COMP/INDR 421/521 HW05: Decision Tree Regression

Asma Hakouz 0063315

November 17, 2017

The purpose of this assignment was to implement a decision tree regression algorithm in R.

The problem consisted of the implementation and evaluation of a univariate regression tree, which was done in these stages:

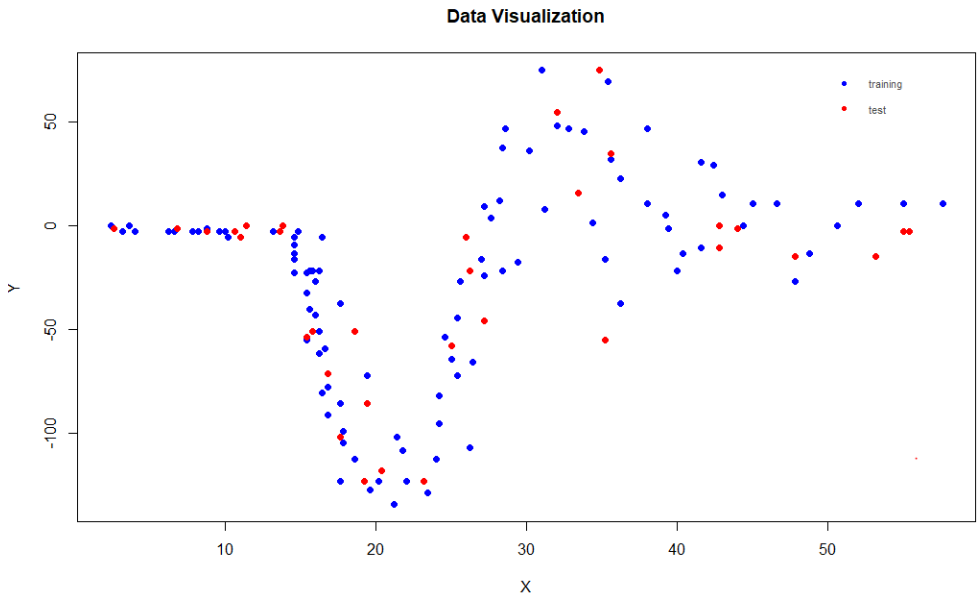
1- **Data pre-processing and visualization**
2- **Decision Tree Training**
3- **Tree Evaluation**

Following, more details will be discussed about each part accompanied by snippets from my source code.

- **Data Pre-processing and visualization**

```
# read data into memory
data_set <- read.csv("hw05_data_set.csv")

# get x and y values
x <- data_set$x
y <- data_set$y

# get number of samples
N <- length(y)

train_sample_count = 100

# randomly dividing the data set to training and test samples
train_indices <-  c(sample(1:N, floor(train_sample_count)))
X_train <- x[train_indices]
y_train <- y[train_indices]
X_test <- x[-train_indices]
y_test <- y[-train_indices]

plot(X_train, y_train, type = "p", pch = 19, col = "blue",
     main = "Data Visualization", xlab = "X", ylab = "Y")
points(X_test, y_test, type = "p", pch = 19, col = "red")

# get numbers of train and test samples
N_train <- length(y_train)
N_test <- length(y_test)
```

This step includes reading data provided in the csv file, then randomly picking 100 out of 133 samples as the training set while the remaining 33 samples are considered as the test set.



- **Decision Tree Training**

1- Initialization

```
# create necessary data structures
node_indices <- list()
is_terminal <- c()
need_split <- c()
node_splits <- c()      # the threshold    (Xj > Wmo)
node_frequencies <- list()

# put all training instances into the root node
node_indices <- list(1:N_train)
is_terminal <- c(FALSE)
need_split <- c(TRUE)

p <- as.integer(readline(prompt="Enter the pre-pruning parameter: "))
```

Where p is a user-defined input.

2- Learning

```
# learning algorithm
while (1) {
  # find nodes that need splitting
  split_nodes <- which(need_split)
  # check whether we reach all terminal nodes
  if (length(split_nodes) == 0) {
    break
  }
  # find best split positions for all nodes
  for (split_node in split_nodes) {
    data_indices <- node_indices[[split_node]]
    need_split[split_node] <- FALSE
    # check whether node is pure
    if (length(data_indices) <= p) {  # if all data point are all from 1 class
      is_terminal[split_node] <- TRUE  # if all point are from the same class, then no need to split
      print(length(data_indices))
      node_frequencies[[split_node]] <- mean(y_train[data_indices])
    } else {
      is_terminal[split_node] <- FALSE #

      unique_values <- sort(unique(X_train[data_indices]))  # to decide what are the possible splits
      split_positions <- (unique_values[-1] + unique_values[-length(unique_values)]) / 2
      split_scores <- rep(0, length(split_positions))
      for (s in 1:length(split_positions)) {
        left_indices <- data_indices[which(X_train[data_indices] < split_positions[s])]
        right_indices <- data_indices[which(X_train[data_indices] >= split_positions[s])]
        left_avg <- mean(y_train[left_indices])
        right_avg <- mean(y_train[right_indices])

        split_scores[s] <- 1 / length(data_indices) * (sum((y_train[left_indices] - left_avg)**2, na.rm = TRUE)
                                                       + sum((y_train[right_indices] - right_avg)**2, na.rm = TRUE))
      }
      best_scores <- min(split_scores)
      best_splits <- split_positions[which.min(split_scores)]

      # decide where to split on which feature
      node_splits[split_node] <- best_splits

      # create left node using the selected split
      left_indices <- data_indices[which(X_train[data_indices] < best_splits)]
      node_indices[[2 * split_node]] <- left_indices
      is_terminal[2 * split_node] <- FALSE
      need_split[2 * split_node] <- TRUE

      # create left node using the selected split
      right_indices <- data_indices[which(X_train[data_indices] >= best_splits)]
      node_indices[[2 * split_node + 1]] <- right_indices
      is_terminal[2 * split_node + 1] <- FALSE
      need_split[2 * split_node + 1] <- TRUE
    }
  }
}
```

3- Rules extraction

```
# extract rules
terminal_nodes <- which(is_terminal)
for (terminal_node in terminal_nodes) {
  index <- terminal_node
  rules <- c()
  while (index > 1) {
    parent <- floor(index / 2)
    if (index %% 2 == 0) {
      # if node is left child of its parent
      rules <- c(sprintf("x < %g", node_splits[parent]), rules)
    } else {
      # if node is right child of its parent
      rules <- c(sprintf("x >= %g", node_splits[parent]), rules)
    }
    index <- parent
  }
  print(sprintf("{%s} => [%s]", paste0(rules, collapse = " AND "), paste0(node_frequencies[[terminal_node]], collapse = "-")))
}
```

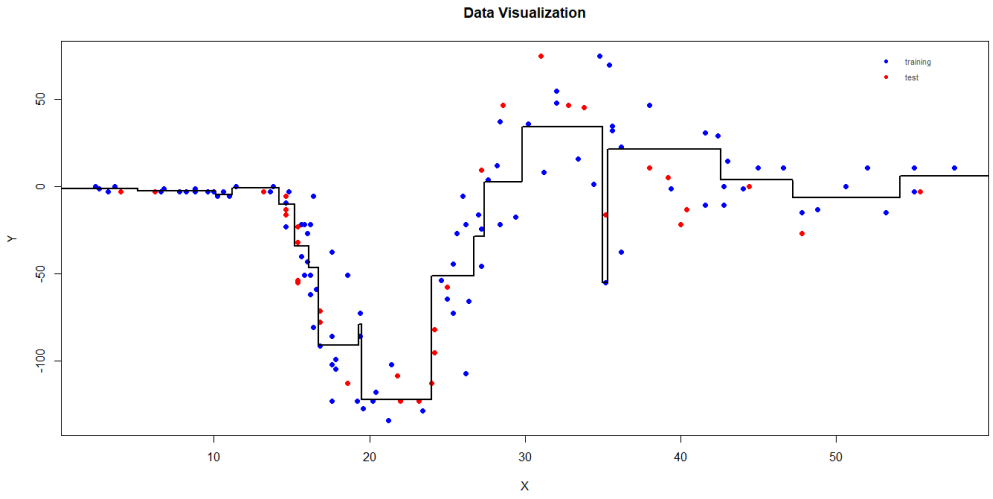
Extracted rules for one of the trials were as follows:

```
[1] "{x < 26.7 AND x >= 16.3 AND x >= 24.6} => [-51.46]"
[1] "{x >= 26.7 AND x < 35 AND x < 27.9} => [-14.075]"
[1] "{x >= 26.7 AND x < 35 AND x >= 27.9} => [33.48]"
[1] "{x >= 26.7 AND x >= 35 AND x < 35.4} => [-35.45]"
[1] "{x < 26.7 AND x < 16.3 AND x < 15.1 AND x >= 14.2} => [-11.24]"
[1] "{x < 26.7 AND x < 16.3 AND x >= 15.1 AND x < 16.1} => [-36.7]"
[1] "{x < 26.7 AND x < 16.3 AND x >= 15.1 AND x >= 16.1} => [-44.6666666666667]"
[1] "{x < 26.7 AND x >= 16.3 AND x < 24.6 AND x >= 19.5} => [-117.96]"
[1] "{x >= 26.7 AND x >= 35 AND x >= 35.4 AND x < 38.6} => [16.075]"
[1] "{x < 26.7 AND x < 16.3 AND x < 15.1 AND x < 14.2 AND x < 3.8} => [-1]"
[1] "{x < 26.7 AND x >= 16.3 AND x < 24.6 AND x < 19.5 AND x < 17.7} => [-71.7571428571429]"
[1] "{x < 26.7 AND x >= 16.3 AND x < 24.6 AND x < 19.5 AND x >= 17.7} => [-87.45]"
[1] "{x >= 26.7 AND x >= 35 AND x >= 35.4 AND x >= 38.6 AND x >= 50.4} => [4]"
[1] "{x < 26.7 AND x < 16.3 AND x < 15.1 AND x < 14.2 AND x >= 3.8 AND x >= 13.7} => [0]"
[1] "{x >= 26.7 AND x >= 35 AND x >= 35.4 AND x >= 38.6 AND x < 50.4 AND x >= 47.2} => [-18.2666666666667]"
[1] "{x < 26.7 AND x < 16.3 AND x < 15.1 AND x < 14.2 AND x >= 3.8 AND x < 13.7 AND x < 10.8} => [-2.38888888888889]"
[1] "{x < 26.7 AND x < 16.3 AND x < 15.1 AND x < 14.2 AND x >= 3.8 AND x < 13.7 AND x >= 10.8} => [-4.05]"
[1] "{x >= 26.7 AND x >= 35 AND x >= 35.4 AND x >= 38.6 AND x < 50.4 AND x < 47.2 AND x < 42.9} => [-6.46666666666667]"
[1] "{x >= 26.7 AND x >= 35 AND x >= 35.4 AND x >= 38.6 AND x < 50.4 AND x < 47.2 AND x >= 42.9} => [6.96]"
```

4- Visualization (for p = 10)

```r
x_interval <- seq(from = 0, to = 60, by = 0.01)
y_predicted <- rep(0, length(x_interval))
for (i in 1:length(x_interval)) {
  index <- 1
  while (1) {
    if (is_terminal[index] == TRUE) {
      y_predicted[i] <- node_frequencies[[index]]
      break
    } else {
      if (x_interval[i] < node_splits[index]) {
        index <- index * 2
      } else {
        index <- index * 2 + 1
      }
    }
  }
}
lines(x_interval, y_predicted, lwd = 2, col = "black")
```

5- Evaluation (RMSE Calculation)



```
# traverse tree for test data points
y_predicted <- rep(0, N_test)
for (i in 1:N_test) {
  index <- 1
  while (1) {
    if (is_terminal[index] == TRUE) {
      y_predicted[i] <- node_frequencies[[index]]
      break
    } else {
      if (X_test[i] < node_splits[index]) {
        index <- index * 2
      } else {
        index <- index * 2 + 1
      }
    }
  }
}

rmse = sqrt(mean((y_test - y_predicted)**2))
print(rmse)
```

```
> sprintf("RMSE is %f when P is %d", rmse, p)
[1] "RMSE is 24.458878 when P is 10"
```

6- Investigating P variation:

After repeating the algorithm with a wide range of P values, the following graph shows the values of error produced, we can note that the optimal value was around p = 11.