Practical Machine Learning Week 4 Project

Nuno R

March 16, 2020

Executive Summary

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://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har

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://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har

Load required packages for Initialization and Analysis

```
# this is only needed for tests in the local env
setwd("C:/Users/nroberto/practical_machine_learning_wk4_proj")

# not needed when running from R STudio
library(knitr)

# Clean up env, recommended by:
# https://community.rstudio.com/t/how-to-clear-the-r-environment/14303
rm(list=ls())

# More information on caret can be found here:
# http://topepo.github.io/caret/index.htmllibrary("caret")
library(caret)
# library(e1071) # Error: package e1071 is required

# More information on rpart can be found here:
# https://www.rdocumentation.org/packages/rpart/versions/4.1-15/topics/rpart
library(rpart)
library(rpart.plot)

# GUI for Data Science
# https://www.rdocumentation.org/packages/rattle/versions/5.3.0
library(rattle)
```

```
# Classification and Regression with Random Forest
# https://www.rdocumentation.org/packages/randomForest/versions/4.6-14/topics/randomForest
library(randomForest)

# Package for really cool correlation matrixes
# http://www.sthda.com/english/wiki/visualize-correlation-matrix-using-correlogram
library(corrplot)

# Generalized Boosted Regression Modeling (GBM)
# https://www.rdocumentation.org/packages/gbm/versions/2.1.5/topics/gbm
library(gbm)
```

Load Data for Analysis

```
seedValue = 202019
set.seed(seedValue)

precisionPoints = 5

# set the URL for the download
pmlTrainingData <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
pmlTestingData  <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

# download the datasets
trainingData <- read.csv(url(pmlTrainingData))
testingData  <- read.csv(url(pmlTestingData))
```

Prepare Data for Analysis

Create a partition with the training dataset

```
# More information can be found here:
# https://www.rdocumentation.org/packages/caret/versions/6.0-85/topics/createDataPartition
percentageForTraining = 0.60
trainingPartition  <- createDataPartition(trainingData$classe,
                                          p=percentageForTraining,
                                          list=FALSE)

trainingSet <- trainingData[trainingPartition, ] # eg, 60% for training
testingSet  <- trainingData[-trainingPartition, ] # eg, 40% for testing
```

Let's examine the Training Set

```
dim(trainingSet)
```

```
## [1] 11776   160
```

Let's examine the Testing Set

```
dim(testingSet)
```

```
## [1] 7846  160
```

NOTE: There are a lot of columns with no variance or close to 0 variance. They are not meaningful for the analysis and we need to remove them. This will further trim the Training and Testing sets for a more streamlined analysis

```
# Note: additional information can be found here borrowed from the caret package:
# https://www.rdocumentation.org/packages/mixOmics/versions/6.3.2/topics/nearZeroVar
columnsZeroVar <- nearZeroVar(trainingSet)
trainingSet <- trainingSet[, -columnsZeroVar]
testingSet  <- testingSet[, -columnsZeroVar]
```

Let's examine the Training set after trimming columns with zero variance

```
dim(trainingSet)
```

```
## [1] 11776   105
```

Let's examine the Testing set after trimming columns with zero variance

```
dim(testingSet)
```

```
## [1] 7846  105
```

NOTE: As can be observed there are a lot of missing data points, including NA, DIV/0 and empty cells. We need to clear this data before further analysis.

```
# find columns with a lot of NAs
percentageOfNAacceptable = 0.97
foundNAs <- sapply(trainingSet,
                   function(x)
                     mean(is.na(x))) >
                      percentageOfNAacceptable

#... and now remove them since they are not relevant for the analysis
trainingSet <- trainingSet[, foundNAs==FALSE]
testingSet  <- testingSet[, foundNAs==FALSE]
```

The Training and Testing sets are now cleaned up. Let's take a look at the Training set again

```
dim(trainingSet)
```

```
## [1] 11776    59
```

Let's take a look at the Training set again

```
dim(testingSet)
```

```
## [1] 7846   59
```

The data sets are are still showing columns that don't have to be used for analysis, so we can remove them before further analyzing the data

```
removeColumns = 5
trainingSet <- trainingSet[, -(1:removeColumns)]
testingSet  <- testingSet[, -(1:removeColumns)]
```

```
dim(trainingSet)
```

```
## [1] 11776    54
```

```
dim(testingSet)
```

```
## [1] 7846   54
```

Let's take a look at the correlation matrix for the relevant features

```r
pickColumnsToRemove = 54
correlationMatrix <- cor(trainingSet[, -pickColumnsToRemove])

# now plot it
# More info: https://www.rdocumentation.org/packages/corrplot/versions/0.2-0/topics/corrplot
corrplot(correlationMatrix,
         order  = "FPC",
         method = "shade",
         shade.method = "all",
         lwd.shade = 1,
         type   = "upper",
         tl.cex = 0.4,
         tl.col = rgb(0,0,1))
```

```
## Warning in text.default(pos.xlabel[, 1], pos.xlabel[, 2], newcolnames, srt =
## tl.srt, : "shade.method" is not a graphical parameter
```

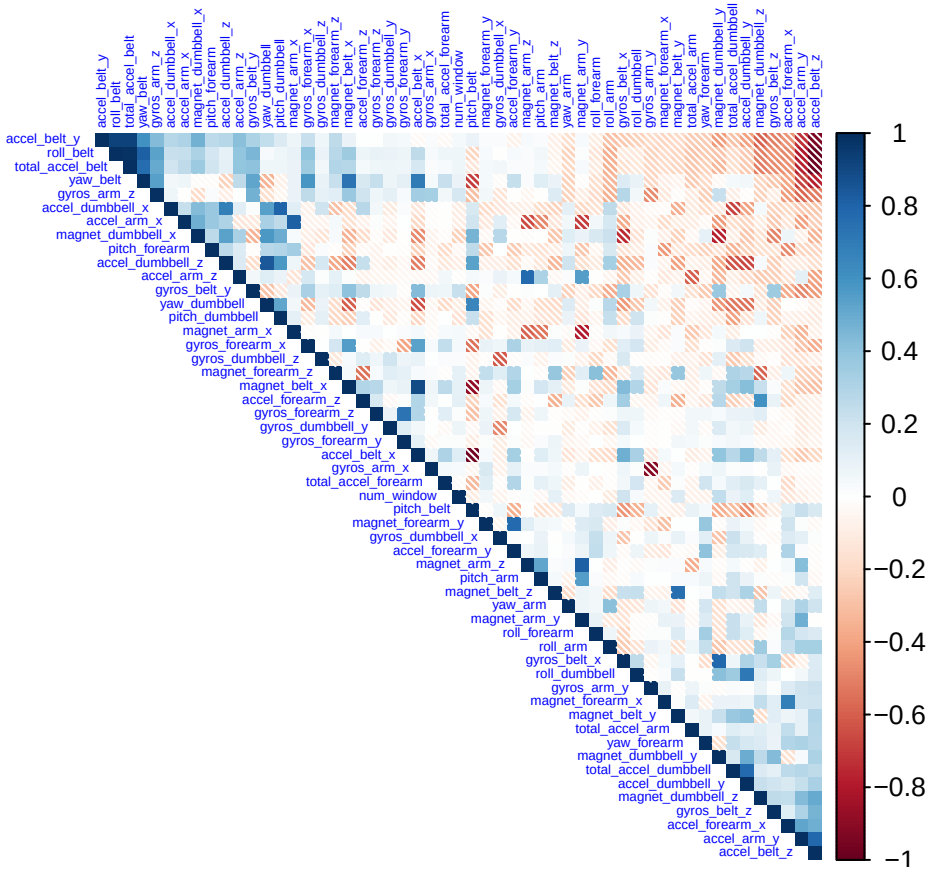
```
## Warning in text.default(pos.xlabel[, 1], pos.xlabel[, 2], newcolnames, srt =
## tl.srt, : "lwd.shade" is not a graphical parameter
```

```
## Warning in text.default(pos.ylabel[, 1], pos.ylabel[, 2], newrownames, col =
## tl.col, : "shade.method" is not a graphical parameter
```

```
## Warning in text.default(pos.ylabel[, 1], pos.ylabel[, 2], newrownames, col =
## tl.col, : "lwd.shade" is not a graphical parameter
```

```
## Warning in title(title, ...): "shade.method" is not a graphical parameter
```

```
## Warning in title(title, ...): "lwd.shade" is not a graphical parameter
```




```
# ##############################################################################
```

ML Algorithm 1: Random Forest

```
set.seed(seedValue)

# Really good information and exmaples for TrainControl
# This function prepares and controls the parameters for the function train
# https://topepo.github.io/caret/model-training-and-tuning.html
#
# official docs: https://www.rdocumentation.org/packages/caret/versions/4.47/topics/trainControl

randomForestTrainControl <- trainControl(method="cv",
                                         number=5,
                                         verboseIter=TRUE,
                                         returnData = TRUE
                                        )
```

ML Algorithm 1: Now let's train the model using the Training data set

```
# More info here: https://www.rdocumentation.org/packages/caret/versions/6.0-85/topics/train
trainedRandomForestModel <- train(classe ~ .,
                                  data=trainingSet,
                                  method="rf",
                                  metric = "Accuracy",
                                  trControl=randomForestTrainControl)
```

```
## + Fold1: mtry= 2
## - Fold1: mtry= 2
## + Fold1: mtry=27
## - Fold1: mtry=27
## + Fold1: mtry=53
## - Fold1: mtry=53
## + Fold2: mtry= 2
## - Fold2: mtry= 2
## + Fold2: mtry=27
## - Fold2: mtry=27
## + Fold2: mtry=53
## - Fold2: mtry=53
## + Fold3: mtry= 2
## - Fold3: mtry= 2
## + Fold3: mtry=27
## - Fold3: mtry=27
## + Fold3: mtry=53
## - Fold3: mtry=53
## + Fold4: mtry= 2
## - Fold4: mtry= 2
## + Fold4: mtry=27
## - Fold4: mtry=27
## + Fold4: mtry=53
## - Fold4: mtry=53
## + Fold5: mtry= 2
## - Fold5: mtry= 2
## + Fold5: mtry=27
## - Fold5: mtry=27
## + Fold5: mtry=53
## - Fold5: mtry=53
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 27 on full training set
```

ML Algorithm 1: The final model fits the data after training as below:

```
trainedRandomForestModel$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.24%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3347    1    0    0    0 0.0002986858
## B    3 2273    3    0    0 0.0026327337
## C    0    4 2050    0    0 0.0019474197
## D    0    0   10 1919    1 0.0056994819
## E    0    1    0    5 2159 0.0027713626
```

ML Algorithm 1: Let's predict using the data set we split earlier for Testing

```
# More info here:
# https://www.rdocumentation.org/packages/raster/versions/3.0-12/topics/predict
randomForestPrediction <- predict(trainedRandomForestModel,
                                  newdata=testingSet)
```

ML Algorithm 1: Let's create a confusion matrix for the Random Forest prediction done above and using the Testing data set

```
# More info here: https://www.rdocumentation.org/packages/caret/versions/3.45/topics/confusionMatrix
randomForestConfusionMatrix <- confusionMatrix(randomForestPrediction,
                                               testingSet$classe)
```

ML Algorithm 1: The confusion matrix based for the Random Forest algorithm using the Testing data set looks like:

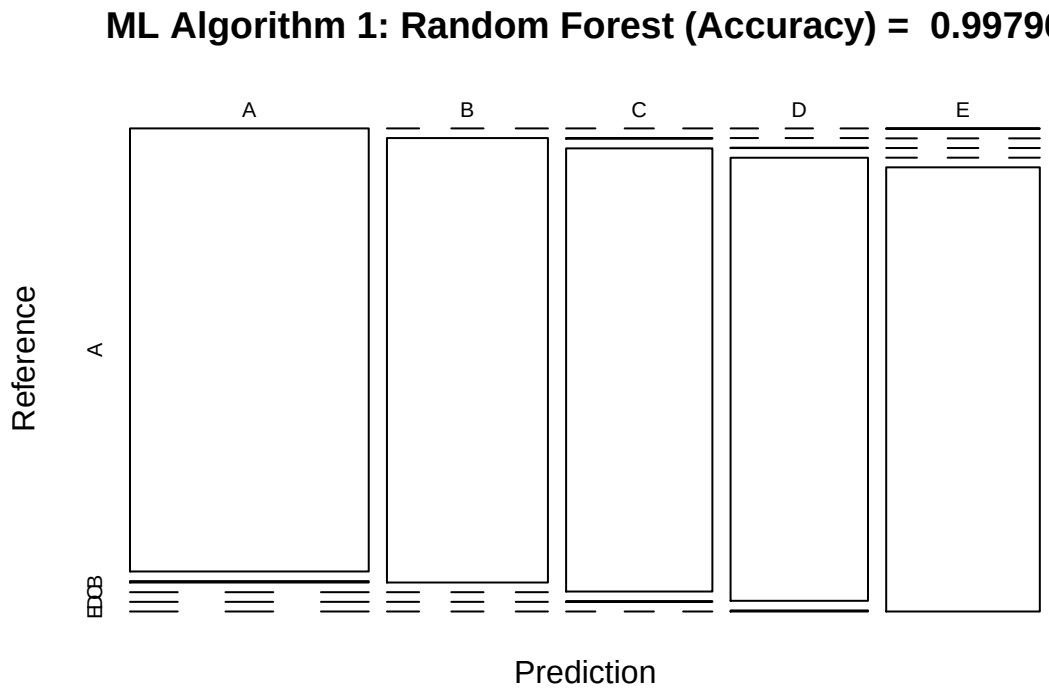
```
randomForestConfusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2231    7    0    0    0
##          B    0 1509    0    0    0
##          C    0    2 1367    2    0
##          D    0    0    1 1284    3
##          E    1    0    0    0 1439
##
## Overall Statistics
##
##                Accuracy : 0.998
##                  95% CI : (0.9967, 0.9988)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9974
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9996   0.9941   0.9993   0.9984   0.9979
## Specificity            0.9988   1.0000   0.9994   0.9994   0.9998
## Pos Pred Value         0.9969   1.0000   0.9971   0.9969   0.9993
## Neg Pred Value         0.9998   0.9986   0.9998   0.9997   0.9995
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1923   0.1742   0.1637   0.1834
## Detection Prevalence   0.2852   0.1923   0.1747   0.1642   0.1835
## Balanced Accuracy      0.9992   0.9970   0.9993   0.9989   0.9989
```

ML Algorithm 1: The Confusion Matrix accuracy plot

```
plot(randomForestConfusionMatrix$table,
     col = randomForestConfusionMatrix$byClass,
     main = paste("ML Algorithm 1: Random Forest (Accuracy) = ",
```

```
                round(randomForestConfusionMatrix$overall['Accuracy'],
                      precisionPoints)))
```




```
# ##############################################################################
```

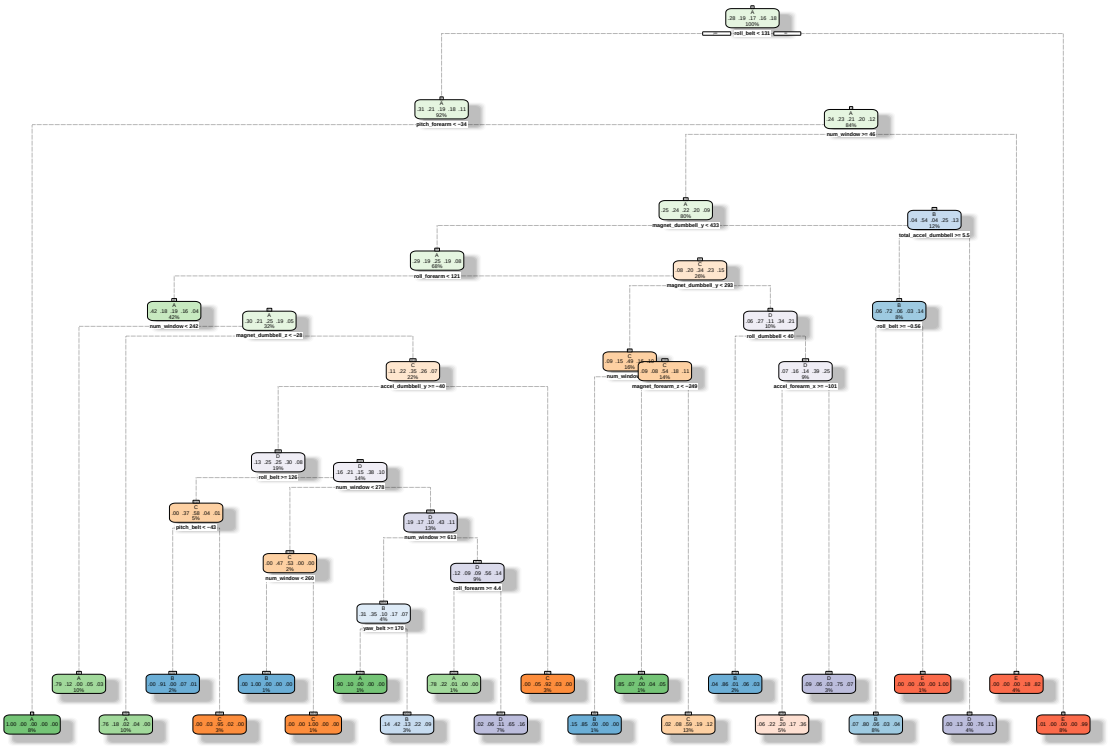
ML Algorithm 2: Decision Trees

```
set.seed(seedValue)

# Recursive partitioning for decision tress
# More info here: https://www.rdocumentation.org/packages/rpart/versions/4.1-15/topics/rpart
decisionTreeTrained <- rpart(classe ~ .,
                             data=trainingSet,
                             method="class")
```

ML Algorithm 2: Let's plot the decision tree as partitioned above

```
# More info here: https://www.rdocumentation.org/packages/rattle/versions/5.3.0/topics/fancyRpartPlot
# site: https://rattle.togaware.com/
fancyRpartPlot(decisionTreeTrained)
```



ML Algorithm 2: Let's predict using the data set we split earlier for Testing

```
# More info here:
# https://www.rdocumentation.org/packages/raster/versions/3.0-12/topics/predict
decisionTreePrediction <- predict(decisionTreeTrained,
                                  newdata=testingSet,
                                  type="class")
```

ML Algorithm 2: Let's create a confusion matrix for the Decision Tree prediction done above and using the Testing data set

```
decisionTreeConfusionMatrix <- confusionMatrix(decisionTreePrediction,
                                               testingSet$classe)
```

ML Algorithm 2: The confusion matrix for the Decision Tree algorithm using the Testing data set looks like:

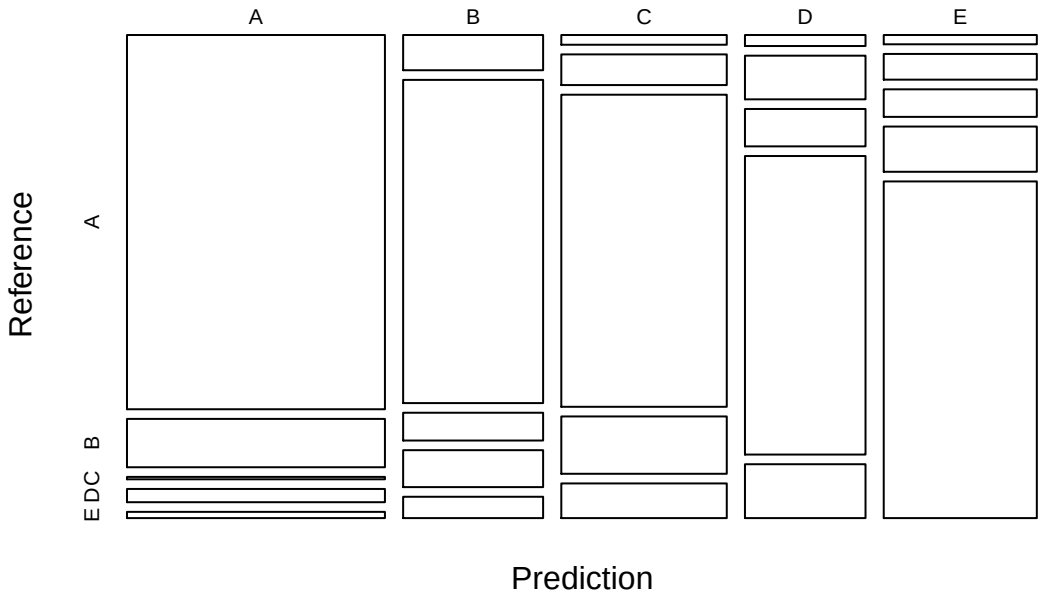
```
decisionTreeConfusionMatrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2036  263   13   72   34
##          B  104  954   82  109   63
##          C   34  107 1089  200  121
##          D   28  111   95  759  137
##          E   30   83   89  146 1087
##
## Overall Statistics
```

```
##
##                Accuracy : 0.7552
##                  95% CI : (0.7455, 0.7646)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6894
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9122   0.6285   0.7961  0.59020   0.7538
## Specificity            0.9320   0.9434   0.9287  0.94345   0.9457
## Pos Pred Value         0.8420   0.7271   0.7021  0.67168   0.7575
## Neg Pred Value         0.9639   0.9137   0.9557  0.92153   0.9446
## Prevalence             0.2845   0.1935   0.1744  0.16391   0.1838
## Detection Rate         0.2595   0.1216   0.1388  0.09674   0.1385
## Detection Prevalence   0.3082   0.1672   0.1977  0.14402   0.1829
## Balanced Accuracy      0.9221   0.7859   0.8624  0.76682   0.8497
```

ML Algorithm 2: The Confusion Matrix accuracy plot

```
plot(decisionTreeConfusionMatrix$table,
     col = decisionTreeConfusionMatrix$byClass,
     main = paste("ML Algorithm 2: Decision Trees (Accuracy) = ",
                  round(decisionTreeConfusionMatrix$overall['Accuracy'],
                        precisionPoints)))
```




```
# ##############################################################################
```

ML Algorithm 3: Generalized Boosted Model

```
set.seed(seedValue)
# Now let's train the model using the Training data set
# More info here: https://www.rdocumentation.org/packages/caret/versions/6.0-85/topics/train
gbmControl <- trainControl(method = "repeatedcv",
                           number = 5,
                           repeats = 1,
                           verboseIter = TRUE,
                           returnData = TRUE)
```

ML Algorithm 3: The final model fits the data after training as below:

```
gbmTrained  <- train(classe ~ .,
                     data=trainingSet,
                     method = "gbm",
                     trControl = gbmControl,
                     verbose = TRUE)
```

```
## + Fold1.Rep1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1306
##      2        1.5230             nan     0.1000    0.0844
##      3        1.4654             nan     0.1000    0.0692
##      4        1.4197             nan     0.1000    0.0525
```

```
##      5        1.3851             nan     0.1000    0.0462
##      6        1.3547             nan     0.1000    0.0456
##      7        1.3243             nan     0.1000    0.0340
##      8        1.3008             nan     0.1000    0.0413
##      9        1.2744             nan     0.1000    0.0351
##     10        1.2522             nan     0.1000    0.0325
##     20        1.0867             nan     0.1000    0.0219
##     40        0.9022             nan     0.1000    0.0094
##     60        0.7908             nan     0.1000    0.0054
##     80        0.7055             nan     0.1000    0.0046
##    100        0.6429             nan     0.1000    0.0026
##    120        0.5869             nan     0.1000    0.0023
##    140        0.5406             nan     0.1000    0.0029
##    150        0.5206             nan     0.1000    0.0031
##
## - Fold1.Rep1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10, n.trees=150
## + Fold1.Rep1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1960
##      2        1.4836             nan     0.1000    0.1334
##      3        1.3983             nan     0.1000    0.1063
##      4        1.3295             nan     0.1000    0.0872
##      5        1.2717             nan     0.1000    0.0799
##      6        1.2203             nan     0.1000    0.0634
##      7        1.1789             nan     0.1000    0.0580
##      8        1.1413             nan     0.1000    0.0503
##      9        1.1091             nan     0.1000    0.0539
##     10        1.0756             nan     0.1000    0.0378
##     20        0.8418             nan     0.1000    0.0268
##     40        0.6037             nan     0.1000    0.0121
##     60        0.4684             nan     0.1000    0.0072
##     80        0.3797             nan     0.1000    0.0061
##    100        0.3124             nan     0.1000    0.0041
##    120        0.2641             nan     0.1000    0.0020
##    140        0.2238             nan     0.1000    0.0028
##    150        0.2049             nan     0.1000    0.0022
##
## - Fold1.Rep1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10, n.trees=150
## + Fold1.Rep1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2400
##      2        1.4564             nan     0.1000    0.1607
##      3        1.3519             nan     0.1000    0.1235
##      4        1.2710             nan     0.1000    0.0945
##      5        1.2084             nan     0.1000    0.0941
##      6        1.1495             nan     0.1000    0.0879
##      7        1.0940             nan     0.1000    0.0666
##      8        1.0515             nan     0.1000    0.0743
##      9        1.0058             nan     0.1000    0.0666
##     10        0.9643             nan     0.1000    0.0515
##     20        0.6845             nan     0.1000    0.0326
##     40        0.4413             nan     0.1000    0.0126
##     60        0.3202             nan     0.1000    0.0066
##     80        0.2468             nan     0.1000    0.0038
```

```
##    100        0.1894             nan     0.1000    0.0030
##    120        0.1496             nan     0.1000    0.0034
##    140        0.1202             nan     0.1000    0.0011
##    150        0.1093             nan     0.1000    0.0020
##
## - Fold1.Rep1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10, n.trees=150
## + Fold2.Rep1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1275
##      2        1.5241             nan     0.1000    0.0906
##      3        1.4645             nan     0.1000    0.0667
##      4        1.4209             nan     0.1000    0.0559
##      5        1.3845             nan     0.1000    0.0514
##      6        1.3518             nan     0.1000    0.0402
##      7        1.3250             nan     0.1000    0.0412
##      8        1.2974             nan     0.1000    0.0348
##      9        1.2743             nan     0.1000    0.0334
##     10        1.2511             nan     0.1000    0.0258
##     20        1.0875             nan     0.1000    0.0197
##     40        0.9034             nan     0.1000    0.0089
##     60        0.7897             nan     0.1000    0.0080
##     80        0.7082             nan     0.1000    0.0054
##    100        0.6457             nan     0.1000    0.0033
##    120        0.5933             nan     0.1000    0.0028
##    140        0.5480             nan     0.1000    0.0027
##    150        0.5281             nan     0.1000    0.0030
##
## - Fold2.Rep1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10, n.trees=150
## + Fold2.Rep1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1855
##      2        1.4894             nan     0.1000    0.1313
##      3        1.4031             nan     0.1000    0.1069
##      4        1.3345             nan     0.1000    0.0875
##      5        1.2785             nan     0.1000    0.0848
##      6        1.2236             nan     0.1000    0.0685
##      7        1.1795             nan     0.1000    0.0687
##      8        1.1368             nan     0.1000    0.0494
##      9        1.1051             nan     0.1000    0.0530
##     10        1.0719             nan     0.1000    0.0539
##     20        0.8474             nan     0.1000    0.0285
##     40        0.6043             nan     0.1000    0.0111
##     60        0.4769             nan     0.1000    0.0063
##     80        0.3870             nan     0.1000    0.0059
##    100        0.3271             nan     0.1000    0.0031
##    120        0.2756             nan     0.1000    0.0027
##    140        0.2323             nan     0.1000    0.0016
##    150        0.2155             nan     0.1000    0.0016
##
## - Fold2.Rep1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10, n.trees=150
## + Fold2.Rep1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2331
##      2        1.4611             nan     0.1000    0.1637
```

```
##      3        1.3571             nan     0.1000    0.1225
##      4        1.2764             nan     0.1000    0.1095
##      5        1.2070             nan     0.1000    0.0942
##      6        1.1472             nan     0.1000    0.0823
##      7        1.0948             nan     0.1000    0.0752
##      8        1.0479             nan     0.1000    0.0577
##      9        1.0090             nan     0.1000    0.0556
##     10        0.9742             nan     0.1000    0.0526
##     20        0.7101             nan     0.1000    0.0288
##     40        0.4552             nan     0.1000    0.0122
##     60        0.3376             nan     0.1000    0.0060
##     80        0.2543             nan     0.1000    0.0069
##    100        0.1951             nan     0.1000    0.0020
##    120        0.1620             nan     0.1000    0.0014
##    140        0.1319             nan     0.1000    0.0019
##    150        0.1179             nan     0.1000    0.0016
##
## - Fold2.Rep1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10, n.trees=150
## + Fold3.Rep1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1269
##      2        1.5234             nan     0.1000    0.0898
##      3        1.4650             nan     0.1000    0.0680
##      4        1.4202             nan     0.1000    0.0527
##      5        1.3847             nan     0.1000    0.0514
##      6        1.3519             nan     0.1000    0.0388
##      7        1.3259             nan     0.1000    0.0419
##      8        1.2993             nan     0.1000    0.0318
##      9        1.2786             nan     0.1000    0.0350
##     10        1.2541             nan     0.1000    0.0317
##     20        1.0877             nan     0.1000    0.0192
##     40        0.9051             nan     0.1000    0.0114
##     60        0.7918             nan     0.1000    0.0069
##     80        0.7056             nan     0.1000    0.0047
##    100        0.6408             nan     0.1000    0.0037
##    120        0.5847             nan     0.1000    0.0025
##    140        0.5397             nan     0.1000    0.0019
##    150        0.5196             nan     0.1000    0.0018
##
## - Fold3.Rep1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10, n.trees=150
## + Fold3.Rep1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1888
##      2        1.4874             nan     0.1000    0.1305
##      3        1.4010             nan     0.1000    0.1077
##      4        1.3313             nan     0.1000    0.0833
##      5        1.2775             nan     0.1000    0.0666
##      6        1.2327             nan     0.1000    0.0708
##      7        1.1871             nan     0.1000    0.0589
##      8        1.1490             nan     0.1000    0.0621
##      9        1.1118             nan     0.1000    0.0505
##     10        1.0802             nan     0.1000    0.0586
##     20        0.8495             nan     0.1000    0.0319
##     40        0.6088             nan     0.1000    0.0110
```

```
##     60        0.4761             nan     0.1000    0.0056
##     80        0.3844             nan     0.1000    0.0057
##    100        0.3178             nan     0.1000    0.0027
##    120        0.2699             nan     0.1000    0.0017
##    140        0.2273             nan     0.1000    0.0023
##    150        0.2101             nan     0.1000    0.0011
##
## - Fold3.Rep1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10, n.trees=150
## + Fold3.Rep1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2434
##      2        1.4565             nan     0.1000    0.1609
##      3        1.3537             nan     0.1000    0.1327
##      4        1.2705             nan     0.1000    0.1131
##      5        1.2003             nan     0.1000    0.0840
##      6        1.1457             nan     0.1000    0.0805
##      7        1.0953             nan     0.1000    0.0745
##      8        1.0478             nan     0.1000    0.0597
##      9        1.0096             nan     0.1000    0.0628
##     10        0.9696             nan     0.1000    0.0577
##     20        0.6886             nan     0.1000    0.0335
##     40        0.4433             nan     0.1000    0.0123
##     60        0.3212             nan     0.1000    0.0066
##     80        0.2442             nan     0.1000    0.0034
##    100        0.1940             nan     0.1000    0.0044
##    120        0.1504             nan     0.1000    0.0027
##    140        0.1218             nan     0.1000    0.0017
##    150        0.1097             nan     0.1000    0.0019
##
## - Fold3.Rep1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10, n.trees=150
## + Fold4.Rep1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1237
##      2        1.5253             nan     0.1000    0.0868
##      3        1.4681             nan     0.1000    0.0654
##      4        1.4240             nan     0.1000    0.0553
##      5        1.3884             nan     0.1000    0.0428
##      6        1.3596             nan     0.1000    0.0443
##      7        1.3312             nan     0.1000    0.0391
##      8        1.3065             nan     0.1000    0.0412
##      9        1.2785             nan     0.1000    0.0335
##     10        1.2562             nan     0.1000    0.0276
##     20        1.0964             nan     0.1000    0.0188
##     40        0.9119             nan     0.1000    0.0103
##     60        0.7976             nan     0.1000    0.0063
##     80        0.7138             nan     0.1000    0.0050
##    100        0.6468             nan     0.1000    0.0040
##    120        0.5916             nan     0.1000    0.0026
##    140        0.5470             nan     0.1000    0.0025
##    150        0.5265             nan     0.1000    0.0020
##
## - Fold4.Rep1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10, n.trees=150
## + Fold4.Rep1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
```

```
##      1        1.6094             nan     0.1000    0.1848
##      2        1.4879             nan     0.1000    0.1272
##      3        1.4046             nan     0.1000    0.1115
##      4        1.3342             nan     0.1000    0.0844
##      5        1.2784             nan     0.1000    0.0777
##      6        1.2298             nan     0.1000    0.0622
##      7        1.1886             nan     0.1000    0.0602
##      8        1.1503             nan     0.1000    0.0541
##      9        1.1142             nan     0.1000    0.0481
##     10        1.0835             nan     0.1000    0.0393
##     20        0.8531             nan     0.1000    0.0251
##     40        0.6194             nan     0.1000    0.0111
##     60        0.4842             nan     0.1000    0.0099
##     80        0.3867             nan     0.1000    0.0048
##    100        0.3233             nan     0.1000    0.0056
##    120        0.2730             nan     0.1000    0.0025
##    140        0.2355             nan     0.1000    0.0018
##    150        0.2183             nan     0.1000    0.0019
##
## - Fold4.Rep1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10, n.trees=150
## + Fold4.Rep1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2306
##      2        1.4590             nan     0.1000    0.1683
##      3        1.3536             nan     0.1000    0.1219
##      4        1.2758             nan     0.1000    0.0984
##      5        1.2112             nan     0.1000    0.0921
##      6        1.1526             nan     0.1000    0.0819
##      7        1.1009             nan     0.1000    0.0831
##      8        1.0497             nan     0.1000    0.0609
##      9        1.0100             nan     0.1000    0.0651
##     10        0.9707             nan     0.1000    0.0630
##     20        0.6981             nan     0.1000    0.0340
##     40        0.4538             nan     0.1000    0.0146
##     60        0.3246             nan     0.1000    0.0061
##     80        0.2478             nan     0.1000    0.0041
##    100        0.1957             nan     0.1000    0.0026
##    120        0.1518             nan     0.1000    0.0023
##    140        0.1231             nan     0.1000    0.0017
##    150        0.1111             nan     0.1000    0.0016
##
## - Fold4.Rep1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10, n.trees=150
## + Fold5.Rep1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1275
##      2        1.5241             nan     0.1000    0.0861
##      3        1.4666             nan     0.1000    0.0682
##      4        1.4222             nan     0.1000    0.0521
##      5        1.3872             nan     0.1000    0.0499
##      6        1.3545             nan     0.1000    0.0427
##      7        1.3269             nan     0.1000    0.0359
##      8        1.3034             nan     0.1000    0.0394
##      9        1.2767             nan     0.1000    0.0306
##     10        1.2563             nan     0.1000    0.0351
```

```
##     20        1.0921             nan     0.1000    0.0209
##     40        0.9094             nan     0.1000    0.0076
##     60        0.7983             nan     0.1000    0.0066
##     80        0.7137             nan     0.1000    0.0036
##    100        0.6481             nan     0.1000    0.0051
##    120        0.5936             nan     0.1000    0.0022
##    140        0.5487             nan     0.1000    0.0027
##    150        0.5274             nan     0.1000    0.0025
##
## - Fold5.Rep1: shrinkage=0.1, interaction.depth=1, n.minobsinnode=10, n.trees=150
## + Fold5.Rep1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1795
##      2        1.4880             nan     0.1000    0.1305
##      3        1.4029             nan     0.1000    0.1109
##      4        1.3334             nan     0.1000    0.0946
##      5        1.2737             nan     0.1000    0.0671
##      6        1.2288             nan     0.1000    0.0653
##      7        1.1854             nan     0.1000    0.0653
##      8        1.1450             nan     0.1000    0.0617
##      9        1.1054             nan     0.1000    0.0499
##     10        1.0728             nan     0.1000    0.0377
##     20        0.8645             nan     0.1000    0.0190
##     40        0.6349             nan     0.1000    0.0129
##     60        0.4861             nan     0.1000    0.0161
##     80        0.3903             nan     0.1000    0.0054
##    100        0.3254             nan     0.1000    0.0039
##    120        0.2688             nan     0.1000    0.0037
##    140        0.2285             nan     0.1000    0.0023
##    150        0.2111             nan     0.1000    0.0017
##
## - Fold5.Rep1: shrinkage=0.1, interaction.depth=2, n.minobsinnode=10, n.trees=150
## + Fold5.Rep1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10, n.trees=150
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2394
##      2        1.4555             nan     0.1000    0.1612
##      3        1.3524             nan     0.1000    0.1303
##      4        1.2716             nan     0.1000    0.1082
##      5        1.2026             nan     0.1000    0.0949
##      6        1.1423             nan     0.1000    0.0751
##      7        1.0950             nan     0.1000    0.0695
##      8        1.0509             nan     0.1000    0.0649
##      9        1.0089             nan     0.1000    0.0639
##     10        0.9692             nan     0.1000    0.0646
##     20        0.7009             nan     0.1000    0.0254
##     40        0.4525             nan     0.1000    0.0158
##     60        0.3264             nan     0.1000    0.0047
##     80        0.2489             nan     0.1000    0.0060
##    100        0.1922             nan     0.1000    0.0024
##    120        0.1535             nan     0.1000    0.0013
##    140        0.1227             nan     0.1000    0.0011
##    150        0.1118             nan     0.1000    0.0007
##
## - Fold5.Rep1: shrinkage=0.1, interaction.depth=3, n.minobsinnode=10, n.trees=150
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting n.trees = 150, interaction.depth = 3, shrinkage = 0.1, n.minobsinnode = 10 on full training s
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2379
##      2        1.4596             nan     0.1000    0.1689
##      3        1.3508             nan     0.1000    0.1287
##      4        1.2708             nan     0.1000    0.1063
##      5        1.2035             nan     0.1000    0.0855
##      6        1.1482             nan     0.1000    0.0858
##      7        1.0948             nan     0.1000    0.0739
##      8        1.0471             nan     0.1000    0.0612
##      9        1.0083             nan     0.1000    0.0623
##     10        0.9685             nan     0.1000    0.0630
##     20        0.6987             nan     0.1000    0.0354
##     40        0.4567             nan     0.1000    0.0143
##     60        0.3274             nan     0.1000    0.0069
##     80        0.2521             nan     0.1000    0.0041
##    100        0.1937             nan     0.1000    0.0039
##    120        0.1524             nan     0.1000    0.0020
##    140        0.1258             nan     0.1000    0.0026
##    150        0.1140             nan     0.1000    0.0017
```

ML Algorithm 3: The final model fits the data after training as below:

```
gbmTrained$finalModel
```

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

ML Algorithm 3: Let's predict using the data set we split earlier for Testing

```
# More info here:
# https://www.rdocumentation.org/packages/raster/versions/3.0-12/topics/predict
gbmPrediction <- predict(gbmTrained,
                         newdata=testingSet)
```

ML Algorithm 3: Let's create a confusion matrix for the Generalized Boosted Model prediction done above and using the Testing data set

```
# More info here: https://www.rdocumentation.org/packages/caret/versions/3.45/topics/confusionMatrix
gbmConfusionMatrix <- confusionMatrix(gbmPrediction,
                                      testingSet$classe)
```

ML Algorithm 3: The confusion matrix based for the Generalized Boosted Model algorithm using the Testing data set looks like:

```
gbmConfusionMatrix
```

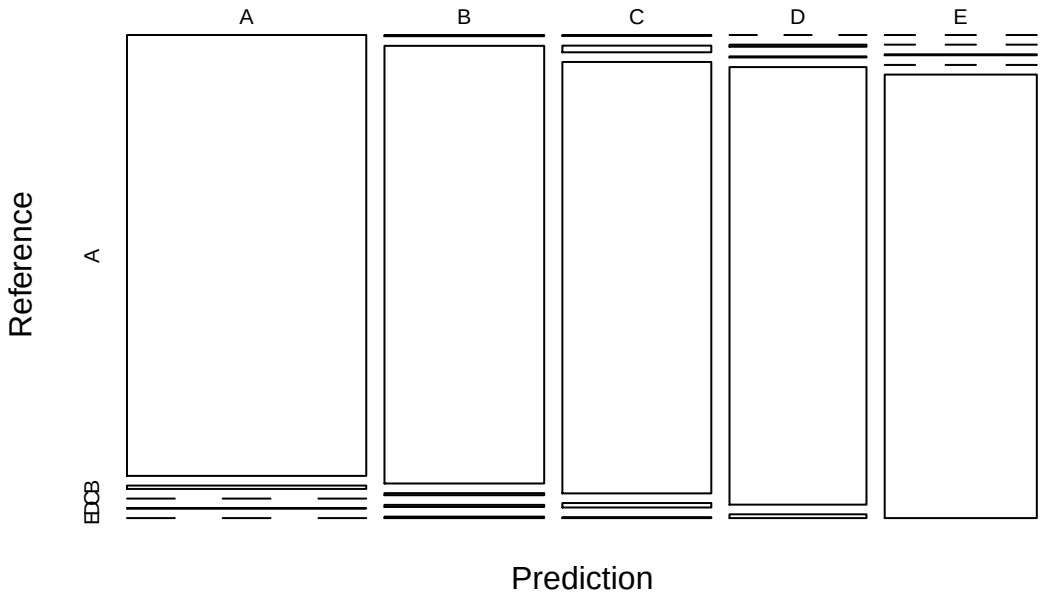
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2225   17    0    1    0
##          B    4 1474    7    7    5
##          C    3   21 1355   14    3
##          D    0    6    3 1264   11
```

```
##          E    0    0    3    0 1423
##
## Overall Statistics
##
##                Accuracy : 0.9866
##                  95% CI : (0.9838, 0.989)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9831
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9969   0.9710   0.9905   0.9829   0.9868
## Specificity            0.9968   0.9964   0.9937   0.9970   0.9995
## Pos Pred Value         0.9920   0.9846   0.9706   0.9844   0.9979
## Neg Pred Value         0.9988   0.9931   0.9980   0.9966   0.9970
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2836   0.1879   0.1727   0.1611   0.1814
## Detection Prevalence   0.2859   0.1908   0.1779   0.1637   0.1817
## Balanced Accuracy      0.9968   0.9837   0.9921   0.9899   0.9932
```

ML Algorithm 3: The Confusion Matrix accuracy plot

```
plot(gbmConfusionMatrix$table,
     col = gbmConfusionMatrix$byClass,
     main = paste("ML Algorithm 3: Generalized Boosted Model (Accuracy) = ",
                  round(gbmConfusionMatrix$overall['Accuracy'],
                        precisionPoints)))
```

ML Algorithm 3: Generalized Boosted Model (Accuracy) = 0.98662




```
# ##############################################################################
```

Final Prediction using the chosen algorithm

```
qzPrediction <- predict(trainedRandomForestModel,
                        newdata=testingData)

qzPrediction
```

```
##  [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
```

Final Thoughts and Conclusion

The results show that the Random Forest algorithm outperforms the Decision Tree in terms of accuracy. We are getting 99% in sample accuracy, followed by the GBM at 98%, whereas the Decision Tree algorithm is only 75% accurate.