# R Notebook

Octavio del Ser

10/11/2020

# Question 1 Source code

## R1

$R_1(j,s) = X|X_j < s$
R2 can be calculated by doing !R1.

```
R1 <- function(X, s) {
  return(X < s)
}
```

## RSS

Returns the RSS:
$\sum_{i:x_i \in R_1(j,s)} (y_i - \hat{y}R_1)^2 + \sum_{i:x_i \in R_2(j,s)} (y_i - \hat{y}R_2)^2$

```
rss <- function(s, Xj, Y) {
  r1 <- R1(Xj, s)
  yi_r1 <- Y[which(r1 == TRUE)]
  yhat_R1 <- mean(yi_r1)
  yi_r2 <- Y[which(r1 == FALSE)]
  yhat_R2 <- mean(yi_r2)
  return(sum((yi_r1 - yhat_R1)^2) + sum((yi_r2 - yhat_R2)^2))
}
```

## Split vector (step size)

This is used to generate each value of s (which splits to consider)
Given a vector e.g. (1,3,10,2,19,0)
An ordered vector is return with the following properties
(min(v) + step size,min(v)+2*step_size,… max(v)-step size)

```
gen_splits_by_step_size <- function(vector, step_size) {
  rg <- range(vector)
  return(seq(rg[1] + step_size, rg[2] - step_size, by = step_size))
}
```

## Split vector

An alternative way to generate s
we can choose how many splits instead of the step size.

```
gen_splits_by_no_splits <- function(vector, no_splits) {
  step_size <- diff(rg) / no_splits
  return(gen_splits_by_step_size(step_size))
}
```

## Min split

Finds the minimum split of a vector using rss <br. and iterating through each feature and each split vector for said feature.
finally choosing s and j by minimizing rss.

```
min_split <- function(X, Y, split_step_sizes) {
  min_rss_vals <- rep(NaN, length(X))
  min_s_vals <- rep(NaN, length(X))
  i <- 1
  for (Xj in X) {
    splits <- gen_splits_by_step_size(Xj, split_step_sizes[i])
    temp_rss <- rep(NaN, length(splits))
    j <- 1
    for (s in splits) {
      temp_rss[j] <- rss(s, Xj, Y)
      j <- j + 1
    }
    min_index <- which.min(temp_rss)
    min_s_vals[i] <- splits[min_index]
    min_rss_vals[i] <- temp_rss[min_index]
    i <- i + 1
  }
  j <- which.min(min_rss_vals)
  s <- min_s_vals[j]
  return(c(j, s))
}
```

## Decision Stump MSE

Finds the mse for a given stump (split)
MSE is defined by $\frac{RSS}{\#X_j}$

```
ds_mse <- function(j, s, X, Y) {
  return(rss(s, X[, j], Y) / length(X[, j]))
}
```

## Declaring necessary Libraries

```
library("MASS")
library("ISLR")
```

```
Warning: package 'ISLR' was built under R version 4.0.3
```

```
# tree (for testing and comparison only)
library("tree")
```

```
Warning: package 'tree' was built under R version 4.0.3
```

```
library("class")
```

# Question 1 - Answers & tests

Answers are not explicitly hardcoded (typed) but rather generated and printed
from the code hence I will not be typing them, but printing them nicely instead.

Using Boston dataset

```
data(Boston)
```

Seed set to birthday MMDD

```
set.seed(1008)
```

X ; Features are rm and lstat from Boston dataset. Y: we are predicting medv from Boston dataset.

```
X <- data.frame(Boston['rm'], Boston['lstat'])
Y <- Boston['medv']
```

For both rm and lstat, we are considering splits at steps of 0.1

```
split_step_sizes <- rep(0.1, length(X))
```

Splitting the dataset into train and test sets
The training set is half of the dataset.

```
train_split_size <- as.integer(length(X[, 1]) / 2)
train_labels <- sample(1:nrow(Y), train_split_size)
Y.train <- Y[train_labels,]
Y.test <- Y[-train_labels,]
X.train <- X[train_labels,]
X.test <- X[-train_labels,]
```

Finding the minimum (j & s) split for the whole of the training dataset.

```
res <- min_split(X.train, Y.train, split_step_sizes)
print(res)
```

```
[1] 2.00 4.63
```

```
print(paste0(colnames(X)[res[1]], ': split at ', res[2]))
```

```
[1] "lstat: split at 4.63"
```

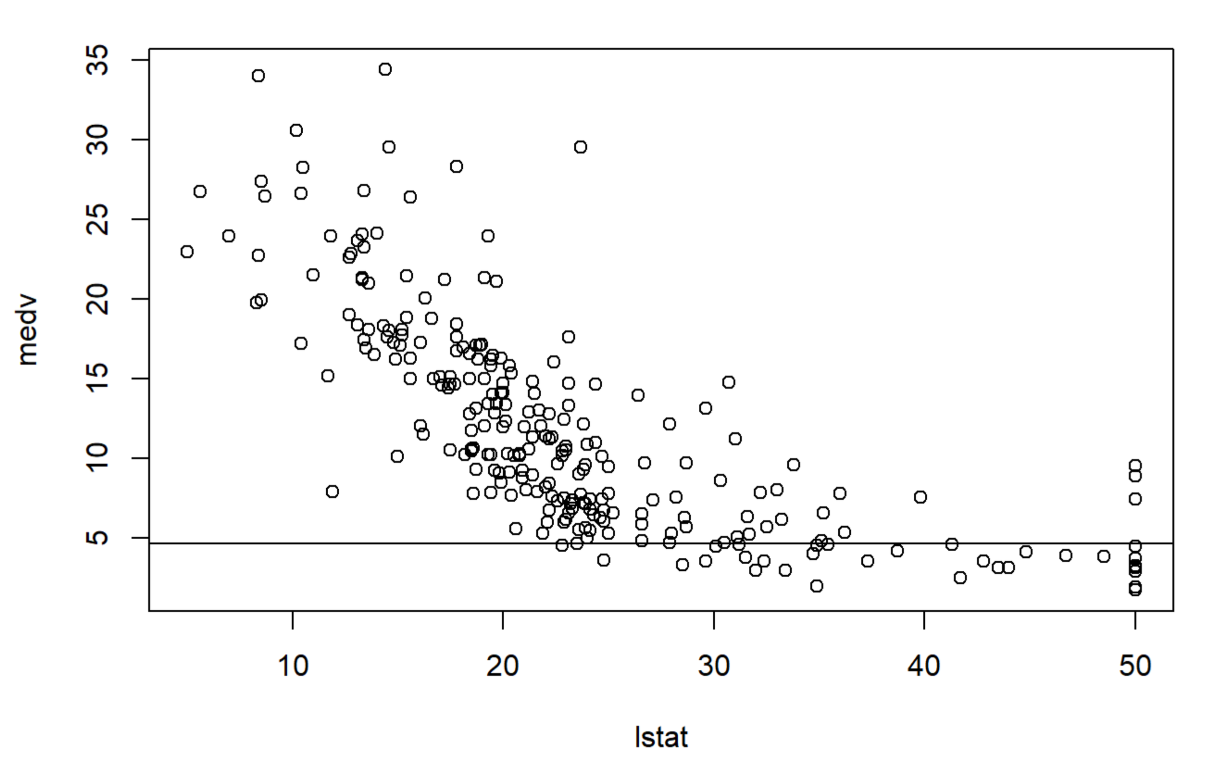
Finding MSE for the whole of the test dataset given s & j from training dataset.

```
ds_mse(res[1], res[2], X.test, Y.test)
```

```
[1] 53.45066
```

A quick plot to visualize the split over the training data

```
plot(Y.train, X.train[, res[1]], ylab = colnames(Y), xlab = colnames(X)[res[1]])
abline(0, 0, res[2])
```



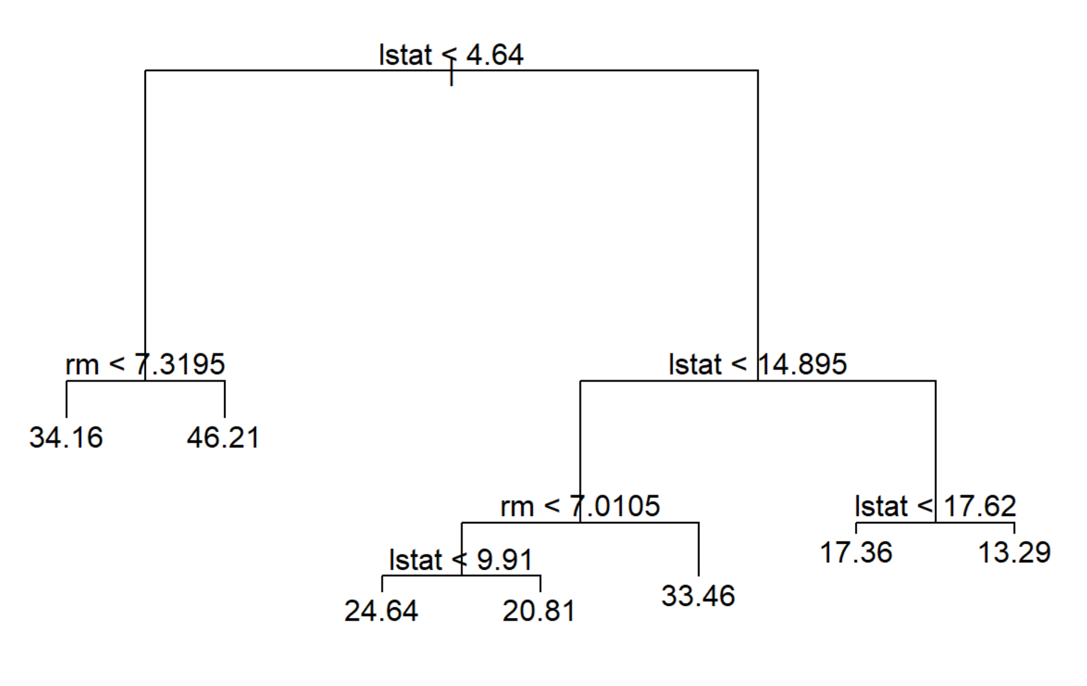

A sanity check against R's implementation (we are only interested in the first split of the tree)

```
tree.carseats <- tree(medv ~ rm + lstat, Boston, subset = train_labels)
summary(tree.carseats)
```

```

Regression tree:
tree(formula = medv ~ rm + lstat, data = Boston, subset = train_labels)
Number of terminal nodes:  7
Residual mean deviance:  20.68 = 5087 / 246
Distribution of residuals:
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-12.7400  -2.4140  -0.3382   0.0000   1.9620  25.3600
```

```
plot(tree.carseats)
text(tree.carseats, pretty = 0)
```




We can see that the first split was at 4.64 which is close the obtained result at lstat: 4.63.
Error can be due to difference in splits considered.

# Question 2,3 Source code

## BDS

BDS algorithm as presented in the assignment Instructions

```
bds <- function(B, Y, X, split_step_sizes, lr) {
  s <- 0
  r <- Y
  for (b in B) {
    f_b <- min_split(X, r, (split_step_sizes))
    s <- s + f_b[2] * lr
    r <- r - lr * f_b[2]
  }
  return(s)
}
```

## Question 2

Only run bds for the stump that had the min oucome int he first result
The output is the test MSE observed for n=0.01 and B=1000

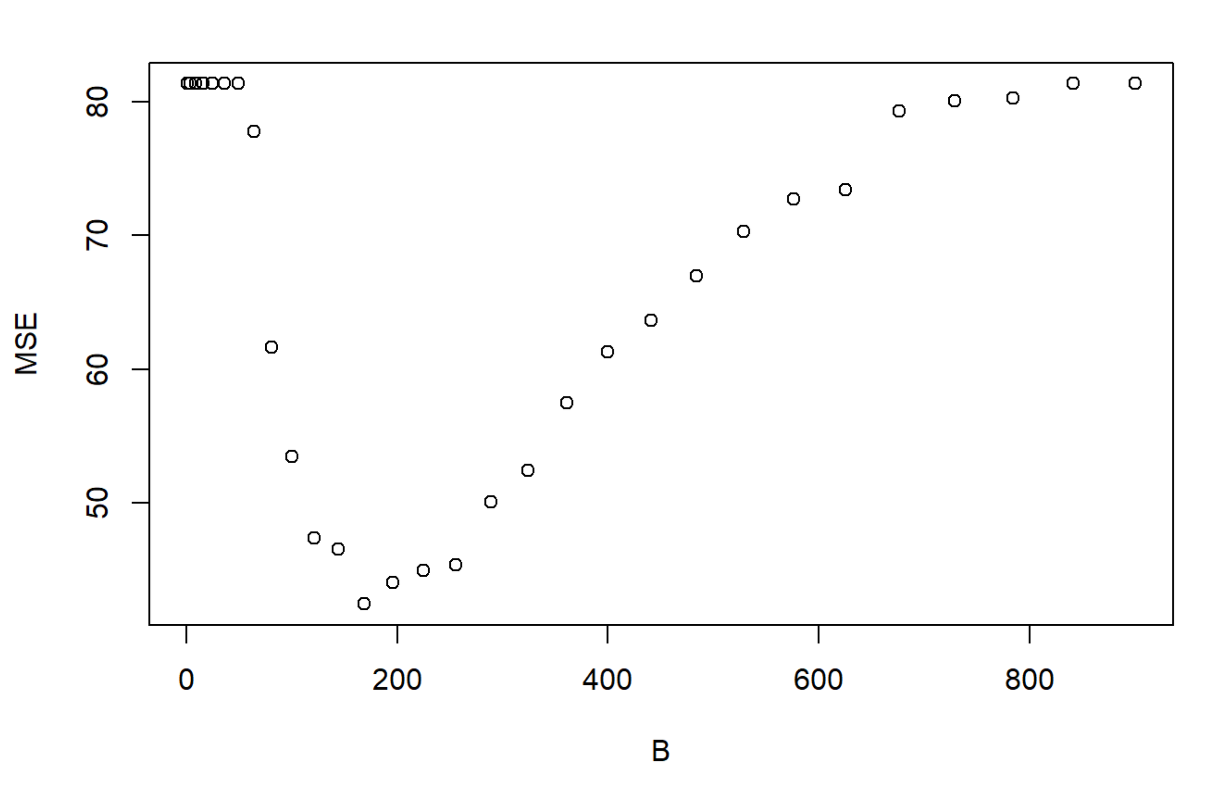
```
ds_mse(res[1], bds(1:1000, Y.train, data.frame(X.train[res[1]]), (0.1), 0.01), X.test, Y.test)
```

```
[1] 81.36959
```

## Question 3

Running bds on 30 values of n 1-> 900 (n^2 series)
Each n represents the number of trees bds was run with.
A graph is then plotted of MSE against n.
learning rate is still 0.01.

```
n <- 30
x <- rep(0, n)
i <- 1
for (b in (1:n)^2) {
  x[i] <- ds_mse(res[1], bds(1:b, Y.train, data.frame(X.train[res[1]]), (0.1), 0.01), X.test, Y.test)
  i <- i + 1
}
plot((1:n)^2, x,ylab = 'MSE',xlab = 'B')
```




We can observe that there is clear over-fitting past the lowest point in the graph
for increasing B number of trees.
There is also clear under-fitting with fewer trees..