Exercise prediction and Quality Analysis

Danish Tamboli

9/2/2020

Background:

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks.

One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it, our goal will be to use data from accelerometers on the belt, forearm, arm, and dumbbell 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 (see the section on the Weight Lifting Exercise Dataset).

Dataset Download:

Training Data is available here: Training Data

Testing Data is available here: Testing Data

```
# Checking if given training set data is locally available, else to download.
if(!file.exists("pml-training.csv")){
        download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv",
                      "pml-training.csv")
}

# Checking if given testing set data is locally available, else to download.
if(!file.exists("pml-testing.csv")){
        download.file("https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv",
                      "pml-testing.csv")
}
```

Loading and Cleaning of Data:

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.2
```

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

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

```
## Warning: package 'ggplot2' was built under R version 4.0.2
```

```
main_data <- read.csv("pml-training.csv")
given_testing <- read.csv("pml-testing.csv")

# Removing variables that have no Variation,as they do not help in the distinction process.
main_data <- main_data[,-nearZeroVar(main_data)]

# Removing Columns that have NA
main_data <- main_data[, colSums(is.na(main_data)) == 0]

# Removing Columns with Reading Number,Participant Name and Timestamps as they are of no use to us.
main_data <- main_data[,-(1:5)]
```

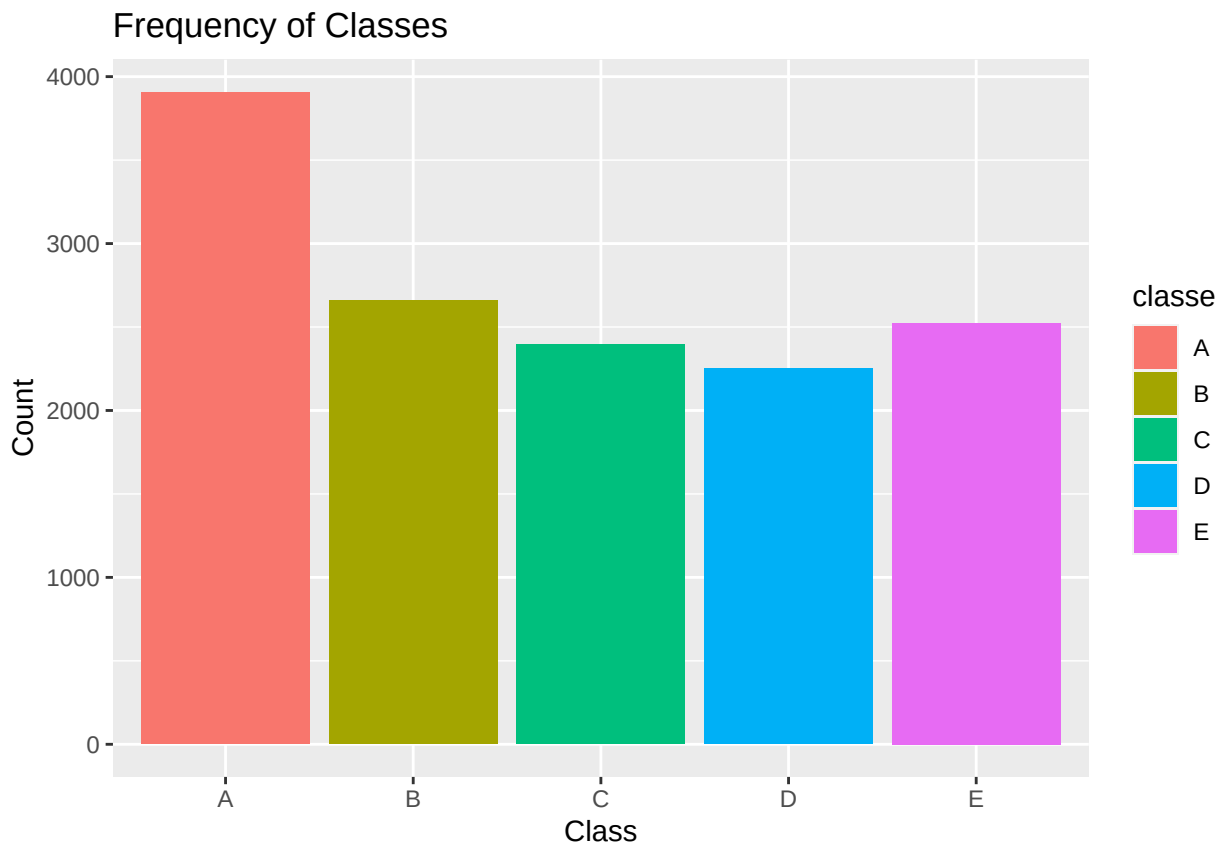
We will use the given_testing data as a final testing set (Validation).

Creating Training and Testing Sets:

```
set.seed(233)

# Dividing the original Training Data (main_data) into Training and Testing sets in 7:3 ratio
inTrain <- createDataPartition(main_data$classe,p = 0.7, list = FALSE)
training <- main_data[inTrain,]
testing <- main_data[-inTrain,]
```

```
library(ggplot2)
# Frequency Plot
ggplot(data = training) + geom_bar(aes(classe,fill=classe)) + labs(x = "Class", y = "Count",
                                                                  title = "Frequency of Classes")
```



Prediction Models:

```
set.seed(212)
library(rpart)
rpmodel <- train(classe ~ .,data = training,method = "rpart")
```

```
library(rattle)
```

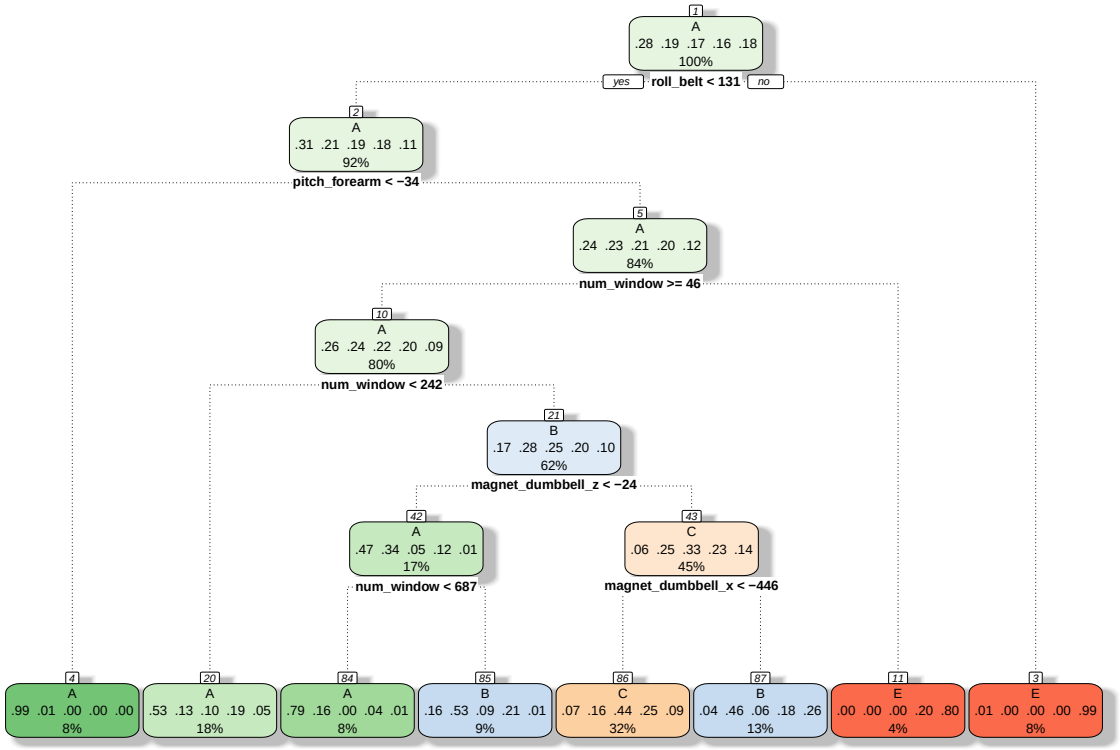
```
## Warning: package 'rattle' was built under R version 4.0.2
```

```
## 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.
```

```
fancyRpartPlot(rpmodel$finalModel)
```



Rattle 2020−Sep−02 22:03:51 danis

```
library(rpart)
set.seed(133)
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.0.2
```

```
## 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
```

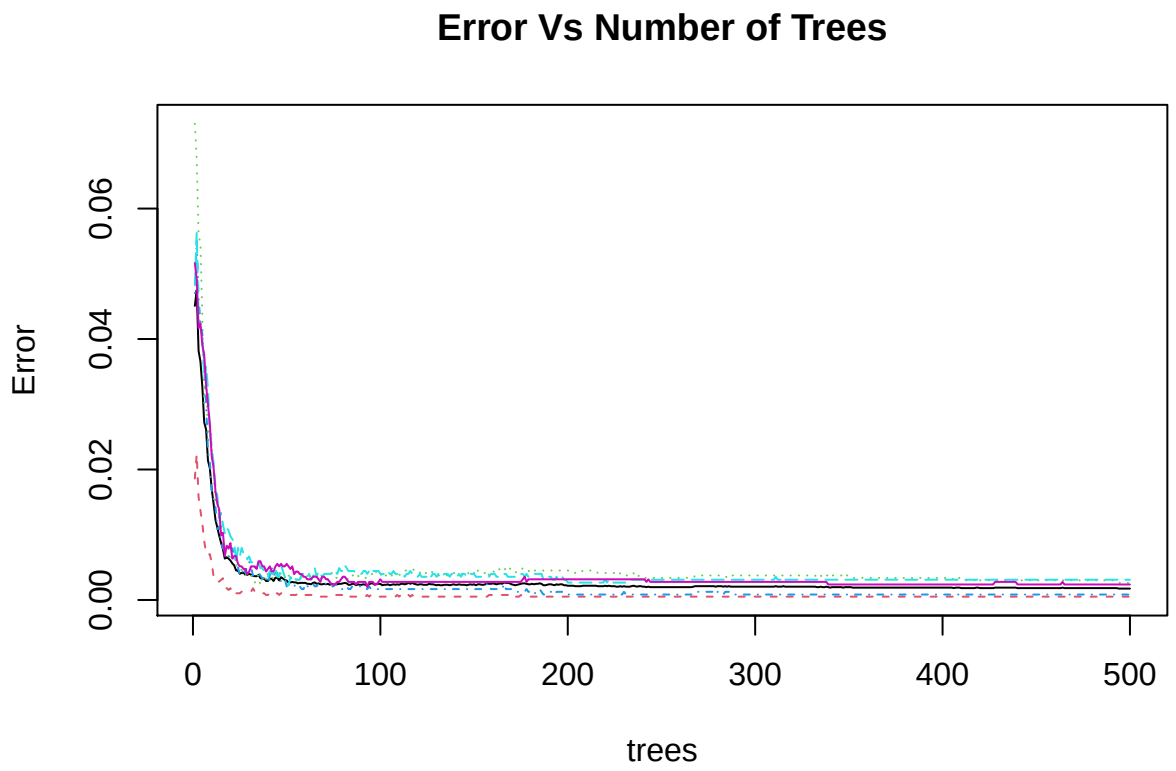
```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
rfmodel <- train(classe ~ .,data = training,method = "rf",
                 trControl= trainControl(method = "cv",number = 5, verboseIter = FALSE),allowParallel ='

rfmodel$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, allowParallel = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.17%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3904    1    0    0    1 0.0005120328
## B    5 2651    2    0    0 0.0026335591
## C    0    2 2394    0    0 0.0008347245
## D    0    0    6 2245    1 0.0031083481
## E    0    0    0    6 2519 0.0023762376
```

```
plot(rfmodel$finalModel,main = "Error Vs Number of Trees")
```



We can clearly see here that after around 100 Trees the Error doesn't change a lot,so we create a new model with just 100 Trees,This will help us in multiple ways:

- Helps in avoiding Over-fitting of Data
- Helps in reduction complexity of algorithm which in turn reduces execution time of the model.

```
set.seed(294)
rfmodel1 <- train(classe ~ .,data = training,method = "rf",ntree=100,
                  trControl= trainControl(method = "cv",number = 5, verboseIter = FALSE),allowParallel =

rfmodel1$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 100, mtry = param$mtry, allowParallel = TRUE)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 27
##
##         OOB estimate of  error rate: 0.2%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 3904    2    0    0    0 0.0005120328
## B    6 2648    4    0    0 0.0037622272
## C    0    4 2392    0    0 0.0016694491
## D    0    0    6 2245    1 0.0031083481
## E    0    1    0    4 2520 0.0019801980
```

```
starttime<- Sys.time()
predictions <- predict(rfmodel,newdata=testing)
endtime <- Sys.time()

endtime-starttime
```

```
## Time difference of 0.2104521 secs
```

```
confmatrf <- confusionMatrix(predictions,factor(testing$classe))
print(confmatrf)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    2    0    0    0
##          B    0 1136    4    0    0
##          C    0    1 1022    5    0
##          D    0    0    0  959    1
##          E    0    0    0    0 1081
##
## Overall Statistics
##
##                Accuracy : 0.9978
##                  95% CI : (0.9962, 0.9988)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9972
##
```

```
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9974   0.9961   0.9948   0.9991
## Specificity            0.9995   0.9992   0.9988   0.9998   1.0000
## Pos Pred Value         0.9988   0.9965   0.9942   0.9990   1.0000
## Neg Pred Value         1.0000   0.9994   0.9992   0.9990   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1930   0.1737   0.1630   0.1837
## Detection Prevalence   0.2848   0.1937   0.1747   0.1631   0.1837
## Balanced Accuracy      0.9998   0.9983   0.9974   0.9973   0.9995
```

```
five_hundred_tree <- diag(table(predictions,testing$classe))
```

```
starttime<- Sys.time()
predictions <- predict(rfmodel1,newdata=testing)
endtime <- Sys.time()

endtime-starttime
```

```
## Time difference of 0.06283092 secs
```

```
confmatrf1 <- confusionMatrix(predictions,factor(testing$classe))
print(confmatrf1)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1674    4    0    0    0
##          B    0 1134    4    0    0
##          C    0    1 1022    4    0
##          D    0    0    0  960    1
##          E    0    0    0    0 1081
##
## Overall Statistics
##
##                Accuracy : 0.9976
##                  95% CI : (0.996, 0.9987)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.997
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9956   0.9961   0.9959   0.9991
```

```
## Specificity            0.9991   0.9992   0.9990   0.9998   1.0000
## Pos Pred Value         0.9976   0.9965   0.9951   0.9990   1.0000
## Neg Pred Value         1.0000   0.9989   0.9992   0.9992   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2845   0.1927   0.1737   0.1631   0.1837
## Detection Prevalence   0.2851   0.1934   0.1745   0.1633   0.1837
## Balanced Accuracy      0.9995   0.9974   0.9975   0.9978   0.9995
```

```
one_hundred_tree <- diag(table(predictions,testing$classe))
```

As we see Both the models (500 Trees and 100 Tress) perform very similarly in terms of accuracy but the execution time of the second model (100 Trees) is about 1/3th of that of the Original model (500 Trees).

```
new_data <- data.frame(one_hundred_tree,five_hundred_tree,
                       as.numeric(table(testing$classe)))

colnames(new_data) <- c("One Hundred Trees","Five Hundred Trees",
                        "Actual Class Distribution")

knitr::kable(new_data,caption = "Comparison of performance of 500 vs 100 Trees")
```

Table 1: Comparison of performance of 500 vs 100 Trees

	One Hundred Trees	Five Hundred Trees	Actual Class Distribution
A	1674	1674	1674
B	1134	1136	1139
C	1022	1022	1026
D	960	959	964
E	1081	1081	1082

We will go along with rfmodel1 (100 Tree Model) as it performs as well as rfmodel (500 Tree model) and considerably better than the Regression Tree model.

Applying Model to predict given Test dataset.

```
predict_given <- predict(rfmodel1,newdata = given_testing)
print(predict_given)
```

```
##  [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
```