Predicting exercise quality with machine learning

Philipp Knoepfle

Disclaimer

This repository contains the final programming assignment for the practical machine learning course in the Data Science specialization of JHU on Coursera.

Executive Summary

Thanks to new fitness devices such as *Jawbone Up, Nike FuelBand,* and *Fitbit* it is more easy to collect a large amount of measurement about personal activity. This data is used by quantitatively oriented fitness geeks to steadily improve their physical performance. One thing that people regularly measure is **how much or long** of an activity they perform, however they rarely quantify ** how well** they do it. In this project our goal is to use data of accelerometers on the belt, glove, upper arm, and dumbell to predict if an exercise is correctly performed or not. Further, we assess the feasibility of automatically assessing the quality of execution of weight lifting exerises. We gratefully acknowledge the provision of the data by Velloso et al., 2013, see http://groupware.les.inf.puc-rio.br/work.jsf?p1=11201).

Six young men (aged 20-28) were asked to perform one set of 10 repetitions. Each repetition counts as one observation. Participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fash- ions: exactly according to the specification (Class A), throw- ing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E). Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common lifting mistakes.

After an extensive cleaning of the data set. We test our hypothesis by emplyoing machine learning techniques, i.e. we strive to find accurate predictions if an exercise was done correctly or not. We test three different algorhythms with machine learning: a random forest model ("rf"), boosting with trees ("gbm"), and linear discriminant analysis ("lda"). The random forest model has the best accuracy which is why we use it for predicting the test data set.

Load and explore the data

Let's start by loading and exploring the data set.

```r
# Change the working directory (not really necessary)
setwd("C:\\Users\\user1\\Desktop\\Data Science\\8. Machine Learning\\Programming Assignment")

# Load packages
library(caret)
library(dplyr)
library(parallel)
library(doParallel)

# Download and load Files.
train.url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
download.file(train.url, destfile="pml-training.csv")
test.url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
```

```r
download.file(test.url, destfile="pml-testing.csv")
WLTraining <- read.csv("pml-training.csv")
WLTesting <- read.csv("pml-testing.csv")
# Do not get confused "WLTraining" refers to the whole data set we use for the machine learning exercis
# "WLTesting" refers to the data set we need for the quiz.

# Let us check how large our data sets are:
dim(WLTraining)
```

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

```r
dim(WLTesting)
```

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

Let us split the data set into a "training" and "testing" set so we can proceed.

```r
set.seed(0-100) # Cause our analysis goes from 0-100 yo!

trainid <- createDataPartition(WLTraining$classe, p=0.7, list=F)

Training <- WLTraining[trainid,]
Testing  <- WLTraining[-trainid,]

# Next we will save our outcome to be predicted in another variable
train.classe <- WLTraining[trainid, "classe"]
test.classe  <- WLTraining[-trainid, "classe"]

# Finaly our data can be explored:
dim(Training)
```

```
## [1] 13737   160
```

```r
dim(Testing)
```

```
## [1] 5885  160
```

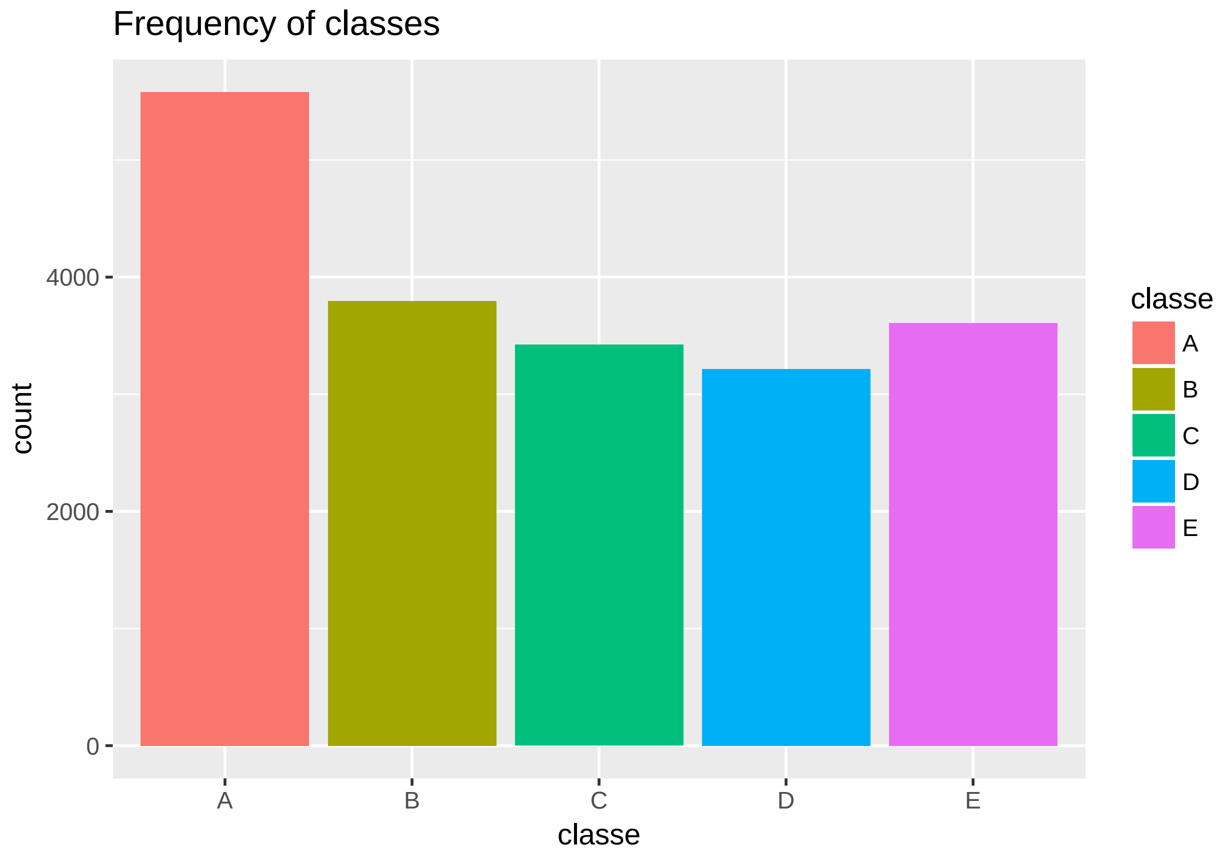
```r
str(Training)
```

```
## 'data.frame':    13737 obs. of  160 variables:
##  $ X                       : int  1 2 3 5 6 7 8 10 11 12 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1
##  $ raw_timestamp_part_2    : int  788290 808298 820366 196328 304277 368296 440390 484434 500302 528
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.45 1.42 1.42 1.45 1.45 1.43 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.07 8.06 8.09 8.13 8.17 8.18 8.18 ...
##  $ yaw_belt                : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt      : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_belt     : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt      : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt.1    : Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
```

```
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : Factor w/ 4 levels "","#DIV/0!","0.00",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.02 0.02 0.03 0.03 0.02 ...
##  $ gyros_belt_y            : num  0 0 0 0.02 0 0 0 0 0 0 ...
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 -0.02 0 -0.02 -0.02 ...
##  $ accel_belt_x            : int  -21 -22 -20 -21 -21 -22 -22 -21 -21 -22 ...
##  $ accel_belt_y            : int  4 4 5 2 4 3 4 4 2 2 ...
##  $ accel_belt_z            : int  22 22 23 24 21 21 21 22 23 23 ...
##  $ magnet_belt_x           : int  -3 -7 -2 -6 0 -4 -2 -3 -5 -2 ...
##  $ magnet_belt_y           : int  599 608 600 600 603 599 603 609 596 602 ...
##  $ magnet_belt_z           : int  -313 -311 -305 -302 -312 -311 -313 -308 -317 -319 ...
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 22 21.9 21.8 21.6 21.5 21.5 ...
##  $ yaw_arm                 : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm         : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ var_accel_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x             : num  0 0.02 0.02 0 0.02 0 0.02 0.02 0.02 0.02 ...
##  $ gyros_arm_y             : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 -0.03 ...
##  $ gyros_arm_z             : num  -0.02 -0.02 -0.02 0 0 0 0 -0.02 0 0 ...
##  $ accel_arm_x             : int  -288 -290 -289 -289 -289 -289 -289 -288 -290 -288 ...
##  $ accel_arm_y             : int  109 110 110 111 111 111 111 110 110 111 ...
##  $ accel_arm_z             : int  -123 -125 -126 -123 -122 -125 -124 -124 -123 -123 ...
##  $ magnet_arm_x            : int  -368 -369 -368 -374 -369 -373 -372 -376 -366 -363 ...
##  $ magnet_arm_y            : int  337 337 344 337 342 336 338 334 339 343 ...
##  $ magnet_arm_z            : int  516 513 513 506 513 509 510 516 509 520 ...
##  $ kurtosis_roll_arm       : Factor w/ 330 levels "","-0.02438",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_arm      : Factor w/ 328 levels "","-0.00484",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_arm        : Factor w/ 395 levels "","-0.01548",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
##  $ skewness_roll_arm       : Factor w/ 331 levels "","-0.00051",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_arm      : Factor w/ 328 levels "","-0.00184",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_arm        : Factor w/ 395 levels "","-0.00311",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ roll_dumbbell           : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell          : num  -70.5 -70.6 -70.3 -70.4 -70.8 ...
##  $ yaw_dumbbell            : num  -84.9 -84.7 -85.1 -84.9 -84.5 ...
##  $ kurtosis_roll_dumbbell  : Factor w/ 398 levels "","-0.0035","-0.0073",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_dumbbell : Factor w/ 401 levels "","-0.0163","-0.0233",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_dumbbell  : Factor w/ 401 levels "","-0.0082","-0.0096",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_dumbbell : Factor w/ 402 levels "","-0.0053","-0.0084",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

```
ggplot(WLTraining, aes(x=classe, fill=classe)) + geom_bar() + ggtitle("Frequency of classes") + labs(ti
```



This plot shows the absolute frequencies of different classes.

Taking a closer look at the data set, we find three mentionable points: 1. The data set contains variables not relevant for our analysis. That is, the first seven columns contain information on subjects. The last column represents the outcome "classe". 2. Many variables have been coded as factors whereas they are obviously numerical, see for instance the column "skewness_roll_dumbbell". 3. The data set includes a lot of NA values.

The first two problems can be easily alleviated by the following code:

```
Training <- as.data.frame(apply(Training[,-c(1:7, 160)], 2, as.numeric))
```

```
## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
```

```
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
```

```
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Training[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt
```

```r
Testing  <- as.data.frame(apply(Testing[,-c(1:7, 160)],  2, as.numeric))
```

```
## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt
```

```
## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt
```

```
## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt

## Warning in apply(Testing[, -c(1:7, 160)], 2, as.numeric): NAs durch
## Umwandlung erzeugt
```

We exclude columns 1-7 and 160 from the data set and turn all variables into numeric ones. The latter is obviously a shortcut which in fact could introduce a bias to our analysis. Yet, the number of factors in our sample is small. Overall, I think this will not introduce a large problem.

The third point made is a bit trickier to solve. Let us check how much of a problem NA values are by finding out how many columns have more than 95% of missing values:

```
NAdata <- colMeans(is.na(Training))
NAdata <- NAdata[NAdata>.95]
length(NAdata)
```

```
## [1] 100
```

That is quite a large number. However, those columns have little to contribute we will exclude them from our data set.

```
Training <- dplyr::select(Training, which(round(apply(is.na(Training), 2, sum ) / dim(Training)[1], 2) 
dim(Training)

Testing <- dplyr::select(Testing, which(round(apply(is.na(Testing), 2, sum ) / dim(Testing)[1], 2) < .9
```

That looks quite cleaned up. As a nice side effect the exclusion of those columns will make our machine learing algorhythm go faster later on! As a last check we are going to use the "nearZeroVar" function to check if there are zero or near zero covariates

```
nearZeroVar(Training)
```

```
## integer(0)
```

That is basically it vis-à-vis the cleaning, as a next step we will finally build our machine learning model.

Machine learning model

Now before we start, I want to prepare my setup such that our analysis runs fast as possible. This guide by (link) is very heplful in this instance. Some machine learning models, in particular random forests, may take up lots of your computing power. Hence, it makes sense to make use of more than one core (R-default). The following code allows us to do this:

```
cluster <- makeCluster(detectCores()-1)
# -1 core to let the OS run on it.
registerDoParallel(cluster)

fitControl <- trainControl(method = "cv",
```

```
                               number = 5,
                               allowParallel = TRUE)
```

We can finally estimate our machine learning models. We run four models: random forests, boosting with trees, and linear discriminant analysis:

```
model_1 <- train(x=Training, y=train.classe, method = "rf", trControl = fitControl, verbose = FALSE)
model_2 <- train(x=Training, y=train.classe, method = "gbm", trControl = fitControl, verbose = FALSE)
model_3 <- train(x=Training, y=train.classe, method = "lda", trControl = fitControl, verbose = FALSE)

# Do not forget to close to stop the cluster we set up:
stopCluster(cluster)
registerDoSEQ()

# Create predictions
prediction_1 <- predict(model_1, Testing)
prediction_2 <- predict(model_2, Testing)
prediction_3 <- predict(model_3, Testing)

# Create prediction matrices
conf_matrix_1 <- confusionMatrix(prediction_1, test.classe)
conf_matrix_2 <- confusionMatrix(prediction_2, test.classe)
conf_matrix_3 <- confusionMatrix(prediction_3, test.classe)

# Column 8 is the model accuracy
conf_matrix_1$table; conf_matrix_1$overall
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1674   12    0    0    0
##          B    0 1122    4    0    0
##          C    0    5 1016   10    3
##          D    0    0    6  952    3
##          E    0    0    0    2 1076
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9923534      0.9903259      0.9897815      0.9944172      0.2844520
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
conf_matrix_2$table; conf_matrix_2$overall
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1651   57    0    1    0
##          B   18 1051   28    2   14
##          C    5   27  980   29   13
##          D    0    3   16  929   13
##          E    0    1    2    3 1042
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9605777      0.9501033      0.9552874      0.9654049      0.2844520
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
conf_matrix_3$table; conf_matrix_3$overall
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1342  191   89   50   44
##          B   32  715   73   41  186
##          C  162  146  698  123   92
##          D  133   39  137  707   92
##          E    5   48   29   43  668
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.017842e-01   6.229110e-01   6.899132e-01   7.134548e-01   2.844520e-01
## AccuracyPValue  McnemarPValue
##   0.000000e+00   2.478063e-71
```

Looks like the random forest model has the highest accuracy. We need this because we want to predict 20 out of 20 questions correct. The chance of doing that with an accuracy of 0.99 is 81.8%. Let us predict the test set.

Predictions

The final step is to predict the test data set comprising of 20 observations, as proposed by the project's specification. We will use the random forest model above, since it gave the best predictions.

```
# Let us prepare the test data set for the prediction by applying the same
WLTesting <- as.data.frame(apply(WLTesting[,-c(1:7, 160)],  2, as.numeric))
WLTesting <- dplyr::select(WLTesting, which(round(apply(is.na(WLTesting), 2, sum ) / dim(WLTesting)[1],

pred_1 <- predict(model_1,WLTesting)
pred_1
```

```
##  [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
```

For this particular exercise, the random forest model ("rf") outperformed both the boosting with trees ("gbm") and linear discriminant analysis ("lda") models. The random forest model delivers overall 20 accurate predictions. Nevertheless, the gbm model performed close to the random forest, suggesting that it could be applicable to a real world scenario.

References

1. Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.