Machine Learning Project

## 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>

The data for this project come from this source: <http://groupware.les.inf.puc-rio.br/har>. If you use the document you create for this class for any purpose please cite them as they have been very generous in allowing their data to be used for this kind of assignment.

## What you should submit

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.  
1. Your submission should consist of a link to a Github repo with your R markdown and compiled HTML file describing your analysis. Please constrain the text of the writeup to < 2000 words and the number of figures to be less than 5. It will make it easier for the graders if you submit a repo with a gh-pages branch so the HTML page can be viewed online (and you always want to make it easy on graders :-).  
2. You should also apply your machine learning algorithm to the 20 test cases available in the test data above. Please submit your predictions in appropriate format to the programming assignment for automated grading. See the programming assignment for additional details.  
##Reproducibility  
Due to security concerns with the exchange of R code, your code will not be run during the evaluation by your classmates. Please be sure that if they download the repo, they will be able to view the compiled HTML version of your analysis.

## Data Processing:

Before looking at data, we want to install the appropriate packages and set the seed to ensure the results are reproducible.

library(caret)

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

library(randomForest)

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

library(rpart)  
library(rpart.plot)  
library(RColorBrewer)  
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.

set.seed(12345)

### Loading and prepping the data:

Due to firewall issues, the data files have been downloaded to a local drive and will be pulled into RStudio from the local location. The source of the data can be found in the assignment outline listed above.

training\_raw <- read.csv(("pml-training.csv"), na.strings=c("NA","#DIV/0!", ""))  
testing\_raw <- read.csv(("pml-testing.csv"), na.strings=c("NA","#DIV/0!", ""))

Next, we need to partition the training data into 60% training and 40% testing

training\_in <-createDataPartition(y=training\_raw$classe, p=0.60, list=FALSE)  
training\_working <-training\_raw[training\_in,]; testing\_working <-training\_raw[-training\_in,]  
dim(training\_raw); dim(training\_working); dim(testing\_working)

## [1] 19622 160

## [1] 11776 160

## [1] 7846 160

The last step of prepping the data is to clean the data set up. First, we will look for near zero variance variables.

training\_nzv <- nearZeroVar(training\_working, saveMetrics=TRUE)  
nzvvars <- names(training\_working) %in% c("new\_window", "kurtosis\_roll\_belt", "kurtosis\_picth\_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\_picth\_arm",  
"kurtosis\_yaw\_arm", "skewness\_roll\_arm", "skewness\_pitch\_arm", "skewness\_yaw\_arm",  
"max\_roll\_arm", "min\_roll\_arm", "min\_pitch\_arm", "amplitude\_roll\_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")  
training\_working <-training\_working[!nzvvars]  
dim(training\_working)

## [1] 11776 100

Next, we will eliminate the first ID variable so as it will not interfere with algorithms to be run later in this analysis

training\_working <- training\_working[c(-1)]

Now, to clean up the variables with more than 60% NAs for data.

training\_working2 <- training\_working #creating another subset to iterate in loop  
for(i in 1:length(training\_working)) { #for every column in the training dataset  
 if( sum( is.na( training\_working[, i] ) ) /nrow(training\_working) >= .6 ) { #if n?? NAs > 60% of total observations  
 for(j in 1:length(training\_working2)) {  
 if( length( grep(names(training\_working[i]), names(training\_working2)[j]) ) ==1) { #if the columns are the same:  
 training\_working2 <- training\_working2[ , -j] #Remove that column  
 }   
 }   
 }  
}  
#To check the new set of observations  
dim(training\_working2)

## [1] 11776 58

training\_working<-training\_working2  
rm(training\_working2)

Now to repeat the same clean up, but on the testingdata\_raw and testingdata\_working data sets.

clean1 <-colnames(training\_working)  
clean2 <-colnames(training\_working[,-58])#classe already removed  
testing\_working<-testing\_working[clean1]  
testing\_raw <- testing\_raw[clean2]  
dim(testing\_working)

## [1] 7846 58

dim(testing\_raw)

## [1] 20 57

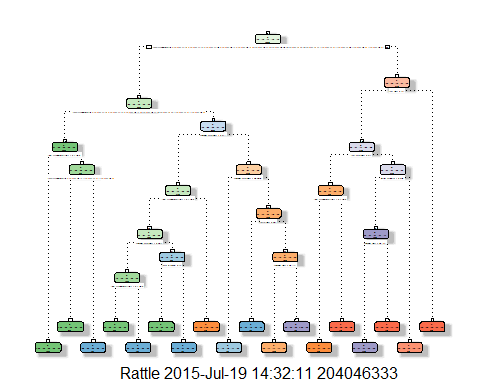
We also need to coerce the data into the same type:

for (i in 1:length(testing\_raw) ) {  
 for(j in 1:length(training\_working)) {  
 if( length( grep(names(training\_working[i]), names(testing\_raw)[j]) ) ==1) {  
 class(testing\_raw[j]) <- class(training\_working[i])  
 }   
 }   
}  
#And a check:  
testing\_raw <- rbind(training\_working[2, -58] , testing\_raw) #note row 2 can be removed:  
testing\_raw <- testing\_raw[-1,]

## ML Algorithms - Decision Tree

Creating a decision tree with fancy

modFitA1 <-rpart(classe ~ ., data=training\_working, method="class")  
fancyRpartPlot(modFitA1)



# Prediction

Create a confusion matrix to test the results:

predictionsA1 <- predict(modFitA1, testing\_working, type="class")  
confusionMatrix(predictionsA1,testing\_working$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2150 60 7 1 0  
## B 61 1260 69 64 0  
## C 21 188 1269 143 4  
## D 0 10 14 857 78  
## E 0 0 9 221 1360  
##   
## Overall Statistics  
##   
## Accuracy : 0.8789   
## 95% CI : (0.8715, 0.8861)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.8468   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9633 0.8300 0.9276 0.6664 0.9431  
## Specificity 0.9879 0.9693 0.9450 0.9845 0.9641  
## Pos Pred Value 0.9693 0.8666 0.7809 0.8936 0.8553  
## Neg Pred Value 0.9854 0.9596 0.9841 0.9377 0.9869  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2740 0.1606 0.1617 0.1092 0.1733  
## Detection Prevalence 0.2827 0.1853 0.2071 0.1222 0.2027  
## Balanced Accuracy 0.9756 0.8997 0.9363 0.8254 0.9536

## ML Algorithms for Random Forest predictions

modFitB1 <- randomForest(classe ~., data=training\_working, method="class")  
#predicting in sample error:  
predictionsB1 <-predict(modFitB1, testing\_working, type="class")  
#using confusion matrix to test results:  
confusionMatrix(predictionsB1, testing\_working$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2231 2 0 0 0  
## B 1 1516 1 0 0  
## C 0 0 1366 3 0  
## D 0 0 1 1282 1  
## E 0 0 0 1 1441  
##   
## Overall Statistics  
##   
## Accuracy : 0.9987   
## 95% CI : (0.9977, 0.9994)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9984   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9996 0.9987 0.9985 0.9969 0.9993  
## Specificity 0.9996 0.9997 0.9995 0.9997 0.9998  
## Pos Pred Value 0.9991 0.9987 0.9978 0.9984 0.9993  
## Neg Pred Value 0.9998 0.9997 0.9997 0.9994 0.9998  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2843 0.1932 0.1741 0.1634 0.1837  
## Detection Prevalence 0.2846 0.1935 0.1745 0.1637 0.1838  
## Balanced Accuracy 0.9996 0.9992 0.9990 0.9983 0.9996

As you can see, Random Forest results gives a much more accurate answer. Confusion Matrix gives an accuracy of 87.89%, whereas the Random Forest gives 99.87%. Next, we will generate predicitons for all 20 test cases using the Random Forest model.

# Create Files for Documentation:

Using the out of sample test set error, we can use the Random Forest, which gave a better result for the in sample data and create our files.

predictionsB2 <-predict(modFitB1, testing\_raw, type="class")  
  
pml\_write=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(predictionsB2)