

# Practical Machine Learning Project

*Snehal*

*November 22, 2016*

## 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 dumbbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways.

## Data Processing

## Data Load

```
# Load the required packages
library(caret); library(rattle); library(rpart); library(rpart.plot)
```

```
## Warning: package 'caret' was built under R version 3.3.2
```

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

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

```
## Warning: package 'rattle' was built under R version 3.3.2
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
## Warning: package 'rpart.plot' was built under R version 3.3.2
```

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

```
## Warning: package 'randomForest' was built under R version 3.3.2
```

```
## 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
```

```
## Warning: package 'repmis' was built under R version 3.3.2
```

```
# import the data from the URLs
# trainurl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
# testurl <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
# training <- source_data(trainurl, na.strings = c("NA", "#DIV/0!", ""), header = TRUE)
# testing <- source_data(testurl, na.strings = c("NA", "#DIV/0!", ""), header = TRUE)
# load data locally
training <- read.csv("pml-training.csv", na.strings = c("NA", ""))
testing <- read.csv("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

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]
```

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 Splitting

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.

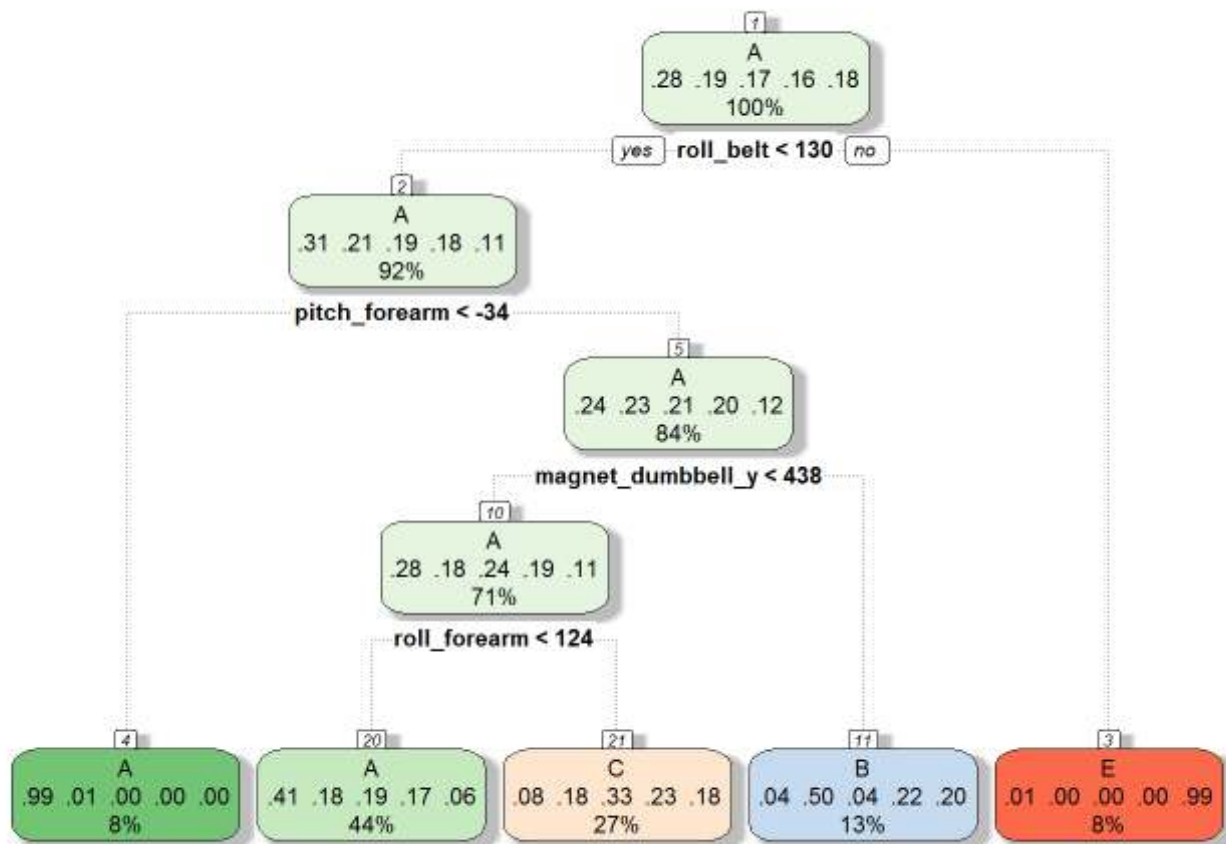
## Classification trees

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 computing time. Since data transformations may be less important in non-linear models like classification 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, 10990, 10989, 10991
## Resampling results across tuning parameters:
##
##    cp          Accuracy    Kappa
##  0.03723  0.5241    0.38748
##  0.05954  0.4144    0.20668
##  0.11423  0.3482    0.09762
##
## 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 2016-Nov-23 13:29:26 AjitSneha

```

# 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: 10990, 10990, 10990, 10988, 10990
## Resampling results across tuning parameters:
##
##    mtry  Accuracy  Kappa
##    2    0.9908    0.9884
##    27    0.9906    0.9881
##    52    0.9859    0.9821
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

```
# 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 1673    0    0    0    1
##           B   1 1136    2    0    0
##           C   0   4 1020    2    0
##           D   0   0   21  941    2
##           E   0   0   0   2 1080
##
## Overall Statistics
##
##           Accuracy : 0.9941
##           95% CI : (0.9917, 0.9959)
##           No Information Rate : 0.2845
##           P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 0.9925
##           McNemar's Test P-Value : NA
##
## Statistics by Class:
##
##           Class: A Class: B Class: C Class: D Class: E
## Sensitivity       0.9994  0.9965  0.9779  0.9958  0.9972
## Specificity       0.9998  0.9994  0.9988  0.9953  0.9996
## Pos Pred Value    0.9994  0.9974  0.9942  0.9761  0.9982
## Neg Pred Value    0.9998  0.9992  0.9953  0.9992  0.9994
## Prevalence        0.2845  0.1937  0.1772  0.1606  0.1840
## Detection Rate    0.2843  0.1930  0.1733  0.1599  0.1835
## Detection Prevalence 0.2845  0.1935  0.1743  0.1638  0.1839
## Balanced Accuracy  0.9996  0.9979  0.9884  0.9956  0.9984
```

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

```
## Accuracy
## 0.9940527
```

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 predictors are highly correlated. Random forests chooses a subset of predictors at each split and decorrelate the trees. This leads to high accuracy.

## 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
```

Project completes here.