# C8S4_Style_Exercise

jlg

28/06/2022

# Project : Predicting style activity

## Goal of this project

Now collecting large amount of data about personal activity is relatively inexpensively with devices such as **Jawbone Up**, **Nike FuelBand**, and **Fitbit**. These type of devices are part of the quantified self movement, who incitate people to take measurements about themselves regularly to improve their health, to find patterns in their behavior. In this project, the **goal is to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants** who were asked to perform barbell lifts correctly and incorrectly in 5 different ways, **to identify how well they did it** (outcome classe values A to E).

## Data and execution preparation

### Read, clean and prepare the data for modeling

These 2 datasets are provided with a **lot of quite empty columns** (only ~400 samples have quite complete 160 variables. The rest of the 19623 samples provided in the file pml-training.csv are incomplete **and about 100 columns have each row set to NA**. So **no need to keep this columns**, they are deleted from the dataset. The **training data set will be separated in a training data set and a validation data set**.

### Preparing parallel execution

The **rf model as gbm model** are particularly heavy **processor capacities consumators**. Then we can use **parallel execution to divide the total jobs in disjoints chunks and run theme concurrently**. Result are : lower time execution due to higher processor load. In this case without parallelization for one rf modelisation it took 18mn elapsed time with a 25% mean processor load, 60% memory versus 99% mean processor load, 93% memory and 13mn elapsed time with parallelization

# Constructing and testing models

## Why choose random forest

In a decision tree model, the data is splitted in small chunks and these samples are chosen according to a **purity measure**. That is, at each node, we want **information gain to be maximized**. For a regression problem, we consider residual sum of square (RSS) and for a classification problem, we consider the Gini index or entropy.

## Features selection

Plotting **classe** versus **user_name** and sequential **X** shows that they generally all start exercise in class A and change toward upper letter class and for some of them a little intensity decrease at the end ### Rank Features By Importance Running a first rf model on the training data set then the function **VarImp** on the descriptor of the model gives us this ranking table for the importance of the features (ten first) :

X

100.0000

roll_belt

7.2014

pitch_forearm

2.0603

raw_timestamp_part_1

1.9917

accel_belt_z

1.6339

roll_dumbbell

1.0777

num_window

0.8352

accel_forearm_x

0.7579

magnet_dumbbell_y

0.7257

magnet_belt_y

0.6094

### Correlation matrix

We also calculate the correlation matrix between the features given in this table, and it turns that features **raw_timestamp_part_1/num_window, roll_belt/accell_belt_z, yaw_belt/magnet_belt_y** are highly correlated.

### Parameters when Running Random Forest models

**Random Forest** aggregate best individual predictor. To make a prediction, we just obtain the predictions of all individuals trees, then predict the class that gets the most votes. So we will run the model rf with the **outcome classe** and the following predictors **roll_belt, pitch_forearm, roll_dumbbell, num_window, accel_forearm_x, magnet_dumbbell_y, cvtd_timestamp, total_accel_belt, yaw_belt+roll_forearm, pitch_belt, magnet_dumbbell_z, accel_dumbbell_y**.

**Arguments :**

- **ntree**: number of trees in the forest
- **mtry**: Number of candidates draw to feed the algorithm. By default, it is the square of the number of columns.

- **maxnodes**: Set the maximum amount of terminal nodes in the forest

- **importance=TRUE**: Whether independent variables importance in the random forest be assessed
  **K-fold cross validation** is controlled by the **trainControl()** function whose main arguments are : **method = cv**: The method used to resample the dataset, **number = n**: Number of folders to create, search = **grid**: Use the search grid method. For randomized method, use **grid**.
  By default, the train function chooses the model with the largest performance value. For the first model **modFitrf1** : **ntree = 350**, default value for **mtry**.

```
## utilisateur     système      écoulé
##       24.73        0.27       94.30
```

**Tuning the model and choose the Final Model**

We try now tuning two parameters, namely the **mtry and the ntree parameters** who are the most likely to have the biggest effect on our final accuracy. **mtry** parameter has effect on the final accuracy but it must be found empirically for a dataset. The **ntree** parameter is different in that it can be as large as you like, and continues to increases the accuracy up to some point. It is less difficult or critical to tune and could be limited more by compute time available more than anything. We focus on ntree parameter to improve our model accuracy, set **ntree to 200** and set **mtry to sqrt(ncol(training)-1)** for the **modFitrf2** model

```
## utilisateur     système      écoulé
##       15.03        0.31       56.44
```

# Conclusion

Adjusting Random Forest parameters permitted to lower time execution (2 execution summaries above) and improve the quality of the prediction of the models as shown below. Here are the values for the 2 models **modFitrf1, modFitrf2** :

```
Accuracy_model1
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9945856      0.9931502      0.9920766      0.9964602      0.2846731
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

```
Accuracy_model2
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      0.9995835      0.9994732      0.9984963      0.9999496      0.2846731
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

We can now produce the **prediction for the 20 samples from the testing** data set : B, A, B, A, A, E, D, B, A, A, B, C, B, A, E, E, A, B, B, B

```
predclass3
```

```
##  [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
```

# Appendix

## Confusion Matrix for the 2 tested models on validating data set

**modFitrf1 with ntree = 350 and default mtry**

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1367    7    0    0    0
##          B    0  915    0    0    0
##          C    0    7  838    7    0
##          D    0    0    0  779    5
##          E    0    0    0    0  877
##
## Overall Statistics
##
##                Accuracy : 0.9946
##                  95% CI : (0.9921, 0.9965)
##     No Information Rate : 0.2847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9932
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9849   1.0000   0.9911   0.9943
## Specificity            0.9980   1.0000   0.9965   0.9988   1.0000
## Pos Pred Value         0.9949   1.0000   0.9836   0.9936   1.0000
## Neg Pred Value         1.0000   0.9964   1.0000   0.9983   0.9987
## Prevalence             0.2847   0.1935   0.1745   0.1637   0.1837
## Detection Rate         0.2847   0.1905   0.1745   0.1622   0.1826
## Detection Prevalence   0.2861   0.1905   0.1774   0.1633   0.1826
## Balanced Accuracy      0.9990   0.9925   0.9982   0.9949   0.9972
```

**modFitrf2 with ntree = 200 and mtry = sqrt(ncol(training)-1)**

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1367    0    0    0    0
##          B    0  929    0    0    0
##          C    0    0  838    1    0
##          D    0    0    0  785    1
##          E    0    0    0    0  881
##
## Overall Statistics
```

```
##
##                Accuracy : 0.9996
##                  95% CI : (0.9985, 0.9999)
##     No Information Rate : 0.2847
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9995
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   1.0000   1.0000   0.9987   0.9989
## Specificity            1.0000   1.0000   0.9997   0.9998   1.0000
## Pos Pred Value         1.0000   1.0000   0.9988   0.9987   1.0000
## Neg Pred Value         1.0000   1.0000   1.0000   0.9998   0.9997
## Prevalence             0.2847   0.1935   0.1745   0.1637   0.1837
## Detection Rate         0.2847   0.1935   0.1745   0.1635   0.1835
## Detection Prevalence   0.2847   0.1935   0.1747   0.1637   0.1835
## Balanced Accuracy      1.0000   1.0000   0.9999   0.9992   0.9994
```