Practical Machine Learning Project: Prediction Assignment

Pablo

January 14, 2018

1. Executive Summary

By using devices such as Jawbone Up, Nike FuelBand, and Fitbit is now possible to collect a large amount of data about personal activity relatively inexpensively. This 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 to quantify how much of a particular activity they do, but they rarely quantify how well they do it. Based on data collected from such devices accelerometers, this document presents how several models were built to predict the manner in which people using such devices do a particular activity. Six young health participants were asked to perform one set of 10 repetitions of the Unilateral Dumbbell Biceps Curl in five different fashions classified as classes. Class A when the activity is conducted exactly according to the specification, Class B when throwing the elbows to the front, Class C if the participant is lifting the dumbbell only halfway, Class D if the dumbbell is lowered only halfway and Class E when the participant is throwing the hips to the front. This document presents the results of several models aiming to predict previous behavior; the best model is chosen, explaining the reasons supporting the selection; the picked model is run once on the testing set, getting some prediction results for this data set.

2. Loading and Processing the Data

The first step is to load and clean the data. The strategy for building the prediction model and not overfitting it to the training set will be to split the training set into two subsets, having 70% of the data for training purposes and the remaining as the test set.

```r
library(RColorBrewer)
library(knitr)
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```r
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.6-12

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(ggplot2)
library(knitr)
library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
##
## Attaching package: 'rattle'
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

```
library(RGtk2Extras)
```

```
## Loading required package: RGtk2
```

```
# Download the training and test sets
training <- read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"))
testing  <- read.csv(url("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"))
```

After loading the data some data cleansing is conducted to remove the mostly NA and near zero variance variables. Likewise, the first seven column variables are removed as they do not contain information relevant to the analysis to be carried out.

```
# Data cleaning
NZV <- nearZeroVar(training)
training <- training [, -NZV]
NZV <- nearZeroVar(testing)
testing <- testing [, -NZV]
Mostly_NA<- sapply(training, function(x) mean(is.na(x))) > 0.95
training <- training[, Mostly_NA==FALSE]
Mostly_NA<- sapply(testing, function(x) mean(is.na(x))) > 0.95
testing <- testing[, Mostly_NA==FALSE]
training <- training[, -c(1:7)]
testing <- testing[, -c(1:7)]
dim(training)
```

```
## [1] 19622    52
```

```
dim(testing)
```

```
## [1] 20 52
```

After the cleansing process, both datasets end up having 52 column variables, the training set will have 19,622 observations (ie. rows) and the testing set 20. The next step is to create the 70%/30% partition on the training set, aiming to be able to conduct some pre-testing on the models, without using the actual testing set to avoid overfitting; the latter data set will be used only once at the end of the analysis when the best prediction model be selected.

```
# create the partition (70/30) within the training dataset
inTrain  <- createDataPartition(training$classe, p=0.7, list=FALSE)
Train_Set <- training[inTrain, ]
```

```r
Train_TestSet  <- training[-inTrain, ]
```

3. Prediction Models Design

Basically, two predictions models were analyzed (ie. random forest and decision trees), picking only one based on the accuracy results after running the confusion matrix; the chosen model will be applied to the test set. Each model will be built and tested using the training set and then over the 30% subset of the training set, the latter initially used for pre-testing each individual model, helping in the model selection process. The firs prediction model simulated was random forest and then the analysis was conducted using decision trees.

3.1 Random Forest

Follows the prediction model using random forest.

```r
# model fit
set.seed(98765)
Forest <- trainControl(method="cv", number=3, verboseIter=FALSE)
Fitting_model_RF <- train(classe ~ ., data=Train_Set, method="rf",
                          trControl=Forest)
Fitting_model_RF$finalModel
```

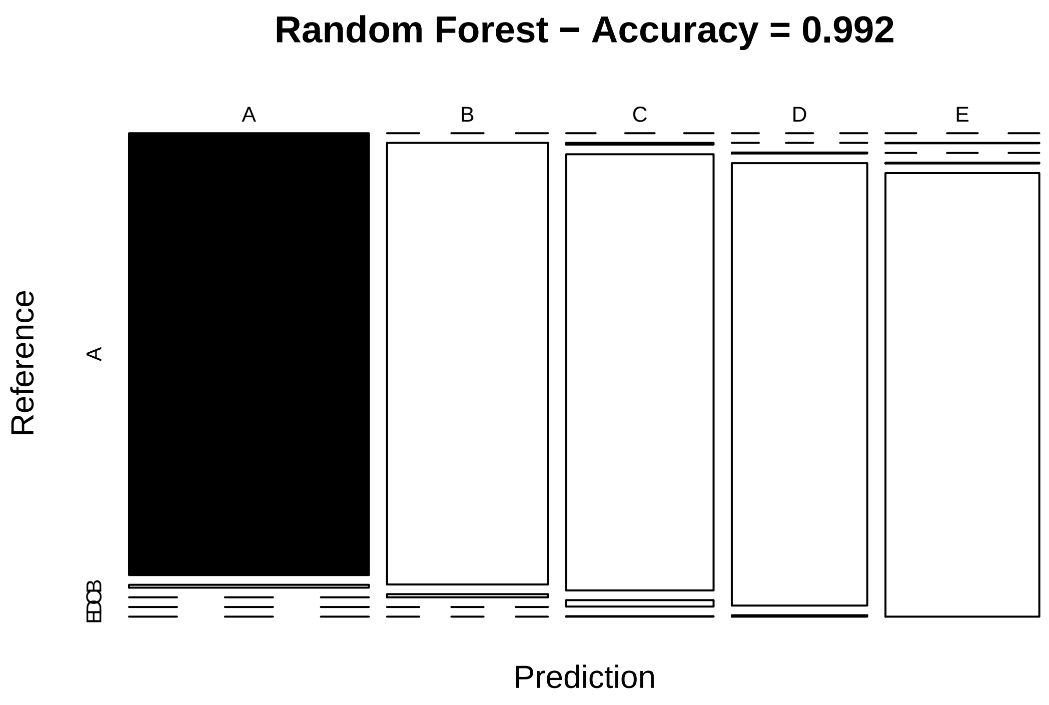
```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 26
##
##         OOB estimate of  error rate: 0.57%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3901    4    0    0    1 0.001280082
## B   21 2633    4    0    0 0.009405568
## C    0   13 2375    7    1 0.008764608
## D    0    0   17 2234    1 0.007992895
## E    0    0    4    5 2516 0.003564356
```

```r
# prediction on Test dataset
predict_RForest <- predict(Fitting_model_RF, newdata=Train_TestSet)
conf_Mat_RForest <- confusionMatrix(predict_RForest, Train_TestSet$classe)
conf_Mat_RForest
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674   11    0    0    0
##          B    0 1123    8    0    0
##          C    0    4 1016   15    1
##          D    0    0    2  947    3
##          E    0    1    0    2 1078
##
```

```
## Overall Statistics
##
##                Accuracy : 0.992
##                  95% CI : (0.9894, 0.9941)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9899
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9860   0.9903   0.9824   0.9963
## Specificity            0.9974   0.9983   0.9959   0.9990   0.9994
## Pos Pred Value         0.9935   0.9929   0.9807   0.9947   0.9972
## Neg Pred Value         1.0000   0.9966   0.9979   0.9966   0.9992
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1908   0.1726   0.1609   0.1832
## Detection Prevalence   0.2863   0.1922   0.1760   0.1618   0.1837
## Balanced Accuracy      0.9987   0.9921   0.9931   0.9907   0.9978
```

```
# plot matrix results
plot(conf_Mat_RForest$table, col = conf_Mat_RForest$byClass,
     main = paste("Random Forest - Accuracy =",
                  round(conf_Mat_RForest$overall['Accuracy'], 4)))
```



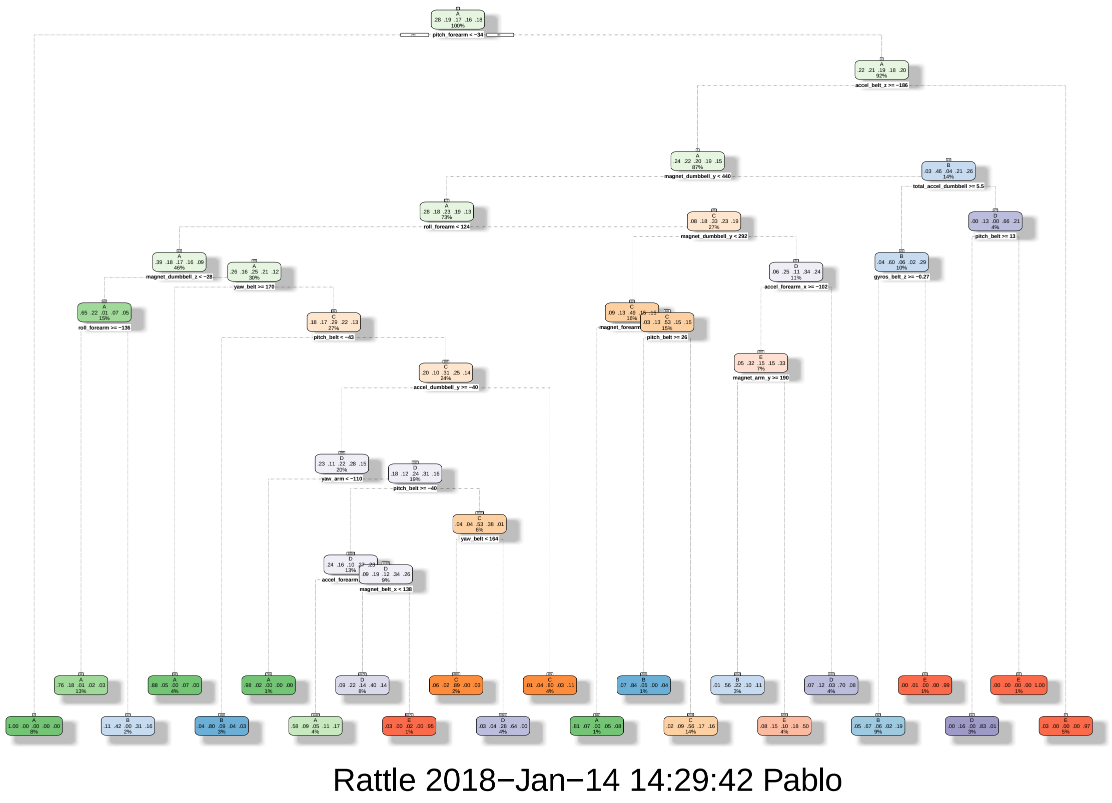
From the confusion matrix result, this model is getting an accuracy of 0.9913, meaning the out of sample error is .0087 which is a very good mark. To choose the best model it is needed to run the same process for the decision tree and compare both out of sample errors.

3.2 Decision Trees

Follows the prediction model using desicion trees.

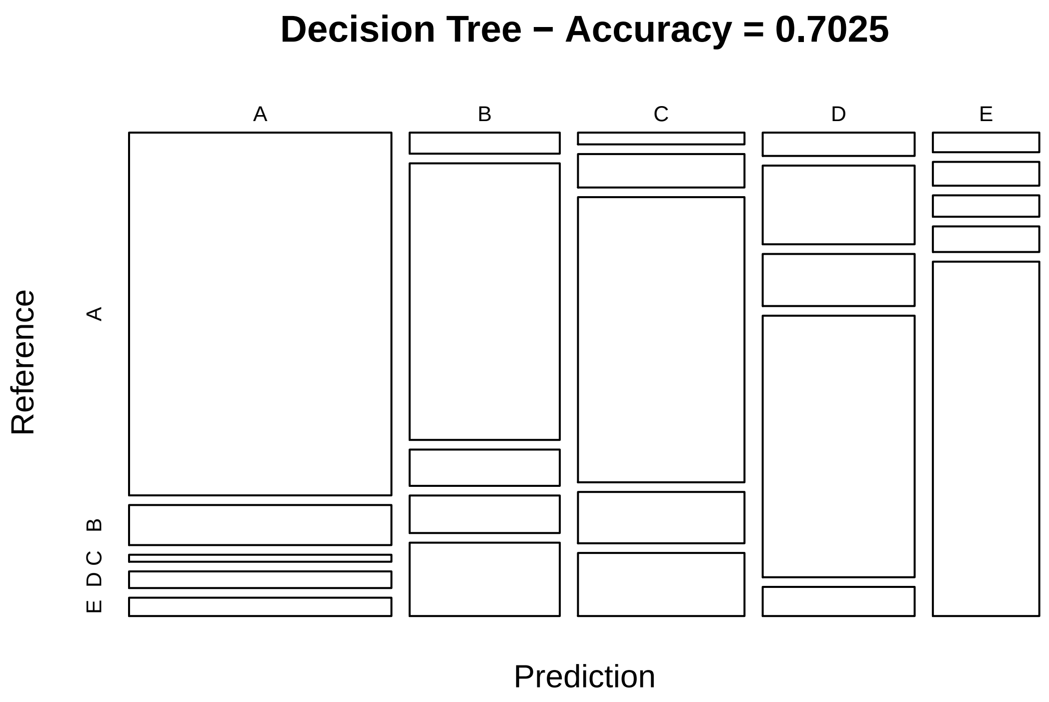
```r
# model fit
set.seed(98765)
Dec_Tree_Mod_Fit <- rpart(classe ~ ., data=Train_Set, method="class")
fancyRpartPlot(Dec_Tree_Mod_Fit)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



```r
# prediction on Test dataset
Predict_Dec_Tree <- predict(Dec_Tree_Mod_Fit, newdata=Train_TestSet, type="class")
conf_Mat_Dec_Tree <- confusionMatrix(Predict_Dec_Tree, Train_TestSet$classe)

# plot matrix results
plot(conf_Mat_Dec_Tree$table, col = conf_Mat_Dec_Tree$byClass,
     main = paste("Decision Tree - Accuracy =",
                  round(conf_Mat_Dec_Tree$overall['Accuracy'], 4)))
```



From the confusion matrix result, this model is getting an accuracy of 0.7011, meaning the out of sample error is .2989, even though it is not a too low value, it shows the prediction model accuracy is poor.

3. Conclusion

Based on the out of sample errors of both prediction models (ie. please keep in mind that the out of sample errors written may differ a little bit compared to those in the graphs as every single time the models are run the accuracy calculation changes a little bit), the random forest is by far more accurate at predicting the classe variable. These results are coherent with the common practice as random forest is normally one of the top performing algorithms, very accurate, even though it is at times difficult to interpret. After conducting the simulation, one can notice this model demands a lot of hardware resources as it takes a lot of time to perform the simulation. The only missing step to finish this study is to run the chosen model on the testing dataset to predict the classe variable on this set; it is done here after.

```r
Final_Prediction <- predict(Fitting_model_RF, newdata=testing)
Final_Prediction
```

```
##  [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
```

Follows the source for some of the comments made in the executive summary and for the dataset: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har.