# 13 october 2020

## TP - Bike Usage Forecasting with Random Forest Algorithm

In this practical work, you will use historical public bike station data to predict the quantity of bike(s) available at a station for 3 different horizons (15, 30, 60 minutes) using a Random Forest algorithm. The provided data has already been largely pre-processed for you so will only need to perform minimal changes to the received data. You will have to load and prepare the data and then train and evaluate the machine-learning algorithm.

You have to hand back this practical work until the 21.10.2019, 23h55. You must return, in a formatted document, your answers to all the questions (with plots and figures when necessary) and your code (You can provide a compressed document (zip) containing your report and the completed <.py> file).

## Let's start the practical work

Load the provided project in PyCharm (or your favourite IDE) and look at the provided code skeleton in the <bike_prediction.py> file. Most of the elements that you need to code are tagged directly in the file itself (*#TODO CPx*).

SciKit documentation about RandomForestRegressor:
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

### Coding Part 1

You must now implement the Random Forest Regressor algorithm to predict the number of bikes at a station for different time horizons. Please follow the steps below (which correspond to the elements tagged *#TODO CP1* in the code). Note that the steps below are here to help you, but you can always take a different approach as long as it does the job.

Suggested steps:

1. Load the data for a specific station
    a. Tips: Inspect once the dataframe using the debug mode of PyCharm and identify the columns that the algorithm must predict!
2. Split the dataframe into train_set and test_set (Warning: do not shuffle data, keep its temporal ordering). Use 75% for train set and 25% for test set.
3. Split train_set into X_train_set (input data) and y_train_set (output/prediction data)
4. Split test_set into X_test_set (input data) and y_test_set (output/ground truth data)
5. Initialize and train your RandomForestRegressor algorithm
    a. You can use the default parameters of the algorithm
6. Perform the predictions on the X_test_set
7. Convert the received predictions back into a pandas DataFrame
    a. Tips: Names the columns the same as the y_test_set columns
8. Look at the predicted values, do they make sense for a quantity of bikes?
    a. Convert them to integers (using the round(0).astype(int) function)
9. Compute the mean absolute error (mae) for each horizon

   a. Tips: you may need to reset_index() of your dataframes before doing any column-wise subtraction
10. Plot the mean absolute error using pandas dataframe <plot.bar> function
    a. Tips 1: you need to use matplotlib <plt.show()> method for the graph to be displayed
    b. Tips 2: you may want to use matplotlib <tight_layout()> function for better plot visibility
11. Plot the Random Forest Gini Coefficient

**Questions 1**

1. What are the most important Gini coefficients for Station <0>?
   - Please provide the generated plot!
2. How many decision trees are used for the Random Forest algorithm?
3. Try to explain the role of the 3 most important features in the prediction mechanism
4. Can you explain why is it important **not** to shuffle the data during the splitting of the data into train and test sets?

**Coding Part 2**

You must now add additional features, namely the <hour> and <dayofweek> information. Then you must evaluate the system with and without using the newly extracted time information. I suggest you to run the evaluate function with each combination of parameters and to store the results into a pandas DataFrame.

| **Station_id** | **MAE without time information** | **MAE with time information** |
|---|---|---|
| 0 | ? | ? |
| 1 | ? | ? |
| 2 | ? | ? |
| Average | ? | ? |

Suggested steps:

1. Implement the hour and day of week from the index of the DataFrame
   a. Tips: use apply() function
2. Code the loops to run multiple times the evaluate method and store the received values directly into a pandas DataFrame.
   a. There is no #CP2 Tag in the code for this part.
   b. Tips: you can do that directly in the <main> or in a dedicated method
   c. Tips: name you index/columns when you initialize your pandas DataFrame
3. [Optional] Plot the final results in a single bar plot

**Questions 2**

1. Provide the table above filled with your results
2. Did the use of the two new features improve your results? (quantify)
   a. Was it the same for all three horizons and all stations?
3. Can you find a way of showing the impact of the new features on the algorithm computations? (not using the table above …)
4. One station has very different results from the two others, could you give a couple of hypotheses for those results? (mostly use your brain, not the data)

**Questions 3**

1. You have a univariate dataset with a value acquired every minute (see table). You now want to predict the future value based on the current value (let's assume it is feasible). What pre-processing operation do you need to perform on the dataframe, such that it can then be fed to train a machine-learning algorithm?

| **t** | **Value(t)** | **?** |
|---|---|---|
| **0** | 1 | |
| **1** | 2 | |
| **2** | 3 | |
| **…** | … | |
| **98** | 6 | |
| **99** | 7 | |

2. Provide a small example of code that would do this operation
    a. Tips: there is a method for pandas DataFrame that does exactly that
    b. Tips: you can create a test dataframe with the following method:
        i. df = pd.DataFrame({'value(t)':range(100)})
    c. Tips: the solution can be done in a few lines of code!

**[OPTIONAL] Going further …**

These below are only a few elements that present you some direction to think a bit further the practical work. The first point is usually mandatory for any ML project if you want to obtain valid results; the second point is more to give you some "food for thought" …

There will not be any correction for those!

1. Finding the best hyper-parameters

   You implemented a very basic RandomForestRegressor algorithm to perform the predictions. The solution you implemented uses the default hyper-parameters of the algorithm. You could look for hyper-parameters that yield better results, manually (time-consuming and tedious) or using strategies such as "GridSearch". You'll see that in a future practical work!

2. Food for thought
   - How do you define the best score and how do you model your whole system?
   - You must also think about how you measure the best solution: each station individually or for all stations?
   - Do you use the same hyper-parameters for all stations or different ones for each station?
   - What are the implication of those choices and their impact on the results?
   - The Gini coefficients showed that many features were not (or almost not) used by the algorithm. Should we remove them?
   - The provided data covers 3 months of usage, is it enough to learn?
   - The data was not normalized, would it be useful in the context of RF?
   - The data was not one-hot encoded, would it be useful in the context of RF?
   - An interesting resource for time-series manipulation using pandas: https://towardsdatascience.com/basic-time-series-manipulation-with-pandas-4432afee64ea