NYCU
National Yang Ming Chiao Tung University

Data Mining
HW1

Sylvain Thong (312551818) sthong.cs12@nycu.edu.tw

01/04/2024 - Version 1.1

<u>Table of contents :</u>

I - Beforehand manipulations

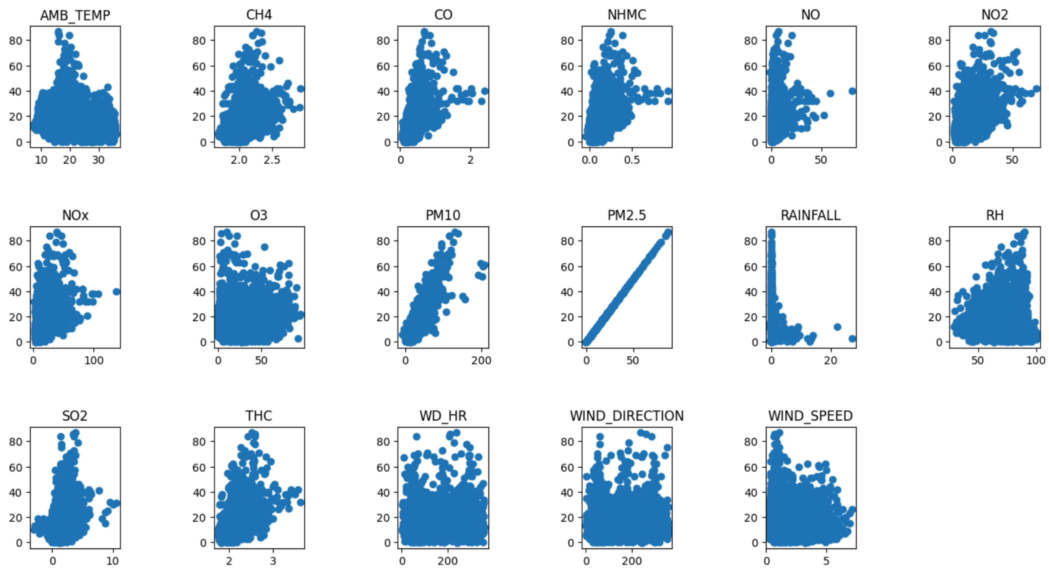
A) Features selection

The dataset comprises several features that might be correlated to PM 2.5 values. Therefore, in order to prevent overfitting that could lead to potential worse performance, we need to determine which features we should take into account when doing our computation.

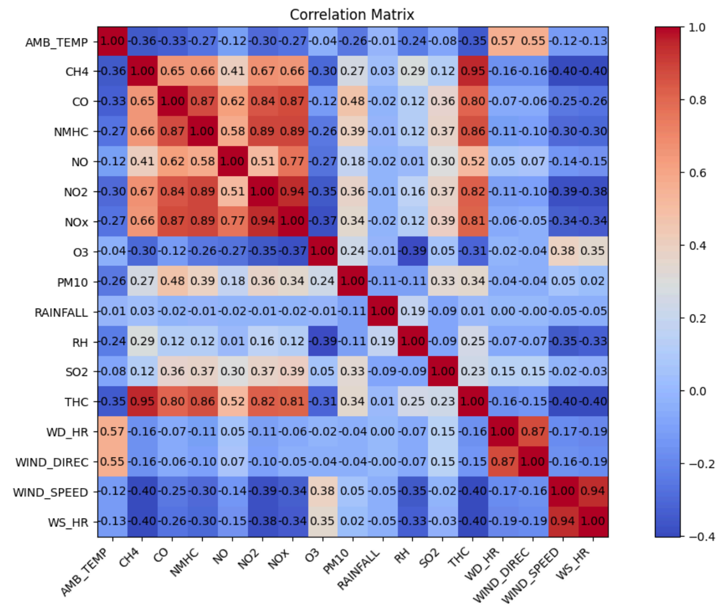
First of all, we make the assumption that the PM 2.5 value depends primarily on the features' values of the previous 9 hours (so that we can make sure that our test dataset will be fully utilized).

NB : This assumption explains our use of the window sliding method in the source code to improve our predictions by taking into account the previous features' values for the past 9 days.

Secondly, with this hypothesis, we checked if some features were strongly correlated with PM2.5 :



Hence why, at this stage, we could assume that there was no obvious correlation and that we could not exclude any features at this point.
Thirdly, we checked the correlation value between all features to see if some were strongly correlated between them : it is important to exclude such features from playing both a role in the computation because it could lead to potential overfitting and increased error margins :



From this standpoint, we excluded the following features with the correlation value limit taken at 0,8 :

- NHMC
- NO2
- NOx
- THC
- WD-HR
- WS-HR

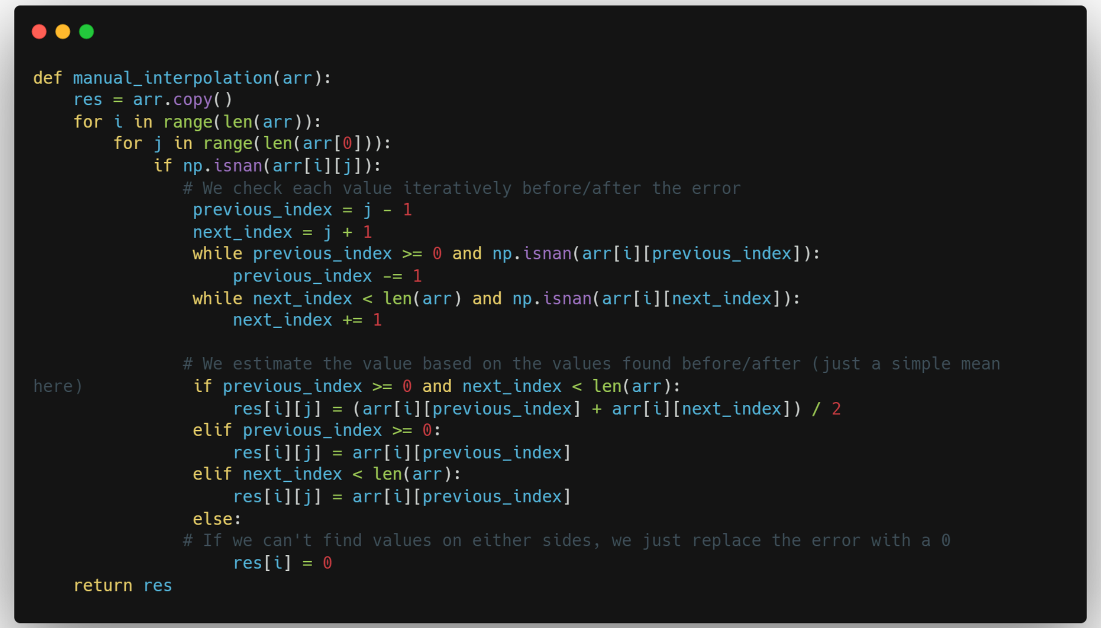
Therefore, we decided to take into account : **AMB_TEMP, CH4, CO, NO, O3, PM2.5, PM10, RAINFALL, RH, SO2, WIND_DIRECTION and WIND_SPEED values.**

B) Preprocessing

Another key element in order to increase the likelihood of returning better results after the computation is to pre-process the input values in order to tackle values that can sometimes be error values, far-fetched from reality because of measurement inconsistencies…

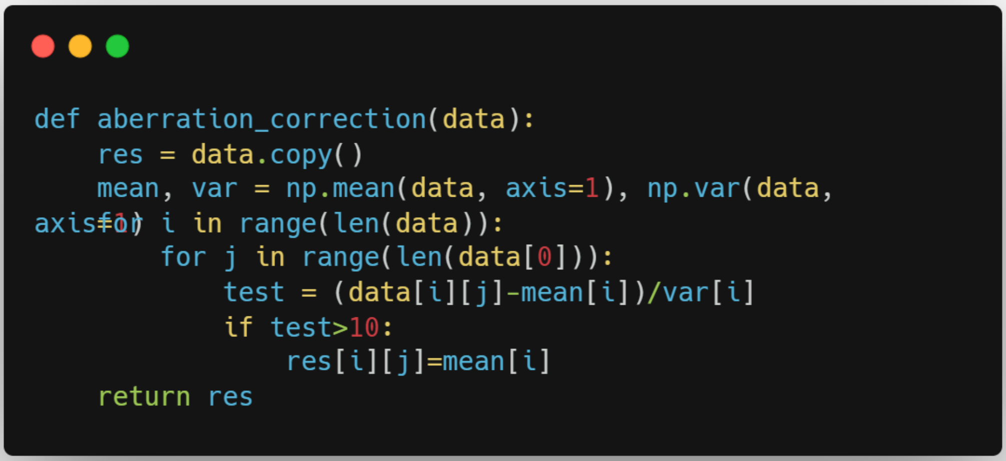
First of all, we replaced the “error values” with values that we estimated via interpolation (in this case, we only limited the process to second order polynomial interpolation).

The interpolation method estimates a maximum amount of values between to correct values. Therefore, we also introduced a simple method to get rid of those : we checked the values before (respectively after) the faulty one and used it to replace the error with an estimation. This process is implemented in the following manual_interpolation method :



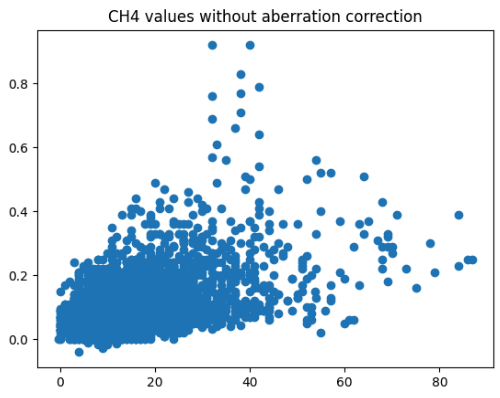
```
def manual_interpolation(arr):
    res = arr.copy()
    for i in range(len(arr)):
        for j in range(len(arr[0])):
            if np.isnan(arr[i][j]):
               # We check each value iteratively before/after the error
                previous_index = j - 1
                next_index = j + 1
                while previous_index >= 0 and np.isnan(arr[i][previous_index]):
                    previous_index -= 1
                while next_index < len(arr) and np.isnan(arr[i][next_index]):
                    next_index += 1

               # We estimate the value based on the values found before/after (just a simple mean
here)           if previous_index >= 0 and next_index < len(arr):
                    res[i][j] = (arr[i][previous_index] + arr[i][next_index]) / 2
                elif previous_index >= 0:
                    res[i][j] = arr[i][previous_index]
                elif next_index < len(arr):
                    res[i][j] = arr[i][previous_index]
                else:
               # If we can't find values on either sides, we just replace the error with a 0
                    res[i] = 0
    return res
```

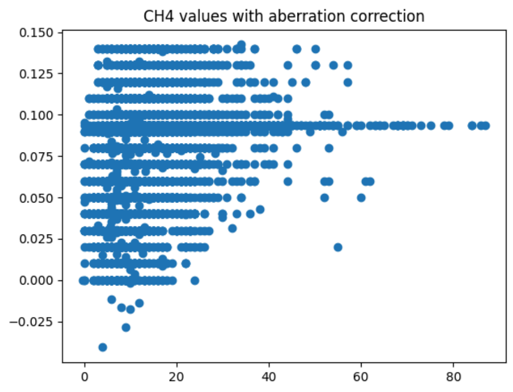
Finally, after checking some of the features' values, we found out that some of them had some aberrant ones (like a value that is too high for it to be considered relevant or not error-related). We are using the following method in order to determine if a value is aberrant and if it needs to be replaced or not (we just chose to use the mean value instead) :



```
def aberration_correction(data):
    res = data.copy()
    mean, var = np.mean(data, axis=1), np.var(data,
axis=1)
    for i in range(len(data)):
        for j in range(len(data[0])):
            test = (data[i][j]-mean[i])/var[i]
            if test>10:
                res[i][j]=mean[i]
    return res
```

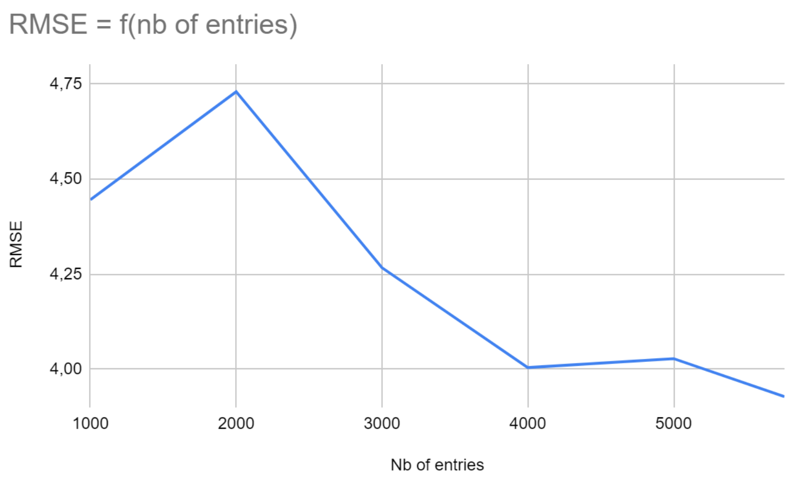
As instance, with this method, we obtain the following correction for CH4 values :





II - Impact of different amounts of training data on the PM2.5 prediction accuracy

Each feature has 5760 values. In order to check if we have the maximum accuracy whether or not we take all the available information into account, we computed the RMSE for different numbers of entries based on the training dataset.

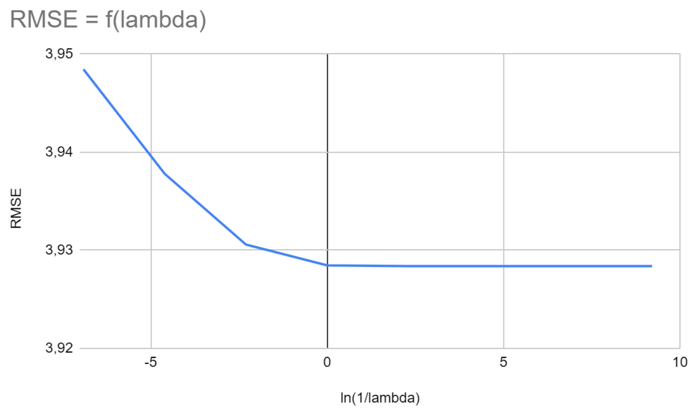


Therefore, in order to compute coefficients that can best-fit a wide range of values and to reduce the RMSE value to a minimum, we need to have the largest training dataset possible ! More sample values means that we can potentially compute more precisely. But we also need to be aware of not overfitting our model to only the training dataset…

III - Regularization

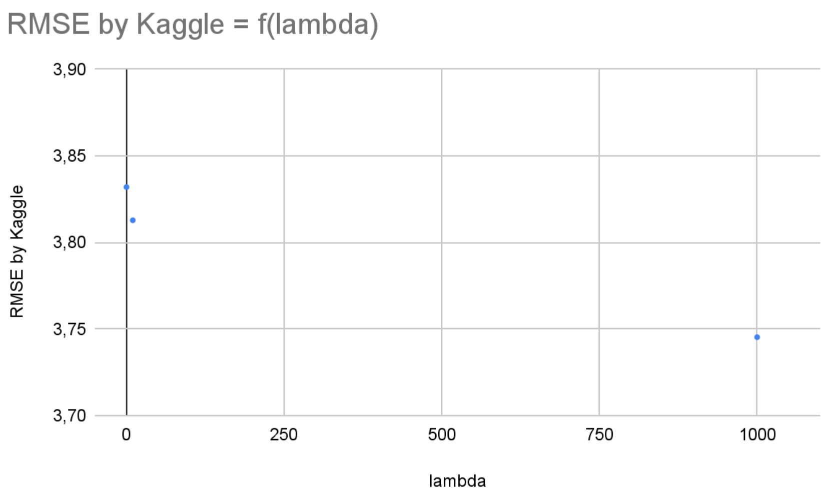
When performing linear regression, regularization is a way to prevent and tackle the problem of overfitting : this phenomenon happens when our trained model attempts to fit all given features to a point that it can mislead the search for the solution.
In order to visualize the impact of regularization, we computed the RMSE values based on the training dataset with different lambda values for regularization.

NB : Without regularization, we have a RMSE value of 3.9283783973026503



With lambda in the range of 0 (no regularization) to 1000, we can see that having a lambda value that is too high can lead to worse results.
However, the graph shows that the regularization method does not actually improve our accuracy based on the training dataset.

The point of regularization is to prevent overfitting. Therefore, we can make the assumption that the method doesn't indeed improve our performance as seen by the RMSE computed on the training dataset but instead allows our computed model to maybe fit our test dataset better.
In order to check this fact, we compared the score given by the Kaggle verification system for different values of lambda :



Hence why, we can assess that regularization improves our results by allowing our model to fit the test dataset better (the score dwindled by 0,08671 with lambda = 1000 compared to no regularization).