Practical Machine Learning Assignment

Sabine, 17 November 2017

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 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://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset). **The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set.**

1) Loading the data

First we read the test and the quiz data:

```
PML <- read.csv(file="C:/Users/Sabine/Documents/R/Course8 /pml_training.csv")
PML_Quiz  <- read.csv(file="C:/Users/Sabine/Documents/R/Course8/pml-testing.csv")
```

2) shrink dataset

Next we replace all empty fields with NA. Then We elimiate all Columns that contain mostly NA. We also remove timestamps, username and window. We do that for test and quiz data. Then we cut the training data into 2 parts using createDataPartition():

```
PML_Quiz[PML_Quiz == ""] <- NA
PML_Quiz_reduced<- PML_Quiz[, (colSums(is.na(PML_Quiz)) <20)]
PML_Quiz_red <-subset( PML_Quiz_reduced,select  = -c(raw_timestamp_part_2,cvtd_timestam
p,X, user_name,new_window,num_window))

PML[PML == ""] <- NA
PML_reduced<- PML[, (colSums(is.na(PML)) < 19215)]
PML_red<-subset( PML_reduced,select = -c(raw_timestamp_part_2,cvtd_timestamp,X, user_na
me,new_window,num_window))

inTrain<-createDataPartition(y=PML_red$classe, p=0.40, list=FALSE)
PML_training  <-PML_red[inTrain,]
PML_testing   <-PML_red[-inTrain,]
```

3) Now we fit two models: a Tree-model and a randomForest-model

```
modFit_Tree <- rpart(classe ~ ., data=PML_training, method="class")   # tree
modFit_Tree
```

```
## n= 7850
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##    1) root 7850 5618 A (0.28 0.19 0.17 0.16 0.18)
##      2) roll_belt< 129.5 7144 4939 A (0.31 0.21 0.19 0.18 0.11)
##        4) pitch_forearm< -33.95 628    6 A (0.99 0.0096 0 0 0) *
##        5) pitch_forearm>=-33.95 6516 4933 A (0.24 0.23 0.21 0.2 0.12)
##         10) yaw_belt>=168.5 383   55 A (0.86 0.063 0 0.07 0.01) *
##         11) yaw_belt< 168.5 6133 4644 B (0.2 0.24 0.22 0.21 0.12)
##           22) magnet_dumbbell_z< -22.5 2228 1409 A (0.37 0.29 0.087 0.21 0.048)
##             44) raw_timestamp_part_1< 1.322838e+09 490   29 A (0.94 0.053 0.0061 0 0
) *
##             45) raw_timestamp_part_1>=1.322838e+09 1738 1120 B (0.21 0.36 0.11 0.27
0.061)
##               90) raw_timestamp_part_1< 1.322838e+09 299    0 B (0 1 0 0 0) *
##               91) raw_timestamp_part_1>=1.322838e+09 1439  974 D (0.25 0.22 0.13 0.3
2 0.074)
##                182) accel_forearm_x>=-189.5 1055  725 A (0.31 0.3 0.16 0.15 0.071)
##                  364) raw_timestamp_part_1< 1.323095e+09 770  440 A (0.43 0.41 0.025
0.069 0.068)
##                    728) raw_timestamp_part_1< 1.323095e+09 573  243 A (0.58 0.21 0.0
26 0.092 0.091)
##                     1456) yaw_dumbbell>=49.58621 317   29 A (0.91 0.057 0 0.019 0.01
6) *
##                     1457) yaw_dumbbell< 49.58621 256  151 B (0.16 0.41 0.059 0.18 0.
18) *
##                    729) raw_timestamp_part_1>=1.323095e+09 197    4 B (0 0.98 0.02 0
0) *
##                  365) raw_timestamp_part_1>=1.323095e+09 285  132 C (0 0 0.54 0.38 0
.081)
##                    730) raw_timestamp_part_1< 1.323095e+09 151    0 C (0 0 1 0 0) *
##                    731) raw_timestamp_part_1>=1.323095e+09 134   25 D (0 0 0.015 0.8
1 0.17) *
##                183) accel_forearm_x< -189.5 384   81 D (0.073 0.0078 0.049 0.79 0.08
1) *
##           23) magnet_dumbbell_z>=-22.5 3905 2730 C (0.11 0.22 0.3 0.2 0.17)
##             46) raw_timestamp_part_1< 1.32249e+09 214    0 A (1 0 0 0 0) *
##             47) raw_timestamp_part_1>=1.32249e+09 3691 2516 C (0.06 0.23 0.32 0.22 0
.18)
##               94) magnet_dumbbell_x< -446.5 2603 1496 C (0.075 0.15 0.43 0.24 0.11)
##                188) roll_dumbbell< -38.14257 592  127 C (0.0017 0.1 0.79 0.071 0.041
) *
##                189) roll_dumbbell>=-38.14257 2011 1369 C (0.096 0.16 0.32 0.29 0.14)
##                  378) raw_timestamp_part_1< 1.322833e+09 1517  901 C (0.12 0.17 0.41
0.17 0.13)
##                    756) raw_timestamp_part_1>=1.322833e+09 272    0 C (0 0 1 0 0) *
##                    757) raw_timestamp_part_1< 1.322833e+09 1245  901 C (0.14 0.21 0.
28 0.2 0.16)
##                     1514) raw_timestamp_part_1>=1.322833e+09 175    0 B (0 1 0 0 0)
*
##                     1515) raw_timestamp_part_1< 1.322833e+09 1070  726 C (0.17 0.083
0.32 0.24 0.19)
##                       3030) raw_timestamp_part_1>=1.322673e+09 284  132 E (0.46 0 0
0.0035 0.54)
##                         6060) raw_timestamp_part_1>=1.322753e+09 131    0 A (1 0 0 0
```

```
0) *
##                           6061) raw_timestamp_part_1< 1.322753e+09 153    1 E (0 0 0 0
.0065 0.99) *
##                         3031) raw_timestamp_part_1< 1.322673e+09 786  442 C (0.062 0.1
1 0.44 0.32 0.064)
##                           6062) raw_timestamp_part_1< 1.322673e+09 641  297 C (0.076 0
.14 0.54 0.17 0.078) *
##                           6063) raw_timestamp_part_1>=1.322673e+09 145    0 D (0 0 0 1
0) *
##                    379) raw_timestamp_part_1>=1.322833e+09 494  166 D (0.026 0.12 0.05
3 0.66 0.14)
##                      758) raw_timestamp_part_1< 1.323084e+09 432  104 D (0.03 0.13 0.0
6 0.76 0.019) *
##                      759) raw_timestamp_part_1>=1.323084e+09 62    0 E (0 0 0 0 1) *
##                 95) magnet_dumbbell_x>=-446.5 1088  624 B (0.026 0.43 0.062 0.16 0.33)
##                  190) raw_timestamp_part_1< 1.32249e+09 193    5 B (0.021 0.97 0.0052
0 0) *
##                  191) raw_timestamp_part_1>=1.32249e+09 895  537 E (0.027 0.31 0.075 0
.19 0.4)
##                    382) raw_timestamp_part_1>=1.32249e+09 772  414 E (0.031 0.36 0.08
0.067 0.46)
##                      764) accel_dumbbell_z< 35.5 464  196 B (0 0.58 0.11 0.11 0.2) *
##                      765) accel_dumbbell_z>=35.5 308   44 E (0.078 0.026 0.036 0.0032
0.86) *
##                    383) raw_timestamp_part_1< 1.32249e+09 123    5 D (0 0 0.041 0.96 0
) *
##      3) roll_belt>=129.5 706   27 E (0.038 0 0 0 0.96) *
```

```r
modFit_RF <- train(classe ~ ., data=PML_training, method="rf", prox=TRUE)    # rforest
modFit_RF
```

```
## Random Forest
## 7850 samples
##   53 predictor
##    5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 7850, 7850, 7850, 7850, 7850, 7850, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9851660  0.9812299
##   27    0.9927127  0.9907795
##   53    0.9890675  0.9861666
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 27.
```

4) Predicting of test-data

Now we use the predict-Function with our 2 models and the PML_testing-Data to predict PML_Testing.

```r
PredictTree<-predict(modFit_Tree, newdata= PML_testing, type="class")
PredictnRF<-predict(modFit_RF, newdata= PML_testing, type="raw")
```

5) comparison of the models

We compare values calculated by predicting our models to the values of PML_testing. To Do this we use confusionMatrix():

```
Cm_tree<-confusionMatrix(PredictTree, PML_testing$classe) cmTree
cm_tree
cm_rf<- confusionMatrix (modFit_RF, PML_testing$classe)
cm_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3062  112    5   33   17
##          B   61 1822  126  179  212
##          C   85  226 1839  221  102
##          D   67   95   64 1490   80
##          E   73   23   19    6 1753
##
## Overall Statistics
##
##                Accuracy : 0.8466
##                  95% CI : (0.8399, 0.8531)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8062
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9146   0.7998   0.8958   0.7724   0.8101
## Specificity            0.9802   0.9391   0.9348   0.9689   0.9874
## Pos Pred Value         0.9483   0.7592   0.7436   0.8296   0.9354
## Neg Pred Value         0.9665   0.9513   0.9770   0.9560   0.9585
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2601   0.1548   0.1562   0.1266   0.1489
## Detection Prevalence   0.2743   0.2039   0.2101   0.1526   0.1592
## Balanced Accuracy      0.9474   0.8695   0.9153   0.8707   0.8987

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3344    8    0    0    0
##          B    4 2259   14    0    0
##          C    0   11 2035   11    0
##          D    0    0    4 1916    1
##          E    0    0    0    2 2163
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9953
##                  95% CI : (0.9939, 0.9965)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9941
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9917   0.9912   0.9933   0.9995
## Specificity            0.9991   0.9981   0.9977   0.9995   0.9998
## Pos Pred Value         0.9976   0.9921   0.9893   0.9974   0.9991
## Neg Pred Value         0.9995   0.9980   0.9981   0.9987   0.9999
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2841   0.1919   0.1729   0.1628   0.1837
## Detection Prevalence   0.2847   0.1934   0.1747   0.1632   0.1839
## Balanced Accuracy      0.9989   0.9949   0.9945   0.9964   0.9997
```

By calculating the confusionMatrix we see that the accuracy of the prediction model "randomForest" (99,6%) is much higher than the accuracy of the prediction model "tree" (84,3%). So we conclude that the prediction model "randomForest" is the best to use.

6) Doing the Quiz

Now we use the 2 prediction models to predict 20 different test cases.

```
QuizWithTree<-predict(modFit_Tree, newdata= PML_Quiz_red, type="class")
QuizWithTree

##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  B  A  C  A  A  E  D  C  A  A  B  C  B  A  E  E  E  B  B  B
## Levels: A B C D E

QuizWithRF<-predict(modFit_RF, newdata= PML_Quiz_red, type="raw")
QuizWithRF

##  [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
```

7) Conclusion

Earlier we saw that the prediction model "randomForest" is the best to use. The same result was obtained when doing the Quiz. When we did the quiz with the data we obtained by the tree-model we got 13/20 right answers which means the quiz was 65% correct. When we did the quiz with the data we obtained by the RandomForest-model we got 20/20 right answers (100%) correct. This shows that indeed the random forest-model is the best model