# Final

Romain LEROY

24/05/2020

Final Exam: Topics to Financial Econometrics

Data Description

At first, we load the two datasets we have for the exam: the train data set that allows us to make the Machine Learning Algorithm and test it. The Final base is the data set where the 20 questions for the final exam is.

```
library('lattice')
library('ggplot2')
library('caret')
library('dplyr')
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
train_base = read.csv('C:/Users/romai/Documents/EDHEC/M1/S2/TTFE/final/pml-training.csv', header=T)
final_base = read.csv('C:/Users/romai/Documents/EDHEC/M1/S2/TTFE/final/pml-testing.csv', header=T)
```

The final base data is composed of 20 rows and 160 columns. The train base data is composed of 19622 rows and 160 columns too. Final base and train base have the same columns except for the final one, since train base have the results and not the final base.

```
dim(final_base)
```

```
## [1]  20 160
```

```
dim(train_base)
```

```
## [1] 19622   160
```

With the function head(), we can show the name of the different columns in the data:

```
dim(final_base)
```

```
## [1]  20 160
```

As we can see looking directly in the data file, a lot of columns have plenty of N/A or plenty of empty cells. Therefore, we have to clean the data. I decided to remove the column manually because the function is.na() does not work on one data set and I do not find a solution. Moreover, I could code the cleaning with a if and for conditions. However, it would have decrease the speed of the algorithm, which is not interesting for us.

```
final_base = final_base[,c(8:11,37:49,60:68,84:86,102,113:124,140,151:160)]
dim(final_base)
```

```
## [1] 20 53
```

```
train_base = train_base[,c(8:11,37:49,60:68,84:86,102,113:124,140,151:160)]
dim(train_base)
```

```
## [1] 19622    53
```

After deleting the data with N/A, empty cells and rows containing date and username, we have now 53 columns that are relevant for the Machine Learning implementation.

We make a Cross Validation of the train_base in order to have a train set and a test set for the Machine Learning. Wwe decided to take p =0.7, i.e. 70% of the data set for the train set and 30% for the test set. We set set.seed(23139) in order to have the same data set when we restart the cross validation, and so have the same result in the Machine Learning

```
set.seed(23139)
cv <- createDataPartition(train_base$classe, p=0.70, list=F)
data_train <- train_base[cv, ]
data_test <- train_base[-cv, ]
```

We want to predict in which manner an athlete perform well an exercise, which is complied in the column “classe”, with 5 grades: [A,B,C,D,E]. Therefore, it seems obvious that the best Machine Learning technique will be a decision tree. Since we want the best accuracy possible, a random forest analysis will be better than a decision tree, because it is a collection of decision tree.

```
resampling <- trainControl(method="cv", 5)
Random_forest <- train(classe ~ ., data= data_train, method="rf", trControl=resampling, ntree=250)
Random_forest
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10989, 10989, 10990, 10990, 10990
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9907551  0.9883046
##   27    0.9907553  0.9883045
##   52    0.9855144  0.9816728
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

As a result, we have an accuracy of 0.99 with our method, which is very good.

We apply then our algorithm to our data_test, in order to validate our result before the final test.

```
test <- predict(Random_forest,data_test)
confusionMatrix(table(data_test$classe, test))
```

```
## Confusion Matrix and Statistics
##
##    test
##        A    B    C    D    E
##   A 1672    1    1    0    0
##   B   12 1125    2    0    0
##   C    0    6 1013    7    0
##   D    0    1   16  946    1
##   E    0    2    4    1 1075
##
## Overall Statistics
##
##                Accuracy : 0.9908
##                  95% CI : (0.988, 0.9931)
##     No Information Rate : 0.2862
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9884
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9929   0.9912   0.9778   0.9916   0.9991
## Specificity            0.9995   0.9971   0.9973   0.9963   0.9985
## Pos Pred Value         0.9988   0.9877   0.9873   0.9813   0.9935
## Neg Pred Value         0.9972   0.9979   0.9953   0.9984   0.9998
## Prevalence             0.2862   0.1929   0.1760   0.1621   0.1828
## Detection Rate         0.2841   0.1912   0.1721   0.1607   0.1827
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9962   0.9941   0.9876   0.9940   0.9988
```

We have an accuracy on the data test of 0.9908, which is very good. Therefore, we can apply our algorithm to the final test data in order to have the result. At first, we remove the “id_number” column (the last one), since it is not relevant for the algorithm.

```
final_base = final_base[,-53]
final <- predict(Random_forest,final_base)
final
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```

This is the result of the application of our ML random forest to the final data.