# Machine learning Project

Emily Silva Araujo

24/05/2020

## Introduction

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. 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, we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants.The participants were asked to perform barbell lifts correctly and incorrectly in 5 different ways. The five ways are 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). Only Class A corresponds to correct performance. The goal of this project is to predict the manner in which they did the exercise, i.e., Class A to E. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

## Data Processing

### *Import the data*

We first load the R packages needed for analysis and then download the training and testing data sets from the given URLs.

### *Load the required packages*

```
library(caret)

library(rattle)

library(rpart)

library(rpart.plot)

library(randomForest)

library(repmis)

library(e1071)
```

## Import the data from the URLs

```
url_training <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-
training.csv"
url_testing <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-
testing.csv"

full_training <- read.csv(url(url_training))
full_testing <- read.csv(url(url_testing))
```

The training dataset has 19622 observations and 160 variables, and the testing data set contains 20 observations and the same variables as the training set. We are trying to predict the outcome of the variable classe in the training set.

## *Preprocessing*

We will remove unnecessary data in two steps. First, we remove data, which adds no or very litte information via a call to nearZeroVar. Second step, we will remove columns, which contains a high amount of NAs.

### *Get all columns that contain no data in testing set, so we can remove them from training set*

```
nvz <- nearZeroVar(full_training)
full_training <- full_training[,-nvz]
full_testing <- full_testing[,-nvz]
remove_cols <- c("X", "raw_timestamp_part_1", "raw_timestamp_part_2",
"cvtd_timestamp", "problem_id")
keep_cols <- !names(full_training) %in% remove_cols

full_training <- full_training[,keep_cols]
full_testing <- full_testing[,keep_cols]

remcols <- sapply(names(full_training), FUN = function(x) {
mean(is.na(full_training[x])) })
full_training <- full_training[,remcols <= 0.5]
full_testing <- full_testing[,remcols <= 0.5]

names(full_testing)

##  [1] "user_name"            "num_window"           "roll_belt"
##  [4] "pitch_belt"           "yaw_belt"             "total_accel_belt"
##  [7] "gyros_belt_x"         "gyros_belt_y"         "gyros_belt_z"
## [10] "accel_belt_x"         "accel_belt_y"         "accel_belt_z"
## [13] "magnet_belt_x"        "magnet_belt_y"        "magnet_belt_z"
## [16] "roll_arm"             "pitch_arm"            "yaw_arm"
## [19] "total_accel_arm"      "gyros_arm_x"          "gyros_arm_y"
## [22] "gyros_arm_z"          "accel_arm_x"          "accel_arm_y"
## [25] "accel_arm_z"          "magnet_arm_x"         "magnet_arm_y"
## [28] "magnet_arm_z"         "roll_dumbbell"        "pitch_dumbbell"
## [31] "yaw_dumbbell"         "total_accel_dumbbell" "gyros_dumbbell_x"
## [34] "gyros_dumbbell_y"     "gyros_dumbbell_z"     "accel_dumbbell_x"
## [37] "accel_dumbbell_y"     "accel_dumbbell_z"     "magnet_dumbbell_x"
## [40] "magnet_dumbbell_y"    "magnet_dumbbell_z"    "roll_forearm"
## [43] "pitch_forearm"        "yaw_forearm"
"total_accel_forearm"
## [46] "gyros_forearm_x"      "gyros_forearm_y"      "gyros_forearm_z"
## [49] "accel_forearm_x"      "accel_forearm_y"      "accel_forearm_z"
## [52] "magnet_forearm_x"     "magnet_forearm_y"     "magnet_forearm_z"
## [55] "problem_id"
```

```
names(full_training)
```

```
##  [1] "user_name"            "num_window"           "roll_belt"
##  [4] "pitch_belt"           "yaw_belt"             "total_accel_belt"
##  [7] "gyros_belt_x"         "gyros_belt_y"         "gyros_belt_z"
## [10] "accel_belt_x"         "accel_belt_y"         "accel_belt_z"
## [13] "magnet_belt_x"        "magnet_belt_y"        "magnet_belt_z"
## [16] "roll_arm"             "pitch_arm"            "yaw_arm"
## [19] "total_accel_arm"      "gyros_arm_x"          "gyros_arm_y"
## [22] "gyros_arm_z"          "accel_arm_x"          "accel_arm_y"
## [25] "accel_arm_z"          "magnet_arm_x"         "magnet_arm_y"
## [28] "magnet_arm_z"         "roll_dumbbell"        "pitch_dumbbell"
## [31] "yaw_dumbbell"         "total_accel_dumbbell" "gyros_dumbbell_x"
## [34] "gyros_dumbbell_y"     "gyros_dumbbell_z"     "accel_dumbbell_x"
## [37] "accel_dumbbell_y"     "accel_dumbbell_z"     "magnet_dumbbell_x"
## [40] "magnet_dumbbell_y"    "magnet_dumbbell_z"    "roll_forearm"
## [43] "pitch_forearm"        "yaw_forearm"
"total_accel_forearm"
## [46] "gyros_forearm_x"      "gyros_forearm_y"      "gyros_forearm_z"
## [49] "accel_forearm_x"      "accel_forearm_y"      "accel_forearm_z"
## [52] "magnet_forearm_x"     "magnet_forearm_y"     "magnet_forearm_z"
## [55] "classe"
```

The cleaned data sets full_testing and full_training both have 55 columns with the same first 55 variables. The training dataset has 19622 rows while testing dataset has 20 rows.

## Data spliting

In order to get out-of-sample errors, we split the cleaned training set trainData into a training set (train, 70%) for prediction and a validation set (valid 30%) to compute the out-of-sample errors.

```
  set.seed(7826)
inTrain <- createDataPartition(full_training$classe, p = 0.7, list =
FALSE)
train <- full_training[inTrain, ]
valid <- full_training[-inTrain, ]
```

## Prediction Algorithms

Use classification trees and random forests to predict the outcome.

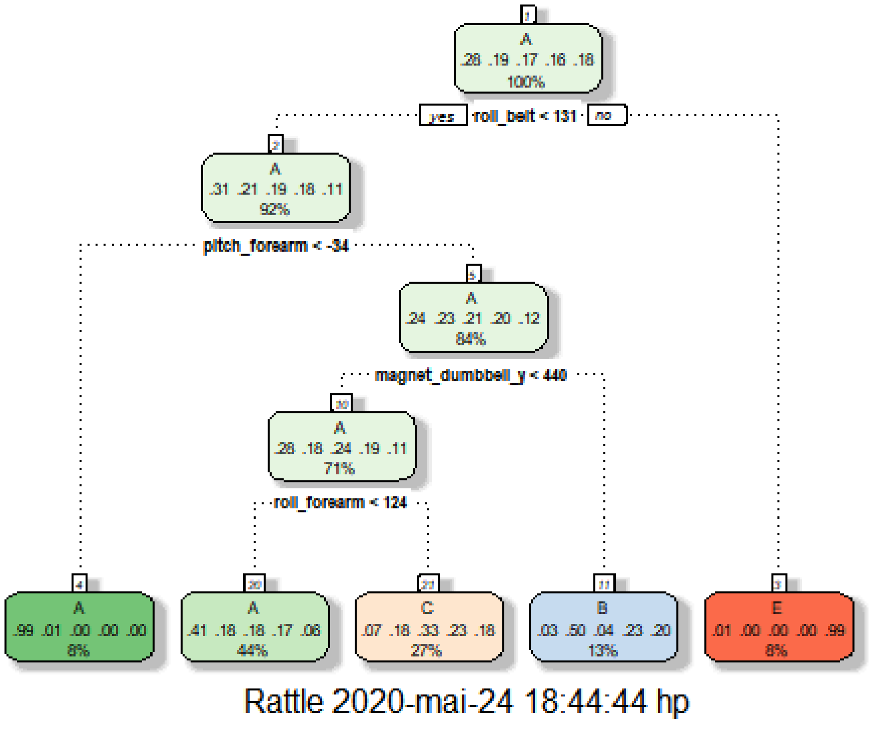
### *Classification trees*

In practice, k=5 or k=10 when doing k-fold cross validation. Here we consider 5-fold cross validation (default setting in trainControl function is 10) when implementing the algorithm to save a little computing time. Since data transformations may be less important in non-linear models like classification trees, we do not transform any variables.

```
control <- trainControl(method = "cv", number = 5)
fit_rpart <- train(classe ~ ., data = train, method = "rpart", trControl
= control)
print(fit_rpart, digits = 4)
```

```
## CART
##
## 13737 samples
##    54 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10990, 10990, 10990, 10989, 10989
## Resampling results across tuning parameters:
##
##   cp       Accuracy  Kappa
##   0.03982  0.5653    0.44478
##   0.05954  0.3673    0.12623
##   0.11586  0.3168    0.04946
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03982.
```

```
fancyRpartPlot(fit_rpart$finalModel)
```



*Predict outcomes using validation set*

```
predict_rpart <- predict(fit_rpart, valid)
```

## *Show prediction result*

```
(conf_rpart <- confusionMatrix(valid$classe, predict_rpart))

## Confusion Matrix and Statistics
##
##             Reference
## Prediction    A    B    C    D    E
##          A 1510   27  134    0    3
##          B  464  410  265    0    0
##          C  467   37  522    0    0
##          D  432  163  369    0    0
##          E  164  144  293    0  481
##
## Overall Statistics
##
##                Accuracy : 0.4967
##                  95% CI : (0.4838, 0.5095)
##     No Information Rate : 0.5161
##     P-Value [Acc > NIR] : 0.9986
##
##                   Kappa : 0.3425
##
##  Mcnemar's Test P-Value : NA
```

```
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4972  0.52497   0.3298       NA  0.99380
## Specificity            0.9424  0.85717   0.8828   0.8362  0.88872
## Pos Pred Value         0.9020  0.35996   0.5088       NA  0.44455
## Neg Pred Value         0.6374  0.92183   0.7816       NA  0.99938
## Prevalence             0.5161  0.13271   0.2690   0.0000  0.08224
## Detection Rate         0.2566  0.06967   0.0887   0.0000  0.08173
## Detection Prevalence   0.2845  0.19354   0.1743   0.1638  0.18386
## Balanced Accuracy      0.7198  0.69107   0.6063       NA  0.94126

(accuracy_rpart <- conf_rpart$overall[1])

##  Accuracy
## 0.4966865
```

From the confusion matrix, the accuracy rate is 0.49, and so the out-of-sample error rate is 0.51.Using classification tree does not predict the outcome classe very well.

## *Random forests*

Since classification tree method does not perform well, we try random forest method instead.

```
fit_rf <- train(classe ~ ., data = train, method = "rf", trControl =
control)

print(fit_rf, digits = 4)

## Random Forest
##
## 13737 samples
##    54 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10991, 10990, 10990, 10988, 10989
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.9937    0.9921
##   30    0.9968    0.9959
##   58    0.9950    0.9936
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 30.
```

Accuracy was used to select the optimal model using the largest value. The final value used for the model was mtry = 30.

### *predict outcomes using validation set*

```
predict_rf <- predict(fit_rf, valid)
```

### *Show prediction result*

```
(conf_rf <- confusionMatrix(valid$classe, predict_rf))

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    0    0    0    1
##          B    0 1139    0    0    0
##          C    0    2 1024    0    0
##          D    0    0    3  961    0
##          E    0    0    0    7 1075
##
## Overall Statistics
##
##                Accuracy : 0.9978
##                  95% CI : (0.9962, 0.9988)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9972
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9982   0.9971   0.9928   0.9991
## Specificity            0.9998   1.0000   0.9996   0.9994   0.9985
## Pos Pred Value         0.9994   1.0000   0.9981   0.9969   0.9935
## Neg Pred Value         1.0000   0.9996   0.9994   0.9986   0.9998
## Prevalence             0.2843   0.1939   0.1745   0.1645   0.1828
## Detection Rate         0.2843   0.1935   0.1740   0.1633   0.1827
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9999   0.9991   0.9983   0.9961   0.9988

(accuracy_rf <- conf_rf$overall[1])

## Accuracy
## 0.997791
```

For this dataset, random forest method is way better than classification tree method. The accuracy rate is 0.997, and so the out-of-sample error rate is 0.003. This may be due to the fact that many predictors are highly correlated. Random forests chooses a subset of predictors at each split and decorrelate the trees. This leads to high accuracy, although this algorithm is sometimes difficult to interpret and computationally inefficient.

## *Prediction on Testing Set*

We now use random forests to predict the outcome variable classe for the testing set.

```
(predict(fit_rf, full_testing))

##  [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
```