Practical Machine Learning Project - Quantified Self Movement Data Analysis

Introduction

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, we will use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants to predict the manner in which they did the exercise.

Data Preprocessing

```r
library(caret)
```

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

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

```r
library(rpart)
library(rpart.plot)
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:ggplot2':
##
##     margin
```

```r
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```r
library(RColorBrewer)
library(rattle)
```

```
## Loading required package: tibble
```

```
## Loading required package: bitops
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.4.0 Copyright (c) 2006-2020 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
##
## Attaching package: 'rattle'
```

```
## The following object is masked from 'package:randomForest':
##
##     importance
```

Load the data

```r
trainRaw <- read.csv("pml-training.csv")
testRaw <- read.csv("pml-testing.csv")
dim(trainRaw)
```

```
## [1] 19622   160
```

```r
dim(testRaw)
```

```
## [1]  20 160
```

The training data set contains 19622 observations and 160 variables, while the testing data set contains 20 observations and 160 variables. The "classe" variable in the training set is the outcome to predict.

Clean the data

In this step, we will clean the data and get rid of observations with missing values as well as some meaningless variables.

```r
sum(complete.cases(trainRaw))
```

```
## [1] 406
```

First, we remove columns that contain NA missing values.

```r
trainRaw <- trainRaw[, colSums(is.na(trainRaw)) == 0]
testRaw <- testRaw[, colSums(is.na(testRaw)) == 0]
```

Next, we get rid of some columns that do not contribute much to the accelerometer measurements.

```r
classe <- trainRaw$classe
trainRemove <- grepl("^X|timestamp|window", names(trainRaw))
trainRaw <- trainRaw[, !trainRemove]
trainCleaned <- trainRaw[, sapply(trainRaw, is.numeric)]
trainCleaned$classe <- classe
testRemove <- grepl("^X|timestamp|window", names(testRaw))
testRaw <- testRaw[, !testRemove]
testCleaned <- testRaw[, sapply(testRaw, is.numeric)]
```

Now, the cleaned training data set contains 19622 observations and 53 variables, while the testing data set contains 20 observations and 53 variables. The "classe" variable is still in the cleaned training set.

Slice the data

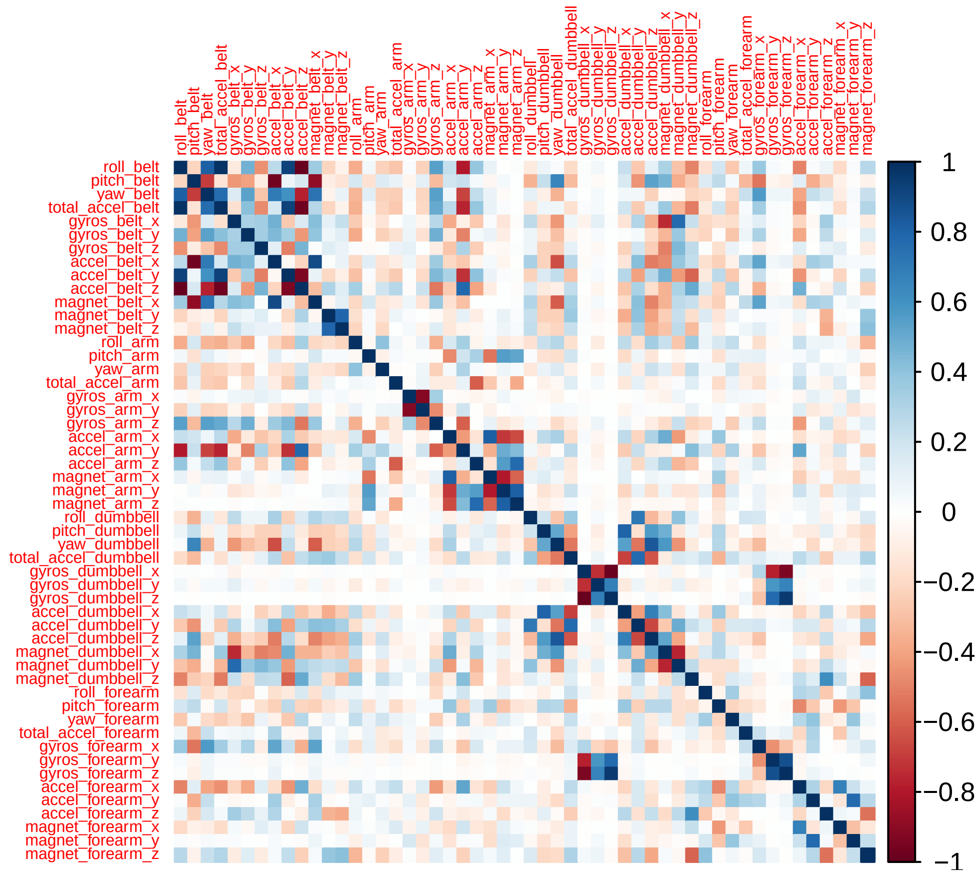
Then, we can split the cleaned training set into a pure training data set (70%) and a validation data set (30%). We will use the validation data set to conduct cross validation in future steps.

```r
set.seed(22519)
inTrain <- createDataPartition(trainCleaned$classe, p=0.70, list=F)
trainData <- trainCleaned[inTrain, ]
testData <- trainCleaned[-inTrain, ]
```

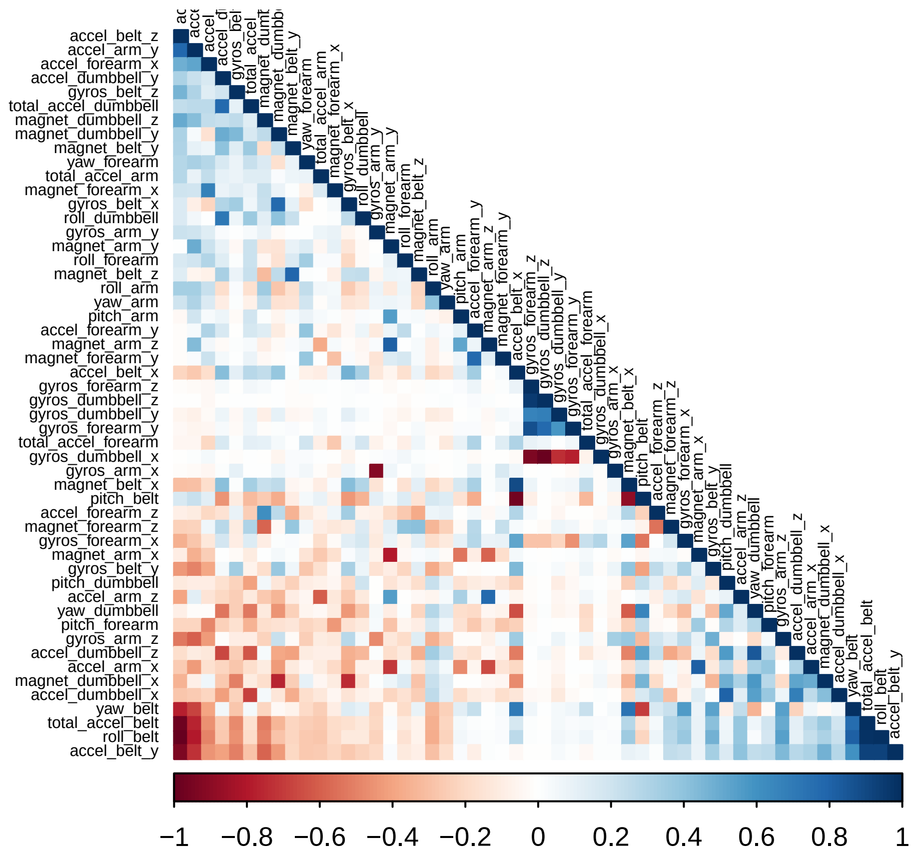
Correlation Analysis

A correlation among variables is analysed before proceeding to the modeling procedures.

```r
corrPlot <- cor(trainData[, -length(names(trainData))])
corrplot(corrPlot, method="color",  tl.cex = 0.5)
```




```r
corrplot(corrPlot, order = "FPC", method = "color", type = "lower",
         tl.cex = 0.5, tl.col = rgb(0, 0, 0))
```



Prediction Model Building

Three popular methods will be applied to model the regressions (in the Train dataset) and the best one (with higher accuracy when applied to the Test dataset) will be used for the quiz predictions. The methods are: Random Forests, Decision Tree and Generalized Boosted Model, as described below.

A Confusion Matrix is plotted at the end of each analysis to better visualize the accuracy of the models.

1. Random Forests

```
controlRf <- trainControl(method="cv", 5)
modelRf <- train(classe ~ ., data=trainData, method="rf", trControl=controlRf, ntree=250)
modelRf
```

```
## 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: 10988, 10989, 10989, 10991, 10991
```

```
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9912654  0.9889499
##   27    0.9916291  0.9894104
##   52    0.9842766  0.9801110
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

Then, we estimate the performance of the model on the validation data set.

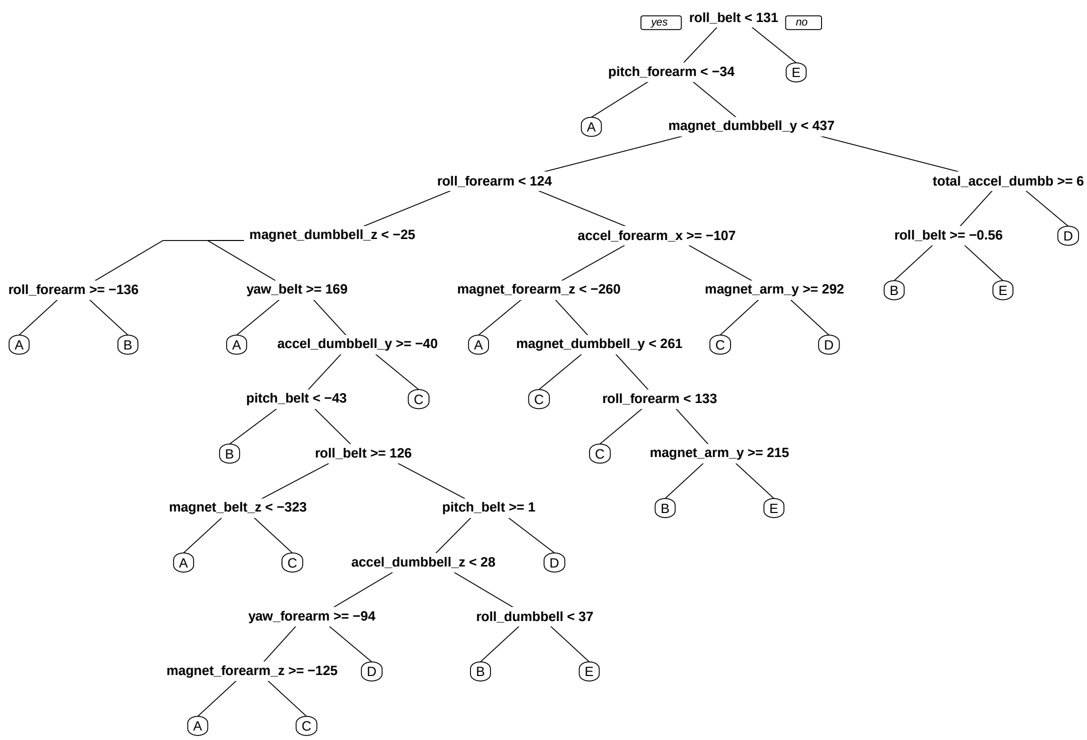
```r
predictRf <- predict(modelRf, testData)
confusionMatrix(table(testData$classe, predictRf))
```

```
## Confusion Matrix and Statistics
##
##    predictRf
##        A    B    C    D    E
##   A 1669    2    3    0    0
##   B    5 1130    3    1    0
##   C    0    4 1019    3    0
##   D    0    0   10  954    0
##   E    0    0    4    2 1076
##
## Overall Statistics
##
##                Accuracy : 0.9937
##                  95% CI : (0.9913, 0.9956)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.992
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9970   0.9947   0.9808   0.9938   1.0000
## Specificity            0.9988   0.9981   0.9986   0.9980   0.9988
## Pos Pred Value         0.9970   0.9921   0.9932   0.9896   0.9945
## Neg Pred Value         0.9988   0.9987   0.9959   0.9988   1.0000
## Prevalence             0.2845   0.1930   0.1766   0.1631   0.1828
## Detection Rate         0.2836   0.1920   0.1732   0.1621   0.1828
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9979   0.9964   0.9897   0.9959   0.9994
```

Accuracy : 0.9937

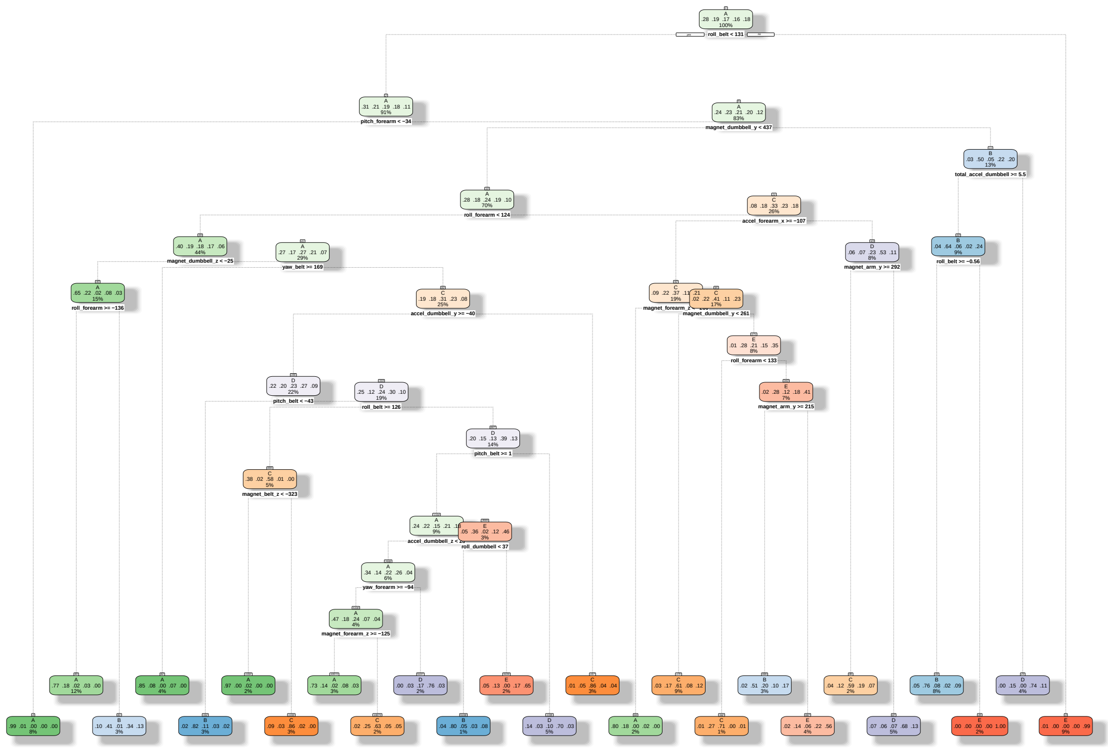
2. Decision Tree Visualization

```r
treeModel <- rpart(classe ~ ., data=trainData, method="class")
prp(treeModel) # fast plot
```




```r
set.seed(1813)
modFitDecTree <- rpart(classe ~ ., data=trainData, method="class")
fancyRpartPlot(modFitDecTree)
```

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



Rattle 2020−nov−01 22:35:08 sarah

```r
predict_decision_tree <- predict(modFitDecTree, newdata = testData, type="class")
conf_matrix_decision_tree <- confusionMatrix (table(predict_decision_tree, testData$classe))
conf_matrix_decision_tree
```

```
## Confusion Matrix and Statistics
##
##
## predict_decision_tree    A    B    C    D    E
##                     A 1502  162   24   60   18
##                     B   62  674   78   87   99
##                     C   44  159  825   74   92
##                     D   52   76   70  660   71
##                     E   14   68   29   83  802
##
## Overall Statistics
##
##                Accuracy : 0.7584
##                  95% CI : (0.7472, 0.7693)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6939
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
```

```
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8973   0.5917   0.8041   0.6846   0.7412
## Specificity            0.9373   0.9313   0.9241   0.9453   0.9596
## Pos Pred Value         0.8505   0.6740   0.6910   0.7104   0.8052
## Neg Pred Value         0.9582   0.9048   0.9572   0.9387   0.9427
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2552   0.1145   0.1402   0.1121   0.1363
## Detection Prevalence   0.3001   0.1699   0.2029   0.1579   0.1692
## Balanced Accuracy      0.9173   0.7615   0.8641   0.8150   0.8504
```

Accuracy : 0.7584

3. Generalized Boosted Model (GBM)

```
# One can also build a generalized boosted model and compare its accuracy
# to random forest model
set.seed(301)
modelBM <- train( classe ~.,
                  data = trainData,
                  method = "gbm",
                  trControl = trainControl(method="repeatedcv",number = 5,repeats = 1),
                  verbose = FALSE)
```

```
predictGBM <- predict(modelBM, newdata=trainData)
confMatGBM <- confusionMatrix (table(predictGBM, trainData$classe))
confMatGBM
```

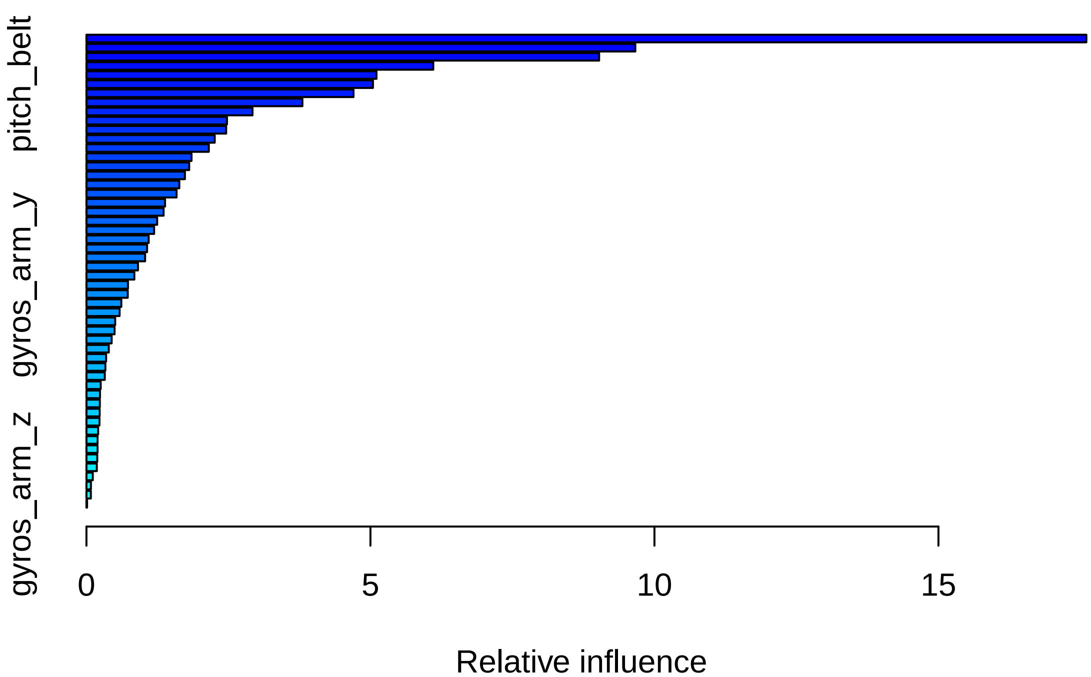
```
## Confusion Matrix and Statistics
##
##
## predictGBM    A    B    C    D    E
##          A 3872   70    0    1    1
##          B   21 2543   37    3   15
##          C    7   43 2333   71   18
##          D    4    2   22 2174   25
##          E    2    0    4    3 2466
##
## Overall Statistics
##
##                Accuracy : 0.9746
##                  95% CI : (0.9718, 0.9772)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9679
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
```

```
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9913   0.9567   0.9737   0.9654   0.9766
## Specificity            0.9927   0.9931   0.9877   0.9954   0.9992
## Pos Pred Value         0.9817   0.9710   0.9438   0.9762   0.9964
## Neg Pred Value         0.9965   0.9897   0.9944   0.9932   0.9948
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2819   0.1851   0.1698   0.1583   0.1795
## Detection Prevalence   0.2871   0.1907   0.1800   0.1621   0.1802
## Balanced Accuracy      0.9920   0.9749   0.9807   0.9804   0.9879
```

Accuracy : 0.9746

We can investigate our generalized boosted model a bit further to see which variables have the highest relative influence

```r
print(summary(modelBM))
```

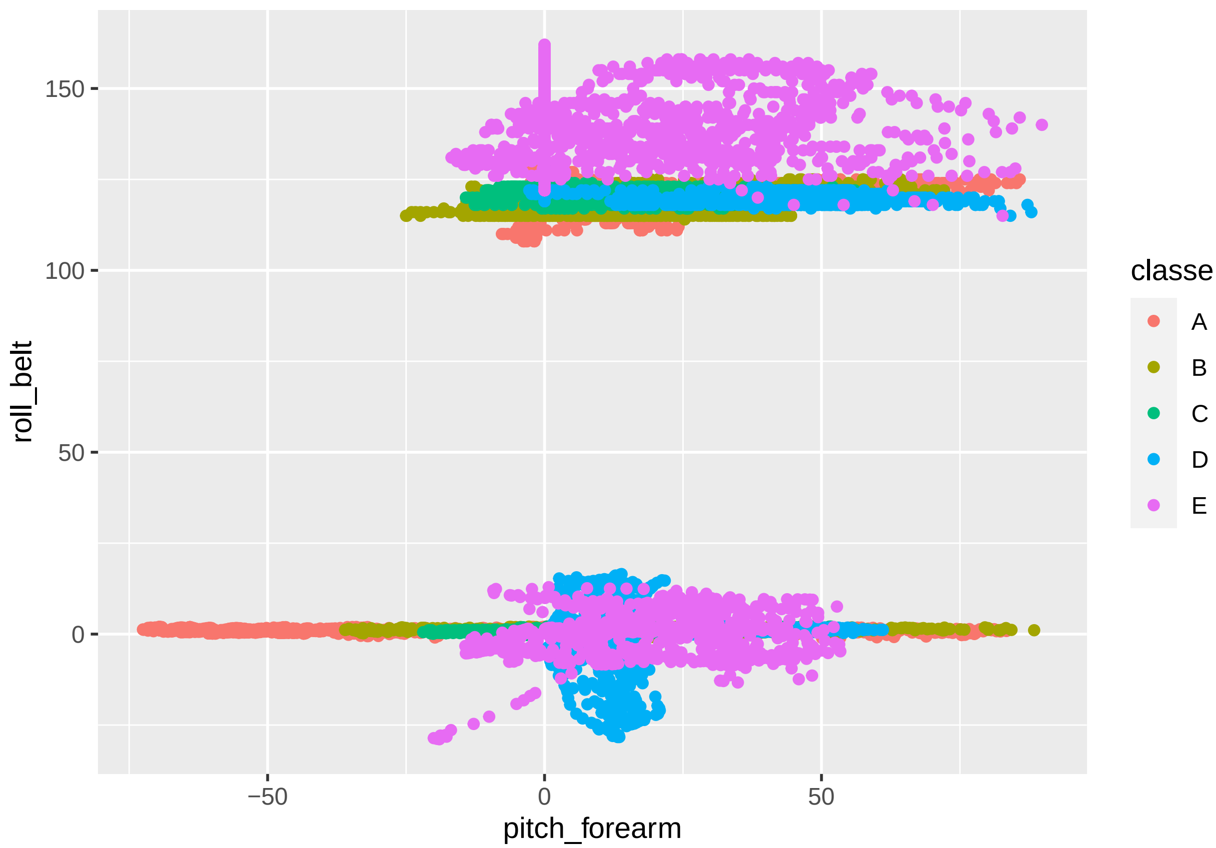



```
##                                     var    rel.inf
## roll_belt                     roll_belt 17.60410742
## pitch_forearm             pitch_forearm  9.66333716
## yaw_belt                       yaw_belt  9.02620885
## magnet_dumbbell_z     magnet_dumbbell_z  6.10599101
## roll_forearm               roll_forearm  5.10637742
## pitch_belt                   pitch_belt  5.04411485
## magnet_dumbbell_y     magnet_dumbbell_y  4.70176637
```

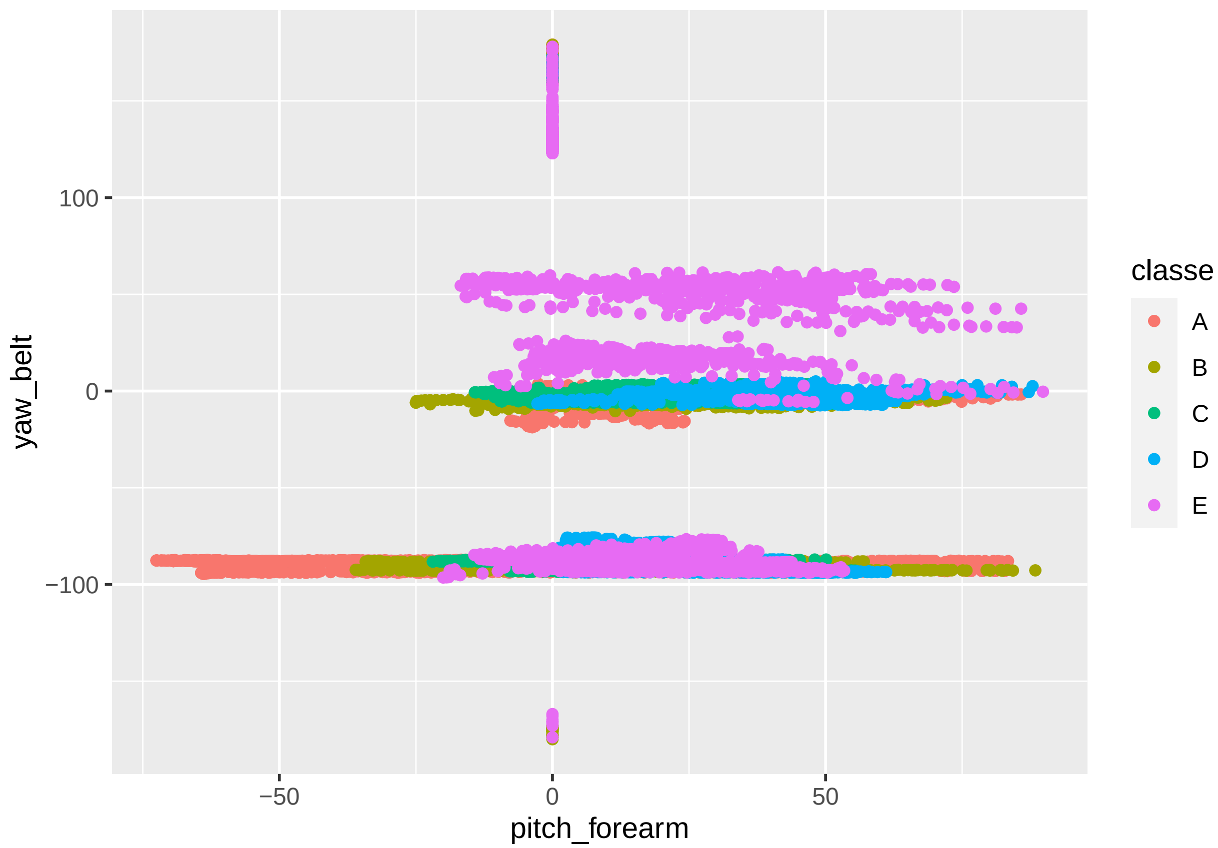
```
## magnet_belt_z               magnet_belt_z  3.80181259
## roll_dumbbell               roll_dumbbell  2.92327944
## magnet_forearm_z         magnet_forearm_z  2.47427553
## gyros_belt_z                 gyros_belt_z  2.45954574
## accel_forearm_x           accel_forearm_x  2.25868743
## accel_forearm_z           accel_forearm_z  2.15433289
## accel_dumbbell_y         accel_dumbbell_y  1.84997718
## yaw_arm                           yaw_arm  1.80891098
## accel_dumbbell_z         accel_dumbbell_z  1.73402258
## gyros_dumbbell_y         gyros_dumbbell_y  1.63461867
## magnet_dumbbell_x       magnet_dumbbell_x  1.58686100
## accel_dumbbell_x         accel_dumbbell_x  1.38410635
## magnet_arm_z                 magnet_arm_z  1.35836597
## magnet_belt_x               magnet_belt_x  1.24509828
## accel_belt_z                 accel_belt_z  1.19164286
## magnet_belt_y               magnet_belt_y  1.09599471
## magnet_forearm_x         magnet_forearm_x  1.06764650
## roll_arm                         roll_arm  1.03278331
## accel_arm_x                   accel_arm_x  0.90744255
## magnet_arm_y                 magnet_arm_y  0.84299149
## gyros_arm_y                   gyros_arm_y  0.73119562
## gyros_belt_y                 gyros_belt_y  0.72726209
## gyros_dumbbell_x         gyros_dumbbell_x  0.61417324
## magnet_arm_x                 magnet_arm_x  0.58352297
## total_accel_dumbbell total_accel_dumbbell  0.50624199
## magnet_forearm_y         magnet_forearm_y  0.49524048
## accel_forearm_y           accel_forearm_y  0.44351226
## total_accel_forearm   total_accel_forearm  0.39344828
## accel_arm_y                   accel_arm_y  0.34662679
## total_accel_arm           total_accel_arm  0.33341379
## gyros_arm_x                   gyros_arm_x  0.32288142
## accel_belt_y                 accel_belt_y  0.25132396
## gyros_forearm_y           gyros_forearm_y  0.23881235
## pitch_dumbbell             pitch_dumbbell  0.23566284
## gyros_forearm_x           gyros_forearm_x  0.23300556
## total_accel_belt         total_accel_belt  0.22919749
## accel_arm_z                   accel_arm_z  0.20631164
## yaw_dumbbell                 yaw_dumbbell  0.19592843
## gyros_dumbbell_z         gyros_dumbbell_z  0.19548057
## gyros_forearm_z           gyros_forearm_z  0.19027077
## pitch_arm                       pitch_arm  0.18247409
## yaw_forearm                   yaw_forearm  0.11184406
## accel_belt_x                 accel_belt_x  0.07866192
## gyros_belt_x                 gyros_belt_x  0.07784547
## gyros_arm_z                   gyros_arm_z  0.01136278
```

The above list shows the ranking of variables in our GBM. We see that roll_belt, pitch_forearm,and yaw_belt are the most performant ones. We can checkout a few plots demonstrating their power:

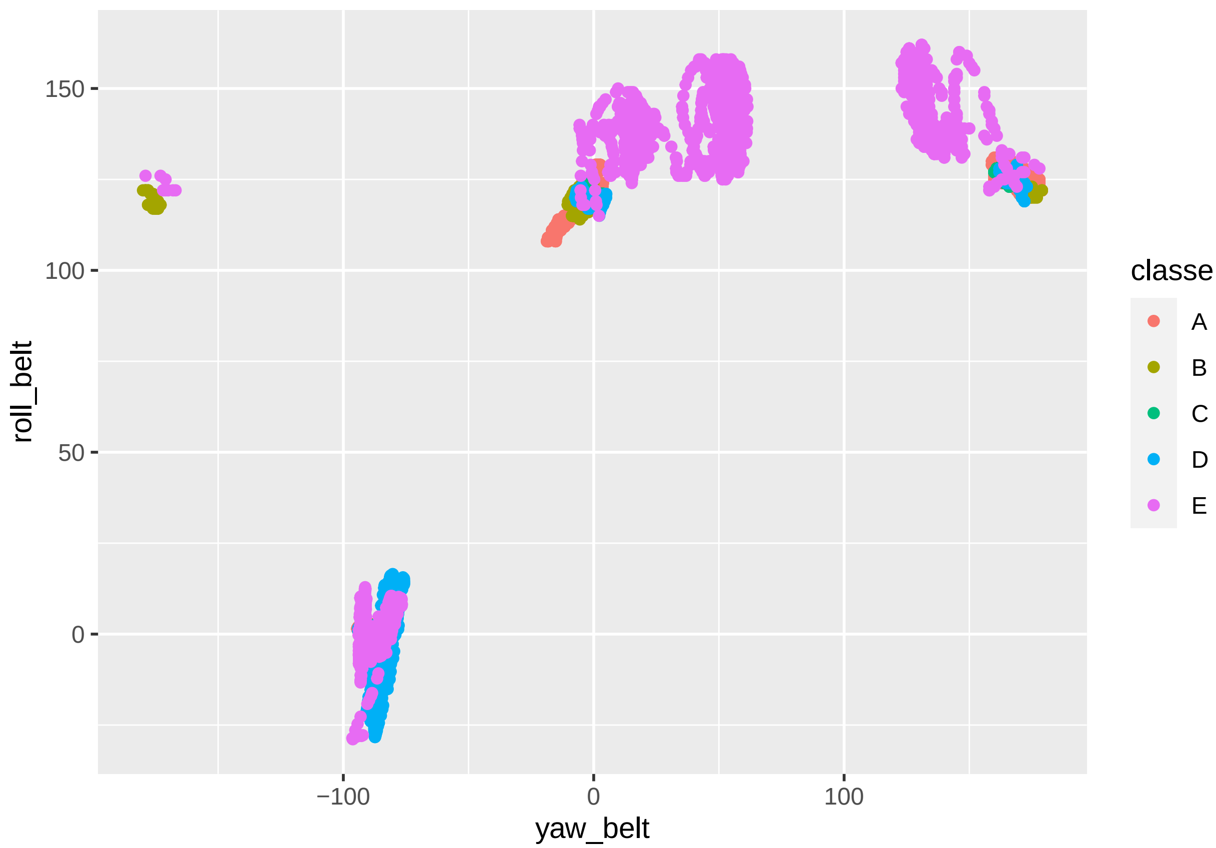
```r
qplot(pitch_forearm, roll_belt, data =trainData, col = classe)
```




```r
qplot(pitch_forearm, yaw_belt, data =trainData, col = classe)
```




```
qplot(yaw_belt, roll_belt, data =trainData, col = classe)
```



Applying the Best Predictive Model to the Test Data

Random Forest Model: 99,37% Generalized Boosted Model:97,46% Decision Tree Model: 75,84 %

The Random Forest model is selected and applied to make predictions on the 20 data points from the original testing dataset (data_quiz)

```
predictTEST <- predict(modelRf, newdata=testRaw)
predictTEST
```

```
##  [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
```