Project for PML

Rui Wang

Background

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).

Data

The training data for this project are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

The test data are available here:

https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

The data for this project come from this source: http://groupware.les.inf.puc-rio.br/har. If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

Load the package and data

I first get started with loading the package and data I might use in the code.

```
library(caret)
library(rpart)
library(rpart.plot)
library(rattle)
library(randomForest)
# load the data I have already saved in the directory
# fill the empty column with NA value for data cleaning later
training <- read.csv("pml-training.csv", na.strings = c("NA", ""), header = TRUE)
testing <- read.csv("pml-testing.csv", na.strings = c("NA", ""), header = TRUE)

# set the same seed
set.seed(666)

# look at the sample size
dim(training)
```

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

```
dim(testing)
```

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

```r
# all the column names are the same except the last one ("classe" and "problem_id")
index <- which(!(colnames(training) == colnames(testing)))
colnames(training)[index]
```

```
## [1] "classe"
```

```r
colnames(testing)[index]
```

```
## [1] "problem_id"
```

Data processing and cleaning

I will eliminate both NAs in the training and testing data.

```r
set.seed(666)

# count the number of non-NAs in each column of both data
count_train <- as.vector(apply(training, 2, function(x) length(which(!is.na(x)))))
# choose the column index of more than 60% non-NAs
index_train <- which(count_train/dim(training)[1] > 0.6)
training <- training[, index_train]

# do the same process for testing set
count_test <- as.vector(apply(testing, 2, function(x) length(which(!is.na(x)))))
index_test <- which(count_test/dim(testing)[1] > 0.6)
testing <- testing[, index_test]

# check the column names again
# all the column names are the same except the last one ("classe" and "problem_id")
index <- which(!(colnames(training) == colnames(testing)))
colnames(training)[index]
```

```
## [1] "classe"
```

```r
colnames(testing)[index]
```

```
## [1] "problem_id"
```

```r
# remove the first 7 columns of both data since they are not related with the measurement data
training <- training[, -c(1:7)]
testing <- testing[, -c(1:7)]
```

Now the data sets are ready to go! Since the data size between training and testing is not comparable, I decide to split my training set into training and testing data sets by 6:4 to find the best model. Last I will use the best model I find out to predict the small data set with only 20 observations.

```r
set.seed(666)
inTrain <- createDataPartition(y = training$classe, p = 0.6, list = FALSE)
my_training <- training[inTrain, ]
my_testing <- training[-inTrain, ]
# check each dimension
dim(my_training)
```

```
## [1] 11776    53
```

```r
dim(my_testing)
```

```
## [1] 7846   53
```

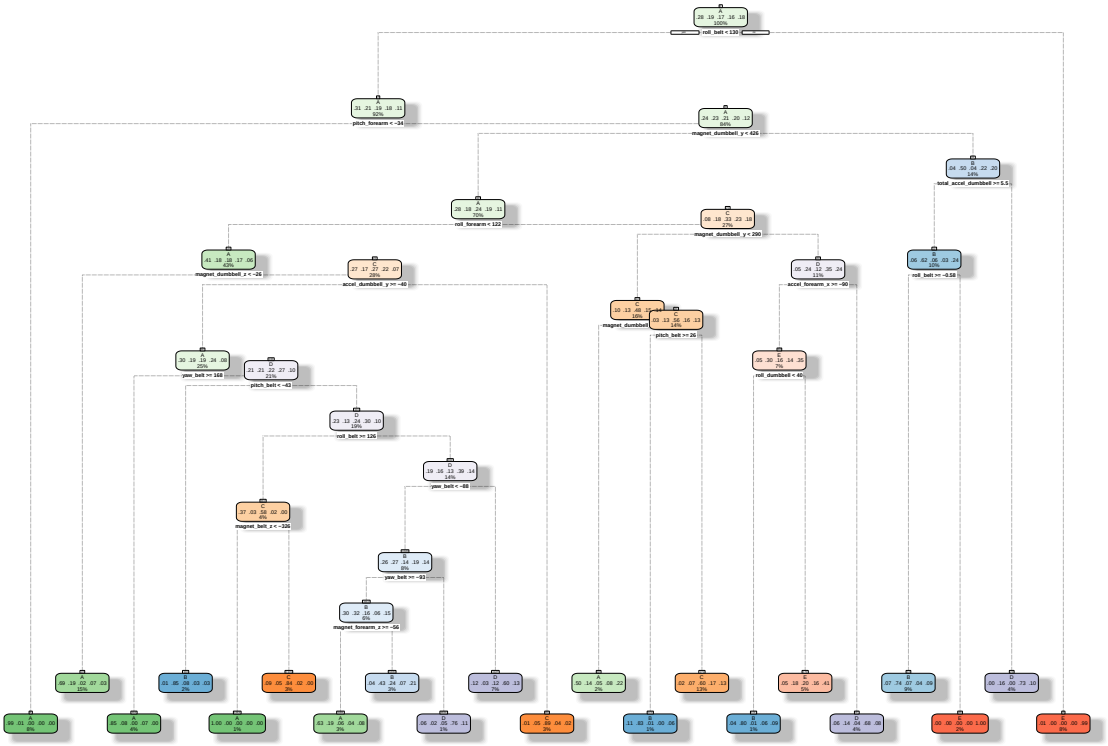
Model and prediction

I will use 2 different `rpart` and `randomForest` packages to fit and predict the data.

1st model with recursive partitioning and regression trees

```
set.seed(666)
mod1 <- rpart(classe ~. , method = "class", data = my_training)
# plot the decision tree
fancyRpartPlot(mod1)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Rattle 2017−Feb−05 14:11:02 ruiwang

```
# print the test results
predict1 <- predict(mod1, my_testing, type = "class")
confusionMatrix(predict1, my_testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1970  344   44  131   99
##          B   85  890  140   49  133
##          C   45  106 1001  178  132
##          D  105  100   99  856  142
```

```
##          E   27   78   84   72  936
##
## Overall Statistics
##
##                Accuracy : 0.7205
##                  95% CI : (0.7104, 0.7304)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6446
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8826   0.5863   0.7317   0.6656   0.6491
## Specificity            0.8899   0.9357   0.9288   0.9320   0.9592
## Pos Pred Value         0.7612   0.6862   0.6847   0.6575   0.7820
## Neg Pred Value         0.9502   0.9041   0.9425   0.9343   0.9239
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2511   0.1134   0.1276   0.1091   0.1193
## Detection Prevalence   0.3298   0.1653   0.1863   0.1659   0.1526
## Balanced Accuracy      0.8863   0.7610   0.8303   0.7988   0.8042
```

```
accuracy1 <- confusionMatrix(predict1, my_testing$classe)$overall[1]
```

Conclusion: After I apply the first model with **rpart**, the accuracy I obtain is 0.72, which seems relatively a good model to predict the data.

2nd model using random forests

```
set.seed(666)
mod2 <- randomForest(classe ~. , method = "class", data = my_training)
predict2 <- predict(mod2, my_testing, type = "class")
confusionMatrix(predict2, my_testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2229   12    0    0    0
##          B    3 1500   20    0    0
##          C    0    6 1345   14    0
##          D    0    0    3 1272    5
##          E    0    0    0    0 1437
##
## Overall Statistics
##
##                Accuracy : 0.992
##                  95% CI : (0.9897, 0.9938)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9898
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9987   0.9881   0.9832   0.9891   0.9965
## Specificity            0.9979   0.9964   0.9969   0.9988   1.0000
## Pos Pred Value         0.9946   0.9849   0.9853   0.9938   1.0000
## Neg Pred Value         0.9995   0.9972   0.9965   0.9979   0.9992
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2841   0.1912   0.1714   0.1621   0.1832
## Detection Prevalence   0.2856   0.1941   0.1740   0.1631   0.1832
## Balanced Accuracy      0.9983   0.9923   0.9900   0.9939   0.9983
```

```
accuracy2 <- confusionMatrix(predict2, my_testing$classe)$overall[1]
```

Conclusion: The second model with `randomForest` provides a nearly-perfect prediction. The accuracy I obtain from this model is 0.992.

Prediction on the test data sets

It is no doubt that I will choose the `randomForest` model to predict the short test data with only 20 observations.

```
predict(mod2, testing, type = "class")
```

```
##  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
```