Practical Machine Learning

Ruben_Mkrtchyan

18/05/2021

Practical Machine Learning

Peer-Graded Assignment

Ruben Mkrtchyan

I.Overview

This report is the final assignment for Coursera's Practical Machine Learning course. This course is a part of the Data Science Specialization track which is organized by John Hopkins University. This report will be graded by peers of the same course.

The report will use data from accelerometers on forearm, arm, belt and dumbel of six participants. The data consists of a training data and a test data, which is for validating the selected model.The goal of the report is to predict the way that the participants performed the excercise. That is the "classe" variable in the training set. There will be three models trained in this report, those are:Decision Tree, Random Forest and Gradient Boosted Trees. The best model will be sellected based on the accuracy rate and out of sample error rate. By the use of the selected model, the twenty test cases available in the test data will be predicted.

II.Backgroung

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).

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

III.Data Loading, preprocessing and cleaning

a)Data Preprocessing and package instalation:

At first, we need to upload the necessary libraries.

```
#install.packages('caret', dependencies = TRUE)
#install.packages('gtable', dependencies = TRUE)
#install.packages('ggplot2', dependencies = TRUE)
#install.packages('gower', dependencies = TRUE)

#install.packages('jquerylib', dependencies = TRUE)

library(caret)

## Warning: package 'caret' was built under R version 4.0.5

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.0.5

library(lattice)
library(ggplot2)
library(corrplot)

## Warning: package 'corrplot' was built under R version 4.0.5

## corrplot 0.88 loaded

library(knitr)
library(rpart)
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 4.0.5

library(rattle)

## Warning: package 'rattle' was built under R version 4.0.5

## Loading required package: tibble

## Loading required package: bitops

## Warning: package 'bitops' was built under R version 4.0.5
```

```
## 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.

library(randomForest)

## Warning: package 'randomForest' was built under R version 4.0.5

## 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

set.seed(404) #Setting the seed
```

b)Data Reading:

Now we read the data.

```
Url_train <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
Url_test  <- "http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

train_data <- read.csv(url(Url_train))
test_data  <- read.csv(url(Url_test))

inTrain  <- createDataPartition(train_data$classe, p=0.7, list=FALSE)
Train_Set <- train_data[inTrain, ]
Test_Set  <- train_data[-inTrain, ]
```

Let's check the number of variables of Train_Set and Test_Set.

```
dim(Train_Set)

## [1] 13737   160

dim(Test_Set)

## [1] 5885  160
```

As we see, we have got 160 initial variables, however after data cleaning the number will be reduced.

c)Data Cleaning:

We need to remove the near zero variance (NZV) variables.

```
NZV <- nearZeroVar(Train_Set)
Train_Set <- Train_Set[, -NZV]
Test_Set  <- Test_Set[, -NZV]

dim(Train_Set)

## [1] 13737   105

dim(Test_Set)

## [1] 5885  105
```

As we see, the number of variables are reduced to 105 for both Test_Set and Train_Set.

Now, we need to clean the variables with mostly NA values.

```
NA_val    <- sapply(Train_Set, function(x) mean(is.na(x))) > 0.8
Train_Set <- Train_Set[, NA_val==FALSE]
Test_Set  <- Test_Set[, NA_val==FALSE]

dim(Train_Set)

## [1] 13737    59

dim(Test_Set)

## [1] 5885   59
```

Let’s check the remaining variables in the reduced sets.

```
head(Test_Set)

##     X user_name raw_timestamp_part_1 raw_timestamp_part_2   cvtd_timestamp
## 5   5  carlitos           1323084232               196328 05/12/2011 11:23
## 7   7  carlitos           1323084232               368296 05/12/2011 11:23
## 8   8  carlitos           1323084232               440390 05/12/2011 11:23
## 13 13  carlitos           1323084232               560359 05/12/2011 11:23
## 14 14  carlitos           1323084232               576390 05/12/2011 11:23
## 22 22  carlitos           1323084232               892313 05/12/2011 11:23
##    num_window roll_belt pitch_belt yaw_belt total_accel_belt gyros_belt_x
## 5          12      1.48       8.07    -94.4                3         0.02
## 7          12      1.42       8.09    -94.4                3         0.02
## 8          12      1.42       8.13    -94.4                3         0.02
## 13         12      1.42       8.20    -94.4                3         0.02
## 14         12      1.42       8.21    -94.4                3         0.02
## 22         12      1.57       8.09    -94.4                3         0.02
##    gyros_belt_y gyros_belt_z accel_belt_x accel_belt_y accel_belt_z
## 5          0.02        -0.02          -21            2           24
## 7          0.00        -0.02          -22            3           21
```

```
## 8          0.00        -0.02           -22            4           21
## 13         0.00         0.00           -22            4           21
## 14         0.00        -0.02           -22            4           21
## 22         0.02        -0.02           -21            3           21
##    magnet_belt_x magnet_belt_y magnet_belt_z roll_arm pitch_arm yaw_arm
## 5             -6           600          -302     -128      22.1    -161
## 7             -4           599          -311     -128      21.9    -161
## 8             -2           603          -313     -128      21.8    -161
## 13            -3           606          -309     -128      21.4    -161
## 14            -8           598          -310     -128      21.4    -161
## 22            -2           604          -313     -129      20.8    -161
##    total_accel_arm gyros_arm_x gyros_arm_y gyros_arm_z accel_arm_x
accel_arm_y
## 5               34        0.00       -0.03        0.00        -289
111
## 7               34        0.00       -0.03        0.00        -289
111
## 8               34        0.02       -0.02        0.00        -289
111
## 13              34        0.02       -0.02       -0.02        -287
111
## 14              34        0.02        0.00       -0.03        -288
111
## 22              34        0.03       -0.02       -0.02        -289
111
##    accel_arm_z magnet_arm_x magnet_arm_y magnet_arm_z roll_dumbbell
## 5         -123         -374          337          506      13.37872
## 7         -125         -373          336          509      13.12695
## 8         -124         -372          338          510      12.75083
## 13        -124         -372          338          509      13.38246
## 14        -124         -371          331          523      13.41048
## 22        -123         -372          338          510      13.37872
##    pitch_dumbbell yaw_dumbbell total_accel_dumbbell gyros_dumbbell_x
## 5       -70.42856    -84.85306                   37             0.00
## 7       -70.24757    -85.09961                   37             0.00
## 8       -70.34768    -85.09708                   37             0.00
## 13      -70.81759    -84.46500                   37             0.00
## 14      -70.99594    -84.28005                   37             0.02
## 22      -70.42856    -84.85306                   37             0.00
##    gyros_dumbbell_y gyros_dumbbell_z accel_dumbbell_x accel_dumbbell_y
## 5             -0.02             0.00             -233               48
## 7             -0.02             0.00             -232               47
## 8             -0.02             0.00             -234               46
## 13            -0.02            -0.02             -234               48
## 14            -0.02            -0.02             -234               48
## 22            -0.02             0.00             -233               48
##    accel_dumbbell_z magnet_dumbbell_x magnet_dumbbell_y magnet_dumbbell_z
## 5              -270              -554               292               -68
## 7              -270              -551               295               -70
## 8              -272              -555               300               -74
```

```
## 13             -269              -552              302               -69
## 14             -268              -554              295               -68
## 22             -270              -554              301               -65
##    roll_forearm pitch_forearm yaw_forearm total_accel_forearm
gyros_forearm_x
## 5          28.0         -63.9        -152                  36
0.02
## 7          27.9         -63.9        -152                  36
0.02
## 8          27.8         -63.8        -152                  36
0.02
## 13         27.2         -63.9        -151                  36
0.00
## 14         27.2         -63.9        -151                  36
0.00
## 22         27.0         -63.9        -151                  36
0.02
##    gyros_forearm_y gyros_forearm_z accel_forearm_x accel_forearm_y
## 5             0.00           -0.02             189             206
## 7             0.00           -0.02             195             205
## 8            -0.02            0.00             193             205
## 13            0.00           -0.03             193             205
## 14           -0.02           -0.03             193             202
## 22           -0.03           -0.02             191             206
##    accel_forearm_z magnet_forearm_x magnet_forearm_y magnet_forearm_z
classe
## 5             -214              -17              655              473
A
## 7             -215              -18              659              470
A
## 8             -213               -9              660              474
A
## 13            -215              -15              655              472
A
## 14            -214              -14              659              478
A
## 22            -213              -17              654              478
A
```

As we see, the first five variables are just identification variables, so we need to clean those too.

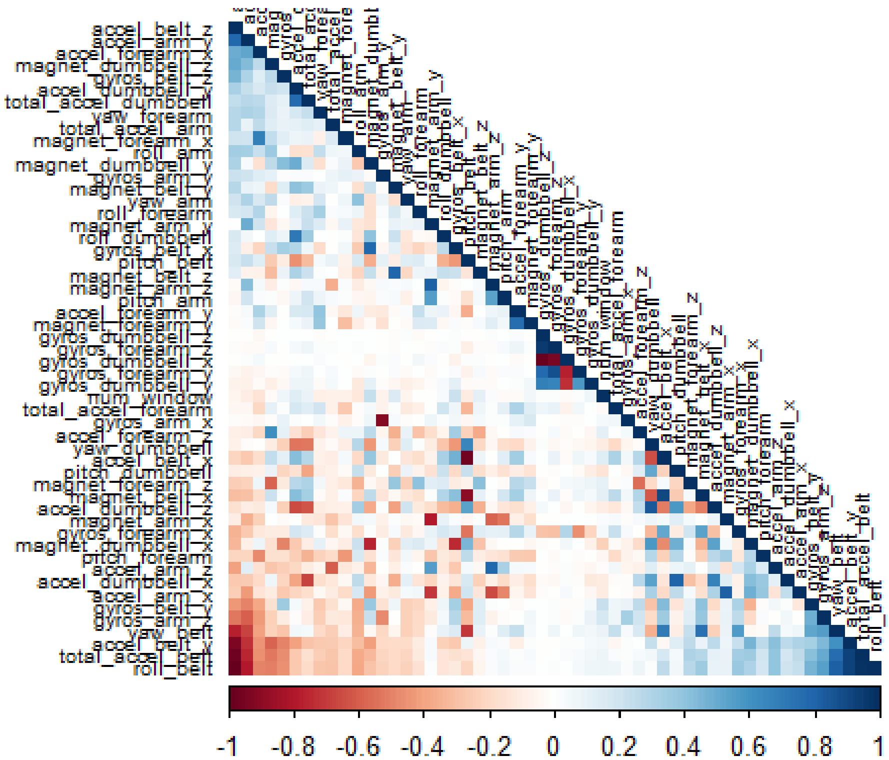
```
Train_Set <- Train_Set[, -(1:5)]
Test_Set  <- Test_Set[, -(1:5)]

dim(Train_Set)

## [1] 13737    54

dim(Test_Set)
```

```
## [1] 5885   54
```

So, after removing mostly NA variables, NZV variables and the first five identification variables we are left to 54 final variables.

d)Correlation Analysis

```
corelation_matrix <- cor(Train_Set[, -54])
corrplot(corelation_matrix, order = "FPC", method = "color", type = "lower",
         tl.cex = 0.6, tl.col = rgb(0, 0, 0))
```



Here, in the correlation matrix, the red and blue colors are associated with -1 and 1 correlation respectively. The bright colors show small amount of correlations between the remaining 54 variables.

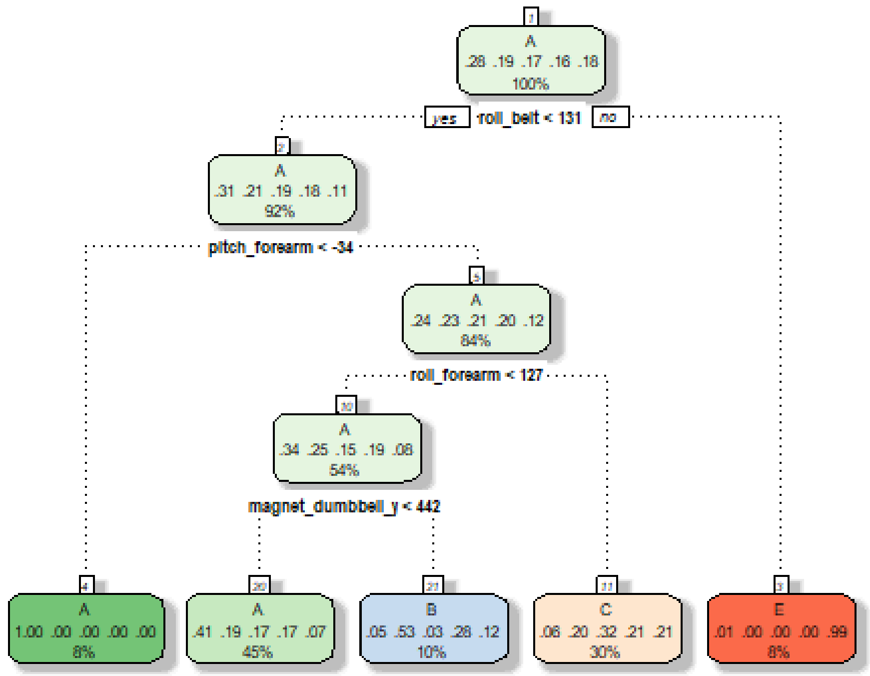
IV.Buliding and Testing the Models

a)Model 1:Decision Tree

i) The Model

The first model that we will use is Decision Trees. At first we need to get the model, then we use fancyRpartPlot() function to plot the classification tree.

```
control_var <- trainControl(method="cv", number=3, verboseIter=F)
DT_Model <- train(classe~., data=Test_Set, method="rpart", trControl =
control_var, tuneLength = 3)
fancyRpartPlot(DT_Model$finalModel)
```



Rattle 2021-May-18 19:26:31 asus

ii) The prediction

Now that we already have the model, we need to validate it on Train_Set.

```
TS_Factored <- factor(Train_Set$classe)
DT_Prediction <- predict(DT_Model, Train_Set)
cmtree <- confusionMatrix(DT_Prediction, TS_Factored)
cmtree

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3571 1162 1164 1028  358
##          B   47  636   35  383  137
##          C  280  860 1197  841  878
##          D    0    0    0    0    0
##          E    8    0    0    0 1152
##
## Overall Statistics
##
##                Accuracy : 0.4773
##                  95% CI : (0.4689, 0.4856)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3165
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9142  0.23928  0.49958   0.0000  0.45624
## Specificity            0.6224  0.94566  0.74791   1.0000  0.99929
## Pos Pred Value         0.4903  0.51373  0.29512      NaN  0.99310
## Neg Pred Value         0.9481  0.83823  0.87615   0.8361  0.89083
## Prevalence             0.2843  0.19349  0.17442   0.1639  0.18381
## Detection Rate         0.2600  0.04630  0.08714   0.0000  0.08386
## Detection Prevalence   0.5302  0.09012  0.29526   0.0000  0.08444
## Balanced Accuracy      0.7683  0.59247  0.62374   0.5000  0.72776
```

As we see, the accuracy rate is 0.4773 and the out of sample error rate is 1-accuracy = 0.5227

b)Model 2:Random Forest

i) The Model

Our second model is Random Forest.

```
control_RF <- trainControl(method="cv", number=3, verboseIter=FALSE)
RF_Model <- train(classe ~ ., data=Train_Set, method="rf",
trControl=control_RF)
RF_Model$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: 0.26%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3904    1    0    0    1 0.0005120328
## B   10 2644    4    0    0 0.0052671181
## C    0    5 2391    0    0 0.0020868114
## D    0    0    7 2244    1 0.0035523979
## E    0    1    0    6 2518 0.0027722772
```

ii) The prediction

```
#RF <- train(classe~., data=Train_Set, method="rf", trControl = control_var)
RF_Prediction <- predict(RF_Model, Train_Set)
cmrf <- confusionMatrix(RF_Prediction, TS_Factored)
cmrf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 3906    0    0    0    0
##          B    0 2658    0    0    0
##          C    0    0 2396    0    0
##          D    0    0    0 2252    0
##          E    0    0    0    0 2525
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9997, 1)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   1.0000   1.0000
## Specificity            1.0000   1.0000   1.0000   1.0000   1.0000
## Pos Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   1.0000   1.0000
## Prevalence             0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2843   0.1935   0.1744   0.1639   0.1838
## Detection Prevalence   0.2843   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      1.0000   1.0000   1.0000   1.0000   1.0000
```

The validation test shows that the accuracy rate is 1 and therefore the out of sample error rate will be 0 or something very close to it. Maybe the accuracy is 1 because of overfiting.

b)Model 3:Generalized Boosted Model (GBM)

i) The Model

We already have a model which has an accuracy rate of 1, however we still perform the training of the third model which is GBM, Generalized Boosted Model.

```
control_GBM <- trainControl(method = "repeatedcv", number = 5, repeats = 1)
GBM_Model  <- train(classe ~ ., data=Train_Set, method = "gbm", trControl =
control_GBM)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1291
##      2        1.5252             nan     0.1000    0.0886
##      3        1.4658             nan     0.1000    0.0645
```

```
##      4        1.4219             nan     0.1000    0.0552
##      5        1.3860             nan     0.1000    0.0440
##      6        1.3571             nan     0.1000    0.0447
##      7        1.3289             nan     0.1000    0.0376
##      8        1.3049             nan     0.1000    0.0340
##      9        1.2804             nan     0.1000    0.0302
##     10        1.2589             nan     0.1000    0.0311
##     20        1.0955             nan     0.1000    0.0186
##     40        0.9117             nan     0.1000    0.0092
##     60        0.8003             nan     0.1000    0.0049
##     80        0.7212             nan     0.1000    0.0047
##    100        0.6581             nan     0.1000    0.0042
##    120        0.6034             nan     0.1000    0.0024
##    140        0.5571             nan     0.1000    0.0022
##    150        0.5340             nan     0.1000    0.0025
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1799
##      2        1.4877             nan     0.1000    0.1349
##      3        1.3997             nan     0.1000    0.0990
##      4        1.3351             nan     0.1000    0.0886
##      5        1.2780             nan     0.1000    0.0686
##      6        1.2336             nan     0.1000    0.0701
##      7        1.1879             nan     0.1000    0.0639
##      8        1.1476             nan     0.1000    0.0565
##      9        1.1113             nan     0.1000    0.0457
##     10        1.0822             nan     0.1000    0.0468
##     20        0.8509             nan     0.1000    0.0280
##     40        0.6255             nan     0.1000    0.0200
##     60        0.4855             nan     0.1000    0.0050
##     80        0.3972             nan     0.1000    0.0097
##    100        0.3239             nan     0.1000    0.0020
##    120        0.2710             nan     0.1000    0.0059
##    140        0.2327             nan     0.1000    0.0035
##    150        0.2125             nan     0.1000    0.0028
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2362
##      2        1.4603             nan     0.1000    0.1673
##      3        1.3567             nan     0.1000    0.1282
##      4        1.2761             nan     0.1000    0.1166
##      5        1.2033             nan     0.1000    0.0870
##      6        1.1467             nan     0.1000    0.0762
##      7        1.0974             nan     0.1000    0.0626
##      8        1.0569             nan     0.1000    0.0816
##      9        1.0068             nan     0.1000    0.0614
##     10        0.9672             nan     0.1000    0.0558
##     20        0.7006             nan     0.1000    0.0273
##     40        0.4631             nan     0.1000    0.0121
##     60        0.3337             nan     0.1000    0.0059
```

```
##     80        0.2564             nan     0.1000    0.0041
##    100        0.1995             nan     0.1000    0.0032
##    120        0.1537             nan     0.1000    0.0016
##    140        0.1249             nan     0.1000    0.0013
##    150        0.1125             nan     0.1000    0.0008
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1286
##      2        1.5224             nan     0.1000    0.0833
##      3        1.4651             nan     0.1000    0.0714
##      4        1.4188             nan     0.1000    0.0533
##      5        1.3830             nan     0.1000    0.0481
##      6        1.3503             nan     0.1000    0.0414
##      7        1.3238             nan     0.1000    0.0388
##      8        1.2978             nan     0.1000    0.0316
##      9        1.2769             nan     0.1000    0.0400
##     10        1.2507             nan     0.1000    0.0322
##     20        1.0874             nan     0.1000    0.0186
##     40        0.9072             nan     0.1000    0.0098
##     60        0.7977             nan     0.1000    0.0072
##     80        0.7157             nan     0.1000    0.0060
##    100        0.6506             nan     0.1000    0.0051
##    120        0.5975             nan     0.1000    0.0033
##    140        0.5503             nan     0.1000    0.0032
##    150        0.5297             nan     0.1000    0.0024
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1936
##      2        1.4853             nan     0.1000    0.1362
##      3        1.3986             nan     0.1000    0.1047
##      4        1.3303             nan     0.1000    0.0872
##      5        1.2744             nan     0.1000    0.0704
##      6        1.2283             nan     0.1000    0.0728
##      7        1.1805             nan     0.1000    0.0588
##      8        1.1422             nan     0.1000    0.0517
##      9        1.1091             nan     0.1000    0.0437
##     10        1.0813             nan     0.1000    0.0542
##     20        0.8660             nan     0.1000    0.0273
##     40        0.6266             nan     0.1000    0.0151
##     60        0.4854             nan     0.1000    0.0063
##     80        0.3922             nan     0.1000    0.0037
##    100        0.3264             nan     0.1000    0.0028
##    120        0.2706             nan     0.1000    0.0046
##    140        0.2309             nan     0.1000    0.0045
##    150        0.2121             nan     0.1000    0.0025
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2457
##      2        1.4560             nan     0.1000    0.1669
##      3        1.3527             nan     0.1000    0.1256
```

```
##      4        1.2720             nan     0.1000    0.0983
##      5        1.2075             nan     0.1000    0.0999
##      6        1.1454             nan     0.1000    0.0800
##      7        1.0934             nan     0.1000    0.0676
##      8        1.0504             nan     0.1000    0.0747
##      9        1.0043             nan     0.1000    0.0657
##     10        0.9636             nan     0.1000    0.0475
##     20        0.6860             nan     0.1000    0.0252
##     40        0.4497             nan     0.1000    0.0147
##     60        0.3260             nan     0.1000    0.0098
##     80        0.2463             nan     0.1000    0.0056
##    100        0.1930             nan     0.1000    0.0033
##    120        0.1540             nan     0.1000    0.0016
##    140        0.1261             nan     0.1000    0.0014
##    150        0.1139             nan     0.1000    0.0013
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1268
##      2        1.5241             nan     0.1000    0.0830
##      3        1.4664             nan     0.1000    0.0662
##      4        1.4222             nan     0.1000    0.0534
##      5        1.3868             nan     0.1000    0.0497
##      6        1.3545             nan     0.1000    0.0401
##      7        1.3271             nan     0.1000    0.0435
##      8        1.3007             nan     0.1000    0.0373
##      9        1.2757             nan     0.1000    0.0302
##     10        1.2562             nan     0.1000    0.0328
##     20        1.0953             nan     0.1000    0.0189
##     40        0.9100             nan     0.1000    0.0094
##     60        0.7993             nan     0.1000    0.0045
##     80        0.7179             nan     0.1000    0.0052
##    100        0.6519             nan     0.1000    0.0033
##    120        0.5980             nan     0.1000    0.0033
##    140        0.5541             nan     0.1000    0.0030
##    150        0.5330             nan     0.1000    0.0019
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1910
##      2        1.4865             nan     0.1000    0.1290
##      3        1.4002             nan     0.1000    0.1052
##      4        1.3310             nan     0.1000    0.0939
##      5        1.2717             nan     0.1000    0.0719
##      6        1.2241             nan     0.1000    0.0715
##      7        1.1793             nan     0.1000    0.0636
##      8        1.1385             nan     0.1000    0.0446
##      9        1.1094             nan     0.1000    0.0536
##     10        1.0762             nan     0.1000    0.0379
##     20        0.8563             nan     0.1000    0.0212
##     40        0.6240             nan     0.1000    0.0100
##     60        0.4882             nan     0.1000    0.0088
```

```
##     80        0.3946             nan     0.1000    0.0035
##    100        0.3339             nan     0.1000    0.0035
##    120        0.2793             nan     0.1000    0.0025
##    140        0.2381             nan     0.1000    0.0024
##    150        0.2204             nan     0.1000    0.0024
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2478
##      2        1.4547             nan     0.1000    0.1568
##      3        1.3538             nan     0.1000    0.1317
##      4        1.2687             nan     0.1000    0.0960
##      5        1.2071             nan     0.1000    0.0877
##      6        1.1504             nan     0.1000    0.0768
##      7        1.1015             nan     0.1000    0.0791
##      8        1.0538             nan     0.1000    0.0881
##      9        1.0005             nan     0.1000    0.0602
##     10        0.9627             nan     0.1000    0.0585
##     20        0.6934             nan     0.1000    0.0219
##     40        0.4552             nan     0.1000    0.0140
##     60        0.3322             nan     0.1000    0.0070
##     80        0.2524             nan     0.1000    0.0043
##    100        0.1974             nan     0.1000    0.0019
##    120        0.1574             nan     0.1000    0.0035
##    140        0.1277             nan     0.1000    0.0026
##    150        0.1146             nan     0.1000    0.0015
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1257
##      2        1.5223             nan     0.1000    0.0827
##      3        1.4649             nan     0.1000    0.0698
##      4        1.4183             nan     0.1000    0.0548
##      5        1.3821             nan     0.1000    0.0435
##      6        1.3529             nan     0.1000    0.0463
##      7        1.3243             nan     0.1000    0.0411
##      8        1.2974             nan     0.1000    0.0367
##      9        1.2741             nan     0.1000    0.0352
##     10        1.2491             nan     0.1000    0.0298
##     20        1.0859             nan     0.1000    0.0189
##     40        0.9054             nan     0.1000    0.0113
##     60        0.7902             nan     0.1000    0.0072
##     80        0.7106             nan     0.1000    0.0039
##    100        0.6455             nan     0.1000    0.0034
##    120        0.5920             nan     0.1000    0.0027
##    140        0.5454             nan     0.1000    0.0017
##    150        0.5251             nan     0.1000    0.0028
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1956
##      2        1.4819             nan     0.1000    0.1316
##      3        1.3960             nan     0.1000    0.1100
```

```
##      4        1.3251             nan     0.1000    0.0877
##      5        1.2671             nan     0.1000    0.0754
##      6        1.2192             nan     0.1000    0.0724
##      7        1.1738             nan     0.1000    0.0674
##      8        1.1309             nan     0.1000    0.0587
##      9        1.0941             nan     0.1000    0.0438
##     10        1.0663             nan     0.1000    0.0364
##     20        0.8434             nan     0.1000    0.0272
##     40        0.6160             nan     0.1000    0.0229
##     60        0.4831             nan     0.1000    0.0103
##     80        0.3869             nan     0.1000    0.0136
##    100        0.3167             nan     0.1000    0.0047
##    120        0.2681             nan     0.1000    0.0043
##    140        0.2285             nan     0.1000    0.0043
##    150        0.2101             nan     0.1000    0.0013
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2455
##      2        1.4529             nan     0.1000    0.1672
##      3        1.3482             nan     0.1000    0.1283
##      4        1.2648             nan     0.1000    0.1065
##      5        1.1967             nan     0.1000    0.0982
##      6        1.1361             nan     0.1000    0.0796
##      7        1.0854             nan     0.1000    0.0640
##      8        1.0444             nan     0.1000    0.0797
##      9        0.9973             nan     0.1000    0.0504
##     10        0.9656             nan     0.1000    0.0645
##     20        0.6857             nan     0.1000    0.0304
##     40        0.4465             nan     0.1000    0.0121
##     60        0.3238             nan     0.1000    0.0099
##     80        0.2483             nan     0.1000    0.0070
##    100        0.1916             nan     0.1000    0.0034
##    120        0.1515             nan     0.1000    0.0026
##    140        0.1228             nan     0.1000    0.0020
##    150        0.1113             nan     0.1000    0.0021
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1259
##      2        1.5227             nan     0.1000    0.0896
##      3        1.4631             nan     0.1000    0.0684
##      4        1.4183             nan     0.1000    0.0542
##      5        1.3832             nan     0.1000    0.0474
##      6        1.3521             nan     0.1000    0.0425
##      7        1.3243             nan     0.1000    0.0405
##      8        1.2980             nan     0.1000    0.0308
##      9        1.2776             nan     0.1000    0.0370
##     10        1.2520             nan     0.1000    0.0322
##     20        1.0878             nan     0.1000    0.0193
##     40        0.9067             nan     0.1000    0.0111
##     60        0.7983             nan     0.1000    0.0070
```

```
##     80        0.7153             nan     0.1000    0.0038
##    100        0.6496             nan     0.1000    0.0040
##    120        0.5976             nan     0.1000    0.0035
##    140        0.5503             nan     0.1000    0.0025
##    150        0.5288             nan     0.1000    0.0029
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1921
##      2        1.4854             nan     0.1000    0.1342
##      3        1.3993             nan     0.1000    0.1052
##      4        1.3304             nan     0.1000    0.0819
##      5        1.2775             nan     0.1000    0.0792
##      6        1.2268             nan     0.1000    0.0670
##      7        1.1833             nan     0.1000    0.0565
##      8        1.1462             nan     0.1000    0.0555
##      9        1.1118             nan     0.1000    0.0537
##     10        1.0788             nan     0.1000    0.0379
##     20        0.8522             nan     0.1000    0.0272
##     40        0.6177             nan     0.1000    0.0096
##     60        0.4829             nan     0.1000    0.0113
##     80        0.3922             nan     0.1000    0.0049
##    100        0.3264             nan     0.1000    0.0030
##    120        0.2767             nan     0.1000    0.0030
##    140        0.2324             nan     0.1000    0.0020
##    150        0.2169             nan     0.1000    0.0013
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2384
##      2        1.4552             nan     0.1000    0.1675
##      3        1.3481             nan     0.1000    0.1224
##      4        1.2702             nan     0.1000    0.1031
##      5        1.2051             nan     0.1000    0.0818
##      6        1.1523             nan     0.1000    0.0731
##      7        1.1052             nan     0.1000    0.0924
##      8        1.0496             nan     0.1000    0.0738
##      9        1.0031             nan     0.1000    0.0558
##     10        0.9669             nan     0.1000    0.0591
##     20        0.7078             nan     0.1000    0.0486
##     40        0.4526             nan     0.1000    0.0169
##     60        0.3280             nan     0.1000    0.0062
##     80        0.2451             nan     0.1000    0.0051
##    100        0.1937             nan     0.1000    0.0029
##    120        0.1528             nan     0.1000    0.0036
##    140        0.1219             nan     0.1000    0.0010
##    150        0.1104             nan     0.1000    0.0014
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2435
##      2        1.4555             nan     0.1000    0.1621
##      3        1.3515             nan     0.1000    0.1331
```

```
##      4        1.2678             nan     0.1000    0.1048
##      5        1.2021             nan     0.1000    0.0968
##      6        1.1410             nan     0.1000    0.0887
##      7        1.0872             nan     0.1000    0.0785
##      8        1.0385             nan     0.1000    0.0599
##      9        1.0014             nan     0.1000    0.0530
##     10        0.9675             nan     0.1000    0.0468
##     20        0.6883             nan     0.1000    0.0251
##     40        0.4565             nan     0.1000    0.0099
##     60        0.3321             nan     0.1000    0.0041
##     80        0.2558             nan     0.1000    0.0046
##    100        0.2016             nan     0.1000    0.0031
##    120        0.1576             nan     0.1000    0.0038
##    140        0.1264             nan     0.1000    0.0019
##    150        0.1126             nan     0.1000    0.0020
```

```
GBM_Model$finalModel
```

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

ii) The prediction

```
GBM_Prediction <- predict(GBM_Model, newdata=Test_Set)
```

```
GBM_Conf_Matrix <- confusionMatrix(factor(GBM_Prediction),
factor(Test_Set$classe))
GBM_Conf_Matrix
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 1672   10    0    0    0
##          B    1 1119    9    4    1
##          C    0   10 1014    8    0
##          D    1    0    3  950    6
##          E    0    0    0    2 1075
## 
## Overall Statistics
## 
##                Accuracy : 0.9907
##                  95% CI : (0.9879, 0.993)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
## 
##                   Kappa : 0.9882
## 
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
```

```
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9988   0.9824   0.9883   0.9855   0.9935
## Specificity            0.9976   0.9968   0.9963   0.9980   0.9996
## Pos Pred Value         0.9941   0.9868   0.9826   0.9896   0.9981
## Neg Pred Value         0.9995   0.9958   0.9975   0.9972   0.9985
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1901   0.1723   0.1614   0.1827
## Detection Prevalence   0.2858   0.1927   0.1754   0.1631   0.1830
## Balanced Accuracy      0.9982   0.9896   0.9923   0.9917   0.9966
```

And, as predicted, the accuracy rate of GBM is less than the same of Random Forest model. Here, the out of sample error rate is very little, 0.0084.

V. Best Model Selection and its Validation on test_data

The three models used gave us the following accuracy rates: Decision Tree - 0.4773 Random Forest - 1 Generalized Boosted Model - 0.9891

So, we choose the model with highest accuracy rate which is Random Forest. And now we will validate and get results of the 20 observations of test_data via applying Random Forest Model.

```
Predict_Results <- predict(RF_Model, test_data)
Predict_Results
```

```
##  [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
```