practicalMachineLearning-project

maheep-raj

21/10/2020

Introduction

This project is a part of Practical Machine Learning course of JHU Data Science specialization. The instruction statement reads as "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". The goal of the project is to predict the manner in which they did the exercise.

Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. 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).

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. We kindly thank them as they have been very generous in allowing their data to be used for this kind of assignment.

Initializing Libraries

Libraries are initialized in a single chunk for easy reference later and the seed is set.

```
library(randomForest)
library(rpart)
library(ggplot2)
library(caret)
```

```
    library(gbm)
    library(plyr)
    set.seed(17790869)
```

Loading and Cleaning Data

Note: The data was previously downloaded from source, it is not included in repository. We observe the data in RStudio and then load the data into file removing empty elements.

```
    training <- read.csv("./pml-training.csv", na.strings=c("NA","#DIV/0!",""))
    testing <- read.csv("./pml-testing.csv", na.strings=c("NA","#DIV/0!",""))
```

Structuring the Dataset

For this model, we only need a limited set of information as discussed in the Introduction, thus we isolate the data from the source with only the information we need.

```
    features <- names(testing[,colSums(is.na(testing)) == 0])[8:59]
    training <- training[,c(features,"classe")]
    testing <- testing[,c(features,"problem_id")]
```

Seperating the Dev and Train set out of training data

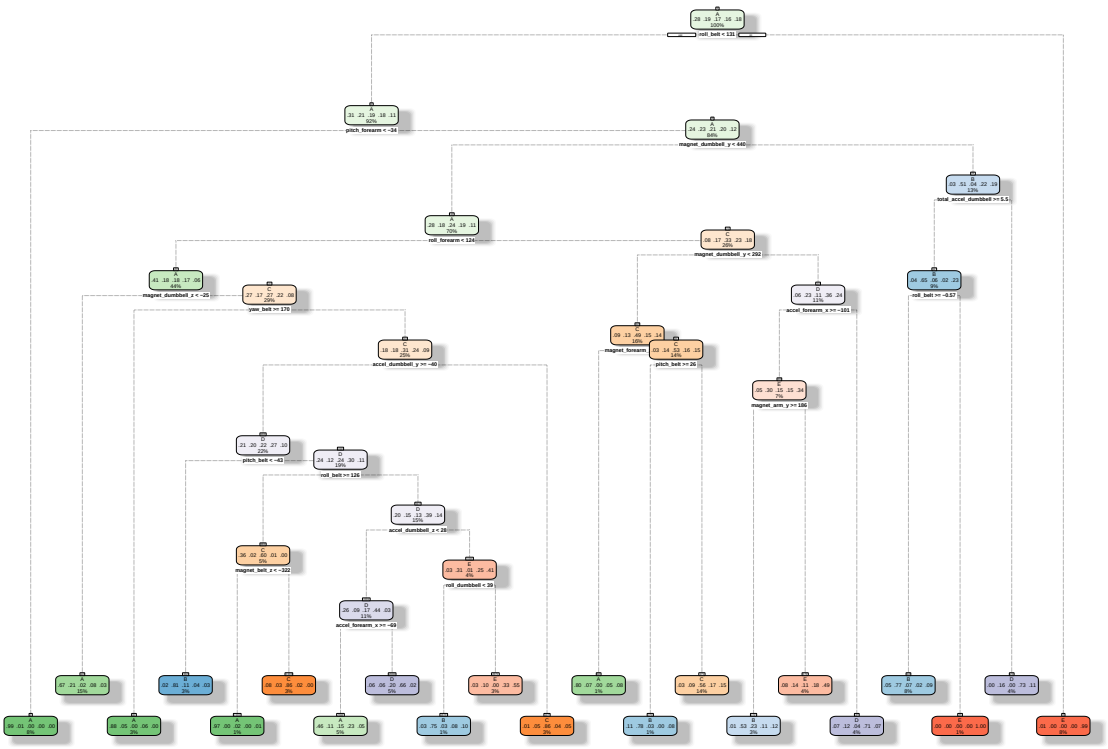
As our test set exists separately, we have to separate our training data into train-set and dev-set. As a rule of thumb I tested a 60:40 split, and 80:20 split. The 80:20 split performed marginally better and is thus the focus of this project report.

```
inTrain <- createDataPartition(training$classe, p=0.8, list=FALSE)
trainSet <- training[inTrain,]
devSet <- training[-inTrain,]
```

Building the Decision Tree

We build a Decision Tree Model on the base data set

```
library(rattle)
training_tree <- rpart(classe ~ ., data = trainSet,
                       method="class",
                       control = rpart.control(method = "cv", number = 10))
fancyRpartPlot(training_tree)
```



Rattle 2020−Oct−22 12:25:48 maheepraj

Predicting with the Decision Tree

We predict the outputs using this Decision Tree Model, we don't expect a good accuracy with the model just yet cause of the out of bound error possibilities.

```
prediction <- predict(training_tree, devSet, type = "class")
classe <- as.factor(devSet$classe)
confusionMatrix(prediction, classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1021  165   52  114   38
##          B   26  441   58   30   51
##          C   29   58  510   86   86
##          D   21   59   45  352   36
##          E   19   36   19   61  510
##
## Overall Statistics
##
##                Accuracy : 0.7224
##                  95% CI : (0.7081, 0.7364)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6457
##
```

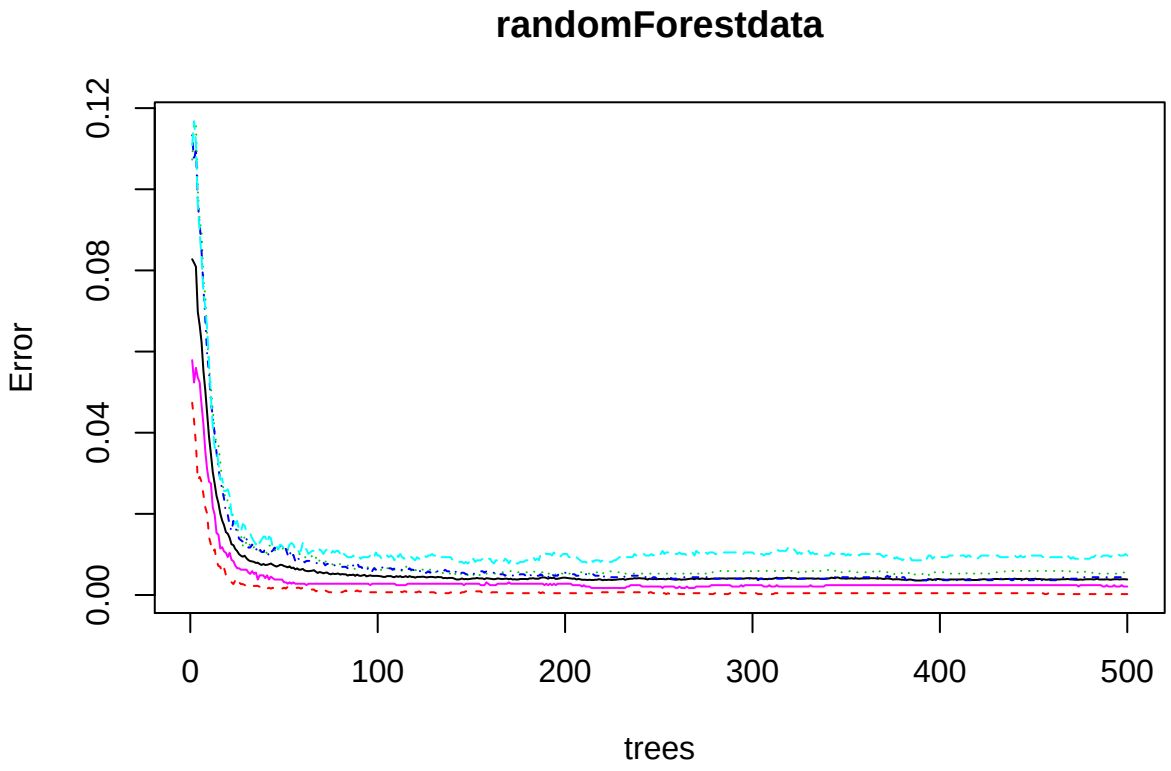
```
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9149   0.5810   0.7456  0.54743   0.7074
## Specificity            0.8685   0.9479   0.9200  0.95091   0.9578
## Pos Pred Value         0.7345   0.7277   0.6632  0.68616   0.7907
## Neg Pred Value         0.9625   0.9041   0.9448  0.91466   0.9356
## Prevalence             0.2845   0.1935   0.1744  0.16391   0.1838
## Detection Rate         0.2603   0.1124   0.1300  0.08973   0.1300
## Detection Prevalence   0.3543   0.1545   0.1960  0.13077   0.1644
## Balanced Accuracy      0.8917   0.7644   0.8328  0.74917   0.8326
```

From the above table we see that the accuracy of the Decision Tree is around 72.24%. This is a low accuracy, we now try to improve the accuracy by trying out a Random Forest Model.

Building Random Forest Model

Now, we build a random forest model

```
randomForestdata <- randomForest(as.factor(classe) ~ ., data = trainSet,
                    method = "rf",
                    importance = T,
                    trControl = trainControl(method = "cv",
                                             classProbs=TRUE,
                                             savePredictions=TRUE,
                                             allowParallel=TRUE,
                                             number = 11))
plot(randomForestdata)
```



Predicting with Random Forest

When the Random forests are created, the model itself carves out a portion of data to avoid out of sample errors, and thus gives a better accuracy.

Let's try and predict the outcomes now and see the accuracy of the same.

```
prediction <- predict(randomForestdata, devSet, type = "class")
classe <- as.factor(devSet$classe)
confusionMatrix(prediction, classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1112    4    0    0    0
##          B    3  752    2    0    0
##          C    0    3  680    3    4
##          D    0    0    2  640    4
##          E    1    0    0    0  713
##
## Overall Statistics
##
##                Accuracy : 0.9934
##                  95% CI : (0.9903, 0.9957)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9916
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.9908   0.9942   0.9953   0.9889
## Specificity            0.9986   0.9984   0.9969   0.9982   0.9997
## Pos Pred Value         0.9964   0.9934   0.9855   0.9907   0.9986
## Neg Pred Value         0.9986   0.9978   0.9988   0.9991   0.9975
## Prevalence             0.2845   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2835   0.1917   0.1733   0.1631   0.1817
## Detection Prevalence   0.2845   0.1930   0.1759   0.1647   0.1820
## Balanced Accuracy      0.9975   0.9946   0.9955   0.9968   0.9943
```

From the confusion matrix we see that the stray data cases are very less and the accuracy of Random Forest Model is around 99.34%. Hence, it would be better than Decision Tree Model to use for predicting the values for the testing data (*pml-testing.csv*).

Predicting on the Testing Data

```
prediction <- predict(randomForestdata, training, type = "class")
classe <- as.factor(training$classe)
confusionMatrix(prediction, classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 5576    4    0    0    0
##          B    3 3790    2    0    0
##          C    0    3 3418    3    4
##          D    0    0    2 3213    4
##          E    1    0    0    0 3599
##
## Overall Statistics
##
##                Accuracy : 0.9987
##                  95% CI : (0.9981, 0.9991)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9983
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9993   0.9982   0.9988   0.9991   0.9978
## Specificity            0.9997   0.9997   0.9994   0.9996   0.9999
## Pos Pred Value         0.9993   0.9987   0.9971   0.9981   0.9997
## Neg Pred Value         0.9997   0.9996   0.9998   0.9998   0.9995
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2842   0.1932   0.1742   0.1637   0.1834
## Detection Prevalence   0.2844   0.1934   0.1747   0.1641   0.1835
## Balanced Accuracy      0.9995   0.9989   0.9991   0.9994   0.9989
```

Thus we see an accuracy of 99.87% on our Testing data set(*pml-testing.csv*).