FINAL PROJECT REPORT : KDD-CUP-of-Fresh-Air-Project

Ella Kummer, Jérémie Guy

June 24, 2020

1 Introduction

The aim of this project is to to predict the pollution level hourly in two cities, precisely 35 stations in Beijing and 13 stations in London.
The pollution level prediction for London is made by predicting the concentration of PM2.5 and PM10 and the pollution level for Beijing is made by predicting the concentration of PM2.5, PM10 and O3.

This project wants to help the population, as over the past years air pollution has become gradually a drastic issue in many larges cities (e.g. Beijing).
For the reason that air particles (Particulate Matters PM) are one of the deadliest forms of air pollutants, accurately monitoring and predicting the concentration of PM2.5 and other air particles is becoming more and more necessary. With precise predictions of air pollution levels, the government would able to warn the population and take protective decisions against the noxious consequences of air pollution.

2 Sorting the data

2.1 London data

2.1.1 Original data files

- London_AirQuality_Stations.csv
  This contains the ids and the coordinates (longitude and latitude) of the Air quality stations.

- London_grid_weather_station.csv
  This contains the coordinates of the grid weather points and the grid's id corresponding.

- London_historical_aqi_forecast_stations_20180331.csv
  This contains the following parameters: PM2.5 (ug-m3), PM10 (ug-m3), NO2 (ug-m3) for every hours from the 2017-01-01 to the 2018-03-31 and for every station we want

to predict the level of pollution for. These are the features related to the air quality and others.

- London_historical_aqi_forecast_stations_20180331_ordered.csv
  This is the same data set as the previous one, except that the order of the column has be switched to match the other data sets.

- London_historical_aqi_other_stations_20180331.csv
  This contains the same as the previous one, except that the data are related to stations we don't want to predict the level of pollution for.

- London_historical_meo_grid.csv
  This contains the meteorological data : temperature, pressure, humidity, wind_direction, wind_speed/kph, for every hours from the 2017-01-01 to the 2018-03-27.

All these data have been cleaned.
The utc_time and the name of the parameters have been standardize for all files. The missing data inside lines have been filled using linear interpolation.
If some stations are missing some hours or days (whole lines), this is negligible as soon as they're not in the test data.

2.1.2 Merged data files ("merge" folder)

Each file corresponds to one station we need to predict the pollution level for. It contains the following parameters: station_id, utc_time, PM2.5 (ug-m3), PM10 (ug-m3), NO2 (ug-m3), stationName, longitude, latitude, temperature, pressure, humidity, wind_direction, wind_speed/kph
These parameters come from different original data set which have been combined in 5 steps :

1. The station id is selected from London_AirQuality_Stations.csv .
2. The same id is used in London_historical_aqi_stations_20180331.csv in order to select the rows corresponding to this station. This allow us to get the PM2.5 (ug-m3), PM10 (ug-m3) and NO2 (ug-m3) parameters.
3. Using the latitude and longitude of the station from London_AirQuality_Stations.csv, the corresponding id (london_grid_XYZ) in London_grid_weather_station.csv is found.
4. With this new id, we select the row corresponding to this station from the London_historical_meo_grid.csv file. This allow us to get the temperature, pressure, humidity, wind_direction and wind_speed/kph parameters.
5. We combine the selected rows, from the two previous files, according to their utc time.

2.1.3 Features selection

We used the correlation matrix in order to proceed to features selection. All the correlation matrix are accessible inside the correlation_matrix/london folder. Figure 1 shows the correlation matrix for the station GR9 as example.

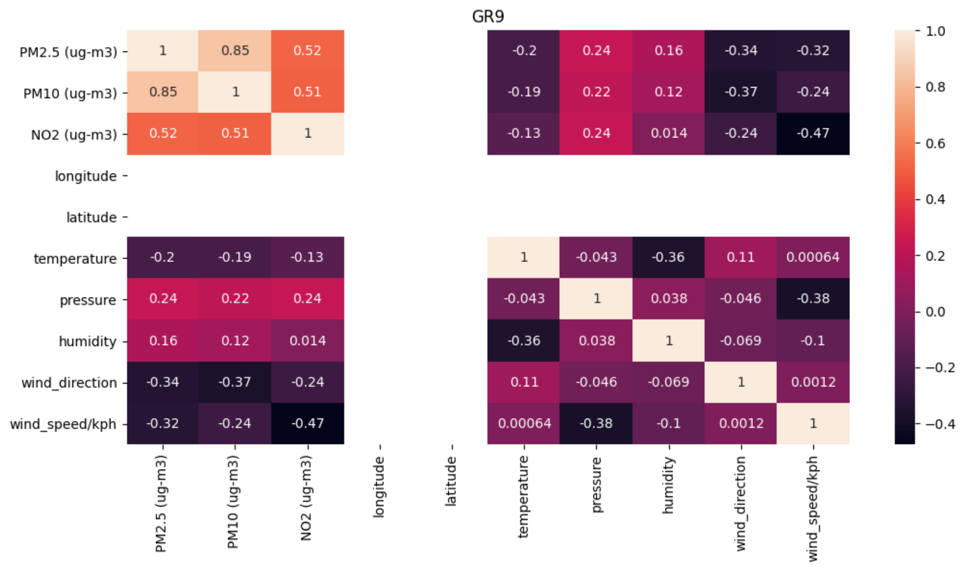

Figure 1: correlation matrix for GR9

Having a look at it, it is clear that there is no strong link between the meteorological data and the one we want to predict (air quality). This is why at the end, meteorological data were not used.
Both methods have been tested (with and without) and better results were found without.

2.2 Beijing data

2.2.1 Data files

- beijing_17_18_aq.csv
  This contains the features related to the air quality: PM2.5 PM2.5, PM10, NO2, CO, O3 and SO2, for every hours from the 2017-01-01 14:00:00 to the 2018-01-31 15:00:00 and for every station we want to predict the level of pollution for.

- beijing_17_18_meo.csv
  This contains the meteorological data : temperature, pressure, humidity, wind_direction and wind_speed, for every hours from the 2017-01-30 16:00:00 to the 2018-01-31 15:00:00.

- beijing_17_18_meo_cut.csv
  This contains the same data as beijing_17_18_meo.csv except that the longitude and

latitude have been removed for practicality. Indeed, once we used the longitude and latitude to link the station id to the grid id, these features are not useful anymore for the prediction.

- beijing_201802_201803_aq.csv
  This contains the following part (next dates) from beijing_17_18_aq.csv. The last data is 2018-03-31 15:00:00.

- beijing_201802_201803_me.csv
  This contains the following part (next dates) from beijing_17_18_meo.csv. The last data is 2018-04-01 00:00:00.

- beijing_201802_201803_me_cut.csv
  This contains the same data as beijing_201802_201803_me.csv except that the longitude and latitude have been removed for practicality.

- Beijing_AirQuality_Stations_en.xml
  Mainly, this contains the coordinates of the Air quality stations and their names (considered as id too).

- Beijing_AirQuality_Stations_en_csv_formated.csv
  This contains the same as the previous one, but it has been changed to csv format for usability.

- Beijing_grid_weather_station.csv
  This contains the coordinates of the grid weather points and their corresponding grid's id.

- Beijing_historical_meo_grid.csv
  This contains the meteorological data : temperature, pressure, humidity, wind_direction, wind_speed, for every hour from 2017-01-00 00:00:00 to 2018-03-27 05:00:00.

Before separating the differents stations in differents files, beijing_17_18_aq.csv and beijing_201802_201803_aq.csv are concatenate as well as beijing_17_18_meo_cut.csv and beijing_201802_201803_me_cut.csv . These files are not saved but let's call them beijing_aq and beijing_meo.
All these data have been cleaned.
The utc_time and the name of the parameters have been standardize for all files. Some station's name have been corrected (typos). The missing data have been filled using linear interpolation and the duplicated lines have been removed.

2.2.2 Merged data files (”mergeBeijing” folder)

Each file corresponds to one station we need to predict the pollution level for. It contains the following parameters: stationId, utc_time, PM2.5, PM10, NO2, CO, O3, SO2, stationName, temperature, pressure, humidity, wind_direction and wind_speed
These parameters come from different original data set which have been combined in 6 steps :

1. The station id (name of the station) is selected from Beijing_AirQuality_Stations_en_csv_formated.csv.

2. The same id (name of the station) is used in beijing_aq in order to select the rows corresponding to this station. This allow us to get the PM2.5, PM10, NO2, CO, O3 and SO2 parameters.

3. If this id is present in the beijing_meo, then go to point 4, otherwise jump to point 5.

4. The same id is used to select the rows corresponding to this station. This allow us to get the temperature, pressure, humidity, wind_direction and wind_speed parameters. Now jump to point 6.

5. Using the latitude and longitude of the station from Beijing_AirQuality_Stations_en_csv_formated.csv, the corresponding id (beijing_grid_XYZ) in Beijing_grid_weather_station.csv is found. With this new id, we select the row corresponding to this station from the eijing_historical_meo_grid.csv file. This allow us to get the temperature, pressure, humidity, wind_direction and wind_speed parameters.

6. We combine the selected rows, from the two previous files, according to their utc time.

2.2.3 Features selection

We used the correlation matrix in order to proceed to features selection. (plot corr. mat) All the correlation matrix are accessible inside the correlation_matrix/beijing folder. Figure 2 and 3 shows the correlation matrix for the stations zhiwuyuan and fangshan as example.

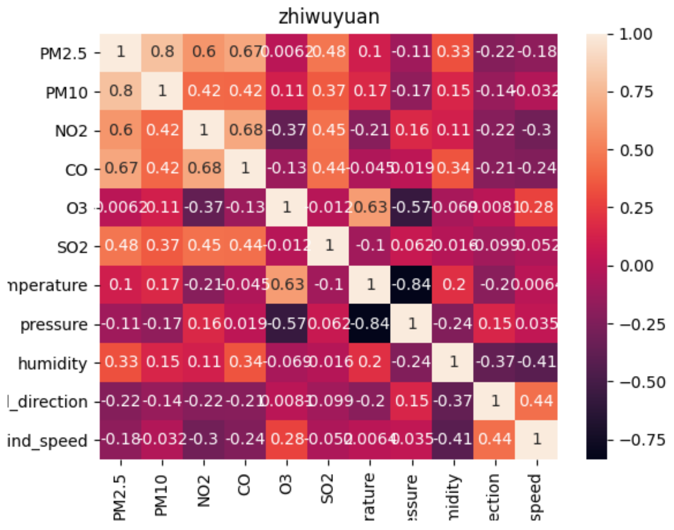

Figure 2: correlation matrix for zhiwuyuan

Having a look at it, it is clear that some features are strongly correlated while some other

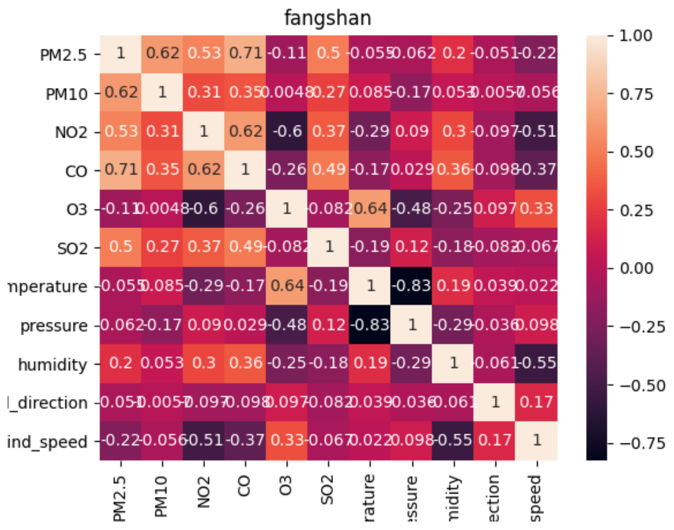
Figure 3: correlation matrix for fangshan

are way less.
To predict the O3 level, all features were kept except the PM2.5 and SO2 levels. While for the prediction of the PM2.5 and PM10 levels, all features were kept except the temperature and the pressure.

2.3 Sets

2.3.1 Training and validation sets

The data until 20 of March 2018 were available. In order to have a correct running time, we took 300 lines of data before the 21st of March 2018 for each stations (so the last one is 2018-03-20 23:00:00).

2.3.2 Test set

21th and 22th of March are used for testing. This means that we ran our final model on these data to predict their level of pollution (PM2.5, PM10 and also O3 for Beijing) hourly. For every hour prediction, we used the data from the hour before. In our code, this can be easily changed (the variables exist) to use the previous three hours to predict, or to predict with the data from two days before.

3 Methods used

3.1 Linear Regression

3.1.1 Algorithm

The main idea is to try and model the relationship between two variables by fitting a linear equation in the observed data. We're going to distinguish the explanatory variables (x - in this case PM2.5, PM10 and NO3) on which the variables to predict (y - PM2.5 and PM10) depend. The explanatory variables are mutually independent. We will use here the idea of the line of best fit : y = ax + b. To get the a and b parameters, we compute :

$$a = \frac{n * (\sum xy) - (\sum x)(\sum y)}{n * (\sum x^2) - (\sum x)^2}$$

$$b = \frac{(\sum y)(\sum x^2) - (\sum x)(\sum xy)}{n * (\sum x^2) - (\sum x)^2}$$

We can further tune the parameters a and b by minimizing the Mean Squared Error (MSE) function:

$$min(\sum_{i=1}^{n}(pred_i - y_i)^2)$$

Using the a and b tuned parameters, we can plot a straight line to predict for each value the resulted output.

3.1.2 Best Hyperparameters

No Hyper parameters.

3.1.3 Results MSE with bests hyperparameters

London :
PM2.5 : 9
PM10 : 31

Beijing :
PM2.5 : 245
PM10 : 231
O3 : 231

3.2 Gradient Tree Boosting

3.2.1 Algorithm

The Gradient Tree Boosting method is based on decisions trees[1]. The main idea is to start from a simple weak model and slowly adjusting it and combining the results to get a strong

[1]A decision tree is a tree created on the distributed data. We ask if a given feature is greater than a certain value in majority and divide our node accordingly

prediction : that's the boosting part. First, we get some data and create a decision tree (let's say $f_1(x) \approx y$ (our true values)) based on the data set. The model is weak and cannot predict much. We are then going to adjust the model using the residuals (aka wrong samples). As we are here in a regression task, we are going to use the MSE again to detect the residuals (the log loss would be more appropriate for a classification task). Using the MSE we compute the residuals : $y - f_1(x)$. The obtained residuals are now use to create a second tree : $f_2(x) \approx y - f_1(x)$. From this tree we compute the new residuals : $y - f_1(x) - f_2(x)$ which are going to be used to create a third tree : $f_3(x) \approx y - f_1(x) - f_2(x)$ etc.

Each of the new trees modifies the overall model. The magnitude of the changes are limited by the learning_rate parameter α. Each new computed tree will now have a residual value weighted by α : $f_2(x)$ becomes $f_2(x) \approx y - \alpha f_1(x)$, $f_3(x)$ becomes $f_3(x) \approx y - \alpha f_1(x) - \alpha f_2(x)$. The smaller the learning rate, the lesser changes will occur between each tree and the slower the model learns (the algorithm being sequential). Learning slower may be great to have a greatly fitting model but it comes at a cost. To limit it, we introduce a second hyper parameter : n_estimator. This will limit the number of tree the model can use before arriving to a conclusion model. If we use a smaller learning rate, we can compensate it by using a big number of trees. The balance is subtle as too much tree will create a high risk over-fitting.

3.2.2 Best Hyperparameters

London :

PM2.5

- n_estimator : 800
  number of boosting stages
- learning_rates : 0.01
  shrinks contribution of each tree
- max_depths : 3
  limits the number of nodes in the tree

PM10

- n_estimator : 1750
- learning_rates : 0.01
- max_depths : 1

Beijing :

PM2.5

- n_estimator : 400
- learning_rates : 0.1

- max_depths : 3

PM10

- n_estimator : 250
- learning_rates : 0.07
- max_depths : 3

O3

- n_estimator : 400
- learning_rates : 0.1
- max_depths : 3

3.2.3 Results MSE with bests hyperparameters

London :
PM2.5 : 9
PM10 : 27

Beijing :
PM2.5 : 204
PM10 : 202
O3 : 141

3.3 Random Forest

3.3.1 Algorithm

The main principle behind the Random Forest is the decision tree. The model is going to create a set of decision trees and creating a mean (for regression tasks) to decide of a final model. First, the model chooses the features for each decision tree. this process can be achieved either by Bootstrapping or by Feature Randomness. Bootstrapping means randomly selecting samples from our data to create new sub-samples. Each of the sub-samples will be used to create a tree. Feature Randomness selects random features to be used on the creation of each tree. Both of the methods allows the random forest to have uncorrelated trees. If the trees are too correlated, the ending mean tree will be very similar as a single tree (too much over-fitting).

The created trees in a forest are all created in parallel (as they are uncorrelated). So the cost in time is not so great. But creating too much trees won't necessarily yield better results and instead only eat up space (and add computation time). To counter it we will be introducing a n_estimator hyper parameter, which is defined as the maximum number of trees in a forest.

3.3.2 Best Hyperparameters

London :

PM2.5

- max_sample : 0.1
  If bootstrap is used, the number of samples to draw from X_train to train each base estimator
- max_feature : 2
  The number of features to consider when looking for the best split
- n_estimator : 300
- max_depths : 7

PM10

- max_sample : 0.9
- max_feature : 2
- n_estimator : 200
- max_depths : 7

Beijing :

PM2.5

- max_sample : 0.9
- max_feature : 2
- n_estimator : 500
- max_depths : 6

PM10

- max_sample : 0.7
- max_feature : 2
- n_estimator : 100
- max_depths : 6

O3

- max_sample : 0.9
- max_feature : 2
- n_estimator : 500
- max_depths : 6

3.3.3 Results MSE with bests hyperparameters

London :
PM2.5 : 9
PM10 : 29

Beijing :
PM2.5 : 415
PM10 : 489
O3 : 205

3.4 Generalized Additive Model

3.4.1 Algorithm

The basis of the GAM model is the same as a generalized linear model : $y = \beta_0 + x_1\beta_1 + \epsilon_1$ with ϵ following a normal distribution of mean 0. Now, instead of having only a linear component, we are going to change it by adding a smoothing term : $y = \beta_0 + f(x_1) + \epsilon_1$. The linear predictor is now some function f : that means that it can now be a curve that follows a pattern of data (compared to the linear which is a straight line). It can also be a combination of both : $y = \beta_0 + x_1\beta_1 + f(x_2) + \epsilon_1$ The $f(x_1)$ function is be determined using a sum of basis functions : $f(x_1) = \sum_{j=1}^{k} b_j(x_1)\beta_j$ where $b_j(x_1)$ can be a polynomial, Fourier or wavelet function (and more). The number of splines to use in the smoothing function is a hyper parameter. In modern techniques, the parameters can be determined by using the likelihood of the covariance : that allows us not to use the cross-validation (and so gain time).

3.4.2 Best Hyperparameters

London :

PM2.5 & PM10

- n_splines : 10
  number of splines

Beijing :

PM2.5 & PM10 & O3

- n_splines : 10

3.4.3 Results MSE with bests hyperparameters

London :
PM2.5 : 9
PM10 : 28

Beijing :
PM2.5 : 232
PM10 : 215
O3 : 164

3.5 Multi-Layer Neural Network

3.5.1 Algorithm

A Multi_Layer Perceptron(Neuron) Network is a hierarchical or multi-layered structure of networks. It's composed of a set of layer, each layer being composed of a set of neurons. A single neuron is a computational unit that has a weighted input signal and produces an output by using an activation function on the input signal. The activation function serves as a threshold at which the neuron is activated (and consequently the strength of the output). The main idea behind a neural network is first doing a forward pass. That means passing all the inputs into the network, making them go through all the layers and comparing the given output to the real one. Then, using the error of the comparison (here we used MSE), we can go back through the network and adjust all the weights accordingly. That's the second part called the back-propagation. All the errors can be saved through multiple training examples to update the network at the end.

The amount at which the weights are update is called the learning rate (here we fixed it at lr = 0.001). And to balance the learning rates, we define the number of iterations the model does through his training : that means the number of forward-pass/back-propagation it does (meaning the number of times the weights are updated). The balance between the two is subtle, as for the Gradient Boosting Tree, as we don't want to over-fit the model.

3.5.2 Best Hyperparameters

London :

PM2.5

- max_iterations : 1050
- activation : 'relu' :
  The rectified linear activation function is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. It became popular because it is easier to train and often achieves better performance

PM10

- max_iterations : 900
- activation : 'relu'

Beijing :

PM2.5

- max_iterations : 900
- activation : 'relu'

PM10

- max_iterations : 800
- activation : 'relu'

O3

- max_iterations : 1050
- activation : 'relu'

3.5.3 Results MSE with bests hyperparameters

London :
PM2.5 : 10
PM10 : 29

Beijing :
PM2.5 : 228
PM10 : 201
O3 : 168

3.6 XGBoost

3.6.1 Algorithm

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm so we won't go into the details as we already explained the gradient tree boosting. The reason why we decided to try XGBoost is that it is know for its many advantages :

- Regularization :
  Standard GBM implementation has no regularization like XGBoost, therefore it also helps to reduce overfitting.
- Parallel Processing :
  XGBoost implements parallel processing and is blazingly faster as compared to GBM.
- High Flexibility :
  XGBoost allows users to define custom optimization objectives and evaluation criteria and this adds a whole new dimension to the model and there is no limit to what we can do.

- Tree Pruning:
  A GBM would stop splitting a node when it encounters a negative loss in the split. Thus it is more of a greedy algorithm. XGBoost on the other hand make splits upto the max_depth specified and then start pruning the tree backwards and remove splits beyond which there is no positive gain.
  Another advantage is that sometimes a split of negative loss say -2 may be followed by a split of positive loss +10. GBM would stop as it encounters -2. But XGBoost will go deeper and it will see a combined effect of +8 of the split and keep both.

- Continue on Existing Model :
  User can start training an XGBoost model from its last iteration of previous run. This can be of significant advantage in certain specific applications.
  We didn't use that option but it had to be mentionned.

3.6.2 Best Hyperparameters

For general parameters :

- booster : [default = gbtree :tree-based models]
  Select the type of model to run at each iteration.

- silent : [default=0]
  When set to 0, running messages are printed. This is usefull for understanding the model.

- nthread : [default = [maximum number of threads available]
  This is used for parallel processing.

Booster parameters :
London :

- colsample_bytree : 0.7 (PM2.5), 0.7 (PM10)
  This denotes the fraction of columns to be randomly samples for each tree.

- max_depth : 1 (PM2.5), 3 (PM10)
  This denotes the maximum depth of a tree. This is used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample.

- subsample : 0.9 (PM2.5), 0.9 (PM10)
  This determines the fraction of observations to be randomly samples for each tree. It has to be carefully set because lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting.

- alpha : 2 (PM2.5), 2 (PM10)
  This determines the L1 regularization term on weight (analogous to Lasso regression) and can be used in case of very high dimensionality (our case) so that the algorithm runs faster when implemented.

Beijing :
All these parameters turned out to be the same for PM2.5, PM10 and O3.

- colsample_bytree : 0.9
- learning_rate : 0.2
- max_depth : 2
- subsample : 0.9
- alpha : 2

Learning task parameters :

- objective : binary:logistic
  This was the most logical for our data, we just had to scale them between 0 and 1.
- metric : rmse (root mean squared error)
  To compare with other methods, it has to be squared.
- seed : 123
  This is used for generating reproducible results.

3.6.3 Results MSE with bests hyperparameters

London :
PM2.5 : 25
PM10 : 49

Beijing :
PM2.5 : 256
PM10 : 225
O3 : 169

4 Method comparison and selection

When models are compared, some of the differences might be simply random. In order to avoid this, significance tests are supposed to be done.
However, in our case, a few details show that we don't particularly need big significance tests.
First, we're using regression and not classification which makes our comparison laying on mean squared error. Knowing this, fewer significance tests are a viable options; for example the famous McNemar's test can not be used.
Then, our training set and validation test are the same for each model tested. This means that our data is the same, with the same distribution.

As a result, we relied on the mean squared error [2] computed for each model. This mean squared error has been calculated using a 10 partitions cross validation on the best hyperparameters (when some are needed for the model), and these hyperparameters have been chosen having the biggest (the closest to 0) negative mean squared error with a 10 partitions cross validations.
As a conclusion, the best model selected is the one with the smallest mean squared errors.

5 Best Method applied

As stated before, when comparing the mean squared errors of our methods, we were able to determine that the most suitable method for our problem is the Gradient Boosting Tree. Having also pre-determined the best hyperparameters, we can now apply and test the method on our dataset.

5.1 London

5.1.1 MSE

PM2.5 : mean(14,16,16,20,9,16,7,23,6,29,10,42,18) = 17.385

PM10 : mean(14,16,15,18,9,22,8,19,6,27,7,38,20) = 16.615
The figures 4,5,6,7,8 and 9 show some comparison between the predicted and real values for two stations. All plots are available in the folder "london_plot_prediction".

5.1.2 Visualisation

This can be noticed that the slope of the predicted values is a little bit smoother, which is normal as we tried to find a best model to approximate values in many cases. Exception like peak are hardly handled if we want to prevent overfitting.

5.2 Beijing

5.2.1 MSE

PM2.5 : mean(81, 60, 74, 66, 78, 71, 45, 569, 125, 30, 164, 34, 185, 46, 69, 50, 58, 69, 30, 54, 102, 378, 70, 409, 67, 93, 100, 56, 0.73, 67, 126, 74, 82, 82) = 104.706

PM10 : mean(288, 104, 95, 119, 192, 494, 46, 1173, 383, 108, 1645, 133, 218, 244, 254, 124, 253, 150, 25, 88, 171, 593, 161, 1322, 43, 52, 3309, 139, 0.62, 83, 155, 86, 182, 146) = 359.389

O3 : mean(158, 451, 183, 155, 109, 164, 205, 162, 84, 168, 431, 109, 118, 147, 95, 328,

[2] MSE $= \frac{1}{n}\sum_{i=1}^{n} Y_i - \hat{Y}_i$, this measure the average squared difference between the estimated values and the actual value

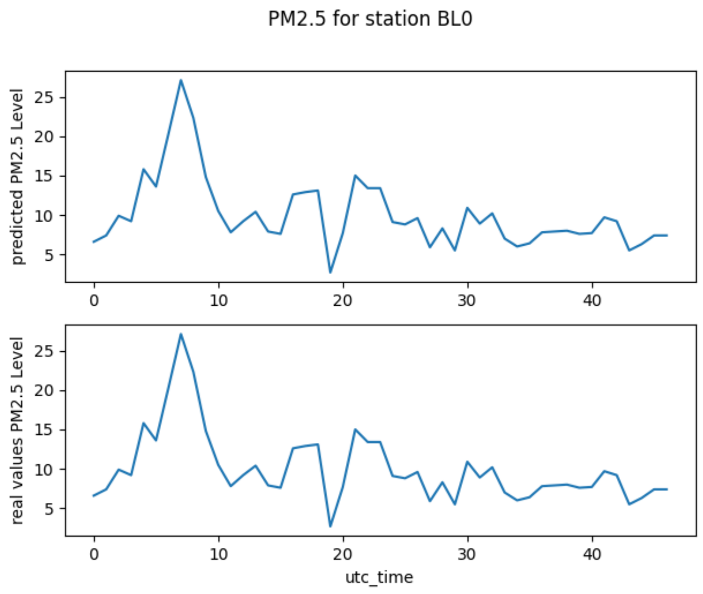

Figure 4: comparison between the predicted and true PM2.5 level for BL0

208, 257, 38, 118, 286, 168, 135, 101, 97, 60, 327, 1070, .015, 169, 142, 8, 72, 207) = 186.572

5.2.2 Visualisation

6 Conclusion

Having a look at the results, our final model is efficient and could be proposed.
All models were relatively similar for the London data set. However, there were some significant differences inside the Beijing data set. The error (MSE) would vary from 10-30 in the London dataset whereas it's as high as 100-200 in the Beijing dataset. The cause could be because of the number of stations studied. London has 13 while Beijing has 35 : the variety and disparity of the data is much greater in Beijing and drawing a model is consequently more difficult. Another reason would be the feature selection : we used the correlation matrix to determine which feature should be maintained in order to have a great prediction in both cities. Perhaps a more adapted and advanced technique would have yield better results.

We can also add that the computation time is to take into account when choosing a method. In our case, we had to first determine the best hyperparameters for each of the methods. Once they were set, the computation time to predict the data with the best method (with the given hyperparameters) was negligible : that means that only the hyperparameters were taken into account to determine the best suitable method.

Nonetheless the method used has given a pretty decent prediction, especially for London.

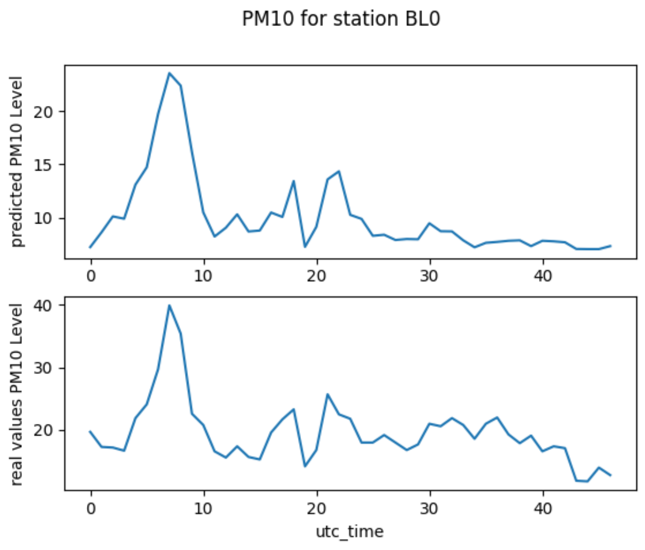

Figure 5: comparison between the predicted and true PM10 level for BL0

7 To go further

For the stations to predict with not enough data we could have taken instead the data from the closes station. This could have been easily done with the latitude and longitude.
An other point if we wanted to improve even more this model is that non-meteorological parameters could be explored .
First, taking in count the day of the week. For example people would less use the car during the week-end because they should not go to work and this would impact the level of pollution.
An other possibility would be taking in count the month of the year. For example, when there are holidays, the pollution might be different than when people are working.

8 Resources Used

8.1 Python libraries

- Numpy
- Panda
- Sklearn
- Pygam
- Xgboost
- Matplotlib

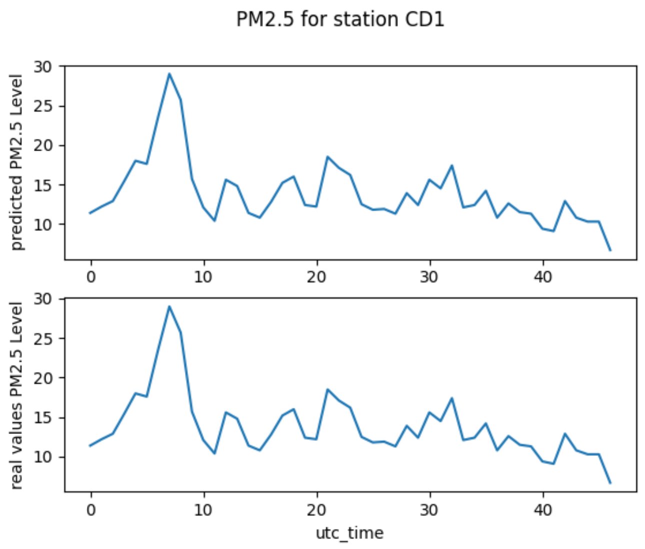

Figure 6: comparison between the predicted and true PM2.5 level for CD1

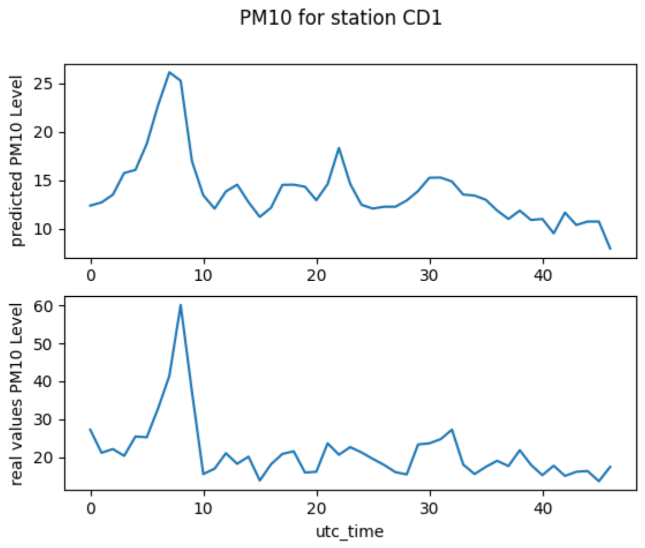

Figure 7: comparison between the predicted and true PM10 level for CD1

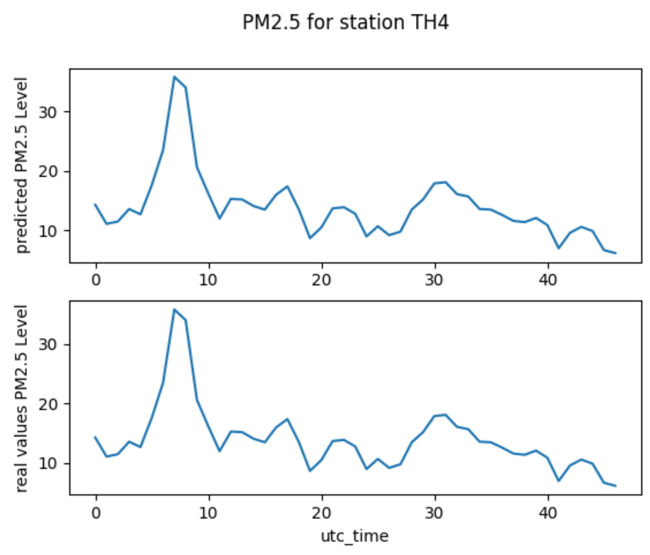

Figure 8: comparison between the predicted and true PM2.5 level for TH4

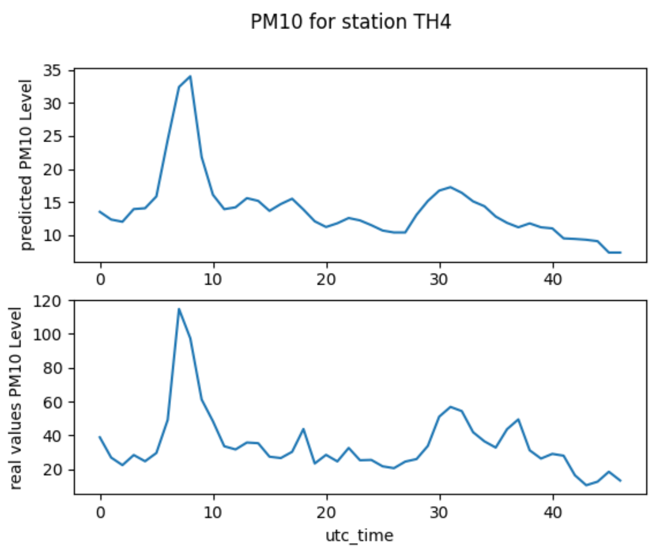

Figure 9: comparison between the predicted and true PM10 level for TH4