Predicting a physical activity type from accelerometer measurements.

By Joanne Breitfelder

Introduction

Using devices such as *Jawbone Up*, *Nike FuelBand* and *Fitbit*, it is now possible to collect a large amount of data about personal activity, and relatively inexpensively. These 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 just 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.

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). Class A corresponds to the specified execution of the exercise, while the other 4 classes correspond to common mistakes.

In this project, we will use the measurements given by accelerometers on the belt, forearm, arm, and dumbell of the participants to predict the class of the activity they were doing.

More information is available [here](http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Reference :

Qualitative Activity Recognition of Weight Lifting Exercises

Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W. and Fuks, H. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) Stuttgart, Germany: ACM SIGCHI, 2013.

Read more: http://groupware.les.inf.puc-rio.br/har#ixzz405DthvvN

Pre-processing

1. Loading packages and setting the seed for reproducibility :

```
library(dplyr); library(ggplot2); library(knitr); library(caret); library(tidyr)
set.seed(123)
```

---

2. Loading and creating the training, testing and validating datasets

We will create a validating set by partitioning the training set. In particular, this set will allow us to calculate the out-of-sample error rate.

```
if (!file.exists("training_data") | !file.exists("testing_data")) {
        url_train <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
        url_test <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
        download.file(url_train, destfile="training_data", method="curl")
        download.file(url_test, destfile="testing_data", method="curl")}

train_big <- read.csv("training_data", na.strings=c("NA", "", "#DIV/0!"))
test <- read.csv("testing_data", na.strings=c("NA", "", "#DIV/0!"))

########## CREATING A VALIDATION SET ##########
train <- train_big[createDataPartition(y=train_big$classe, p=0.6, list=FALSE), ]
validation <- train_big[-createDataPartition(y=train_big$classe, p=0.6, list=FALSE), ]
```

Dimensions of the resulting tables :

```
##              train validation test
## observations 11776       7846   20
## variables      160        160  160
```

---

3. Removing irrelevant variables

The data processing is done in the exact same way on the three datasets.

```
nzv <- nearZeroVar(train)
train <- train[, -nzv]
test <- test[, -nzv]
validation <- validation[, -nzv]
```

Removing the near-zero variables :

Removing the factor variables : The dataset is composed of 3 factor variables. These variables are not well handled by machine learning, and dummy variables can be tricky to use too. *classe* is our outcome, so we won't consider it for the moment. *cvtd_timestamp* and *user_name* are not correlated with our outcome, so we will simply remove them.

```
train <- select(train, -c(cvtd_timestamp, user_name))
test <- select(test, -c(cvtd_timestamp, user_name))
validation <- select(validation, -c(cvtd_timestamp, user_name))
```

Removing other irrelevant features : *X*, *raw_timestamp_part_1*, *raw_timestamp_part_2* and *num_window* are not relevant, because not physically correlated with the outcome. In fact, the variable *X* is unphysically but highly correlated to the outcome, what could even introduce a strong biais in the results.

```
train <- select(train, -c(X, raw_timestamp_part_1:num_window))
test <- select(test, -c(X, raw_timestamp_part_1:num_window))
validation <- select(validation, -c(X, raw_timestamp_part_1:num_window))
```

Removing the features with mostly missing data : An exploratory analysis shows that the 160 variables are divided into two groups regarding missing values :

- 60 variables have 0 missing values
- 100 variables have more than 97% of NAs!

Removing these variables does not reduce significantly the accuracy of the algorithm. In this case, it seems to be a better option than imputing missing values, because it reduces the number of predictors.

```
no_NAs <- sapply(train, function(x) sum(!is.na(x))) > 11775
train <- train[no_NAs]
test <- test[no_NAs]
validation <- validation[no_NAs]
```

These simple steps allowed us to divide by 3 the number of predictors. We could go further, for example with a Principal Component Analysis, but in the present study it does not improve the accuracy, and tends to slow things down.

```
##              train validation test
## observations 11776       7846   20
## variables       53         53   53
```

Model fitting and validation

1. Fitting of a random forest model

We fit the data with a random forest model. The cross-validation is done by a 5-fold algorithm.

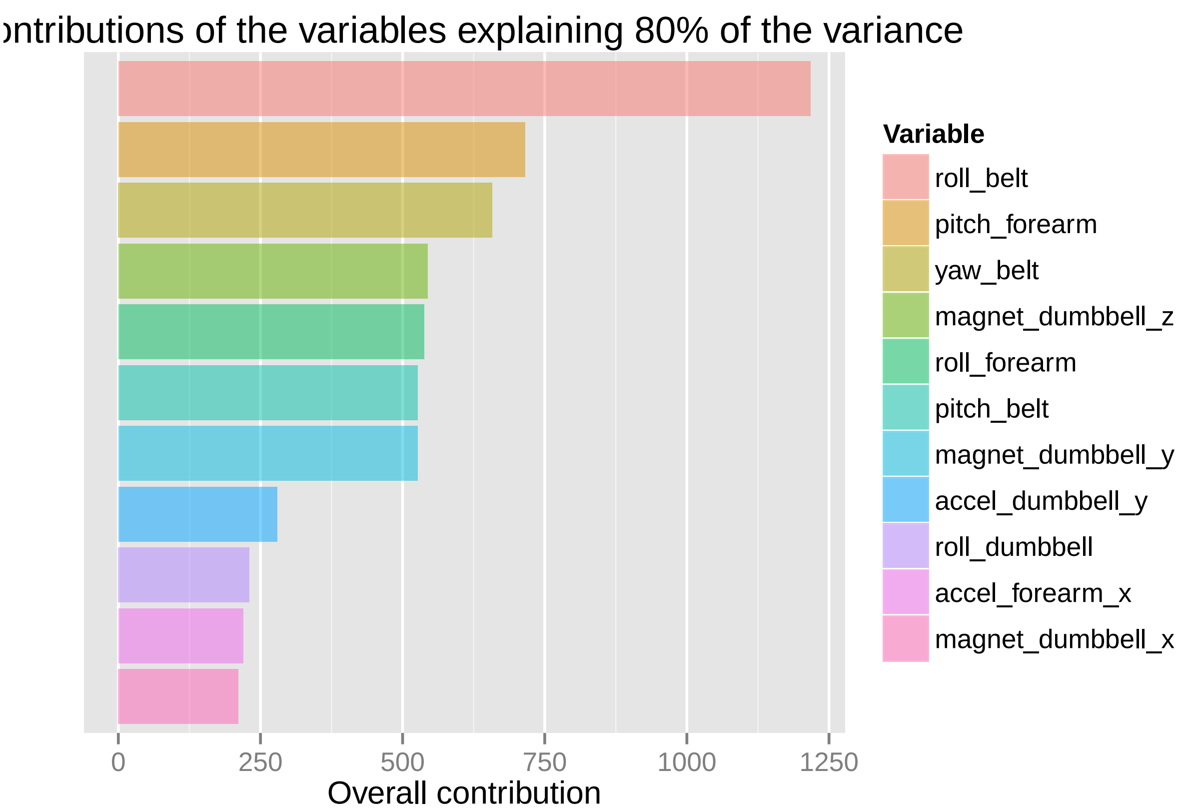
```
train_control <- trainControl(method="cv", number=5)
modelFit_rf <- train(train$classe ~ .,
                     data=train,
                     method="rf",
                     preProcess=c("center", "scale"),
                     trControl=train_control)
```

Main characteristics of the model :

```
modelFit_rf
```

```
## Random Forest
##
## 11776 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (52), scaled (52)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 9420, 9420, 9420, 9423, 9421
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9881972  0.9850673  0.002704567  0.003423212
##   27    0.9893007  0.9864650  0.001248219  0.001578435
##   52    0.9846309  0.9805570  0.005327074  0.006741144
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

The following figure shows the individual contribution of each variable (by simplicity, I represented only the variables which explain 80% of the variability). As we can see, the **roll_belt**, **pitch_forearm** and **yaw_belt** are the most relevant predictors for the study of the activity class. We could use this approach to select a smaller number of predictors, but we won't do it here. Indeed, it will slow things down without improving much the accuracy.



2. Validation of the result

The results are validated on the validation dataset. The confusion matrix describes the performance of the random forest model, by comparing the prediction of the algorithm with true data. We get a very good accuracy of 99.64% !

```
confusionMatrix(validation$classe, predict(modelFit_rf, validation))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    0    0    0    0
##          B    3 1515    0    0    0
##          C    0    2 1358    8    0
##          D    0    0    6 1279    1
##          E    0    0    2    1 1439
##
## Overall Statistics
##
##                Accuracy : 0.9971
##                  95% CI : (0.9956, 0.9981)
##     No Information Rate : 0.2849
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9963
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9987   0.9987   0.9941   0.9930   0.9993
## Specificity            1.0000   0.9995   0.9985   0.9989   0.9995
## Pos Pred Value         1.0000   0.9980   0.9927   0.9946   0.9979
## Neg Pred Value         0.9995   0.9997   0.9988   0.9986   0.9998
## Prevalence             0.2849   0.1933   0.1741   0.1642   0.1835
## Detection Rate         0.2845   0.1931   0.1731   0.1630   0.1834
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9993   0.9991   0.9963   0.9960   0.9994
```

Now let's calculate the out-of-sample error :

```
sum(predict(modelFit_rf, validation) != validation$classe)/length(validation$classe)
```

```
## [1] 0.00293143
```

Predictions on test cases

The test dataset has no *classe* variable, but we can predict it thanks to our algorithm :

```r
########## PREDICTION ON TEST CASES ##########
predict(modelFit_rf, test)
```

```
##  [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
```

Appendix

```r
sessionInfo()
```

```
## R version 3.2.3 (2015-12-10)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.10.5 (Yosemite)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] randomForest_4.6-12 tidyr_0.4.1         caret_6.0-64
## [4] lattice_0.20-33     knitr_1.11          ggplot2_1.0.1
## [7] dplyr_0.4.3
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.1        compiler_3.2.3     formatR_1.2.1
##  [4] nloptr_1.0.4       plyr_1.8.3         class_7.3-14
##  [7] iterators_1.0.8    tools_3.2.3        digest_0.6.8
## [10] lme4_1.1-10        evaluate_0.8       gtable_0.1.2
## [13] nlme_3.1-122       mgcv_1.8-9         Matrix_1.2-3
## [16] foreach_1.4.3      DBI_0.3.1          yaml_2.1.13
## [19] parallel_3.2.3     SparseM_1.7        proto_0.3-10
## [22] e1071_1.6-7        stringr_1.0.0      MatrixModels_0.4-1
## [25] stats4_3.2.3       grid_3.2.3         nnet_7.3-11
## [28] R6_2.1.1           rmarkdown_0.8.1    minqa_1.2.4
## [31] reshape2_1.4.1     car_2.1-1          magrittr_1.5
## [34] scales_0.3.0       codetools_0.2-14   htmltools_0.2.6
## [37] MASS_7.3-45        splines_3.2.3      assertthat_0.1
## [40] pbkrtest_0.4-6     colorspace_1.2-6   labeling_0.3
## [43] quantreg_5.19      stringi_0.5-5      lazyeval_0.1.10
## [46] munsell_0.4.2
```