Practical Machine Learning Course Project

February 12, 2018

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

Goal

The goal of the project is to predict the manner in which the partcipants did the exercise. The predictor the "classe" variable in the training set. Create a report describing how the model is built, how cross validation is used, what is beleived is the expected out of the sample error.Use the prediction model to predict 20 different test cases.

Load required libraries for this project

```
library(rattle)
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.1.0 Copyright (c) 2006-2017 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(caret)
```

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

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

```
library(rpart)
library(rpart.plot)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
library(randomForest)
```

```
## 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
## The following object is masked from 'package:rattle':
##
##     importance
library(RColorBrewer)
```

Read and load data for processing

Data was previously downloaded to a subdirectory "Data" within "r" working directory

```
trainData <- read.csv("./Data/pml-training.csv")
testData <- read.csv("./Data/pml-testing.csv")
testing <- testData
str(trainData)
```

```
## 'data.frame':    19622 obs. of  160 variables:
##  $ X                       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ user_name               : Factor w/ 6 levels "adelmo","carlitos",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ raw_timestamp_part_1    : int  1323084231 1323084231 1323084231 1323084232 1323084232 1323084232 1
##  $ raw_timestamp_part_2    : int  788290 808298 820366 120339 196328 304277 368296 440390 484323 484
##  $ cvtd_timestamp          : Factor w/ 20 levels "02/12/2011 13:32",..: 9 9 9 9 9 9 9 9 9 9 ...
##  $ new_window              : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ num_window              : int  11 11 11 12 12 12 12 12 12 12 ...
##  $ roll_belt               : num  1.41 1.41 1.42 1.48 1.48 1.45 1.42 1.42 1.43 1.45 ...
##  $ pitch_belt              : num  8.07 8.07 8.07 8.05 8.07 8.06 8.09 8.13 8.16 8.17 ...
##  $ yaw_belt                : num  -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 -94.4 ...
##  $ total_accel_belt        : int  3 3 3 3 3 3 3 3 3 3 ...
##  $ kurtosis_roll_belt      : Factor w/ 397 levels "","-0.016850",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_belt     : Factor w/ 317 levels "","-0.021887",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt      : Factor w/ 395 levels "","-0.003095",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_belt.1    : Factor w/ 338 levels "","-0.005928",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_belt       : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_belt          : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_belt            : Factor w/ 68 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_belt     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_belt    : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_belt      : Factor w/ 4 levels "","#DIV/0!","0.00",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ var_total_accel_belt    : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_belt        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_belt           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
```

```
##  $ stddev_pitch_belt       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_belt          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_belt         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_belt            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_belt_x            : num  0 0.02 0 0.02 0.02 0.02 0.02 0.02 0.02 0.03 ...
##  $ gyros_belt_y            : num  0 0 0 0 0.02 0 0 0 0 0 ...
##  $ gyros_belt_z            : num  -0.02 -0.02 -0.02 -0.03 -0.02 -0.02 -0.02 -0.02 -0.02 0 ...
##  $ accel_belt_x            : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y            : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z            : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ magnet_belt_x           : int  -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...
##  $ magnet_belt_y           : int  599 608 600 604 600 603 599 603 602 609 ...
##  $ magnet_belt_z           : int  -313 -311 -305 -310 -302 -312 -311 -313 -312 -308 ...
##  $ roll_arm                : num  -128 -128 -128 -128 -128 -128 -128 -128 -128 -128 ...
##  $ pitch_arm               : num  22.5 22.5 22.5 22.1 22.1 22 21.9 21.8 21.7 21.6 ...
##  $ yaw_arm                 : num  -161 -161 -161 -161 -161 -161 -161 -161 -161 -161 ...
##  $ total_accel_arm         : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ var_accel_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_roll_arm         : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_pitch_arm        : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ avg_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ stddev_yaw_arm          : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ var_yaw_arm             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ gyros_arm_x             : num  0 0.02 0.02 0.02 0 0.02 0 0.02 0.02 0.02 ...
##  $ gyros_arm_y             : num  0 -0.02 -0.02 -0.03 -0.03 -0.03 -0.03 -0.02 -0.03 -0.03 ...
##  $ gyros_arm_z             : num  -0.02 -0.02 -0.02 0.02 0 0 0 0 -0.02 -0.02 ...
##  $ accel_arm_x             : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y             : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z             : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ magnet_arm_x            : int  -368 -369 -368 -372 -374 -369 -373 -372 -369 -376 ...
##  $ magnet_arm_y            : int  337 337 344 344 337 342 336 338 341 334 ...
##  $ magnet_arm_z            : int  516 513 513 512 506 513 509 510 518 516 ...
##  $ kurtosis_roll_arm       : Factor w/ 330 levels "","-0.02438",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_arm      : Factor w/ 328 levels "","-0.00484",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_arm        : Factor w/ 395 levels "","-0.01548",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_arm       : Factor w/ 331 levels "","-0.00051",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_arm      : Factor w/ 328 levels "","-0.00184",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_arm        : Factor w/ 395 levels "","-0.00311",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_roll_arm            : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_arm           : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_arm             : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_roll_arm      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_pitch_arm     : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ amplitude_yaw_arm       : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ roll_dumbbell           : num  13.1 13.1 12.9 13.4 13.4 ...
##  $ pitch_dumbbell          : num  -70.5 -70.6 -70.3 -70.4 -70.4 ...
```

```
##  $ yaw_dumbbell            : num  -84.9 -84.7 -85.1 -84.9 -84.9 ...
##  $ kurtosis_roll_dumbbell  : Factor w/ 398 levels "","-0.0035","-0.0073",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_picth_dumbbell : Factor w/ 401 levels "","-0.0163","-0.0233",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ kurtosis_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_roll_dumbbell  : Factor w/ 401 levels "","-0.0082","-0.0096",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_pitch_dumbbell : Factor w/ 402 levels "","-0.0053","-0.0084",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ skewness_yaw_dumbbell   : Factor w/ 2 levels "","#DIV/0!": 1 1 1 1 1 1 1 1 1 1 ...
##  $ max_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_picth_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ max_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ min_roll_dumbbell       : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_pitch_dumbbell      : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ min_yaw_dumbbell        : Factor w/ 73 levels "","-0.1","-0.2",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ amplitude_roll_dumbbell : num  NA NA NA NA NA NA NA NA NA NA ...
##   [list output truncated]
```

Exploratory Data Analysis and Data Cleaning

The training dataset contains 159 predictors and a dependent variable "classe". By reviewing the output of str(trainData) we can see that the data contains columns that have "NAs", empty cells(""), and "#DIV/0!" .

```
# Columns 1-7 are row numbers,time stamps,new_window,and num_window and are
# unnecessay for this analysis
training <- trainData[,-c(1:7)]

# Remove columns with more than 95% NAs
training <- training[, -which(colMeans(is.na(training)) > 0.95)]

# Function to add a "0" level to factor columns. Will be required to
# be able to normalize the empty cells and cells that contains "#DIV/0"
# with a value of "0"
addZeroLevel <- function(x){
      if(is.factor(x)) return(factor(x, levels=c(levels(x), "0")))
      return(x)
}
# Add a "0" level to factor columns using addZeroLevel function created prior
training <- as.data.frame(lapply(training, addZeroLevel))

# Replace all "#DIV/0"" with "0";
training[training=="#DIV/0!"] <- "0"

# Replace all empty cells with "0";
training[training==""] <- "0"

# remove columns where over 95% of the values are "0"
training <- training[, -which(colMeans(training=="0") > 0.95)]

# drop unused levels
training <- droplevels(training)
```

Split training data into two components

a) training - used to create the model

b) validation - used to validate the model

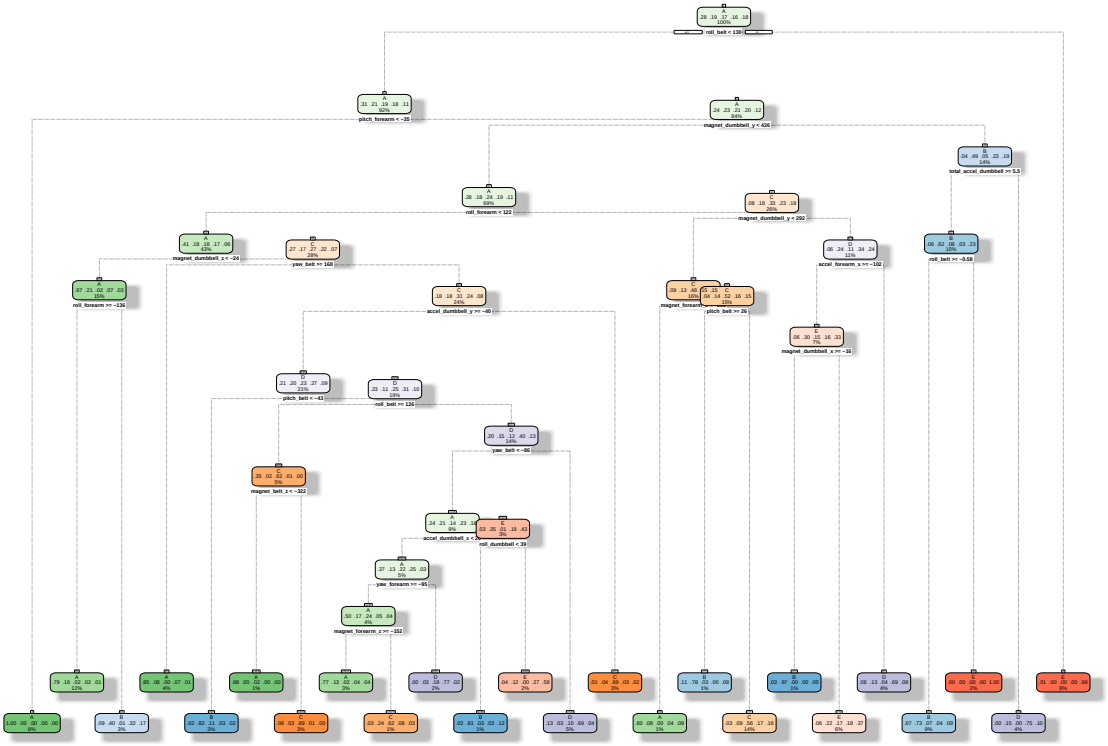
```
set.seed(256)
inTrain <- createDataPartition(y = training$classe, p = 0.75, list = FALSE)
training <- training[inTrain, ]
validation <- training[-inTrain, ]
```

Create a model using decision trees on the training dataset.

Use the model to predict "classe" in the validation dataset and use the confusion matrix to compare the predicted versus the actual labels:

```
modTree <- rpart(classe ~ ., data = training, method = "class")
fancyRpartPlot(modTree)
```

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




```
predTree <- predict(modTree, validation, type = "class")

# Use confusion matrix to get estimate of out-of-sample error. Get the accuracy
# of the prediction
confusionMatrix(validation$classe, predTree)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
```

```
##          A 903  37  32  34  15
##          B  95 469  69  56  69
##          C  11  37 509  29  38
##          D  19  54  92 366  56
##          E  10  50  98  31 488
##
## Overall Statistics
##
##                Accuracy : 0.7458
##                  95% CI : (0.7314, 0.7599)
##     No Information Rate : 0.2831
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6789
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8699   0.7249   0.6362  0.70930   0.7327
## Specificity            0.9551   0.9043   0.9599  0.92986   0.9370
## Pos Pred Value         0.8844   0.6187   0.8157  0.62351   0.7208
## Neg Pred Value         0.9490   0.9388   0.9044  0.95130   0.9405
## Prevalence             0.2831   0.1764   0.2182  0.14071   0.1816
## Detection Rate         0.2463   0.1279   0.1388  0.09981   0.1331
## Detection Prevalence   0.2784   0.2067   0.1702  0.16008   0.1846
## Balanced Accuracy      0.9125   0.8146   0.7981  0.81958   0.8349
```

Decision Tree accuracy and out of sample error

```
accuracy <-postResample(predTree, validation$classe)
print(paste("Accuracy %", accuracy[1]*100, sep=": "))
```

```
## [1] "Accuracy %: 74.5841287155713"
```

```
print(paste("Error %", (1-accuracy[1])*100, sep=": "))
```

```
## [1] "Error %: 25.4158712844287"
```

Create a model Using random forest on the training dataset.

Use the model to predict "classe" in the validation dataset and use the confusion matrix to compare the predicted versus the actual labels:

```
modelRF <- train(classe ~ ., data = training, method = "rf", trControl = trainControl(method = "cv", 5)
modelRF
```

```
## Random Forest
##
## 14718 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
```

```
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 11775, 11775, 11773, 11774, 11775
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9915748  0.9893409
##   27    0.9912351  0.9889117
##   52    0.9848483  0.9808304
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
predRF <- predict(modelRF, validation)

# Use confusion matrix to get estimate of out-of-sample error
confusionMatrix(validation$classe, predRF)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1021    0    0    0    0
##          B    0  758    0    0    0
##          C    0    0  624    0    0
##          D    0    0    0  587    0
##          E    0    0    0    0  677
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.999, 1)
##     No Information Rate : 0.2784
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2784   0.2067   0.1702   0.1601   0.1846
## Detection Rate         0.2784   0.2067   0.1702   0.1601   0.1846
## Detection Prevalence   0.2784   0.2067   0.1702   0.1601   0.1846
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

Random Forest accuracy and out of sample error

```
accuracy <-postResample(predRF, validation$classe)
print(paste("Accuracy %", accuracy[1]*100, sep=": "))
```

```
## [1] "Accuracy %: 100"
```

```
print(paste("Error %", (1-accuracy[1])*100, sep=": "))
```

```
## [1] "Error %: 0"
```

Predicting the "classe" variable in the test data

The accuracy of the random forest model is 100%. Use this model to predict the "classe" variable in the test dataset.

```
#predicTest <- predict(modelRF, testing[, -length(names(testing))])
predicTest <- predict(modelRF, testing)
predicTest
```

```
##  [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
```

```
pml_write_files = function(x){
      n = length(x)
      for(i in 1:n){
            filename = paste0("problem_id_",i,".txt")
            write.table(x[i], file = filename, quote = FALSE, row.names = FALSE, col.names = FALSE)
      }
}

# create prediction files to submit
pml_write_files(predicTest)
```