Uppsala University

Machine Learning

Assignment 2

General information

- The recommended tool in this course is R (with the IDE R-Studio). You can download R **here** and R-Studio **here**. You can use Python and Jupyter Notebooks, although the assignments may use data available only through the R package, a problem you would need to solve yourself.
- Report all results in a single, `*.pdf`-file. *Other formats, such as Word, Rmd, Jupyter Notebook, or similar, will automatically be failed.* Although, you are allowed first to knit your document to Word or HTML and then print the assignment as a PDF from Word or HTML if you find it difficult to get TeX to work.
- The report should be written in English.
- If a question is unclear in the assignment. Write down how you interpret the question and formulate your answer.
- You should submit the report to **Studium**. Deadlines for all assignments are **Sunday 23.59**. See **Studium** for dates. Assignments will be graded within 10 working days from the assignment deadline.
- To pass the assignments, *you should answer all questions not marked with **, and get at least 75% correct.
- To get VG on the assignment, *also the questions marked with a ** should be answered. Sometimes you can choose between different VG assignments, then this is explicitly stated. To get VG you need 75% both on questions to pass and the VG part of the assignment. VG will only be awarded on the first deadline of the assignment.
- A report that does not contain the general information (see the **template**), will be automatically rejected.
- When working with R, we recommend writing the reports using R markdown and the provided **R markdown template**. The template includes the formatting instructions and how to include code and figures.
- Instead of R markdown, you can use other software to make the pdf report, but you should use the same instructions for formatting. These instructions are also available in **the PDF produced from the R markdown template**.
- The course has its own R package `uuml` with data and functionality to simplify coding. To install the package just run the following:

  1. `install.packages("remotes")`
  2. `remotes::install_github("MansMeg/IntroML", subdir = "rpackage")`

- We collect common questions regarding installation and technical problems in a course Frequently Asked Questions (FAQ). This can be found **here**.
- You are not allowed to show your assignments (text or code) to anyone. Only discuss the assignments with your fellow students. The student that show their assignment to anyone else could also be considered to cheat. Similarly, on zoom labs, only screen share when you are in a separate zoom room with teaching assistants.

- The computer labs are for asking all types of questions. Do not hesitate to ask! The purpose of the computer labs are to improve your learning. We will hence focus on more computer labs and less on assignment feedback. *Warning!* There might be bugs in the assignments! Hence, it is important to ask questions early on so you dont waste time of unintentional bugs.
- If you have any suggestions or improvements to the course material, please post in the course chat feedback channel, create an issue **here**, or submit a pull request to the public repository.

Contents

1 Decision Trees

The dataset `Hitters` contain data on the salary of Baseball players and additional information on years in the baseball league (`Years`) and the number of hits (`Hits`). To access the data, use:

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

We are now going to build an algorithm to grow decision trees. The algorithm used below is based on the algorithm explained in Chapter 9.2.2 in Hastie et al. (2009). To split a tree, see Eq. (9.12-9.14).

1. Create a test set by setting aside the 30 first observations. Remove those observations from your training set.

```
# Remove NA values
Hitters <- Hitters[complete.cases(Hitters),]

# Create test and training set
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")]
```

2. (3p) Implement an R function to split observations binary greedily (ignore `NA` values). Here we use the Hastie et al. (2009, Eq. 9.12-9.14), but with the addition that we will keep at least l leafs. The function should take a design matrix `X`, an `y` variable and a minimal leaf size `l`. The function should return a list with the index set of the observations in `R1`, `R2`, the split value `s` and the covariate used for the split `j`. Below is a rough sketch how you can implement the function.

```
tree_split <- function(X, y, l){
  checkmate::assert_matrix(X)
  checkmate::assert_numeric(y, len = nrow(X))
  checkmate::assert_int(l)

  # Store the sum of squares
  SS <- matrix(Inf, nrow = nrow(X), ncol = ncol(X))
  for(j in 1:ncol(X)){
    for(k in 1:nrow(X)){
      s <- X[k,j] # Choose split point
      R1 <- which(X[,j] <= s)
      R2 <- which(X[,j] > s)

      # Do if R1 or R2 is smaller than the leaf size l
      # Compute c1
      # Compute c2
      SS[k, j] <- # Compute the sum of squares
    }
```

```
  }

  # The final results for the min SS (arg min)
  j <- # Which column was used for the split-point
  s <- # What value of X was used as the split-point
  R1 <- which(X[,j] <= s)
  R2 <- which(X[,j] > s)
  list(j = j,
       s = s,
       R1 = R1,
       R2 = R2,
       SS = min(SS))
}
```

Hint 1! You will need to handle ties (i.e. different observations with different values). This should not be a very large problem. In the example code, I choose the 'first' minima as:

```
which(min(SS), arr.ind = TRUE)[1,]
```

Hint 2! When making a split, if all possible split points for all input variables result in nodes with too few observations, then that node should be made into a leaf node. It is recommended to include this check inside `tree_split()`. Otherwise, `grow_tree()` (or the functions in the VG task) may get stuck in an infinite loop. If you suspect that your code is looping infinitely, try adding print statements inside `grow_tree()` to display the iteration number or the length of S_m at each step for debugging. This can help you see whether the tree is actually progressing.

See below for an example how the function should work.

```
# Lets choose a small data to try out our algorithm with
X_check <- Hitters[31:50, c("Years", "Hits")]
y_check <- Hitters[31:50, c("Salary")]
# These are the names of the players we look at
rownames(Hitters)[31:50]
```

```
##  [1] "-Bob Melvin"       "-BillyJo Robidoux" "-Bill Schroeder"
##  [4] "-Chris Bando"      "-Chris Brown"      "-Carmen Castillo"
##  [7] "-Chili Davis"      "-Carlton Fisk"     "-Curt Ford"
## [10] "-Carney Lansford"  "-Chet Lemon"       "-Candy Maldonado"
## [13] "-Carmelo Martinez" "-Craig Reynolds"   "-Cal Ripken"
## [16] "-Cory Snyder"      "-Chris Speier"     "-Curt Wilkerson"
## [19] "-Dave Anderson"    "-Don Baylor"
```

```
# This is how it should work
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
```

```
# We can also make a split without any limit on leaf size
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
```

3. (1p) What is the first split based on the whole training data `X_train` and `y_train`? Use `l=5`.

4. (1p) What is the Sum of Squares (SS) for this first split?

5. (3p) Use the function `tree_split()` to create a function `grow_tree()` that takes the arguments `X`, `y`, and `l` and then build a decision tree. The returned tree can be a `data.frame` that looks as below. This is just one example implementation, you are free to implement this however you want.

   Hint 1! Use a list in R to store the set of indecies you are splitting up.

   Hint 2! Remember that the indecies that `split_tree()` returns are based on the data you put into that functions. Hence, you need to convert them to the full data indecies to simplify.

   Here is an template how the function could be implemented, but you are free to implement it however you want.

```
grow_tree <- function(X, y, l=5){
  checkmate::assert_matrix(X)
  checkmate::assert_numeric(y, len = nrow(X))
  checkmate::assert_int(l)
```

```r
# We do an initial split
init <- tree_split(X, y, l)
# We use S_m to store the set of observation indicies
# This is used to keep track of the groups of
# observations to be split
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)
results <- data.frame(j = init$j,
                      s = init$s,
                      R1_i = -1,
                      R2_i = -1,
                      gamma = NA)
while (length(S_m) > 0) {
  # As long as not all parts of the tree has been handled
  # we will either split or compute lambda
  if(length(S_m[[1]]) >= 2*l){

    # Do stuff here to grow the tree
    # based on observations (indecies) in S_m[[1]]

    # Add split point
    new_results <- data.frame(j = j,
                              s = s,
                              R1_i = -1,
                              R2_i = -1,
                              gamma = NA)
    results <- rbind(results,
                     new_results)
    # Add R1 and R2 to S_m
    S_m <- c(S_m, R1, R2)
    # Remove the set we just handled
    S_m[[1]] <- NULL
  } else {

    # Do stuff here to when you cant grow the tree (i.e. compute gamma)

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

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

```
  }
  return(results)
}
```

Here is an example how it should work.

```
tr <- grow_tree(X_check, y_check, l = 5)
tr
```

```
##    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 106.5
## 5 NA  NA   NA   NA 244.5
## 6 NA  NA   NA   NA 509.3
## 7 NA  NA   NA   NA 946.0
```

6. (3p) Finally implement a function `predict_with_tree(new_data, tree)`. Here is a starting template for the function.

```
predict_with_tree <- function(new_data, tree){
  checkmate::assert_matrix(new_data)

  predictions <- numeric(nrow(new_data))

  for(i in seq_along(predictions)){
    not_in_leaf <- TRUE
    while(not_in_leaf){

      # Check the split and go to next row
      # until we end up with a gamma

    }
    predictions[i] <- predicted_value
  }
  return(predictions)
}
```

To use the tree to predict new observations X_{new} as follows.

```
X_new <- Hitters[51:52, c("Years", "Hits")]
X_new
```

```
##                Years Hits
## -Daryl Boston      3   53
## -Darnell Coles     4  142
```

```
y_new <- Hitters[51:52, c("Salary")]
y_new
```

```
## [1]  75 105
```

```
pred_y <- predict_with_tree(new_data = X_new, tree = tr)
pred_y
```

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

7. (1p) The function `rmse(x,y)` in the `uuml` R package computes the root mean squared error between `x` and `y`.

```
rmse
```

```
## function (x, y)
## {
##     checkmate::assert_numeric(x, any.missing = FALSE)
##     checkmate::assert_numeric(y, any.missing = FALSE, len = length(x))
##     sqrt(mean((x - y)^2))
## }
## <bytecode: 0x00000216d63a9418>
## <environment: namespace:uuml>
```

```
rmse(x = c(75, 105, 33), y = c(102, 99, 43))
```

```
## [1] 16.98038
```

What is the root mean squared error (RMSE) on the test set for a tree trained on the whole training data `X_train` and `y_train`? Use `l=5`.

2 Running standard tools for boosting and random forests

The last part of the assignment consists of using Random Forest and XGBoost (Boosted regression trees). The data is taken in slightly different ways:

1. Use the `randomForest` R package to fit a random forest regression to the training data using the `randomForest` function.

```
library(uuml)
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)
```

2. (1p) How many variables are used at each split. Why do you get this number?
3. (1p) Use the trained random forest to predict using the `predict` function at test set and compute the RMSE of your predictions using the `rmse` function.
4. (1p) Next, use the `xgboost` R package to fit a boosted regression tree model to the training data using the `xgboost` function.

```
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)
```

5. (1p) What is the RMSE of the predictions (again using the `rmse` function) using the boosted regression tree model?
6. (1p) How does that compare to the random forest regression model?

3 *Bagged tree and random forest implementation

This task is only necessary to get one point toward a *pass with distinction* (VG) grade. Hence, if you do not want to get a *pass with distinction* (VG) point, you can ignore this part of the assignment.

Now we can use our regression tree to do a Bagged Tree regression model. The main difference from the previous tree is that we now will draw B bootstrap samples of size N *with replacement* from our dataset, then train B trees, use them for a combined bagged prediction.

See Section 8.7 in Hastie et al. (2009), and especially Eq. (8.51) for details.

1. (1p) Implement a bagged tree regression model training function called `train_bagged_trees(X, y, l, B)` using the `grow_tree()` function that return an R list of B grown trees.
2. (1p) Use `predict_with_tree()` to create a function `predict_with_bagged_trees()` that takes a list of grown trees and produce one prediction per observation.
3. (1p) Train your bagged tree regression model on the training data and predict on the test data. Try the values $B = \{1, 10, 100, 500, 1000\}$. What is the RMSE of your predictions on the test set? *Hint!* See Figure 15.1 in Hastie et al. (2009).

Using our previous functions we can now build our own random forest regression model using our previous functions `grow_tree()` and `predict_with_tree()`. Based on the bagged tree regression model, we only need to add the parameter m.

See Section 15.2 in Hastie et al. (2009), and especially Algorithm 15.1 for details.

1. (1p) Create a new function `grow_tree_m()` with the additional argument m. The function should grow a tree, but at every split, it should draw a sample of size m of the covariates to make the split.
2. (1p) Implement a random forest regression model using `grow_tree_m()` and `predict_with_tree()`.
3. (1p) Train your random forest regression model on the training data with $m = 1$, $B = 100$ and predict on the test data. What is the RMSE of your predictions on the test set (using your `rmse` function)?

   Note! Since we only have just two covariates in the test and train data, the difference is small (if any) compared to the Bagging algorithm. Also, differences compared to the `randomForest` package probably comes from different way the `grow_tree` functions are implemented in your code and the `randomForest` package.

References

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.