PML-CourseProject

Carl Ebbinghaus

7 10 2017

Background

The goal of this 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.

Data and libraries

At first we load libraries and download the data from the URLs given in the assignment. Then we read the data.

```
library(kernlab); library(caret); library(e1071)
```

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

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

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```
fileUrl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.cs
v"
download.file(fileUrl, destfile="./pml-training.csv")
fileUrl2 <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.cs
v"
download.file(fileUrl2, destfile="./pml-testing.csv")
training <- read.csv("pml-training.csv")
testing <- read.csv("pml-testing.csv")
```

First inspection of data

To get an overview of the data I looked at head, tail, dim, str and summary of the training and test data. Because these functions have a lot of output i commented most of them out.

```
# head(training)
# tail(training)
dim(training)
```

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

```
# str(training)
# summary(training)

# head(testing)
# tail(testing)
dim(testing)
```

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

```
# str(testing)
# summary(testing)
```

From these outputs I was able to see, that a lot of columns seem to have NAs or empty rows.

Preparation of data

We now get rid of unnecessary columns and convert all columns in training and testing to num - data types were different across training and testing (some int in testing and num in training). Now both data frames are more equal.

```
## making the data set smaller
# we don't need timestamp data
timeColumns <- grep("timestamp", names(training))
training <- training[ , -c(1, timeColumns)] # also getting rid of the first col
umn
testing <- testing[ , -c(1, timeColumns)] # also getting rid of the first colum
n

# remove columns where more than 25 percent are empty or NA
maxNAok <- (dim(training)[1]) * .25
naColumns <- which(colSums(is.na(training) | training=="") > maxNAok)
training <- training[ , -naColumns]
testing <- testing[ , -naColumns]

# what classe means
# Six young health participants were asked to perform one set of 10 repetitions
of the Unilateral Dumbbell Biceps Curl in five different fashions: exactly acco
rding to the specification (Class A), throwing the elbows to the front (Class B
), lifting the dumbbell only halfway (Class C), lowering the dumbbell only half
way (Class D) and throwing the hips to the front (Class E).

# convert all to num except for classe
clLevels <- levels(training$classe)
training <- data.frame(data.matrix(training))
testing <- data.frame(data.matrix(testing))
training$classe <- factor(training$classe, labels = clLevels)
```

Fitting the models

We now split our training sample again to have a testset to check our models with. After that we fit a decision tree and a random forest. Note: Fitting the random forest takes super long on my computer - but it works :) If you have a suggestion of how to enhance the randomForest performance I would be happy to get it.

```
set.seed(3212345)
library(caret); library(kernlab)
clIndex <- which(names(training) == "classe")
inTrain <- createDataPartition(y = training$classe, p = 0.75, list = FALSE)
trtraining <- training[inTrain, ]
trtesting <- training[-inTrain, ]

modDT <- train(classe ~ ., data = trtraining, method = "rpart")
modRF <- train(classe ~ ., data = trtraining, method = "rf")
```

```
## 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
```

Prediction of training set

We now use the models to predict our trtesting sample. Then we compare the predicted classe to the real classe via a confusionmatrix (and only show the accuracy).

```
predDT <- predict(modDT, trtesting)
predRF <- predict(modRF, trtesting)

(confusionMatrix(trtesting$classe, predDT))$overall["Accuracy"]
```

```
##  Accuracy
## 0.4965334
```

```
(confusionMatrix(trtesting$classe, predRF))$overall["Accuracy"]
```

```
##  Accuracy
## 0.9963295
```

The accuracy of the randomForest model is way better than the decision tree accuracy. The randomForest accuracy is 0.9961256 and hence the expected out of sample error is 0.0038744 (1-accuracy).

Prediction for the testing data

Beacuse the randomForest accuracy is better I use the randomForest model to make the prediction for the “real” test set (testing).

```
predictionRF <- predict(modRF, testing)
predictionRF # show the predicted values for the test set given inside the assi
gnment
```

```
##  [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
```

Conclusion

I used randomForest to predict the classe of the given testing data because it had a high accuracy. Like explained in the lectures the randomForest takes long to compute though.