Practical Machine Learning Project

githubaliba

Sunday, November 23, 2014

Practical Machine Learning : Peer Assessment

Assignment: 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. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

The first thing we will do is set the seed to ensure reproducible results.

```
set.seed(2112)
```

Next, we will create the data directory and download the data files as needed. We will also load the raw “trainData” data frame and get a quick summary of it.

```
dataDir <- "./data"
if(!file.exists(dataDir)) {dir.create(dataDir)}
trainFile.url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
trainFile.name <- "pml-training.csv"
trainFile.dpath <- paste(dataDir,"/",trainFile.name,sep="")
download.file(trainFile.url,destfile=trainFile.dpath,method="curl")
testFile.url <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
testFile.name <- "pml-testing.csv"
testFile.dpath <- paste(dataDir,"/",testFile.name,sep="")
download.file(testFile.url,destfile=testFile.dpath,method="curl")

trainData <- read.csv(trainFile.dpath)
#summary(trainData)
```

The summary for this raw data has many NAs and other bad values. There are a variety of ways to address this. First we will use the nearZeroVar function from the CARET package to remove values that have no variance. Then, after further inspection, it would appear that there are a variety of summary values (min, max, var, stddev, etc) that have large number of NAs and that we can exclude as the information is already captured in the raw data. Reviewing the summaries as we trim out the variables shows we have a much cleaner and more information dense data set to work with.

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.1.2
```

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

```
zeroCols <- nearZeroVar(trainData)
clipData <- trainData[, -zeroCols]
#summary(clipData)

trimData <- clipData[ , -grep("max_",names(clipData))]
trimData <- trimData[ , -grep("min_",names(trimData))]
trimData <- trimData[ , -grep("var_",names(trimData))]
trimData <- trimData[ , -grep("amplitude_",names(trimData))]
trimData <- trimData[ , -grep("stddev_",names(trimData))]
trimData <- trimData[ , -grep("avg_",names(trimData))]
#summary(trimData)
```

The next thing to do is to subset this cleaned up data into training and test data that we set aside for final validation.

```
inTrain <- createDataPartition(y=trimData$classe, p=0.7, list=FALSE)
trainSet <- trimData[inTrain,-c(1:6)]
testSet <- trimData[-inTrain,-c(1:6)]
```

Before diving in and working with this full data set, however, it seemed more prudent and time-sensitive to work with some models based on an experimental subset of data. I shave off a random section of the data to work with here.

```
exSet1 <- trainSet[sample(1:nrow(trainSet), 1000,replace=FALSE),]
inTrainEx1 <- createDataPartition(y=exSet1$classe, p=0.7, list=FALSE)
trainEx1 <- exSet1[inTrainEx1,]
testEx1 <- exSet1[-inTrainEx1,]
```

First, I take a look at a quick rpart / classification tree.

```
exSet1Modelrpart <- train(classe ~., method="rpart",data=trainEx1)
```

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

```
print(exSet1Modelrpart$finalModel)
```

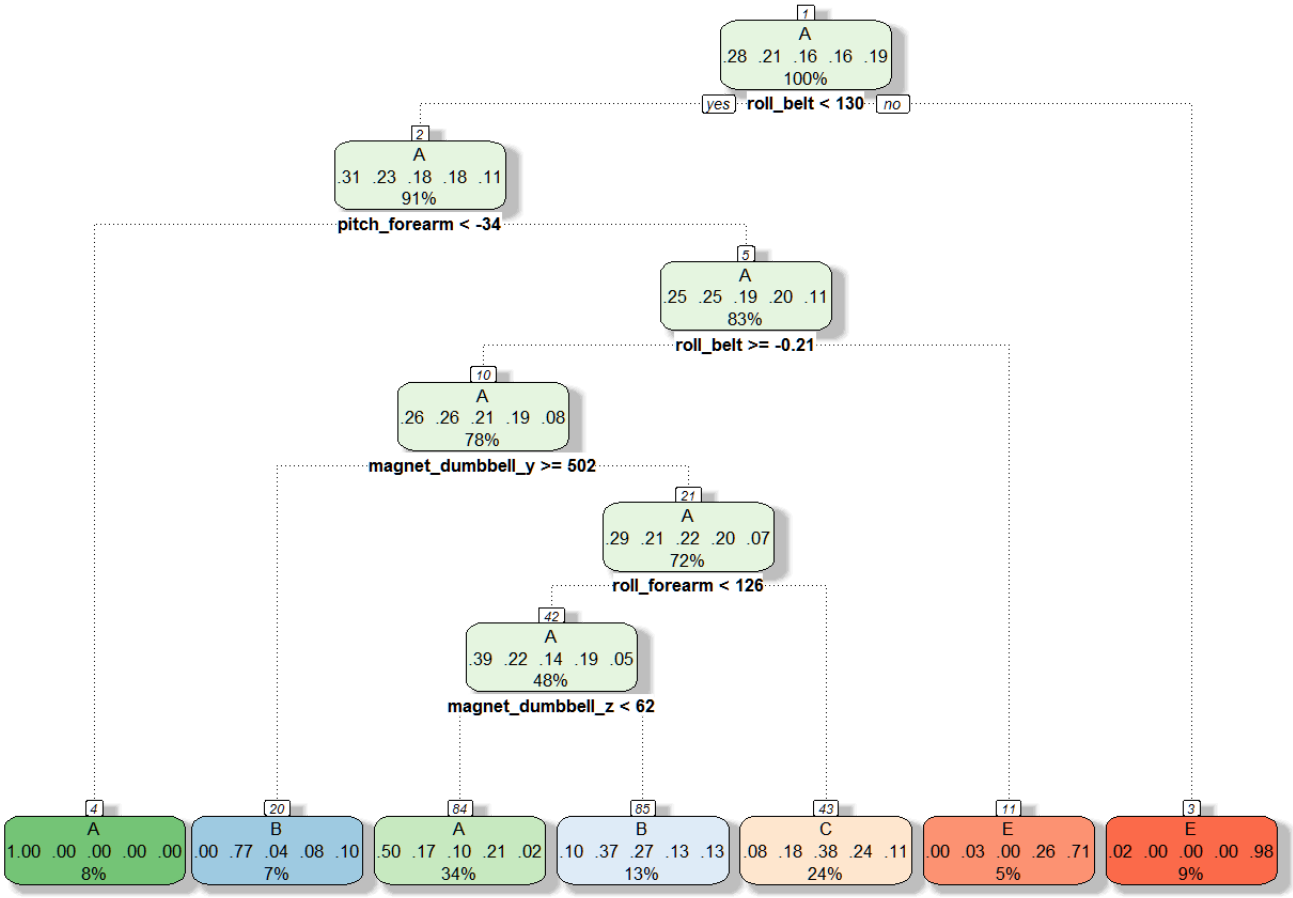
```
## n= 702
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 702 503 A (0.28 0.21 0.16 0.16 0.19)
##    2) roll_belt< 130.5 638 440 A (0.31 0.23 0.18 0.18 0.11)
##      4) pitch_forearm< -33.95 53   0 A (1 0 0 0 0) *
##      5) pitch_forearm>=-33.95 585 440 A (0.25 0.25 0.19 0.2 0.11)
##       10) roll_belt>=-0.21 550 405 A (0.26 0.26 0.21 0.19 0.076)
##         20) magnet_dumbbell_y>=501.5 48  11 B (0 0.77 0.042 0.083 0.1) *
##         21) magnet_dumbbell_y< 501.5 502 357 A (0.29 0.21 0.22 0.2 0.074)
##           42) roll_forearm< 125.5 334 203 A (0.39 0.22 0.14 0.19 0.054)
##             84) magnet_dumbbell_z< 61.5 242 120 A (0.5 0.17 0.095 0.21 0.025) *
##             85) magnet_dumbbell_z>=61.5 92  58 B (0.098 0.37 0.27 0.13 0.13) *
##           43) roll_forearm>=125.5 168 104 C (0.083 0.18 0.38 0.24 0.11) *
##       11) roll_belt< -0.21 35  10 E (0 0.029 0 0.26 0.71) *
##    3) roll_belt>=130.5 64   1 E (0.016 0 0 0 0.98) *
```

```
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.1.2
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 3.3.0 Copyright (c) 2006-2014 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
fancyRpartPlot(exSet1Modelrpart$finalModel)
```



Rattle 2014-Nov-23 14:18:29 Eric

This is a nice quick way to review the information, but it seems as if there is insufficient accuracy here. In some variants of this, not all classifications are represented! Let us try a random forest approach instead.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.1.2
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
exSet1Modelrf <- randomForest(classe~.,data=trainEx1)
exSet1Modelrf
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = trainEx1)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 9.4%
## Confusion matrix:
##     A   B   C   D   E class.error
## A 193   2   0   3   1     0.03015
## B  12 123   8   1   0     0.14583
## C   2   7 104   1   0     0.08772
## D   4   2   9 100   0     0.13043
## E   0   7   5   2 116     0.10769
```

This version of the random forest predicts an error rate of 9.54%. This may not be ideal, but let us take a look with the full data set which may help us cut down on the variance.

```
modelRf <- randomForest(classe~.,data=trainSet)
modelRf
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = trainSet)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.41%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3903    1    0    1    1    0.000768
## B   10 2645    3    0    0    0.004891
## C    0    8 2385    3    0    0.004591
## D    0    0   22 2228    2    0.010657
## E    0    0    0    6 2519    0.002376
```

The expected error rate is way down and the confusion matrix looks much better. Let’s see how well it performs against the test data.

```
pred <- predict(modelRf,testSet)
testSet$predRight <- pred == testSet$classe
table(pred,testSet$classe)
```

```
##
## pred    A    B    C    D    E
##    A 1674    3    0    0    0
##    B    0 1133    5    0    0
##    C    0    3 1021    5    0
##    D    0    0    0  959    0
##    E    0    0    0    0 1082
```

This shows a consistent degree of accuracy in the confusion matrix, and this is ultimately the model I used to submit below.

APPENDIX - submitting files

```
testData <- read.csv(testFile.dpath)
submitPredictionsModelRf <- predict(modelRf,testData)
submitPredictionsModelRf
```

```
##  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
```

```
answersDir <- "./answers"
if(!file.exists(answersDir)) {dir.create(answersDir)}

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

pml_write_files(submitPredictionsModelRf,answersDir)
```

These files received 20/20 correct.