Human Activity Recognition

Csaba Farago

2017.04.23.

Overview

This is the solution of the assignment of Practical Machine Learning Courcera course. The assignment is described here: https://www.coursera.org/learn/practical-machine-learning/supplement/PvInj/course-project-instructions-read-first.

In a nuthsell: human activity recognicion device measures a great number of parameterers when doing one of the following exercises: sitting-down, standing-up, standing, walking, and sitting. The task is to make predictive models from the input data

Obtaining data

First we download the data manually. Then we read in and clean them.

Reading data

It is assumed that the data related to the task is found in the current directory. First we read them which comprises of a training and a testig data.

```
pml.training <- read.csv("pml-training.csv", na.strings = c("NA", "", "NULL", "#DIV/0!"))
pml.testing <- read.csv("pml-testing.csv", na.strings = c("NA", "", "NULL", "#DIV/0!"))
dim(pml.training)
```

```
## [1] 19622   160
```

```
dim(pml.testing)
```

```
## [1]  20 160
```

```
table(pml.training$classe)
```

```
##
##    A    B    C    D    E
## 5580 3797 3422 3216 3607
```

Cleaning data

As we see the number of dimensions of the data is 160. Many of the cointins NA data, furhtermore, some columns contain ordinal number, name or date related data which cannot be used for prediction. First we remove the not necessary columns.

```
columnsWithoutNA <- colSums(is.na(pml.training)) == 0
pml.training <- pml.training[, columnsWithoutNA]
names(pml.training)
```

```
##  [1] "X"                    "user_name"            "raw_timestamp_part_1"
##  [4] "raw_timestamp_part_2" "cvtd_timestamp"       "new_window"
##  [7] "num_window"           "roll_belt"            "pitch_belt"
## [10] "yaw_belt"             "total_accel_belt"     "gyros_belt_x"
## [13] "gyros_belt_y"         "gyros_belt_z"         "accel_belt_x"
## [16] "accel_belt_y"         "accel_belt_z"         "magnet_belt_x"
## [19] "magnet_belt_y"        "magnet_belt_z"        "roll_arm"
## [22] "pitch_arm"            "yaw_arm"              "total_accel_arm"
## [25] "gyros_arm_x"          "gyros_arm_y"          "gyros_arm_z"
## [28] "accel_arm_x"          "accel_arm_y"          "accel_arm_z"
## [31] "magnet_arm_x"         "magnet_arm_y"         "magnet_arm_z"
## [34] "roll_dumbbell"        "pitch_dumbbell"       "yaw_dumbbell"
## [37] "total_accel_dumbbell" "gyros_dumbbell_x"     "gyros_dumbbell_y"
## [40] "gyros_dumbbell_z"     "accel_dumbbell_x"     "accel_dumbbell_y"
## [43] "accel_dumbbell_z"     "magnet_dumbbell_x"    "magnet_dumbbell_y"
## [46] "magnet_dumbbell_z"    "roll_forearm"         "pitch_forearm"
## [49] "yaw_forearm"          "total_accel_forearm"  "gyros_forearm_x"
## [52] "gyros_forearm_y"      "gyros_forearm_z"      "accel_forearm_x"
## [55] "accel_forearm_y"      "accel_forearm_z"      "magnet_forearm_x"
## [58] "magnet_forearm_y"     "magnet_forearm_z"     "classe"
```

```
pml.training <- pml.training[, -c(1:7)]
pml.testing <- pml.testing[, columnsWithoutNA]
pml.testing <- pml.testing[, -c(1:7, 60)]
```

Creating prediction models

Now we build some prediction models. First we divide the training data into train and validation subparts.

```
library(caret)
```

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

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

```
set.seed(197777)
inTrain <- createDataPartition(pml.training$classe, p=0.75, list=FALSE)
pml.train <- pml.training[inTrain,]
pml.validation <- pml.training[-inTrain,]
```

We predict a non-numeric factor variable. For this we use 3 models for the prediction: random forest, decision tree and boosing.

Random forest

We perform random forest prediction directly with randomForest function, beacuse the train is too slow.

```
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
```

```
model.randomForest <- randomForest(classe ~ ., data = pml.train, importance = TRUE, ntree = 50)
predict.randomForest <- predict(model.randomForest, pml.validation)
confusionMatrix(predict.randomForest, pml.validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1394    2    0    0    0
##          B    0  947    7    0    1
##          C    0    0  846    4    1
##          D    1    0    2  799    0
##          E    0    0    0    1  899
##
## Overall Statistics
##
##                Accuracy : 0.9961
##                  95% CI : (0.994, 0.9977)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9951
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9993   0.9979   0.9895   0.9938   0.9978
## Specificity            0.9994   0.9980   0.9988   0.9993   0.9998
## Pos Pred Value         0.9986   0.9916   0.9941   0.9963   0.9989
## Neg Pred Value         0.9997   0.9995   0.9978   0.9988   0.9995
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2843   0.1931   0.1725   0.1629   0.1833
## Detection Prevalence   0.2847   0.1947   0.1735   0.1635   0.1835
## Balanced Accuracy      0.9994   0.9979   0.9941   0.9965   0.9988
```

As we can see, the overall accuracy of this prediction model is 0.9961256, whcih is quite high.

Decision tree

We also perform the decision tree analysis using the rpart function, also due to speed reasons.

```
library(rpart)
model.decisionTree <- rpart(classe ~ ., data = pml.train, method = "class")
predict.decisionTree <- predict(model.decisionTree, pml.validation, type="class")
confusionMatrix(predict.decisionTree, pml.validation$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
```

```
##          A 1270  225   19  100   28
##          B   28  430   26   18   33
##          C   34  162  752   81  125
##          D   54   60   57  529   68
##          E    9   72    1   76  647
##
## Overall Statistics
##
##                Accuracy : 0.7398
##                  95% CI : (0.7273, 0.752)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6693
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9104  0.45311   0.8795   0.6580   0.7181
## Specificity            0.8940  0.97345   0.9007   0.9417   0.9605
## Pos Pred Value         0.7734  0.80374   0.6516   0.6888   0.8037
## Neg Pred Value         0.9617  0.88121   0.9725   0.9335   0.9380
## Prevalence             0.2845  0.19352   0.1743   0.1639   0.1837
## Detection Rate         0.2590  0.08768   0.1533   0.1079   0.1319
## Detection Prevalence   0.3348  0.10909   0.2353   0.1566   0.1642
## Balanced Accuracy      0.9022  0.71328   0.8901   0.7998   0.8393
```

The overall accuracy of this model is not so good: 0.7398042.

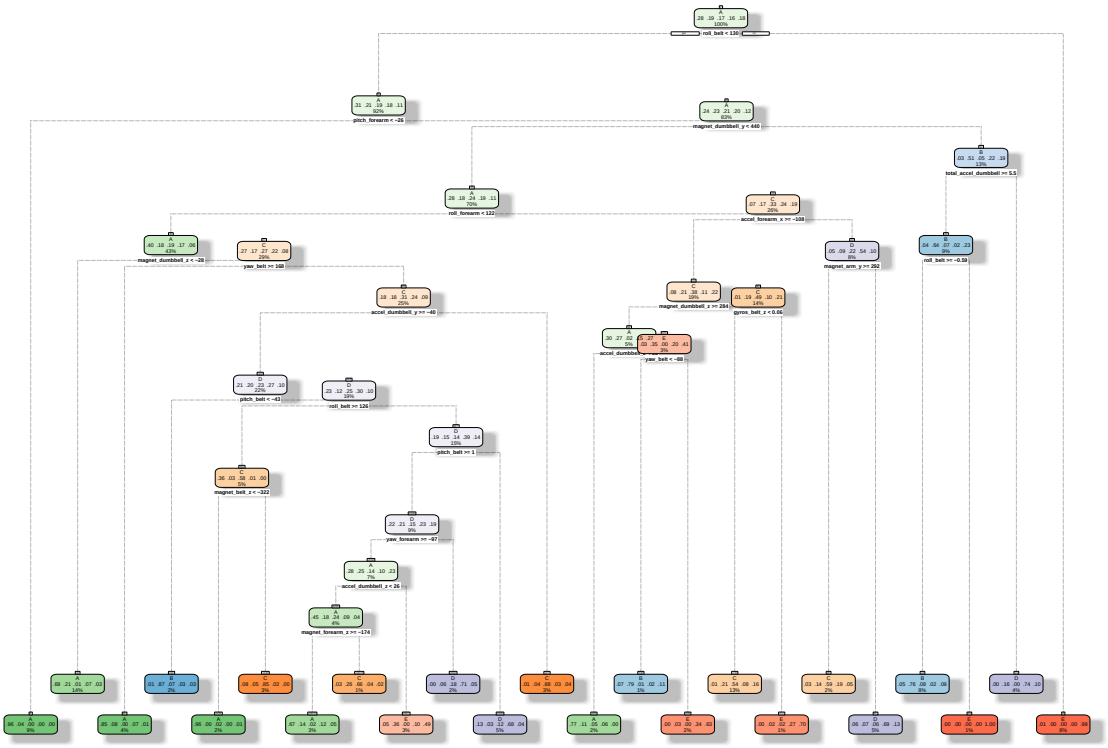
The resulting decision tree looks like this:

```
library(rattle)
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
fancyRpartPlot(model.decisionTree)
```

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



Rattle 2017−ápr.−23 20:34:01 U521684

Boosting

Finally let us check how boosting works:

```
library(gbm)
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: splines
```

```
## Loading required package: parallel
```

```
## Loaded gbm 2.1.3
```

```
model.boost <- train(classe ~ ., data = pml.train, method = "gbm", trControl = trainControl(method = "r
```

```
## Loading required package: plyr
```

```
predict.boost <- predict(model.boost, pml.validation)
confusionMatrix(predict.boost, pml.validation$classe)
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1369   27    0    0    2
##          B   16  892   21    2   12
##          C    6   27  830   29    9
##          D    4    0    4  768    8
##          E    0    3    0    5  870
##
## Overall Statistics
##
##                Accuracy : 0.9643
##                  95% CI : (0.9587, 0.9693)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9549
##  Mcnemar's Test P-Value : 1.356e-07
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9814   0.9399   0.9708   0.9552   0.9656
## Specificity            0.9917   0.9871   0.9825   0.9961   0.9980
## Pos Pred Value         0.9793   0.9459   0.9212   0.9796   0.9909
## Neg Pred Value         0.9926   0.9856   0.9938   0.9913   0.9923
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2792   0.1819   0.1692   0.1566   0.1774
## Detection Prevalence   0.2851   0.1923   0.1837   0.1599   0.1790
## Balanced Accuracy      0.9865   0.9635   0.9766   0.9757   0.9818
```

Between the above two models: 0.9643148.

Prediction test data

As we have the best prediction accuracy with random forest model, we use to predict it the testing data:

```
predict(model.randomForest, pml.testing)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  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
```