Predict the manner in which we did the exercise

Cylia YACEF

20/07/2022

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

Projct Purpose

The goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. You may use any of the other variables to predict with. You should create a report describing how you built your model, how you used cross validation, what you think the expected out of sample error is, and why you made the choices you did. You will also use your prediction model to predict 20 different test cases.

Preproccessing the training and testing dataset

Loading the library

```
library(plyr)
library(dplyr)
```

```
## Warning: remplacement de l'importation précédente
'lifecycle::last_warnings' par
## 'rlang::last_warnings' lors du chargement de 'pillar'
```

```
##
## Attachement du package : 'dplyr'

## Les objets suivants sont masqués depuis 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag

## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union

library(lattice)
library(ggplot2)
library(caret)
library(rpart)
library(rpart.plot)
library(RColorBrewer)
library(kernlab)

##
## Attachement du package : 'kernlab'

## L'objet suivant est masqué depuis 'package:ggplot2':
##
##     alpha

library(randomForest)

## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attachement du package : 'randomForest'

## L'objet suivant est masqué depuis 'package:ggplot2':
##
##     margin

## L'objet suivant est masqué depuis 'package:dplyr':
##
##     combine

library(knitr)
library(e1071)
```

```
#leading the data
trainingst <- read.csv("pml-training.csv")
testingst <- read.csv("pml-testing.csv")

dim(trainingst)

## [1] 19622   160

dim(testingst)

## [1]  20 160
```

Cleaning data

we should Exclude the obvious columns i.e “X”, “user_name”, “raw_timestamp_part_1”, “raw_timestamp_part_2”, “cvtd_timestamp”,“roll_belt” which are the first 7 columns. We should also delete missing values and variables with near zero variance.

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

#Removing variables with near zero variance
trainingst<-trainingst[,colSums(is.na(trainingst)) == 0]
testingst <-testingst[,colSums(is.na(testingst)) == 0]

#Removing columns that are not predictors, which are the the seven first
columns
trainingst   <-trainingst[,-c(1:7)]
testingst <-testingst[,-c(1:7)]

dim(trainingst)

## [1] 19622    53

dim(testingst )

## [1] 20 53
```

From the above code block sum(completeCase) == nrows confirm that the number of complete case is equal to number of rows in trainingdf same for testingdf

Now we have only 53 columns(features) are left. we can preproccess the training and testing i.e converting into scales of 0 to 1 and replacing any NA values to average of that columns.

Preprocessing data

removed the near Zero Var columns.

Normalize the data

```
processedData <- function(rawdata) {
  # each columns NA should be replaced with average of that columns
```

```
  for(column in names(rawdata)) {
    if(column == "classe") {
      next;
    }
    columnValue <- as.numeric(rawdata[, column]);
    avgColumnValue <- mean(columnValue, na.rm=TRUE)
    minColumnValue <- min(columnValue, na.rm=TRUE)
    maxColumnValue <- max(columnValue, na.rm=TRUE)
    columnValue[is.na(columnValue)] <- avgColumnValue

    if (maxColumnValue == minColumnValue) {
      next;
    }

    for(i in 1:length(columnValue)) {
      columnValue[i] <- round((columnValue[i] - minColumnValue) /
(maxColumnValue - minColumnValue), 4);
    }

    rawdata[, column] <- columnValue
  }
  rawdata
}
## Get the processed training data frame
trainingst <- processedData(trainingst)
testingst <- processedData(testingst)

dim(trainingst)

## [1] 19622    53

dim(testingst )

## [1] 20 53
```

Partition the data set into training and testing data from trainingst

```
inTrain <- createDataPartition(y = trainingst$classe, p=0.75, list = FALSE)
training <- trainingst[inTrain, ]
testing <- trainingst[-inTrain, ]
```

Training the model

Two methods will be applied to model, and the best one will be used for the(testingst) predictions.

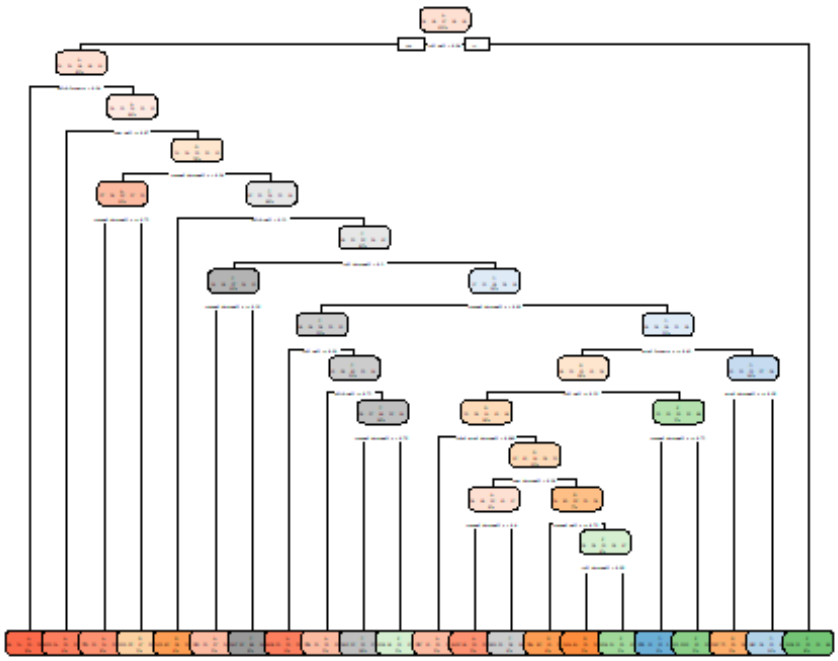
The methods are: Decision Tree and Random Forests.

Training the model with Decision Trees

```
set.seed(40000)
fitDT <- rpart(classe ~ .,training, method="class")
```

```
# Normal plot
rpart.plot(fitDT)

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



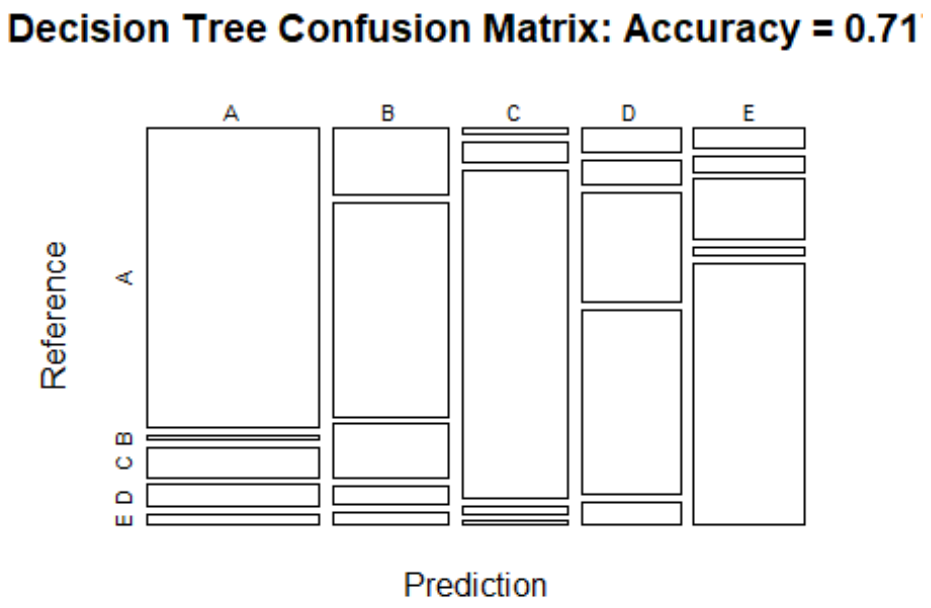
```
#Use model to predict classe in validation testing set
predictionDT <- predict(fitDT, testing, type = "class")

#Estimate the errors of the prediction algorithm in the Decision Tree model
cmdt <-confusionMatrix(as.factor(testing$classe), predictionDT)
cmdt

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1146   16  114   83   36
##          B  174  557  138   48   32
##          C   17   49  768   15    6
##          D   57   51  244  402   50
##          E   51   39  146   22  643
##
## Overall Statistics
##
##                Accuracy : 0.717
##                  95% CI : (0.7041, 0.7295)
##     No Information Rate : 0.2947
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.6418
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.7931   0.7823   0.5447  0.70526   0.8383
## Specificity            0.9280   0.9065   0.9751  0.90725   0.9376
## Pos Pred Value         0.8215   0.5869   0.8982  0.50000   0.7137
## Neg Pred Value         0.9148   0.9608   0.8414  0.95902   0.9690
## Prevalence             0.2947   0.1452   0.2875  0.11623   0.1564
## Detection Rate         0.2337   0.1136   0.1566  0.08197   0.1311
## Detection Prevalence   0.2845   0.1935   0.1743  0.16395   0.1837
## Balanced Accuracy      0.8605   0.8444   0.7599  0.80625   0.8880
```

```
# Accuracy plot
plot(cmdt$table, col = cmdt$byClass,
main = paste("Decision Tree Confusion Matrix: Accuracy =",
round(cmdt$overall['Accuracy'], 4)))
```

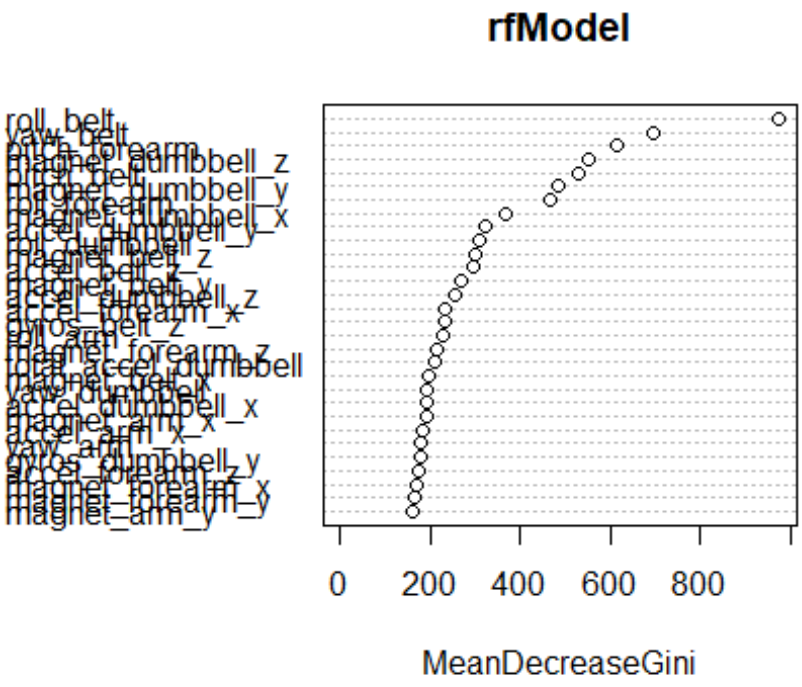


Training the model using Random Forest

```
rfModel <- randomForest(as.factor(classe)~., data=training)
# Summary of the model
rfModel
```

```
##
## Call:
##  randomForest(formula = as.factor(classe) ~ ., data = training)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.51%
## Confusion matrix:
##      A    B    C    D    E  class.error
## A 4182    3    0    0    0 0.0007168459
## B   15 2827    6    0    0 0.0073735955
## C    0   17 2547    3    0 0.0077911959
## D    0    0   23 2388    1 0.0099502488
## E    0    0    2    5 2699 0.0025868441
```

```
# Plot the variable importance
varImpPlot(rfModel)
```

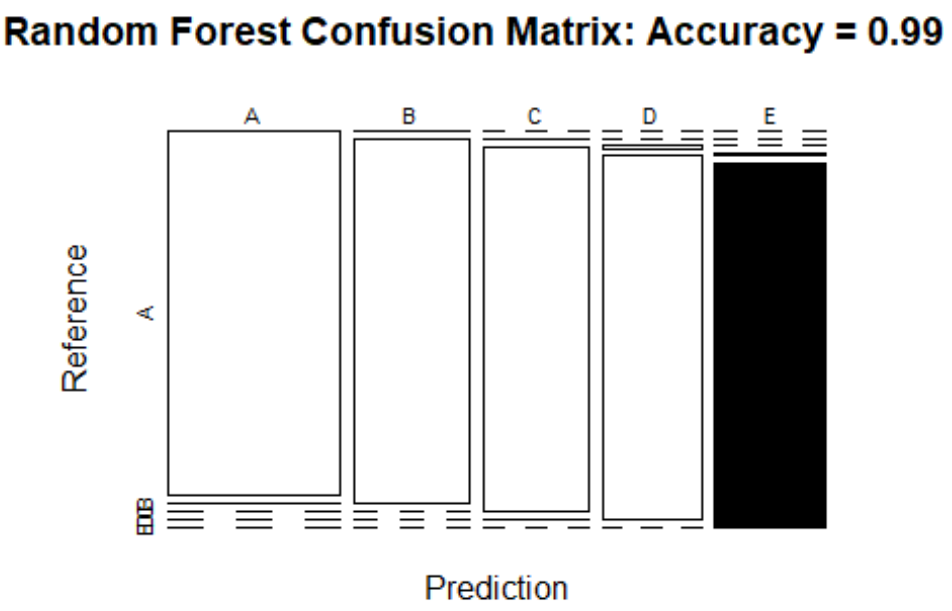



```
# Confusion matrix with testing
predTesting <- predict(rfModel, testing)
rfcfm  <- confusionMatrix(as.factor(testing$classe), predTesting)
rfcfm
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    A    B    C    D    E
##          A 1394    1    0    0    0
##          B    4  945    0    0    0
##          C    0    1  853    1    0
##          D    0    0    5  799    0
##          E    0    0    0    2  899
##
## Overall Statistics
##
##                Accuracy : 0.9971
##                  95% CI : (0.9952, 0.9984)
##     No Information Rate : 0.2851
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9964
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9971   0.9979   0.9942   0.9963   1.0000
## Specificity            0.9997   0.9990   0.9995   0.9988   0.9995
## Pos Pred Value         0.9993   0.9958   0.9977   0.9938   0.9978
## Neg Pred Value         0.9989   0.9995   0.9988   0.9993   1.0000
## Prevalence             0.2851   0.1931   0.1750   0.1635   0.1833
## Detection Rate         0.2843   0.1927   0.1739   0.1629   0.1833
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9984   0.9984   0.9968   0.9975   0.9998

plot(rfcfm$table, col = rfcfm$byClass, main = paste("Random Forest Confusion
Matrix: Accuracy =", round(rfcfm$overall['Accuracy'], 4)))
```



Predicting Results on the Test Data

Summary of the results: - Decision tree model - is the worst model running, has the low mean and the highest standard deviation. -Random Fores model - has the highest mean accuracy and lowest standard deviation Depending on how your model is to be used, the interpretation of the kappa statistic might vary. One common interpretation is shown as follows: • Poor agreement = Less than 0.20 • Fair agreement = 0.20 to 0.40 • Moderate agreement = 0.40 to 0.60 • Good agreement = 0.60 to 0.80 • Very good agreement = 0.80 to 1.00 This two models preforms as expected, the deviation from the cross validation accuracy is low and I do not see a reason to change resampling method or adding repetitons.

Summary of the results:

- Decision tree model is the worst model running, it has the low mean and the highest standard deviation.

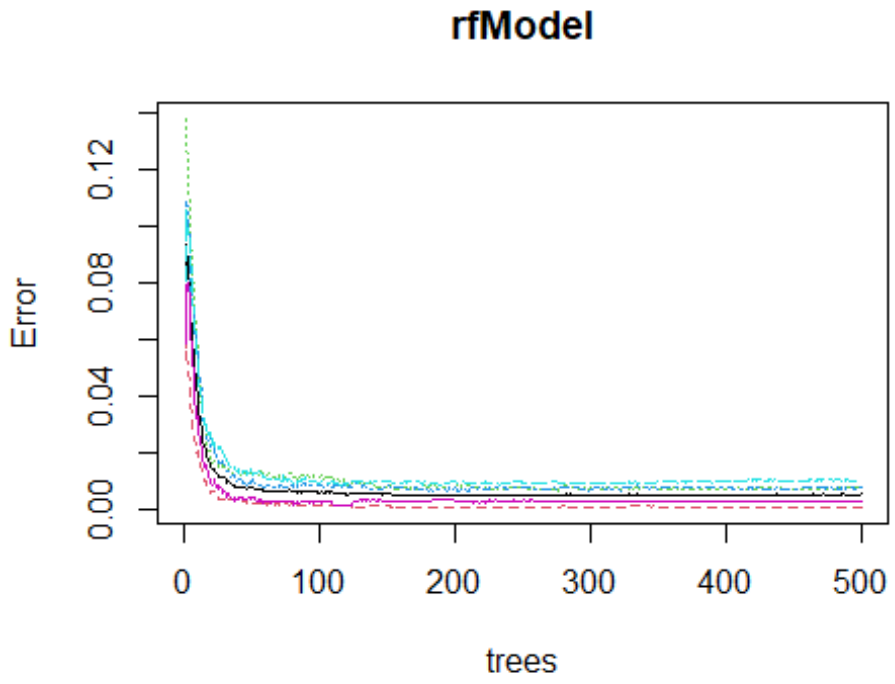
-Random Fores model it has the highest mean accuracy and lowest standard deviation.

Depending on how your model is to be used, the interpretation of the kappa statistic might vary One common interpretation is shown as follows:

• Poor agreement = Less than 0.20 • Fair agreement = 0.20 to 0.40 • Moderate agreement = 0.40 to 0.60 • Good agreement = 0.60 to 0.80 • Very good agreement = 0.80 to 1.00

This two models preforms as expected, the deviation from the cross validation accuracy is low.

```
#plot the model
plot(rfModel)
```



The predictive accuracy of the Random Forest model is excellent at 99.8 %. Accuracy has plateaued, and further tuning would only yield decimal gain.

- The best tuning parameters hads 150 trees (boosting iterations), interaction depth 3, shrinkage 0.1.

###Making prediction on the 20 data pointsusing random forest

Decision Tree Model: 74.58 %, Random Forest Model: 99.86 % The Random Forest model is selected and applied to make predictions on the 20 data points from the original testing dataset (testingst)

```
rfPredictions <- predict(rfModel, testingst,type= "class")
rfPredictions
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  E  A  E  E  D  E  E  E  A  E  E  E  D  A  E  E  E  E  E  E
## Levels: A B C D E
```