Practical Machine Learing: Analysis of activity data

Ivana Seric

January 31, 2016

Executive Summary

This is a project for Coursera Practical Machine Learing course, by Johns Hopkins Bloomberg Shool of Public Health.

Background (Project Statement)

"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 (see the section on the Weight Lifting Exercise Dataset)."

```
options(scipen = 1, digits = 2)
knitr::opts_chunk$set(fig.width = 6, fig.height = 3, echo = TRUE, warning = FALSE,
                      message = FALSE, cache = TRUE)
library(ggplot2)
library(caret)
```

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

```
setwd('~/R/')
```

Reading and cleaning the data

```
trainingData <- read.csv("~/R/Machine_learning/pml-training.csv")
```

First thing to do is to create the partition for training and testing data.

```
# create partition for cross validation
set.seed(13)
inTrain <- createDataPartition(y = trainingData$classe, p = 0.75, list = FALSE)
training <- trainingData[inTrain,]
testing <- trainingData[-inTrain,]
```

By inspecting the data set, we can see that there are many variables with most NA values. There are 67 variables with more than 75% of NA values. Removing those values will make the prediction more accurate and the training algorithm will be faster.

```
naVars <- colnames(training)[colSums(is.na(training)) >= 0.75*length(training$X)]
# there are 67 of those variables -> remove them from the data
keepVars <- colSums(is.na(training)) < 0.75*length(training$X)
training1 <- training[keepVars]
# we are doum to 93 variables
```

There are still 93 predictors in the data set. To try to simplify it further, let's look for the preditors with near zero variance, and remove them.

```
dropNZV <- nearZeroVar(training1, saveMetrics = FALSE)
training2 <- training1[,-dropNZV ]
# now we are doum to 59 variables
```

The first 6 variables in the data set are name, raw time stamp variables and time stamp, and they will not be usefull for creating classification prediction.

```
training3 <- training2[,-(1:6)]
```

Fitting the model

Using parallel computation even with just 3 cores reduces the computation time significantly. Random forests can do a good prediction on a classification problem, so I will try it first. I will use the cross validation for resampling method with 10 folds.

```
library(parallel)
library(doParallel)
cluster <- makeCluster(3)
registerDoParallel(cluster)

fitControl <- trainControl(method = "cv",
                           number = 10,
                           allowParallel = TRUE)
timeStart <- proc.time()
fit <- train(classe~., method="rf", data=training3, trControl = fitControl)
proc.time() - timeStart
```

```
##    user  system elapsed
##   38.16    0.15  770.50
```

```
stopCluster(cluster)
```

```
fit
```

```
## Random Forest
##
## 14718 samples
##    52 predictors
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
```

```
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 13245, 13245, 13247, 13245, 13246, 13246, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa  Accuracy SD  Kappa SD
##    2    0.99      0.99   0.0024       0.0031
##   27    0.99      0.99   0.0025       0.0032
##   52    0.99      0.98   0.0051       0.0064
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
confusionMatrix.train(fit)
```

```
## Cross-Validated (10 fold) Confusion Matrix
##
## (entries are percentages of table totals)
##
##           Reference
## Prediction    A    B    C    D    E
##          A 28.4  0.1  0.0  0.0  0.0
##          B  0.0 19.2  0.1  0.0  0.0
##          C  0.0  0.0 17.3  0.3  0.0
##          D  0.0  0.0  0.0 16.1  0.1
##          E  0.0  0.0  0.0  0.0 18.3
```

The in-sample accuracy is 99%. Let's check how it performs on the test set. First I will do the same transformation to the data as was done to the training set.

```
testing1 <- testing[keepVars]
testing2 <- testing1[,-dropNZV ]
testing3 <- testing2[,-(1:6)]
testPart <- predict(fit, testing3)
confusionMatrix(testPart, testing3$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    3    0    0    0
##          B    0  943    8    0    0
##          C    0    3  847   11    0
##          D    0    0    0  793    1
##          E    0    0    0    0  900
##
## Overall Statistics
##
##                Accuracy : 0.995
##                  95% CI : (0.992, 0.997)
##     No Information Rate : 0.284
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.993
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity             1.000    0.994    0.991    0.986    0.999
## Specificity             0.999    0.998    0.997    1.000    1.000
## Pos Pred Value          0.998    0.992    0.984    0.999    1.000
## Neg Pred Value          1.000    0.998    0.998    0.997    1.000
## Prevalence              0.284    0.194    0.174    0.164    0.184
## Detection Rate          0.284    0.192    0.173    0.162    0.184
## Detection Prevalence    0.285    0.194    0.176    0.162    0.184
## Balanced Accuracy       1.000    0.996    0.994    0.993    0.999
```

The model performs very well on the testing set as well. Therefore, I will not try to fit any different model for this project.

```
testingData <- read.csv("~/R/Machine_learning/pml-testing.csv")
testingData1 <- testingData[keepVars]
testingData2 <- testingData1[,-dropNZV ]
testingData3 <- testingData2[,-(1:6)]
quizAns <- predict(fit, testingData)
```

This model gives a perfect score on the quiz tests. :)