JH - M08 - Final assignment

shostiou

08/01/2021

Project Instructions

One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants.

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

The classe variable is encoded as follow :
Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly according to the specification (Class A), throwing 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).

Project Environment setup

Loading the required packages

```
library(tidyverse)
library(caret)
library(lubridate)
library(utils)
library(tidyr)
```

Collecting training and testing sets

```
# reading the csv files from the links provided in the instructions
training_csv <- download.file('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv','t
testing_csv <- download.file('https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv','tes
# Building training & testing dataframes
training_df <- read.csv('training.csv')
testing_df <- read.csv('testing.csv')
```

Setting seeds (for reproducibility)

```
set.seed(5977)
```

Data Cleaning

Let's start to have a look at the content of the training set

The training set contains 19622 observations of 160 variables.

Using the str command to get a clearer overview of the data contained into the training set.

```
str(training_df)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ 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 120339 196328 304277 368296 440390 484323 484
##  $ 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 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ 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 0.02 0.02 0.03 ...
##  $ gyros_belt_y            : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z            : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x           : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y           : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z           : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ 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 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y            : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z            : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x            : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y            : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z            : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x           : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y           : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z           : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ 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 -70.4 ...
##  $ yaw_dumbbell           : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ 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]
```

Dealing with NAs

It appears that a lot variables seem to have NAs. We need to quantify this in order to select irrelevant variables.
The code below calculates what is the proportion of NAs for each variable

```
training_na_pct <- training_df %>%
  summarise_all(funs(sum(is.na(.)))) %>% mutate_all(funs(./nrow(training_df))) %>%
  t()
```

We can observe that a significant number of variables contain more than 97% NA values.
A pragmatic approach will be to ignore those variables.

```
irrelevant_var <- names(training_na_pct[training_na_pct  > 0.9,])
irrelevant_var
```

```
##  [1] "max_roll_belt"            "max_picth_belt"
##  [3] "min_roll_belt"            "min_pitch_belt"
##  [5] "amplitude_roll_belt"      "amplitude_pitch_belt"
##  [7] "var_total_accel_belt"     "avg_roll_belt"
##  [9] "stddev_roll_belt"         "var_roll_belt"
## [11] "avg_pitch_belt"           "stddev_pitch_belt"
## [13] "var_pitch_belt"           "avg_yaw_belt"
## [15] "stddev_yaw_belt"          "var_yaw_belt"
## [17] "var_accel_arm"            "avg_roll_arm"
## [19] "stddev_roll_arm"          "var_roll_arm"
## [21] "avg_pitch_arm"            "stddev_pitch_arm"
## [23] "var_pitch_arm"            "avg_yaw_arm"
## [25] "stddev_yaw_arm"           "var_yaw_arm"
## [27] "max_roll_arm"             "max_picth_arm"
## [29] "max_yaw_arm"              "min_roll_arm"
## [31] "min_pitch_arm"            "min_yaw_arm"
## [33] "amplitude_roll_arm"       "amplitude_pitch_arm"
## [35] "amplitude_yaw_arm"        "max_roll_dumbbell"
## [37] "max_picth_dumbbell"       "min_roll_dumbbell"
## [39] "min_pitch_dumbbell"       "amplitude_roll_dumbbell"
## [41] "amplitude_pitch_dumbbell" "var_accel_dumbbell"
## [43] "avg_roll_dumbbell"        "stddev_roll_dumbbell"
## [45] "var_roll_dumbbell"        "avg_pitch_dumbbell"
## [47] "stddev_pitch_dumbbell"    "var_pitch_dumbbell"
## [49] "avg_yaw_dumbbell"         "stddev_yaw_dumbbell"
## [51] "var_yaw_dumbbell"         "max_roll_forearm"
## [53] "max_picth_forearm"        "min_roll_forearm"
## [55] "min_pitch_forearm"        "amplitude_roll_forearm"
## [57] "amplitude_pitch_forearm"  "var_accel_forearm"
## [59] "avg_roll_forearm"         "stddev_roll_forearm"
## [61] "var_roll_forearm"         "avg_pitch_forearm"
## [63] "stddev_pitch_forearm"     "var_pitch_forearm"
## [65] "avg_yaw_forearm"          "stddev_yaw_forearm"
## [67] "var_yaw_forearm"
```

There are 67 variables with a high proportion of NAs (superior to 90% of NAs)

A reasonable approach consists in removing those irrelevant variables.

```
training_df_01 <- training_df %>% select(-c(all_of(irrelevant_var)))
```

This new training set contains 19622 observations of 93 variables.

Next step is to identify variables having no content (encoded as "" which corresponds to no value and shall have been encoded as NA)

```
trainin_empty_pct <- training_df_01 %>%
 summarise_all(~sum(.=="")) %>% mutate_all(funs(./nrow(training_df_01))) %>%
  t()
```

Dealing with empty values

We can observe that a significant number of variables contain more than 97% "empty" values which shall have been encoded as NAs.
A pragmatic approach will be to ignore those variables.

```
irrelevant_var_02 <- names(trainin_empty_pct[trainin_empty_pct  > 0.9,])
irrelevant_var_02
```

```
##  [1] "kurtosis_roll_belt"      "kurtosis_picth_belt"
##  [3] "kurtosis_yaw_belt"       "skewness_roll_belt"
##  [5] "skewness_roll_belt.1"    "skewness_yaw_belt"
##  [7] "max_yaw_belt"            "min_yaw_belt"
##  [9] "amplitude_yaw_belt"      "kurtosis_roll_arm"
## [11] "kurtosis_picth_arm"      "kurtosis_yaw_arm"
## [13] "skewness_roll_arm"       "skewness_pitch_arm"
## [15] "skewness_yaw_arm"        "kurtosis_roll_dumbbell"
## [17] "kurtosis_picth_dumbbell" "kurtosis_yaw_dumbbell"
## [19] "skewness_roll_dumbbell"  "skewness_pitch_dumbbell"
## [21] "skewness_yaw_dumbbell"   "max_yaw_dumbbell"
## [23] "min_yaw_dumbbell"        "amplitude_yaw_dumbbell"
## [25] "kurtosis_roll_forearm"   "kurtosis_picth_forearm"
## [27] "kurtosis_yaw_forearm"    "skewness_roll_forearm"
## [29] "skewness_pitch_forearm"  "skewness_yaw_forearm"
## [31] "max_yaw_forearm"         "min_yaw_forearm"
## [33] "amplitude_yaw_forearm"
```

There are 33 variables with a high proportion of empty values (superior to 90% of empty values)

A reasonable approach consists in removing those irrelevant variables.

```
training_df_02 <- training_df_01 %>% select(-all_of(irrelevant_var_02))
```

This new training set contains 19622 observations of 60 variables.

Suppressing timestamp, user_name, X variables

If we consider all observations as independent, then timestamp variables will only contribute to introduce bias into our model.
They need to be removed.
For the same bias control reason, we will exclude the user_name variable (otherwise, our model will not be able to generalize because it will be linked to a small sample of users appearing into the training set).
The X variable is just an index. It shall not be taken into account for defining our model.

```
# variables to be excluded
to_be_excluded <- c('X','user_name','raw_timestamp_part_1','raw_timestamp_part_2','cvtd_timestamp')
# Exclusion
training_df_03 <- training_df_02 %>% select(-all_of(to_be_excluded))
```

Near Zero Variability check

As introduced into the course, this check will be done thanks to the nearZeroVar function of the Caret package

```
training_df_03 %>% select(-classe) %>% nearZeroVar(.,saveMetrics = TRUE) %>% filter(nzv ==TRUE)
```

```
##            freqRatio percentUnique zeroVar  nzv
## new_window  47.33005    0.01019264   FALSE TRUE
```

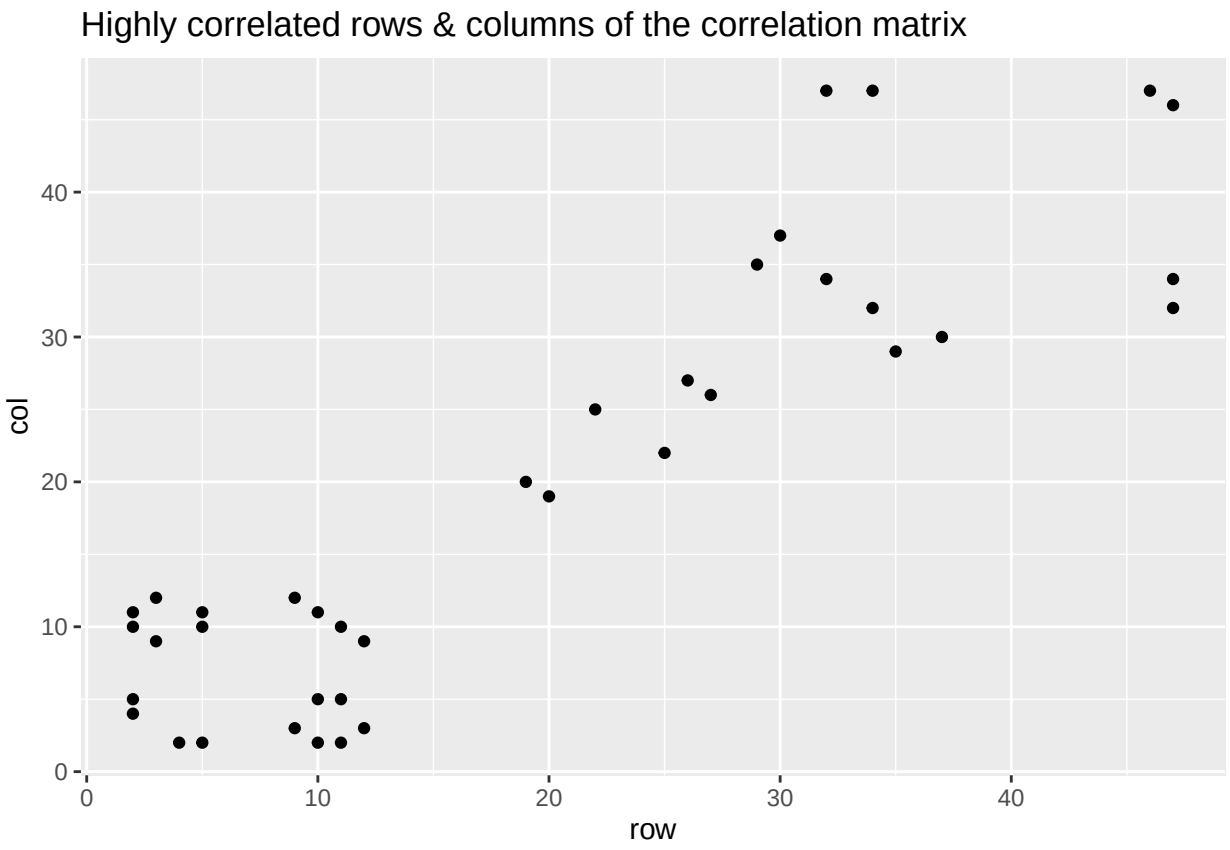
The new_window variable can be removed as it has a near zero variability

```
training_df_04 <- training_df_03 %>% select(-new_window)
```

Correlation between variables

The purpose here is to see if some variables are highly correlated in order to reduce the dimension of the model.

```
M <- training_df_04 %>% select(-classe) %>% cor() %>% abs()
diag(M) <- 0
corr_matrix <- which(M > 0.8,arr.ind=T) %>% as.data.frame()

ggplot(corr_matrix) +
  geom_point(mapping = aes(x=row,y=col)) +
  ggtitle('Highly correlated rows & columns of the correlation matrix')
```



Let's visualize the relationship between 2 of those highly correlated variables

```
var1 <- corr_matrix$row[20]
var2 <- corr_matrix$col[20]
```

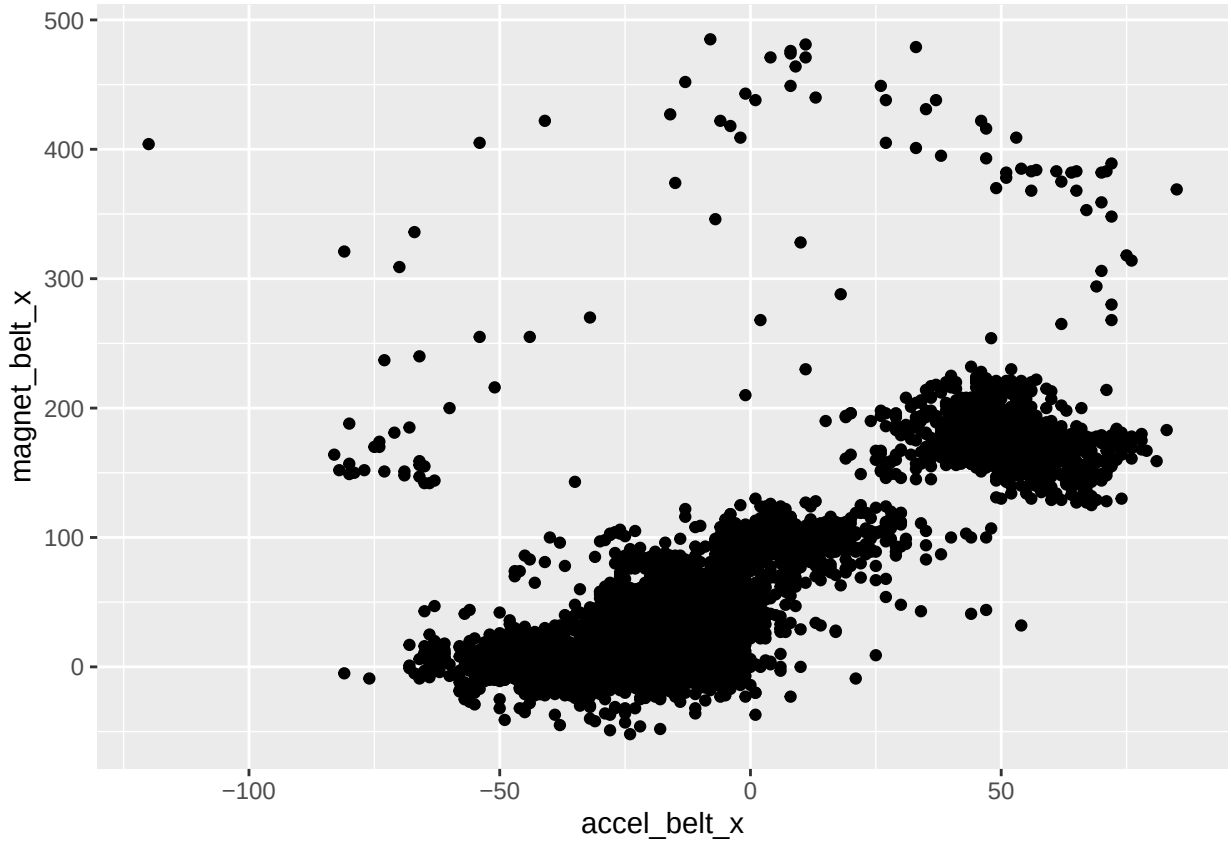
```
var1_col <- colnames(training_df_04[var1])
var2_col <- colnames(training_df_04[var2])

var1_col
```

```
## [1] "accel_belt_x"
```

```
var2_col
```

```
## [1] "magnet_belt_x"
```

```
ggplot()+
  geom_point(mapping = aes(x=training_df_04[,var1_col] ,y=training_df_04[,var2_col]))+
  xlab(var1_col)+
  ylab(var2_col)
```



On this specific example, we can identify a positive correlation with some outliers.

So, we can observe that some variables are highly correlated to each other.
This highlights that dimension reduction techniques shall be applied with the objective of simplifying our model.

```
PreProcPCA <- training_df_04 %>% select(-classe) %>% preProcess(.,method="pca")
PreProcPCA
```

```
## Created from 19622 samples and 53 variables
##
## Pre-processing:
##   - centered (53)
##   - ignored (0)
```

```
##   - principal component signal extraction (53)
##   - scaled (53)
##
## PCA needed 26 components to capture 95 percent of the variance
```

Principal Components Analysis allow to capture 95% of the variance with only 26 components. The predict function will be used to build our final training set

```
training_pca <- training_df_04 %>% select(-classe) %>% predict(PreProcPCA,.)
training_pca$classe <- training_df_04$classe
```

In order to prepare the cross validation, we need to Preprocess the test set like we did with the training set. Note that we will consider the "test" csv file as a validation set.

```
validation_df_01 <- testing_df %>% select(-c(all_of(irrelevant_var)))
validation_df_02 <- validation_df_01 %>% select(-all_of(irrelevant_var_02))
validation_df_03 <- validation_df_02 %>% select(-all_of(to_be_excluded))
validation_df_04 <- validation_df_03 %>% select(-new_window)

validation_pca <- validation_df_04 %>% select(-problem_id) %>% predict(PreProcPCA,.)
validation_pca$problem_id <- validation_df_04$problem_id
```

Training of the Model

The model will have to handle a classification problem (from a set of predictors => predict a "classe" variable). Random Forest appear to be one of the best method to be applied.
Our preprocessed training set (PCA) will be used to build the model.

Defining a test set

The testing set provided with the project instructions will be considered as a validation set.
In order to be able to perform cross validation, we need to split data contained into the "training" file into a training set and a test set.

```
inTrain <- createDataPartition(training_pca$classe,p=0.75,list=FALSE)
training_pca_mod <- training_pca[inTrain,]
testing_pca_mod <- training_pca[-inTrain,]
```

Model Training

We will train 2 different kind of models with the Caret package :
- Regression tree
- Random forest

Regression Tree - model fit

```
modFitTree <- train(classe ~ .,data =  training_pca_mod, method="rpart")
```

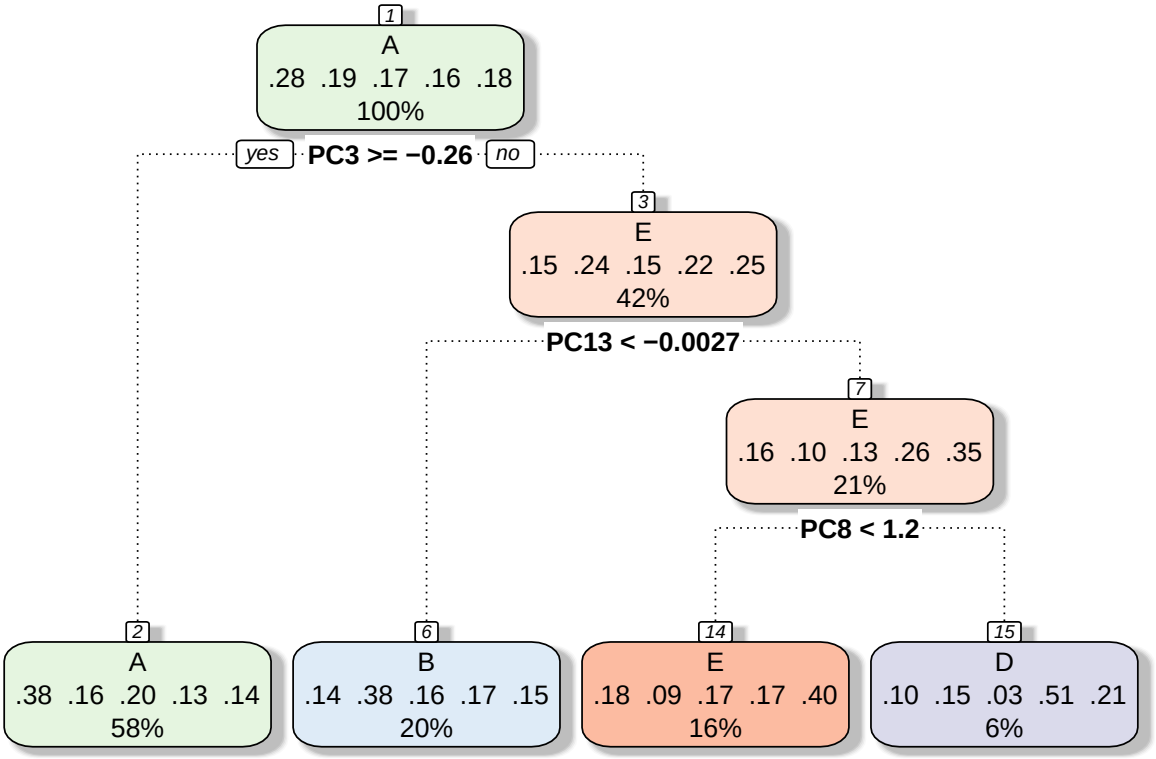
Plotting the Tree

```
library(rattle)
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Entrez 'rattle()' pour secouer, faire vibrer, et faire défiler vos données.
```

```
fancyRpartPlot(modFitTree$finalModel)
```



Random Forest - model fit

```
modFitRF <- train(classe ~ .,data =  training_pca_mod, method="rf", ntree=100)
```

Prediction Error (on the training set) - Regression Tree

```
train_pred <- training_pca_mod %>% select(-classe) %>% predict(modFitTree,.)
cm_train_tree <- confusionMatrix(train_pred,training_pca_mod$classe)
cm_train_tree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3273 1367 1669 1078 1158
##          B  415 1157  481  516  441
##          C    0    0    0    0    0
##          D   85  124   23  425  173
##          E  412  200  394  393  934
##
## Overall Statistics
##
##                Accuracy : 0.3933
##                  95% CI : (0.3854, 0.4013)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.1985
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7821  0.40625   0.0000  0.17620  0.34516
## Specificity            0.4995  0.84389   1.0000  0.96709  0.88353
## Pos Pred Value         0.3830  0.38439      NaN  0.51205  0.40034
## Neg Pred Value         0.8523  0.85557   0.8256  0.85693  0.85692
## Prevalence             0.2843  0.19350   0.1744  0.16388  0.18386
## Detection Rate         0.2224  0.07861   0.0000  0.02888  0.06346
## Detection Prevalence   0.5806  0.20451   0.0000  0.05639  0.15851
## Balanced Accuracy      0.6408  0.62507   0.5000  0.57165  0.61435
```

Prediction Error (on the test set) - Regression Tree

```
test_pred <- testing_pca_mod %>% select(-classe) %>% predict(modFitTree,.)
cm_test_tree <- confusionMatrix(test_pred,testing_pca_mod$classe)
cm_test_tree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1090  453  566  336  405
##          B  145  383  149  199  150
##          C    0    0    0    0    0
##          D   28   39    6  141   59
##          E  132   74  134  128  287
##
## Overall Statistics
##
##                Accuracy : 0.3876
##                  95% CI : (0.374, 0.4014)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.1908
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7814   0.4036   0.0000  0.17537  0.31853
## Specificity            0.4984   0.8374   1.0000  0.96780  0.88309
## Pos Pred Value         0.3825   0.3733      NaN  0.51648  0.38013
## Neg Pred Value         0.8515   0.8540   0.8257  0.85683  0.85201
## Prevalence             0.2845   0.1935   0.1743  0.16395  0.18373
## Detection Rate         0.2223   0.0781   0.0000  0.02875  0.05852
## Detection Prevalence   0.5812   0.2092   0.0000  0.05567  0.15396
## Balanced Accuracy      0.6399   0.6205   0.5000  0.57159  0.60081
```

Prediction Error (on the training set) - Random Forest

```
train_pred <- training_pca_mod %>% select(-classe) %>% predict(modFitRF,.)
cm_train_rf <- confusionMatrix(train_pred,training_pca_mod$classe)
cm_train_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 4185    0    0    0    0
##          B    0 2848    0    0    0
##          C    0    0 2567    0    0
##          D    0    0    0 2412    0
##          E    0    0    0    0 2706
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9997, 1)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1839
## Detection Rate         0.2843   0.1935   0.1744   0.1639   0.1839
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1839
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

Prediction Error (on the test set) - Random Forest

```
test_pred <- testing_pca_mod %>% select(-classe) %>% predict(modFitRF,.)
cm_test_rf <- confusionMatrix(test_pred,testing_pca_mod$classe)
cm_test_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1386    6    1    1    8
##          B    3  935   11    2    3
##          C    6    6  833   25    1
##          D    0    0    7  776    5
##          E    0    2    3    0  884
##
## Overall Statistics
##
```

```
##                Accuracy : 0.9816
##                  95% CI : (0.9775, 0.9852)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9768
##
##  Mcnemar's Test P-Value : 0.0002363
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9935   0.9852   0.9743   0.9652   0.9811
## Specificity            0.9954   0.9952   0.9906   0.9971   0.9988
## Pos Pred Value         0.9886   0.9801   0.9564   0.9848   0.9944
## Neg Pred Value         0.9974   0.9965   0.9945   0.9932   0.9958
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2826   0.1907   0.1699   0.1582   0.1803
## Detection Prevalence   0.2859   0.1945   0.1776   0.1607   0.1813
## Balanced Accuracy      0.9945   0.9902   0.9824   0.9811   0.9899
```

Interpretation

We can observe huge differences in accuracy between the 2 approaches :
- Regression tree - accuracy on training set : 0.3933279 on testing set : 0.3876427
- Random Forest - accuracy on training set : 1 on testing set : 0.9816476

The performance of random forest is perfect (Accuracy = 1) on the training set we could then suspect overfitting.
The accuracy on the test set is also nearly perfect and in any case far more accurate than the regression tree.

Based on the results of the cross validations, we will select the Random Forest as the best model and apply it to the validation set.

Validations

As a final step, we will predict the outcomes of the validation sets with the random forest model.

```
predict(modFitRF,validation_pca)
```

```
##  [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
```