Coursera Practical Machine Learning Course Project

Tan Thiam Huat

November 5, 2018

Background and Introduction

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit 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 regu-larly 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, we will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participant They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. The five ways are exactly according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E). Only Class A corresponds to correct performance. The goal of this project is to predict the manner in which they did the exercise, i.e., Class A to E. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (http://groupware.les.inf.puc-rio.br/har) (see the section on the Weight Lifting Exercise Dataset).

Import the data

We first load the R packages needed for analysis and then download the training and testing data sets from the given URLs. Those training and testing data sets are then loaded locally.

```
# Load the required packages
options(warn=-1)
library(caret); library(rattle); library(rpart); library(rpart.plot)
```

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

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

```
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(randomForest); library(repmis)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:rattle':
##
##     importance
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
setwd("D:/Coursera/data")
url1 <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
url2 <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

download.file(url=url1,destfile='pml-training.csv', method='curl')
download.file(url=url2,destfile='pml-testing.csv', method='curl')

# load data locally
training <- read.csv("D:\\Coursera\\data\\pml-training.csv", na.strings = c("NA", ""))
testing <- read.csv("D:\\Coursera\\data\\pml-testing.csv", na.strings = c("NA", ""))
```

The training dataset has 19622 observations and 160 variables, and the testing data set contains 20 observations and the same variables as the training set. We are trying to predict the outcome of the variable classe in the training set.

Data cleaning

We now delete columns (predictors) of the training set that contain any missing values.

```
training <- training[, colSums(is.na(training)) == 0]
testing <- testing[, colSums(is.na(testing)) == 0]
```

We also remove the first seven predictors since these variables have little predicting power for the outcome classe.

```
trainData <- training[, -c(1:7)]
testData <- testing[, -c(1:7)]
```

The cleaned data sets trainData and testData both have 53 columns with the same first 52 variables and the last variable classe and problem_id individually. trainData has 19622 rows while testData has 20 rows.

Data spliting

In order to get out-of-sample errors, we split the cleaned training set trainData into a training set (train, 70%) for prediction and a validation set (valid 30%) to compute the out-of-sample errors.

```
set.seed(7826)
inTrain <- createDataPartition(trainData$classe, p = 0.7, list = FALSE)
train <- trainData[inTrain, ]
valid <- trainData[-inTrain, ]
```

Prediction Algorithms

We use classification trees and random forests to predict the outcome.

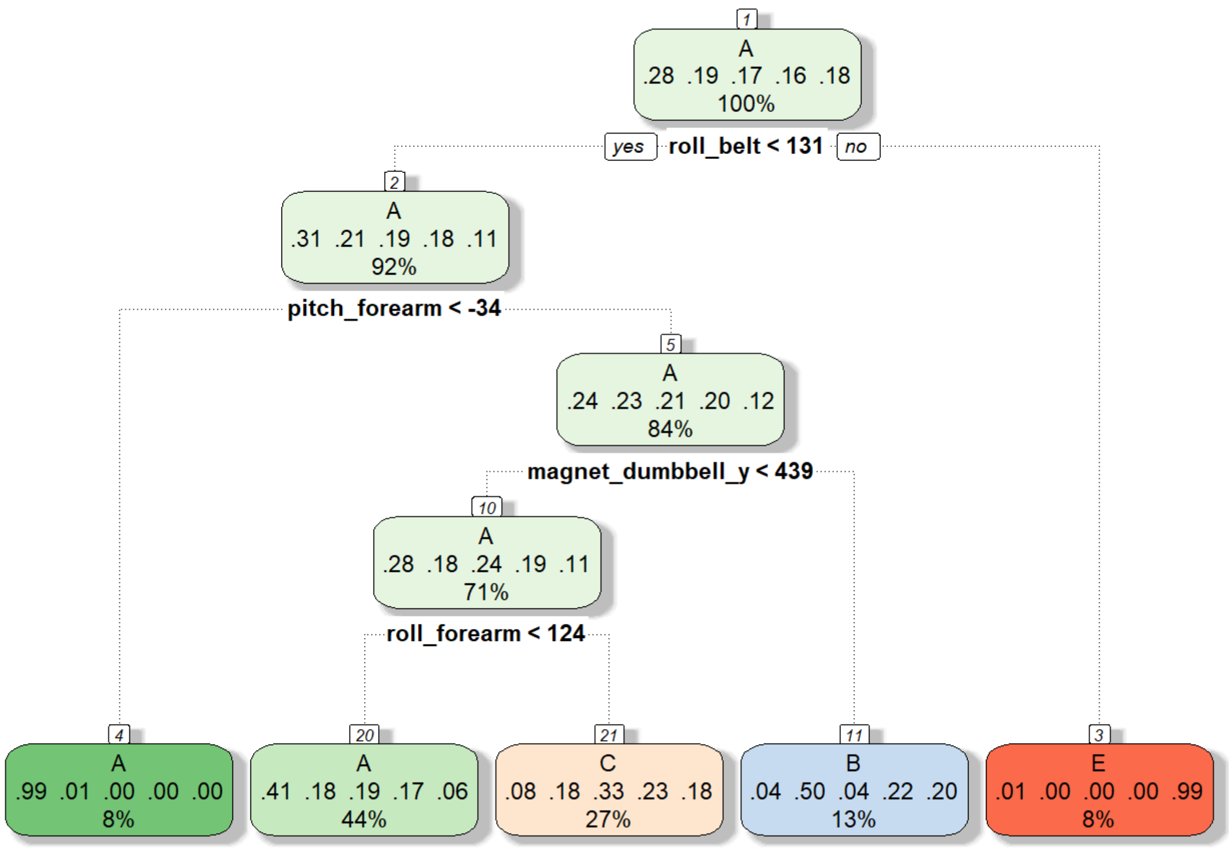
Classification trees

In practice, k=5 or k=10 when doing k-fold cross validation. Here we consider 5-fold cross validation (default setting in trainControl function is 10) when implementing the algorithm to save a little com-puting time. Since data transformations may be less important in non-linear models like classifica-tion trees, we do not transform any variables.

```
control <- trainControl(method = "cv", number = 5)
fit_rpart <- train(classe ~ ., data = train, method = "rpart", trControl = control)
print(fit_rpart, digits = 4)
```

```
## CART
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10989, 10989, 10989, 10991, 10990
## Resampling results across tuning parameters:
##
##   cp       Accuracy  Kappa
##   0.03723  0.5181    0.37581
##   0.05954  0.4154    0.20866
##   0.11423  0.3488    0.09842
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.03723.
```

```
fancyRpartPlot(fit_rpart$finalModel)
```



Rattle 2018-Nov-05 14:56:22 Tan Thiam Huat

```
# predict outcomes using validation set
predict_rpart <- predict(fit_rpart, valid)
# Show prediction result
(conf_rpart <- confusionMatrix(valid$classe, predict_rpart))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1544   21  107    0    2
##          B  492  391  256    0    0
##          C  474   38  514    0    0
##          D  436  175  353    0    0
##          E  155  138  293    0  496
## 
## Overall Statistics
## 
##                Accuracy : 0.5004
##                  95% CI : (0.4876, 0.5133)
##     No Information Rate : 0.5269
##     P-Value [Acc > NIR] : 1
## 
##                   Kappa : 0.3464
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4979  0.51245  0.33749       NA  0.99598
## Specificity            0.9533  0.85396  0.88262   0.8362  0.89122
## Pos Pred Value         0.9223  0.34328  0.50097       NA  0.45841
## Neg Pred Value         0.6303  0.92162  0.79234       NA  0.99958
## Prevalence             0.5269  0.12965  0.25879   0.0000  0.08462
## Detection Rate         0.2624  0.06644  0.08734   0.0000  0.08428
## Detection Prevalence   0.2845  0.19354  0.17434   0.1638  0.18386
## Balanced Accuracy      0.7256  0.68321  0.61006       NA  0.94360
```

```
(accuracy_rpart <- conf_rpart$overall[1])
```

```
##  Accuracy 
## 0.5004248
```

From the confusion matrix, the accuracy rate is 0.5, and so the out-of-sample error rate is 0.5. Using classification tree does not predict the outcome classe very well.

Random forests

Since classification tree method does not perform well, we try random forest method instead.

```
fit_rf <- train(classe ~ ., data = train, method = "rf", trControl = control)
print(fit_rf, digits = 4)
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 10991, 10990, 10988, 10990, 10989
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##    2    0.9905    0.9879
##   27    0.9918    0.9897
##   52    0.9864    0.9828
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```
# predict outcomes using validation set
predict_rf <- predict(fit_rf, valid)
# Show prediction result
(conf_rf <- confusionMatrix(valid$classe, predict_rf))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1669    3    0    0    2
##          B    9 1129    1    0    0
##          C    0    2 1017    7    0
##          D    0    0   15  946    3
##          E    1    1    1    5 1074
##
## Overall Statistics
##
##                Accuracy : 0.9915
##                  95% CI : (0.9888, 0.9937)
##     No Information Rate : 0.2853
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9893
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9940   0.9947   0.9836   0.9875   0.9954
## Specificity            0.9988   0.9979   0.9981   0.9963   0.9983
## Pos Pred Value         0.9970   0.9912   0.9912   0.9813   0.9926
## Neg Pred Value         0.9976   0.9987   0.9965   0.9976   0.9990
## Prevalence             0.2853   0.1929   0.1757   0.1628   0.1833
## Detection Rate         0.2836   0.1918   0.1728   0.1607   0.1825
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9964   0.9963   0.9909   0.9919   0.9969
```

```
(accuracy_rf <- conf_rf$overall[1])
```

```
##  Accuracy
## 0.9915038
```

For this dataset, random forest method is way better than classification tree method. The accuracy rate is 0.991, and so the out-of-sample error rate is 0.009. This may be due to the fact that many pre-dictors are highly correlated. Random forests chooses a subset of predictors at each split and decorrelate the trees. This leads to high accuracy, although this algorithm is sometimes difficult to interpret and computationally inefficient.

Prediction on Testing Set

We now use random forests to predict the outcome variable classe for the testing set.

```
(predict(fit_rf, testData))
```

```
##  [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
```

This is the end of the Course Project for Practical Machine Learning.