Practical Machine Learning Course Project

Marc Arroyo

25/1/2021

0. Introduction

This document is Marc Arroyo proposed solution for the Course Project of Practical Machine Learning module, in Data Science Specialization given by Johns Hopkins University via Coursera.

1. Overview

In this project we will use data of six individuals performing a set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions, one of them is the correct execution (Class A), and the other four are different wrong executions of the exercise (Classes B to E).

Using the data extracted from four different sensors being worn by the individuals during the execution of the different repetitions we will try to predict to which class does the repetition belong to.

1.1 Citation

This data has been generously given by:

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.

Read more in their home page

2. Data Processing

2.1. Preparing Environment

First thing we will do is to load required libraries, set common chunk parameters and locale language to English, as mine it is not.

```r
library(knitr)
library(caret)

opts_chunk$set(echo = TRUE, message = FALSE, warning = FALSE, error = FALSE)

Sys.setlocale("LC_ALL", "English")
```

2.2. Loading and Exploring Data

Now we will load training data and test data.

```r
rawdata <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")

validation <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")

dim(rawdata)
```

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

```r
dim(validation)
```

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

As we can see, both data.frame have the same number of columns, **160**, but while rawdata has **19622** rows, validation has only **20**.

```r
str(rawdata)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : chr  "carlitos" "carlitos" "carlitos" "carlitos" ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484
##  $ cvtd_timestamp          : chr  "05/12/2011 11:23" "05/12/2011 11:23" "05/12/2011 11:23" "05/12/20
##  $ new_window              : chr  "no" "no" "no" "no" ...
##  $ 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      : chr  "" "" "" "" ...
##  $ kurtosis_picth_belt     : chr  "" "" "" "" ...
##  $ kurtosis_yaw_belt       : chr  "" "" "" "" ...
##  $ skewness_roll_belt      : chr  "" "" "" "" ...
##  $ skewness_roll_belt.1    : chr  "" "" "" "" ...
##  $ skewness_yaw_belt       : chr  "" "" "" "" ...
##  $ 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            : chr  "" "" "" "" ...
##  $ 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            : chr  "" "" "" "" ...
##  $ 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      : chr  "" "" "" "" ...
##  $ 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       : chr  "" "" "" "" ...
##  $ kurtosis_picth_arm      : chr  "" "" "" "" ...
##  $ kurtosis_yaw_arm        : chr  "" "" "" "" ...
##  $ skewness_roll_arm       : chr  "" "" "" "" ...
##  $ skewness_pitch_arm      : chr  "" "" "" "" ...
##  $ skewness_yaw_arm        : chr  "" "" "" "" ...
##  $ 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  : chr  "" "" "" "" ...
##  $ kurtosis_picth_dumbbell : chr  "" "" "" "" ...
##  $ kurtosis_yaw_dumbbell   : chr  "" "" "" "" ...
##  $ skewness_roll_dumbbell  : chr  "" "" "" "" ...
##  $ skewness_pitch_dumbbell : chr  "" "" "" "" ...
##  $ skewness_yaw_dumbbell   : chr  "" "" "" "" ...
##  $ 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        : chr  "" "" "" "" ...
##  $ 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        : chr  "" "" "" "" ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

2.3. Preprocessing data

2.3.1. Cleaning NA values A lot of columns appear to have a high number of NA's, first cleaning we will do is to remove columns with too much NA's, considering too much as the 50% of the values being NA.

```r
thresNAs <- 0.5 * nrow(rawdata)

removecols <- colSums(is.na(rawdata)) <= thresNAs

Newdata <- rawdata[ , removecols]
validation <- validation[ , removecols] # Remove same columns form validation
```

Now we have passed from 160 to 93 columns.

2.3.2. Selection of pertinent variables Now we will eliminate columns that have one unique value or having more than one, they are few and the ratio of the frequency of the most common value to the frequency of the second most common value is large. To do so, we will use the function nearZerovar() from the caret package.

Finally, we will eliminate all the columns regarding row number, individual and timestamp, because won't add information to our prediction, and will transform classe into a factor variable.

```r
ZeroVarCol <- nearZeroVar(Newdata)
Newdata <- Newdata[,-ZeroVarCol]
validation <- validation[,-ZeroVarCol]

Newdata <- Newdata[,-c(1:5)]
validation <- validation[,-c(1:5)]

Newdata$classe <- as.factor(Newdata$classe)
```

Now, our data.frames have been reduced to only 54 were our outcome, classe is the last column.

2.3.3. Training and Testing data creation Next step is to divide our data.frame into a training set and a testing set.

```r
set.seed(25012021)
inTrain <- createDataPartition(Newdata$classe, p = 0.7, list = FALSE)
training <- Newdata[inTrain, ]
testing <- Newdata[-inTrain, ]
```

And now we have three dataframes to work with:

```
1. training --> with **13737** rows and **54** cols to train the system
2. testing --> with **5885** rows and **54** cols to test the validity of every model
3. validation --> with **20** rows and 54 cols to validate our final model
```

3. Model building

3.1. Model Selection

As we have a classification problem in front of us, we will create and predict with following models: - Classification tree using rpart method on the caret package - Random Forest - Boosting with trees method

```r
# Due to the computational needs of Random Forest and Boosting, we need to control computational nuance

set.seed(25012021)

mod_rpart <- train(classe ~ ., method = "rpart", data = training)

contrf <- trainControl(method="cv", 5)

mod_rf <- train(classe ~ ., method = "rf", data = training, trControl = contrf, ntree = 200)

contrgbm <- trainControl(method = "repeatedcv", number = 5, repeats = 1)

mod_gbm <- train(classe ~ ., method = "gbm", data = training, trControl = contrgbm, verbose = FALSE)

# Now will use every model to predict testing data.frame and create their confusion Matrix to evaluate

pred_rpart <- predict(mod_rpart, newdata = testing)
cm_rpart <- confusionMatrix(pred_rpart, testing$classe)

pred_rf <- predict(mod_rf, newdata = testing)
cm_rf <- confusionMatrix(pred_rf, testing$classe)

pred_gbm <- predict(mod_gbm, newdata = testing)
cm_gbm <- confusionMatrix(pred_gbm, testing$classe)
```

And now, using the confusion Matrix, we can confirm that the rpart method has an an accuracy of **48.67%**, the random forest has an accuracy of **99.63%** and the boosting method achieves **99%** of accuracy.

Then, we will discard the rpart model because of its lack of accuracy, and will prefer the random forest over the boosting, even having boosting a very good accuracy too.

3.2. Applying the model validation data

Now we will apply our random forest and boosting models to the validation dataframe. Even if we will prefer the random forest model, will be a nice exercise to compare the result of both models.

```
valid_rf <- predict(mod_rf, validation)
valid_gbm <- predict(mod_gbm, validation)
```

And now we can note that between the classe predicted by both models for every line of the validation dataframe there are **0** differences, and the predicted values are:

```
print(valid_rf)
```

```
##  [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
```

A. APPENDIX I

A.I.1 Classification tree model

The rpart model is as follows:

```
print(mod_rpart)
```

```
## CART
##
## 13737 samples
##    53 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.03931441  0.5543783  0.42743990
##   0.06113315  0.3925686  0.17000661
##   0.11738379  0.3295036  0.07021148
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03931441.
```

And its confusion matrix is:

```
print(cm_rpart)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1505  485  485  416  174
##          B   34  375   25  189  151
##          C  130  279  516  359  289
##          D    0    0    0    0    0
##          E    5    0    0    0  468
##
## Overall Statistics
##
##                Accuracy : 0.4867
##                  95% CI : (0.4738, 0.4995)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.329
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8990  0.32924  0.50292   0.0000  0.43253
## Specificity            0.6295  0.91593  0.78247   1.0000  0.99896
## Pos Pred Value         0.4910  0.48450  0.32804      NaN  0.98943
## Neg Pred Value         0.9401  0.85052  0.88173   0.8362  0.88655
## Prevalence             0.2845  0.19354  0.17434   0.1638  0.18386
## Detection Rate         0.2557  0.06372  0.08768   0.0000  0.07952
## Detection Prevalence   0.5208  0.13152  0.26729   0.0000  0.08037
## Balanced Accuracy      0.7643  0.62258  0.64269   0.5000  0.71575
```

Where we can check the high amount of values outside the diagonal of the matrix, turning the model in not useful.

A.I.2 Random Forest

Printing the Random Forest model we can check its great accuracy and the small values outside the diagonal of the confusion matrix with the train data frame:

```r
print(mod_rf)
```

```
## Random Forest
##
## 13737 samples
##    53 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10988, 10989, 10990, 10990, 10991
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9937393  0.9920804
##   27    0.9964332  0.9954881
##   53    0.9946861  0.9932779
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```r
print(mod_rf$finalModel)
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 200, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 200
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.17%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3905    1    0    0    0 0.0002560164
## B    6 2648    2    2    0 0.0037622272
## C    0    4 2392    0    0 0.0016694491
## D    0    0    8 2243    1 0.0039964476
## E    0    0    0    0 2525 0.0000000000
```

And its confusion matrix with the testing data frame is:

```r
print(cm_rf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    7    0    0    0
##          B    0 1130    2    0    0
##          C    0    1 1024    7    0
##          D    0    1    0  957    3
##          E    1    0    0    0 1079
##
## Overall Statistics
##
##                Accuracy : 0.9963
##                  95% CI : (0.9943, 0.9977)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9953
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9994   0.9921   0.9981   0.9927   0.9972
## Specificity            0.9983   0.9996   0.9984   0.9992   0.9998
## Pos Pred Value         0.9958   0.9982   0.9922   0.9958   0.9991
## Neg Pred Value         0.9998   0.9981   0.9996   0.9986   0.9994
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2843   0.1920   0.1740   0.1626   0.1833
## Detection Prevalence   0.2855   0.1924   0.1754   0.1633   0.1835
## Balanced Accuracy      0.9989   0.9958   0.9982   0.9960   0.9985
```

Where we can check the small values outside the diagonal of the matrix, giving a great performance to the model.

A.I.3 Boosting

Printing the general boosting model we can check its more than decent accuracy:

```
print(mod_gbm$finalModel)
```

```
## A gradient boosted model with multinomial loss function.
## 150 iterations were performed.
## There were 53 predictors of which 53 had non-zero influence.
```

```
print(mod_gbm$results)
```

```
##   shrinkage interaction.depth n.minobsinnode n.trees  Accuracy     Kappa
## 1       0.1                 1             10      50 0.7614458 0.6974615
## 4       0.1                 2             10      50 0.8825060 0.8512447
## 7       0.1                 3             10      50 0.9333186 0.9155957
## 2       0.1                 1             10     100 0.8317678 0.7871183
## 5       0.1                 2             10     100 0.9385603 0.9222479
## 8       0.1                 3             10     100 0.9728462 0.9656457
## 3       0.1                 1             10     150 0.8694760 0.8348234
## 6       0.1                 2             10     150 0.9624376 0.9524736
## 9       0.1                 3             10     150 0.9863865 0.9827782
##    AccuracySD     KappaSD
## 1 0.010272118 0.012754869
## 4 0.011608467 0.014613756
## 7 0.004115460 0.005211283
## 2 0.007917880 0.009822639
## 5 0.005009361 0.006336185
## 8 0.003653007 0.004620400
## 3 0.009035916 0.011327841
## 6 0.002793686 0.003526414
## 9 0.003423004 0.004332585
```

And its confusion matrix with the testing data frame is:

```
print(cm_gbm)
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    A    B    C    D    E
##          A 1671    9    0    0    1
##          B    2 1122    7    2    4
##          C    0    7 1015    9    2
##          D    1    0    4  950    7
##          E    0    1    0    3 1068
##
## Overall Statistics
##
##                Accuracy : 0.99
##                  95% CI : (0.9871, 0.9924)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9873
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9982   0.9851   0.9893   0.9855   0.9871
## Specificity            0.9976   0.9968   0.9963   0.9976   0.9992
## Pos Pred Value         0.9941   0.9868   0.9826   0.9875   0.9963
## Neg Pred Value         0.9993   0.9964   0.9977   0.9972   0.9971
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2839   0.1907   0.1725   0.1614   0.1815
## Detection Prevalence   0.2856   0.1932   0.1755   0.1635   0.1822
## Balanced Accuracy      0.9979   0.9910   0.9928   0.9915   0.9931
```

Where we can check the small values outside the diagonal of the matrix, giving a great performance to the model.