Practical Machine Learning - Final Project
Developer: Meng Qiu
Date: 10/30/2016

Part 1: Data Processing

1. Load libraries

```
library(caret)
library(rattle)
library(rpart)
library(rpart.plot)
library(randomForest)
library(repmis)
```

2. Import datasets

```
setwd("C:/Users/Owen/Desktop/final project")
training <- read.csv("C:/Users/Owen/Desktop/final project/datasets/pml-training.csv", na.strings = c("NA", ""))
testing <- read.csv("C:/Users/Owen/Desktop/final project/datasets/pml-testing.csv", na.strings = c("NA", ""))
dim(training); dim(testing)
names(training); names(testing)
```

We see that 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 to predict the outcome of the variable “classe” in the training set.

3. Data cleaning

First, we delete columns that contain missing values.

```
training <- training[, -c(which(colSums(is.na(training))>1))]
testing <- testing[, -c(which(colSums(is.na(training))>1))]
```

Second, delete the first seven predictors which have little predicting power.

```
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. The trainData has 19622 rows while testData has 20 rows.

4. 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(1234)
inTrain <- createDataPartition(trainData$classe, p = 0.7, list = FALSE)
train <- trainData[inTrain, ]
valid <- trainData[-inTrain, ]
```

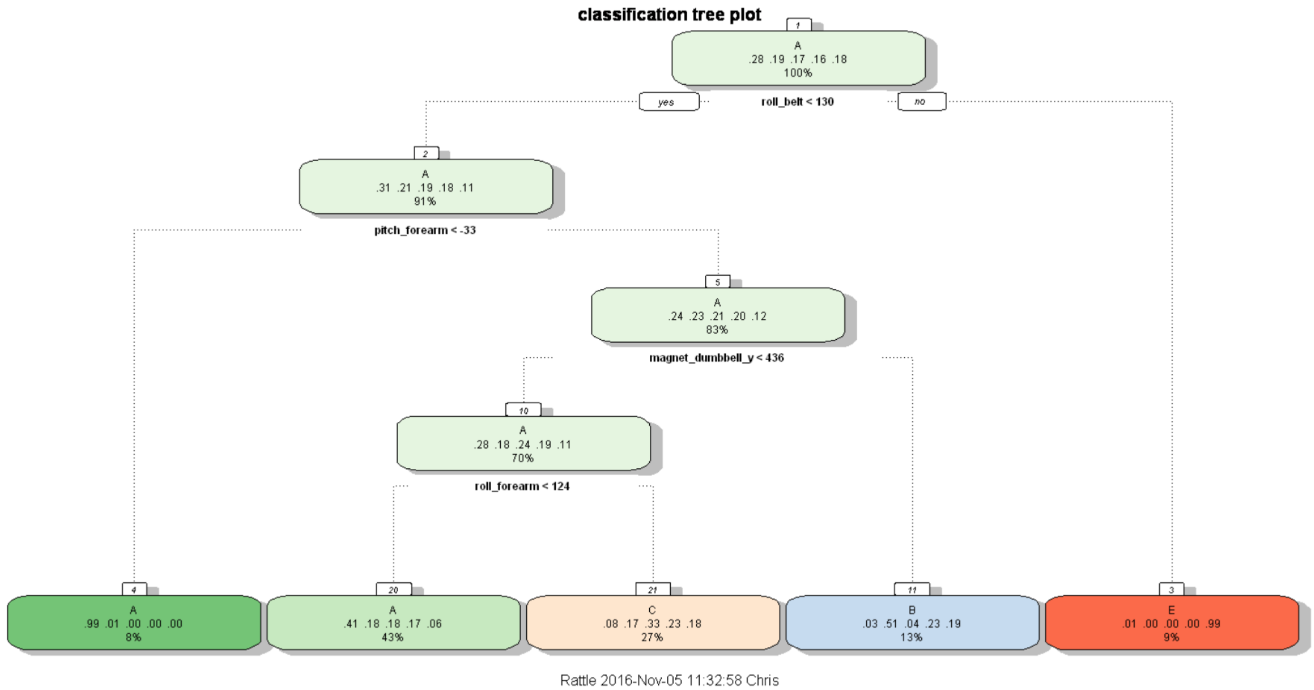
Part 2: Prediction Algorithm - Classification Tree & Random Forest

1. Classification trees

```
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: 10990, 10988, 10991, 10990, 10989
Resampling results across tuning parameters:

  cp       Accuracy  Kappa
  0.03550  0.5214    0.38010
  0.06093  0.4175    0.21094
  0.11738  0.3333    0.07467

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was cp = 0.0355.
```

```
fancyRpartPlot(fit_rpart$finalModel, main="classification tree plot", tweak=1)
```



Predict outcomes using validation set

```
predict_rpart <- predict(fit_rpart, valid)
```

Show prediction result

```
(conf_rpart <- confusionMatrix(valid$classe, predict_rpart))
```

```
$positive
NULL

$table
          Reference
Prediction    A    B    C    D    E
         A 1530   35  105    0    4
         B  486  379  274    0    0
         C  493   31  502    0    0
         D  452  164  348    0    0
         E  168  145  302    0  467
```

```
$overall
    Accuracy       Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
   0.4890399     0.3311096      0.4761916      0.5018991      0.5316907
AccuracyPValue  McnemarPValue
   1.0000000          NaN

$byClass
       Sensitivity Specificity Pos Pred Value Neg Pred Value Precision
Class: A   0.4889741   0.9477504      0.9139785      0.6202802 0.9139785
Class: B   0.5026525   0.8518807      0.3327480      0.9209861 0.3327480
Class: C   0.3278903   0.8796509      0.4892788      0.7882280 0.4892788
Class: D          NA   0.8361937             NA             NA 0.0000000
Class: E   0.9915074   0.8864056      0.4316081      0.9991672 0.4316081
          Recall        F1 Prevalence Detection Rate Detection Prevalence
Class: A 0.4889741 0.6371018 0.53169074     0.25998301           0.2844520
Class: B 0.5026525 0.4004226 0.12812234     0.06440102           0.1935429
Class: C 0.3278903 0.3926476 0.26015293     0.08530161           0.1743415
Class: D        NA        NA 0.00000000     0.00000000           0.1638063
Class: E 0.9915074 0.6014166 0.08003398     0.07935429           0.1838573

Balanced Accuracy
Class: A        0.7183622
Class: B        0.6772666
Class: C        0.6037706
Class: D               NA
Class: E        0.9389565

$mode
[1] "sens_spec"

$dots
list()
```

Show prediction accuracy

(accuracy_rpart <- conf_rpart$overall[1])

```
Accuracy
0.4890399
```

From the confusion matrix, the accuracy rate is 0.489, and so the out-of-sample error rate is 0.511. Using classification tree did not predict the outcome classe very well.

2. Random forest

Since the classification tree didn't perform well, we'll turn to random forest method.

```
set.seed(12345)
fit_rf <- train(classe ~ ., data = train, method = "rf", trControl = control)
print(fit_rf, digits = 4)
```

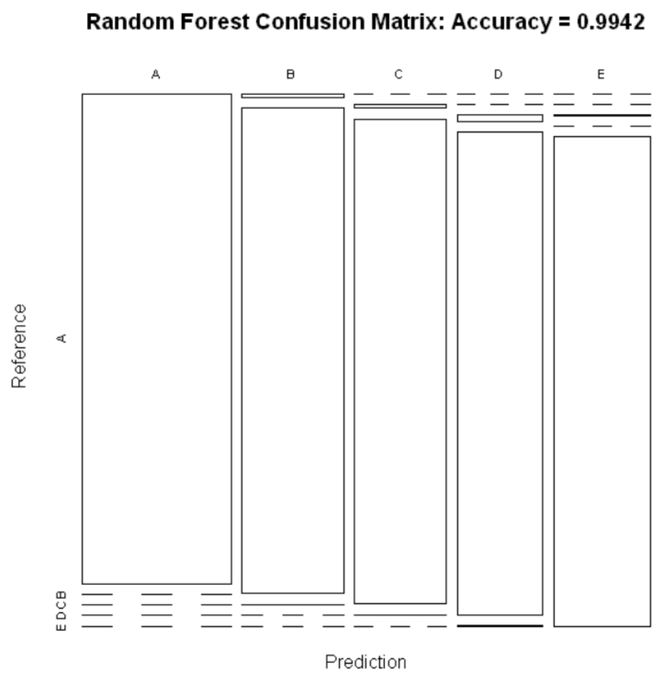
```
Random Forest

13737 samples
   52 predictor
    5 classes: 'A', 'B', 'C', 'D', 'E'

No pre-processing
Resampling: Cross-Validated (5 fold)
Summary of sample sizes: 10989, 10989, 10989, 10990, 10991
Resampling results across tuning parameters:

  mtry  Accuracy  Kappa
   2    0.9901    0.9875
  27    0.9900    0.9873
  52    0.9865    0.9829

Accuracy was used to select the optimal model using  the largest value.
The final value used for the model was mtry = 2.
```

```
predictRf <- predict(fit_rf, valid)
cm_rf <- confusionMatrix(valid$classe, predictRf)
(accuracy_rf <- cm_rf$overall[1])
```

```
Accuracy
0.9942226
```

```
plot(cm_rf$table, col = cm_rf$byClass, main = paste("Random Forest Confusion Matrix: Accuracy =",
round(cm_rf$overall['Accuracy'], 4)))
```

```
predictRf <- predict(fit_rf, valid)
cm_rf <- confusionMatrix(valid$classe, predictRf)
(accuracy_rf <- cm_rf$overall[1])
plot(cm_rf$table, col = cm_rf$byClass, main = paste("Random Forest Confusion Matrix: Accuracy =",
round(cm_rf$overall['Accuracy'], 4)))
```



For this dataset, random forest method is way better than classification tree method. The accuracy rate is 0.994, and so the out-of-sample error rate is 0.006.

This may be due to the fact that many predictors 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.

Part 3: Prediction on the Test Dataset

predict(fit_rf, testData)

```
A E D B A A B C B A E E A B B B
Levels: A B C D E
```

Project Ends