Machine Learning Project

Plato Karageorgis

29/11/2021

A Quick Intro

As the report states "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" and that is what we are about to examine. We are going to create a model to predict the variable "classe" of our data. This variable pinpoints the way these people did their exercise, something that we can see thanks to the accelerometers that they wore.train_in <- read.csv('./pml-training.csv', header=T)

To begin the analysis, we will assign our data to local variables.

```
train_data <- read.csv('C:/Users/Platon/Desktop/Coursera/pml-training.csv', header=T)
test_data <- read.csv('C:/Users/Platon/Desktop/Coursera/pml-testing.csv', header=T)
```

Since the test data will be used later in order to test, as the name states, our prediction model, we will start with the train data. Following the course's instructions, we will use cross validation and separate the training data in partitions. The 75% will be used for training and the rest 25% will be responsible for the validation.

```
set.seed(33000)
inTrain <- createDataPartition(y= train_data$classe, p=0.7, list=FALSE)
training <- train_data[inTrain, ]
testing <- train_data[-inTrain, ]
```

Now, we have to start "playing" with our data. The main problem here is that we have a lot of NA values. This affects us mostly on our test set and not so much on our training set, although if we want to be precise, we should do it on both. There are multiples ways to go. We could erase all of them but then we might have a problem with the quantity of the remaining data. Also, we could erase some of them randomly and uniformly. I will choose the simpler solution and erase all of them since we have a lot of data.

```
nas <- sapply(names(test_data), function(x) all(is.na(test_data[,x])==TRUE))
nas_names <- names(nas)[nas==FALSE]
nas_names <- nas_names[-(1:7)]
nas_names <- nas_names[1:(length(nas_names)-1)]
fit <- trainControl(method='cv', number = 3)
```

Algorithms

I will apply 2 algorithms to check if my model is working correctly. I will use Random Forest and Boosting Trees. Firstly, for the random forest algorithm:

```
modelFitRF <- train(classe ~ ., data=training[, c('classe', nas_names)], trControl = fit, method='rf',n
```

And for the prediction:

```
predictFitRF <- predict(modelFitRF, testing, type = "raw")
```

And moving on the GBM algorithm:

```
modelFitGBM <- train(classe ~ ., data=training[, c('classe', nas_names)], trControl = fit, method='gbm')
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1268
##      2        1.5240             nan     0.1000    0.0824
##      3        1.4673             nan     0.1000    0.0675
##      4        1.4225             nan     0.1000    0.0557
##      5        1.3872             nan     0.1000    0.0404
##      6        1.3594             nan     0.1000    0.0411
##      7        1.3325             nan     0.1000    0.0376
##      8        1.3072             nan     0.1000    0.0316
##      9        1.2852             nan     0.1000    0.0369
##     10        1.2615             nan     0.1000    0.0323
##     20        1.1040             nan     0.1000    0.0169
##     40        0.9313             nan     0.1000    0.0084
##     60        0.8211             nan     0.1000    0.0052
##     80        0.7388             nan     0.1000    0.0043
##    100        0.6748             nan     0.1000    0.0037
##    120        0.6215             nan     0.1000    0.0037
##    140        0.5780             nan     0.1000    0.0015
##    150        0.5589             nan     0.1000    0.0018
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1861
##      2        1.4903             nan     0.1000    0.1281
##      3        1.4064             nan     0.1000    0.1002
##      4        1.3411             nan     0.1000    0.0828
##      5        1.2875             nan     0.1000    0.0663
##      6        1.2437             nan     0.1000    0.0722
##      7        1.1972             nan     0.1000    0.0605
##      8        1.1585             nan     0.1000    0.0486
##      9        1.1256             nan     0.1000    0.0513
##     10        1.0930             nan     0.1000    0.0387
##     20        0.8917             nan     0.1000    0.0238
##     40        0.6746             nan     0.1000    0.0131
##     60        0.5426             nan     0.1000    0.0062
##     80        0.4561             nan     0.1000    0.0040
##    100        0.3892             nan     0.1000    0.0023
##    120        0.3404             nan     0.1000    0.0032
##    140        0.2997             nan     0.1000    0.0030
##    150        0.2815             nan     0.1000    0.0010
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2245
```

```
##      2        1.4646             nan     0.1000    0.1610
##      3        1.3614             nan     0.1000    0.1184
##      4        1.2865             nan     0.1000    0.1044
##      5        1.2183             nan     0.1000    0.0841
##      6        1.1629             nan     0.1000    0.0705
##      7        1.1174             nan     0.1000    0.0700
##      8        1.0733             nan     0.1000    0.0595
##      9        1.0354             nan     0.1000    0.0622
##     10        0.9973             nan     0.1000    0.0592
##     20        0.7583             nan     0.1000    0.0244
##     40        0.5294             nan     0.1000    0.0119
##     60        0.4001             nan     0.1000    0.0077
##     80        0.3167             nan     0.1000    0.0054
##    100        0.2580             nan     0.1000    0.0032
##    120        0.2140             nan     0.1000    0.0023
##    140        0.1827             nan     0.1000    0.0014
##    150        0.1691             nan     0.1000    0.0010
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1275
##      2        1.5226             nan     0.1000    0.0868
##      3        1.4644             nan     0.1000    0.0648
##      4        1.4205             nan     0.1000    0.0525
##      5        1.3852             nan     0.1000    0.0406
##      6        1.3572             nan     0.1000    0.0446
##      7        1.3284             nan     0.1000    0.0394
##      8        1.3035             nan     0.1000    0.0334
##      9        1.2816             nan     0.1000    0.0335
##     10        1.2591             nan     0.1000    0.0323
##     20        1.1001             nan     0.1000    0.0171
##     40        0.9313             nan     0.1000    0.0083
##     60        0.8206             nan     0.1000    0.0050
##     80        0.7405             nan     0.1000    0.0033
##    100        0.6787             nan     0.1000    0.0036
##    120        0.6264             nan     0.1000    0.0036
##    140        0.5833             nan     0.1000    0.0015
##    150        0.5633             nan     0.1000    0.0026
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1891
##      2        1.4883             nan     0.1000    0.1263
##      3        1.4061             nan     0.1000    0.1020
##      4        1.3401             nan     0.1000    0.0872
##      5        1.2840             nan     0.1000    0.0790
##      6        1.2347             nan     0.1000    0.0629
##      7        1.1948             nan     0.1000    0.0577
##      8        1.1576             nan     0.1000    0.0491
##      9        1.1262             nan     0.1000    0.0416
##     10        1.0984             nan     0.1000    0.0434
##     20        0.8974             nan     0.1000    0.0201
##     40        0.6839             nan     0.1000    0.0097
##     60        0.5569             nan     0.1000    0.0054
##     80        0.4666             nan     0.1000    0.0056
##    100        0.3984             nan     0.1000    0.0045
```

```
##    120        0.3455             nan     0.1000    0.0033
##    140        0.3028             nan     0.1000    0.0021
##    150        0.2831             nan     0.1000    0.0015
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2358
##      2        1.4612             nan     0.1000    0.1638
##      3        1.3563             nan     0.1000    0.1291
##      4        1.2751             nan     0.1000    0.1042
##      5        1.2091             nan     0.1000    0.0792
##      6        1.1576             nan     0.1000    0.0698
##      7        1.1126             nan     0.1000    0.0792
##      8        1.0628             nan     0.1000    0.0577
##      9        1.0250             nan     0.1000    0.0539
##     10        0.9896             nan     0.1000    0.0512
##     20        0.7584             nan     0.1000    0.0248
##     40        0.5329             nan     0.1000    0.0190
##     60        0.4004             nan     0.1000    0.0112
##     80        0.3118             nan     0.1000    0.0034
##    100        0.2579             nan     0.1000    0.0022
##    120        0.2154             nan     0.1000    0.0011
##    140        0.1842             nan     0.1000    0.0010
##    150        0.1709             nan     0.1000    0.0022
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1285
##      2        1.5234             nan     0.1000    0.0906
##      3        1.4655             nan     0.1000    0.0654
##      4        1.4211             nan     0.1000    0.0523
##      5        1.3852             nan     0.1000    0.0436
##      6        1.3556             nan     0.1000    0.0434
##      7        1.3269             nan     0.1000    0.0379
##      8        1.3029             nan     0.1000    0.0350
##      9        1.2806             nan     0.1000    0.0352
##     10        1.2589             nan     0.1000    0.0323
##     20        1.1031             nan     0.1000    0.0189
##     40        0.9297             nan     0.1000    0.0099
##     60        0.8189             nan     0.1000    0.0061
##     80        0.7405             nan     0.1000    0.0029
##    100        0.6758             nan     0.1000    0.0026
##    120        0.6234             nan     0.1000    0.0029
##    140        0.5793             nan     0.1000    0.0019
##    150        0.5605             nan     0.1000    0.0023
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.1847
##      2        1.4892             nan     0.1000    0.1293
##      3        1.4059             nan     0.1000    0.1045
##      4        1.3388             nan     0.1000    0.0861
##      5        1.2823             nan     0.1000    0.0667
##      6        1.2379             nan     0.1000    0.0590
##      7        1.1993             nan     0.1000    0.0639
##      8        1.1596             nan     0.1000    0.0575
##      9        1.1230             nan     0.1000    0.0445
```

```
##     10        1.0937             nan     0.1000    0.0455
##     20        0.8870             nan     0.1000    0.0227
##     40        0.6712             nan     0.1000    0.0077
##     60        0.5475             nan     0.1000    0.0064
##     80        0.4628             nan     0.1000    0.0058
##    100        0.3951             nan     0.1000    0.0028
##    120        0.3412             nan     0.1000    0.0016
##    140        0.3024             nan     0.1000    0.0025
##    150        0.2838             nan     0.1000    0.0014
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2385
##      2        1.4580             nan     0.1000    0.1563
##      3        1.3587             nan     0.1000    0.1208
##      4        1.2822             nan     0.1000    0.0989
##      5        1.2175             nan     0.1000    0.0878
##      6        1.1617             nan     0.1000    0.0737
##      7        1.1139             nan     0.1000    0.0663
##      8        1.0706             nan     0.1000    0.0644
##      9        1.0289             nan     0.1000    0.0468
##     10        0.9989             nan     0.1000    0.0539
##     20        0.7576             nan     0.1000    0.0222
##     40        0.5220             nan     0.1000    0.0136
##     60        0.3952             nan     0.1000    0.0056
##     80        0.3151             nan     0.1000    0.0045
##    100        0.2586             nan     0.1000    0.0026
##    120        0.2148             nan     0.1000    0.0022
##    140        0.1820             nan     0.1000    0.0013
##    150        0.1685             nan     0.1000    0.0013
##
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094             nan     0.1000    0.2301
##      2        1.4629             nan     0.1000    0.1600
##      3        1.3615             nan     0.1000    0.1155
##      4        1.2879             nan     0.1000    0.1024
##      5        1.2238             nan     0.1000    0.0885
##      6        1.1680             nan     0.1000    0.0733
##      7        1.1218             nan     0.1000    0.0715
##      8        1.0776             nan     0.1000    0.0676
##      9        1.0346             nan     0.1000    0.0632
##     10        0.9949             nan     0.1000    0.0508
##     20        0.7650             nan     0.1000    0.0295
##     40        0.5370             nan     0.1000    0.0131
##     60        0.4086             nan     0.1000    0.0100
##     80        0.3281             nan     0.1000    0.0044
##    100        0.2683             nan     0.1000    0.0023
##    120        0.2237             nan     0.1000    0.0026
##    140        0.1894             nan     0.1000    0.0016
##    150        0.1757             nan     0.1000    0.0016
```

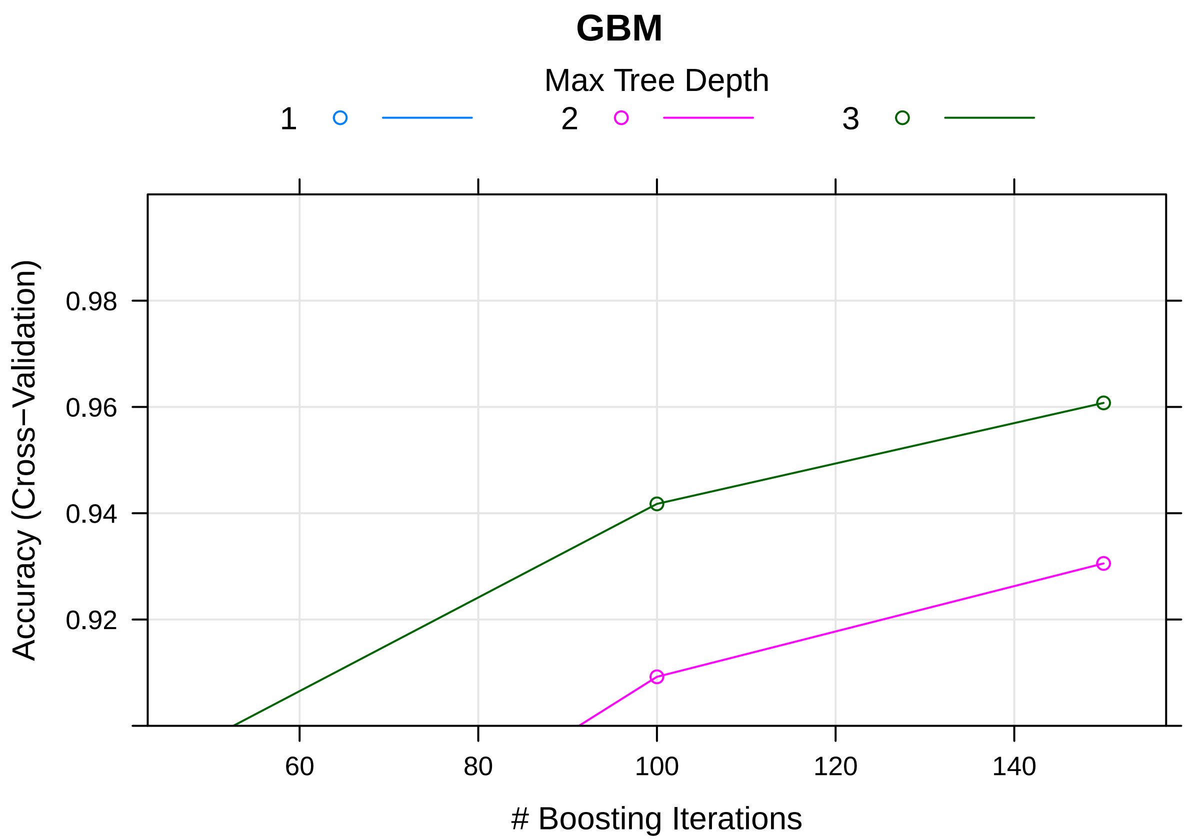
And for the prediction:

```r
predictFitGBM <- predict(modelFitGBM, testing, type = "raw")
```

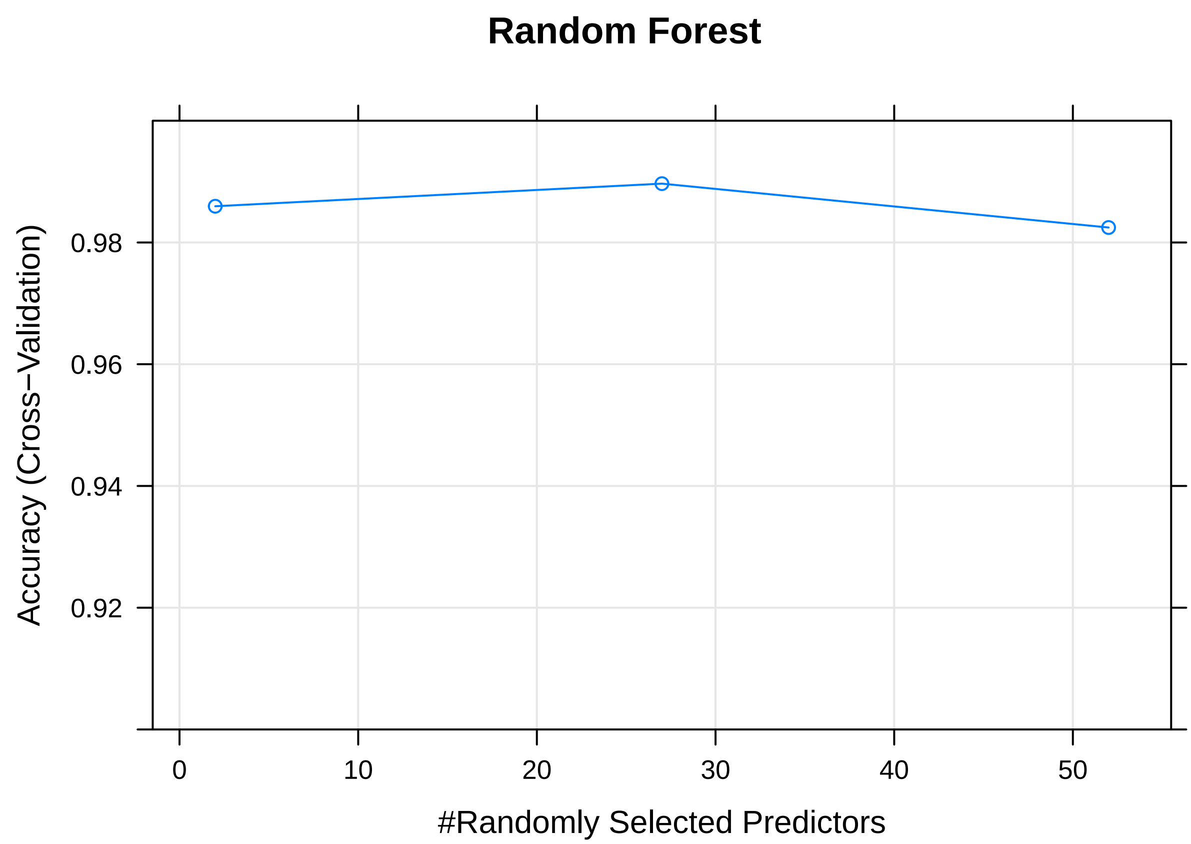
Plots

Moving on, I will do some very simple plots with the plot() function in order to visualize the accuracies.

```r
plot(modelFitGBM, ylim = c(0.9, 1), main= "GBM")
```




```r
plot(modelFitRF, ylim = c(0.9, 1),main= "Random Forest")
```



Results

Moreover, I am creating a confusion matrix exactly like the ones shown in the lectures in order to see the exact values of the accuracies.

```
confusionGBM <- confusionMatrix(predictFitGBM, as.factor(testing$classe))
confusionRF <- confusionMatrix(predictFitRF, as.factor(testing$classe))

accuracy <- data.frame(Model = c('RF', 'GBM'),Accuracy = rbind(confusionRF$overall[1], confusionGBM$ove
print(accuracy)
```

```
##   Model  Accuracy
## 1    RF 0.9935429
## 2   GBM 0.9639762
```

Final Predictions

Last but not least, the 20 predictions that the exercise requests. I will use Random Forest since it was better.

```
predictions <- predict(modelFitRF, newdata=test_data)
final <- data.frame(problem_id= test_data$problem_id, predicted=predictions)
print(final)
```

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

Conclusion

We can clearly see that the random forest algorithm is more than capable of predicting the right values from the plots but even more clearly on our matrix. The GBM model takes is also running quite impressively. You can see the results on the .html file.