Practical Machine Learning Coursera Project

Abstract

This is a peer assignment of Coursera's Practical Machine Learning course 2015. The goal was to build a model able to classify the body actions (standing, running, walking, etc) out of data provided by smartphone accelerometers. I used randomForest algorithm nad obtained satisfactory results.

Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of 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 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. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har.

Load libraries

```
# load libraries
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
library(randomForest)
```

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

Load datasets

And deletion of columns with missing values.

```
# Import training database
raw.data<- read.csv(file = "pml-training.csv",
na.strings=c("","NA"))
# remove colums with NA's
raw.data2<- raw.data[, !(colSums(is.na(raw.data)) >= 1 )]
#load validation dataset (test set for submission)
validation.data<- read.csv(file = "pml-testing.csv",
na.strings=c("","NA"))
```

Predictor selection - deletion of meaningles variables to the model

```
dataset <- raw.data2[, c(-1,-2,-5,-6)]
str(dataset)
```

```
## 'data.frame':    19622 obs. of  56 variables:
##  $ 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 ...
##  $ 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 ...
##  $ 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
...
##  $ 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 ...
##  $ 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
...
##  $ total_accel_dumbbell: int  37 37 37 37 37 37 37 37 37 37
...
##  $ gyros_dumbbell_x    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ gyros_dumbbell_y    : num  -0.02 -0.02 -0.02 -0.02 -0.02
-0.02 -0.02 -0.02 -0.02 -0.02 ...
##  $ gyros_dumbbell_z    : num  0 0 0 -0.02 0 0 0 0 0 0 ...
##  $ accel_dumbbell_x    : int  -234 -233 -232 -232 -233 -234
-232 -234 -232 -235 ...
##  $ accel_dumbbell_y    : int  47 47 46 48 48 48 47 46 47 48
...
##  $ accel_dumbbell_z    : int  -271 -269 -270 -269 -270 -269
-270 -272 -269 -270 ...
##  $ magnet_dumbbell_x   : int  -559 -555 -561 -552 -554 -558
-551 -555 -549 -558 ...
##  $ magnet_dumbbell_y   : int  293 296 298 303 292 294 295
300 292 291 ...
##  $ magnet_dumbbell_z   : num  -65 -64 -63 -60 -68 -66 -70
-74 -65 -69 ...
##  $ roll_forearm        : num  28.4 28.3 28.3 28.1 28 27.9
27.9 27.8 27.7 27.7 ...
##  $ pitch_forearm       : num  -63.9 -63.9 -63.9 -63.9 -63.9
-63.9 -63.9 -63.8 -63.8 -63.8 ...
##  $ yaw_forearm         : num  -153 -153 -152 -152 -152 -152
-152 -152 -152 -152 ...
##  $ total_accel_forearm : int  36 36 36 36 36 36 36 36 36 36
...
##  $ gyros_forearm_x     : num  0.03 0.02 0.03 0.02 0.02 0.02
0.02 0.02 0.03 0.02 ...
##  $ gyros_forearm_y     : num  0 0 -0.02 -0.02 0 -0.02 0
-0.02 0 0 ...
```

```
##  $ gyros_forearm_z     : num  -0.02 -0.02 0 0 -0.02 -0.03
-0.02 0 -0.02 -0.02 ...
##  $ accel_forearm_x     : int  192 192 196 189 189 193 195
193 193 190 ...
##  $ accel_forearm_y     : int  203 203 204 206 206 203 205
205 204 205 ...
##  $ accel_forearm_z     : int  -215 -216 -213 -214 -214 -215
-215 -213 -214 -215 ...
##  $ magnet_forearm_x    : int  -17 -18 -18 -16 -17 -9 -18 -9
-16 -22 ...
##  $ magnet_forearm_y    : num  654 661 658 658 655 660 659
660 653 656 ...
##  $ magnet_forearm_z    : num  476 473 469 469 473 478 470
474 476 473 ...
##  $ classe              : Factor w/ 5 levels
"A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Splitting the dataset into the training and testing sets:

```
inTrain = createDataPartition(dataset$classe, p = 0.1, list =
F)
training = dataset[ inTrain,]
testing = dataset[-inTrain,]
```

The training set consisted of 1964 rows. The testing set consisted of 17658 rows.

The validation set (for submission) consisted of 20 rows.

This inapproriate split ratio is due to the fact, that old IBM tablet was used and training took 2hrs even on 10% of records… But still, prediction error is suprisingly low.

Model training

Actual model training:

```
#modFit <- train(classe ~ ., data=training, method="rf")  #
hashed because knitted from saved model
#save(modFit, file="modFit1.RData")
```

randomForest was choosen as a modeling tool due to its good clasification capabilities. It also does cross-validation automatically.

Loading previously built model:

```
load("modFit1.RData")
#oad("modFit2.RData")  #second run
```

Model details:

```
print(modFit)
```

```
## Random Forest
##
## 1964 samples
##   55 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1964, 1964, 1964, 1964, 1964,
1964, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa   Accuracy SD  Kappa SD
##    2    0.9473    0.9333  0.008751     0.011028
##   28    0.9715    0.9640  0.005841     0.007392
##   55    0.9658    0.9567  0.008931     0.011329
##
## Accuracy was used to select the optimal model using  the
largest value.
## The final value used for the model was mtry = 28.
```

```
print(modFit$finalModel)
```

```
## 
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry) 
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 28
## 
##         OOB estimate of  error rate: 2.09%
## Confusion matrix:
##     A   B   C   D   E class.error
## A 556   0   1   0   1    0.003584
## B   6 368   6   0   0    0.031579
## C   0   8 334   1   0    0.026239
## D   0   0  10 312   0    0.031056
## E   0   2   2   4 353    0.022161
```

ConfusionMatrix for traininset (internal validation)

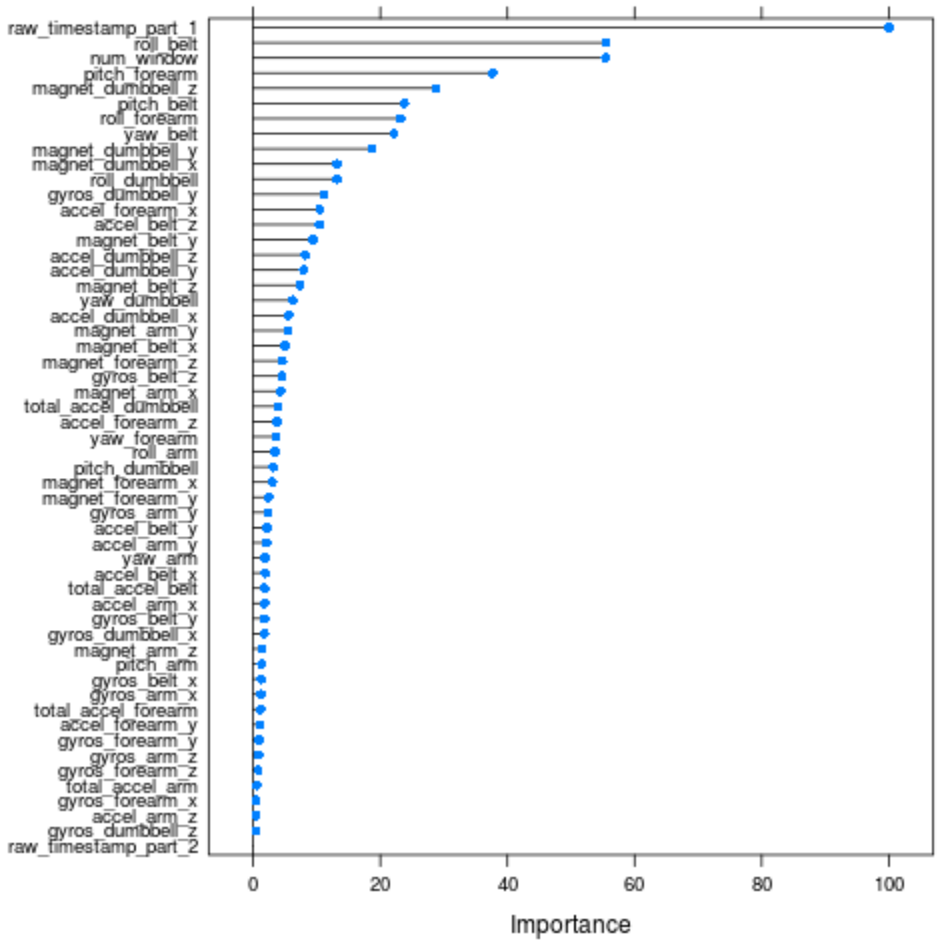
```
train.pred <- predict(modFit, training)
confusionMatrix(training$classe, train.pred)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   A   B   C   D   E
##          A 557   1   0   0   0
##          B   5 363  12   0   0
##          C   0   9 333   1   0
##          D   0   0   5 317   0
##          E   0   0   2   6 353
## 
## Overall Statistics
## 
##                Accuracy : 0.979
##                  95% CI : (0.972, 0.985)
##     No Information Rate : 0.286
##     P-Value [Acc > NIR] : <2e-16
## 
##                   Kappa : 0.974
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D
Class: E
## Sensitivity             0.991    0.973    0.946    0.978
1.000
## Specificity             0.999    0.989    0.994    0.997
0.995
## Pos Pred Value          0.998    0.955    0.971    0.984
0.978
## Neg Pred Value          0.996    0.994    0.988    0.996
1.000
## Prevalence              0.286    0.190    0.179    0.165
0.180
## Detection Rate          0.284    0.185    0.170    0.161
0.180
## Detection Prevalence    0.284    0.193    0.175    0.164
0.184
## Balanced Accuracy       0.995    0.981    0.970    0.988
0.998
```

```
confmx1 <-confusionMatrix(training$classe, train.pred)
accuracy1 <- round(confmx1$overall[[1]]*100, 1)
```

Input variables relative importance is shown on the plot:

```
plot(varImp(modFit, ))
```



Validation with testing dataset

confusionMatrix for testing dataset:

```
test.pred <- predict(modFit, testing)
confusionMatrix(testing$classe, test.pred)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 5012   10    0    0    0
##          B   50 3295   71    0    1
##          C    0   95 2979    5    0
##          D    0    0   62 2830    2
##          E    0    3   25   23 3195
##
## Overall Statistics
##
##                Accuracy : 0.98
##                  95% CI : (0.978, 0.982)
##     No Information Rate : 0.287
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.975
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D
Class: E
## Sensitivity             0.990    0.968    0.950    0.990
0.999
## Specificity             0.999    0.991    0.993    0.996
0.996
## Pos Pred Value          0.998    0.964    0.968    0.978
0.984
## Neg Pred Value          0.996    0.992    0.989    0.998
1.000
## Prevalence              0.287    0.193    0.178    0.162
0.181
## Detection Rate          0.284    0.187    0.169    0.160
0.181
## Detection Prevalence    0.284    0.194    0.174    0.164
0.184
## Balanced Accuracy       0.995    0.980    0.971    0.993
0.998
```

```
confmx2 <-confusionMatrix(testing$classe, test.pred)
accuracy2 <- round(confmx2$overall[[1]]*100, 1)
```

Accuracy has increased from 97.9% for training set to 98% for validation test. It means that in this example out-of-sample error is even lower than this obtained on training dataset. Suprisingly, considering that only 10% of dataset was used for training.

Validation online (test set of 20 records)

Model was validated via online submission:

```
val.pred <- predict(modFit, validation.data)
val.pred
```

```
##  [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
```

20/20 correct