

# Human Activity Recognition

Davide

Saturday, August 23, 2014

## Premise

Have look at: <https://www.youtube.com/watch?v=fjMXDINkYvU>

## Introduction

This project has been written for the fulfillment of the requirements of the *Johns Hopkins Bloomberg School of Public Health's Practical Machine Learning* course.

This project will build a prediction system for recognition of human activity based on selected measurements from a number of wearable sensors. The dataset at hand was collected to try to recognize not the specific activity that was performed. As a matter of fact all observations refer to the same movement: a set of ten repetitions of the "Unilateral Dumbbell Biceps Curl" exercise (see: [https://www.youtube.com/watch?v=YxtwA7XRK\\_g](https://www.youtube.com/watch?v=YxtwA7XRK_g) for an example).

The specificity of the data is that the same exercise is performed in 6 different ways (*classes*):

- A: correctly, i.e. exactly according to the specifications;
- B: throwing the elbows to the front;
- C: lifting the dumbbell only halfway;
- D: lowering the dumbbell only halfway;
- E: throwing the hips to the front.

Only A is the correct way to perform the exercise. Purpose of the recognition system is to process the data to tell if the trainee is performing the exercise correctly or is doing some mistakes. Once the appropriate classes A, B, ..., E is discriminated, the system could provide appropriate feedback. The data contains features that come straight from sensors measurements, and *derived* features as well, that are computed from others: total, kurtosis, skewness, means, ... More information on the data gathering and pre-processing operations can be found in parts 3 and 4 of [2].

## Exploratory Data Analysis

```
rm(list = ls())
setwd('C:/Nuova cartella/Practical Machine Learning/Assignment')

fileUrl_train = 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-
training.csv'
destfile = 'pml-training.csv'
if (!file.exists(destfile)) {
  download.file(fileUrl_train, destfile = destfile)
```

```

}

fileUrl_test = 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-
testing.csv'
destfile = 'pml-testing.csv'
if (!file.exists(destfile)) {
  download.file(fileUrl_test, destfile = destfile)
}

data_train_file = read.table('pml-training.csv', header = TRUE, sep = ',',
na.string = 'NA')
str(data_train_file)

## '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 1323084232 1323084232 1323084232 1323084232
...
## $ raw_timestamp_part_2 : int  788290 808298 820366 120339 196328 304277
368296 440390 484323 484434 ...
## $ 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_pitch_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_pitch_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 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 NA ...
## $ avg_roll_belt        : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ stddev_roll_belt     : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ var_roll_belt        : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ avg_pitch_belt       : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ stddev_pitch_belt    : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ var_pitch_belt       : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ avg_yaw_belt         : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ stddev_yaw_belt      : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ var_yaw_belt         : num NA 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 0.02
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 -0.02 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 NA ...
## $ avg_roll_arm         : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ stddev_roll_arm      : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ var_roll_arm         : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ avg_pitch_arm        : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ stddev_pitch_arm     : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ var_pitch_arm        : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ avg_yaw_arm          : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ stddev_yaw_arm       : num NA NA NA NA NA NA NA NA NA NA NA ...
## $ var_yaw_arm          : num NA 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 -

```



The training data consists of 19622 observations of 160 variables. The distribution of classes in the data is as follows:

```
class_dist_train <- table(data_train_file$classe)
class_dist_train

##
##      A      B      C      D      E
## 5580 3797 3422 3216 3607

barplot(class_dist_train, ylab = 'Num. of Samples', main = 'Distribution of
Classes\nTraining Data')
```



Looking at the data we notice as there are several cells in the table that contain the value: #DIV/0!: 3502. This for sure comes from computed features in cases where the specific value could not be computed. These values are, from all respects, just like NAs for the following analysis. So no information is lost when I set the to NA:

```
data_train_file[data_train_file == '#DIV/0!'] = NA
sum(data_train_file == '#DIV/0!', na.rm = TRUE) # Consistency check: should be
zero

## [1] 0
```

It looks to me clear that timestamp-like features, num. of observation, or user name should have *nothing* to do with predictions, so I remove them from the dataset:

```
drop <- c('X', 'user_name', 'raw_timestamp_part_1', 'raw_timestamp_part_2',
'cvtd_timestamp', 'new_window', 'num_window')
data_train_file <- data_train_file[, !(names(data_train_file) %in% drop)]
```

Now let's rebuild the data frame taking the original dataframe and converting all values to numeric (apart from the target variable, stored in the 153-rd column):

```
data_train_file = data.frame(lapply(data_train_file[, -153], function(x)
as.numeric(x)), classe = data_train_file[, 153])
```

... and remove any column that has NAs "values":

```
index = sapply(data_train_file, function(x) !any(is.na(x)))
names(index) = NULL
data_train_file = data_train_file[, index]
```

It should be noted that the previous operation removed a lot columns: now we have 53 columns only. Unfortunately, this was a *needed* operation, since the caret's package train function does not deal well with NAs. Nonetheless, apart from the NAs, the removed columns could contain relevant information. In the *Further study* section I will explore some methods to retain this information (essentially using an ensemble predictor).

```
require('caret')

## Loading required package: caret
## Warning: package 'caret' was built under R version 3.0.3
## Loading required package: lattice
## Warning: package 'lattice' was built under R version 3.0.3
## Loading required package: ggplot2
## Warning: package 'ggplot2' was built under R version 3.0.3

library(caret)
```

## Building the Predictor

Since we are starting to perform operations that depend on the outcome or (pseudo)random variables, let's set the seed in order to ensure reproducibility:

```
set.seed(1234)
```

Now let's divide the trainint dataset into two parts, an actual *training* set (60% of data) and a *cross validation* set (40%). The latter will be used to asses (estimate) the out-of-sample error using the prediction error on the corss validation dataset.

```
training_obs_index = createDataPartition(y = data_train_file$classe, p = 0.6,
list = FALSE)
data_train = data_train_file[training_obs_index,]
data_cross = data_train_file[-training_obs_index,]
```

We have 11776 observations in the training set and 7846 observations in the cross validation set.

Let's build the prediction model (beware: this takes a **long** time; this is why I have computed it *only once and then saved it on a file*. This way I can work on the markdown without computing it every time I compile the markdown, simply reading it from disk):

```
#model.rf = train(classe ~ ., data = data_train, method = 'rf')
#save(model.rf, file = 'model.rf') # Save it in order not to waste computation
time!
```

```
load(file = 'model.rf')
model.rf

## Random Forest
##
## 11776 samples
##    52 predictors
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 11776, 11776, 11776, 11776, 11776, 11776, ...
##
## Resampling results across tuning parameters:
##
##  mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2      1        1    0.002      0.003
##   30      1        1    0.002      0.003
##   50      1        1    0.005      0.006
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

## Performance Assessment

In this section we're going to assess the performance of the fitted model.

### In-sample

Here's the in-sample confusion matrix of the fitted model. Looks like it works pretty well on the training set, keeping the in-sample classification error lower than 1.8%:

```
model.rf$finalModel$confusion

##      A      B      C      D      E class.error
## A 3345      2      0      0      1  0.0008961
## B   21 2252      6      0      0  0.0118473
## C    1   21 2028      4      0  0.0126582
## D    0    0   34 1894      2  0.0186528
## E    0    0    3    8 2154  0.0050808
```

You can also check this out:

```
predict_train <- predict(model.rf)

## Loading required package: randomForest

## Warning: package 'randomForest' was built under R version 3.0.3

## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.

confusionMatrix(predict_train, data_train$classe)

## Warning: package 'e1071' was built under R version 3.0.3
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##           A 3348    0    0    0    0
##           B    0 2279    0    0    0
##           C    0    0 2054    0    0
##           D    0    0    0 1930    0
##           E    0    0    0    0 2165
##
```

```
## Overall Statistics
##
##           Accuracy : 1
##           95% CI : (1, 1)
##           No Information Rate : 0.284
##           P-Value [Acc > NIR] : <2e-16
##
##           Kappa : 1
##           McNemar's Test P-Value : NA
##
```

```
## Statistics by Class:
```

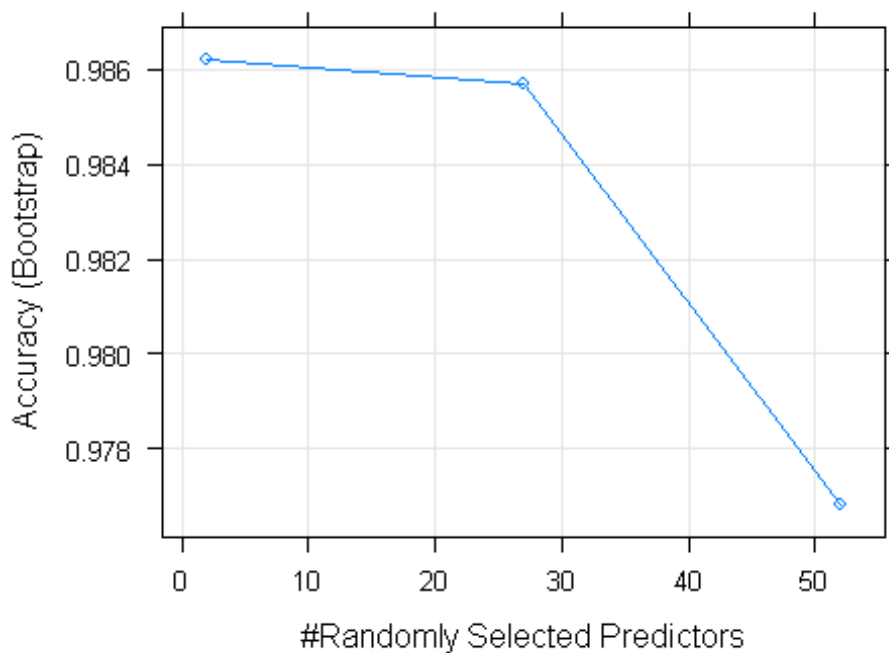
```
##
##           Class: A Class: B Class: C Class: D Class: E
## Sensitivity      1.000    1.000    1.000    1.000    1.000
## Specificity      1.000    1.000    1.000    1.000    1.000
## Pos Pred Value   1.000    1.000    1.000    1.000    1.000
## Neg Pred Value   1.000    1.000    1.000    1.000    1.000
## Prevalence       0.284    0.194    0.174    0.164    0.184
## Detection Rate   0.284    0.194    0.174    0.164    0.184
## Detection Prevalence 0.284    0.194    0.174    0.164    0.184
## Balanced Accuracy 1.000    1.000    1.000    1.000    1.000
```

Another assessment of the accuracy of the predictor vs the number of variables can be computed using the *bootstrap* technique:

```
plot(model.rf, main = 'Accuracy of the Model')
```



## Accuracy of the Model



## Cross-validation

Let's check the model on the cross validation set:

```
predict_cv <- predict(model.rf, newdata = data_cross)
table(predict_cv, data_cross$classe)
```

```
##
## predict_cv      A      B      C      D      E
##      A 2230     10      0      0      0
##      B   2 1505     11      0      0
##      C   0   3 1354     28      3
##      D   0   0   3 1256      3
##      E   0   0   0   2 1436
```

We have an error of 0.8284%, so less than 1% (accuracy > 99%). This can be used as an estimate of the *out-of-sample* error. This *by-hand* computation is confirmed:

```
confusionMatrix(predict_cv, data_cross$classe)
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction      A      B      C      D      E
##      A 2230     10      0      0      0
##      B   2 1505     11      0      0
##      C   0   3 1354     28      3
##      D   0   0   3 1256      3
##      E   0   0   0   2 1436
##
## Overall Statistics
##
##          Accuracy : 0.992
```

```
##          95% CI : (0.989, 0.994)
##    No Information Rate : 0.284
##    P-Value [Acc > NIR] : <2e-16
##
##          Kappa : 0.99
##  McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##          Class: A Class: B Class: C Class: D Class: E
## Sensitivity      0.999   0.991   0.990   0.977   0.996
## Specificity      0.998   0.998   0.995   0.999   1.000
## Pos Pred Value   0.996   0.991   0.976   0.995   0.999
## Neg Pred Value   1.000   0.998   0.998   0.995   0.999
## Prevalence       0.284   0.193   0.174   0.164   0.184
## Detection Rate   0.284   0.192   0.173   0.160   0.183
## Detection Prevalence 0.285   0.193   0.177   0.161   0.183
## Balanced Accuracy 0.999   0.995   0.992   0.988   0.998
```

## Out-of-sample

To work on the test set, we need to perform on it all the processing and clean-up we did on the training data set:

```
rm(data_test_file)

## Warning: object 'data_test_file' not found

data_test_file = read.table('pml-testing.csv', header = TRUE, sep = ',',
na.string = 'NA')
data_test_file[data_test_file == '#DIV/0!'] = NA
drop <- c('X', 'user_name', 'raw_timestamp_part_1', 'raw_timestamp_part_2',
'cvtd_timestamp', 'new_window', 'num_window')
data_test_file <- data_test_file[, !(names(data_test_file) %in% drop)]
data_test_file = data.frame(lapply(data_test_file, function(x) as.numeric(x)),
classe = data_test_file[,153])
index = sapply(data_test_file, function(x) !any(is.na(x)))
names(index) = NULL
data_test_file = data_test_file[,index]
```

Now we're ready to apply the predictor on the data:

```
predict(model.rf, newdata = data_test_file)

## [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
```

Turns out that if you submit these results in the "Prediction Assignment Submission" page of the course, these are the *right* values. This allows us to say that this predictor has **100%** estimated out-of-sample accuracy.

## Further study

It is interesting to note that some of the features come from processing of other features and values of some of the former make sense only once per samplig period. This is clearly noted in

the dataset looking at the `new_window` and `num_window` columns . See, for example, values of `kurtosis_roll_bell` feature. Unfortunately, the procedure I used excluded these variables, just due to the fact that most of their cells are empty. But we cannot say for sure that they do not contain valuable information. On the other hand these had to be thrown, otherwise the `train` function would have *complained*.

How to deal with this fact? One possible strategy would be to pull these rows and columns out of the dataset to form another dataset. This would have all cells filled with precious data. Then model a predictor based on the former, cleaned dataset and another based on the latter dataset. Lastly, combine these two predictors into one to form an *ensemble* predictor. This is left as a further study.

Moreover, maybe we can lower the number of features we use to train the predictor. This can be done via a PCA analysis to see if there are some variables that can be discarded because they give little contribution to the overall variance of the data. This is left as a further study as well.

## Conclusions

In this study we modeled a predictor for the class of activity a trainee is performing. Data come from a set of sensors she wears during the exercise. The task at hand is to tell if an exercise is well performed, according to some standard, from one that is performed with mistakes. After a pre-processing phase to clean up and prepare the data, we trained a random forest predictor that shows very good performance. Despite the good results we got, some space for improvements will likely lie out there, since we discarded some features. A guideline for further analysis is therefore drawn.

## 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, ([http://groupware.les.inf.puc-rio.br/har#wle\\_paper\\_section#ixzz3BDpfkmR8](http://groupware.les.inf.puc-rio.br/har#wle_paper_section#ixzz3BDpfkmR8))
- [2] **Wallace Ugulino<sup>1</sup>, Débora Cardador<sup>1</sup>, Katia Vega<sup>1</sup>, Eduardo Velloso<sup>2</sup>, Ruy Milidiú<sup>1</sup>, and Hugo Fuks<sup>1</sup>** *Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements* SBIA 2012, LNAI 7589, pp. 52-61, 2012.