, so it describes the category that this observation belongs t
      #So it's reasonable to say node == curr_parent because we need to filter among all the nodes belo
      new_tree_info[i, "node_size"] <- sum(x[node == curr_parent, split_var] < split_val)
      # This step we update the node indices for the i th row. So we now only update those who belongs
      node[node == curr_parent][x[node == curr_parent, split_var]<split_val] <- i
    }else{

      #Same as left, but change the inequality direction
      new_tree_info[i, "node_size"] <- sum(x[node == curr_parent, split_var] >= split_val)
      node[node == curr_parent][x[node == curr_parent, split_var] >= split_val] <- i

    }

  }

  return(list(info = new_tree_info, node = node))


}
```

Testing for Code from part 1-5

```
tree <- create_stump(x,y)[[1]]


tree_up_1 <- update_tree(x,y,type = "grow", curr_tree = tree, node_min_size = 10)

tree_up_2 <- update_tree(x,y,type =  "grow", curr_tree = tree_up_1, node_min_size = 10)

tree_up_3 <- update_tree(x,y, type = "grow", curr_tree = tree_up_2, node_min_size = 10)

print(tree)
```

```
## $info
```

```
##      terminal child_L child_R parent split_variable split_value mu node_size
## [1,]        1      NA      NA     NA             NA          NA  0      1000
##
## $node
##    [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##   [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##   [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [334] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [371] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [408] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [445] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [482] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [519] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [556] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [593] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [630] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [667] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [704] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [741] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [778] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [815] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [852] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [889] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [926] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [963] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [1000] 1
```

```
print(tree_up_1)
```

```
## $info
##      terminal child_L child_R parent split_variable split_value mu node_size
## [1,]        0       2       3     NA              1    8.383972  0      1000
## [2,]        1      NA      NA      1             NA          NA NA       414
## [3,]        1      NA      NA      1             NA          NA NA       586
##
## $node
##    [1] 3 3 2 3 3 3 2 3 2 2 2 3 3 2 2 3 2 2 3 3 3 2 2 2 2 3 2 3 3 2 2 2 3 2 2 2 3
##   [38] 3 2 2 2 3 3 3 3 3 3 3 3 3 3 2 2 3 2 3 2 3 3 3 2 2 3 3 3 3 3 3 2 3 2 2 2 3
##   [75] 2 2 3 2 2 3 2 2 3 3 3 3 2 3 3 2 2 2 3 3 3 3 3 2 3 2 3 3 3 2 3 2 3 2 2 3 2
##  [112] 2 3 3 2 2 3 2 3 2 3 3 3 3 2 3 2 3 3 2 3 3 3 3 3 3 3 2 2 2 2 3 2 2 2 3 3 2
##  [149] 3 2 3 2 3 2 3 2 3 3 3 2 2 2 2 3 2 3 2 3 3 2 3 2 2 2 3 3 3 2 3 2 2 3 3 3 3
##  [186] 3 2 2 3 3 2 2 2 3 3 3 3 3 2 3 3 2 2 2 3 2 2 3 3 2 2 3 3 3 3 2 3 3 2 3 3 2
##  [223] 2 2 2 3 3 3 3 2 3 3 3 2 3 2 2 2 3 3 3 2 3 2 2 2 2 3 2 3 3 2 3 3 2 2 2 3 3
##  [260] 2 2 3 3 3 2 3 3 2 3 2 3 3 2 3 3 3 3 3 3 3 2 3 2 2 3 2 3 3 3 3 2 2 3 3 2 2
##  [297] 3 3 3 2 2 3 2 2 3 3 3 2 2 3 2 2 2 3 3 3 2 3 2 3 2 3 3 3 2 2 3 3 2 2 3 3 2
##  [334] 3 2 3 2 2 2 2 3 3 2 3 3 2 3 3 2 2 3 3 3 3 2 3 3 2 3 3 2 3 3 3 2 3 3 3 3 2 3
##  [371] 3 3 3 3 3 3 3 3 3 2 3 3 2 2 2 2 3 3 3 2 2 3 2 3 2 3 3 3 2 2 3 3 3 2 2 3 2
##  [408] 2 2 3 3 2 2 3 3 2 3 3 3 2 2 2 2 3 3 3 3 3 3 3 3 2 2 3 2 2 3 3 3 2 3 2 2 3
```

```
##  [445] 3 3 3 2 3 3 3 2 3 3 3 3 3 3 3 3 2 3 3 2 2 2 3 3 2 2 2 3 2 3 2 3 2 2 3 2 3
##  [482] 3 3 3 3 2 3 3 3 3 2 2 2 3 3 3 2 3 2 3 3 2 2 3 2 2 3 3 3 2 3 3 3 2 3 3 3 3
##  [519] 2 3 3 3 3 3 2 2 3 2 2 2 2 3 2 3 3 3 3 2 2 2 2 2 3 3 3 2 2 2 3 3 2 2 3 3 2
##  [556] 3 3 3 2 3 3 3 3 2 2 2 2 2 3 2 3 3 2 2 3 2 2 2 3 3 2 2 3 2 3 3 2 3 2 3 2 3
##  [593] 2 2 3 2 2 3 3 2 2 3 3 3 3 2 3 3 2 3 3 2 3 2 3 3 2 3 3 3 3 2 3 3 3 3 2 3 3
##  [630] 2 2 2 3 3 2 3 3 2 2 3 3 3 3 2 2 3 3 3 3 2 3 3 2 2 3 2 2 3 2 2 2 3 3 2 3 3
##  [667] 2 3 3 3 2 2 2 2 2 3 3 2 3 2 3 3 3 2 3 2 3 3 2 3 3 2 3 3 2 2 2 3 3 2 2 3 3
##  [704] 3 2 2 2 2 3 3 2 3 3 3 3 3 3 3 2 2 2 3 3 2 3 3 2 2 3 2 3 2 3 2 2 2 2 2 2 3
##  [741] 3 3 2 2 3 2 2 3 3 3 3 2 2 3 3 3 2 2 3 3 3 3 2 3 3 3 3 2 3 3 2 3 2 3 3 3 3
##  [778] 3 2 3 3 3 2 3 2 3 3 3 3 2 3 3 3 2 3 2 2 2 3 3 3 3 3 3 3 3 3 2 2 3 3 3 2 3
##  [815] 2 3 3 3 3 3 3 3 2 2 3 2 2 2 2 2 3 3 2 2 3 2 3 3 3 3 2 3 3 3 3 3 3 3 3 3 2
##  [852] 2 3 2 3 3 3 3 3 3 3 3 2 3 3 2 3 3 3 3 3 3 3 3 2 2 3 2 3 3 3 2 2 3 2 3 3 3
##  [889] 2 3 3 2 2 3 3 3 3 3 3 2 3 3 3 3 3 2 2 2 2 2 3 3 2 2 2 3 2 3 2 3 3 2 2 3 2
##  [926] 2 3 2 3 3 3 3 2 3 3 3 3 2 2 2 3 3 2 3 3 2 3 2 3 2 2 2 2 3 3 3 2 3 2 3 2 2
##  [963] 3 3 3 2 2 3 3 3 3 2 2 2 3 3 3 2 3 3 3 3 2 3 2 3 3 2 3 3 3 3 3 3 2 2 2 2 3
## [1000] 3
```

```
print(tree_up_2)
```

```
## $info
##      terminal child_L child_R parent split_variable split_value mu node_size
## [1,]        0       2       3     NA              1    8.383972  0      1000
## [2,]        0       4       5      1              1    6.165864 NA       414
## [3,]        1      NA      NA      1             NA          NA NA       586
## [4,]        1      NA      NA      2             NA          NA NA       183
## [5,]        1      NA      NA      2             NA          NA NA       231
##
## $node
##    [1] 3 3 4 3 3 3 5 3 4 4 5 3 3 4 4 3 4 4 3 3 3 5 4 5 5 3 4 3 3 5 4 5 3 5 5 4 3
##   [38] 3 4 5 4 3 3 3 3 3 3 3 3 3 3 4 4 3 5 3 5 3 3 3 4 5 3 3 3 3 3 3 5 3 4 4 5 3
##   [75] 5 5 3 5 5 3 4 4 3 3 3 3 4 3 3 4 4 5 3 3 3 3 3 5 3 5 3 3 3 5 3 5 3 4 4 3 4
##  [112] 4 3 3 5 5 3 4 3 5 3 3 3 3 4 3 5 3 3 5 3 3 3 3 3 3 3 5 4 5 4 3 4 5 5 3 3 4
##  [149] 3 5 3 4 3 5 3 5 3 3 3 4 5 4 4 3 5 3 5 3 3 5 3 5 4 4 3 3 3 5 3 4 4 3 3 3 3
##  [186] 3 4 5 3 3 5 4 4 3 3 3 3 3 5 3 3 5 5 5 3 5 4 3 3 4 4 3 3 3 3 4 3 3 5 3 3 5
##  [223] 5 5 5 3 3 3 3 4 3 3 3 4 3 4 5 4 3 3 3 5 3 5 5 5 5 3 4 3 3 4 3 3 5 5 5 3 3
##  [260] 4 5 3 3 3 5 3 3 5 3 5 3 3 5 3 3 3 3 3 3 3 4 3 4 5 3 4 3 3 3 3 4 4 3 3 4 5
##  [297] 3 3 3 5 5 3 5 5 3 3 3 5 5 3 4 4 4 3 3 3 5 3 4 3 5 3 3 3 5 4 3 3 5 5 3 3 5
##  [334] 3 4 3 4 5 4 4 3 3 4 3 3 4 3 3 5 5 3 3 3 3 4 3 3 4 3 3 4 3 3 3 5 3 3 3 5 3
##  [371] 3 3 3 3 3 3 3 3 3 5 3 3 5 5 5 5 3 3 3 4 5 3 5 3 4 3 3 3 5 5 3 3 3 5 5 3 5
##  [408] 4 4 3 3 5 4 3 3 5 3 3 3 5 4 5 5 3 3 3 3 3 3 3 3 5 5 3 5 4 3 3 3 4 3 5 5 3
##  [445] 3 3 3 4 3 3 3 4 3 3 3 3 3 3 3 3 4 3 3 5 4 5 3 3 4 5 5 3 4 3 4 3 5 5 3 4 3
##  [482] 3 3 3 3 5 3 3 3 3 5 5 5 3 3 3 5 3 5 3 3 4 5 3 5 4 3 3 3 5 3 3 3 5 3 3 3 3
##  [519] 5 3 3 3 3 3 4 4 3 4 5 4 4 3 4 3 3 3 3 4 5 4 4 4 3 3 3 4 5 5 3 3 5 5 3 3 5
##  [556] 3 3 3 4 3 3 3 3 4 4 5 5 5 3 5 3 3 4 5 3 5 5 4 3 3 4 5 3 5 3 3 4 3 4 3 5 3
##  [593] 5 4 3 5 4 3 3 4 5 3 3 3 3 5 3 3 5 3 3 4 3 5 3 3 5 3 3 3 3 5 3 3 3 3 4 3 3
##  [630] 5 4 4 3 3 5 3 3 4 5 3 3 3 3 5 5 3 3 3 3 5 3 3 5 4 3 5 5 3 4 4 4 3 3 5 3 3
##  [667] 5 3 3 3 5 4 4 4 5 3 3 5 3 5 3 3 3 4 3 5 3 3 4 3 3 5 3 3 5 5 5 3 3 5 4 3 3
##  [704] 3 5 5 5 5 3 3 4 3 3 3 3 3 3 3 5 4 4 3 3 5 3 3 4 4 3 4 3 5 3 5 5 5 4 5 5 3
##  [741] 3 3 4 5 3 4 5 3 3 3 3 4 5 3 3 3 5 5 3 3 3 3 5 3 3 3 3 5 3 3 4 3 5 3 3 3 3
##  [778] 3 4 3 3 3 4 3 4 3 3 3 3 5 3 3 3 4 3 4 4 5 3 3 3 3 3 3 3 3 3 5 4 3 3 3 4 3
##  [815] 5 3 3 3 3 3 3 3 4 4 3 5 5 4 5 5 3 3 4 5 3 5 3 3 3 3 5 3 3 3 3 3 3 3 3 3 5
##  [852] 5 3 5 3 3 3 3 3 3 3 3 4 3 3 4 3 3 3 3 3 3 3 3 4 4 3 4 3 3 3 4 5 3 4 3 3 3
##  [889] 5 3 3 4 4 3 3 3 3 3 3 5 3 3 3 3 3 5 5 5 5 5 3 3 5 4 4 3 5 3 5 3 3 5 4 3 5
##  [926] 4 3 5 3 3 3 3 4 3 3 3 3 5 4 4 3 3 5 3 3 4 3 4 3 5 4 4 5 3 3 3 5 3 5 3 4 4
```

```
##  [963] 3 3 3 4 4 3 3 3 3 4 5 4 3 3 3 4 3 3 3 3 5 3 5 3 3 5 3 3 3 3 3 3 4 4 4 5 3
## [1000] 3
```

```
print(tree_up_3)
```

```
## $info
##      terminal child_L child_R parent split_variable split_value mu node_size
## [1,]        0       2       3     NA              1    8.383972  0      1000
## [2,]        0       4       5      1              1    6.165864 NA       414
## [3,]        1      NA      NA      1             NA          NA NA       586
## [4,]        1      NA      NA      2             NA          NA NA       183
## [5,]        0       6       7      2              2   31.273066 NA       231
## [6,]        1      NA      NA      5             NA          NA NA       128
## [7,]        1      NA      NA      5             NA          NA NA       103
##
## $node
##    [1] 3 3 4 3 3 3 7 3 4 4 6 3 3 4 4 3 4 4 3 3 3 7 4 6 7 3 4 3 3 6 4 6 3 6 7 4 3
##   [38] 3 4 6 4 3 3 3 3 3 3 3 3 3 3 4 4 3 6 3 7 3 3 3 4 7 3 3 3 3 3 3 6 3 4 4 6 3
##   [75] 6 7 3 7 6 3 4 4 3 3 3 3 4 3 3 4 4 6 3 3 3 3 3 7 3 6 3 3 3 6 3 6 3 4 4 3 4
##  [112] 4 3 3 7 6 3 4 3 6 3 3 3 3 4 3 7 3 3 6 3 3 3 3 3 3 3 6 4 7 4 3 4 7 7 3 3 4
##  [149] 3 6 3 4 3 7 3 7 3 3 3 4 6 4 4 3 6 3 6 3 3 7 3 6 4 4 3 3 3 7 3 4 4 3 3 3 3
##  [186] 3 4 6 3 3 7 4 4 3 3 3 3 3 7 3 3 7 6 6 3 7 4 3 3 4 4 3 3 3 3 4 3 3 7 3 3 6
##  [223] 7 6 6 3 3 3 3 4 3 3 3 4 3 4 6 4 3 3 3 6 3 7 6 6 6 3 4 3 3 4 3 3 7 6 6 3 3
##  [260] 4 6 3 3 3 7 3 3 6 3 6 3 3 6 3 3 3 3 3 3 3 4 3 4 6 3 4 3 3 3 3 4 4 3 3 4 7
##  [297] 3 3 3 6 6 3 7 7 3 3 3 7 6 3 4 4 4 3 3 3 7 3 4 3 7 3 3 3 7 4 3 3 7 6 3 3 6
##  [334] 3 4 3 4 7 4 4 3 3 4 3 3 4 3 3 6 7 3 3 3 3 4 3 3 4 3 3 4 3 3 3 6 3 3 3 6 3
##  [371] 3 3 3 3 3 3 3 3 3 6 3 3 7 7 6 7 3 3 3 4 6 3 7 3 4 3 3 3 6 7 3 3 3 7 7 3 6
##  [408] 4 4 3 3 6 4 3 3 7 3 3 3 6 4 7 7 3 3 3 3 3 3 3 3 6 6 3 6 4 3 3 3 4 3 6 6 3
##  [445] 3 3 3 4 3 3 3 4 3 3 3 3 3 3 3 3 4 3 3 6 4 7 3 3 4 6 7 3 4 3 4 3 7 7 3 4 3
##  [482] 3 3 3 3 6 3 3 3 3 7 7 6 3 3 3 6 3 7 3 3 4 7 3 6 4 3 3 3 6 3 3 3 6 3 3 3 3
##  [519] 7 3 3 3 3 3 4 4 3 4 7 4 4 3 4 3 3 3 3 4 6 4 4 4 3 3 3 4 6 7 3 3 6 6 3 3 6
##  [556] 3 3 3 4 3 3 3 3 4 4 7 6 6 3 6 3 3 4 6 3 6 6 4 3 3 4 6 3 6 3 3 4 3 4 3 7 3
##  [593] 6 4 3 6 4 3 3 4 7 3 3 3 3 6 3 3 6 3 3 4 3 6 3 3 6 3 3 3 3 6 3 3 3 3 4 3 3
##  [630] 6 4 4 3 3 6 3 3 4 7 3 3 3 3 7 7 3 3 3 3 7 3 3 7 4 3 7 7 3 4 4 4 3 3 6 3 3
##  [667] 6 3 3 3 6 4 4 4 7 3 3 7 3 7 3 3 3 4 3 7 3 3 4 3 3 6 3 3 7 6 7 3 3 7 4 3 3
##  [704] 3 7 6 6 6 3 3 4 3 3 3 3 3 3 3 7 4 4 3 3 7 3 3 4 4 3 4 3 6 3 6 6 6 4 7 7 3
##  [741] 3 3 4 6 3 4 7 3 3 3 3 4 6 3 3 3 6 6 3 3 3 3 6 3 3 3 3 6 3 3 4 3 7 3 3 3 3
##  [778] 3 4 3 3 3 4 3 4 3 3 3 3 6 3 3 3 4 3 4 4 6 3 3 3 3 3 3 3 3 3 7 4 3 3 3 4 3
##  [815] 7 3 3 3 3 3 3 3 4 4 3 6 7 4 6 7 3 3 4 6 3 6 3 3 3 3 6 3 3 3 3 3 3 3 3 3 7
##  [852] 6 3 7 3 3 3 3 3 3 3 3 4 3 3 4 3 3 3 3 3 3 3 3 4 4 3 4 3 3 3 4 7 3 4 3 3 3
##  [889] 6 3 3 4 4 3 3 3 3 3 3 7 3 3 3 3 3 7 6 7 7 7 3 3 6 4 4 3 7 3 6 3 3 7 4 3 6
##  [926] 4 3 7 3 3 3 3 4 3 3 3 3 7 4 4 3 3 7 3 3 4 3 4 3 7 4 4 7 3 3 3 7 3 6 3 4 4
##  [963] 3 3 3 4 4 3 3 3 3 4 6 4 3 3 3 4 3 3 3 3 7 3 6 3 3 6 3 3 3 3 3 3 4 4 4 6 3
## [1000] 3
```

6. Prune Tree Function

6.1 Coding for Prune Tree Function

In this section, we are going to write some function that prunes a tree, i.e., merge two terminal nodes who has the same parent.

```
prune_tree <- function(x, y, curr_tree){

  # To begin with, we create a holder for the new tree, where we can store the information about the ne
  new_tree <- curr_tree

  # If the tree has been the stump, we cannot prune it anymore.
  if(nrow(new_tree$info) == 1){ return(new_tree)}

  # Then, we will get the list of the terminal nodes.
  terminal_nodes <- which(as.numeric(new_tree$info[,"terminal"])==1)

  # Then we randomly pick up a terminal to prune, but it's important that both of the terminal points h
  bad_node <- TRUE
  while (bad_node) {

    # Randomly pick up a point to prune
    selected_node <- sample(terminal_nodes, 1)

    # Then, find the parent of this node
    selected_parent <- as.numeric(new_tree$info[selected_node, "parent"])

    # Then, get the children of this parent
    children_left <- as.numeric(new_tree$info[selected_parent,"child_L"])
    children_right <- as.numeric(new_tree$info[selected_parent, "child_R"])

    # Check whether they are both terminal nodes
    children_left_ter <- as.numeric(new_tree$info[children_left,"terminal"])
    children_right_ter <- as.numeric(new_tree$info[children_right, "terminal"])

    # If both of the nodes are terminal nodes, then it's okay
    if( children_right_ter + children_left_ter == 2 ){ bad_node <- FALSE }


  }

  # After selecting the two terminal nodes, we need to delete the two rows of these nodes.
  new_tree$info <- new_tree$info[-c(children_left, children_right),]

  # Update the information for the parent node since now it's a terminal node, so it does not have the
  new_tree$info[selected_parent,c("terminal", "child_L", "child_R", "split_variable", "split_value")] <

  # If the tree comes back to a stump, there is no need to fill the tree details.
  if(nrow(new_tree$info)==1){new_tree$node <- rep(1, nrow(x))}else{

    # Since we have deleted several rows, there are some row indices changing, if we didn't delete the
    if(selected_node <= nrow(new_tree$info)){

      # Find those whose parents are affected by deleting rows
      bad_parents <- which(as.numeric(new_tree$info[,"parent"])>= selected_node)

      # Since the deleting rows must be continuous two rows, -2 is to make sure the parents are correct
      new_tree$info[bad_parents, "parent"] <- as.numeric(new_tree$info[bad_parents, "parent"]) - 2

```

```
      for (i in selected_node:nrow( new_tree$info )) {

        # Update the child index who has been affected.
        # First, find the parent of the ith row, because we have updated parents before, this row now h
        curr_parent <- as.numeric(new_tree$info[i, "parent"])

        # Then, find the correct children index of the parent row to update the wrong children row inde
        curr_children <-  which(as.numeric(new_tree$info[,"parent"])==curr_parent)

        # Then, update the row indexes.
        new_tree$info[curr_parent, c("child_L", "child_R")] <- sort(curr_children)

      } # End loop for updating children nodes.

    } # End loop for updating tree information

      # Fill tree details
      new_tree <- fill_tree_details(new_tree, x)
  }

      return(new_tree)

}
```

6.2 Testing for Prune Tree Function

```
prune_tree(x, y, tree_up_2)
```

```
## $info
##      terminal child_L child_R parent split_variable split_value mu node_size
## [1,]        0       2       3     NA              1    8.383972  0      1000
## [2,]        1      NA      NA      1             NA          NA NA       414
## [3,]        1      NA      NA      1             NA          NA NA       586
##
## $node
##    [1] 3 3 2 3 3 3 2 3 2 2 2 3 3 2 2 3 2 2 3 3 3 2 2 2 2 3 2 3 3 2 2 2 3 2 2 2 3
##   [38] 3 2 2 2 3 3 3 3 3 3 3 3 3 3 2 2 3 2 3 2 3 3 3 2 2 3 3 3 3 3 3 2 3 2 2 2 3
##   [75] 2 2 3 2 2 3 2 2 3 3 3 3 2 3 3 2 2 2 3 3 3 3 3 2 3 2 3 3 3 2 3 2 3 2 2 3 2
##  [112] 2 3 3 2 2 3 2 3 2 3 3 3 3 2 3 2 3 3 2 3 3 3 3 3 3 3 2 2 2 2 3 2 2 2 3 3 2
##  [149] 3 2 3 2 3 2 3 2 3 3 3 2 2 2 2 3 2 3 2 3 3 2 3 2 2 2 3 3 3 2 3 2 2 3 3 3 3
##  [186] 3 2 2 3 3 2 2 2 3 3 3 3 3 2 3 3 2 2 2 3 2 2 3 3 2 2 3 3 3 3 2 3 3 2 3 3 2
##  [223] 2 2 2 3 3 3 3 2 3 3 3 2 3 2 2 2 3 3 3 2 3 2 2 2 2 3 2 3 3 2 3 3 2 2 2 3 3
##  [260] 2 2 3 3 3 2 3 3 2 3 2 3 3 2 3 3 3 3 3 3 3 2 3 2 2 3 2 3 3 3 3 2 2 3 3 2 2
##  [297] 3 3 3 2 2 3 2 2 3 3 3 2 2 3 2 2 2 3 3 3 2 3 2 3 2 3 3 3 2 2 3 3 2 2 3 3 2
##  [334] 3 2 3 2 2 2 2 3 3 2 3 3 2 3 3 2 2 3 3 3 3 2 3 3 2 3 3 2 3 3 3 2 3 3 3 2 3
##  [371] 3 3 3 3 3 3 3 3 3 2 3 3 2 2 2 2 3 3 3 2 2 3 2 3 2 3 3 3 2 2 3 3 3 2 2 3 2
##  [408] 2 2 3 3 2 2 3 3 2 3 3 3 2 2 2 2 3 3 3 3 3 3 3 2 2 3 2 2 3 3 3 2 3 2 2 3
##  [445] 3 3 3 2 3 3 3 2 3 3 3 3 3 3 3 3 2 3 3 2 2 2 3 3 2 2 2 3 2 3 2 3 2 2 3 2 3
##  [482] 3 3 3 3 2 3 3 3 3 2 2 2 3 3 3 2 3 2 3 3 2 2 3 2 2 3 3 3 2 3 3 3 2 3 3 3 3
##  [519] 2 3 3 3 3 3 2 2 3 2 2 2 2 3 2 3 3 3 3 2 2 2 2 2 3 3 3 2 2 2 3 3 2 2 3 3 2
##  [556] 3 3 3 2 3 3 3 3 2 2 2 2 2 3 2 3 3 2 2 3 2 2 2 3 3 2 2 3 2 3 3 2 3 2 3 2 3
```

```
##  [593] 2 2 3 2 2 3 3 2 2 3 3 3 3 2 3 3 2 3 3 2 3 2 3 3 2 3 3 3 3 2 3 3 3 3 2 3 3
##  [630] 2 2 2 3 3 2 3 3 2 2 3 3 3 3 2 2 3 3 3 3 2 3 3 2 2 3 2 2 3 2 2 2 3 3 2 3 3
##  [667] 2 3 3 3 2 2 2 2 2 3 3 2 3 2 3 3 3 2 3 2 3 3 2 3 3 2 3 3 2 2 2 3 3 2 2 3 3
##  [704] 3 2 2 2 2 3 3 2 3 3 3 3 3 3 3 2 2 2 3 3 2 3 3 2 2 3 2 3 2 3 2 2 2 2 2 2 3
##  [741] 3 3 2 2 3 2 2 3 3 3 3 2 2 3 3 3 2 2 3 3 3 3 2 3 3 3 3 2 3 3 2 3 2 3 3 3 3
##  [778] 3 2 3 3 3 2 3 2 3 3 3 3 2 3 3 3 2 3 2 2 2 3 3 3 3 3 3 3 3 3 2 2 3 3 3 2 3
##  [815] 2 3 3 3 3 3 3 3 2 2 3 2 2 2 2 2 3 3 2 2 3 2 3 3 3 3 2 3 3 3 3 3 3 3 3 3 2
##  [852] 2 3 2 3 3 3 3 3 3 3 3 2 3 3 2 3 3 3 3 3 3 3 3 2 2 3 2 3 3 3 2 2 3 2 3 3 3
##  [889] 2 3 3 2 2 3 3 3 3 3 3 2 3 3 3 3 3 2 2 2 2 2 3 3 2 2 2 3 2 3 2 3 3 2 2 3 2
##  [926] 2 3 2 3 3 3 3 2 3 3 3 3 2 2 2 3 3 2 3 3 2 3 2 3 2 2 2 2 3 3 3 2 3 2 3 2 2
##  [963] 3 3 3 2 2 3 3 3 3 2 2 2 3 3 3 2 3 3 3 3 2 3 2 3 3 2 3 3 3 3 3 3 2 2 2 2 3
## [1000] 3
```

7. Changing Tree

7.1 Preparation

7.1.1 R Coding Tricks - Recursive

In this function, we need to get the children of a parent, and then get the children of the children, which sounds very complicated. However, there is a kind of function in coding, that allows functions to call themselves again and agian, called “recursive” function. For further information, refer to: https://www.datamentor.io/r-programming/recursion/ .

Here are some examples for this kind of function:

```r
recursive_fac <- function(a){

  if(a == 0) {return(1)}
  if(a != 0) {return(a * recursive_fac(a-1))}

}

recursive_fac(5)
```

```
## [1] 120
```

```r
factorial(5)
```

```
## [1] 120
```

7.1.2 Preparation Function: Get Children

So if we changed one parent splitting variable and splitting rule, all the children and grandchildren and so on will be affected. We should corrected the affected childrens. So we have to use a function to get the children and the list of the childrens. Recursive function is a good way to achieve this.

```r
get_children <- function(tree_info, parent){

  all_chilren <- NULL
```

```r
  # If the current node is the terminal node, then return the children list and the parent so far.
  if(as.numeric(tree_info[parent, "terminal"]) == 1){return(c(all_chilren, parent))}else{

    # If the current node is not the terminal node, then we have to recursively get the node list.
    curr_child_left <- as.numeric(tree_info[parent, "child_L"])
    curr_child_right <- as.numeric(tree_info[parent, "child_R"])

    # Return the children and the children of the children recursively
    return(c(all_chilren, get_children(tree_info,curr_child_left), get_children(tree_info, curr_child_ri

  }

}
```

```r
tree_up_3
```

```
## $info
##      terminal child_L child_R parent split_variable split_value mu node_size
## [1,]        0       2       3     NA              1    8.383972  0      1000
## [2,]        0       4       5      1              1    6.165864 NA       414
## [3,]        1      NA      NA      1             NA          NA NA       586
## [4,]        1      NA      NA      2             NA          NA NA       183
## [5,]        0       6       7      2              2   31.273066 NA       231
## [6,]        1      NA      NA      5             NA          NA NA       128
## [7,]        1      NA      NA      5             NA          NA NA       103
##
## $node
##    [1] 3 3 4 3 3 3 7 3 4 4 6 3 3 4 4 3 4 4 3 3 3 7 4 6 7 3 4 3 3 6 4 6 3 6 7 4 3
##   [38] 3 4 6 4 3 3 3 3 3 3 3 3 3 3 4 4 3 6 3 7 3 3 3 4 7 3 3 3 3 3 3 6 3 4 4 6 3
##   [75] 6 7 3 7 6 3 4 4 3 3 3 3 4 3 3 4 4 6 3 3 3 3 3 7 3 6 3 3 3 6 3 6 3 4 4 3 4
##  [112] 4 3 3 7 6 3 4 3 6 3 3 3 3 4 3 7 3 3 6 3 3 3 3 3 3 3 6 4 7 4 3 4 7 7 3 3 4
##  [149] 3 6 3 4 3 7 3 7 3 3 3 4 6 4 4 3 6 3 6 3 3 7 3 6 4 4 3 3 3 7 3 4 4 3 3 3 3
##  [186] 3 4 6 3 3 7 4 4 3 3 3 3 3 7 3 3 7 6 6 3 7 4 3 3 4 4 3 3 3 3 4 3 3 7 3 3 6
##  [223] 7 6 6 3 3 3 3 4 3 3 3 4 3 4 6 4 3 3 3 6 3 7 6 6 6 3 4 3 3 4 3 3 7 6 6 3 3
##  [260] 4 6 3 3 3 7 3 3 6 3 6 3 3 6 3 3 3 3 3 3 3 4 3 4 6 3 4 3 3 3 3 4 4 3 3 4 7
##  [297] 3 3 3 6 6 3 7 7 3 3 3 7 6 3 4 4 4 3 3 3 7 3 4 3 7 3 3 3 7 4 3 3 7 6 3 3 6
##  [334] 3 4 3 4 7 4 4 3 3 4 3 3 4 3 3 6 7 3 3 3 3 4 3 3 4 3 3 4 3 3 3 6 3 3 3 6 3
##  [371] 3 3 3 3 3 3 3 3 3 6 3 3 7 7 6 7 3 3 3 4 6 3 7 3 4 3 3 3 6 7 3 3 3 7 7 3 6
##  [408] 4 4 3 3 6 4 3 3 7 3 3 3 6 4 7 7 3 3 3 3 3 3 3 3 6 6 3 6 4 3 3 3 4 3 6 6 3
##  [445] 3 3 3 4 3 3 3 4 3 3 3 3 3 3 3 3 4 3 3 6 4 7 3 3 4 6 7 3 4 3 4 3 7 7 3 4 3
##  [482] 3 3 3 3 6 3 3 3 3 7 7 6 3 3 3 6 3 7 3 3 4 7 3 6 4 3 3 3 6 3 3 3 6 3 3 3 3
##  [519] 7 3 3 3 3 3 4 4 3 4 7 4 4 3 4 3 3 3 3 4 6 4 4 4 3 3 3 4 6 7 3 3 6 6 3 3 6
##  [556] 3 3 3 4 3 3 3 3 4 4 7 6 6 3 6 3 3 4 6 3 6 6 4 3 3 4 6 3 6 3 3 4 3 4 3 7 3
##  [593] 6 4 3 6 4 3 3 4 7 3 3 3 3 6 3 3 6 3 3 4 3 6 3 3 6 3 3 3 3 6 3 3 3 3 4 3 3
##  [630] 6 4 4 3 3 6 3 3 4 7 3 3 3 3 7 7 3 3 3 3 7 3 3 7 4 3 7 7 3 4 4 4 3 3 6 3 3
##  [667] 6 3 3 3 6 4 4 4 7 3 3 7 3 7 3 3 3 4 3 7 3 3 4 3 3 6 3 3 7 6 7 3 3 7 4 3 3
##  [704] 3 7 6 6 6 3 3 4 3 3 3 3 3 3 3 7 4 4 3 3 7 3 3 4 4 3 4 3 6 3 6 6 6 4 7 7 3
##  [741] 3 3 4 6 3 4 7 3 3 3 3 4 6 3 3 3 6 6 3 3 3 3 6 3 3 3 3 6 3 3 4 3 7 3 3 3 3
##  [778] 3 4 3 3 3 4 3 4 3 3 3 3 6 3 3 3 4 3 4 4 6 3 3 3 3 3 3 3 3 3 7 4 3 3 3 4 3
##  [815] 7 3 3 3 3 3 3 3 4 4 3 6 7 4 6 7 3 3 4 6 3 6 3 3 3 3 6 3 3 3 3 3 3 3 3 3 7
##  [852] 6 3 7 3 3 3 3 3 3 3 3 4 3 3 4 3 3 3 3 3 3 3 3 4 4 3 4 3 3 3 4 7 3 4 3 3 3
##  [889] 6 3 3 4 4 3 3 3 3 3 3 7 3 3 3 3 3 7 6 7 7 7 3 3 6 4 4 3 7 3 6 3 3 7 4 3 6
##  [926] 4 3 7 3 3 3 3 4 3 3 3 3 7 4 4 3 3 7 3 3 4 3 4 3 7 4 4 7 3 3 3 7 3 6 3 4 4
```

```
##  [963] 3 3 3 4 4 3 3 3 3 4 6 4 3 3 3 4 3 3 3 3 7 3 6 3 3 6 3 3 3 3 3 3 4 4 4 6 3
## [1000] 3
```

```
get_children(tree_up_3$info, 1)
```

```
## [1] 4 6 7 3
```

7.1.3 New Function - "Match"

"Match" is a function that look for which elements of the second vector match the first vector. For example:

```
match(1, c(2,3,4,1))
```

```
## [1] 4
```

It shows that the fourth element of the second vector match the first vector , 1.

```
match(c(1,2), c(2,3,4,5,1))
```

```
## [1] 5 1
```

This shows that the fifth element in the vector is 1, and the first matches 2.

Also, it can be written as A %in% B. If elements of A is in B, then it will return TRUE, otherwise FALSE.

7.2 Coding for Changing Tree

```
change_tree <- function(x, y, curr_tree, node_min_size){

  # In this function, since we are changing the tree structure, we have to make sure that the new tree
  # It has at least some observations in each terminal node, which is decided by the "node_min_size".

  # If it's a stump, there is nothing to change.
  if(nrow(curr_tree$info) == 1){return(curr_tree)}

  # Then, create a information box for the new three
  new_tree <- curr_tree

  # Since changing means change out the split variable and split rule, we need to make sure that this i
  internal_node <- which(as.numeric(new_tree$info[,"terminal"]) == 0)
  terminal_node <- which(as.numeric(new_tree$info[,"terminal"]) == 1)

  # Then, we can use a while loop to get a good tree.
  max_bad <- 100
  count_bad <- 0
  bad_tree <- TRUE

  while (bad_tree) {
    # When it's a bad tree, our changing has to be reset to make a new tree.
```

```
    new_tree <- curr_tree

    # Then we select a internal node to change.
    selected_node <- sample(internal_node, 1)

    # Use the get_children function to get all the children nodes of this node
    all_children <- get_children(new_tree$info, selected_node)

    # First, we find the nodes that corresponding to these children, then we delete those who are not i
    # This step is used to getting the further available value because we have to first find which poin
    use_node <- !is.na(match(new_tree$node, all_children))

    # Then we create a new split variable and a new split value.
    available_values <- NULL
    new_split <- sample(1:ncol(x), 1)

    # By using the selected points, we can get the available values.
    available_values <- sort(unique(x[use_node, new_split]))

    if(length(available_values) == 1){split_val <- available_values[1]}else if(length(available_values)
      split_val <- available_values[2]
    }else{

      split_val <- gdata::resample(available_values[-c(1, length(available_values))], 1)
    }

    # Then, we can update the tree information
    new_tree$info[selected_node, c("split_variable", "split_value")] <- c(new_split, split_val)

    # Updating the tree details using the fill tree function
    new_tree <- fill_tree_details(new_tree, x)

    # Now, we finished changing the splitting variable and splitting rule, but we have to check whether
    if(any(new_tree$info[terminal_node,"node_size"] <= node_min_size)){count_bad <- count_bad + 1}else{
      bad_tree <- FALSE
    }

    if(count_bad == max_bad){return(curr_tree)}

  }


  return(new_tree)

}
```

7.3 Check the Code

```
change_tree(x,y,tree_up_1, node_min_size = 10)
```

```
## $info
```

```
##      terminal child_L child_R parent split_variable split_value mu node_size
## [1,]        0       2       3     NA              2    43.86086  0      1000
## [2,]        1      NA      NA      1             NA          NA NA       914
## [3,]        1      NA      NA      1             NA          NA NA        86
##
## $node
##    [1] 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 3 2
##   [38] 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 3 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##   [75] 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [112] 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 2
##  [149] 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2
##  [186] 2 2 2 2 2 2 2 2 2 2 2 3 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [223] 2 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2
##  [260] 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2
##  [297] 2 2 3 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2
##  [334] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 3 2 3 2 2 2 2 2 2 2 2
##  [371] 2 3 3 2 3 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 3 3 2
##  [408] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [445] 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 3 2 2 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 3
##  [482] 2 2 2 2 2 2 2 3 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3
##  [519] 3 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [556] 2 2 2 3 2 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2
##  [593] 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2
##  [630] 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2
##  [667] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [704] 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 3 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [741] 2 2 2 2 2 2 2 3 2 3 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2
##  [778] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3
##  [815] 2 2 2 2 2 2 2 2 3 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2
##  [852] 2 2 2 2 2 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 3 3 2 2 3 2 2 2 2 2
##  [889] 2 2 2 2 3 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [926] 2 2 2 2 2 3 2 2 2 2 2 2 3 3 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 3
##  [963] 2 2 2 2 2 2 2 3 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [1000] 2
```

```r
change_tree(x,y,tree_up_2, node_min_size = 10)
```

```
## $info
##      terminal child_L child_R parent split_variable split_value mu node_size
## [1,]        0       2       3     NA              1    8.383972  0      1000
## [2,]        0       4       5      1              1    6.235634 NA       414
## [3,]        1      NA      NA      1             NA          NA NA       586
## [4,]        1      NA      NA      2             NA          NA NA       188
## [5,]        1      NA      NA      2             NA          NA NA       226
##
## $node
##    [1] 3 3 4 3 3 3 5 3 4 4 5 3 3 4 4 3 4 4 3 3 3 5 4 5 5 3 4 3 3 5 4 5 3 5 5 4 3
##   [38] 3 4 4 4 3 3 3 3 3 3 3 3 3 3 4 4 3 5 3 5 3 3 3 4 5 3 3 3 3 3 3 5 3 4 4 5 3
##   [75] 5 5 3 5 5 3 4 4 3 3 3 3 4 3 3 4 4 5 3 3 3 3 3 5 3 5 3 3 3 5 3 5 3 4 4 3 4
##  [112] 4 3 3 5 5 3 4 3 5 3 3 3 3 4 3 5 3 3 5 3 3 3 3 3 3 3 5 4 5 4 3 4 5 5 3 3 4
##  [149] 3 5 3 4 3 5 3 5 3 3 3 4 5 4 4 3 5 3 5 3 3 5 3 5 4 4 3 3 3 5 3 4 4 3 3 3 3
##  [186] 3 4 5 3 3 5 4 4 3 3 3 3 3 5 3 3 5 5 5 3 5 4 3 3 4 4 3 3 3 3 4 3 3 5 3 3 5
##  [223] 5 5 5 3 3 3 3 4 3 3 3 4 3 4 5 4 3 3 3 5 3 4 5 5 5 3 4 3 3 4 3 3 5 5 5 3 3
##  [260] 4 5 3 3 3 5 3 3 5 3 5 3 3 4 3 3 3 3 3 3 3 4 3 4 5 3 4 3 3 3 3 4 4 3 3 4 5
```

```
##  [297] 3 3 3 5 5 3 5 5 3 3 3 5 5 3 4 4 4 3 3 3 5 3 4 3 5 3 3 3 5 4 3 3 5 5 3 3 5
##  [334] 3 4 3 4 5 4 4 3 3 4 3 3 4 3 3 5 5 3 3 3 3 4 3 3 4 3 3 4 3 3 3 5 3 3 3 5 3
##  [371] 3 3 3 3 3 3 3 3 3 5 3 3 5 5 5 5 3 3 3 4 5 3 5 3 4 3 3 3 5 5 3 3 3 5 5 3 5
##  [408] 4 4 3 3 5 4 3 3 5 3 3 3 4 4 5 5 3 3 3 3 3 3 3 3 3 5 5 3 5 4 3 3 3 4 3 5 5 3
##  [445] 3 3 3 4 3 3 3 4 3 3 3 3 3 3 3 3 4 3 3 5 4 5 3 3 4 5 5 3 4 3 4 3 5 5 3 4 3
##  [482] 3 3 3 3 4 3 3 3 3 5 5 5 3 3 3 5 3 5 3 3 4 5 3 5 4 3 3 3 5 3 3 3 5 3 3 3 3
##  [519] 5 3 3 3 3 3 4 4 3 4 5 4 4 3 4 3 3 3 3 4 5 4 4 4 3 3 3 4 5 5 3 3 5 5 3 3 5
##  [556] 3 3 3 4 3 3 3 3 4 4 5 5 5 3 5 3 3 4 5 3 5 5 4 3 3 4 5 3 5 3 3 4 3 4 3 5 3
##  [593] 5 4 3 5 4 3 3 4 5 3 3 3 3 5 3 3 5 3 3 4 3 5 3 3 5 3 3 3 3 5 3 3 3 3 4 3 3
##  [630] 5 4 4 3 3 5 3 3 4 5 3 3 3 3 5 5 3 3 3 3 5 3 3 5 4 3 5 5 3 4 4 4 3 3 5 3 3
##  [667] 5 3 3 3 5 4 4 4 5 3 3 5 3 5 3 3 3 4 3 5 3 3 4 3 3 5 3 3 5 5 5 3 3 5 4 3 3
##  [704] 3 5 5 5 5 3 3 4 3 3 3 3 3 3 3 5 4 4 3 3 5 3 3 4 4 3 4 3 5 3 5 5 5 4 5 5 3
##  [741] 3 3 4 5 3 4 5 3 3 3 3 4 5 3 3 3 5 5 3 3 3 3 5 3 3 3 3 5 3 3 4 3 5 3 3 3 3
##  [778] 3 4 3 3 3 4 3 4 3 3 3 3 5 3 3 3 4 3 4 4 5 3 3 3 3 3 3 3 3 3 5 4 3 3 3 4 3
##  [815] 5 3 3 3 3 3 3 3 4 4 3 5 5 4 5 5 3 3 4 5 3 5 3 3 3 3 5 3 3 3 3 3 3 3 3 3 5
##  [852] 5 3 5 3 3 3 3 3 3 3 3 4 3 3 4 3 3 3 3 3 3 3 3 4 4 3 4 3 3 3 4 5 3 4 3 3 3
##  [889] 5 3 3 4 4 3 3 3 3 3 3 5 3 3 3 3 3 5 5 5 5 5 3 3 5 4 4 3 5 3 5 3 3 5 4 3 5
##  [926] 4 3 5 3 3 3 3 4 3 3 3 3 5 4 4 3 3 5 3 3 4 3 4 3 5 4 4 5 3 3 3 5 3 5 3 4 4
##  [963] 3 3 3 4 4 3 3 3 3 4 5 4 3 3 3 4 3 3 3 3 5 3 5 3 3 5 3 3 3 3 3 3 4 4 4 5 3
## [1000] 3
```

```
change_tree(x,y,tree_up_3, node_min_size = 10)
```

```
## $info
##      terminal child_L child_R parent split_variable split_value mu node_size
## [1,]        0       2       3     NA              1    8.383972  0      1000
## [2,]        0       4       5      1              2   24.057945 NA       414
## [3,]        1      NA      NA      1             NA          NA NA       586
## [4,]        1      NA      NA      2             NA          NA NA       100
## [5,]        0       6       7      2              2   31.273066 NA       314
## [6,]        1      NA      NA      5             NA          NA NA       124
## [7,]        1      NA      NA      5             NA          NA NA       190
##
## $node
##    [1] 3 3 6 3 3 3 7 3 7 6 4 3 3 7 7 3 4 6 3 3 3 7 7 6 7 3 7 3 3 6 6 6 3 4 7 7 3
##   [38] 3 4 6 4 3 3 3 3 3 3 3 3 3 3 3 7 4 3 6 3 7 3 3 3 6 7 3 3 3 3 3 3 4 3 4 7 4 3
##   [75] 6 7 3 7 6 3 4 4 3 3 3 3 4 3 3 7 7 4 3 3 3 3 3 3 7 3 4 3 3 3 4 3 6 3 6 6 3 6
##  [112] 4 3 3 7 6 3 7 3 4 3 3 3 3 7 3 7 3 3 6 3 3 3 3 3 3 3 4 7 7 4 3 7 7 7 3 3 7
##  [149] 3 4 3 6 3 7 3 7 3 3 3 6 4 6 4 3 4 3 4 3 3 7 3 6 7 7 3 3 3 7 3 7 6 3 3 3 3
##  [186] 3 4 6 3 3 7 6 4 3 3 3 3 3 7 3 3 7 4 6 3 7 7 3 3 4 7 3 3 3 3 4 3 3 7 3 3 6
##  [223] 7 6 4 3 3 3 3 6 3 3 3 7 3 4 4 4 3 3 3 6 3 7 4 6 4 3 4 3 3 7 3 3 7 6 4 3 3
##  [260] 4 4 3 3 3 7 3 3 6 3 4 3 3 6 3 3 3 3 3 3 3 3 6 3 7 6 3 6 3 3 3 3 4 7 3 3 7 7
##  [297] 3 3 3 6 4 3 7 7 3 3 3 7 4 3 7 7 6 3 3 3 7 3 6 3 7 3 3 3 7 7 3 3 7 4 3 3 4
##  [334] 3 6 3 6 7 7 4 3 3 6 3 3 6 3 3 4 7 3 3 3 3 4 3 3 7 3 3 6 3 3 3 4 3 3 3 4 3
##  [371] 3 3 3 3 3 3 3 3 3 6 3 3 7 7 4 7 3 3 3 6 6 3 7 3 7 3 3 3 4 7 3 3 3 7 7 3 4
##  [408] 7 7 3 3 6 4 3 3 7 3 3 3 6 4 7 7 3 3 3 3 3 3 3 3 6 6 3 4 7 3 3 3 7 3 4 4 3
##  [445] 3 3 3 7 3 3 3 6 3 3 3 3 3 3 3 3 7 3 3 4 6 7 3 3 6 6 7 3 6 3 6 3 7 7 3 6 3
##  [482] 3 3 3 3 4 3 3 3 3 7 7 6 3 3 3 4 3 7 3 3 6 7 3 6 7 3 3 3 4 3 3 3 6 3 3 3 3
##  [519] 7 3 3 3 3 3 7 4 3 4 7 7 4 3 7 3 3 3 3 6 4 4 7 7 3 3 3 7 6 7 3 3 6 6 3 3 4
##  [556] 3 3 3 7 3 3 3 3 6 4 7 6 4 3 6 3 3 7 6 3 4 6 7 3 3 7 4 3 4 3 3 7 3 4 3 7 3
##  [593] 6 7 3 6 7 3 3 7 7 3 3 3 3 6 3 3 6 3 3 7 3 6 3 3 4 3 3 3 3 4 3 3 3 3 7 3 3
##  [630] 6 7 7 3 3 6 3 3 6 7 3 3 3 3 7 7 3 3 3 3 7 3 3 7 4 3 7 7 3 6 7 4 3 3 6 3 3
##  [667] 6 3 3 3 6 7 7 4 7 3 3 7 3 7 3 3 3 7 3 7 3 3 4 3 3 4 3 3 7 6 7 3 3 7 6 3 3
##  [704] 3 7 4 4 6 3 3 7 3 3 3 3 3 3 3 7 6 6 3 3 7 3 3 7 4 3 7 3 6 3 6 4 4 6 7 7 3
```

```
##  [741] 3 3 6 4 3 7 7 3 3 3 3 4 6 3 3 3 6 6 3 3 3 3 6 3 3 3 3 4 3 3 6 3 7 3 3 3 3
##  [778] 3 7 3 3 3 7 3 6 3 3 3 3 6 3 3 3 7 3 4 4 6 3 3 3 3 3 3 3 3 3 7 6 3 3 3 4 3
##  [815] 7 3 3 3 3 3 3 3 7 7 3 6 7 6 6 7 3 3 7 6 3 6 3 3 3 3 4 3 3 3 3 3 3 3 3 3 7
##  [852] 6 3 7 3 3 3 3 3 3 3 3 6 3 3 7 3 3 3 3 3 3 3 3 6 4 3 4 3 3 3 4 7 3 7 3 3 3
##  [889] 4 3 3 6 7 3 3 3 3 3 3 7 3 3 3 3 3 7 6 7 7 7 3 3 6 7 7 3 7 3 6 3 3 7 4 3 6
##  [926] 7 3 7 3 3 3 3 6 3 3 3 3 7 7 7 3 3 7 3 3 6 3 4 3 7 7 7 7 3 3 3 7 3 6 3 6 7
##  [963] 3 3 3 7 6 3 3 3 3 7 4 7 3 3 3 7 3 3 3 3 7 3 6 3 3 6 3 3 3 3 3 3 6 7 7 6 3
## [1000] 3
```

8. Swap the Tree Function

8.1 Swap the Tree Function

In the swap setting, we swap two neighboring internal nodes around their split values and split variables, so we have to find the neighbouring internal nodes, then switch the splitting rules.

```
swap_tree <- function(x, y, curr_tree, node_min_size){

  # Swap means we have to find two neighboring internal nodes, and then swaps their splitting values an

  # If the tree is a stump, then we cannot swap anymore
  if(nrow(curr_tree$info) == 1){ return(curr_tree) }

  # Create an information box for new tree
  new_tree <- curr_tree

  # Same as "change", we need to find which nodes are internal and which are terminal
  internal_node <- which(as.numeric(new_tree$info[,"terminal"]) == 0)
  terminal_node <- which(as.numeric(new_tree$info[,"terminal"]) == 1)

  # If the tree is too small, like it only has one or two internal node, then swapping is useless since
  if(length(internal_node) < 3 ){ return(curr_tree)}

  # Then we need to find a pair of neighboring internal nodes
  parent_internal <- as.numeric(new_tree$info[internal_node, "parent"])

  # This step, we bind two internal nodes by column and create the pairs of internal nodes.
  # -1 because the root node doesn't have a parent, so it's useless as a child.
  # Be careful that this step creates a p*2 matrix, because internal_node is a vector, and parent is al
  # This matrix describes:
      # 1st column: Which points are internal nodes
      # 2nd column: Which nodes are the parent of the 1st column correspondingly.
  pairs_internal <- cbind(internal_node, parent_internal)[-1,]


  # Then create a loop to get good trees.
  # Set the maximum number of bad trees.

  max_bad <- 10
  count_bad <- 0
  bad_tree <- TRUE
```

```
  while (bad_tree) {
    new_tree <- curr_tree

    # Pick up a random pair
    selected_node <- sample(1:nrow(pairs_internal),1)

    # Get the split variable and split value for this pair
    # 1 for some internal node, 2 for the parent of this node
    swap_1_rule <- as.numeric(new_tree$info[pairs_internal[selected_node,1], c("split_variable", "split_
    swap_2_rule <- as.numeric(new_tree$info[pairs_internal[selected_node,2], c("split_variable", "split_

    # Change the tree information matrix, interchange the splitting rules
    new_tree$info[pairs_internal[selected_node,1], c("split_variable", "split_value")] <- swap_2_rule
    new_tree$info[pairs_internal[selected_node,2], c("split_variable", "split_value")] <- swap_1_rule


    # Then we should fill in the tree details
    new_tree <- fill_tree_details(new_tree, x)

    # Check whether it's a bad tree
    if(any(as.numeric(new_tree$info[,"node_size"]) <= node_min_size)){ count_bad <- count_bad + 1}else{
      bad_tree <- FALSE
    }

    if(max_bad == count_bad){ return(curr_tree)}

  }

  return(new_tree)

}
```

8.2 Testing Swap Function

```
tree_up_4 <- grow_tree(x,y,tree_up_3, node_min_size = 10)
swap_tree(x,y,curr_tree = tree_up_4, node_min_size = 10)
```

```
## $info
##       terminal child_L child_R parent split_variable split_value mu node_size
##  [1,]        0       2       3     NA              1    8.383972  0      1000
##  [2,]        0       4       5      1              2   31.273066 NA       414
##  [3,]        1      NA      NA      1             NA          NA NA       586
##  [4,]        1      NA      NA      2             NA          NA NA       224
##  [5,]        0       6       7      2              1    6.165864 NA       190
##  [6,]        1      NA      NA      5             NA          NA NA        87
##  [7,]        0       8       9      5              2   37.789860 NA       103
##  [8,]        1      NA      NA      7             NA          NA NA        45
##  [9,]        1      NA      NA      7             NA          NA NA        58
##
## $node
##    [1] 3 3 4 3 3 3 9 3 6 4 4 3 3 6 6 3 4 4 3 3 3 8 6 4 8 3 6 3 3 4 4 4 3 4 8 6 3
```

```
##    [38] 3 4 4 4 3 3 3 3 3 3 3 3 3 3 6 4 3 4 3 9 3 3 3 4 9 3 3 3 3 3 3 4 3 4 6 4 3
##    [75] 4 8 3 8 4 3 4 4 3 3 3 3 4 3 3 6 6 4 3 3 3 3 3 9 3 4 3 3 3 4 3 4 3 4 4 3 4
##   [112] 4 3 3 8 4 3 6 3 4 3 3 3 3 6 3 9 3 3 4 3 3 3 3 3 3 3 4 6 9 4 3 6 9 8 3 3 6
##   [149] 3 4 3 4 3 8 3 9 3 3 3 4 4 4 4 3 4 3 4 3 3 9 3 4 6 6 3 3 3 9 3 6 4 3 3 3 3
##   [186] 3 4 4 3 3 8 4 4 3 3 3 3 3 9 3 3 8 4 4 3 9 6 3 3 4 6 3 3 3 3 4 3 3 9 3 3 4
##   [223] 9 4 4 3 3 3 3 4 3 3 3 6 3 4 4 4 3 3 3 4 3 8 4 4 4 3 4 3 3 6 3 3 9 4 4 3 3
##   [260] 4 4 3 3 3 8 3 3 4 3 4 3 3 4 3 3 3 3 3 3 3 4 3 6 4 3 4 3 3 3 3 4 6 3 3 6 8
##   [297] 3 3 3 4 4 3 8 9 3 3 3 8 4 3 6 6 4 3 3 3 9 3 4 3 9 3 3 3 8 6 3 3 9 4 3 3 4
##   [334] 3 4 3 4 9 6 4 3 3 4 3 3 4 3 3 4 8 3 3 3 3 4 3 3 6 3 3 4 3 3 3 4 3 3 3 4 3
##   [371] 3 3 3 3 3 3 3 3 3 4 3 3 9 9 4 9 3 3 3 4 4 3 9 3 6 3 3 3 4 9 3 3 3 8 9 3 4
##   [408] 6 6 3 3 4 4 3 3 8 3 3 3 4 4 8 9 3 3 3 3 3 3 3 3 4 4 3 4 6 3 3 3 6 3 4 4 3
##   [445] 3 3 3 6 3 3 3 4 3 3 3 3 3 3 3 3 6 3 3 4 4 8 3 3 4 4 9 3 4 3 4 3 9 8 3 4 3
##   [482] 3 3 3 3 4 3 3 3 3 8 9 4 3 3 3 4 3 8 3 3 4 8 3 4 6 3 3 3 4 3 3 3 4 3 3 3 3
##   [519] 9 3 3 3 3 3 6 4 3 4 9 6 4 3 6 3 3 3 3 4 4 4 6 6 3 3 3 6 4 8 3 3 4 4 3 3 4
##   [556] 3 3 3 6 3 3 3 3 4 4 9 4 4 3 4 3 3 6 4 3 4 4 6 3 3 6 4 3 4 3 3 6 3 4 3 8 3
##   [593] 4 6 3 4 6 3 3 6 8 3 3 3 3 4 3 3 4 3 3 6 3 4 3 3 4 3 3 3 3 4 3 3 3 3 6 3 3
##   [630] 4 6 6 3 3 4 3 3 4 8 3 3 3 3 8 9 3 3 3 3 9 3 3 9 4 3 9 9 3 4 6 4 3 3 4 3 3
##   [667] 4 3 3 3 4 6 6 4 8 3 3 8 3 9 3 3 3 6 3 9 3 3 4 3 3 4 3 3 9 4 8 3 3 9 4 3 3
##   [704] 3 9 4 4 4 3 3 6 3 3 3 3 3 3 3 9 4 4 3 3 8 3 3 6 4 3 6 3 4 3 4 4 4 4 9 8 3
##   [741] 3 3 4 4 3 6 8 3 3 3 3 4 4 3 3 3 4 4 3 3 3 3 4 3 3 3 3 4 3 3 4 3 8 3 3 3 3
##   [778] 3 6 3 3 3 6 3 4 3 3 3 3 4 3 3 3 6 3 4 4 4 3 3 3 3 3 3 3 3 3 8 4 3 3 3 4 3
##   [815] 9 3 3 3 3 3 3 3 6 6 3 4 9 4 4 9 3 3 6 4 3 4 3 3 3 3 4 3 3 3 3 3 3 3 3 3 8
##   [852] 4 3 9 3 3 3 3 3 3 3 3 4 3 3 6 3 3 3 3 3 3 3 3 4 4 3 4 3 3 3 4 9 3 6 3 3 3
##   [889] 4 3 3 4 6 3 3 3 3 3 3 8 3 3 3 3 3 9 4 8 9 9 3 3 4 6 6 3 9 3 4 3 3 9 4 3 4
##   [926] 6 3 8 3 3 3 3 4 3 3 3 3 9 6 6 3 3 8 3 3 4 3 4 3 9 6 6 9 3 3 3 8 3 4 3 4 6
##   [963] 3 3 3 6 4 3 3 3 3 6 4 6 3 3 3 6 3 3 3 3 8 3 4 3 3 4 3 3 3 3 3 3 4 6 6 4 3
##  [1000] 3
```

9. Full Conditional Distribution

Since the distribution of the tree is not in a closed form, we need to get the exact value of the $p(Tree|others)$. But we can get a proportional number to use the MH steps. Before getting the code to calculate the tree full conditional, the other parameters could be presented in a closed form as follows.

This model can be described as follows:

$$Y_{ij} \sim^{i.i.d} N(\mu_i, \sigma_i^2)$$

where μ_i is the mean of the i^{th} terminal node, and the σ_i^2 is the variance of the i^{th} terminal node. We can further set the model as follows:

$$\mu_i \sim^{i.i.d} N(\theta, \tau_\mu), \quad \sigma_i \sim^{ind} IG(a_i, b_i)$$

However, we can also set a flat prior on μ_i and σ_i as follows:

$$p(\mu_i, \sigma_i^2) \propto \frac{1}{\sigma_i^2}$$

Actually, we can further set a hierarchical model and put priors on θ, τ_μ and other parameters. But here we just set them as hyperparameters and fixed.

Therefore, the posterior distribution of each parameter can be presented as follows if the tree structure is given:

$$p(\mu_i|others) \propto exp(-\frac{\sum_{j=1}^{n_i}(y_{ij}-\mu_i)^2}{2\sigma_i^2} - \frac{(\mu_i-\theta)^2}{2\tau_\mu^2}) \propto exp\left(-\frac{1}{2}\left((\frac{n_i}{\sigma_i^2}+\frac{1}{\tau_\mu^2})\mu_i^2 - 2\mu_i(\frac{\sum_{j=1}^{n_i} y_{ij}}{\sigma_i^2}+\frac{\theta}{\tau_\mu^2})\right)\right)$$

$$\mu_i \sim N(\frac{\frac{\sum_{j=1}^{n_i} y_{ij}}{\sigma_i^2}+\frac{\theta}{\tau_\mu^2}}{\frac{n_i}{\sigma_i^2}+\frac{1}{\tau_\mu^2}}, \frac{1}{\frac{n_i}{\sigma_i^2}+\frac{1}{\tau_\mu^2}})$$

For σ_i^2:

$$p(\sigma_i^2|others) \propto (\sigma_i^2)^{-\frac{n_i}{2}} exp(-\frac{\sum_{j=1}^{n_i}(y_{ij}-\mu_i)^2}{2}(\sigma_i^2)^{-1}) \times (\sigma_i^2)^{-a_i+1} exp(-\frac{1}{b_i}(\sigma_i^2)^{-1})$$

$$\sigma_i^2 \sim IG(\frac{n_i}{2}+a_i, \frac{\sum_{j=1}^{n_i}(y_{ij}-\mu_i)^2}{2}+b_i)$$

Here is the case that I regard only μ_i and σ_i as unknown parameters. If we further assume all σ_i^2 are equal to each other, and $\theta = 0$, the full conditional posterior of μ_i and σ^2 become:

$$\mu_i \sim N(\frac{\frac{\sum_{j=1}^{n_i} y_{ij}}{\sigma_i^2}}{\frac{n_i}{\sigma_i^2}+\frac{1}{\tau_\mu^2}}, \frac{1}{\frac{n_i}{\sigma_i^2}+\frac{1}{\tau_\mu^2}})$$

If we use a reparameterization that $\tau_\mu^2 = \frac{\sigma^2}{a}$, the distribution could be written as:

$$\mu_i \sim N(\frac{\sum_{i=1}^{n_i} y_{ij}}{n_i+a}, \frac{\sigma_i^2}{n_i+a})$$

$$\sigma^2 \sim IG(\frac{N}{2}+a, \frac{\sum_{i=1}^{I}\sum_{j=1}^{n_i}(y_{ij}-\mu_i)^2}{2}+b)$$

```
tree_marginal <- function(tree, x, y, c, alpha_sig, beta_sig){
  # Calculate the marginal distribution for the tree to get a sample from.
  I <- length(table(tree$node))
  ni <- table(tree$node)
  terminal_node <- which(as.numeric(tree$info[,"terminal"]) == 1 )
  part1 <-  -1/2 * sum(log(ni+c))


  M <- 0

  for (i in 1:I) {
    y_i <- y[tree$node==terminal_node[i]]
    M <- M + (sum(y_i^2) - sum(y_i)^2/(ni[i] + c))/2

  }

  part2 <- -(length(y)/2 + alpha_sig)*log(beta_sig+M)

  # Get the log marginal likelihood
  Log_mar <- part1 + part2
  return(Log_mar)
}
```

```r
# Since the distribution of mu and sigma are conjugate, we can directly sample from the posterior distr
# This is a function that sample the mean of each terminal node as a vector together.



fill_update_mu <- function(y, sigma, c, curr_tree){
  ni <- table(curr_tree$node)
  mean_vec <- NULL
  var_vec <- NULL
  # Update each mean and variance, since they are iid distributed, I can generate them together by usin

   mean_vec <- aggregate(x = y, by = list(curr_tree$node), FUN = sum)$V1 / (ni+c)
  var_vec <- sigma / (ni+c)

#  for (i in 1:length(ni)) {
#    mean_vec[i] <- (sum(y[curr_tree$node==names(ni)[i]])) / (ni[i] + c )
#    var_vec[i] <- sigma / (ni[i] + c )
#  }

  mu_update <- mvtnorm::rmvnorm(1, mean = mean_vec,  sigma = diag(var_vec, ncol = length(ni)))
  new_tree <- curr_tree

  new_tree$info[which(as.numeric(new_tree$info[,"terminal"])==1), "mu"] <- mu_update
  return(new_tree)
}

sample_sigma <- function(y, mu, alpha_sig, beta_sig, tree, c){

  # Get the posterior parameters for the inverse gamma distribution
  ni <- table(tree$node)
  alpha <- length(y)/2 + length(ni)/2 + alpha_sig
  SS <- NULL
  for (i in 1:length(ni)) {
    SS[i] <- sum((y[tree$node==ni[i]]-mu[i])^2)
  }
  beta <- sum(SS)/2 + c*sum(mu^2)/2 + beta_sig
  sigma_update <- 1/rgamma(1, shape = alpha, rate = beta)
  return(sigma_update)
}
```

10. Get the prior distribution of the tree p(tree)

Remember that the splitting probability can be written as:

$$\alpha(1+d)^{-\beta}$$

```r
get_tree_prior <- function(tree, alpha, beta){
  # First, we need to work the depth of the tree
  # So we need to find the level of each node, then the depth is the maximum of the level

  level <- rep(NA, nrow(tree$info))
```

```
  # The first node is the 0 depth of the tree
  level[1] <- 0

  if(nrow(tree$info)==1){return(log(1-alpha))} # Because the tree depth is 0.

  for (i in 2:nrow(tree$info)) {
    # We need to first find the current parent
    curr_parent <- as.numeric(tree$info[i,"parent"])

    # Then this child must have one more level than its parent
    level[i] <- level[curr_parent] + 1
  }

  # If we only compute the internal nodes
  internal_node <- which(as.numeric(tree$info[,"terminal"])==0)
  log_prior <- 0
  for (i in 1:length(internal_node)) {
    log_prior <- log_prior + log(alpha) - beta*log(1 + level[internal_node[i]])
  }

  # Also, in each terminal node, it does not split
  terminal_node <- which(as.numeric(tree$info[,"terminal"])==1)
  for (j in 1:length(terminal_node)) {
    log_prior <- log_prior + log(1-alpha*(1+level[terminal_node[j]])^(-beta))

  }

  return(log_prior)
}
```

11. Train the Model

```
train_bart <- function(x, y, num_trees = 1, # number of trees
                       control = list(node_min_size = 5), # control the minimum terminal node size
                       hyper = list(alpha = 0.95, beta = 2, c = 0.01, alpha_sig=5, beta_sig=5), # Give
                       init = list(sigma = 1), # sigma2 initial value
                       mcmc_par = list(iter = 1000, # number of total iterations
                                       burn = 100, # number of burn-in
                                       thin = 1) # how to thin the chain

){

  # Scale the covariates
  # We need to record how we scaled them so that we can predict new observations.

  # Only calculate those columns who are numeric


  # Get the columns that are numeric
  # Create a holder
```

```
if(is.null(ncol(x))){
  center_x_num = mean(x)
  scale_x_num = sd(x)
  }else{
    center_x <- apply(x, 2, mean)
    scale_x <- apply(x, 2, sd)}

  x <- scale(x)

    center_y <- mean(y)
    scale_y <- sd(y)
    y <- scale(y)





# Extract control parameters
node_min_size <- control$node_min_size

# Extract initial values
sigma <- init$sigma
log_lik <- 0

# Extract hyperparameters
alpha <- hyper$alpha
beta <- hyper$beta
c <- hyper$c
alpha_sig <- hyper$alpha_sig
beta_sig <- hyper$beta_sig


# Extract MCMC details
iter <- mcmc_par$iter
burn <- mcmc_par$burn
thin <- mcmc_par$thin
totIter <- burn + iter*thin

# Create containers
store_size <- iter
tree_store <- vector("list", store_size)
sigma2_store <- rep(NA, store_size)
log_like_store <- rep(NA, store_size)
num_node <- rep(NA, store_size)
pred <- matrix(NA, nrow = store_size, ncol = length(y))
log_mar_tree <- rep(NA, store_size)

if(num_trees == 1 ){full_condition_store <- rep(NA, store_size)}else{
  full_condition_store <- matrix(NA, ncol = num_trees, nrow = store_size)
}


# Create a tree stump
```

```
curr_tree <- create_stump(num_trees = num_trees, y = y, x = x)[[1]]

# Initialize
new_tree <- curr_tree


pb = utils::txtProgressBar(min = 1, max = totIter,
                           style = 3, width = 60,
                           title = 'Running Models...')


# MCMC Iteration
for(i in 1:totIter){
  utils::setTxtProgressBar(pb, i)
  if((i > burn ) & ((i-burn)%%thin == 0 )){

    curr <- (i-burn)/thin
    tree_store[[curr]] <- curr_tree
    sigma2_store[curr] <- sigma
    log_like_store[curr] <- log_lik
    pred[curr,] <- prediction
    num_node[curr] <- num.node
    log_mar_tree[curr] <- log_mar_tree_curr
  }

  # Propose a new tree.
  # To prevent the tree from being too small, we need to grow at the beginning

  type <- sample(c("grow","prune","change","swap"), 1)

  if(i < max( floor(0.1*burn), 10) ){type = "grow"}

  # This is a proposed tree, but we need to figure out whether we should use this tree.
  new_tree <- update_tree(x,y, type = type, curr_tree = curr_tree, node_min_size = node_min_size)



  # New tree, compute the log of marginalized likelihood plus the log of the tree prior
  # First, subtract the mean of the tree




  # Use Metropolis-Hasting to sample a new tree here, by using the marginal distribution of tree

  # Get the new log probability
  l_new <- tree_marginal(new_tree, x, y, c, alpha_sig, beta_sig) + get_tree_prior(new_tree, alpha = al

  # Get the old log probability
  l_old <- tree_marginal(curr_tree, x, y, c, alpha_sig, beta_sig) + get_tree_prior(curr_tree, alpha =


  # Since the proposal is symmetric, we can record the transition probability as follows:
  a <- exp(l_new - l_old)
```

```
  l <- runif(1)




  # Need to save the full conditional to check the convergence
  if(  (i>burn) & ( ((i-burn)%%thin) ==0) ) {full_condition_store[curr] <- l_old}

  if(a > l){curr_tree <- new_tree }

  # Then, we should update the mu for each terminal node and update the sigma

  curr_tree <- fill_update_mu(y, sigma = sigma, c = c, curr_tree = curr_tree)

  mu <- curr_tree$info[which(as.numeric(curr_tree$info[,"terminal"])==1),"mu"]

  sigma <- sample_sigma(y, mu = mu, alpha_sig, beta_sig, tree = curr_tree, c)

  # Get the log likelihood for the model

  log_lik <- get_like(curr_tree = curr_tree, x, y, sigma = sigma)


  #For now, I will also get the prediction of this iteration
  # We have subtracted mu in the previous step

  all_mean <- curr_tree$info[,"mu"]
  pred_mean <- all_mean[curr_tree$node]
  prediction_scaled <- rnorm(length(y), mean = pred_mean, sd = sqrt(sigma))
  prediction <- prediction_scaled*scale_y + center_y


  num.node <- sum(as.numeric(curr_tree$info[,"terminal"]==1))

  log_mar_tree_curr <- tree_marginal(curr_tree, x, y, c, alpha_sig, beta_sig) + get_tree_prior(curr_tr

}
# End the iteration loop
cat('\n')

return(list(
  tree = tree_store,
  sigma2_scaled = sigma2_store,
  log_like = log_like_store,
  center_x = center_x,
  scale_x = scale_x,
  center_y = center_y,
  scale_y = scale_y,
  iter = iter,
  burn = burn,
  thin = thin,
  store_size = store_size,
  prediction = pred,
```

```
    num_node = num_node,
    log_mar_tree = log_mar_tree

  ))


}
```

```
get_like <- function(curr_tree, x, y, sigma){

  # Find which node is terminal node and find its mean

  terminal_node <- which(as.numeric(curr_tree$info[,"terminal"])==1)
  terminal_mean <- curr_tree$info[terminal_node, "mu"]

  res <- 0

  for (i in 1:length(terminal_node)) {

    group_index <- which(curr_tree$node == terminal_node[i])
    res <- res + sum(dnorm(y[group_index], mean = terminal_mean[i], sd = sqrt(sigma), log = TRUE))

  }

  return(res)


}
```

12. Simulated Data Set

A same simulated model as the paper presented was as follows:

$$Y = f(X_1, X_2) + 2\epsilon,\ \epsilon \sim N(0,1)$$

$$f(X_1, X_2) = \begin{cases} 8.0 \;\; if \;\; X_1 \leq 5.0 \; and \; X_2 \in \{A, B\} \\ 2.0 \;\; if \;\; X_1 > 5.0 \; and \; X_2 \in \{A, B\} \\ 1.0 \;\; if \;\; X_1 \leq 3.0 \; and \; X_2 \in \{C, D\} \\ 5.0 \;\; if \;\; 3.0 < X_1 \leq 7.0 \; and \; X_2 \in \{C, D\} \\ 8.0 \;\; if \;\; X_1 > 7.0 \; and \; X_2 \in \{C, D\} \end{cases}$$

Let X_1 be the sequence from 0.1 to 10, with a break of 0.1, 100 in total. Let X_2 be a repeated uniform random sample from A, B, C, D.

```
set.seed(0)
x_grid <- seq(from = 0.5, to = 9.5, by = 1)
x1 <- sample(x_grid, 1000, replace = TRUE)
x2 <- sample(1:4, replace = TRUE, size = 1000)
y <- rep(NA, 1000)
res <- 2*rnorm(1000, mean = 0, sd = 1)

y[ x1<=5 & (x2==1 | x2==2)] <- 8
```

```
y[ x1>5 & (x2==1 | x2==2)] <- 2
y[ x1<=3 & (x2==3 | x2==4)] <- 1
y[ (x1>3 & x1<=7) & (x2==3 | x2==4)] <- 5
y[ x1>7 & (x2==3 | x2==4)] <- 8

true_mean <- y

cat <- c("A","B","C","D")
x2 <- cat[x2]
sim_dat <- data.frame(y = as.numeric(y+res), x1= as.numeric(x1), x2=x2)
sim_x <- model.matrix(~ -1 + sim_dat[,2]+sim_dat[,3])
sim_y <- sim_dat[,1]
```

13. Simulated Prediction

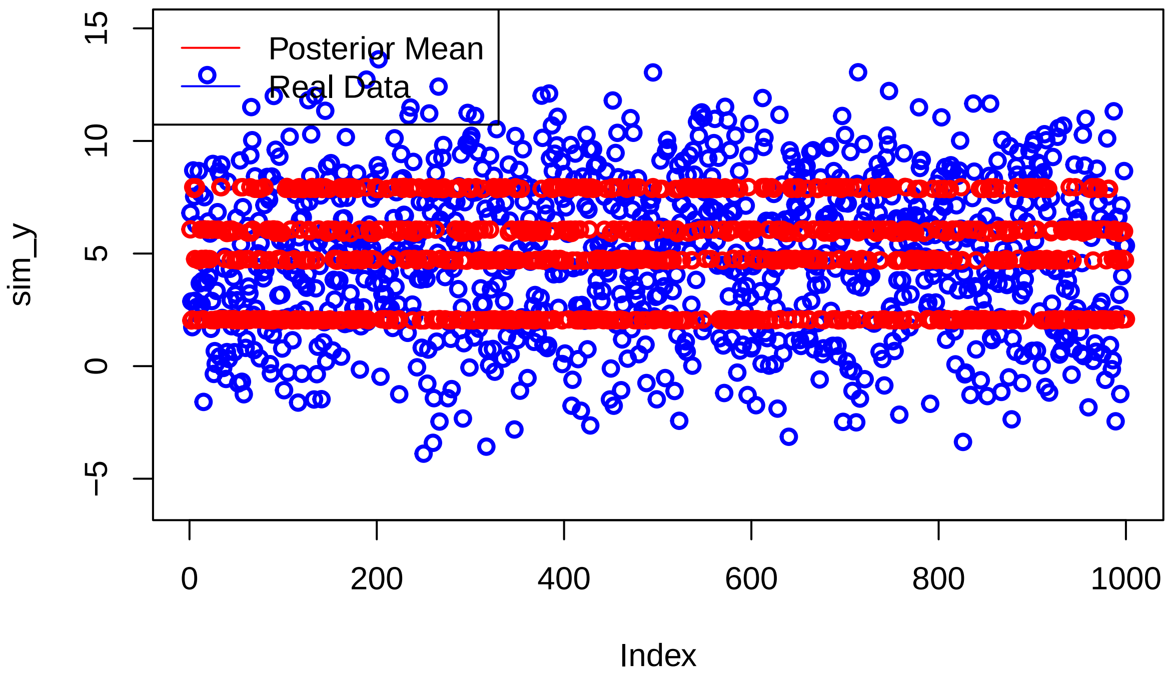
```
sim_res <- train_bart(x = sim_x, y = sim_y, mcmc_par = list(iter = 3000, burn = 0, thin = 2),hyper = lis
```

```
##   |                                                                      |
```

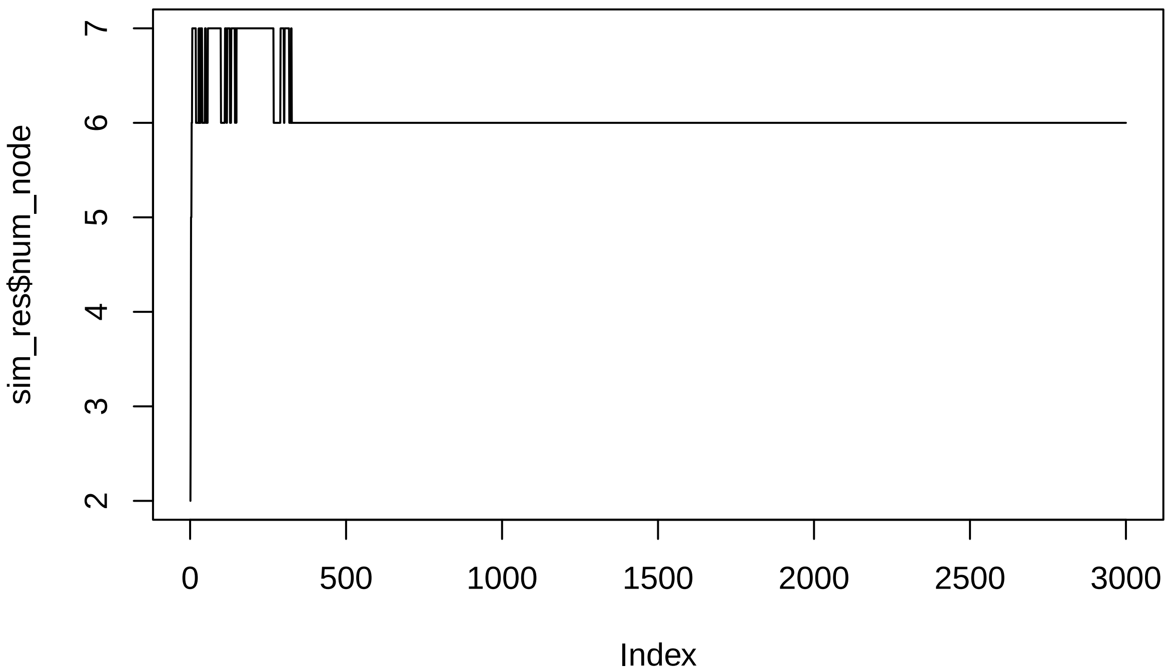
```
sim_res$tree[[10]]$info[which(sim_res$tree[[10]]$info[,"terminal"]==1),"mu"]*sim_res$scale_y + sim_res$
```

```
## [1] 6.065155 7.961815 1.995636 2.050615 7.872657 4.686395 5.676082
```

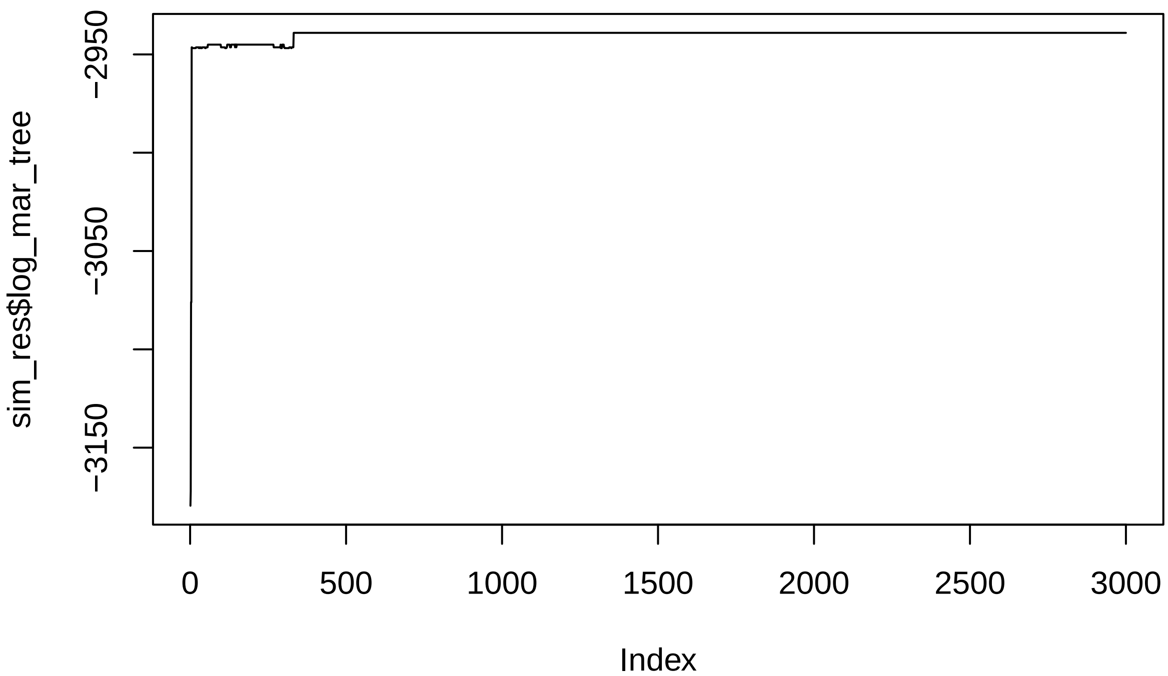
```
pred_y_mean_all <- apply(sim_res$prediction, 2, mean)

plot(sim_y, type = "p", col = "blue", lwd = 2, ylim = c(-6, 15))
lines(pred_y_mean_all, type = "p", col = "red", lwd = 2)
legend("topleft",c("Posterior Mean","Real Data"), col = c("red","blue"), lty = c(1,1))
```



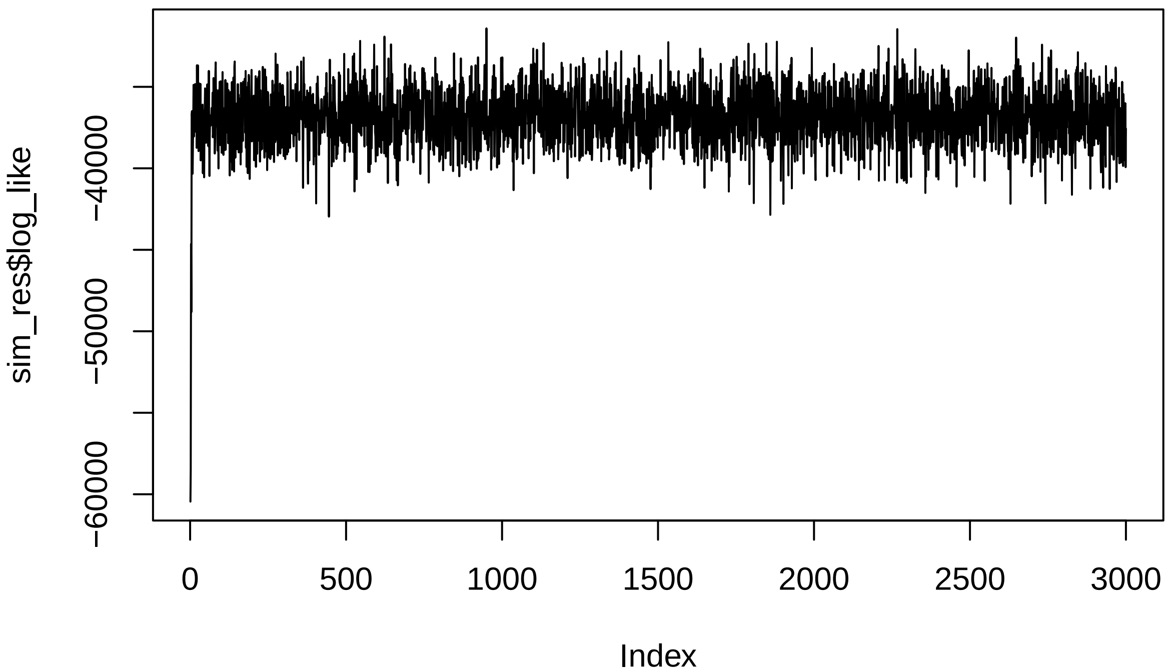

```
plot(sim_res$num_node, type = 'l')
```




```
plot(sim_res$log_mar_tree, type = 'l')
```




```r
plot(sim_res$log_like, type = 'l')
```

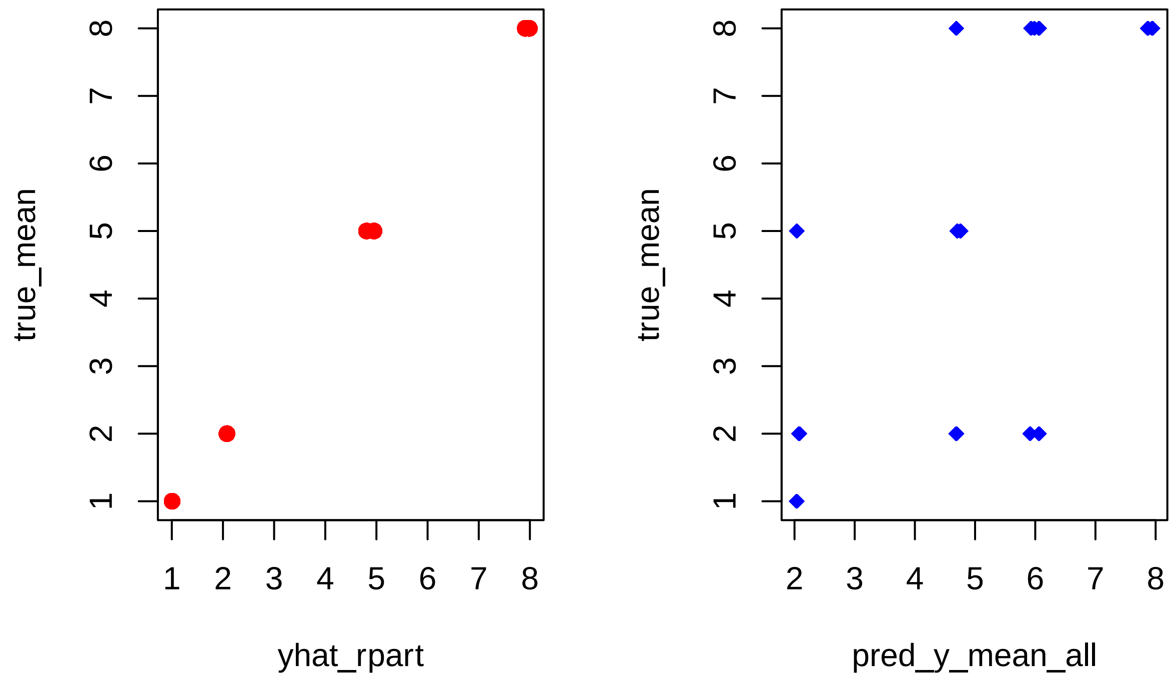


It seems that due to the big residual I added, the overall trend of y is not so obvious, however, I changed the residual to have a normal with mean 0 and var 0.01 and tried again, the result is as follows.

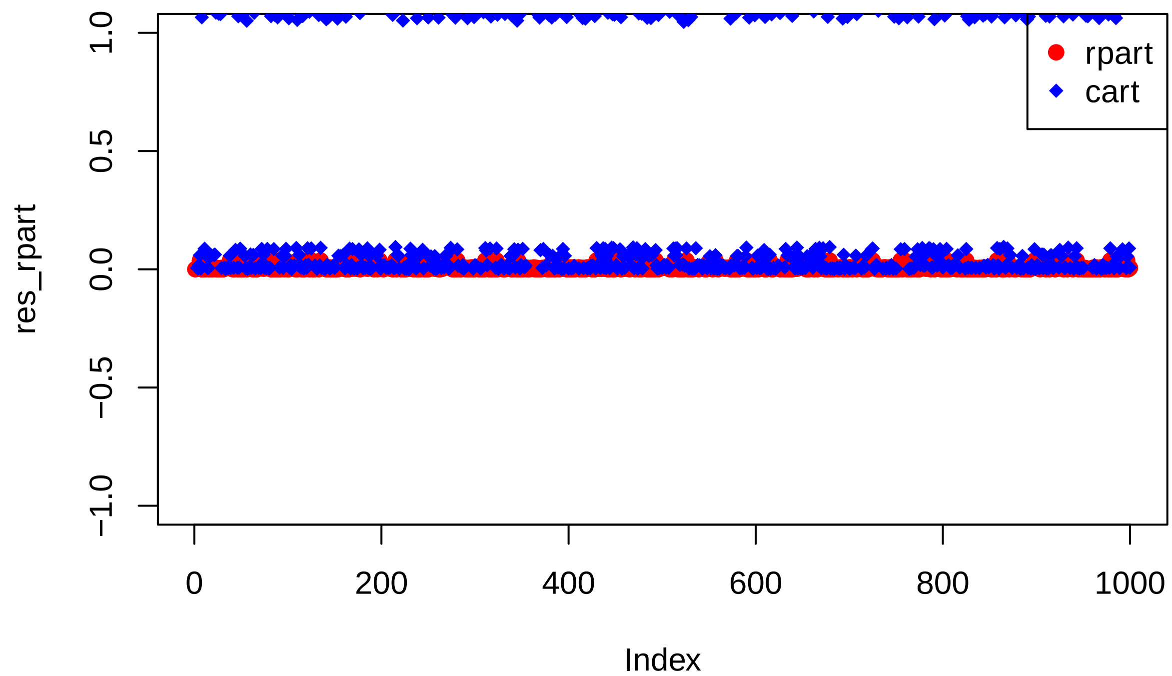
14. Compare with rpart(All Data Included as Train Set)

In this section, I am going to compare the prediction squared error with the rpart function.

```
library(rpart)
rmod <- rpart(y~., data = sim_dat)
yhat_rpart <- predict(rmod, sim_dat[,2:3])

par(mfrow = c(1,2))
#plot(yhat_rpart, type = 'p', pch = 19, col = "red")
#lines(pred_y_mean_all, type = 'p', pch = 18, col = "blue")
#legend("topright", c("rpart","cart"), pch = c(19,18), col = c("red","blue"))


plot(x = yhat_rpart, y = true_mean, type = 'p', pch = 19, col = "red")
plot(x = pred_y_mean_all, y = true_mean, type = 'p', pch = 18, col = "blue")
```




```
res_rpart <- (true_mean - yhat_rpart)^2
res_cart <- (pred_y_mean_all - true_mean)^2
plot(res_rpart, type = 'p', pch = 19, col = "red", ylim = c(-1,1))
lines(res_cart, type = 'p', pch = 18, col = "blue")
legend("topright", c("rpart","cart"), pch = c(19,18), col = c("red","blue"))
```




```
sum(res_rpart)
```

```
## [1] 7.659329
```

```
sum(res_cart)
```

```
## [1] 3035.409
```

15. Result of rbart Package

```
library(rbart)
bart_res <- rbart(sim_x, sim_y, ntree = 1, nskip = 2000, ndpost = 1000, k = 2, power = 2, base = 0.95)
```

```
## *****Into main of cpsambrt
## row, cols chvm: 5, 5
## **********************
## n: 1000
## p: 5
## first row: 8.500000, 0.000000
## second row: 3.500000, 0.000000
## last row: 9.500000, 0.000000
```

```
## first and last y: 1.895345, 0.441644
## no test observations
## number of trees mean: 1
## number of trees stan dev: 40
## tau: 4.378070
## overalllambda: 11.978146
## overallnu: 10.000000
## burn (nskip): 2000
## nd (ndpost): 1000
## nadapt: 1000
## adaptevery: 100
## mean tree prior base: 0.950000
## mean tree prior power: 2.000000
## variance tree prior base: 0.950000
## variance tree prior power: 2.000000
## thread count: 1
## first and last sigmav: 1.000000, 1.000000
## chgv first row: 1.000000, 0.068920
## chgv last row: 0.068920, 1.000000
## mean trees prob birth/death: 0.700000
## mean trees prob birth: 0.500000
## variance trees prob birth/death: 0.700000
## variance trees prob birth: 0.500000
## mean trees initial step width pert move: 0.100000
## variance trees initial step width pert move: 0.100000
## mean trees prob of a change var move : 0.100000
## variance trees prob of a change var move : 0.100000
## mean trees min num obs in bottom node: 5
## variance trees min num obs in bottom node: 5
## *****printevery: 100
## &&& made xinfo
## Variable 0 has numcuts=100 : 0.589109 ... 9.41089
## Variable 1 has numcuts=100 : 0.00990099 ... 0.990099
## Variable 2 has numcuts=100 : 0.00990099 ... 0.990099
## Variable 3 has numcuts=100 : 0.00990099 ... 0.990099
## Variable 4 has numcuts=100 : 0.00990099 ... 0.990099
## Starting MCMC...
##
## Adapt ambrt[0]:pert_rate=0.933884 pertalpha=0.212246 chgv_rate=0.5   m_rate=0.14
## Adapt sbrt[0]:pert_rate=0.990826 pertalpha=0.225188 chgv_rate=0.888889   m_rate=0.24
## Adapt sbrt[1]:pert_rate=0.976471 pertalpha=0.221925 chgv_rate=0.923077   m_rate=0.26
## Adapt sbrt[2]:pert_rate=0.972222 pertalpha=0.22096 chgv_rate=0.555556   m_rate=0.26
## Adapt sbrt[3]:pert_rate=0.947917 pertalpha=0.215436 chgv_rate=0.666667   m_rate=0.35
## Adapt sbrt[4]:pert_rate=0.978947 pertalpha=0.222488 chgv_rate=0.785714   m_rate=0.24
## Adapt sbrt[5]:pert_rate=0.990476 pertalpha=0.225108 chgv_rate=0.785714   m_rate=0.16
## Adapt sbrt[6]:pert_rate=0.989583 pertalpha=0.224905 chgv_rate=0.615385   m_rate=0.25
## Adapt sbrt[7]:pert_rate=0.965909 pertalpha=0.219525 chgv_rate=0.6875   m_rate=0.22
## Adapt sbrt[8]:pert_rate=0.954023 pertalpha=0.216823 chgv_rate=0.846154   m_rate=0.18
## Adapt sbrt[9]:pert_rate=0.978495 pertalpha=0.222385 chgv_rate=0.7   m_rate=0.23
## Adapt sbrt[10]:pert_rate=0.978947 pertalpha=0.222488 chgv_rate=0.454545   m_rate=0.19
## Adapt sbrt[11]:pert_rate=0.980392 pertalpha=0.222816 chgv_rate=0.444444   m_rate=0.24
## Adapt sbrt[12]:pert_rate=0.941176 pertalpha=0.213904 chgv_rate=0.642857   m_rate=0.22
## Adapt sbrt[13]:pert_rate=0.927711 pertalpha=0.210843 chgv_rate=0.789474   m_rate=0.14
## Adapt sbrt[14]:pert_rate=0.970874 pertalpha=0.220653 chgv_rate=0.529412   m_rate=0.24
```

```
## Adapt sbrt[15]:pert_rate=0.96 pertalpha=0.218182 chgv_rate=0.764706   m_rate=0.22
## Adapt sbrt[16]:pert_rate=0.957895 pertalpha=0.217703 chgv_rate=0.769231   m_rate=0.27
## Adapt sbrt[17]:pert_rate=0.98913 pertalpha=0.224802 chgv_rate=0.714286   m_rate=0.14
## Adapt sbrt[18]:pert_rate=0.92029 pertalpha=0.209157 chgv_rate=0.625   m_rate=0.31
## Adapt sbrt[19]:pert_rate=0.956989 pertalpha=0.217498 chgv_rate=0.875   m_rate=0.31
## Adapt sbrt[20]:pert_rate=0.988636 pertalpha=0.22469 chgv_rate=0.75   m_rate=0.24
## Adapt sbrt[21]:pert_rate=0.978495 pertalpha=0.222385 chgv_rate=0.625   m_rate=0.41
## Adapt sbrt[22]:pert_rate=0.989247 pertalpha=0.224829 chgv_rate=0.769231   m_rate=0.25
## Adapt sbrt[23]:pert_rate=0.979798 pertalpha=0.222681 chgv_rate=0.636364   m_rate=0.26
## Adapt sbrt[24]:pert_rate=0.958333 pertalpha=0.217803 chgv_rate=0.769231   m_rate=0.27
## Adapt sbrt[25]:pert_rate=0.987013 pertalpha=0.224321 chgv_rate=0.846154   m_rate=0.22
## Adapt sbrt[26]:pert_rate=0.943396 pertalpha=0.214408 chgv_rate=0.7   m_rate=0.32
## Adapt sbrt[27]:pert_rate=0.933962 pertalpha=0.212264 chgv_rate=0.863636   m_rate=0.18
## Adapt sbrt[28]:pert_rate=0.880952 pertalpha=0.200216 chgv_rate=0.470588   m_rate=0.24
## Adapt sbrt[29]:pert_rate=0.958904 pertalpha=0.217933 chgv_rate=0.75   m_rate=0.33
## Adapt sbrt[30]:pert_rate=0.982759 pertalpha=0.223354 chgv_rate=0.722222   m_rate=0.21
## Adapt sbrt[31]:pert_rate=0.945055 pertalpha=0.214785 chgv_rate=0.846154   m_rate=0.34
## Adapt sbrt[32]:pert_rate=0.978261 pertalpha=0.222332 chgv_rate=0.714286   m_rate=0.26
## Adapt sbrt[33]:pert_rate=0.955056 pertalpha=0.217058 chgv_rate=0.933333   m_rate=0.33
## Adapt sbrt[34]:pert_rate=0.975 pertalpha=0.221591 chgv_rate=0.5   m_rate=0.22
## Adapt sbrt[35]:pert_rate=0.988636 pertalpha=0.22469 chgv_rate=0.875   m_rate=0.21
## Adapt sbrt[36]:pert_rate=0.96748 pertalpha=0.219882 chgv_rate=0.6   m_rate=0.25
## Adapt sbrt[37]:pert_rate=0.95 pertalpha=0.215909 chgv_rate=0.846154   m_rate=0.29
## Adapt sbrt[38]:pert_rate=0.970803 pertalpha=0.220637 chgv_rate=0.5625   m_rate=0.27
## Adapt sbrt[39]:pert_rate=0.989011 pertalpha=0.224775 chgv_rate=0.785714   m_rate=0.24
## Adapt ambrt[0]:pert_rate=0.892608 pertalpha=0.430575 chgv_rate=0.410072   m_rate=0.0679612
## Adapt sbrt[0]:pert_rate=0.946809 pertalpha=0.484567 chgv_rate=0.888889   m_rate=0.213592
## Adapt sbrt[1]:pert_rate=0.91453 pertalpha=0.461266 chgv_rate=0.857143   m_rate=0.291262
## Adapt sbrt[2]:pert_rate=0.925532 pertalpha=0.464784 chgv_rate=0.769231   m_rate=0.15534
## Adapt sbrt[3]:pert_rate=0.958333 pertalpha=0.469225 chgv_rate=0.857143   m_rate=0.174757
## Adapt sbrt[4]:pert_rate=0.954128 pertalpha=0.482459 chgv_rate=0.733333   m_rate=0.252427
## Adapt sbrt[5]:pert_rate=0.943182 pertalpha=0.482541 chgv_rate=0.8   m_rate=0.242718
## Adapt sbrt[6]:pert_rate=0.965986 pertalpha=0.493762 chgv_rate=0.771429   m_rate=0.223301
## Adapt sbrt[7]:pert_rate=0.920455 pertalpha=0.459233 chgv_rate=0.677419   m_rate=0.165049
## Adapt sbrt[8]:pert_rate=0.945652 pertalpha=0.465999 chgv_rate=0.842105   m_rate=0.213592
## Adapt sbrt[9]:pert_rate=0.988235 pertalpha=0.499475 chgv_rate=0.769231   m_rate=0.135922
## Adapt sbrt[10]:pert_rate=0.943396 pertalpha=0.477033 chgv_rate=0.684211   m_rate=0.252427
## Adapt sbrt[11]:pert_rate=0.919355 pertalpha=0.465562 chgv_rate=0.653846   m_rate=0.300971
## Adapt sbrt[12]:pert_rate=0.978723 pertalpha=0.475801 chgv_rate=0.761905   m_rate=0.165049
## Adapt sbrt[13]:pert_rate=0.952381 pertalpha=0.456371 chgv_rate=0.774194   m_rate=0.23301
## Adapt sbrt[14]:pert_rate=0.895652 pertalpha=0.449156 chgv_rate=0.631579   m_rate=0.281553
## Adapt sbrt[15]:pert_rate=0.934579 pertalpha=0.463428 chgv_rate=0.777778   m_rate=0.300971
## Adapt sbrt[16]:pert_rate=0.955556 pertalpha=0.47279 chgv_rate=0.789474   m_rate=0.194175
## Adapt sbrt[17]:pert_rate=0.957447 pertalpha=0.489173 chgv_rate=0.833333   m_rate=0.242718
## Adapt sbrt[18]:pert_rate=0.977011 pertalpha=0.464429 chgv_rate=0.6875   m_rate=0.126214
## Adapt sbrt[19]:pert_rate=0.946429 pertalpha=0.467832 chgv_rate=0.846154   m_rate=0.223301
## Adapt sbrt[20]:pert_rate=0.962264 pertalpha=0.491389 chgv_rate=0.730769   m_rate=0.242718
## Adapt sbrt[21]:pert_rate=0.971963 pertalpha=0.49125 chgv_rate=0.666667   m_rate=0.242718
## Adapt sbrt[22]:pert_rate=0.947917 pertalpha=0.484362 chgv_rate=0.807692   m_rate=0.213592
## Adapt sbrt[23]:pert_rate=0.938462 pertalpha=0.47495 chgv_rate=0.791667   m_rate=0.31068
## Adapt sbrt[24]:pert_rate=0.978495 pertalpha=0.484362 chgv_rate=0.842105   m_rate=0.0970874
## Adapt sbrt[25]:pert_rate=0.944444 pertalpha=0.481497 chgv_rate=0.928571   m_rate=0.242718
## Adapt sbrt[26]:pert_rate=0.967391 pertalpha=0.471402 chgv_rate=0.72   m_rate=0.15534
## Adapt sbrt[27]:pert_rate=0.972727 pertalpha=0.469262 chgv_rate=0.787879   m_rate=0.252427
```

```
## Adapt sbrt[28]:pert_rate=0.958333 pertalpha=0.436077 chgv_rate=0.583333   m_rate=0.213592
## Adapt sbrt[29]:pert_rate=0.959596 pertalpha=0.47529 chgv_rate=0.69697   m_rate=0.15534
## Adapt sbrt[30]:pert_rate=0.956897 pertalpha=0.485743 chgv_rate=0.740741   m_rate=0.203883
## Adapt sbrt[31]:pert_rate=0.918605 pertalpha=0.448415 chgv_rate=0.857143   m_rate=0.203883
## Adapt sbrt[32]:pert_rate=0.960784 pertalpha=0.485484 chgv_rate=0.809524   m_rate=0.213592
## Adapt sbrt[33]:pert_rate=0.942149 pertalpha=0.464775 chgv_rate=0.782609   m_rate=0.194175
## Adapt sbrt[34]:pert_rate=0.921348 pertalpha=0.464005 chgv_rate=0.777778   m_rate=0.184466
## Adapt sbrt[35]:pert_rate=0.956044 pertalpha=0.488213 chgv_rate=0.944444   m_rate=0.203883
## Adapt sbrt[36]:pert_rate=0.908163 pertalpha=0.453838 chgv_rate=0.826087   m_rate=0.15534
## Adapt sbrt[37]:pert_rate=0.951923 pertalpha=0.467111 chgv_rate=0.84   m_rate=0.23301
## Adapt sbrt[38]:pert_rate=0.933333 pertalpha=0.468018 chgv_rate=0.705882   m_rate=0.184466
## Adapt sbrt[39]:pert_rate=0.898876 pertalpha=0.459194 chgv_rate=0.821429   m_rate=0.145631
## Adapt ambrt[0]:pert_rate=0.826519 pertalpha=0.808814 chgv_rate=0.468085   m_rate=0.0194175
## Adapt sbrt[0]:pert_rate=0.938053 pertalpha=1.03307 chgv_rate=0.884615   m_rate=0.223301
## Adapt sbrt[1]:pert_rate=0.952381 pertalpha=0.998412 chgv_rate=0.722222   m_rate=0.242718
## Adapt sbrt[2]:pert_rate=0.918367 pertalpha=0.970097 chgv_rate=0.72973   m_rate=0.0776699
## Adapt sbrt[3]:pert_rate=0.903226 pertalpha=0.963219 chgv_rate=0.90625   m_rate=0.174757
## Adapt sbrt[4]:pert_rate=0.869565 pertalpha=0.953477 chgv_rate=0.804878   m_rate=0.184466
## Adapt sbrt[5]:pert_rate=0.896907 pertalpha=0.983624 chgv_rate=0.883721   m_rate=0.23301
## Adapt sbrt[6]:pert_rate=0.929412 pertalpha=1.04297 chgv_rate=0.785714   m_rate=0.174757
## Adapt sbrt[7]:pert_rate=0.966667 pertalpha=1.00892 chgv_rate=0.733333   m_rate=0.174757
## Adapt sbrt[8]:pert_rate=0.907216 pertalpha=0.960823 chgv_rate=0.814815   m_rate=0.184466
## Adapt sbrt[9]:pert_rate=0.9125 pertalpha=1.03584 chgv_rate=0.827586   m_rate=0.106796
## Adapt sbrt[10]:pert_rate=0.903226 pertalpha=0.979246 chgv_rate=0.677419   m_rate=0.242718
## Adapt sbrt[11]:pert_rate=0.857143 pertalpha=0.906939 chgv_rate=0.742857   m_rate=0.23301
## Adapt sbrt[12]:pert_rate=0.915888 pertalpha=0.990411 chgv_rate=0.8   m_rate=0.223301
## Adapt sbrt[13]:pert_rate=0.896226 pertalpha=0.929572 chgv_rate=0.75   m_rate=0.271845
## Adapt sbrt[14]:pert_rate=0.881356 pertalpha=0.899695 chgv_rate=0.698113   m_rate=0.23301
## Adapt sbrt[15]:pert_rate=0.879121 pertalpha=0.92593 chgv_rate=0.806452   m_rate=0.223301
## Adapt sbrt[16]:pert_rate=0.954545 pertalpha=1.02568 chgv_rate=0.708333   m_rate=0.174757
## Adapt sbrt[17]:pert_rate=0.896296 pertalpha=0.996464 chgv_rate=0.851852   m_rate=0.252427
## Adapt sbrt[18]:pert_rate=0.94382 pertalpha=0.996221 chgv_rate=0.717949   m_rate=0.0679612
## Adapt sbrt[19]:pert_rate=0.884615 pertalpha=0.940571 chgv_rate=0.790698   m_rate=0.398058
## Adapt sbrt[20]:pert_rate=0.867347 pertalpha=0.968647 chgv_rate=0.787879   m_rate=0.23301
## Adapt sbrt[21]:pert_rate=0.896104 pertalpha=1.00048 chgv_rate=0.666667   m_rate=0.145631
## Adapt sbrt[22]:pert_rate=0.933333 pertalpha=1.02743 chgv_rate=0.818182   m_rate=0.213592
## Adapt sbrt[23]:pert_rate=0.918919 pertalpha=0.99191 chgv_rate=0.823529   m_rate=0.213592
## Adapt sbrt[24]:pert_rate=0.956044 pertalpha=1.05243 chgv_rate=0.862069   m_rate=0.0776699
## Adapt sbrt[25]:pert_rate=0.87037 pertalpha=0.952457 chgv_rate=0.939394   m_rate=0.194175
## Adapt sbrt[26]:pert_rate=0.925926 pertalpha=0.992007 chgv_rate=0.777778   m_rate=0.135922
## Adapt sbrt[27]:pert_rate=0.886364 pertalpha=0.94531 chgv_rate=0.804348   m_rate=0.165049
## Adapt sbrt[28]:pert_rate=0.932692 pertalpha=0.924378 chgv_rate=0.621622   m_rate=0.184466
## Adapt sbrt[29]:pert_rate=0.880734 pertalpha=0.951372 chgv_rate=0.731707   m_rate=0.213592
## Adapt sbrt[30]:pert_rate=0.878049 pertalpha=0.969332 chgv_rate=0.771429   m_rate=0.15534
## Adapt sbrt[31]:pert_rate=0.878788 pertalpha=0.895595 chgv_rate=0.862069   m_rate=0.252427
## Adapt sbrt[32]:pert_rate=0.896226 pertalpha=0.988872 chgv_rate=0.833333   m_rate=0.174757
## Adapt sbrt[33]:pert_rate=0.934066 pertalpha=0.986661 chgv_rate=0.72973   m_rate=0.213592
## Adapt sbrt[34]:pert_rate=0.93617 pertalpha=0.987246 chgv_rate=0.794872   m_rate=0.184466
## Adapt sbrt[35]:pert_rate=0.912088 pertalpha=1.01203 chgv_rate=0.92   m_rate=0.223301
## Adapt sbrt[36]:pert_rate=0.9375 pertalpha=0.966983 chgv_rate=0.848485   m_rate=0.0873786
## Adapt sbrt[37]:pert_rate=0.858209 pertalpha=0.911088 chgv_rate=0.904762   m_rate=0.262136
## Adapt sbrt[38]:pert_rate=0.970297 pertalpha=1.03208 chgv_rate=0.688889   m_rate=0.194175
## Adapt sbrt[39]:pert_rate=0.939394 pertalpha=0.980372 chgv_rate=0.794118   m_rate=0.213592
## Adapt ambrt[0]:pert_rate=0.701794 pertalpha=1.29005 chgv_rate=0.446866   m_rate=0.0485437
```

```
## Adapt sbrt[0]:pert_rate=0.915254 pertalpha=2 chgv_rate=0.9   m_rate=0.203883
## Adapt sbrt[1]:pert_rate=0.84375 pertalpha=1.91457 chgv_rate=0.761905   m_rate=0.213592
## Adapt sbrt[2]:pert_rate=0.904762 pertalpha=1.99479 chgv_rate=0.788462   m_rate=0.203883
## Adapt sbrt[3]:pert_rate=0.930769 pertalpha=2 chgv_rate=0.886364   m_rate=0.320388
## Adapt sbrt[4]:pert_rate=0.876289 pertalpha=1.89891 chgv_rate=0.830189   m_rate=0.271845
## Adapt sbrt[5]:pert_rate=0.882353 pertalpha=1.97251 chgv_rate=0.857143   m_rate=0.291262
## Adapt sbrt[6]:pert_rate=0.913978 pertalpha=2 chgv_rate=0.78   m_rate=0.213592
## Adapt sbrt[7]:pert_rate=0.739583 pertalpha=1.69587 chgv_rate=0.732143   m_rate=0.126214
## Adapt sbrt[8]:pert_rate=0.9 pertalpha=1.96532 chgv_rate=0.852941   m_rate=0.0970874
## Adapt sbrt[9]:pert_rate=0.906667 pertalpha=2 chgv_rate=0.844444   m_rate=0.15534
## Adapt sbrt[10]:pert_rate=0.859813 pertalpha=1.91356 chgv_rate=0.738095   m_rate=0.252427
## Adapt sbrt[11]:pert_rate=0.913386 pertalpha=1.88269 chgv_rate=0.744186   m_rate=0.23301
## Adapt sbrt[12]:pert_rate=0.79 pertalpha=1.77824 chgv_rate=0.785714   m_rate=0.145631
## Adapt sbrt[13]:pert_rate=0.89899 pertalpha=1.89926 chgv_rate=0.765957   m_rate=0.23301
## Adapt sbrt[14]:pert_rate=0.861702 pertalpha=1.76198 chgv_rate=0.704918   m_rate=0.194175
## Adapt sbrt[15]:pert_rate=0.879121 pertalpha=1.85001 chgv_rate=0.820513   m_rate=0.184466
## Adapt sbrt[16]:pert_rate=0.91358 pertalpha=2 chgv_rate=0.820513   m_rate=0.194175
## Adapt sbrt[17]:pert_rate=0.885417 pertalpha=2 chgv_rate=0.742857   m_rate=0.174757
## Adapt sbrt[18]:pert_rate=0.857143 pertalpha=1.94069 chgv_rate=0.764706   m_rate=0.223301
## Adapt sbrt[19]:pert_rate=0.881188 pertalpha=1.88368 chgv_rate=0.814815   m_rate=0.203883
## Adapt sbrt[20]:pert_rate=0.886364 pertalpha=1.9513 chgv_rate=0.789474   m_rate=0.23301
## Adapt sbrt[21]:pert_rate=0.918367 pertalpha=2 chgv_rate=0.772727   m_rate=0.223301
## Adapt sbrt[22]:pert_rate=0.839623 pertalpha=1.96058 chgv_rate=0.767442   m_rate=0.23301
## Adapt sbrt[23]:pert_rate=0.891304 pertalpha=2 chgv_rate=0.804348   m_rate=0.300971
## Adapt sbrt[24]:pert_rate=0.846939 pertalpha=2 chgv_rate=0.871795   m_rate=0.23301
## Adapt sbrt[25]:pert_rate=0.87234 pertalpha=1.88833 chgv_rate=0.931818   m_rate=0.145631
## Adapt sbrt[26]:pert_rate=0.88764 pertalpha=2 chgv_rate=0.795455   m_rate=0.184466
## Adapt sbrt[27]:pert_rate=0.891089 pertalpha=1.91444 chgv_rate=0.8   m_rate=0.194175
## Adapt sbrt[28]:pert_rate=0.826531 pertalpha=1.73642 chgv_rate=0.681818   m_rate=0.223301
## Adapt sbrt[29]:pert_rate=0.857143 pertalpha=1.85332 chgv_rate=0.8   m_rate=0.23301
## Adapt sbrt[30]:pert_rate=0.866667 pertalpha=1.90929 chgv_rate=0.823529   m_rate=0.271845
## Adapt sbrt[31]:pert_rate=0.892473 pertalpha=1.81658 chgv_rate=0.891892   m_rate=0.174757
## Adapt sbrt[32]:pert_rate=0.903226 pertalpha=2 chgv_rate=0.854167   m_rate=0.194175
## Adapt sbrt[33]:pert_rate=0.857143 pertalpha=1.92207 chgv_rate=0.777778   m_rate=0.0970874
## Adapt sbrt[34]:pert_rate=0.877551 pertalpha=1.969 chgv_rate=0.8   m_rate=0.184466
## Adapt sbrt[35]:pert_rate=0.86087 pertalpha=1.98006 chgv_rate=0.945946   m_rate=0.184466
## Adapt sbrt[36]:pert_rate=0.849057 pertalpha=1.86596 chgv_rate=0.888889   m_rate=0.23301
## Adapt sbrt[37]:pert_rate=0.872483 pertalpha=1.80661 chgv_rate=0.844828   m_rate=0.203883
## Adapt sbrt[38]:pert_rate=0.896 pertalpha=2 chgv_rate=0.678571   m_rate=0.262136
## Adapt sbrt[39]:pert_rate=0.831683 pertalpha=1.85309 chgv_rate=0.8   m_rate=0.300971
## Adapt ambrt[0]:pert_rate=0.674723 pertalpha=1.97824 chgv_rate=0.44   m_rate=0.0194175
## Adapt sbrt[0]:pert_rate=0.977778 pertalpha=2 chgv_rate=0.862745   m_rate=0.223301
## Adapt sbrt[1]:pert_rate=0.940594 pertalpha=2 chgv_rate=0.75   m_rate=0.223301
## Adapt sbrt[2]:pert_rate=0.94898 pertalpha=2 chgv_rate=0.822581   m_rate=0.203883
## Adapt sbrt[3]:pert_rate=0.95098 pertalpha=2 chgv_rate=0.884615   m_rate=0.252427
## Adapt sbrt[4]:pert_rate=0.978022 pertalpha=2 chgv_rate=0.848485   m_rate=0.165049
## Adapt sbrt[5]:pert_rate=0.898305 pertalpha=2 chgv_rate=0.852941   m_rate=0.242718
## Adapt sbrt[6]:pert_rate=0.966102 pertalpha=2 chgv_rate=0.789474   m_rate=0.281553
## Adapt sbrt[7]:pert_rate=0.968085 pertalpha=2 chgv_rate=0.754098   m_rate=0.165049
## Adapt sbrt[8]:pert_rate=0.975207 pertalpha=2 chgv_rate=0.866667   m_rate=0.23301
## Adapt sbrt[9]:pert_rate=0.986301 pertalpha=2 chgv_rate=0.862069   m_rate=0.23301
## Adapt sbrt[10]:pert_rate=0.946809 pertalpha=2 chgv_rate=0.769231   m_rate=0.135922
## Adapt sbrt[11]:pert_rate=0.972222 pertalpha=2 chgv_rate=0.767857   m_rate=0.262136
## Adapt sbrt[12]:pert_rate=0.976471 pertalpha=2 chgv_rate=0.764706   m_rate=0.223301
```

```
## Adapt sbrt[13]:pert_rate=0.888889 pertalpha=2 chgv_rate=0.785714   m_rate=0.184466
## Adapt sbrt[14]:pert_rate=0.987342 pertalpha=2 chgv_rate=0.75   m_rate=0.194175
## Adapt sbrt[15]:pert_rate=0.977528 pertalpha=2 chgv_rate=0.849057   m_rate=0.0873786
## Adapt sbrt[16]:pert_rate=0.954023 pertalpha=2 chgv_rate=0.829787   m_rate=0.300971
## Adapt sbrt[17]:pert_rate=0.964286 pertalpha=2 chgv_rate=0.772727   m_rate=0.194175
## Adapt sbrt[18]:pert_rate=0.917431 pertalpha=2 chgv_rate=0.784615   m_rate=0.223301
## Adapt sbrt[19]:pert_rate=0.983871 pertalpha=2 chgv_rate=0.849315   m_rate=0.174757
## Adapt sbrt[20]:pert_rate=0.962025 pertalpha=2 chgv_rate=0.788462   m_rate=0.194175
## Adapt sbrt[21]:pert_rate=0.963636 pertalpha=2 chgv_rate=0.788462   m_rate=0.23301
## Adapt sbrt[22]:pert_rate=0.909091 pertalpha=2 chgv_rate=0.773585   m_rate=0.23301
## Adapt sbrt[23]:pert_rate=0.97541 pertalpha=2 chgv_rate=0.807692   m_rate=0.223301
## Adapt sbrt[24]:pert_rate=0.963303 pertalpha=2 chgv_rate=0.823529   m_rate=0.271845
## Adapt sbrt[25]:pert_rate=0.924731 pertalpha=2 chgv_rate=0.842105   m_rate=0.242718
## Adapt sbrt[26]:pert_rate=0.989796 pertalpha=2 chgv_rate=0.777778   m_rate=0.223301
## Adapt sbrt[27]:pert_rate=0.955556 pertalpha=2 chgv_rate=0.80597   m_rate=0.126214
## Adapt sbrt[28]:pert_rate=0.923077 pertalpha=2 chgv_rate=0.727273   m_rate=0.223301
## Adapt sbrt[29]:pert_rate=0.976744 pertalpha=2 chgv_rate=0.782609   m_rate=0.184466
## Adapt sbrt[30]:pert_rate=0.825581 pertalpha=2 chgv_rate=0.810345   m_rate=0.271845
## Adapt sbrt[31]:pert_rate=0.960396 pertalpha=2 chgv_rate=0.886364   m_rate=0.0970874
## Adapt sbrt[32]:pert_rate=0.956522 pertalpha=2 chgv_rate=0.85   m_rate=0.281553
## Adapt sbrt[33]:pert_rate=0.94898 pertalpha=2 chgv_rate=0.777778   m_rate=0.213592
## Adapt sbrt[34]:pert_rate=0.923077 pertalpha=2 chgv_rate=0.823529   m_rate=0.174757
## Adapt sbrt[35]:pert_rate=0.988095 pertalpha=2 chgv_rate=0.96   m_rate=0.242718
## Adapt sbrt[36]:pert_rate=0.979798 pertalpha=2 chgv_rate=0.857143   m_rate=0.15534
## Adapt sbrt[37]:pert_rate=0.989362 pertalpha=2 chgv_rate=0.847222   m_rate=0.165049
## Adapt sbrt[38]:pert_rate=0.956522 pertalpha=2 chgv_rate=0.701493   m_rate=0.31068
## Adapt sbrt[39]:pert_rate=0.895238 pertalpha=2 chgv_rate=0.789474   m_rate=0.213592
## Adapt ambrt[0]:pert_rate=0.729293 pertalpha=2 chgv_rate=0.424915   m_rate=0.038835
## Adapt sbrt[0]:pert_rate=0.9875 pertalpha=2 chgv_rate=0.87931   m_rate=0.184466
## Adapt sbrt[1]:pert_rate=0.988372 pertalpha=2 chgv_rate=0.790323   m_rate=0.126214
## Adapt sbrt[2]:pert_rate=0.848837 pertalpha=2 chgv_rate=0.816901   m_rate=0.15534
## Adapt sbrt[3]:pert_rate=0.980198 pertalpha=2 chgv_rate=0.880597   m_rate=0.242718
## Adapt sbrt[4]:pert_rate=0.989474 pertalpha=2 chgv_rate=0.851351   m_rate=0.135922
## Adapt sbrt[5]:pert_rate=0.969072 pertalpha=2 chgv_rate=0.844156   m_rate=0.242718
## Adapt sbrt[6]:pert_rate=0.958333 pertalpha=2 chgv_rate=0.796875   m_rate=0.145631
## Adapt sbrt[7]:pert_rate=0.945652 pertalpha=2 chgv_rate=0.786667   m_rate=0.223301
## Adapt sbrt[8]:pert_rate=0.988235 pertalpha=2 chgv_rate=0.875   m_rate=0.23301
## Adapt sbrt[9]:pert_rate=0.967033 pertalpha=2 chgv_rate=0.875   m_rate=0.194175
## Adapt sbrt[10]:pert_rate=0.961905 pertalpha=2 chgv_rate=0.803279   m_rate=0.23301
## Adapt sbrt[11]:pert_rate=0.882979 pertalpha=2 chgv_rate=0.808824   m_rate=0.184466
## Adapt sbrt[12]:pert_rate=0.988372 pertalpha=2 chgv_rate=0.758621   m_rate=0.0970874
## Adapt sbrt[13]:pert_rate=0.971429 pertalpha=2 chgv_rate=0.769231   m_rate=0.242718
## Adapt sbrt[14]:pert_rate=0.976 pertalpha=2 chgv_rate=0.752941   m_rate=0.184466
## Adapt sbrt[15]:pert_rate=0.97561 pertalpha=2 chgv_rate=0.8125   m_rate=0.262136
## Adapt sbrt[16]:pert_rate=0.990099 pertalpha=2 chgv_rate=0.82   m_rate=0.203883
## Adapt sbrt[17]:pert_rate=0.952381 pertalpha=2 chgv_rate=0.754386   m_rate=0.194175
## Adapt sbrt[18]:pert_rate=0.944954 pertalpha=2 chgv_rate=0.756098   m_rate=0.223301
## Adapt sbrt[19]:pert_rate=0.949495 pertalpha=2 chgv_rate=0.855422   m_rate=0.174757
## Adapt sbrt[20]:pert_rate=0.930233 pertalpha=2 chgv_rate=0.806452   m_rate=0.174757
## Adapt sbrt[21]:pert_rate=0.966292 pertalpha=2 chgv_rate=0.8125   m_rate=0.223301
## Adapt sbrt[22]:pert_rate=0.924731 pertalpha=2 chgv_rate=0.796875   m_rate=0.174757
## Adapt sbrt[23]:pert_rate=0.935484 pertalpha=2 chgv_rate=0.833333   m_rate=0.165049
## Adapt sbrt[24]:pert_rate=0.94186 pertalpha=2 chgv_rate=0.833333   m_rate=0.126214
## Adapt sbrt[25]:pert_rate=0.87234 pertalpha=2 chgv_rate=0.823529   m_rate=0.23301
```

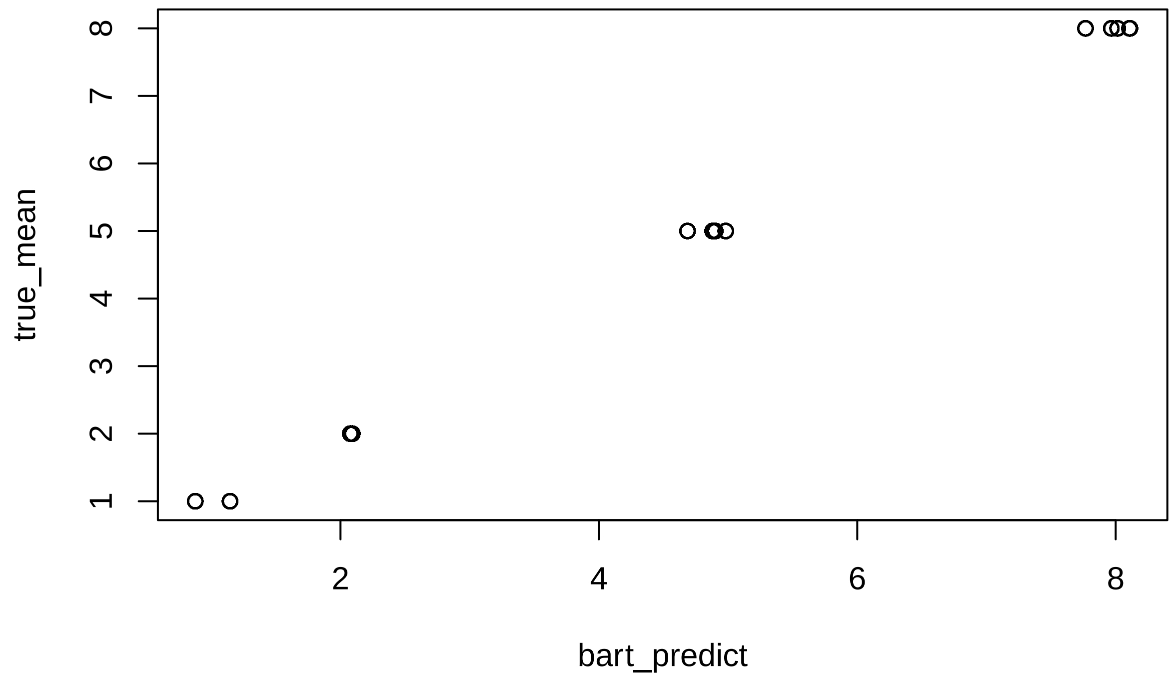
```
## Adapt sbrt[26]:pert_rate=0.989474 pertalpha=2 chgv_rate=0.815385   m_rate=0.145631
## Adapt sbrt[27]:pert_rate=0.987952 pertalpha=2 chgv_rate=0.779221   m_rate=0.106796
## Adapt sbrt[28]:pert_rate=0.866667 pertalpha=2 chgv_rate=0.741935   m_rate=0.349515
## Adapt sbrt[29]:pert_rate=0.988636 pertalpha=2 chgv_rate=0.804878   m_rate=0.281553
## Adapt sbrt[30]:pert_rate=0.941176 pertalpha=2 chgv_rate=0.825397   m_rate=0.262136
## Adapt sbrt[31]:pert_rate=0.980392 pertalpha=2 chgv_rate=0.833333   m_rate=0.242718
## Adapt sbrt[32]:pert_rate=0.910112 pertalpha=2 chgv_rate=0.838235   m_rate=0.184466
## Adapt sbrt[33]:pert_rate=0.980583 pertalpha=2 chgv_rate=0.794118   m_rate=0.23301
## Adapt sbrt[34]:pert_rate=0.969388 pertalpha=2 chgv_rate=0.831169   m_rate=0.271845
## Adapt sbrt[35]:pert_rate=0.959596 pertalpha=2 chgv_rate=0.938462   m_rate=0.165049
## Adapt sbrt[36]:pert_rate=0.917526 pertalpha=2 chgv_rate=0.867647   m_rate=0.174757
## Adapt sbrt[37]:pert_rate=0.989247 pertalpha=2 chgv_rate=0.863636   m_rate=0.15534
## Adapt sbrt[38]:pert_rate=0.988235 pertalpha=2 chgv_rate=0.72   m_rate=0.174757
## Adapt sbrt[39]:pert_rate=0.988372 pertalpha=2 chgv_rate=0.815385   m_rate=0.213592
## Adapt ambrt[0]:pert_rate=0.723383 pertalpha=2 chgv_rate=0.434018   m_rate=0.0291262
## Adapt sbrt[0]:pert_rate=0.989796 pertalpha=2 chgv_rate=0.878788   m_rate=0.174757
## Adapt sbrt[1]:pert_rate=0.988235 pertalpha=2 chgv_rate=0.810811   m_rate=0.0970874
## Adapt sbrt[2]:pert_rate=0.977528 pertalpha=2 chgv_rate=0.828947   m_rate=0.174757
## Adapt sbrt[3]:pert_rate=0.95 pertalpha=2 chgv_rate=0.873418   m_rate=0.281553
## Adapt sbrt[4]:pert_rate=0.929825 pertalpha=2 chgv_rate=0.825581   m_rate=0.378641
## Adapt sbrt[5]:pert_rate=0.979798 pertalpha=2 chgv_rate=0.852273   m_rate=0.174757
## Adapt sbrt[6]:pert_rate=0.9875 pertalpha=2 chgv_rate=0.8   m_rate=0.203883
## Adapt sbrt[7]:pert_rate=0.928571 pertalpha=2 chgv_rate=0.790123   m_rate=0.203883
## Adapt sbrt[8]:pert_rate=0.943182 pertalpha=2 chgv_rate=0.884058   m_rate=0.213592
## Adapt sbrt[9]:pert_rate=0.908046 pertalpha=2 chgv_rate=0.847059   m_rate=0.184466
## Adapt sbrt[10]:pert_rate=0.991304 pertalpha=2 chgv_rate=0.816901   m_rate=0.213592
## Adapt sbrt[11]:pert_rate=0.960396 pertalpha=2 chgv_rate=0.818182   m_rate=0.23301
## Adapt sbrt[12]:pert_rate=0.941176 pertalpha=2 chgv_rate=0.785714   m_rate=0.252427
## Adapt sbrt[13]:pert_rate=0.888889 pertalpha=2 chgv_rate=0.725   m_rate=0.242718
## Adapt sbrt[14]:pert_rate=0.989796 pertalpha=2 chgv_rate=0.744681   m_rate=0.165049
## Adapt sbrt[15]:pert_rate=0.988636 pertalpha=2 chgv_rate=0.828947   m_rate=0.145631
## Adapt sbrt[16]:pert_rate=0.92381 pertalpha=2 chgv_rate=0.830508   m_rate=0.242718
## Adapt sbrt[17]:pert_rate=0.931373 pertalpha=2 chgv_rate=0.782609   m_rate=0.165049
## Adapt sbrt[18]:pert_rate=0.966667 pertalpha=2 chgv_rate=0.747253   m_rate=0.339806
## Adapt sbrt[19]:pert_rate=0.933333 pertalpha=2 chgv_rate=0.848485   m_rate=0.165049
## Adapt sbrt[20]:pert_rate=0.955556 pertalpha=2 chgv_rate=0.828947   m_rate=0.281553
## Adapt sbrt[21]:pert_rate=0.965116 pertalpha=2 chgv_rate=0.85   m_rate=0.116505
## Adapt sbrt[22]:pert_rate=0.943396 pertalpha=2 chgv_rate=0.792208   m_rate=0.203883
## Adapt sbrt[23]:pert_rate=0.989474 pertalpha=2 chgv_rate=0.828571   m_rate=0.194175
## Adapt sbrt[24]:pert_rate=0.927835 pertalpha=2 chgv_rate=0.84375   m_rate=0.23301
## Adapt sbrt[25]:pert_rate=0.930556 pertalpha=2 chgv_rate=0.820513   m_rate=0.242718
## Adapt sbrt[26]:pert_rate=0.937931 pertalpha=2 chgv_rate=0.810127   m_rate=0.242718
## Adapt sbrt[27]:pert_rate=0.943925 pertalpha=2 chgv_rate=0.761364   m_rate=0.213592
## Adapt sbrt[28]:pert_rate=0.979592 pertalpha=2 chgv_rate=0.786667   m_rate=0.174757
## Adapt sbrt[29]:pert_rate=0.963636 pertalpha=2 chgv_rate=0.793103   m_rate=0.252427
## Adapt sbrt[30]:pert_rate=0.952381 pertalpha=2 chgv_rate=0.857143   m_rate=0.135922
## Adapt sbrt[31]:pert_rate=0.967391 pertalpha=2 chgv_rate=0.833333   m_rate=0.271845
## Adapt sbrt[32]:pert_rate=0.967033 pertalpha=2 chgv_rate=0.855263   m_rate=0.213592
## Adapt sbrt[33]:pert_rate=0.847059 pertalpha=2 chgv_rate=0.794872   m_rate=0.242718
## Adapt sbrt[34]:pert_rate=0.949495 pertalpha=2 chgv_rate=0.806818   m_rate=0.252427
## Adapt sbrt[35]:pert_rate=0.945652 pertalpha=2 chgv_rate=0.925926   m_rate=0.291262
## Adapt sbrt[36]:pert_rate=0.989583 pertalpha=2 chgv_rate=0.873418   m_rate=0.145631
## Adapt sbrt[37]:pert_rate=0.99 pertalpha=2 chgv_rate=0.861386   m_rate=0.194175
## Adapt sbrt[38]:pert_rate=0.988889 pertalpha=2 chgv_rate=0.743902   m_rate=0.165049
```

```
## Adapt sbrt[39]:pert_rate=0.988372 pertalpha=2 chgv_rate=0.835616   m_rate=0.223301
## Adapt ambrt[0]:pert_rate=0.725726 pertalpha=2 chgv_rate=0.433673   m_rate=0.038835
## Adapt sbrt[0]:pert_rate=0.983607 pertalpha=2 chgv_rate=0.886076   m_rate=0.213592
## Adapt sbrt[1]:pert_rate=0.95614 pertalpha=2 chgv_rate=0.804878   m_rate=0.23301
## Adapt sbrt[2]:pert_rate=0.982759 pertalpha=2 chgv_rate=0.844444   m_rate=0.242718
## Adapt sbrt[3]:pert_rate=0.865979 pertalpha=2 chgv_rate=0.868132   m_rate=0.281553
## Adapt sbrt[4]:pert_rate=0.971429 pertalpha=2 chgv_rate=0.821782   m_rate=0.213592
## Adapt sbrt[5]:pert_rate=0.938272 pertalpha=2 chgv_rate=0.84466   m_rate=0.252427
## Adapt sbrt[6]:pert_rate=0.990291 pertalpha=2 chgv_rate=0.815789   m_rate=0.223301
## Adapt sbrt[7]:pert_rate=0.966942 pertalpha=2 chgv_rate=0.806122   m_rate=0.203883
## Adapt sbrt[8]:pert_rate=0.921348 pertalpha=2 chgv_rate=0.871795   m_rate=0.223301
## Adapt sbrt[9]:pert_rate=0.977778 pertalpha=2 chgv_rate=0.857143   m_rate=0.271845
## Adapt sbrt[10]:pert_rate=0.951456 pertalpha=2 chgv_rate=0.788235   m_rate=0.271845
## Adapt sbrt[11]:pert_rate=0.854369 pertalpha=2 chgv_rate=0.827957   m_rate=0.359223
## Adapt sbrt[12]:pert_rate=0.968421 pertalpha=2 chgv_rate=0.769231   m_rate=0.23301
## Adapt sbrt[13]:pert_rate=0.945055 pertalpha=2 chgv_rate=0.735632   m_rate=0.135922
## Adapt sbrt[14]:pert_rate=0.928571 pertalpha=2 chgv_rate=0.768519   m_rate=0.15534
## Adapt sbrt[15]:pert_rate=0.979798 pertalpha=2 chgv_rate=0.829268   m_rate=0.203883
## Adapt sbrt[16]:pert_rate=0.961165 pertalpha=2 chgv_rate=0.842857   m_rate=0.252427
## Adapt sbrt[17]:pert_rate=0.95098 pertalpha=2 chgv_rate=0.766234   m_rate=0.174757
## Adapt sbrt[18]:pert_rate=0.970297 pertalpha=2 chgv_rate=0.755102   m_rate=0.184466
## Adapt sbrt[19]:pert_rate=0.972727 pertalpha=2 chgv_rate=0.841121   m_rate=0.145631
## Adapt sbrt[20]:pert_rate=0.928571 pertalpha=2 chgv_rate=0.8375   m_rate=0.242718
## Adapt sbrt[21]:pert_rate=0.946237 pertalpha=2 chgv_rate=0.850575   m_rate=0.0970874
## Adapt sbrt[22]:pert_rate=0.93617 pertalpha=2 chgv_rate=0.804598   m_rate=0.194175
## Adapt sbrt[23]:pert_rate=0.961165 pertalpha=2 chgv_rate=0.822785   m_rate=0.223301
## Adapt sbrt[24]:pert_rate=0.926829 pertalpha=2 chgv_rate=0.844156   m_rate=0.281553
## Adapt sbrt[25]:pert_rate=0.977528 pertalpha=2 chgv_rate=0.825581   m_rate=0.213592
## Adapt sbrt[26]:pert_rate=0.964286 pertalpha=2 chgv_rate=0.813187   m_rate=0.281553
## Adapt sbrt[27]:pert_rate=0.921569 pertalpha=2 chgv_rate=0.77   m_rate=0.291262
## Adapt sbrt[28]:pert_rate=0.988506 pertalpha=2 chgv_rate=0.797619   m_rate=0.135922
## Adapt sbrt[29]:pert_rate=0.971429 pertalpha=2 chgv_rate=0.804124   m_rate=0.271845
## Adapt sbrt[30]:pert_rate=0.975 pertalpha=2 chgv_rate=0.858824   m_rate=0.291262
## Adapt sbrt[31]:pert_rate=0.980952 pertalpha=2 chgv_rate=0.817073   m_rate=0.359223
## Adapt sbrt[32]:pert_rate=0.902913 pertalpha=2 chgv_rate=0.851852   m_rate=0.23301
## Adapt sbrt[33]:pert_rate=0.988889 pertalpha=2 chgv_rate=0.790698   m_rate=0.184466
## Adapt sbrt[34]:pert_rate=0.976744 pertalpha=2 chgv_rate=0.824742   m_rate=0.135922
## Adapt sbrt[35]:pert_rate=0.989362 pertalpha=2 chgv_rate=0.930233   m_rate=0.116505
## Adapt sbrt[36]:pert_rate=0.893805 pertalpha=2 chgv_rate=0.862069   m_rate=0.23301
## Adapt sbrt[37]:pert_rate=0.936842 pertalpha=2 chgv_rate=0.843478   m_rate=0.262136
## Adapt sbrt[38]:pert_rate=0.987654 pertalpha=2 chgv_rate=0.758242   m_rate=0.242718
## Adapt sbrt[39]:pert_rate=0.950495 pertalpha=2 chgv_rate=0.817073   m_rate=0.252427
## Adapt ambrt[0]:pert_rate=0.713433 pertalpha=2 chgv_rate=0.426136   m_rate=0.038835
## Adapt sbrt[0]:pert_rate=0.956989 pertalpha=2 chgv_rate=0.896552   m_rate=0.126214
## Adapt sbrt[1]:pert_rate=0.983051 pertalpha=2 chgv_rate=0.785714   m_rate=0.116505
## Adapt sbrt[2]:pert_rate=0.935185 pertalpha=2 chgv_rate=0.833333   m_rate=0.281553
## Adapt sbrt[3]:pert_rate=0.979592 pertalpha=2 chgv_rate=0.862745   m_rate=0.291262
## Adapt sbrt[4]:pert_rate=0.977273 pertalpha=2 chgv_rate=0.830357   m_rate=0.174757
## Adapt sbrt[5]:pert_rate=0.979592 pertalpha=2 chgv_rate=0.826087   m_rate=0.184466
## Adapt sbrt[6]:pert_rate=0.965517 pertalpha=2 chgv_rate=0.802326   m_rate=0.203883
## Adapt sbrt[7]:pert_rate=0.989011 pertalpha=2 chgv_rate=0.816514   m_rate=0.135922
## Adapt sbrt[8]:pert_rate=0.975904 pertalpha=2 chgv_rate=0.866667   m_rate=0.203883
## Adapt sbrt[9]:pert_rate=0.989247 pertalpha=2 chgv_rate=0.866667   m_rate=0.135922
## Adapt sbrt[10]:pert_rate=0.990741 pertalpha=2 chgv_rate=0.787234   m_rate=0.271845
```

```
## Adapt sbrt[11]:pert_rate=0.981481 pertalpha=2 chgv_rate=0.833333   m_rate=0.184466
## Adapt sbrt[12]:pert_rate=0.987952 pertalpha=2 chgv_rate=0.791209   m_rate=0.184466
## Adapt sbrt[13]:pert_rate=0.957447 pertalpha=2 chgv_rate=0.737864   m_rate=0.194175
## Adapt sbrt[14]:pert_rate=0.913462 pertalpha=2 chgv_rate=0.762712   m_rate=0.252427
## Adapt sbrt[15]:pert_rate=0.989691 pertalpha=2 chgv_rate=0.819048   m_rate=0.203883
## Adapt sbrt[16]:pert_rate=0.980769 pertalpha=2 chgv_rate=0.858824   m_rate=0.23301
## Adapt sbrt[17]:pert_rate=0.980198 pertalpha=2 chgv_rate=0.772727   m_rate=0.165049
## Adapt sbrt[18]:pert_rate=0.947917 pertalpha=2 chgv_rate=0.773913   m_rate=0.291262
## Adapt sbrt[19]:pert_rate=0.989691 pertalpha=2 chgv_rate=0.845455   m_rate=0.174757
## Adapt sbrt[20]:pert_rate=0.989247 pertalpha=2 chgv_rate=0.829268   m_rate=0.15534
## Adapt sbrt[21]:pert_rate=0.988372 pertalpha=2 chgv_rate=0.865979   m_rate=0.194175
## Adapt sbrt[22]:pert_rate=0.987805 pertalpha=2 chgv_rate=0.8125   m_rate=0.116505
## Adapt sbrt[23]:pert_rate=0.990476 pertalpha=2 chgv_rate=0.797753   m_rate=0.194175
## Adapt sbrt[24]:pert_rate=0.957895 pertalpha=2 chgv_rate=0.808989   m_rate=0.174757
## Adapt sbrt[25]:pert_rate=0.87619 pertalpha=2 chgv_rate=0.848485   m_rate=0.252427
## Adapt sbrt[26]:pert_rate=0.961165 pertalpha=2 chgv_rate=0.82   m_rate=0.203883
## Adapt sbrt[27]:pert_rate=0.987654 pertalpha=2 chgv_rate=0.776786   m_rate=0.165049
## Adapt sbrt[28]:pert_rate=0.956835 pertalpha=2 chgv_rate=0.802083   m_rate=0.300971
## Adapt sbrt[29]:pert_rate=0.94382 pertalpha=2 chgv_rate=0.807339   m_rate=0.174757
## Adapt sbrt[30]:pert_rate=0.979592 pertalpha=2 chgv_rate=0.861702   m_rate=0.252427
## Adapt sbrt[31]:pert_rate=0.886792 pertalpha=2 chgv_rate=0.829787   m_rate=0.291262
## Adapt sbrt[32]:pert_rate=0.989796 pertalpha=2 chgv_rate=0.854167   m_rate=0.145631
## Adapt sbrt[33]:pert_rate=0.87 pertalpha=2 chgv_rate=0.785714   m_rate=0.252427
## Adapt sbrt[34]:pert_rate=0.978947 pertalpha=2 chgv_rate=0.831683   m_rate=0.15534
## Adapt sbrt[35]:pert_rate=0.966292 pertalpha=2 chgv_rate=0.935484   m_rate=0.15534
## Adapt sbrt[36]:pert_rate=0.911111 pertalpha=2 chgv_rate=0.836735   m_rate=0.223301
## Adapt sbrt[37]:pert_rate=0.979167 pertalpha=2 chgv_rate=0.852459   m_rate=0.23301
## Adapt sbrt[38]:pert_rate=0.933333 pertalpha=2 chgv_rate=0.764706   m_rate=0.252427
## Adapt sbrt[39]:pert_rate=0.926316 pertalpha=2 chgv_rate=0.802198   m_rate=0.252427
## Adapt ambrt[0]:pert_rate=0.726083 pertalpha=2 chgv_rate=0.415992   m_rate=0.0194175
## Adapt sbrt[0]:pert_rate=0.955357 pertalpha=2 chgv_rate=0.9   m_rate=0.23301
## Adapt sbrt[1]:pert_rate=0.95098 pertalpha=2 chgv_rate=0.798165   m_rate=0.242718
## Adapt sbrt[2]:pert_rate=0.988506 pertalpha=2 chgv_rate=0.852174   m_rate=0.135922
## Adapt sbrt[3]:pert_rate=0.989691 pertalpha=2 chgv_rate=0.866071   m_rate=0.194175
## Adapt sbrt[4]:pert_rate=0.883117 pertalpha=2 chgv_rate=0.829457   m_rate=0.174757
## Adapt sbrt[5]:pert_rate=0.99 pertalpha=2 chgv_rate=0.833333   m_rate=0.194175
## Adapt sbrt[6]:pert_rate=0.989899 pertalpha=2 chgv_rate=0.797872   m_rate=0.145631
## Adapt sbrt[7]:pert_rate=0.931034 pertalpha=2 chgv_rate=0.822034   m_rate=0.194175
## Adapt sbrt[8]:pert_rate=0.958763 pertalpha=2 chgv_rate=0.861702   m_rate=0.194175
## Adapt sbrt[9]:pert_rate=0.967033 pertalpha=2 chgv_rate=0.875   m_rate=0.135922
## Adapt sbrt[10]:pert_rate=0.980392 pertalpha=2 chgv_rate=0.775701   m_rate=0.242718
## Adapt sbrt[11]:pert_rate=0.945652 pertalpha=2 chgv_rate=0.842975   m_rate=0.252427
## Adapt sbrt[12]:pert_rate=0.960396 pertalpha=2 chgv_rate=0.817308   m_rate=0.213592
## Adapt sbrt[13]:pert_rate=0.987952 pertalpha=2 chgv_rate=0.743363   m_rate=0.174757
## Adapt sbrt[14]:pert_rate=0.964286 pertalpha=2 chgv_rate=0.769231   m_rate=0.184466
## Adapt sbrt[15]:pert_rate=0.991667 pertalpha=2 chgv_rate=0.820513   m_rate=0.262136
## Adapt sbrt[16]:pert_rate=0.987952 pertalpha=2 chgv_rate=0.864583   m_rate=0.106796
## Adapt sbrt[17]:pert_rate=0.989011 pertalpha=2 chgv_rate=0.772277   m_rate=0.174757
## Adapt sbrt[18]:pert_rate=0.908163 pertalpha=2 chgv_rate=0.779528   m_rate=0.252427
## Adapt sbrt[19]:pert_rate=0.979381 pertalpha=2 chgv_rate=0.847458   m_rate=0.116505
## Adapt sbrt[20]:pert_rate=0.989583 pertalpha=2 chgv_rate=0.827957   m_rate=0.174757
## Adapt sbrt[21]:pert_rate=0.963855 pertalpha=2 chgv_rate=0.87037   m_rate=0.174757
## Adapt sbrt[22]:pert_rate=0.990741 pertalpha=2 chgv_rate=0.817308   m_rate=0.213592
## Adapt sbrt[23]:pert_rate=0.95 pertalpha=2 chgv_rate=0.795918   m_rate=0.271845
```

```
## Adapt sbrt[24]:pert_rate=0.981132 pertalpha=2 chgv_rate=0.828283   m_rate=0.223301
## Adapt sbrt[25]:pert_rate=0.974359 pertalpha=2 chgv_rate=0.846154   m_rate=0.203883
## Adapt sbrt[26]:pert_rate=0.946809 pertalpha=2 chgv_rate=0.828829   m_rate=0.252427
## Adapt sbrt[27]:pert_rate=0.989474 pertalpha=2 chgv_rate=0.776   m_rate=0.145631
## Adapt sbrt[28]:pert_rate=0.952381 pertalpha=2 chgv_rate=0.789474   m_rate=0.223301
## Adapt sbrt[29]:pert_rate=0.945652 pertalpha=2 chgv_rate=0.822581   m_rate=0.31068
## Adapt sbrt[30]:pert_rate=0.965217 pertalpha=2 chgv_rate=0.857143   m_rate=0.31068
## Adapt sbrt[31]:pert_rate=0.978261 pertalpha=2 chgv_rate=0.83   m_rate=0.174757
## Adapt sbrt[32]:pert_rate=0.978022 pertalpha=2 chgv_rate=0.849057   m_rate=0.165049
## Adapt sbrt[33]:pert_rate=0.979381 pertalpha=2 chgv_rate=0.792453   m_rate=0.174757
## Adapt sbrt[34]:pert_rate=0.956522 pertalpha=2 chgv_rate=0.833333   m_rate=0.145631
## Adapt sbrt[35]:pert_rate=0.974359 pertalpha=2 chgv_rate=0.91   m_rate=0.15534
## Adapt sbrt[36]:pert_rate=0.98913 pertalpha=2 chgv_rate=0.828829   m_rate=0.174757
## Adapt sbrt[37]:pert_rate=0.901099 pertalpha=2 chgv_rate=0.852941   m_rate=0.223301
## Adapt sbrt[38]:pert_rate=0.945055 pertalpha=2 chgv_rate=0.763636   m_rate=0.0776699
## Adapt sbrt[39]:pert_rate=0.979798 pertalpha=2 chgv_rate=0.818182   m_rate=0.174757
## draw  burn 0
## draw  burn 100
## draw  burn 200
## draw  burn 300
## draw  burn 400
## draw  burn 500
## draw  burn 600
## draw  burn 700
## draw  burn 800
## draw  burn 900
## draw  burn 1000
## draw  burn 1100
## draw  burn 1200
## draw  burn 1300
## draw  burn 1400
## draw  burn 1500
## draw  burn 1600
## draw  burn 1700
## draw  burn 1800
## draw  burn 1900
## draw  keep 0
## draw  keep 100
## draw  keep 200
## draw  keep 300
## draw  keep 400
## draw  keep 500
## draw  keep 600
## draw  keep 700
## draw  keep 800
## draw  keep 900
```

```
bart_predict <- predict(bart_res, sim_x)$mmean
plot(x = bart_predict, y = true_mean)
```




```
res_bart <- (bart_predict-true_mean)^2
matrix(c(sum(res_bart), sum(res_cart), sum(res_rpart)), ncol = 3)
```

```
##          [,1]     [,2]     [,3]
## [1,] 19.39387 3035.409 7.659329
```

16. Out of Sample Prediction of CART

Here, we are going to use the result of the bart function to predict the out of sample observations. 800 of the data was randomly selected as train set and the rest as validation set.

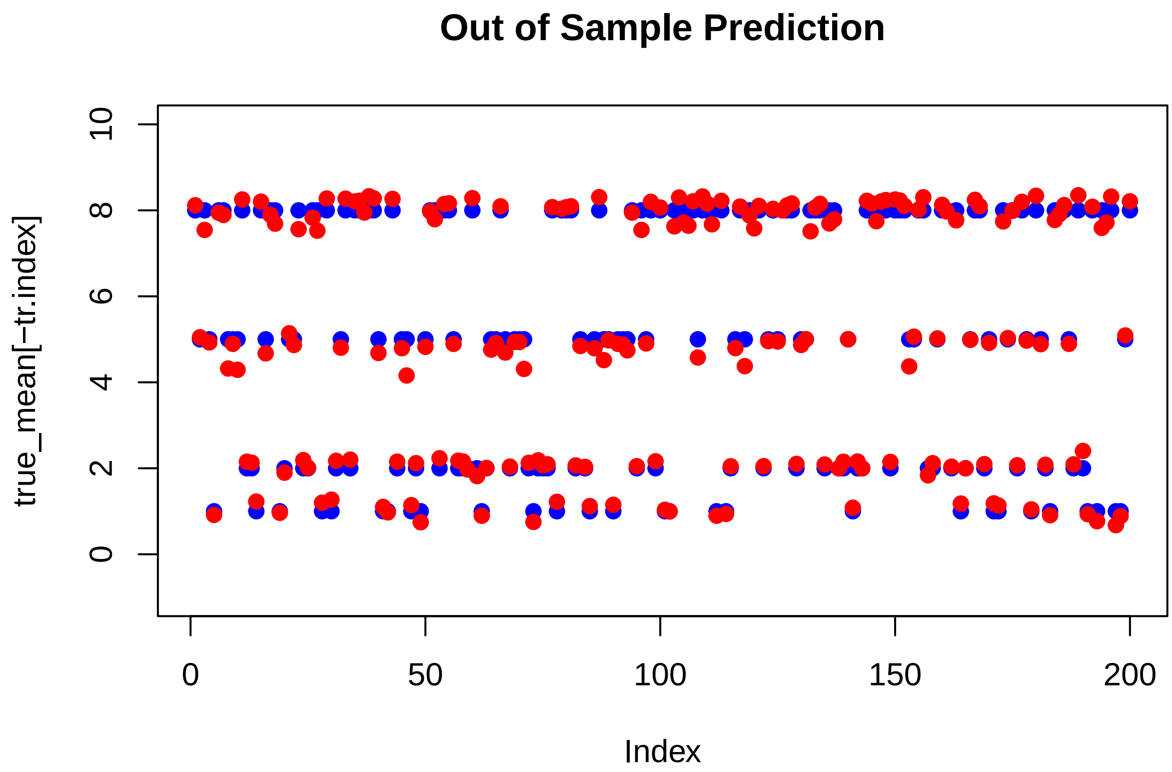
```
tr.index <- sample(size = 800, 1:1000, replace = FALSE)
x.tr <- sim_x[tr.index,]
x.va <- sim_x[-tr.index,]
y.tr <- sim_y[tr.index]
y.va <- sim_y[-tr.index]
cv.bart <- train_bart(x.tr, y.tr, mcmc_par = list(iter = 1000, burn = 2000, thin = 2),hyper = list(alpha
```

```
##   |                                                                      |
```

```
bart_oo_pred<- function(bart_result, x.va){

  store_size <- length(bart_result$tree)
```

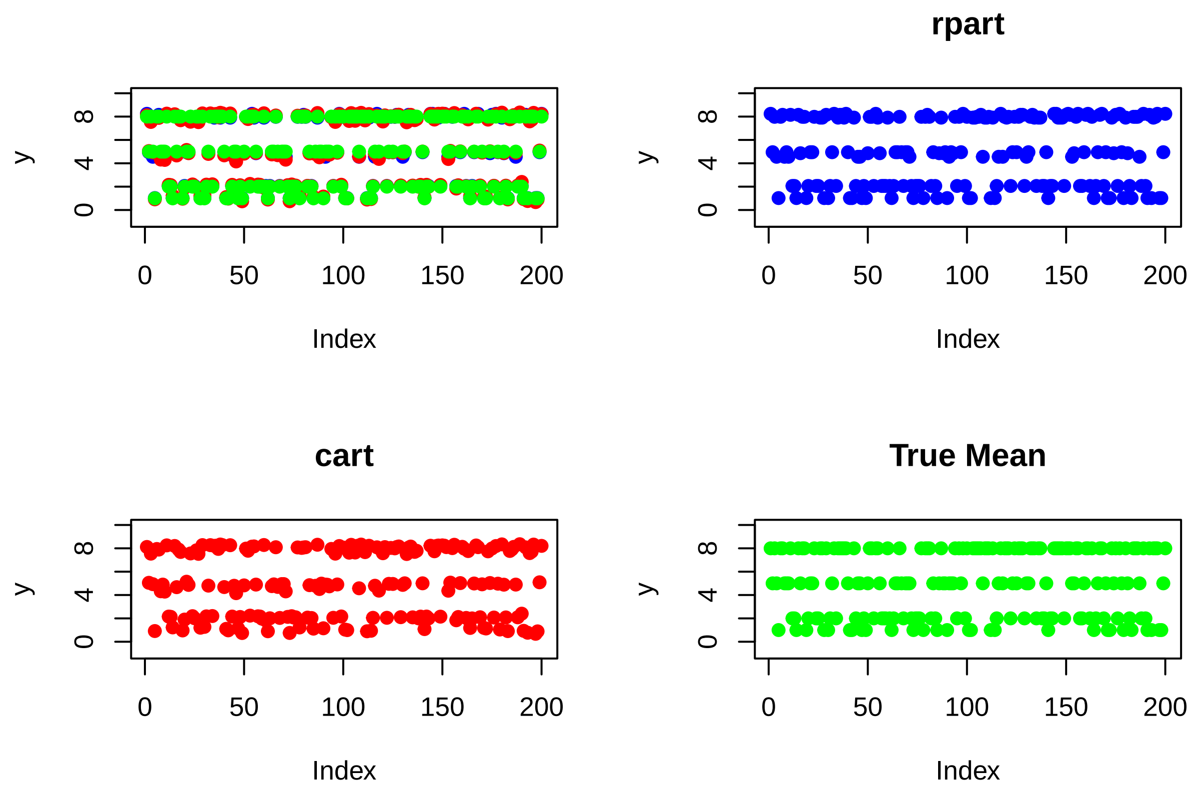
```
  xva.scaled <- x.va
  center_y <- bart_result$center_y
  scale_y <- bart_result$scale_y
  for (j in 1:ncol(x.va)) {
     xva.scaled[,j] <- (x.va[,j]-bart_result$center_x[j])/bart_result$scale_x[j]
  }

  pred <- matrix(NA, nrow = store_size, ncol = nrow(x.va))
  for (i in 1:store_size) {
    curr_tree <- bart_result$tree[[i]]
    curr_sig <- bart_result$sigma2_scaled[i]
    all_mu <- curr_tree$info[,"mu"]
    pred_node <- fill_tree_details(curr_tree, xva.scaled)$node
    pred_mean <- all_mu[pred_node]
    pred_scaled <- rnorm(n = nrow(x.va), mean = pred_mean, sd = sqrt(curr_sig))
    pred[i,] <- pred_scaled * scale_y + center_y


  }

  return(pred)


}
```

```
oo_pred <- bart_oo_pred(cv.bart, x.va)
pos_oo_mean <- apply(oo_pred, 2, mean)

plot(true_mean[-tr.index] , col = "blue", pch = 19, main = "Out of Sample Prediction", ylim = c(-1,10))
lines(pos_oo_mean, col = "red", pch = 19, type = "p")
```



17. Out of Sample Prediction of Rpart

```
tr_dat <- as.data.frame(cbind(y.tr, x.tr))
colnames(tr_dat) <- c("y","x1","A","B","C","D")
colnames(x.va) <- c("x1","A","B","C","D")
cv_mod <- rpart(y ~., data = tr_dat)
rpart_pred <- predict(cv_mod, newdata = as.data.frame(x.va))


par(mfrow = c(2,2))


plot(rpart_pred, type='p', col = "blue", pch = 19, ylim = c(-1,10), ylab = "y")
lines(pos_oo_mean, col = "red", pch = 19, type = "p")
lines(true_mean[-tr.index], col = "green", pch = 19, type = "p")


plot(rpart_pred, type='p', col = "blue", pch = 19, ylim = c(-1,10), ylab = "y", main = "rpart")
plot(pos_oo_mean, col = "red", pch = 19, type = "p", ylab = "y", main = "cart", ylim = c(-1,10))
plot(true_mean[-tr.index], col = "green", pch = 19, type = "p", ylab = "y" ,main = "True Mean", ylim =
```




```
pred_err_rpart <- sum((rpart_pred - true_mean[-tr.index])^2)
pred_err_cart <- sum((pos_oo_mean - true_mean[-tr.index])^2)
paste("pred_err_cart = ",pred_err_cart)
```

```
## [1] "pred_err_cart =  10.4444872366368"
```

```
paste("pred_err_rpart = ",pred_err_rpart)
```

```
## [1] "pred_err_rpart =  4.39562737987731"
```