Coursera_Practical Machine Learning_Course Project

Niklas Pommer

13 Oktober 2017

Introduction

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: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har (http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

Data processing

Load libraries

```
library(caret);
library(rpart);
library(rpart.plot);
library(randomForest);
library(repmis);
```

Import the data

```
trainurl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testurl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
training <- source_data(trainurl, na.strings = c("NA", "#DIV/0!", ""), header = TRUE)
testing <- source_data(testurl, na.strings = c("NA", "#DIV/0!", ""), header = TRUE)
```

```
dim(training); dim(testing)
```

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

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

We have 19622 rows and 160 columns in the training data set and 20 rows and 160 columns in the testing data set. We use the testing data set in the end of the whole process. For a better understanding of the data we list all available columns.

```
names(training)
```

```
##   [1] "V1"                       "user_name"
##   [3] "raw_timestamp_part_1"     "raw_timestamp_part_2"
##   [5] "cvtd_timestamp"           "new_window"
##   [7] "num_window"               "roll_belt"
##   [9] "pitch_belt"               "yaw_belt"
##  [11] "total_accel_belt"         "kurtosis_roll_belt"
##  [13] "kurtosis_picth_belt"      "kurtosis_yaw_belt"
##  [15] "skewness_roll_belt"       "skewness_roll_belt.1"
##  [17] "skewness_yaw_belt"        "max_roll_belt"
##  [19] "max_picth_belt"           "max_yaw_belt"
##  [21] "min_roll_belt"            "min_pitch_belt"
##  [23] "min_yaw_belt"             "amplitude_roll_belt"
##  [25] "amplitude_pitch_belt"     "amplitude_yaw_belt"
##  [27] "var_total_accel_belt"     "avg_roll_belt"
##  [29] "stddev_roll_belt"         "var_roll_belt"
##  [31] "avg_pitch_belt"           "stddev_pitch_belt"
##  [33] "var_pitch_belt"           "avg_yaw_belt"
##  [35] "stddev_yaw_belt"          "var_yaw_belt"
##  [37] "gyros_belt_x"             "gyros_belt_y"
##  [39] "gyros_belt_z"             "accel_belt_x"
##  [41] "accel_belt_y"             "accel_belt_z"
##  [43] "magnet_belt_x"            "magnet_belt_y"
##  [45] "magnet_belt_z"            "roll_arm"
##  [47] "pitch_arm"                "yaw_arm"
##  [49] "total_accel_arm"          "var_accel_arm"
##  [51] "avg_roll_arm"             "stddev_roll_arm"
##  [53] "var_roll_arm"             "avg_pitch_arm"
##  [55] "stddev_pitch_arm"         "var_pitch_arm"
##  [57] "avg_yaw_arm"              "stddev_yaw_arm"
##  [59] "var_yaw_arm"              "gyros_arm_x"
##  [61] "gyros_arm_y"              "gyros_arm_z"
##  [63] "accel_arm_x"              "accel_arm_y"
##  [65] "accel_arm_z"              "magnet_arm_x"
##  [67] "magnet_arm_y"             "magnet_arm_z"
##  [69] "kurtosis_roll_arm"        "kurtosis_picth_arm"
##  [71] "kurtosis_yaw_arm"         "skewness_roll_arm"
##  [73] "skewness_pitch_arm"       "skewness_yaw_arm"
##  [75] "max_roll_arm"             "max_picth_arm"
##  [77] "max_yaw_arm"              "min_roll_arm"
##  [79] "min_pitch_arm"            "min_yaw_arm"
##  [81] "amplitude_roll_arm"       "amplitude_pitch_arm"
##  [83] "amplitude_yaw_arm"        "roll_dumbbell"
##  [85] "pitch_dumbbell"           "yaw_dumbbell"
##  [87] "kurtosis_roll_dumbbell"   "kurtosis_picth_dumbbell"
##  [89] "kurtosis_yaw_dumbbell"    "skewness_roll_dumbbell"
##  [91] "skewness_pitch_dumbbell"  "skewness_yaw_dumbbell"
##  [93] "max_roll_dumbbell"        "max_picth_dumbbell"
##  [95] "max_yaw_dumbbell"         "min_roll_dumbbell"
##  [97] "min_pitch_dumbbell"       "min_yaw_dumbbell"
##  [99] "amplitude_roll_dumbbell"  "amplitude_pitch_dumbbell"
## [101] "amplitude_yaw_dumbbell"   "total_accel_dumbbell"
## [103] "var_accel_dumbbell"       "avg_roll_dumbbell"
## [105] "stddev_roll_dumbbell"     "var_roll_dumbbell"
## [107] "avg_pitch_dumbbell"       "stddev_pitch_dumbbell"
## [109] "var_pitch_dumbbell"       "avg_yaw_dumbbell"
## [111] "stddev_yaw_dumbbell"      "var_yaw_dumbbell"
## [113] "gyros_dumbbell_x"         "gyros_dumbbell_y"
```

```
## [115] "gyros_dumbbell_z"          "accel_dumbbell_x"
## [117] "accel_dumbbell_y"          "accel_dumbbell_z"
## [119] "magnet_dumbbell_x"         "magnet_dumbbell_y"
## [121] "magnet_dumbbell_z"         "roll_forearm"
## [123] "pitch_forearm"             "yaw_forearm"
## [125] "kurtosis_roll_forearm"     "kurtosis_picth_forearm"
## [127] "kurtosis_yaw_forearm"      "skewness_roll_forearm"
## [129] "skewness_pitch_forearm"    "skewness_yaw_forearm"
## [131] "max_roll_forearm"          "max_picth_forearm"
## [133] "max_yaw_forearm"           "min_roll_forearm"
## [135] "min_pitch_forearm"         "min_yaw_forearm"
## [137] "amplitude_roll_forearm"    "amplitude_pitch_forearm"
## [139] "amplitude_yaw_forearm"     "total_accel_forearm"
## [141] "var_accel_forearm"         "avg_roll_forearm"
## [143] "stddev_roll_forearm"       "var_roll_forearm"
## [145] "avg_pitch_forearm"         "stddev_pitch_forearm"
## [147] "var_pitch_forearm"         "avg_yaw_forearm"
## [149] "stddev_yaw_forearm"        "var_yaw_forearm"
## [151] "gyros_forearm_x"           "gyros_forearm_y"
## [153] "gyros_forearm_z"           "accel_forearm_x"
## [155] "accel_forearm_y"           "accel_forearm_z"
## [157] "magnet_forearm_x"          "magnet_forearm_y"
## [159] "magnet_forearm_z"          "classe"
```

We can see here that the first 7 columns are not usable for our prediction. We skip them later.

Cleaning the data

We have to exclude columns with NaNs.

```
training <- training[, colSums(is.na(training)) == 0]
testing <- testing[, colSums(is.na(testing)) == 0]
```

Did it work? –> Test for NaN

```
training[,!complete.cases(training)]
```

```
## data frame with 0 columns and 19622 rows
```

Yes :-)

Splitting the data

```
trainData <- training[, -c(1:7)]
testData <- testing[, -c(1:7)]
```

One important part of the prediction process is cross-validation. Here we divide the trainings data in a train data set and a validation data set.

```
set.seed(4444)
inTrain <- createDataPartition(trainData$classe, p = 0.7, list = FALSE)
train <- trainData[inTrain, ]
valid <- trainData[-inTrain, ]
```

Prediction with Decision Trees

Decision Trees

We integrate a k-fold cross-validation. “k-fold” means that the training set is split into k smaller sets. The following procedure is followed for each of the k folds: 1. A model is trained using k-1 of the folds as trainin data 2. The resulting model is validated on the remaining part of the data The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop

(For more information see here: https://medium.com/towards-data-science/train-test-split-and-cross-validation-in-python-80b61beca4b6 (https://medium.com/towards-data-science/train-test-split-and-cross-validation-in-python-80b61beca4b6))

We define 5 folds for the cross-validation process

```
control <- trainControl(method = "cv", number = 5)
fit_rpart <- train(classe ~ ., data = train, method = "rpart", trControl = control)
print(fit_rpart, digits = 4)
```

```
## CART
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10990, 10991, 10988, 10991, 10988
## Resampling results across tuning parameters:
##
##   cp       Accuracy  Kappa
##   0.03418  0.5258    0.39003
##   0.05981  0.4164    0.20900
##   0.11698  0.3331    0.07427
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.03418.
```

We now predict the outcome and use the validation data set. Then we print the result and the accuracy.

```
predict_rpart <- predict(fit_rpart, valid)
conf_rpart <- confusionMatrix(valid$classe, predict_rpart)
conf_rpart
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1521   26  124    0    3
##          B  466  396  277    0    0
##          C  476   29  521    0    0
##          D  412  178  374    0    0
##          E  163  160  289    0  470
##
## Overall Statistics
##
##                Accuracy : 0.4941
##                  95% CI : (0.4813, 0.507)
##     No Information Rate : 0.5162
##     P-Value [Acc > NIR] : 0.9997
##
##                   Kappa : 0.3392
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.5007  0.50190  0.32871       NA  0.99366
## Specificity            0.9463  0.85420  0.88256   0.8362  0.88692
## Pos Pred Value         0.9086  0.34767  0.50780       NA  0.43438
## Neg Pred Value         0.6398  0.91719  0.78102       NA  0.99938
## Prevalence             0.5162  0.13407  0.26933   0.0000  0.08037
## Detection Rate         0.2585  0.06729  0.08853   0.0000  0.07986
## Detection Prevalence   0.2845  0.19354  0.17434   0.1638  0.18386
## Balanced Accuracy      0.7235  0.67805  0.60563       NA  0.94029
```

```
accuracy_rpart <- conf_rpart$overall[1]
accuracy_rpart
```

```
##  Accuracy
## 0.4941376
```

The accuracy is 0.4941376.

Random Forests

```
fit_rf <- train(classe ~ ., data = train, method = "rf", trControl = control, ntree = 250)
print(fit_rf, digits = 4)
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10991, 10989, 10990, 10990, 10988
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.9919    0.9898
##   27    0.9905    0.9880
##   52    0.9852    0.9813
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
predict_rf <- predict(fit_rf, valid)
# Show prediction result
conf_rf <- confusionMatrix(valid$classe, predict_rf)
conf_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1673    0    0    0    1
##          B   11 1125    3    0    0
##          C    0   14 1010    2    0
##          D    0    0   16  948    0
##          E    0    0    0    2 1080
##
## Overall Statistics
##
##                Accuracy : 0.9917
##                  95% CI : (0.989, 0.9938)
##     No Information Rate : 0.2862
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9895
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9935   0.9877   0.9815   0.9958   0.9991
## Specificity            0.9998   0.9971   0.9967   0.9968   0.9996
## Pos Pred Value         0.9994   0.9877   0.9844   0.9834   0.9982
## Neg Pred Value         0.9974   0.9971   0.9961   0.9992   0.9998
## Prevalence             0.2862   0.1935   0.1749   0.1618   0.1837
## Detection Rate         0.2843   0.1912   0.1716   0.1611   0.1835
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9966   0.9924   0.9891   0.9963   0.9993
```

```
accuracy_rf <- conf_rf$overall[1]
accuracy_rf
```

```
##  Accuracy
## 0.9916737
```

The accuracy is 0.9916737 which is better than the score of the decision tree model.The out-of-sample error is 1-0.9916737 = 0.0083263 which is very good.

Prediction of our test data set

We now use the random forest model to predict the outcome of our test data set which is also used for quiz 4.

```
predict(fit_rf, testData)
```

```
##  [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
```