MachineLearning

M. Alfonso López

Friday, June 19, 2015

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

Loading data:

```
urlTr="https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
download.file(urlTr,destfile="training.csv")
urlTest="https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
download.file(urlTest,destfile="testing.csv")
```

Reading data

```
trainingData<-read.csv('training.csv',na.strings=c("#DIV/0!"))
testData<-read.csv('testing.csv',na.strings=c("#DIV/0!"))
```

The training data has 19622 observations and 160 features, while the test data has 20 observations and the same number of features:

```
dim(trainingData)
```

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

```
dim(testData)
```

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

Data Preprocessing

```r
library(caret)
```

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

```r
featureSet <- colnames(trainingData[colSums(is.na(trainingData)) == 0])[-(1:7)]
data <- trainingData[featureSet]

nzv <- nearZeroVar(data) #Identification of near zero variance predictors
data <- data[,-nzv]
```

Data splitting

We are going to split the data in a 80% training set and a 20% validation set.

```r
set.seed(7484479) #For reproduciblitity
intrain <- createDataPartition(data$classe, p = 0.8, list = FALSE)
training <- data[intrain, ]
validation <- data[-intrain, ]
```

Data Modeling

We are going to use a predictive model for activity recognition based on the construction of a tree.

```r
library(rpart)

modelRf <- train(classe ~ ., data=training,method="rpart")
```

Now we predict this modeling on validation data set

```r
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.1.3
```

```
## 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.
```

```r
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.1.3
```

```
## Loading required package: survival
## Loading required package: splines
##
## Attaching package: 'survival'
```

```
##
## The following object is masked from 'package:caret':
##
##     cluster
##
## Loading required package: parallel
## Loaded gbm 2.1.1
```

```r
modelRf
```

```
## CART
##
## 15699 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 15699, 15699, 15699, 15699, 15699, 15699, ...
##
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa       Accuracy SD  Kappa SD
##   0.03604806  0.5020283  0.34751334  0.03284918   0.05379113
##   0.06070316  0.4126128  0.20224584  0.06469677   0.10596248
##   0.11579884  0.3439393  0.08928441  0.03751081   0.05689692
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.03604806.
```

```r
predictRf <- predict(modelRf, validation)
```

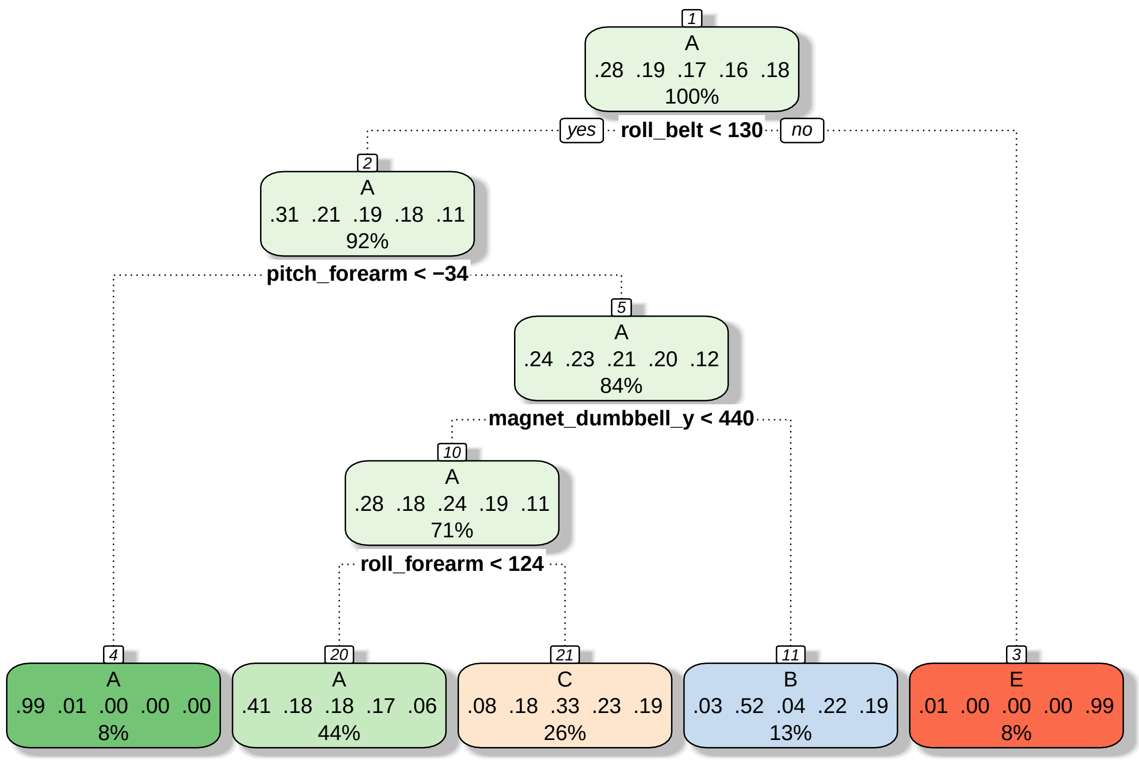
```
## Loading required package: rpart
```

```r
confusionMatrix(validation$classe, predictRf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1005   19   89    0    3
##          B  332  237  190    0    0
##          C  307   23  354    0    0
##          D  276  119  248    0    0
##          E  111  110  181    0  319
##
## Overall Statistics
##
##                Accuracy : 0.4881
##                  95% CI : (0.4724, 0.5039)
##     No Information Rate : 0.5177
```

```
##     P-Value [Acc > NIR] : 0.9999
##
##                   Kappa : 0.3312
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4948  0.46654  0.33333       NA  0.99068
## Specificity            0.9413  0.84714  0.88466   0.8361  0.88836
## Pos Pred Value         0.9005  0.31225  0.51754       NA  0.44244
## Neg Pred Value         0.6345  0.91435  0.78141       NA  0.99906
## Prevalence             0.5177  0.12949  0.27071   0.0000  0.08208
## Detection Rate         0.2562  0.06041  0.09024   0.0000  0.08132
## Detection Prevalence   0.2845  0.19347  0.17436   0.1639  0.18379
## Balanced Accuracy      0.7181  0.65684  0.60899       NA  0.93952
```

```r
fancyRpartPlot(modelRf$finalModel)
```



Rattle 2015−jun−21 19:47:02 Computaex

```r
accuracy <- postResample(predictRf, validation$classe)
error <- 1 - as.numeric(confusionMatrix(validation$classe, predictRf)$overall[1])
accuracy
```

```
##  Accuracy     Kappa
## 0.4881468 0.3312458
```

```r
error
```

```
## [1] 0.5118532
```

As we see, the accuracy is 0.48 which is clearly low, and the error 0.51.

On the other hand, we are going to create a new modeling by using boosting with trees. It is expected to get a greater accuracy, because the algorithm weights possibly weak predictors in order to get stronger ones.

```r
modelRf2 <- train(classe ~ ., data=training, method="gbm", verbose=FALSE)
```

```r
modelRf2
```

```
## Stochastic Gradient Boosting
##
## 15699 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
##
## Summary of sample sizes: 15699, 15699, 15699, 15699, 15699, 15699, ...
##
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa      Accuracy SD
##   1                   50      0.7534225  0.6873171  0.007382658
##   1                  100      0.8198346  0.7718837  0.004684433
##   1                  150      0.8537332  0.8148597  0.003721432
##   2                   50      0.8540216  0.8149665  0.004302879
##   2                  100      0.9065755  0.8817186  0.004327577
##   2                  150      0.9302886  0.9117507  0.004200362
##   3                   50      0.8965095  0.8689216  0.004568141
##   3                  100      0.9405008  0.9246824  0.003772164
##   3                  150      0.9590956  0.9482340  0.003021331
##   Kappa SD
##   0.009181201
##   0.005848630
##   0.004644015
##   0.005454223
##   0.005511582
##   0.005319688
##   0.005772901
##   0.004774568
##   0.003823240
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
## Accuracy was used to select the optimal model using  the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3 and shrinkage = 0.1.
```

```r
predictRf2 <- predict(modelRf2, validation)
```

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

```r
confusionMatrix(validation$classe, predictRf2)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1085   18    8    3    2
##          B   26  709   22    2    0
##          C    0   34  643    6    1
##          D    0    3   19  619    2
##          E    2   15    6   11  687
##
## Overall Statistics
##
##                Accuracy : 0.9541
##                  95% CI : (0.9471, 0.9605)
##     No Information Rate : 0.2837
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.942
##  Mcnemar's Test P-Value : 1.032e-06
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9748   0.9101   0.9212   0.9657   0.9928
## Specificity            0.9890   0.9841   0.9873   0.9927   0.9895
## Pos Pred Value         0.9722   0.9341   0.9401   0.9627   0.9528
## Neg Pred Value         0.9900   0.9779   0.9830   0.9933   0.9984
## Prevalence             0.2837   0.1986   0.1779   0.1634   0.1764
## Detection Rate         0.2766   0.1807   0.1639   0.1578   0.1751
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9819   0.9471   0.9542   0.9792   0.9911
```

```r
accuracy2 <- postResample(predictRf2, validation$classe)
error2 <- 1 - as.numeric(confusionMatrix(validation$classe, predictRf2)$overall[1])
accuracy2
```

```
##  Accuracy     Kappa
## 0.9541167 0.9419676
```

```r
error2
```

```
## [1] 0.04588325
```

In this case, the accuracy is 0,95 and the error only 0,04.

Conclusions

The confusion matrix of the second model shows a very accurate model due to the very low accuracy.

Test Data

```
pml_write_files = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("problem_id_",i,".txt")
    write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
  }
}

x <- testData
x <- x[featureSet[featureSet!='classe']]
answers <- predict(modelRf2, newdata=x)

answers
```

```
##  [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
```

```
pml_write_files(answers)
```