BSDA: Assignment 2

November 19, 2023

Time used for reading: 9
Time used for the basic assignment: 14
Time used for extra assignment (VG): 9
Good with lab: There are several good things with the lab The best one is that I now understand the algorithm behind the decision tree, Bagging and the random forest **Things to improve in the lab:** I would not categorize this as an area of development but more as a preference. I would prefer to replace some subtask questions where we computed prediction with the algorithm behind xgboost
Anything that was difficult with the material? Not really. In the beginning, it was difficult to understand the algorithms described in words and make a code out of it but now it feels like one get into it gradually.

Task 1: Decision tree

For this assignment we are going to work with the Hitters data set which we are getting access to by running the following code.

```
library(uuml)
data("Hitters")
```

1.1 Creating a test set

Here we are setting aside the 30 first observations from the data set and that subset corresponds to the test set while the rest of th data set is the train set.

```
Hitters <- Hitters[complete.cases(Hitters), ]
X_test <- Hitters[1:30, c("Years", "Hits")]
y_test <- Hitters[1:30, c("Salary")]
X_train <- Hitters[31:nrow(Hitters),c("Years", "Hits") ]
y_train <- Hitters[31:nrow(Hitters),c("Salary")]
```

1.2 Implementing a function to split observations binary

For this part we will implement an R function that split observations from binary greedily where NA values is being ignored. The function was implemented based on Equation 9.12 − 9.14 in the text book but we added on the leaf size as a function argument. Further, for the function we have three argument. The function takes on a design matrix $\boldsymbol{X}$, one variable y and minimum leaf size ℓ. The function returns a list of indexed set of observations for region $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$, the split value s and the covariate used for the split j. lastly, the Sum of squared SS is also being returned in the function.

```r
tree_split <- function(X = X_train, y = y_train, l = 5) {

  # Here we are checking if the inputs are as they should be
  # which is that X should be a design matrix and the minimum
  # number of leaf should  be a integer as well as the
  # length of the y variable should corresponds to the number
   # of rows in the design matrix X

  checkmate::assert_matrix(X)
  checkmate::assert_numeric(y, len = nrow(X))
  checkmate::assert_int(l)

  # We are creating an matrix object for where we are going
  # to store the sum of squares values for a given row k
  # and given column j. The element in the object will be
  # initially assigned INF and then being updated
  # in the algorithm.

  SS  <- matrix(Inf, nrow = nrow(X), ncol = ncol(X))

  for ( j in 1:ncol(X)){

    for ( k in 1:nrow(X)) {

      # choosing the split point to see which observations for a
      # given column / covariate j that is being
      # assigned region 1 and region 2.

      s <- X[k,j]
      R1 <- which(X[,j] <= s)
      R2 <- which(X[,j] > s)

      # Here we are looking if the size of the respective region
      # is smaller than the minimal leaf size l
      if (length(R1) < l || length(R2) < l) {next}

      # below we are calculating the c1^ and the c2^.
      # this is being done based on the formula for
      # the c1 and c2 where taking the mean of
      # variable y for the specific region.
      c1 <- mean(y[R1])
      c2 <- mean(y[R2])
      # computing the sum of squares for a given k and given j
      SS[k,j] <-  sum( (y[R1] - c1)^2 ) + sum( (y[R2] - c2)^2 )
    }
  }

  # here we are looking at the value of the
  # minimal sum of squares
```

```
  minSS <- min(SS)
  indexing <- which(SS == minSS, arr.ind = TRUE)

  # here we are looking for which row k and for which column j
  # that corresponded to the minimal value of the
  # sum of squares in  SS matrix.

  k <- indexing[1,1] # the row where the SS is the lowest
  j <- indexing[1,2] # the column where the SS is the lowest

  # now when the specific row and the specific column
  # for the min value of SS has been found we are
  # attaching that to the design matrix X to find
  # the split point.

  s <- X[k, j]
  R1 <- which(X[,j] <= s)
  R2 <- which(X[,j] > s)

  names(j) <- NULL
  names(R1) <- NULL
  names(R2) <- NULL

  output <- list(j = j,
                 s = s,
                 R1 = R1,
                 R2 = R2,
                 SS = min(SS))
  return(output)
}
```

Now when the function has been implemented we will check if the funciton does work as it should and for this we have been given a subset of the design matrix and the variable y to test this on. For the first test we will have 5 as the minimal leaf size and for the second test we will have 1 as the minimal leaf size. This wil be implemented using the implemented function.

```
X_check <- as.matrix(Hitters[31:50, c("Years", "Hits")])
y_check <- as.matrix(Hitters[31:50, c("Salary")])

tree_split(X_check, y_check, l = 5)
```

```
## $j
## [1] 1
##
## $s
## [1] 5
##
## $R1
##  [1]  1  2  3  5  6  9 13 16 18 19
##
## $R2
##  [1]  4  7  8 10 11 12 14 15 17 20
```

```
##
## $SS
## [1] 1346633
```

```
tree_split(X_check, y_check, l = 1)
```

```
## $j
## [1] 2
##
## $s
## [1] 132
##
## $R1
##  [1]  1  2  3  4  5  6  8  9 11 12 13 14 16 17 18 19
##
## $R2
## [1]  7 10 15 20
##
## $SS
## [1] 904383.4
```

When comparing the result above with the expected one then we have that the output is as expected for the subset data. Hence, the function work as it should.

1.3 & 1.4 Finding first split and SS for the training data

For this subtask we will look for the first split and its sum of squares, denoted as SS, when using the training data set and $l = 5$. We will use the tree split function created above which computes the first split and the SS.

```
# Finding the first split for the training data given l = 5
X_train <- as.matrix( Hitters[31:nrow(Hitters),c("Years", "Hits") ] )
y_train <- as.matrix( Hitters[31:nrow(Hitters),c("Salary")] )
tree.train <- tree_split(X = X_train, y = y_train, l = 5)
tree.train$s #4
```

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

```
tree.train$SS # 38464163
```

```
## [1] 38464163
```

When looking at the output we can see that the first split corresponds to $s = 4$ and the Sum of squares for the first split corresponds to $SS = 38464163$.

1.5 Build a decision tree

Now, we will use the tree split function to create a function grow tree that as before takes on a design matrix X, variable y and the minimum leaf size l. However, the grow tree functions returns a data frame consisting of the covariate used for the split, the split value, the regions and the prediction for a specific region $\hat{\gamma}_m$.

```
grow_tree <- function(X, y, l) {

  checkmate::assert_matrix(X)
  checkmate::assert_numeric(y, len = nrow(X))
  checkmate::assert_int(l)
```

```r
# as a first step we are computing the initial split using the
# tree_split function created above and extracting which observations
# that has been assigned which region. For this function we have that
# R1 is the first region when not split has been done ie R1 consist of
# the whole design matrix X. Hence, the first split is where the obs
# in the design matrix X is being assigned either R2 or R3.

init <- tree_split(X, y, 1)

# Here we are extracting the observations being assigned R1 and R2
# from the output and these values is being used in the while loop
# as well as in the if statement below.

# This is being done to know in which region
# each observation has been assigned.

S_m <- list(init$R1, init$R2)

# Storing the results
# j = column to use for split
# s = split point
# R1_i = pointer to row where to go next if in R1
# R2_i = pointer to row where to go next if in R2
# gamma = the gamma value of the leaf (if j and s are NA)
R1_i <- 2
R2_i <- 3

# This is the initial split and the first regions are the 2 and 3,
# this will be assigned, combined with j, s, to the first row of
# the output data matrix.

results <- data.frame(j = init$j,
                      s = init$s,
                      R1_i = R1_i,
                      R2_i = R2_i,
                      gamma = NA)

# When all regions has been created and these regions
# has been assigned a prediction / gamma hat then
# S_m will be empty and we end the while loop


while (length(S_m) > 0){

  # As long as not all parts of the tree has been handled
  # we will either split or compute the gamma hat.
  # The if statement below are looking if the length of the R1 are
  # at least twice the size of the minimal leaf size l then we are
  # continue splitting otherwise we are computing gamma gamma hat
  # ( this happens in the else section ).
```

```
if(length(S_m[[1]]) >= 2*l){

  # Doing stuff here to grow the tree into a decision tree
  # here we are doing an extra split where we are using the
  # tree split function created above but the only difference
  # is that we are indexing the design matrix X and
  # Y based on the S_m which is keeping track on
  # which observations that has been assigned which region.

  # we are indexing the design matrix and the variable based on S_m[[1]]
  # since we are going to remove S_m[[1]] for each loop in the while loop
  # by S_m[[1]]  <- NULL

  splittig_new <- tree_split(X[S_m[[1]],,drop = FALSE], y[S_m[[1]]], l)

  # it happens that the tree split function are not able to do a extra split that
  # satisfies the leaf size condition hence it can be seen that the
  # observation has reached the end region ie the leaf hence we
  # solve this problem by computing the gamma hat for that
  # region when this happens using a if statement.


  if (is.infinite(splittig_new$SS)) {
    gamma <- mean(y[S_m[[1]]])

    new_results <- data.frame(j = NA,
                              s = NA,
                              R1_i = NA,
                              R2_i = NA,
                              gamma = gamma)
    results <- rbind(results,
                     new_results)

    S_m[[1]] <- NULL

  } else {

    # if being in the else block then we have that splittig_new£SS < inf
    # and we are splitting the tree and letting it grow.

    # adjusting the region because we are working with
    # binary steps we are just adjusting by adding each region
    # with 2 for each loop in the while loop until we have that S_m
    # is empty.


    R1_i <- R1_i + 2
    R2_i <- R2_i + 2

    new_results <- data.frame(j = splittig_new$j,
```

```
                                  s = splittig_new$s,
                                  R1_i = R1_i,
                                  R2_i = R2_i,
                                  gamma = NA)

        results <- rbind(results,
                         new_results)
        # Add R1 and R2 to S_m
        S_m <- c(S_m, list(S_m[[1]][splittig_new$R1],
                           S_m[[1]][splittig_new$R2]))

        #Remove the set we just handled
        S_m[[1]] <- NULL

      }
    } else {

    # This else block is related to the first if statement
    # where we are testing the leaf size for the different
    # regions and comparing it with l. and here we have
    # that the length(S_m[[1]]) < 2*l so we predict by
    # calculating the gamma hat

        gamma <- mean(y[S_m[[1]]])

        new_results <- data.frame(j = NA,
                                  s = NA,
                                  R1_i = NA,
                                  R2_i = NA,
                                  gamma = gamma)
        results <- rbind(results,
                         new_results)

        S_m[[1]] <- NULL


      }

    }
  return(results)
}
grow_tree(X_check, y_check, l = 5)
```

```
##    j   s R1_i R2_i    gamma
## 1  1   5    2    3       NA
## 2  1   3    4    5       NA
## 3  2 101    6    7       NA
## 4 NA  NA   NA   NA 106.5000
## 5 NA  NA   NA   NA 244.5000
## 6 NA  NA   NA   NA 509.3334
## 7 NA  NA   NA   NA 946.0000
```

1.6 Implementing a predicition with tree function

Now when the "grow tree" function has been created we are able to predict in what region the specific observation will end up as well as the regions estimated value denoted as $\hat{\gamma}_m$. This is being done in the function below prediction with tree. The function consist of two function arguments, the first is the new set observation called "new data" and the second is the decision tree that we will assign "new data" to.

```
X_new <- as.matrix(Hitters[51:52, c("Years", "Hits")])
y_new <- c(Hitters[51:52, c("Salary")])
tr <- grow_tree(X_check, y_check, l = 5)

predict_with_tree <- function(new_data, tree){

  # # checking if the object is matrix
  checkmate::assert_matrix(new_data)

  # the predictions object will be used for the sequence
  # in the for loop as well as the output
  # we will return in the function.

  predictions <- numeric(nrow(new_data))

  for(i in seq_along(predictions)){

    # we are checking if the specific observation within
    # the new_data object is in a leaf or not. If we have
    # that the specific observation is not in the leaf
    # then we continue splitting until it is.

    not_in_leaf <- TRUE
    r <- 1
    while(not_in_leaf == TRUE){

      # we are checking if the X_{i,j} for the new observations
      # is less (or equal) than or higher than the splitting value.
      # if we have that the x_{i,j} <= s then we are looking at the
      # R1_i column otherwise we are looking at the R2_i column
      # and this s being done until we get a prediction value for
      # the specific observation

      ifelse ( new_data[i,  tree$j[r] ] <= tree$s[r],
               region <- tree$R1_i[r] ,
               region <-  tree$R2_i[r] )

      predicted_value <- tree$gamma[r]

      # We have set not_in_leaf == TRUE to enter the while loop
      # and we are stopping the while loop if the predicted_value
      # is a real value since then not_in_leaf == FALSE stopping
      # the while loop
      not_in_leaf <- is.na(predicted_value)
      r <- region
```

```
    }
    predictions[i] <- predicted_value
  }
  return(predictions)
}
predict_with_tree(new_data = X_new, tree = tr)
```

```
## [1] 106.5 244.5
```

1.7 Root mean squared error RMSE

Now we will compute the root mean squared error, RMSE, on the test set for a tree trained on the whole training data.

```
output1 <- grow_tree(as.matrix(X_train), y_train, l= 5)

output2 <- predict_with_tree(new_data = cbind(as.matrix(X_test), y_test),
                             tree = output1)
rmse(y_test,output2)
```

```
## [1] 322.2891
```

The Root mean squared error is estimated to be $RMSE = 322.2891$.

Task 2: Running standard tools for boosting and random forest

2.1 Fit a random forest regression on the training data

Here we will fit a random forest regression based on the training data. Further, for the next subtask we will analyze fitted model and output.

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
set.seed(123)
data("Hitters")
Hitters <- Hitters[complete.cases(Hitters),]
dat_test <- Hitters[1:30, c("Salary","Years", "Hits")]
dat_train <- Hitters[31:nrow(Hitters), c("Salary","Years", "Hits")]
Hitters.rf <- randomForest(Salary ~ Years + Hits, data = dat_train)
Hitters.rf
```

```
##
## Call:
##  randomForest(formula = Salary ~ Years + Hits, data = dat_train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 1
##
##           Mean of squared residuals: 141888.5
##                     % Var explained: 34.68
```

2.2 How many variables where used in the split and why

One variable was used for each split. When computing a random forest regression model we have on default that the subset of covariates used in the split, k, being calculated as $k = max(\frac{K}{3}, 1)$ when $\frac{K}{3}$ has been rounded down to closest integer and where K is the number of columns in the design matrix X. Hence, the number of variables used for each split corresponds to $k = max(0, 1) = 1$ in our case.

2.3 Using trained random forest to predict at test set & RMSE

Now we will use the trained random forest object to predict using the predict function at the test set and after that has been done we will compute the RMSE of the predictions using rmse function.

```
prediction1 <- predict(Hitters.rf, newdata = dat_test)
rmse(prediction1, dat_test$Salary) # 233.9964
```

```
## [1] 233.9964
```

The root mean squared error for the random forest at test set corresponds to 233.9964.

2.4 Fit a boosted regression tree model to the training data

Now we will fit a boosted regression tree model on the training data and this will be done using xgboost function in R.

```r
library(xgboost)
X_test <- dat_test[, c("Years", "Hits")]
y_test <- dat_test[, c("Salary")]
X_train <- dat_train[, c("Years", "Hits")]
y_train <- dat_train[, c("Salary")]
xgb <- xgboost(as.matrix(X_train), as.matrix(y_train), nrounds = 200)
```

2.5 RMSE of the prediction using the boosted regression tree model

```r
prediction <- predict(xgb, newdata = as.matrix(X_test))
rmse(prediction, y_test) # 227.5978
```

```
## [1] 227.5978
```

$RMSE = 322.2891$.

2.6 Comparing boosted tree moedl with the random forest model

When comparing the prediction between the boosted tree model and the random forest model then we are looking at the RMSE on the test set. Worth mentioning is that the model has been trained on the training data. The prediction on the test data when using random forest model corresponds to 233.9964 while equivalent using boosted tree model corresponds to 227.5978 then we can see that the RMSE for the boosted tree model is lower than the RMSE for the random forest model.

Task 3 Bagged tree and random forest implementation

3.1.1 Creating the train bagged trees function

Here we will implement a bagged tree regression model using the grow tree function which was presented and implemented above. The function returns an R list of B numbers of grown trees.

```
train_bagged_trees <- function(X, y, l = 5, B){

  set.seed(123)

  dataset <- cbind(X, y)

  result <- list()

  for (i in 1:B) {

    drawn_sample <- dataset[sample(nrow(dataset),
                                   size = length(dataset[,3] ),
                                   replace = TRUE), ]

    grow_tree_boot <- grow_tree(X  = as.matrix(drawn_sample[,1:2] ),
                                y = drawn_sample[,3], l = l)

    result[[i]]  <- list(grow_tree_boot)

  }
  result <- list(result, B = B)
  return(result)
}
train_bagged_trees(X_train, y_train, l = 5, B = 5)
```

3.1.2 Create a ”predict with bagged trees” function

Now we will create a function that takes on a list of grown trees and produces one prediction per observation. The function is called ”predict with bagged trees” and is being presented according to below:

```
tr <- train_bagged_trees(X_train, y_train, l = 5, B = 5)

predict_with_bagged_trees <- function(new_data = X_new, tree) {

  B <- tree$B

  output <- matrix(NA, nrow = B, ncol = nrow(new_data))

  for (i in 1:B) {

  output[i,] <- predict_with_tree(new_data = new_data,
                              tree = tree[[1]][[i]][[1]] )

  }
  result <- colMeans(output)
```

```
  return(result)
}
X_new <- as.matrix(Hitters[51:52, c("Years", "Hits")])
predict_with_bagged_trees(X_new, tree = tr)
```

```
## [1] 103.8895 373.0000
```

3.1.3 Calculating the RMSE for the predictions

```
bagged_B1 <- train_bagged_trees(X_train, y_train,
                                l = 5, B = 1)

bagged_B10 <- train_bagged_trees(X_train, y_train,
                                 l = 5, B = 10)

bagged_B100 <- train_bagged_trees(X_train, y_train,
                                  l = 5, B = 100)

bagged_B500 <- train_bagged_trees(X_train, y_train,
                                  l = 5, B = 500)

bagged_B1000 <- train_bagged_trees(X_train, y_train,
                                   l = 5, B = 1000)

pred_B1 <- predict_with_bagged_trees(new_data = as.matrix(X_test),
                                     tree = bagged_B1)
rmse(y_test, pred_B1) # 437.0089
```

```
## [1] 437.0089
```

```
pred_B10 <- predict_with_bagged_trees(new_data = as.matrix(X_test),
                                      tree = bagged_B10)
rmse(y_test, pred_B10) # 289.0196
```

```
## [1] 289.0196
```

```
pred_B100 <- predict_with_bagged_trees(new_data = as.matrix(X_test),
                                       tree = bagged_B100)
rmse(y_test, pred_B100) # 272.7637
```

```
## [1] 272.7637
```

```
pred_B500 <- predict_with_bagged_trees(new_data = as.matrix(X_test),
                                       tree = bagged_B500)
rmse(y_test, pred_B500) # 277.1029
```

```
## [1] 277.1029
```

```
pred_B1000 <- predict_with_bagged_trees(new_data = as.matrix(X_test),
                                        tree = bagged_B1000)
rmse(y_test, pred_B1000)  # 274.0614
```

```
## [1] 274.0614
```

3.2.1 Creating the "grow tree m" function

The function is based on the grow tree function created previously but now we will create another function based on the grown tree function but with a additional function argument m. The function will draw a random sample size m of covariates to split the observations.

```
grow_tree_m <- function(X, y, l, B, m) {


  dataset <- cbind(X, y)

  result <- list()

  for (i in 1:B) {
  drawn_sample  <- dataset[sample(nrow(dataset), size = length(dataset[,3]),
                                  replace = TRUE),]

  X <- drawn_sample[,1:2]

  X_m <- as.matrix( X[, sample(c(1:ncol(X)), size = m, replace = FALSE)])

  result[[i]] <- list ( grow_tree(X  = X_m , y = drawn_sample[,3], l = 5)  )

  }
  result <- list(result, B = B, m = m)
  return(result)

}
```

```
grow_tree_m(X_train, y_train, B = 10, m = 1)
```

3.2.2 Implementing the random forest regression model

Now when the grown tree function with respect to sample m covariates has been created we will implement the random forest regression model by using the grow tree m function and the prediction with tree function created above.

```
regression_rf <- function(X_train, y_train, X_new, l = 5, m = 2 , B = 10) {

  tree  <-  grow_tree_m(X_train, y_train, l, B, m )

  B <- tree$B

  output <- matrix(NA, nrow = B, ncol = nrow(X_new))

  for (i in 1:B) {

  output[i,] <- predict_with_tree(new_data = X_new,
                            tree = tree[[1]][[i]][[1]] )

  }
  result <- colMeans(output)
  return(result)
```

```
}
regression_rf(X_train = X_train, y_train = y_train,
              X_new = as.matrix(X_test), l = 5, m = 1, B =10)
```

```
##  [1]  883.1262  411.5291  917.5729  480.4989  917.5729  480.4989  411.5291
##  [8]  480.4989 1205.1617  791.3617  945.7690  523.8910  729.8393 1205.1617
## [15]  672.3068  689.5630  842.9442  402.5564  402.5564  729.8393  654.2470
## [22]  729.8393  811.5656  791.3617  689.5630  883.1262  411.5291  689.5630
## [29]  811.5656  883.1262
```

3.2.3 Training random forest model for $B = 100$ and $m = 1$

Lastly, we will train our random forest regression model on the training data for $B = 100$ and sample covariate size set to $m = 1$ and predict on the test data. Further, we will compute the RMSE of the predictions.

```
prediction_rf <- regression_rf(X_train = X_train, y_train = y_train,
                               X_new = as.matrix(X_test),
                               l = 5, m = 1, B =10)
rmse(y_test, prediction_rf)
```

```
## [1] 357.1902
```

The root mean squared error for the prediction using the random forest regression model on the test set corresponds to 306.5823.