Weight Lifting Exercises Classification

by Renato Pedroso Neto

Synopsis

This study aims to model the Weight Lifting Exercises Dataset (http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises), provided by PUC university, using machine learning tools to predict the manner in which a person is doing the Unilateral Dumbbell Biceps Curl exercise.
The study will use the following class labels:
a) Exercise exactly according to the specification (Class A)
b) Exercise throwing the elbows to the front (Class B)
c) Exercise lifting the dumbbell only halfway (Class C)
d) Exercise lowering the dumbbell only halfway (Class D)
e) Exercise throwing the hips to the front (Class E)

Data Processing

1. To begin with, we need to load the data, available here (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)

```
library(caret, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
library(plyr, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
library(dplyr, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
library(data.table, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
library(randomForest, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
```

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(gbm, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loaded gbm 2.1.1
```

```
library(survival, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
library(splines, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
library(parallel, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
library(rpart, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
library(klaR, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
library(MASS, quietly = TRUE, warn.conflicts = FALSE, verbose = FALSE)
setwd("C:\\Users\\Renato\\WLE_Data")
# load the training data
wle_training <- fread("pml-training.csv", sep = ",", header = TRUE, stringsAsFactors = FALSE,
 na.strings=c("", "NA"))
# load the test data
wle_test     <- fread("pml-testing.csv", sep = ",", header = TRUE, stringsAsFactors = FALSE,
na.strings=c("", "NA"))
```

2. After that we can check the variables and start to think about include them, or not, in one machine learning technique.

```
# is there any column with all values na?
na_count <- as.data.frame(colSums(is.na(wle_training)))
na_count[na_count == nrow(wle_training),]
```

```
## numeric(0)
```

```
# no columns with only NA values!

# drop all variables that has NA (training and test)
drop <- colSums(is.na(wle_training)) == 0
drop <- drop[drop == TRUE]
wle_training <- subset(wle_training, select = names(drop))
wle_test$classe <- ""
wle_test <- subset(wle_test, select = names(drop))
remove(drop)

# remove first 6 columns
wle_training <- subset(wle_training, select = -c(1:6))
wle_test     <- subset(wle_test, select = -c(1:6))

# transform classe in factor
wle_training$classe = as.factor(wle_training$classe)
```

3. Subdividing the training data set loaded from PUC website

```
wle_inf <- createDataPartition(wle_training$classe, p = 0.6, list = FALSE)
wle_training_train <- wle_training[wle_inf,]
wle_training_test  <- wle_training[-wle_inf,]
```

Model Planning / Model Building

1. For classifications pourposes we will try the most effective algorithms:

a. Random Forests

b. Boosting
c. Decision Tree
d. Naive Bayes

All of them using the cross validation with 5 folds.

```
set.seed(9191)
model_rf  <- train(classe ~ . , data = wle_training_train, method = "rf",
                  trControl = trainControl(method = "cv", number = 5))

model_gbm <- train(classe ~ . , data = wle_training_train, method = "gbm",
                  trControl = trainControl(method = "cv", number = 5),
                  verbose = FALSE)

model_dt  <- train(classe ~ . , data = wle_training_train, method = "rpart",
                  trControl = trainControl(method = "cv", number = 5))

model_nb  <- train(classe ~ . , data = wle_training_train, method = "nb",
                  trControl = trainControl(method = "cv", number = 5),
                  verbose = FALSE)

# in sample accuracy
acc_rf_is  <- confusionMatrix(predict(model_rf, wle_training_train), wle_training_train$class
e)$overall[1]
acc_gbm_is <- confusionMatrix(predict(model_gbm, wle_training_train), wle_training_train$clas
se)$overall[1]
acc_dt_is  <- confusionMatrix(predict(model_dt, wle_training_train), wle_training_train$class
e)$overall[1]
acc_nb_is  <- confusionMatrix(predict(model_nb, wle_training_train), wle_training_train$class
e)$overall[1]
insample_acc <- data.frame(acc_rf_is, acc_gbm_is, acc_dt_is, acc_nb_is)

# out of sample accuracy
acc_rf_os  <- confusionMatrix(predict(model_rf, wle_training_test),
wle_training_test$classe)$overall[1]
acc_gbm_os <- confusionMatrix(predict(model_gbm, wle_training_test),
wle_training_test$classe)$overall[1]
acc_dt_os  <- confusionMatrix(predict(model_dt, wle_training_test),
wle_training_test$classe)$overall[1]
acc_nb_os  <- confusionMatrix(predict(model_nb, wle_training_test),
wle_training_test$classe)$overall[1]
outsample_acc <- data.frame(acc_rf_os, acc_gbm_os, acc_dt_os, acc_nb_os)

# In Sample and Out of Sample Accuracy
print(insample_acc)
```

```
##          acc_rf_is acc_gbm_is acc_dt_is acc_nb_is
## Accuracy         1  0.9939708 0.4972826 0.7680027
```

```
print(outsample_acc)
```

```
##          acc_rf_os acc_gbm_os acc_dt_os acc_nb_os
## Accuracy 0.9978333   0.986235  0.491843 0.7628091
```

We can compare all the confusion matrix generated (out of sample only):

```r
# Random Forest Trees Confusion Matrix
confusionMatrix(predict(model_rf, wle_training_test), wle_training_test$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    2    0    0    0
##          B    0 1515    4    0    0
##          C    0    1 1364    7    0
##          D    0    0    0 1279    3
##          E    0    0    0    0 1439
##
## Overall Statistics
##
##                Accuracy : 0.9978
##                  95% CI : (0.9965, 0.9987)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9973
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9980   0.9971   0.9946   0.9979
## Specificity            0.9996   0.9994   0.9988   0.9995   1.0000
## Pos Pred Value         0.9991   0.9974   0.9942   0.9977   1.0000
## Neg Pred Value         1.0000   0.9995   0.9994   0.9989   0.9995
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1931   0.1738   0.1630   0.1834
## Detection Prevalence   0.2847   0.1936   0.1749   0.1634   0.1834
## Balanced Accuracy      0.9998   0.9987   0.9979   0.9970   0.9990
```

```r
# Boosting Confusion Matrix
confusionMatrix(predict(model_gbm, wle_training_test), wle_training_test$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2230   21    0    0    1
##          B    2 1475   12    5    2
##          C    0   18 1354   20    6
##          D    0    4    1 1260   14
##          E    0    0    1    1 1419
## 
## Overall Statistics
## 
##                Accuracy : 0.9862
##                  95% CI : (0.9834, 0.9887)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.9826
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9991   0.9717   0.9898   0.9798   0.9840
## Specificity            0.9961   0.9967   0.9932   0.9971   0.9997
## Pos Pred Value         0.9902   0.9860   0.9685   0.9851   0.9986
## Neg Pred Value         0.9996   0.9932   0.9978   0.9960   0.9964
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1880   0.1726   0.1606   0.1809
## Detection Prevalence   0.2870   0.1907   0.1782   0.1630   0.1811
## Balanced Accuracy      0.9976   0.9842   0.9915   0.9884   0.9919
```

```
# Decision Tree Confusion Matrix
confusionMatrix(predict(model_dt, wle_training_test), wle_training_test$classe)
```

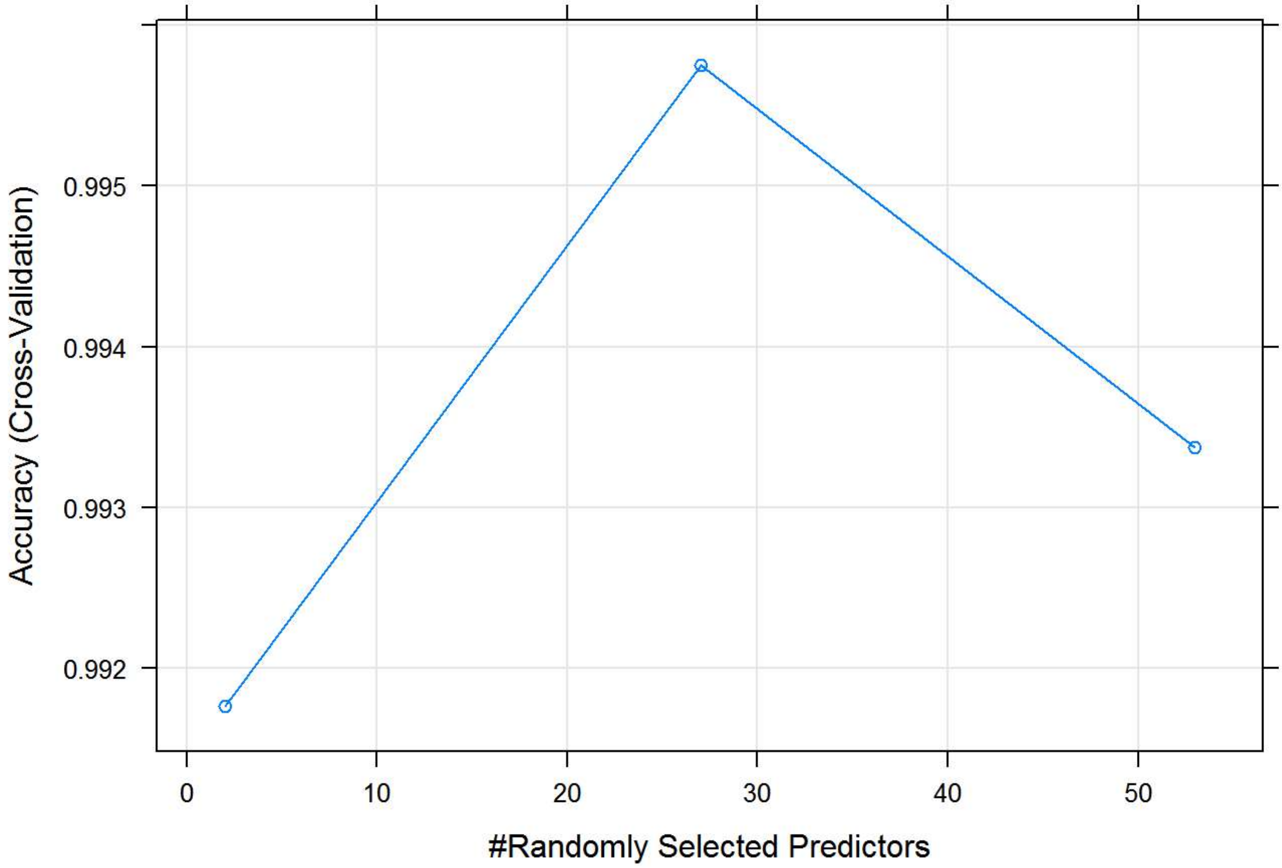
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2037  630  651  562  211
##          B   32  503   46  240  191
##          C  156  385  671  484  392
##          D    0    0    0    0    0
##          E    7    0    0    0  648
##
## Overall Statistics
##
##                Accuracy : 0.4918
##                  95% CI : (0.4807, 0.503)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3357
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9126  0.33136  0.49050   0.0000  0.44938
## Specificity            0.6341  0.91956  0.78126   1.0000  0.99891
## Pos Pred Value         0.4979  0.49704  0.32136      NaN  0.98931
## Neg Pred Value         0.9481  0.85148  0.87895   0.8361  0.88958
## Prevalence             0.2845  0.19347  0.17436   0.1639  0.18379
## Detection Rate         0.2596  0.06411  0.08552   0.0000  0.08259
## Detection Prevalence   0.5214  0.12898  0.26612   0.0000  0.08348
## Balanced Accuracy      0.7734  0.62546  0.63588   0.5000  0.72414
```

```
# Naive Bayes Confusion Matrix
confusionMatrix(predict(model_nb, wle_training_test), wle_training_test$classe)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2000  300  250  191   85
##          B   51 1019  104    7  126
##          C   56  135  986  173   58
##          D  118   57   28  863   56
##          E    7    7    0   52 1117
## 
## Overall Statistics
## 
##                Accuracy : 0.7628          
##                  95% CI : (0.7532, 0.7722)
##     No Information Rate : 0.2845          
##     P-Value [Acc > NIR] : < 2.2e-16       
## 
##                   Kappa : 0.697           
##  Mcnemar's Test P-Value : < 2.2e-16       
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8961   0.6713   0.7208   0.6711   0.7746
## Specificity            0.8529   0.9545   0.9349   0.9605   0.9897
## Pos Pred Value         0.7077   0.7796   0.7003   0.7692   0.9442
## Neg Pred Value         0.9538   0.9237   0.9407   0.9371   0.9512
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2549   0.1299   0.1257   0.1100   0.1424
## Detection Prevalence   0.3602   0.1666   0.1795   0.1430   0.1508
## Balanced Accuracy      0.8745   0.8129   0.8278   0.8158   0.8822
```

Conclusions and Prediction

```
plot(model_rf)
```



The model that offered the best accuracy was the ***random forest***. It reached ***99,8%*** of accuracy in the out of sample test.
If we consider this accuracy, the expected out of sample error is ***0,2%***

The prediction of the test data is:

```
wle_test$classe <- predict(model_rf, wle_test)
print(wle_test$classe)
```

```
##  [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
```

```
str(wle_test)
```

```
## Classes 'data.table' and 'data.frame':   20 obs. of  54 variables:
##  $ num_window          : int  74 431 439 194 235 504 485 440 323 664 ...
##  $ roll_belt           : num  123 1.02 0.87 125 1.35 -5.92 1.2 0.43 0.93 114 ...
##  $ pitch_belt          : num  27 4.87 1.82 -41.6 3.33 1.59 4.44 4.15 6.72 22.4 ...
##  $ yaw_belt            : num  -4.75 -88.9 -88.5 162 -88.6 -87.7 -87.3 -88.5 -93.7 -13.1
 ...
##  $ total_accel_belt    : int  20 4 5 17 3 4 4 4 4 18 ...
##  $ gyros_belt_x        : num  -0.5 -0.06 0.05 0.11 0.03 0.1 -0.06 -0.18 0.1 0.14 ...
##  $ gyros_belt_y        : num  -0.02 -0.02 0.02 0.11 0.02 0.05 0 -0.02 0 0.11 ...
##  $ gyros_belt_z        : num  -0.46 -0.07 0.03 -0.16 0 -0.13 0 -0.03 -0.02 -0.16 ...
##  $ accel_belt_x        : int  -38 -13 1 46 -8 -11 -14 -10 -15 -25 ...
##  $ accel_belt_y        : int  69 11 -1 45 4 -16 2 -2 1 63 ...
##  $ accel_belt_z        : int  -179 39 49 -156 27 38 35 42 32 -158 ...
##  $ magnet_belt_x       : int  -13 43 29 169 33 31 50 39 -6 10 ...
##  $ magnet_belt_y       : int  581 636 631 608 566 638 622 635 600 601 ...
##  $ magnet_belt_z       : int  -382 -309 -312 -304 -418 -291 -315 -305 -302 -330 ...
##  $ roll_arm            : num  40.7 0 0 -109 76.1 0 0 0 -137 -82.4 ...
##  $ pitch_arm           : num  -27.8 0 0 55 2.76 0 0 0 11.2 -63.8 ...
##  $ yaw_arm             : num  178 0 0 -142 102 0 0 0 -167 -75.3 ...
##  $ total_accel_arm     : int  10 38 44 25 29 14 15 22 34 32 ...
##  $ gyros_arm_x         : num  -1.65 -1.17 2.1 0.22 -1.96 0.02 2.36 -3.71 0.03 0.26 ...
##  $ gyros_arm_y         : num  0.48 0.85 -1.36 -0.51 0.79 0.05 -1.01 1.85 -0.02 -0.5 ...
##  $ gyros_arm_z         : num  -0.18 -0.43 1.13 0.92 -0.54 -0.07 0.89 -0.69 -0.02 0.79 ...
##  $ accel_arm_x         : int  16 -290 -341 -238 -197 -26 99 -98 -287 -301 ...
##  $ accel_arm_y         : int  38 215 245 -57 200 130 79 175 111 -42 ...
##  $ accel_arm_z         : int  93 -90 -87 6 -30 -19 -67 -78 -122 -80 ...
##  $ magnet_arm_x        : int  -326 -325 -264 -173 -170 396 702 535 -367 -420 ...
##  $ magnet_arm_y        : int  385 447 474 257 275 176 15 215 335 294 ...
##  $ magnet_arm_z        : int  481 434 413 633 617 516 217 385 520 493 ...
##  $ roll_dumbbell       : num  -17.7 54.5 57.1 43.1 -101.4 ...
##  $ pitch_dumbbell      : num  25 -53.7 -51.4 -30 -53.4 ...
##  $ yaw_dumbbell        : num  126.2 -75.5 -75.2 -103.3 -14.2 ...
##  $ total_accel_dumbbell: int  9 31 29 18 4 29 29 29 3 2 ...
##  $ gyros_dumbbell_x    : num  0.64 0.34 0.39 0.1 0.29 -0.59 0.34 0.37 0.03 0.42 ...
##  $ gyros_dumbbell_y    : num  0.06 0.05 0.14 -0.02 -0.47 0.8 0.16 0.14 -0.21 0.51 ...
##  $ gyros_dumbbell_z    : num  -0.61 -0.71 -0.34 0.05 -0.46 1.1 -0.23 -0.39 -0.21 -0.03 ...
##  $ accel_dumbbell_x    : int  21 -153 -141 -51 -18 -138 -145 -140 0 -7 ...
##  $ accel_dumbbell_y    : int  -15 155 155 72 -30 166 150 159 25 -20 ...
##  $ accel_dumbbell_z    : int  81 -205 -196 -148 -5 -186 -190 -191 9 7 ...
##  $ magnet_dumbbell_x   : int  523 -502 -506 -576 -424 -543 -484 -515 -519 -531 ...
##  $ magnet_dumbbell_y   : int  -528 388 349 238 252 262 354 350 348 321 ...
##  $ magnet_dumbbell_z   : int  -56 -36 41 53 312 96 97 53 -32 -164 ...
##  $ roll_forearm        : num  141 109 131 0 -176 150 155 -161 15.5 13.2 ...
##  $ pitch_forearm       : num  49.3 -17.6 -32.6 0 -2.16 1.46 34.5 43.6 -63.5 19.4 ...
##  $ yaw_forearm         : num  156 106 93 0 -47.9 89.7 152 -89.5 -139 -105 ...
##  $ total_accel_forearm : int  33 39 34 43 24 43 32 47 36 24 ...
##  $ gyros_forearm_x     : num  0.74 1.12 0.18 1.38 -0.75 -0.88 -0.53 0.63 0.03 0.02 ...
##  $ gyros_forearm_y     : num  -3.34 -2.78 -0.79 0.69 3.1 4.26 1.8 -0.74 0.02 0.13 ...
##  $ gyros_forearm_z     : num  -0.59 -0.18 0.28 1.8 0.8 1.35 0.75 0.49 -0.02 -0.07 ...
##  $ accel_forearm_x     : int  -110 212 154 -92 131 230 -192 -151 195 -212 ...
##  $ accel_forearm_y     : int  267 297 271 406 -93 322 170 -331 204 98 ...
##  $ accel_forearm_z     : int  -149 -118 -129 -39 172 -144 -175 -282 -217 -7 ...
##  $ magnet_forearm_x    : int  -714 -237 -51 -233 375 -300 -678 -109 0 -403 ...
##  $ magnet_forearm_y    : int  419 791 698 783 -787 800 284 -619 652 723 ...
##  $ magnet_forearm_z    : int  617 873 783 521 91 884 585 -32 469 512 ...
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 2 1 2 1 1 5 4 2 1 1 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```