Practical Machine Learning: Prediction Assignment Writeup

1.0 Introduction

1.1 Purpose

To use data from accelerometers worn on the belt, forearm, arm, or dumbell of six participants to identify how well each participant performed their activity.

1.2 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. Common smartphone apps enable individuals to use this data to track how much of a particular activity they perform. So far, smartphone apps are not able to track how well individuals perform their activities.

In this analysis we seek to use R to categorize each individual's experience with each activity to a letter grade, analogous to grades in primary school.

1.3 R Environment

In order to complete this project, we must load the following libraries.

```
library(caret)
```

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

```
library(rpart)
library(rpart.plot)
library(rattle)
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 3.4.1 Copyright (c) 2006-2014 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(randomForest)
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

In order to be able to reproduce the results on demand, we must set the 'seed' used by the random number generator (Yes, I know that means the random numbers are not really random).

```
set.seed(1234)
```

2.0 Data

2.1 Data Sources

The 'Human Activity Recognition' (HAR) group has kindly provided the data for this project. You can read more details about this group on their website. We will load the two files, 'training' and 'testing'. Note that the 'test' file provided is intended for a test by an external party (what software developers typically call an 'acceptanct' test) and that we still need to partion the 'training' data into 'train' and'test' partitions so that we can use the 'test' partition for our own test -- what developers commonly call a 'unit test'.

```
trainUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
testUrl <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
train <- read.csv(url(trainUrl), na.strings=c("NA","#DIV/0!",""))
test <- read.csv(url(testUrl), na.strings=c("NA","#DIV/0!",""))
```

2.2 Partition the Training Dataset for Cross-Validation

Here, I partition to the 'pml-training.csv' file to create the file 'train' and 'test' files that I will use as part of developing the analysis. This technique is called 'cross-validation'.

Cross-validation is a statistical technique to estimate the prediction error rate by splitting the data into training and test datasets. A prediction model is obtained using the training set, while the test set is held for empirical error estimation.

Unlike the 'statistical analysis' and 'Regression' courses in the Data Science certificate program, where we calculated the confidence interval using the same dataset we used to calculate the model, the confusion-matrix that we will calculate below, will use the 'test' data that we partition from the pml-training.csv file (and do NOT use in calculating the model) to evaluate the accuracy range for the prediction models.

```
inTrain <- createDataPartition(y=train$classe, p=0.6, list=FALSE)
myTrain <- train[inTrain,]; myTest <- train[-inTrain,]
```

2.3 Clean Training Data Partition

The device vendors did create their device log files with the intent of supplying data just to R analysis programs. As a result, we will have to 'clean' the data before we can analyze it. 'Cleaning' performs multiple functions:

- Step-1 - Remove variables that cannot aid in categorizing activity performance because they have too little variance from one observation to the next
- Step-2 - Remove variables that could have an accidental corelation
- Step-3 - Remove variables that a majority of users do not collect, and that would therefore not provide a useful basis for categorizing future users going forward

Since this report will be publicly available on GitHub, we would also remove any personal identifying attributes (PIA), but the HAR group has already done that for us.

Step-1: Remove NearZeroVariance variables
'Near Zero Variance variables' add nothing to the analysis because they did not change over the data collection period.

```
NZVvars <- names(myTrain) %in% c("new_window", "kurtosis_roll_belt", "kurtosis_pitch_belt",
                                 "kurtosis_yaw_belt", "skewness_roll_belt", "skewness_roll_belt.1",
                                 "skewness_yaw_belt", "max_yaw_belt", "min_yaw_belt",
                                 "amplitude_yaw_belt", "avg_roll_arm", "stddev_roll_arm",
                                 "var_roll_arm", "avg_pitch_arm", "stddev_pitch_arm",
                                 "var_pitch_arm", "avg_yaw_arm", "stddev_yaw_arm",
                                 "var_yaw_arm", "kurtosis_roll_arm", "kurtosis_pitch_arm",
                                 "kurtosis_yaw_arm", "skewness_roll_arm", "skewness_pitch_arm",
                                 "skewness_yaw_arm", "max_roll_arm", "min_roll_arm",
                                 "min_pitch_arm", "amplitude_role_arm", "amplitude_pitch_arm",
                                 "kurtosis_roll_dumbbell", "kurtosis_picth_dumbbell", "kurtosis_yaw_dumbbell",
                                 "skewness_roll_dumbbell", "skewness_pitch_dumbbell", "skewness_yaw_dumbbell",
                                 "max_yaw_dumbbell", "min_yaw_dumbbell", "amplitude_yaw_dumbbell",
                                 "kurtosis_roll_forearm", "kurtosis_picth_forearm", "kurtosis_yaw_forearm",
                                 "skewness_roll_forearm", "skewness_pitch_forearm", "skewness_yaw_forearm",
                                 "max_roll_forearm", "max_yaw_forearm", "min_roll_forearm",
                                 "min_yaw_forearm", "amplitude_roll_forearm", "amplitude_yaw_forearm",
                                 "avg_roll_forearm", "stddev_roll_forearm", "var_roll_forearm",
                                 "avg_pitch_forearm", "stddev_pitch_forearm", "var_pitch_forearm",
                                 "avg_yaw_forearm", "stddev_yaw_forearm", "var_yaw_forearm")
myTrain <- myTrain[!NZVvars]
```

Step-2: Remove the ID column
The 'ID' column is a record identifier. We would not know how to interpret any relationship that the analysis may find to a generated sequence number, so we are removing the ID column from the dataset. Specifically, any relationship to ID would indicate a relationship to the order in which observations were recorded, and if such a relationship were found, that would mean we are missing at least one attribute. That would not help this analysis since we seek to assess performance based on the attributes that are being collected.

```
myTrain <- myTrain[c(-1)]
```

Step-3: Remove columns that are 60% or more 'NA'
The course notes used 60% as an example of threshold, so I have used the same threshold. Rationale: If a majority of device users do not collect the attribute, we cannot rely on the attribute to categrize the performance of new users going forward.

```
cntTrain <- length(myTrain)
for(i in cntTrain:1) { #start with the last column and go forwards
    if(sum(is.na(myTrain[,i])) / nrow(myTrain) >= 0.6) { #if more than 60% NA
        myTrain <- myTrain[ , -i] #remove the column
    }
}
dim(myTrain)
```

```
## [1] 11776    58
```

2.4 Clean the Testing Partition

We now perform the same cleaning steps on the unit test data that I partitioned from the pml-training.csv file. I don't break out the steps since you have already seen them.

```
myTest <- myTest[!NZVvars] #remove variables with near zero variance
myTest <- myTest[c(-1)] #remove the ID column

cntTest <- length(myTest)
for(i in cntTest:1) { #start with the last column and go forwards
    if(sum(is.na(myTest[,i])) / nrow(myTest) >= 0.6) { #if more than 60% NA
        myTest <- myTest[ , -i] #remove the column
    }
}
dim(myTest)
```

```
## [1] 7846   58
```

2.5 Clean the Acceptance Test Data Provided

In this section we clean the acceptance test data provided in the file 'pml-testing.csv'. Because this is a new dataset, there are few additional considerations:

- Step-1 - We do have to apply the same cleaning steps as we applied to the training data
- Step-2 - The file 'pml-testing.csv' contains a column 'problem ID' that does not appear in the trainging data. We remove this column as the model built from the

training data would not use it.
- Step-3 - The R 'read.csv' process defaults the class based on the initial data values it reads for each column. This can lead to inconsistencies between different extracts from the same source, when, for example, a character field just happens to contain all numerals. For this reason, we need to check that R is able to coerce subsequent extracts from the same source to the same classes for each column. If the coercion fails, we know that the data has really changed, and our analysis does not fail just because 'read.csv' inferred the column class differently.

Step-1: Clean the 'Test' Data
We must also apply the steps applied to clean the 'Training' data, to the 'Test' data for the acceptance test.

```
test <- test[!NZVvars] #remove variables with near zero variance
test <- test[c(-1)] #remove the ID column

cntTest <- length(test)
for(i in cntTest:1) { #start with the last column and go forwards
    if(sum(is.na(test[,i])) / nrow(test) >= 0.6) { #if more than 60% NA
        test <- test[ , -i] #remove the column
    }
}
```

Step-2: Remove 'problem ID' column
The 'Testing' dataset contains an additional column for 'problem ID' that is not part of the 'Training' dataset and that does not appear to contain information that is relevant to our analysis.

```
test <- test[-length(test)] #remove the 'problem ID' column not found in the Training dataset
```

Step-3: Force the 'Testing' dataset columns to have the same class as the 'Training' dataset
The RandomForest algorithm may fail on the 'Testing' dataset if the classes are not consistent with the 'Training' dataset. At a minimum, we prefer that the class coercion fail rather than the RandomForest procedure, because if the class coercion fails we know the data is really different.

Assumption: The acceptance test dataset (pml-testing.csv) has the same variables, in the same order.

```
test <- rbind(myTrain[2, -58], test) #rbind will throw an error if the class coercion did not work
test <- test[-1,] #remove the row we just bound
```

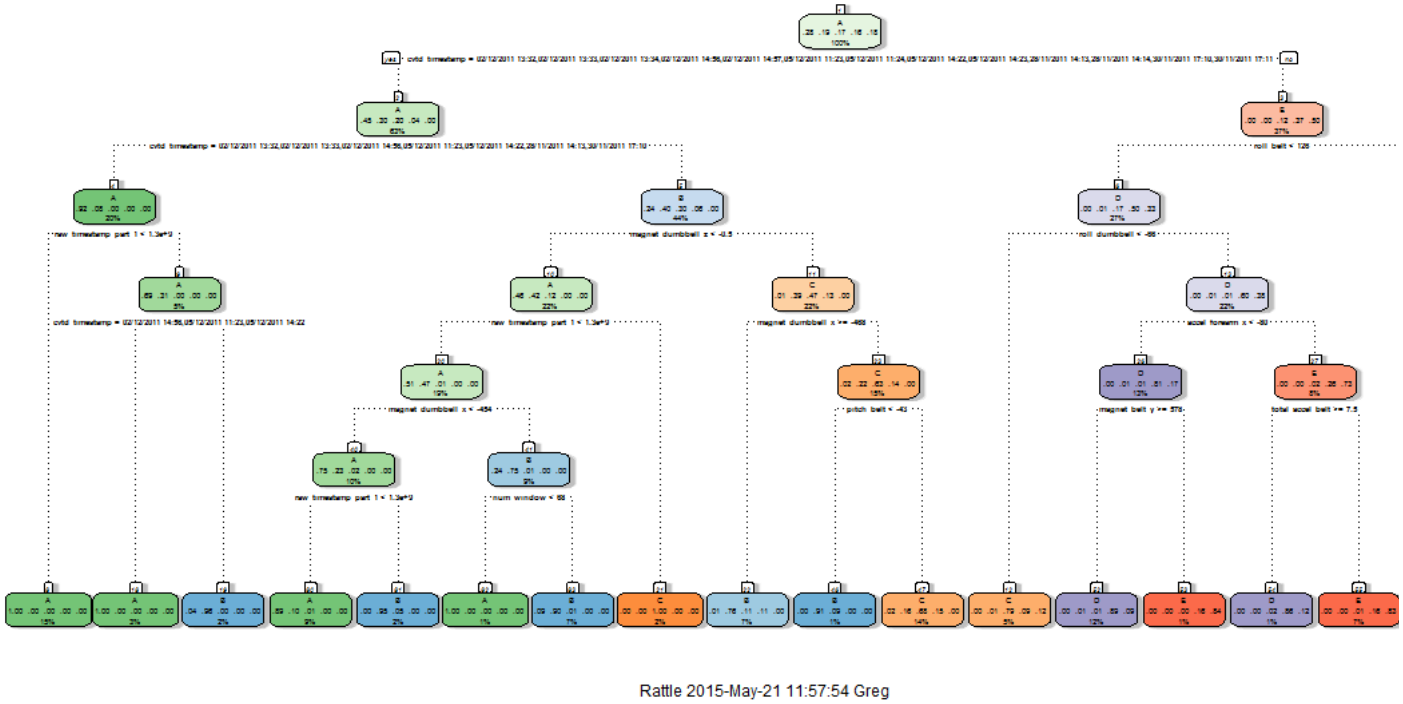
Apply Machine Learning

I will apply two methods of machine learning. The first, 'decision tree' is less accurate in execution but produces a useful diagram that will help readers of this report interpret the results. The second, 'random forest', is the method that will use to assess individual perormance as it is more accurate.

Decision Tree

Step-1: Build the Tree
I will use 'rpart', 'Recusive Partitioning and Regression Trees' package, to build the tree model and then call 'fancyRpartPlot', which is part of the 'Rattle' package, to display the tree.

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



Step-2: Evaluate the Tree
This step is analagous to a unit test: I will calculate a prediction based on the 'myTrain' data that I earlier partioned from the 'Training' dataset, and then calculate the 'confusion matrix'.

As you see in the output from the confusion matrix, the categorization of the observations in the unit test partition, based on the decision tree, is between 86.65% and 88.13% accurate. The matrix shows that even when wrong, the categorization is not very wrong for good performers: An 'A' may be categorized as a 'B' or 'C', but not as a 'D' or 'E'. However, the categorization varies more for poor performers ('D' and 'E').

Conclusion: The Decision Tree based on rpart is good-enough to build a diagram that explains the categorization logic, and would be sufficiently accurate for end-users who are good athletes ('A' and 'B' performance) and just want more feedback. However, the accuracy would NOT be sufficient for users who really need guidance ('D' and 'E' performance) as the risk of mis-categorization is higher for poor performance than for good performance.

```
confusionMatrix(predict(modFitTree, myTest, type="class"), myTest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2161   61    5    3    0
##          B   50 1271   95   64    0
##          C   21  177 1242  203   65
##          D    0    9   19  899   92
##          E    0    0    7  117 1285
##
## Overall Statistics
##
##                Accuracy : 0.8741
##                  95% CI : (0.8665, 0.8813)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8407
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9682   0.8373   0.9079   0.6991   0.8911
## Specificity            0.9877   0.9670   0.9281   0.9817   0.9806
## Pos Pred Value         0.9691   0.8588   0.7272   0.8822   0.9120
## Neg Pred Value         0.9874   0.9612   0.9795   0.9433   0.9756
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2754   0.1620   0.1583   0.1146   0.1638
## Detection Prevalence   0.2842   0.1886   0.2177   0.1299   0.1796
## Balanced Accuracy      0.9779   0.9021   0.9180   0.8404   0.9359
```

Random Forest

The 'randomForest' R package implements Breiman's random forest algorithm. The code has stood the test of time: the current R package is based on Breiman and Cutler's original Fortran code for classification and regression.

```
modFitForest <- randomForest(classe ~ ., data=myTrain)
predictionsForest <- predict(modFitForest, myTest, type="class")
```

As you see in the output from the confusion matrix, the categorization of the observations in the unit test partition, bansed on the randomForest, is between 99.7% and 99.9% accurate. More importantly, we see from the matrix that even when wrong, the categorization is never off by more than one category -- a 'D' may be categorized as a 'C' or 'E' in a very few cases, but never more.

Conclusion: People who really need guidance on their performance of exercise activities (those who score 'D' and 'E') could use the randomForest assessment to tell them for which activities they need to improve.

```
confusionMatrix(predictionsForest, myTest$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2232    2    0    0    0
##          B    0 1516    4    0    0
##          C    0    0 1362    5    0
##          D    0    0    2 1280    0
##          E    0    0    0    1 1442
##
## Overall Statistics
##
##                Accuracy : 0.9982
##                  95% CI : (0.997, 0.999)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9977
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9987   0.9956   0.9953   1.0000
## Specificity            0.9996   0.9994   0.9992   0.9997   0.9998
## Pos Pred Value         0.9991   0.9974   0.9963   0.9984   0.9993
## Neg Pred Value         1.0000   0.9997   0.9991   0.9991   1.0000
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2845   0.1932   0.1736   0.1631   0.1838
## Detection Prevalence   0.2847   0.1937   0.1742   0.1634   0.1839
## Balanced Accuracy      0.9998   0.9990   0.9974   0.9975   0.9999
```

Acceptance Test

In this section, I apply the randomForest model created above to the Acceptance test data, 'pml-testing.csv', and generate the test results to be submitted, using the file creation code provided in the assignment.

```
predictAcceptance <- predict(modFitForest, test, type="class")
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)
    }
}
pml_write_files(predictAcceptance)
```