Predicting Exercise Manner using Random Forest

geoffchia

Sunday, April 26, 2015

Objective

The objective of this project is to build a predictive model using machine learning method to correctly predict how people perform their exercises by classifying them into one of the 5 categories: A, B, C, D and E.

Data

The training data for this project are provided and can be found here: (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv)

The test data are also made available: (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv (https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv)

Data Processing and Cleaning

First we download both data files to the local folder: c:/Coursera. We then use R to load the data and to take a quick glance at the data

```
library(caret); library(data.table); library(randomForest)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
setwd("C://Coursera")
dat <- read.csv("pml-training.csv", na.strings="NA")
dim(dat)
```

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

We then get rid of useless columns (e.g. those with a lot of NAs, or blanks), and only retain the predictors which are those with column name containing “belt”, “arm” and “dumbbell”

```
# get rid of columns with a lot of na values
dat <- dat[colSums(is.na(dat)) < 1000]

# exclude columns where most values are ""
cols <- c()
for (cname in colnames(dat))
    if (sum(dat[, cname] == "") < 1000) {
        cols <- c(cols, cname)
    }
dat <- dat[, cols]

# only retain columns with names consisting of "belt", "arm", "dumbbell"
cols <- grep("belt", colnames(dat))
cols <- c(cols, grep("arm", colnames(dat)))
cols <- c(cols, grep("dumbbell", colnames(dat)))

# add back "classe", which is the last col
cols <- c(cols, dim(dat)[2])
dat <- dat[, cols]
dim(dat)
```

```
## [1] 19622    53
```

As can be seen, the predictors have been reduced from 160 to 53, a more manageable number for modeling.

Training and Testing Data

We then sub-divide the data to 75% training and 25% testing. The purpose is for us to calculate out-of-sample error later.

```
inTrain <- createDataPartition(dat$classe, p=.75, list=FALSE)
training <- dat[inTrain,]
testing <- dat[-inTrain,]
```

Building Random Forest Model

We choose Random Forest model because it is one of the more powerful and commonly used machine learning model. We first use all the default settings.

```
set.seed(23221)
rf <- randomForest(classe ~., data=training)
rf
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.5%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4180    4    0    0    1    0.001195
## B   14 2831    3    0    0    0.005969
## C    0   16 2547    4    0    0.007791
## D    0    0   22 2386    4    0.010779
## E    0    0    0    6 2700    0.002217
```

Calculate Out-of-Sample Error

To assess the performance of the model, we apply it to make prediction on the testing data and work out the out-of-sample error:

```
# make prediction on testing data using our model
pred <- predict(rf, testing)

# calculate out-of-sample error
tbl <- table(pred, testing$classe)
err <- 1 - sum(diag(tbl)) / sum(tbl)
err
```

```
## [1] 0.006525
```

The model performs quite well, so there is no need to tweak the parameters further.

Putting Our Model to Work

We now use our model to predict the 20 cases in the test data:

Note: codes omitted to comply with Code of Honour of Coursera.

Conclusion

In this simple exercise, the default random forest model proves to be a fairly suitable model and we do not need to perform other tweaking. In practice, we would normally require to explore various models before deciding on the final one.