Prediction Assignment Writeup

Milton Candela

1/14/2021

Introduction

This project use data from *groupware*, in their investigation of Human Activity Recognition http://groupware.les.inf.puc-rio.br/har, which has the purpose of predicting how well is an exercise made, the factor variable **classe** tells how well the exercise (Unilateral Dumbbell Biceps Curl) is done: according to the specification (Class A), throwing the elbows to the front (Class B), lifting the dumbbell only halfway (Class C), lowering the dumbbell only halfway (Class D) and throwing the hips to the front (Class E).

the data was collected via an ambient sensing approach (by using Microsoft Kinect) and had a lot of features in it, but for the purpose of this investigation, only features corresponding to the acceleration of the person will be used.

Loading the libraries

Five packages will be needed in order to do the analysis:

- **dplyr**: Used to manipulate the training data set
- **randomForest**: Used to create Model 3 and Model 4
- **caret**: Used to model fitting
- **rattle**: Used to visualize **rpart** type of objects
- **rpart**: Used to create Model 1 and Model 2
- **lattice**: used to create the heatmap of the final model

It is important to set the seed to **1000**, so that **randomForest** provides consistent, reproducible results.

```r
library(dplyr)
library(randomForest)
library(caret)
library(rattle)
library(rpart)
library(lattice)
set.seed(1000)
```

Loading the data

Data sets used:

- Training dataset: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv

- Testing dataset: https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv

```
if(!file.exists('training.csv')){
       download.file(url = 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv',
                     destfile = 'training.csv', method = 'curl')
}
training <- read.csv('training.csv')

if(!file.exists('testing.csv')){
       download.file(url = 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv',
                     destfile = 'testing.csv', method = 'curl')
}
testing <- read.csv('testing.csv')
```

Both data sets have 160 columns, fortunately we will not be using everyone of this features:

```
print(data.frame(train_col = length(colnames(training)), test_col = length(colnames(testing))))
```

```
##   train_col test_col
## 1       160      160
```

Now let's see how which of the features correspond to the acceleration of the subject

```
print(grep('accel',colnames(training), value = TRUE))
```

```
##  [1] "total_accel_belt"     "var_total_accel_belt" "accel_belt_x"
##  [4] "accel_belt_y"         "accel_belt_z"         "total_accel_arm"
##  [7] "var_accel_arm"        "accel_arm_x"          "accel_arm_y"
## [10] "accel_arm_z"          "total_accel_dumbbell" "var_accel_dumbbell"
## [13] "accel_dumbbell_x"     "accel_dumbbell_y"     "accel_dumbbell_z"
## [16] "total_accel_forearm"  "var_accel_forearm"    "accel_forearm_x"
## [19] "accel_forearm_y"      "accel_forearm_z"
```

From these features, two main datasets will be created:

- **training_acc_xyz** which contains the x, y, z positions
- **training_acc_Tot** which contains the total acceleration, the variance and total variance will not be included because of the big number of NAs that the data contains.

```
for (feature in grep('var_accel',colnames(training), value = TRUE)) {
       print(paste('Percent of NA from the feature', feature, ' ', (sum(is.na(training[,feature])))/(len
}
```

```
## [1] "Percent of NA from the feature var_accel_arm   97.9308938946081"
## [1] "Percent of NA from the feature var_accel_dumbbell   97.9308938946081"
## [1] "Percent of NA from the feature var_accel_forearm   97.9308938946081"
```

Datasets creation

We will be creating both datasets in this section

First dataset: ***training_acc_xyz*** In order to create this data set, we need to subset he data and only obtain the acceleration features, and then subsetting again so that only features that contain the x, y and z parameters from each of the movements.

```
training_accel <- training[,c(1, 2, grep('accel',colnames(training)), 160)]
training_accel <- mutate(training_accel, classe = as.factor(classe))

training_acc_xyz <- training_accel[c(length(training_accel[1,]), grep('accel', colnames(training_accel)
training_acc_xyz <- training_acc_xyz[,-grep("total|var_", colnames(training_acc_xyz))]
str(training_acc_xyz)
```

```
## 'data.frame':    19622 obs. of  13 variables:
##  $ classe          : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ accel_belt_x    : int  -21 -22 -20 -22 -21 -21 -22 -22 -20 -21 ...
##  $ accel_belt_y    : int  4 4 5 3 2 4 3 4 2 4 ...
##  $ accel_belt_z    : int  22 22 23 21 24 21 21 21 24 22 ...
##  $ accel_arm_x     : int  -288 -290 -289 -289 -289 -289 -289 -289 -288 -288 ...
##  $ accel_arm_y     : int  109 110 110 111 111 111 111 111 109 110 ...
##  $ accel_arm_z     : int  -123 -125 -126 -123 -123 -122 -125 -124 -122 -124 ...
##  $ accel_dumbbell_x: int  -234 -233 -232 -232 -233 -234 -232 -234 -232 -235 ...
##  $ accel_dumbbell_y: int  47 47 46 48 48 48 47 46 47 48 ...
##  $ accel_dumbbell_z: int  -271 -269 -270 -269 -270 -269 -270 -272 -269 -270 ...
##  $ accel_forearm_x : int  192 192 196 189 189 193 195 193 193 190 ...
##  $ accel_forearm_y : int  203 203 204 206 206 203 205 205 204 205 ...
##  $ accel_forearm_z : int  -215 -216 -213 -214 -214 -215 -215 -213 -214 -215 ...
```

Second data set: ***training_acc_Tot*** The same procedure is done to create the total part of the acceleration, this data was ordered so that the class is the first column of the data.

```
training_acc_Tot <- training_accel[,c(grep('total_accel', colnames(training_accel)),
                                      length(colnames(training_accel)))]
training_acc_Tot <- training_acc_Tot[,length(colnames(training_acc_Tot)):1]
training_acc_Tot <- subset(training_acc_Tot, select = -5)
str(training_acc_Tot)
```

```
## 'data.frame':    19622 obs. of  5 variables:
##  $ classe              : Factor w/ 5 levels "A","B","C","D",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ total_accel_forearm : int  36 36 36 36 36 36 36 36 36 36 ...
##  $ total_accel_dumbbell: int  37 37 37 37 37 37 37 37 37 37 ...
##  $ total_accel_arm     : int  34 34 34 34 34 34 34 34 34 34 ...
##  $ total_accel_belt    : int  3 3 3 3 3 3 3 3 3 3 ...
```

Models creation

Predictive models will be created using their respective R functions, as an example: **randomForest()** and **rpart()**.

Model 1 The first model corresponds to a tree created by the **rpart** function, which displays a big decision tree but with enough zoom the decisions can be visualized, it is worth say that the **fancyRpartPlot** function corresponds to the **rattle** package, which displays a good-looking dendogram. This model use the total acceleration (as it can be seen inside the **rpart** funcion *data = training_acc_Tot*)

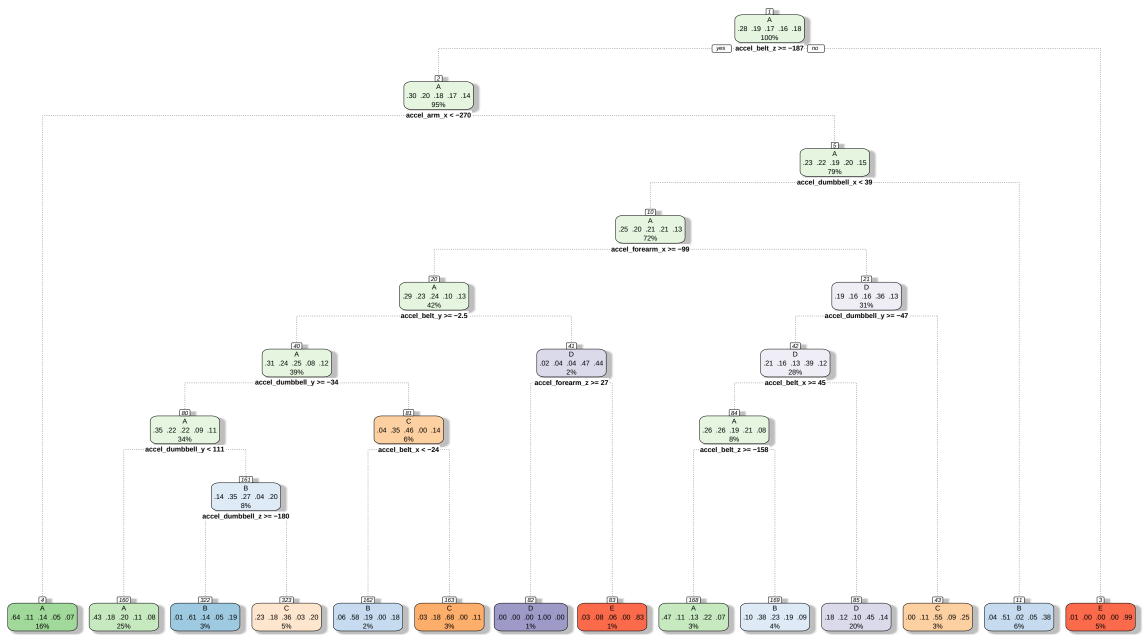
```r
model1 <- rpart(classe ~ ., data = training_acc_Tot, na.action = na.omit)
fancyRpartPlot(model1, main = 'Rpart visualization of Model 1', sub = '')
```


```r
mod1Pred <- predict(model1, testing, type = 'class')
```

Model 2 For the second model, the same procedure will be used but with the *training_acc_xyz* dataset, which takes into account the raw positions.

```r
model2 <- rpart(classe ~ ., data = training_acc_xyz, na.action = na.omit)
fancyRpartPlot(model2, main = 'Rpart visualization of Model 2', sub = '')
```

Rpart visualization of Model 2



```r
mod2Pred <- predict(model2, testing, type = 'class')
```

Model 3 And for the third model, **randomForest** will be used, using data from *training_acc_Tot* dataset. It can be observed that the OOB estimate of error rate is 30.34%, which is really high, and there is a lot of error in the confusion matrix.

```r
model3 <- randomForest(classe ~ ., data = training_acc_Tot)
print(model3)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training_acc_Tot)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 30.34%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 4521  266  386  340   67   0.1897849
## B  466 2251  496  259  325   0.4071636
## C  407  315 2266  318  116   0.3378141
## D  447  196  461 1960  152   0.3905473
## E  165  372  200  199 2671   0.2594954
```

```r
mod3Pred <- predict(model3, testing)
```

Model 4 While the fourth model will use the same method as Model 3 but using the data from *training_acc_xyz* dataset. This model presents an OOB estimate of error rate of 4.23%, which depicts a stronger model with more confidence in its predictions.

```r
model4 <- randomForest(classe ~ ., data = training_acc_xyz)
print(model4)
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = training_acc_xyz)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 4.23%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 5414   30   60   73    3  0.02974910
## B  107 3560   81   23   26  0.06241770
## C   49   69 3270   28    6  0.04441847
## D   53   12  114 3024   13  0.05970149
## E    5   31   22   25 3524  0.02301081
```

```r
mod4Pred <- predict(model4, testing)
```

Accuracy

We then test the accuracy from the predictive models and their predictions, we will first set the predictions on a data frame in order to visualice if their first elements appear to be similar.

```r
real <- as.factor(c('B', 'A', 'B', 'A', 'A',
                    'E', 'D', 'B', 'A', 'A',
                    'B', 'C', 'B', 'A', 'E',
                    'E', 'A', 'B', 'B', 'B'))
predDF <- data.frame(mod1Pred, mod2Pred, mod3Pred, mod4Pred, real)
head(predDF)
```

```
##   mod1Pred mod2Pred mod3Pred mod4Pred real
## 1        A        D        A        B    B
## 2        A        A        A        A    A
## 3        C        A        C        C    B
## 4        A        A        A        A    A
## 5        A        A        A        A    A
## 6        C        E        E        E    E
```

There is not much of a similarity between the models, there are some similar classes in which they all seem to agree (mainly on guessing the A class), which means that these predictive models can be used for binary classification (if a person is doing the exercise properly or no), we shall then test their accuracy in regard of the real class.

```
for (pred in predDF[,1:4]) {
       print(confusionMatrix(real, pred)$overall)
}
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.4500000      0.2307692      0.2305779      0.6847219      0.5500000
## AccuracyPValue  McnemarPValue
##      0.8692350            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.5500000      0.4078947      0.3152781      0.7694221      0.5000000
## AccuracyPValue  McnemarPValue
##      0.4119015            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.70000000     0.58041958     0.45721082     0.88106841     0.50000000
## AccuracyPValue  McnemarPValue
##     0.05765915            NaN
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   9.500000e-01   9.293286e-01   7.512672e-01   9.987349e-01   3.500000e-01
## AccuracyPValue  McnemarPValue
##   2.902513e-08            NaN
```

We can see that the first two models perform poorly, AUC < 0.50 is similar to flipping a coin so those models are not relevant. While the last two models got 95% accuracy, which proves that **randomForest** method functions well on the prediction of these behaviors. Although accuracy was 95% for model 3 and 4, model 4 predicted the $< 5\%$ error rate while model 3 predicted $> 30\%$. So combining predictors is not a great idea in this case, because only model 4 was accurate enough with the training set, as well as the test set, including the final validation classes, let's further analyze the final model (model 4).

```
modelFinal <- model4
```

Analysis

We will first take a look at the confusion matrix.

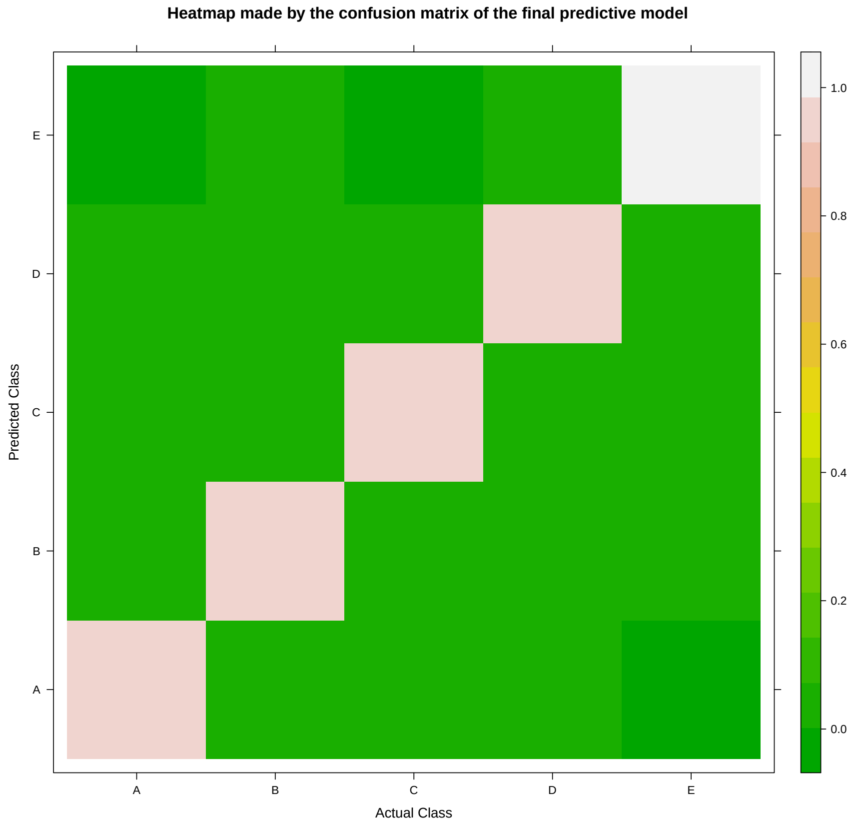
```
print(modelFinal$confusion)
```

```
##      A    B    C    D    E class.error
## A 5414   30   60   73    3  0.02974910
## B  107 3560   81   23   26  0.06241770
## C   49   69 3270   28    6  0.04441847
## D   53   12  114 3024   13  0.05970149
## E    5   31   22   25 3524  0.02301081
```

```
confm <- modelFinal$confusion[,-6]
```

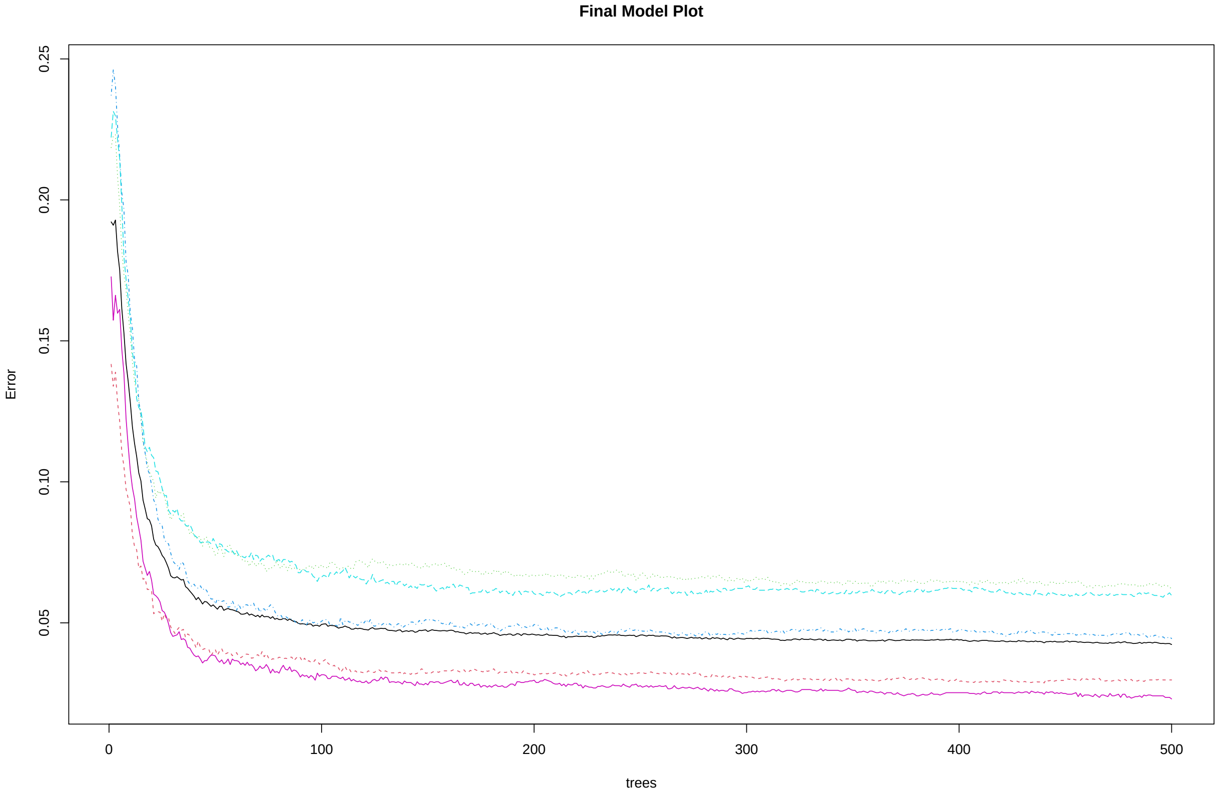
As it can be seen, values are most concentrated across the diagonal, which means that the accuracy is really high because most of the predicted values were the actual values. Now let's use the **levelplot** function from the **lattice** package in order to plot the heatmap of this matrix.

```r
sum_row <- apply(confm, FUN = sum, MARGIN = 2)
conf_centage <- sweep(confm, 2, sum_row, FUN = '/')
levelplot(conf_centage, col.regions = terrain.colors(100), xlab = 'Actual Class', ylab = 'Predicted Cla
          main = 'Heatmap made by the confusion matrix of the final predictive model')
```



Finally, we will take a look at the plot that the *randomForest* object generates, this will be a graph that represents the quantity of error with respect to the number of trees generates in each iteration.

```r
plot(modelFinal, main = 'Final Model Plot')
```



As it can be seen, the error drops below 10% when, approximately, 50 tress are generated, maybe that is the reason why the *rpart* method could not deliver a good accuracy rate as the predictive models with *randomForest* did.