Final Project ML

O. Paz

29-11-2020

Background information:

This report is meant to fulfill the final project of Coursera's Practical Machine learning course. We are asked to study and make predictions of the variable "classe", part of the data sets shown below, based on all predictors that one deems fit.

The setting of the activity where the data come from is as follows: 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). The event was run such that Class "A" corresponded to "correct way" and the rest were just 4 different ways of doing it wrong. All data were collected from movement detector devices installed on the six individuals. Hence, the task is to come up with a ML model to predict the outcome from the testing set.

Read more: http:/groupware.les.inf.puc-rio.br/har#ixzz4Tjuy9Csb (http://groupware.les.inf.puc-rio.br/har#ixzz4Tjuy9Csb)

Training data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)

Testing Data: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

Exploratory Analysis:

First off the data were dowloaded, read.csv seemed appropriate, dropping the first column (redundant afterwards):

```
training <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv")
training<- training[,-1]

testing <- read.csv("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv")
testing <- testing[,-1]
```

A first glimpse to the structure and content of the data showed quite a few variables with "NA", in a percentage that rendered them useless:

```
table(colSums(is.na(training)))
```

```
##
##     0 19216
##    92    67
```

Therefore, there are 67 variables with exactly 19.216 NAs each, which is about 98% of all the values, hence it's safe to assume that they will not influence our outcome and can be retrieved:

```
index <- colSums(is.na(training))!=0
training_red <- training %>%  select_if(!index)
```

Now we are left with 92 variables, but there are also quite a few with blanks "" in great number:

```
table(colSums(training_red==""))
```

```
##
##     0 19216
##    59    33
```

Almost 98% blanks as well, so I better take them out, and be left with 59 variables:

```
index2 <- colSums(training_red=="")!=0
training_red2 <- training_red %>%  select_if(!index2)
```

I will do the same for testing set, all at once now:

```
indext <- colSums(is.na(testing))!=0|colSums(testing=="")!=0
testing_red <- testing %>%  select_if(!indext)
```

Let's take a look at the participants and classes distributions in the reduced training set:

```
table(training_red2$user_name)
```

```
##
##   adelmo carlitos  charles   eurico   jeremy    pedro
##     3892     3112     3536     3070     3402     2610
```

```
table(training_red2$classe)
```

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

So, the participants activities were more or less evenly distributed and the most performed activity was "A" (correct). Hopefully the samples were taken so that they are normally distributed, despite the natural constraints of the type of experiment.

Finally, I will also remove 6 qualitative variables, that are not defined as outcome and are not contributing features, namely: "user_name","raw_timestamp_part_1", "raw_timestamp_part_2" , "cvtd_timestamp"
, "new_window" and "num_window".

```
training_red3 <- training_red2[,-c(1:6)]
testing_red3 <- testing_red[,-c(1:6)]
```

I end up with a reduced data set now, to start the real work, and I don't have to impute any missing value, which is nice. The data frame is also deemed as "tidy", provided that each variable forms a column and each observation forms a row. The variable names are quite descriptive so I will not mess with them.

Pre-processing:

Just to start stating the obvious, we are dealing with a supervised/classification model. I will adequate the data by transforming our outcome (classe) to factor and all others to numeric:

```
training_red3$classe <- factor(training_red3$classe)
training_red3 <- training_red3 %>%
  mutate_at(vars(1:52), as.numeric)
```

Now is a good time to look for "colinearity", particularly in this set-up where we know before hand that all data come from 6 fix sources (individuals) and the the produced data themselves are produced by constrained events (movements). I will show only the first and last variable correlations and just first 6 variables, for space constraints:

```
Cor <- cor(training_red3[,-53])
head(Cor)[,1:6]
```

```
##                   roll_belt pitch_belt   yaw_belt total_accel_belt gyros_belt_x
## roll_belt         1.0000000 -0.2159251  0.8152297        0.9809241   -0.1174696
## pitch_belt       -0.2159251  1.0000000 -0.6997519       -0.1389812   -0.4359640
## yaw_belt          0.8152297 -0.6997519  1.0000000        0.7620963    0.1450478
## total_accel_belt  0.9809241 -0.1389812  0.7620963        1.0000000   -0.1653112
## gyros_belt_x     -0.1174696 -0.4359640  0.1450478       -0.1653112    1.0000000
## gyros_belt_y      0.4637184 -0.3971118  0.5300448        0.4093150    0.3331461
##                  gyros_belt_y
## roll_belt           0.4637184
## pitch_belt         -0.3971118
## yaw_belt            0.5300448
## total_accel_belt    0.4093150
## gyros_belt_x        0.3331461
## gyros_belt_y        1.0000000
```

```
tail(Cor)[,1:6]
```

```
##                    roll_belt  pitch_belt    yaw_belt total_accel_belt
## accel_forearm_x  -0.48544966  0.12787561 -0.39346742      -0.44779279
## accel_forearm_y   0.03082366 -0.36170195  0.23750551       0.01934804
## accel_forearm_z   0.08194973 -0.23047865  0.17139010       0.04425207
## magnet_forearm_x -0.19180096 -0.07209996 -0.09026972      -0.18575275
## magnet_forearm_y  0.02950358 -0.01700917  0.04369715       0.03974689
## magnet_forearm_z  0.27259651 -0.06382742  0.22848665       0.29699417
##                  gyros_belt_x gyros_belt_y
## accel_forearm_x   -0.21234754  -0.37941989
## accel_forearm_y   -0.04280212  -0.01297882
## accel_forearm_z    0.34244043   0.26365255
## magnet_forearm_x   0.04879487  -0.03173921
## magnet_forearm_y   0.01876251   0.01241846
## magnet_forearm_z  -0.11062464   0.11914935
```

Not surprisingly, the total acceleration, for instance, is closely related to accelerations in y and z axis, although not so much with with x axis (my guess is that the up and down movements were recorded in y-z axis). In general though, there seem to be many non-linear relationships as to make a good prediction model, once we narrow down the predictors.

To make it more precise:

```
sum(abs(Cor[upper.tri(Cor)]) > .75)
```

```
## [1] 31
```

```
sum(abs(Cor[upper.tri(Cor)]) > .90)
```

```
## [1] 11
```

```
sum(abs(Cor[upper.tri(Cor)]) > .95)
```

```
## [1] 5
```

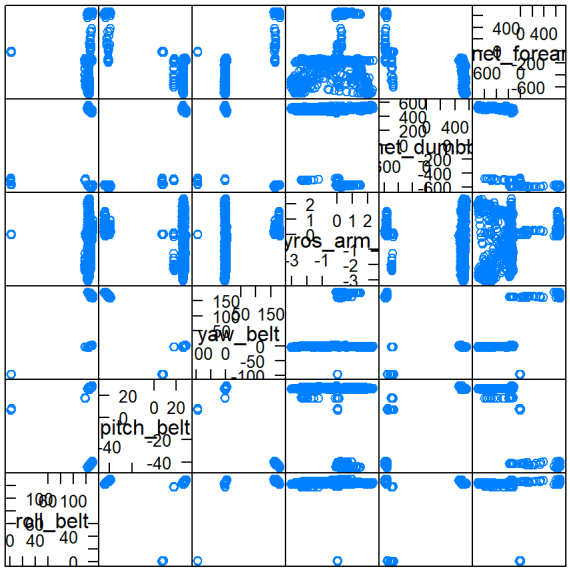
```
summary(Cor[upper.tri(Cor)])
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.992008 -0.110080  0.002092  0.001790  0.092552  0.980924
```

There are 31 features with correlation > 75% , 11 with correlation > 90% and 5 with correlation > 95%. Also the short summary shows the distribution.

Let's look at the graphs, taking only a few variables of different nature (roll, pitch,yaw, gyroscope, magnet, acceleration):

```
featurePlot(x = training_red3[1:1000,c(1:3,18,37,50)], y = training_red3$classe, plot = "pairs")
```



Scatter Plot Matrix

I know it's messy but it just goes to show that there are clear patterns among the features.

We can set up a cut-off value for the correlation of 0.75 and subset our data before continuing on (will also apply on testing set):

```
indexCor <- findCorrelation(Cor, cutoff = .75)
training_red3 <- training_red3[,-indexCor]
testing_red3 <- testing_red3[,-indexCor]
Cor2 <- cor(training_red3[,-32])
summary(Cor2[upper.tri(Cor2)])
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.606983 -0.103773  0.006527  0.003332  0.087527  0.736546
```

Now the linearity was reduced substantially.

There are other things one can do to prep-up the data before passing to choose the algorithm but I will use random forest, which, just to name one feature, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error because it's implicit in the process by bootstraping within every tree (other advantages in the next section).

Modeling and predicting:

There are many reasons to choose random forest for this classification problem. Basically, it's tree-based and, as an ensemble algorithm, it puts together the known advantages of other algorithms as well with little to no tuning and minimum pre-processing as most of it is implicit in the process.

Here's an extract from "Random Forests, by Leo Breiman and Adele Cutler" that describes other very interesting features:

- It is unexcelled in accuracy among current algorithms.
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.
- Prototypes are computed that give information about the relation between the variables and the classification.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.

Now the fun part, training and predicting:

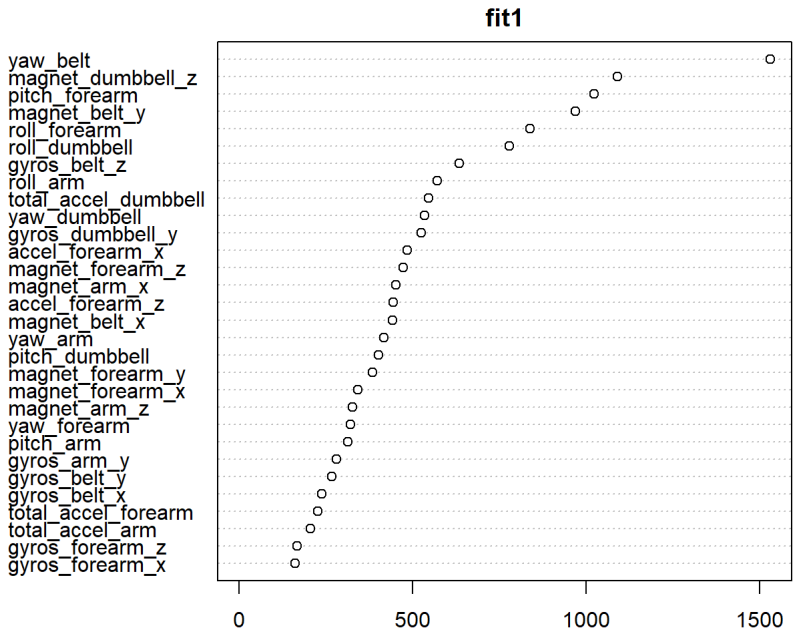
```
fit1 <- randomForest(classe~., data = training_red3, type="class")
fit1
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training_red3, type = "class")
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 0.44%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 5577    3    0    0    0 0.0005376344
## B   10 3778    6    0    3 0.0050039505
## C    0   15 3391   16    0 0.0090590298
## D    0    0   25 3186    5 0.0093283582
## E    0    0    1    3 3603 0.0011089548
```

It shows in the summary an out-of-bag error of 0.44% (very very good) and the confusion matrix shows very few Type I or II errors (excellent). The algorith used 500 trees and each node was split into 5, according to the information.

It's always interesting to see the importance of the variables:

```
par(mar = rep(2,4))
varImpPlot(fit1)
```

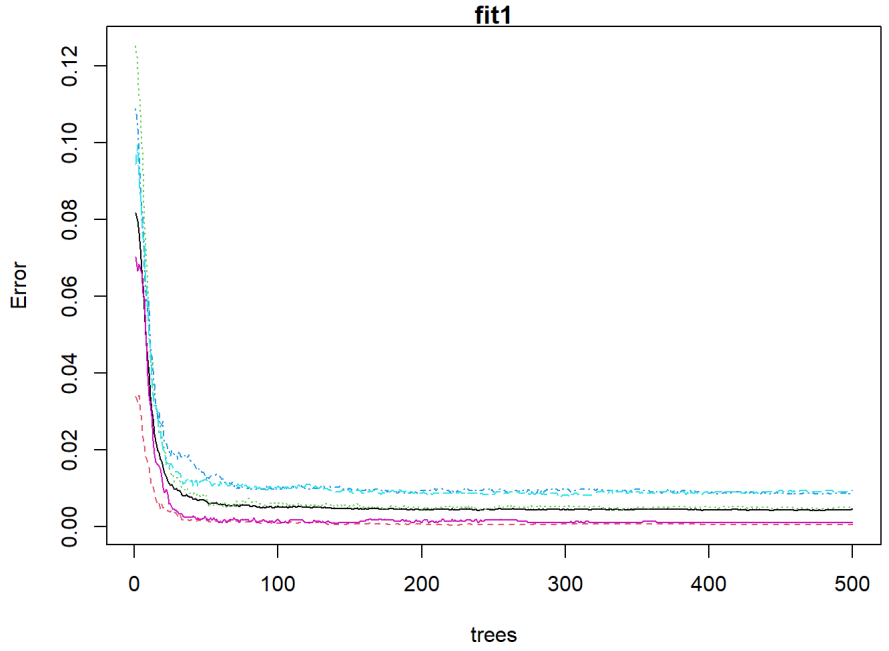



```
importance(fit1)
```

```
##                        MeanDecreaseGini
## yaw_belt                     1530.6491
## gyros_belt_x                  237.5733
## gyros_belt_y                  267.4813
## gyros_belt_z                  633.8295
## magnet_belt_x                 441.9060
## magnet_belt_y                 968.7362
## roll_arm                      570.3235
## pitch_arm                     313.5053
## yaw_arm                       417.0736
## total_accel_arm               205.6915
## gyros_arm_y                   280.8794
## gyros_arm_z                   123.3862
## magnet_arm_x                  451.7266
## magnet_arm_z                  327.2658
## roll_dumbbell                 779.6782
## pitch_dumbbell                401.7427
## yaw_dumbbell                  534.0656
## total_accel_dumbbell          545.6351
## gyros_dumbbell_y              525.2842
## magnet_dumbbell_z            1089.5285
## roll_forearm                  839.0721
## pitch_forearm                1022.8726
## yaw_forearm                   321.8202
## total_accel_forearm           226.6147
## gyros_forearm_x               161.3690
## gyros_forearm_z               167.2035
## accel_forearm_x               484.2498
## accel_forearm_z               443.9062
## magnet_forearm_x              342.0030
## magnet_forearm_y              384.4374
## magnet_forearm_z              473.3306
```

Let's see the performance of the trees compared to errors:

```
par(mar= c(3.8,3.8,1,1))
plot(fit1)
```



Looks like we could have reached errors close to zero with far less trees, random forest is very potent!

Finally, let's validate our model with the testing set by making a prediction:

```
pred1 <- predict(fit1, newdata=testing_red3, type="class")
pred1
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```

The prediction "pred1" turned aout to be 100% accurate according to the quiz 4 of the course.

Finally, it would have been interesting to run another confusion matrix to compare results of the prediction against the testing set, but we were not given those classes (it would have been cheating) and I didn't take any additional testing set (maybe I should have), but the result of the quiz and the OOB parameter (< 1%) show the accuracy of the model.

Final words:

I acknowledge that maybe I didn't have to do so much pre-processing (for instance, narrow down the predictors so much) and even some of the data cleaning because many of those things are taken care of by the random forest algorithm itself, but since this is an academic exercise I wanted to take it step by step and show the line of thought of some general methodology to approach these interesting problems.

THE END :-)