Practical Machine Learning Assignment

### 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.

#### Data Preprocessing

##### Loading data

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

training\_set <- read.csv(url("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"))  
testing\_set <- read.csv(url("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"))

##### Partitioning the training dataset into training and testing datasets (due to small sample in testing set)

training\_label <- createDataPartition(training\_set$classe, p=0.6, list=FALSE)  
train <- training\_set[training\_label,]  
test <- training\_set[-training\_label,]

##### Data cleaning - Preprocessing to reduce the number of variables used for analysis.

#Remove variables with nearly zero variance  
NZV <- nearZeroVar(train)  
train <- train[,-NZV]  
test <- test[,-NZV]  
  
#Remove variables with a lot of NA terms  
label <- apply(train, 2, function(x) mean(is.na(x))) > 0.90  
train <- train[, -which(label, label == FALSE)]  
test <- test[, -which(label, label == FALSE)]  
  
#Remove other 5 variables used for identification  
train <- train[ , -(1:5)]  
test <- test[ , -(1:5)]

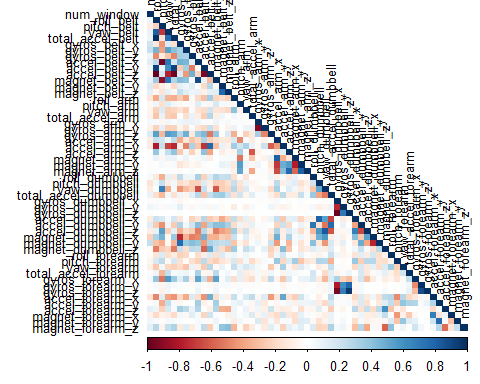
##### The number of variables have reduced from 160 to 54.

#### Exploratory Data Analysis

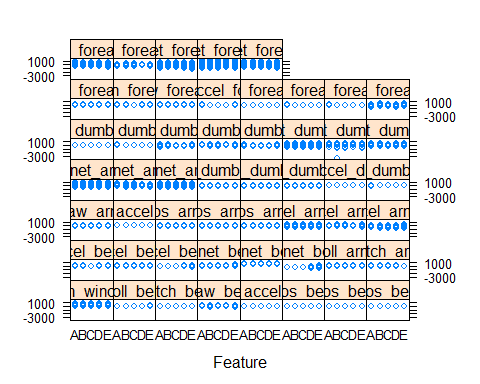
#Correlation matrix between features  
library(corrplot)

## corrplot 0.84 loaded

corrMat <- cor(train[,-54])  
corrplot(corrMat, method = "color", type = "lower", tl.cex = 0.8, tl.col = rgb(0,0,0))



# Based on the plot, the darker gradient of color corresponds to having a higher correlation.  
  
#Correlation matrix between features and outcome  
featurePlot(train[,-54], train[,54], "strip")



# Based on the plot, each feature has relatively the same distribution among the 5 outcomes.

#### Prediction Model Selection - We would use 4 methods (Decision Tree, Random Forest, Generalised Boosting model and Extreme Boosting Model) on the training set and choose the model with the best accuracy to predict the outcome variable in the testing set.

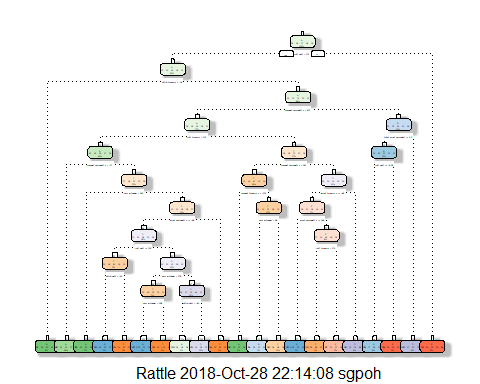
#### Decision Tree

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

## Rattle: A free graphical interface for data science with R.  
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

set.seed(12345)  
modelDT <- rpart(classe ~ ., data = train, method = "class")  
fancyRpartPlot(modelDT)

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



predictDT <- predict(modelDT, test, type = "class")  
  
# Performance of the model on the test data set  
confusionMatrix(predictDT, test$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2004 343 56 139 98  
## B 56 772 48 26 93  
## C 26 121 1153 204 115  
## D 123 177 76 843 175  
## E 23 105 35 74 961  
##   
## Overall Statistics  
##   
## Accuracy : 0.7307   
## 95% CI : (0.7207, 0.7405)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.6576   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.8978 0.50856 0.8428 0.6555 0.6664  
## Specificity 0.8867 0.96476 0.9281 0.9160 0.9630  
## Pos Pred Value 0.7591 0.77588 0.7122 0.6047 0.8022  
## Neg Pred Value 0.9562 0.89111 0.9655 0.9313 0.9276  
## Prevalence 0.2845 0.19347 0.1744 0.1639 0.1838  
## Detection Rate 0.2554 0.09839 0.1470 0.1074 0.1225  
## Detection Prevalence 0.3365 0.12682 0.2063 0.1777 0.1527  
## Balanced Accuracy 0.8923 0.73666 0.8855 0.7858 0.8147

#### Random Forest

library(randomForest)

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##   
## Attaching package: 'randomForest'

## The following object is masked from 'package:rattle':  
##   
## importance

## The following object is masked from 'package:ggplot2':  
##   
## margin

modelRF <- randomForest(classe ~ ., data = train, importance = TRUE)  
modelRF

##   
## Call:  
## randomForest(formula = classe ~ ., data = train, importance = TRUE)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 7  
##   
## OOB estimate of error rate: 0.5%  
## Confusion matrix:  
## A B C D E class.error  
## A 3346 1 0 0 1 0.0005973716  
## B 7 2269 3 0 0 0.0043878894  
## C 0 16 2035 3 0 0.0092502434  
## D 0 0 18 1910 2 0.0103626943  
## E 0 0 0 8 2157 0.0036951501

predictRF <- predict(modelRF, test, type = "class")  
  
# Performance of the model on the test data set  
confusionMatrix(predictRF, test$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2232 5 0 0 0  
## B 0 1512 11 0 0  
## C 0 1 1357 8 0  
## D 0 0 0 1277 7  
## E 0 0 0 1 1435  
##   
## Overall Statistics  
##   
## Accuracy : 0.9958   
## 95% CI : (0.9941, 0.9971)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9947   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 1.0000 0.9960 0.9920 0.9930 0.9951  
## Specificity 0.9991 0.9983 0.9986 0.9989 0.9998  
## Pos Pred Value 0.9978 0.9928 0.9934 0.9945 0.9993  
## Neg Pred Value 1.0000 0.9991 0.9983 0.9986 0.9989  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2845 0.1927 0.1730 0.1628 0.1829  
## Detection Prevalence 0.2851 0.1941 0.1741 0.1637 0.1830  
## Balanced Accuracy 0.9996 0.9972 0.9953 0.9960 0.9975

#### Generalised Boosting Model

library(caret)  
set.seed(12345)  
controlGBM <- trainControl(method = "repeatedcv", number = 5, repeats = 1, verboseIter = FALSE)  
modelGBM <- train(classe ~ ., data = train, trControl = controlGBM, method = "gbm", verbose = FALSE)  
modelGBM$finalModel

## A gradient boosted model with multinomial loss function.  
## 150 iterations were performed.  
## There were 53 predictors of which 41 had non-zero influence.

predictGBM <- predict(modelGBM, test)  
confusionMatrix(predictGBM, test$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2227 24 0 0 1  
## B 4 1459 20 4 8  
## C 0 34 1345 16 3  
## D 1 0 2 1265 17  
## E 0 1 1 1 1413  
##   
## Overall Statistics  
##   
## Accuracy : 0.9825   
## 95% CI : (0.9794, 0.9853)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9779   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 0.9978 0.9611 0.9832 0.9837 0.9799  
## Specificity 0.9955 0.9943 0.9918 0.9970 0.9995  
## Pos Pred Value 0.9889 0.9759 0.9621 0.9844 0.9979  
## Neg Pred Value 0.9991 0.9907 0.9964 0.9968 0.9955  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2838 0.1860 0.1714 0.1612 0.1801  
## Detection Prevalence 0.2870 0.1905 0.1782 0.1638 0.1805  
## Balanced Accuracy 0.9967 0.9777 0.9875 0.9903 0.9897

#### Extreme Gradient Boosting - We have chosen this additional model due to its good accuracy and having a good reputation as a leading model in Kaggle competitions.

library(xgboost)

##   
## Attaching package: 'xgboost'

## The following object is masked from 'package:rattle':  
##   
## xgboost

set.seed(12345)  
controlXGB <- trainControl(method = "cv", number = 5, allowParallel = TRUE)  
modelXGB <- train(classe ~ ., data = train, method = "xgbTree", trControl = controlXGB)  
modelXGB

## eXtreme Gradient Boosting   
##   
## 11776 samples  
## 53 predictor  
## 5 classes: 'A', 'B', 'C', 'D', 'E'   
##   
## No pre-processing  
## Resampling: Cross-Validated (5 fold)   
## Summary of sample sizes: 9421, 9421, 9422, 9420, 9420   
## Resampling results across tuning parameters:  
##   
## eta max\_depth colsample\_bytree subsample nrounds Accuracy   
## 0.3 1 0.6 0.50 50 0.8104606  
## 0.3 1 0.6 0.50 100 0.8792454  
## 0.3 1 0.6 0.50 150 0.9159289  
## 0.3 1 0.6 0.75 50 0.8124150  
## 0.3 1 0.6 0.75 100 0.8794997  
## 0.3 1 0.6 0.75 150 0.9173734  
## 0.3 1 0.6 1.00 50 0.8084241  
## 0.3 1 0.6 1.00 100 0.8761884  
## 0.3 1 0.6 1.00 150 0.9131270  
## 0.3 1 0.8 0.50 50 0.8122445  
## 0.3 1 0.8 0.50 100 0.8843402  
## 0.3 1 0.8 0.50 150 0.9183075  
## 0.3 1 0.8 0.75 50 0.8136032  
## 0.3 1 0.8 0.75 100 0.8818772  
## 0.3 1 0.8 0.75 150 0.9121930  
## 0.3 1 0.8 1.00 50 0.8125850  
## 0.3 1 0.8 1.00 100 0.8784810  
## 0.3 1 0.8 1.00 150 0.9158446  
## 0.3 2 0.6 0.50 50 0.9499820  
## 0.3 2 0.6 0.50 100 0.9845449  
## 0.3 2 0.6 0.50 150 0.9932914  
## 0.3 2 0.6 0.75 50 0.9500679  
## 0.3 2 0.6 0.75 100 0.9864133  
## 0.3 2 0.6 0.75 150 0.9938860  
## 0.3 2 0.6 1.00 50 0.9523606  
## 0.3 2 0.6 1.00 100 0.9866679  
## 0.3 2 0.6 1.00 150 0.9942254  
## 0.3 2 0.8 0.50 50 0.9543989  
## 0.3 2 0.8 0.50 100 0.9868375  
## 0.3 2 0.8 0.50 150 0.9938856  
## 0.3 2 0.8 0.75 50 0.9503216  
## 0.3 2 0.8 0.75 100 0.9872624  
## 0.3 2 0.8 0.75 150 0.9938858  
## 0.3 2 0.8 1.00 50 0.9518506  
## 0.3 2 0.8 1.00 100 0.9876019  
## 0.3 2 0.8 1.00 150 0.9948199  
## 0.3 3 0.6 0.50 50 0.9876014  
## 0.3 3 0.6 0.50 100 0.9959238  
## 0.3 3 0.6 0.50 150 0.9970279  
## 0.3 3 0.6 0.75 50 0.9886211  
## 0.3 3 0.6 0.75 100 0.9965181  
## 0.3 3 0.6 0.75 150 0.9977071  
## 0.3 3 0.6 1.00 50 0.9897246  
## 0.3 3 0.6 1.00 100 0.9971976  
## 0.3 3 0.6 1.00 150 0.9979619  
## 0.3 3 0.8 0.50 50 0.9880266  
## 0.3 3 0.8 0.50 100 0.9963483  
## 0.3 3 0.8 0.50 150 0.9975374  
## 0.3 3 0.8 0.75 50 0.9876869  
## 0.3 3 0.8 0.75 100 0.9966882  
## 0.3 3 0.8 0.75 150 0.9979619  
## 0.3 3 0.8 1.00 50 0.9897249  
## 0.3 3 0.8 1.00 100 0.9964333  
## 0.3 3 0.8 1.00 150 0.9984714  
## 0.4 1 0.6 0.50 50 0.8420529  
## 0.4 1 0.6 0.50 100 0.9061644  
## 0.4 1 0.6 0.50 150 0.9366504  
## 0.4 1 0.6 0.75 50 0.8424764  
## 0.4 1 0.6 0.75 100 0.9058252  
## 0.4 1 0.6 0.75 150 0.9356320  
## 0.4 1 0.6 1.00 50 0.8436639  
## 0.4 1 0.6 1.00 100 0.9070980  
## 0.4 1 0.6 1.00 150 0.9341872  
## 0.4 1 0.8 0.50 50 0.8447678  
## 0.4 1 0.8 0.50 100 0.9085421  
## 0.4 1 0.8 0.50 150 0.9387735  
## 0.4 1 0.8 0.75 50 0.8425608  
## 0.4 1 0.8 0.75 100 0.9073523  
## 0.4 1 0.8 0.75 150 0.9367357  
## 0.4 1 0.8 1.00 50 0.8474007  
## 0.4 1 0.8 1.00 100 0.9051442  
## 0.4 1 0.8 1.00 150 0.9366504  
## 0.4 2 0.6 0.50 50 0.9695143  
## 0.4 2 0.6 0.50 100 0.9915081  
## 0.4 2 0.6 0.50 150 0.9958388  
## 0.4 2 0.6 0.75 50 0.9678164  
## 0.4 2 0.6 0.75 100 0.9918478  
## 0.4 2 0.6 0.75 150 0.9956690  
## 0.4 2 0.6 1.00 50 0.9701086  
## 0.4 2 0.6 1.00 100 0.9930367  
## 0.4 2 0.6 1.00 150 0.9970277  
## 0.4 2 0.8 0.50 50 0.9682403  
## 0.4 2 0.8 0.50 100 0.9912533  
## 0.4 2 0.8 0.50 150 0.9957540  
## 0.4 2 0.8 0.75 50 0.9686656  
## 0.4 2 0.8 0.75 100 0.9921027  
## 0.4 2 0.8 0.75 150 0.9957539  
## 0.4 2 0.8 1.00 50 0.9735056  
## 0.4 2 0.8 1.00 100 0.9939710  
## 0.4 2 0.8 1.00 150 0.9968579  
## 0.4 3 0.6 0.50 50 0.9933763  
## 0.4 3 0.6 0.50 100 0.9972826  
## 0.4 3 0.6 0.50 150 0.9982167  
## 0.4 3 0.6 0.75 50 0.9929517  
## 0.4 3 0.6 0.75 100 0.9971975  
## 0.4 3 0.6 0.75 150 0.9981317  
## 0.4 3 0.6 1.00 50 0.9932914  
## 0.4 3 0.6 1.00 100 0.9981318  
## 0.4 3 0.6 1.00 150 0.9984715  
## 0.4 3 0.8 0.50 50 0.9923573  
## 0.4 3 0.8 0.50 100 0.9972823  
## 0.4 3 0.8 0.50 150 0.9979619  
## 0.4 3 0.8 0.75 50 0.9934612  
## 0.4 3 0.8 0.75 100 0.9977920  
## 0.4 3 0.8 0.75 150 0.9981319  
## 0.4 3 0.8 1.00 50 0.9943954  
## 0.4 3 0.8 1.00 100 0.9974522  
## 0.4 3 0.8 1.00 150 0.9982166  
## Kappa   
## 0.7598628  
## 0.8472086  
## 0.8936057  
## 0.7621606  
## 0.8474889  
## 0.8954675  
## 0.7571270  
## 0.8433243  
## 0.8900987  
## 0.7619490  
## 0.8536357  
## 0.8966414  
## 0.7637550  
## 0.8505095  
## 0.8888845  
## 0.7622963  
## 0.8462306  
## 0.8935291  
## 0.9367453  
## 0.9804499  
## 0.9915150  
## 0.9368498  
## 0.9828161  
## 0.9922664  
## 0.9397360  
## 0.9831352  
## 0.9926957  
## 0.9423186  
## 0.9833518  
## 0.9922660  
## 0.9371696  
## 0.9838867  
## 0.9922662  
## 0.9391055  
## 0.9843182  
## 0.9934479  
## 0.9843162  
## 0.9948443  
## 0.9962407  
## 0.9856069  
## 0.9955960  
## 0.9970998  
## 0.9870024  
## 0.9964554  
## 0.9974220  
## 0.9848553  
## 0.9953814  
## 0.9968851  
## 0.9844244  
## 0.9958111  
## 0.9974220  
## 0.9870033  
## 0.9954886  
## 0.9980665  
## 0.8000167  
## 0.8812382  
## 0.9198349  
## 0.8005356  
## 0.8808457  
## 0.9185658  
## 0.8019757  
## 0.8824637  
## 0.9167430  
## 0.8034236  
## 0.8842779  
## 0.9225372  
## 0.8006417  
## 0.8827828  
## 0.9199680  
## 0.8067457  
## 0.8799967  
## 0.9198711  
## 0.9614343  
## 0.9892586  
## 0.9947367  
## 0.9592957  
## 0.9896883  
## 0.9945214  
## 0.9621909  
## 0.9911921  
## 0.9962404  
## 0.9598244  
## 0.9889359  
## 0.9946294  
## 0.9603750  
## 0.9900107  
## 0.9946290  
## 0.9664837  
## 0.9923740  
## 0.9960257  
## 0.9916222  
## 0.9965628  
## 0.9977444  
## 0.9910848  
## 0.9964553  
## 0.9976369  
## 0.9915141  
## 0.9976370  
## 0.9980666  
## 0.9903326  
## 0.9965627  
## 0.9974221  
## 0.9917293  
## 0.9972072  
## 0.9976370  
## 0.9929109  
## 0.9967775  
## 0.9977442  
##   
## Tuning parameter 'gamma' was held constant at a value of 0  
##   
## Tuning parameter 'min\_child\_weight' was held constant at a value of 1  
## Accuracy was used to select the optimal model using the largest value.  
## The final values used for the model were nrounds = 150, max\_depth = 3,  
## eta = 0.4, gamma = 0, colsample\_bytree = 0.6, min\_child\_weight = 1  
## and subsample = 1.

predictXGB <- predict(modelXGB, test)  
  
# Performance of the model on the test data set  
confusionMatrix(predictXGB, test$classe)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction A B C D E  
## A 2232 1 0 0 0  
## B 0 1517 0 0 2  
## C 0 0 1368 0 0  
## D 0 0 0 1286 2  
## E 0 0 0 0 1438  
##   
## Overall Statistics  
##   
## Accuracy : 0.9994   
## 95% CI : (0.9985, 0.9998)  
## No Information Rate : 0.2845   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.9992   
## Mcnemar's Test P-Value : NA   
##   
## Statistics by Class:  
##   
## Class: A Class: B Class: C Class: D Class: E  
## Sensitivity 1.0000 0.9993 1.0000 1.0000 0.9972  
## Specificity 0.9998 0.9997 1.0000 0.9997 1.0000  
## Pos Pred Value 0.9996 0.9987 1.0000 0.9984 1.0000  
## Neg Pred Value 1.0000 0.9998 1.0000 1.0000 0.9994  
## Prevalence 0.2845 0.1935 0.1744 0.1639 0.1838  
## Detection Rate 0.2845 0.1933 0.1744 0.1639 0.1833  
## Detection Prevalence 0.2846 0.1936 0.1744 0.1642 0.1833  
## Balanced Accuracy 0.9999 0.9995 1.0000 0.9998 0.9986

##### Conclusion: As Extreme Gradient Boosting has the highest accuracy of 0.9995, we will use Extreme Gradient Boosting Model to predict the test data class variable.

#### Predicting test data output

predictXGB\_testingset <- predict(modelXGB, testing\_set)  
predictXGB\_testingset

## [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