Predicting the quality of weight lifting exercises

Michel de Leeuw

July 24, 2016

Overview

In this report for the Coursera Data Science Peer Graded Assignment Machine Learning we will investigate the possibility to predict how well people perform a weight lifting excercise. Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in a number of different fashions: Doing it right (Class A) and doing it wrong (Class B to E). We will investigate the result from the experiment by means of the “Weight Lifting Exercises Dataset” from the Human Activity Recognition Project (http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises#ixzz4FN1LOyrC (http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises#ixzz4FN1LOyrC)). We will fit some models to expiriment with machine learning and we will see that a random forest model is a very good fit.

Data preparation

First we download and read the training and testing sets to dataframes. Note that some columns have #DIV/0! as value. Because that doesn’t have meaning in our context, we see them as NA’s.

```
# download and read the training set
url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
training <- read.csv(url(url), na.strings=c("NA","NaN", "#DIV/0!", " ") )

# download and read the testing set
url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
testing <- read.csv(url(url), na.strings=c("NA","NaN", "#DIV/0!", " ") )
```

We remove de following columns: X (the row number), username, columns that reference the date or time of the exercise, new_window, num_window and all the columns that are NA for rows. For this check we exclude rows where new_window equals yes because of the columns that are aggregates for the window and we want to create a model that estimates the classe for individual measures and not for windows.

```
# Columns to be removed by name
trainingColsRemove <- c("X", "user_name", "raw_timestamp_part_1",
                        "raw_timestamp_part_2", "cvtd_timestamp",
                        "new_window", "num_window")

# Columns to be removed because all NA's for new_window is no
trainingColsNA <- apply(is.na(training[training$new_window=='no',]),2,all)
trainingColsNA <- names(training)[trainingColsNA]

# Remove the columns
training <- training[, !(names(training) %in% c(trainingColsRemove, trainingColsNA
))]
```

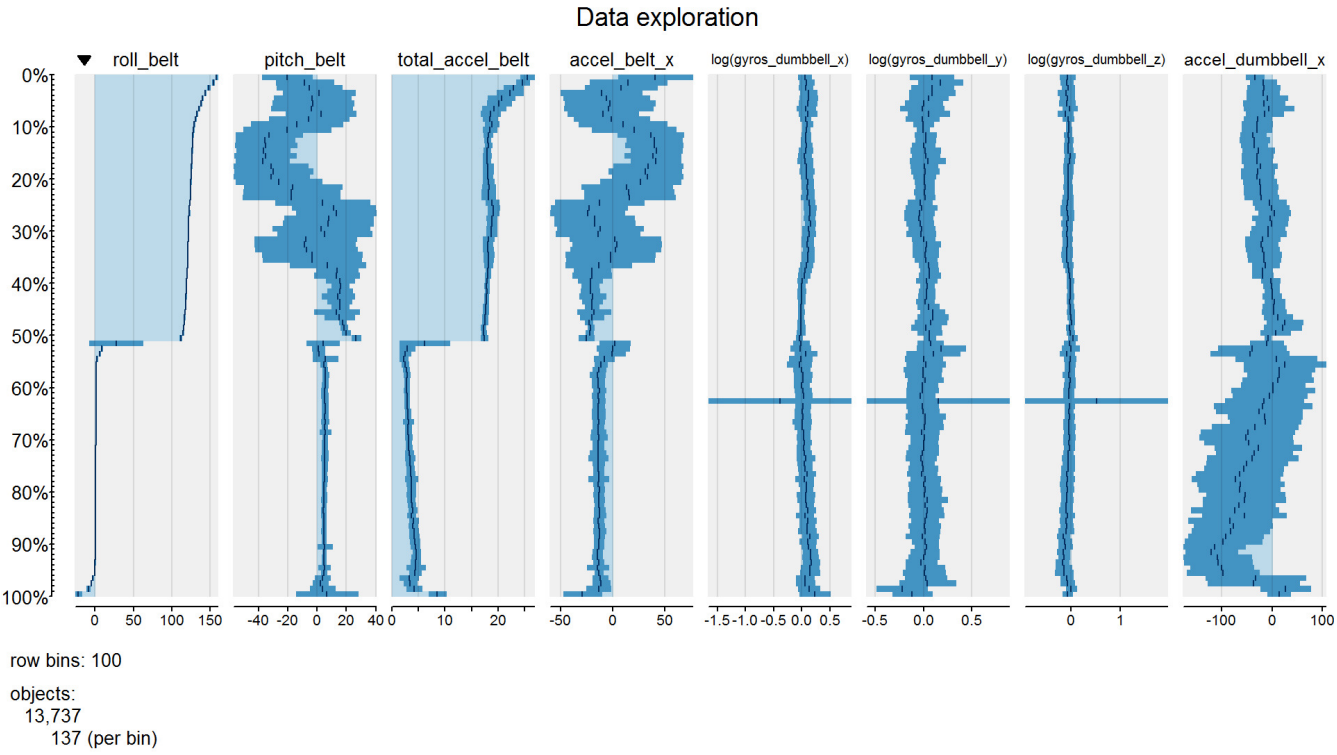
We then split the training data in a 70% part that is used for training the models (trainingTrain) and a 30% part that is used for testing and evaluating the models after they are trained (trainingTest).

```
library(caret)
set.seed(1234)

# Use 70% for the training set and 30% for the test set
inTrain = createDataPartition(training$classe, p = 0.7)[[1]]
trainingTrain = training[ inTrain,]
trainingTest = training[ -inTrain,]
```

Before fitting some models, we want to see if there are outliers we want to correct. To do so we use the tableplot graph from the tabplot library and combine it with a describe for all the columns. In the appendix the code is provided to do so. In the tableplot below only a few columns are displayed.

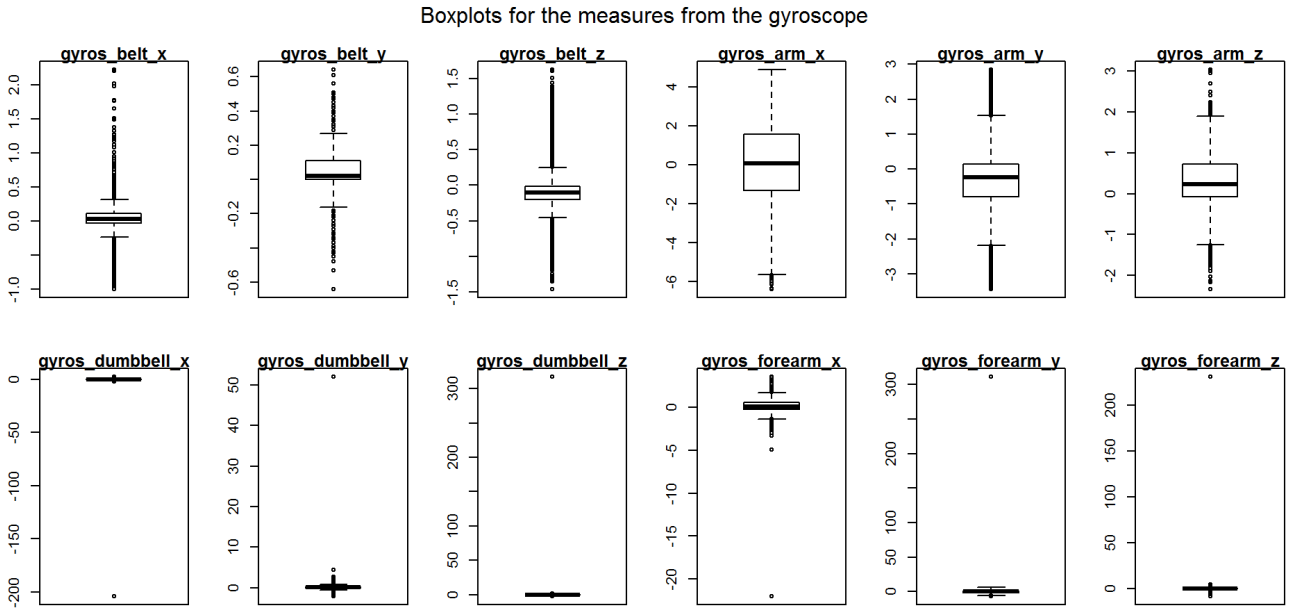
```
library(tabplot)
tableplot(trainingTrain[, c(1, 2, 4, 8, 31:34)], title="Data exploration", cex = 1
.8)
```



From the plots and describes (as mentioned, the code is in the appendix), we conclude we need to investigate the measures from the gyroscope, because some of the values seem to be out of order. To investigate we create boxplots for them.

```
# Create a dataframe with only the measures from the gyroscope
trainingTrainGyros <- trainingTrain[,grep("gyros", names(trainingTrain))]

# Draw boxplots for them into a single plot
oldpar <- par(mfrow=c(1,2), mar=c(3,3,1,1), oma=c(0,0,3,1))  ## oma creates space
par(mfrow=c(2,6))
for (gyrosCol in names(trainingTrainGyros)) {
  boxplot(trainingTrainGyros[, gyrosCol], main=gyrosCol)
}
mtext("Boxplots for the measures from the gyroscope", side=3, line=1, outer=TRUE,
cex=1, font=1)
```



It is now clear that some of the values are outliers. We decide to replace all values smaller than -10 or larger than 10 by their corresponding mean value for all the gyroscope columns. Possibly, there is a bug in the device that registers the values from the gyroscope.

```
# Create a function that replaces out of bounds values with their corresponding me
an
removeOutliersGyros <- function(dataset) {
  dataset$gyros_belt_x [dataset$gyros_belt_x < -10 | dataset$gyros_belt_x > 10] <-
 mean(dataset$gyros_belt_x)
  dataset$gyros_belt_y [dataset$gyros_belt_y < -10 | dataset$gyros_belt_y > 10] <-
 mean(dataset$gyros_belt_y)
  dataset$gyros_belt_z [dataset$gyros_belt_z < -10 | dataset$gyros_belt_z > 10] <-
 mean(dataset$gyros_belt_z)
  dataset$gyros_arm_x [dataset$gyros_arm_x < -10 | dataset$gyros_arm_x > 10] <- me
an(dataset$gyros_arm_x)
  dataset$gyros_arm_y [dataset$gyros_arm_y < -10 | dataset$gyros_arm_y > 10] <- me
an(dataset$gyros_arm_y)
  dataset$gyros_arm_z [dataset$gyros_arm_z < -10 | dataset$gyros_arm_z > 10] <- me
an(dataset$gyros_arm_z)
  dataset$gyros_dumbbell_x [dataset$gyros_dumbbell_x < -10 | dataset$gyros_dumbbel
l_x > 10] <-
          mean(dataset$gyros_dumbbell_x)
  dataset$gyros_dumbbell_y [dataset$gyros_dumbbell_y < -10 | dataset$gyros_dumbbel
l_y > 10] <-
          mean(dataset$gyros_dumbbell_y)
  dataset$gyros_dumbbell_z [dataset$gyros_dumbbell_z < -10 | dataset$gyros_dumbbel
l_z > 10] <-
          mean(dataset$gyros_dumbbell_z)
  dataset$gyros_forearm_x [dataset$gyros_forearm_x < -10 | dataset$gyros_forearm_x
 > 10] <-
          mean(dataset$gyros_forearm_x)
  dataset$gyros_forearm_y [dataset$gyros_forearm_y < -10 | dataset$gyros_forearm_y
 > 10] <-
          mean(dataset$gyros_forearm_y)
  dataset$gyros_forearm_z [dataset$gyros_forearm_z < -10 | dataset$gyros_forearm_z
 > 10] <-
          mean(dataset$gyros_forearm_z)
  return(dataset)
}

# Apply the function on the trainingset
trainingTrain <- removeOutliersGyros(trainingTrain)
```

Model training

Our training set (trainTrain) is now ready for fitting models. We decide to fit 3 models with 3 different methods we studied in the course and see how well they fit. The first method is the boosting with trees (gbm) method. It uses boosted decision trees. The second method is a non-boosted decision tree (rpart). From this we want to see what the effect of boosting is. The third method is the random forest (rf) method. As I understand, this is a widely used method, so we want to consider it.

Cross validation within the training of the models will be done by k-fold cross validation. We use the trainControle function for this and set the number of folds equal to 7. Perhaps it is good to mention that this is not the same as the out of sample validation, we will do after the models are trained.

```
set.seed(2345)
tc <- trainControl(method = "cv", number = 7)
fit_gbm <- train(classe~., data=trainingTrain, method="gbm", verbose = FALSE, trCo
ntrol= tc)
fit_rpart <- train(classe~., data=trainingTrain, method="rpart", trControl= tc)
fit_rf <- train(classe~., data=trainingTrain, method="rf", trControl= tc)
```

Model selection

Now that we have trained the models we want to choose the best one. First, let's look at the in sample accuracy.

```
library(knitr)

# Get the in sample accuracy from the results part of the model
result <- cbind (Method=c("GBM", "Rpart", "RF"),
                 Accuracy=round(c(max(fit_gbm$results$Accuracy),
                                  max(fit_rpart$results$Accuracy),
                                  max(fit_rf$results$Accuracy)), 3))
kable(result)
```

Method	Accuracy
GBM	0.961
Rpart	0.535
RF	0.992

As we expected, the boosted version of decision tree (GBM) is much more accurate than the none boosted version (Rpart). The random forest variant has the best in sample accuracy. Before we make a decision on the model, we will look at the out of sample accuracy. We apply the GBM and RF models to the trainTest set.

```
# Remove outlier as with the training set
trainingTest <- removeOutliersGyros(trainingTest)

# Apply the GBM model and show confusion matrix
pred_gbm <- predict(fit_gbm, trainingTest)
xtab_gbm <- table (pred_gbm, real=trainingTest$classe)
confusionMatrix(xtab_gbm)
```

```
## Confusion Matrix and Statistics
##
##          real
## pred_gbm    A    B    C    D    E
##        A 1659   43    0    0    1
##        B   10 1069   36    6    9
##        C    3   25  977   19   14
##        D    1    2   10  930   14
##        E    1    0    3    9 1044
##
## Overall Statistics
##
##                Accuracy : 0.965
##                  95% CI : (0.96, 0.9695)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9557
##  Mcnemar's Test P-Value : 4.969e-07
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9910   0.9385   0.9522   0.9647   0.9649
## Specificity            0.9896   0.9871   0.9874   0.9945   0.9973
## Pos Pred Value         0.9742   0.9460   0.9412   0.9718   0.9877
## Neg Pred Value         0.9964   0.9853   0.9899   0.9931   0.9921
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2819   0.1816   0.1660   0.1580   0.1774
## Detection Prevalence   0.2894   0.1920   0.1764   0.1626   0.1796
## Balanced Accuracy      0.9903   0.9628   0.9698   0.9796   0.9811
```

```
# Apply the RF model and show confusion matrix
pred_rf <- predict(fit_rf, trainingTest)
xtab_rf <- table (pred_rf, real=trainingTest$classe)
confusionMatrix(xtab_rf)
```

```
## Confusion Matrix and Statistics
##
##          real
## pred_rf    A    B    C    D    E
##        A 1674    9    0    0    0
##        B    0 1129    4    1    0
##        C    0    1 1018    6    2
##        D    0    0    4  956    4
##        E    0    0    0    1 1076
##
## Overall Statistics
##
##                Accuracy : 0.9946
##                  95% CI : (0.9923, 0.9963)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9931
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9912   0.9922   0.9917   0.9945
## Specificity            0.9979   0.9989   0.9981   0.9984   0.9998
## Pos Pred Value         0.9947   0.9956   0.9912   0.9917   0.9991
## Neg Pred Value         1.0000   0.9979   0.9984   0.9984   0.9988
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1918   0.1730   0.1624   0.1828
## Detection Prevalence   0.2860   0.1927   0.1745   0.1638   0.1830
## Balanced Accuracy      0.9989   0.9951   0.9952   0.9950   0.9971
```

We see that the random forest model has the better out of sample accuracy. Even more, it already had the better in sample accuracy and with an overall out of sample accuracy of 0.99 and with an accuracy of 1 for predicting classe A (which is the right way to do the exercise), we decide that it is the model of our chose. The out of sample error is expected to be 1 minus the out of sample accuracy, which is 1 - 0.99 = 0.01.

Appendix

Credits

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.

Read more: http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises#ixzz4FN26QWUq (http://groupware.les.inf.puc-rio.br/har#weight_lifting_exercises#ixzz4FN26QWUq)

Exploratory analysis

Create tableplot for all variables. Note that the output is not shown, as it would take multiple pages/screens.

```
library(tabplot)

# Split the 53 columns in parts of 5 columns and show them together with the class
e variable
for (i in 0:10) {
  topVar <- min(52,(5*i+5))
  trainingPlot <- training [,c(53, (5*i+1):topVar)]
  tableplot(trainingPlot, title=paste("Column", (5*i+1), "-", min(52,(5*i+5))), ce
x = 1.8)
  print(describe(trainingPlot[,-grep("classe", names(trainingPlot))]))
}
```

Used tooling

This report is created with R version 3.2.3 and Knitr 1.12.3 on x86_64-w64-mingw32/x64 (64-bit) on Windows 10.