Practical Machine Learning: Course Project

Ferran Martí

Monday, June 08, 2015

INTRODUCTION

There are certain devices that allow people to collect certain amount of data about personal activity. In this project, we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the manner in which they did the exercise. The training data for this project are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv The test data are available here: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

LOADING THE DATA

First, we load the data in the memory (assuming you have the csv files already in your computer, downloaded from the links provided above):

```
test.initial <- read.csv("C:/Users/ftorrent/Desktop/Data Science
Track1/Coursera/Practical Machine Learning/pml-testing.csv",
na.strings=c("NA","#DIV/0!",""))

train.initial <-read.csv("C:/Users/ftorrent/Desktop/Data Science
Track1/Coursera/Practical Machine Learning/pml-training.csv",
na.strings=c("NA","#DIV/0!",""))
```

Now, you need to load certain packages for the reproduction of the code to work:

```
## Loading required package: lattice
## Loading required package: ggplot2
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'dplyr'
##
## The following object is masked from 'package:randomForest':
##
##     combine
##
## The following object is masked from 'package:stats':
##
##     filter
##
## The following objects are masked from 'package:base':
##
```

```
##     intersect, setdiff, setequal, union
##
## Rattle: A free graphical interface for data mining with R.
## Versión 3.4.1 Copyright (c) 2006-2014 Togaware Pty Ltd.
## Escriba 'rattle()' para agitar, sacudir y  rotar sus datos.
```

CLEANING THE DATA

First, we remove columns that have NA values on them. As we can see, this way we reduce the amount of variables from 160 to 60, thus reducing the dataset (and the computation work) considerably.

```
train.initial <- train.initial[, colSums(is.na(train.initial)) == 0]
test.initial <- test.initial[, colSums(is.na(test.initial)) == 0]
```

Now, we delete those variables that are related to time (timestamp) as well as the X variable (which is the number of the row, basically) and the "window" variable, which is only a counting of the series and has nothing to do with the exercise per se:

```
test.clean<-select(test.initial, -X, -cvtd_timestamp, -
raw_timestamp_part_1, -raw_timestamp_part_2, -new_window, -num_window)
train.clean<-select(train.initial, -X, -cvtd_timestamp, -
raw_timestamp_part_1, -raw_timestamp_part_2, -new_window, -num_window)
```

So we have a training dataset of 19622 observations and 54 variables, and a test dataset of 20 observations and 54 variables as well.

The variable output is "classe", which has 5 levels: A, B, C, D and E.

SLICING THE DATA

Now we split the cleaned training set into a pure training data set (70%) and a validation data set (30%). We will use the validation data set to conduct cross validation later on. We also set up a set.seed for reproducible purposes:

```
set.seed(13475)
inTrain <- createDataPartition(train.clean$classe, p=0.70, list=FALSE)
training<-train.clean[inTrain, ]
testing<-train.clean[-inTrain, ]
```

DATA MODELLING

For this kind of datasets, I believe the best predictive algorithm to use is the Random Forest, as it automatically selects the most important variables, and is robust to outliers in general. I will use a 5-fold cross validation when applying the algorithm, as it is kind of the standard procedure.

```
controlRf<-trainControl(method="cv", 5)
modelRf <- train(classe ~ ., data=training, method="rf",
trControl=controlRf, ntree=250)
modelRf
```

```
## Random Forest
##
## 13737 samples
##    53 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
##
## Summary of sample sizes: 10991, 10989, 10989, 10989, 10990
##
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa      Accuracy SD  Kappa SD
##    2    0.9897357  0.9870147  0.001844989  0.002335123
##   29    0.9901724  0.9875673  0.001856243  0.002348430
##   57    0.9851496  0.9812114  0.001595062  0.002019386
##
## Accuracy was used to select the optimal model using  the largest
value.
## The final value used for the model was mtry = 29.
```

Now we have to estimate the performance of the algorithm in our validation data set.

```
validationRf<-predict(modelRf, testing)
confusionMatrix(testing$classe, validationRf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    0    0    0    0
##          B   10 1126    3    0    0
##          C    0    1 1022    3    0
##          D    0    0   14  948    2
##          E    0    0    0    0 1082
##
## Overall Statistics
##
##                Accuracy : 0.9944
##                  95% CI : (0.9921, 0.9961)
##     No Information Rate : 0.2862
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9929
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9941   0.9991   0.9836   0.9968   0.9982
```

```
## Specificity            1.0000   0.9973   0.9992   0.9968   1.0000
## Pos Pred Value         1.0000   0.9886   0.9961   0.9834   1.0000
## Neg Pred Value         0.9976   0.9998   0.9965   0.9994   0.9996
## Prevalence             0.2862   0.1915   0.1766   0.1616   0.1842
## Detection Rate         0.2845   0.1913   0.1737   0.1611   0.1839
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9970   0.9982   0.9914   0.9968   0.9991
```

Accuracy and out of sample error.

```
accuracy<-postResample(validationRf, testing$classe)
accuracy

##  Accuracy     Kappa
## 0.9943925 0.9929057

outofsampleerror <- 1 - as.numeric(confusionMatrix(testing$classe,
validationRf)$overall[1])
outofsampleerror

## [1] 0.005607477
```

Therefore, we have a 99.4% accuracy and 0.56% out of sample error.

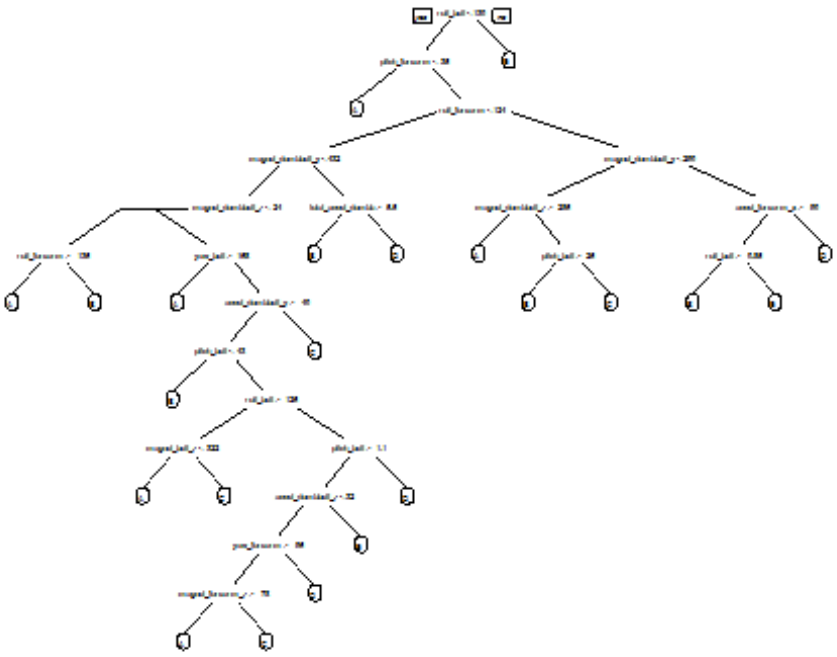
Prediction with the test dataset.

```
result <- predict(modelRf, test.clean[, -length(names(test.clean))])
result

##  [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
```

So the observations in the test dataset should be labelled as above.

```
treeModel <- rpart(classe ~ ., data=training, method="class")
prp(treeModel)
```



Finally, we have a fast plot of the algorithm used.