Uppsala University


Introduction to Machine Learning, Big Data, and AI

# 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.
- You should submit the report to **Studium**.
- To pass the assignments, *you should answer all questions not marked with **, although minor errors are ok.
- To get VG on the assignment, *all questions should be answered, including questions marked with a **, although minor errors are ok.
- 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**.
- Deadlines for all assignments are **Sunday 23.59**. See the course page for dates.
- 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.
     ```

# 1 General Questions

1. Describe bagging and boosting with your own words, whats the difference between the two approaches? (1-2 paragraphs)
2. When (and why) is random forests performing poorly with a small $m$? (max 1 paragraph)
3. What is a learning ensamble? (max 1 paragraph)

# 2 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 ESL. To split a tree, see Eq. (9.13) in ESL.

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. Implement an R function to split observations binary greedily (ignore `NA` values). Here we use the algorithm in ESL. 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`. See below for an example.

```
# 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  9 13 16 18 19
##
## $R2
##  [1]  4  6  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] 139
##
## $R1
##  [1]  1  2  3  4  6  8  9 11 12 13 14 16 17 18 19
##
## $R2
## [1]  5  7 10 15 20
##
## $SS
## [1] 904383.4
```

3. What is the first split based on the whole training data `X_train` and `y_train`?
4. What is the Sum of Squares (SS) for this first split?
5. 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 follows. Note that `R1_i` and `R2_i` indicates the row of the `data.frame` where to go next. This is just one example implementation, you

are free to implement this however you want. *Hint!* Use a list in R to store the set of indecies you are splitting up.

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

```
##    j   s R1_i R2_i    gamma
## 1  1   5    2    3       NA
## 2 NA   0   NA   NA 167.5000
## 3  2 101    4    5       NA
## 4 NA  NA   NA   NA 558.1250
## 5 NA  NA   NA   NA 358.2143
```

6. Finally implement a function `predict_with_tree(new_data, tree)` 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] 167.5 167.5
```

7. Implement a function `rmse(x,y)` that computes the root mean squared error between `x` and `y`. It should work like the following

```
rmse(y_new, pred_y)
```

```
## [1] 78.938
```

8. What is the root mean squared error on the test set for a tree trained on the whole training data `X_train` and `y_train`?

## 3 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. How many variables are used at each split. Why do you get this number?

3. Use the trained random forest to predict using the `predict` function at test set and compute the RMSE of your predictions using your `rmse` function.

4. 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[1:30, c("Years", "Hits")]
y_test <- dat_test[1:30, c("Salary")]
X_train <- dat_train[31:nrow(Hitters), c("Years", "Hits")]
y_train <- dat_train[31:nrow(Hitters), c("Salary")]
xgb <- xgboost(as.matrix(X_train), as.matrix(y_train), nrounds = 200)
```

5. What is the RMSE of the predictions (again using your `rmse` function) using the boosted regression tree model?

6. How does that compare to the random forest regression model?

## 4 *Bagged tree implementation

This assignment is a VG point assignment. If you do not want to get a VG-point, you can ignore this 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 ESL, and especially Eq. 8.51 for details.

1. Implement a bagged tree regression model training function called `train_bagged_trees(X, y, 1, B)` using the `grow_tree()` function that return an R list of $B$ grown trees.

2. 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. Train your bagged tree regression model on the training data and predict on the test data. Try the values $B = \{100, 500, 1000\}$. What is the RMSE (using your `rmse` function) of your predictions on the test set?

## 5 *Random forest implementation

This assignment is a VG point assignment. If you do not want to get a VG-point, you can ignore this assignment.

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 ESL, and especially Alg. 15.1 for details.

1. 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. Implement a random forest regression model using `grow_tree_m()` and `predict_with_tree()`.

3. 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).