European Commission | Horizon 2020 European Union funding for Research & Innovation

CAPTOR

Collective Awareness Platform for Tropospheric Ozone Pollution

Work package	WP2
Deliverable number	D2.9
Deliverable title	Release of Estimation Software (b).M24
Deliverable type	R
Dissemination level	PU (Public)
Estimated delivery date	M24
Actual delivery date	30/12/2017
Actual delivery date after EC review	--
Version	1.0
Comments	

Authors
Jose M. Barcelo-Ordinas, Jorge Garcia-Vidal, Manel Guerrero-Zapata (UPC)

Project funded by the European Union's Horizon 2020 research and innovation programme under Grant Agreement N° 688110

Document History			
Version	**Date**	**Contributor**	**Comments**
V0.1	5/12/2017	Jose M. Barcelo-Ordinas (UPC)	Outline
V0.2	10/12/2017	Jose M. Barcelo-Ordinas, Jorge Garcia-Vidal, Manel Guerrero-Zapata (UPC)	First draft
V0.3	28/12/2017	Roger Garcia (guifi)	Peer review
V1.0	30/12/2017	Jose M. Barcelo-Ordinas, Jorge Garcia-Vidal, Manel Guerrero-Zapata (UPC)	Final version

Table of Contents

List of Figures

List of Abbreviations

3G	Third Generation (mobile cellular phones)
API	Application Programming Interface
ARPA	Agencia Regionale per le Protezione Ambientale
CO	**C**arbon monoxide
CSV	Comma Separated Values
mAh	mili Ampere hour
MLR	Multivariate Linear Regression
NOx	Nitrogen Oxides
NO_2	Nitrogen Dioxide
O_3	Ozone
PR	Palau Reial reference Station
RH	Relative Humidity
RSS	Residual Sum of Squares
RSE	Residual Standard Error
T	Temperature
UTC	Coordinated Universal Time
VCO	Volatile Organic Compounds

Executive Summary

Description of the work

This deliverable is a continuation of Deliverable 2.3 that described how the data taken from the CAPTOR ozone sensor nodes was processed in order to produce ozone data with certain degree of quality. In this deliverable, is described the software implementation produced for calibrating Ozone in CAPTOR nodes.

Objectives

The main objectives of the deliverable are:

- Describe the mechanism for calibrating low-cost sensors
- Provide a software tool for calibrating nodes with low-cost Ozone sensors for captor and raptor nodes.

1. Research and Technological Context

The purpose of this deliverable is to provide a tool for calibrating Ozone for low-cost sensors. Deliverable 2.3 described how to calibrate metal-oxide Ozone sensors. Here, in this deliverable, we describe and give the code to calibrate Ozone using the formulation described in Deliverable 2.3.

The process of calibration consists of transforming raw data taken from the CAPTOR sensor nodes to real ozone concentrations with the best possible quality in terms of relative error with respect ground truth data measured by accurate reference stations. The **ground truth data** is defined as the data taken by direct observation, i.e., by a reference station, in contrast to that one that is provided by inference.

CAPTOR nodes are built with low-cost sensors, e.g., metal-oxide ozone sensors or electro-chemical sensors that are not calibrated in specialized laboratories like the reference stations. For example, when a low-cost metal-oxide sensor interacts with the pollutant its resistor measures a value that represents the ozone concentration in terms of electric resistance. A multivariate linear regression is then used in order to calculate the ozone concentrations. In this deliverable, it is described how to obtain ozone concentrations by regressing over ground truth surface ozone concentrations measured by reference station instrumentation.

Gas sensors are sensors that follow multiple linear responses. Tropospheric ozone, (O_3), formation occurs when nitrogen oxides (NOx), carbon monoxide (CO) and volatile organic compounds (VOCs), react in the atmosphere in the presence of sunlight. In general, in order to calibrate the O_3 sensors and depending on the type of sensor (metal-oxide or electro-chemical), it is needed to measure O_3, NO_2, temperature and relative humidity. Experiments to measure O_3 have been performed in the H2020 CAPTOR project testbeds in Spain, Italy and Austria during the 2017 summer ozone campaign. The testbed consists of two types of nodes. The first type of node called ***Captor nodes*** and built by UPC, Barcelona, Spain, following the DIY (Do It Yourself) philosophy, uses Arduino technology with a sensor shield board that attaches four SGX Sensortech MICS 2614 metal-oxide O_3 sensors in each Captor node, a MQ131 metal-oxide O_3 sensor for a total of five O_3 sensors, a temperature (Temp) sensor and a relative humidity (RH) sensor. Each Captor node is powered from an external power supply and it is connected to Internet using Wifi or 3G.
The second type of node called ***Raptor*** and built by Limos-UCA, France, uses Raspberry technology with one Alphasense O3B4 electro-chemical O_3 sensor, one AlphaSense NO2B4 electro-chemical NO_2 sensor, a temperature sensor and a relative humidity sensor. The Raptor outdoor node is powered by a 9V 4000mAh battery for a lifetime of 3 months, and connected using a IEEE802.15.4 (ZigBee) wireless access medium to an indoor Raptor local server, powered from an external power supply and connected to Internet using Wifi or 3G.
Captor nodes have been calibrated using reference stations in Spain and Italy: **Palau Reial** reference station in Barcelona town, Spain (41º23'14''N, 2º6'56''E), operated by CSIC (Spanish National Research Council) and the Regional Government of Catalonia (Spain), **Manlleu** (42º0'6.966''N, 2º17'13.7868’’E), **Tona** (41º50'49.7796''N, 2º13'14.7864''E), **Vic** (41º56'08.4''N, 2º14'18.8''E) and **Montseny** (41º46'45.6''N, 2º21'28.9''E) are reference stations operated by the Regional Government of Catalonia (Spain). Cuneo (44º22'53.6''N, 7º32'18.4''E), in Piemonte, **Parco di ColliEuganei** (45º17'21.76''N, 11º38'32.43''E) in Veneto, Parco **di MonteCucco** (45º02'18.8''N, 9º40'09.7''E) in Emilia Romagna, and **OssioSoto** (45º37'14.1''N, 9º36'41.6''E), in Lombardia are reference stations operated by ARPA (Agencia Regionale per le Protezione Ambientale) in Italy. Palau Reial reference station is an urban reference station in a large town like Barcelona, where Ozone in average in summer is low. The other reference stations, in Spain and Italy, are placed in a

country-side area where Ozone in summer is high and are nearby the volunteer houses where the nodes were placed.

Raptor nodes have been calibrated using reference stations in Spain, Italy and Austria: **Palau Reial** reference station in Barcelona town, Spain (41º23'14''N, 2º6'56''E) in Spain. Cuneo (44º22'53.6''N, 7º32'18.4''E), in Piemonte, **Parco di ColliEuganei** (45º17'21.76''N, 11º38'32.43''E) in Veneto, Parco **di MonteCucco** (45º02'18.8''N, 9º40'09.7''E) in Emilia Romagna, and **OssioSoto** (45º37'14.1''N, 9º36'41.6''E), in Lombardia are reference stations operated by ARPA (Agencia Regionale per le Protezione Ambientale) in Italy. Finally, **Vienna** (48º14'17.4''N, 16º22'42.6''E) reference station for raptor nodes in Austria.

For the calibration of sensors, we have followed a pre-post calibration approach. The objective is to learn what is the impact of the environment in the calibration process and what is the impact of the passage of time on the sensors. The nodes have been submitted to the following process:

- **Phase 0:** all sensors have been calibrated between two to three weeks in Palau Reial reference station during the month of May, 2017. In this phase, the objective is to calibrate the nodes in an environment near the place the nodes have been built.
- **Phase 1:** in this phase, called *pre-calibration* phase, all captor nodes have been placed in reference stations nearby the final places in which the nodes were located during phase 3.
- Phase 2: Twenty nodes have been placed in volunteer houses nearby reference stations during the months of July, August and September. Several nodes have remained in local reference stations with the objective of having several nodes during large periods of time in a reference station in order to check their performance against ground-truth data.
- **Phase 3:** in this phase, called *post-calibration* phase all nodes have been placed for recalibration during two weeks in the same locations that were placed during the pre-calibration process.

2. Calibration of CAPTOR ozone nodes

In the following, we summarize one of the methods described in Deliverable 2.3 and that finally was chosen to be implemented for the calibration of the Captor nodes.

2.1 Mathematical background

Each CAPTOR node is deployed on the roof of a reference station during a period time of at least 3 weeks. In general, the calibration of a sensor means to approximate the true value Y by a function f(X):

$$Y = f(X) + \varepsilon \qquad (2)$$

Where *f* is a fixed but unknown function, X is a vector of *p* predictors or input variables and ε is a random error term distributed as a zero mean Gaussian random variable with variance σ^2, i.e, $\varepsilon \sim N(0, \sigma^2)$ and independent of X. In this approximation, eq (2) is modelled by saying that we are **regressing Y on X** (or Y onto X).

In order to find a regression of the data, we may consider linear combinations of fixed non-linear functions of the input variables, of the form:

$$f(X) = \beta_0 + \sum_{i=1}^{p} \beta_i \phi_i(X)$$

where $\phi_i(X)$ are known as **basis functions** and β_0 is the **slope or intercept** and the β's (i=1,…,p) are the **regression coefficients**. The most basic model is using a Multivariate Linear Regression (MLR) in which each basis function ϕ_i is linear with respect X_i i.e., $\phi_i(X)=X_i$:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon = \sum_{i=0}^{p} \beta_i X_i + \varepsilon = \beta X + \varepsilon \qquad (3)$$

It is to say, Y is approximated by a linear combination of the predictors. Note that the dimension of Y and X_i (i=1,…,p) is N, the size of the sample set ($X_i, Y \in R^N$ or $X \in R^{N \cdot (p+1)}$), where X has been extended by a vector X_0 of 1's and the slope β_0 has been integrated in the β. More complicated basis functions may be used, such as powers of x, $\phi_i(X)=X_i^j$ or polynomial functions of several features.

In the calibration process for the CAPTOR project, the Multivariate Linear Regression (MLR) will be used. In order to regress Y on X, the coefficients β have to be approximated by β'. Our goal is to obtain coefficient estimates β' such that the linear model of eq (3) fits the available data well, that is, so that $y \approx \beta' X$. In other words, we want to find those coefficients β' where by the resulting line is as close as possible to the N data points. We refer to James et al[1] for finding the β', e.g., minimizing the least squares criterion.

Let $y_i' = \beta' X_i$ be the prediction of y_i based on the value of x_i. The difference between the estimated value y_i' and the original value y_i is called the **residual**, $e_i = y_i - y_i'$. We define the Residual Sum of Squares (RSS) as:

$$RSS = e_1^2 + \ldots + e_n^2 = \sum_{i=1}^{n} (y_i - y_i')^2 = \sum_{i=1}^{n} (y_i - \beta_i' x_i)^2 \qquad (4)$$

We wonder how close are the β' from the real true β. In computing the standard errors in β', they depend on the variance σ^2 of the error ε. However, this variance is unknown. A way of estimating this variance is to define the Residual Standard Error (RSE), defined as:

$$RSE = \sqrt{\frac{RSS}{n - p + 1}} \qquad (5)$$

2.2 Calibration Procedure for a captor node with a single ozone sensor

In this section, we focus in captor nodes that use metal-oxide sensors. Let us assume that the data set for calibration has size N. We assume that each of the M ozone sensors is independent of each other. The data consist of:

- The reference station data $Y \in R^N$,
- The ozone (O_3) data captured by each sensor $X_1 = X \in R^N$, with M ozone sensors,
- The Relative Humidity (RH) data captured by the sensor $X_2 = HR \in R^N$,
- The Temperature (T) data captured by the sensor $X_3 = T \in R^N$,

The MLR model used is, then:

[1] Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, "An introduction to statistical learning, with applications in R", Springer, 2013.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \varepsilon \qquad (6)$$

Where we have recalled X_1=X (ozone), X_2=HR (Relative Humidity) and X_3=T (Temperature) for commodity. In order to calibrate a CAPTOR node with a single ozone sensor, we proceed as follows:

- The data set N is divided in two sets: the **training set** of size N_1 and the **test or validation set** of size N_2.
- Obtain the β' by minimizing the least squares criterion over the training set and obtain the RRSE as quality parameter of the training set by using the RSS of the training set.
- Predict the $y' = \beta_0' + \beta_1' X + \beta_2' HR + \beta_1' T$ where $X, HR, T \in R^{N2}$ are data of the validation set. Obtain the RSE of the validation set by using the RSS of the test set.

At the end of the process, each M individual sensor is calibrated per each CAPTOR node. Now the question is which one represents best the CAPTOR node. The sensor that has less validation RSE is taken as reference sensor for that node. Now, the ozone sensor is calibrated and the ozone concentration can be predicted by new values using formula, where X_{cal} is the calibrated value and the subscript new means new uncalibrated collected data:

$$X_{cal} = \beta'_0 + \beta'_1 X_{1new} + \beta'_2 X_{2new} + \beta'_3 X_{3new} \qquad (7)$$

2.3 Calibration Procedure for a raptor node with a single ozone sensor

In this section, we focus in raptor nodes that use electrochemical sensors. Let us assume that the data set for calibration has size N. The electrochemical sensors use NO_2, O_3, T and HR sensors to calibrate Ozone. The data, then, consist of:

- The reference station data $Y \in R^N$,
- The NO_2 data captured by each sensor $X_2 = X \in R^N$,
- The ozone (O_3) data captured by each sensor $X_1 = X \in R^N$,
- The Relative Humidity (RH) data captured by the sensor $X_3 = HR \in R^N$,
- The Temperature (T) data captured by the sensor $X_4 = T \in R^N$,

The MLR model used is, then:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \varepsilon \qquad (8)$$

Where we have recalled X_1=X (NO_2), X_2=X (O_3), X_3=HR (Relative Humidity) and X_4=T (Temperature) for commodity. In order to calibrate a raptor node with an electrochemical ozone sensor, we proceed as follows:

- The data set N is divided in two sets: the **training set** of size N_1 and the **test or validation set** of size N_2.
- Obtain the β' by minimizing the least squares criterion over the training set and obtain the RSE as quality parameter of the training set by using the RSS of the training set.
- Predict the $y' = \beta_0' + \beta_1' X + \beta_2' HR + \beta_1' T$ where $X, HR, T \in R^{N2}$ are data of the validation set. Obtain the RRSE of the validation set by using the RSS of the test set.

At the end of the process, the ozone sensor is calibrated and the ozone concentration can be predicted by new values using formula, where X_{cal} is the calibrated value and the subscript new means new uncalibrated collected data:

$$X_{cal} = \beta'_0 + \beta'_1 X_{1new} + \beta'_2 X_{2new} + \beta'_3 X_{3new} + \beta'_4 X_{4new} \qquad (9)$$

3. Software development for the calibration of captor/raptor sensor nodes

The software tool to calibrate the sensors of Captor nodes have been developed by UPC and uses python as a programming language. For using the software it is needed that the user install python (open source) and install the following python modules:

- Numpy
- sklearn.utils
- sklearn
- csv
- matplotlib

These modules are used for uploading the data, plotting and form mathematical manipulation. In the following the software is described.

3.1 Data format for the captor calibration software

The software needs that the data is provided as a CSV file with data in the following format:

date; RefSt; S1; S2; S3; S4; S5; T; RH

where:

- **Date:** is the date in which the sample was taken. The date is in UTC (Coordinated Universal Time) format, e.g. dd-mm-yyyy T hh:mm:ss or dd/mm/yy hh:mm:ss.
- **RefSt:** is the value of Ozone of the reference station in which the node has been placed
- **S1; S2; S3; S4; S5:** are the values taken by the five Ozone sensors (resistance values)
- **T:** is the value of the temperature sensor.
- **RH:** is the value of the Relative Humidity sensor.

For example, here there are five samples taken from a node:

```
date; RefSt; S1; S2; S3; S4; S5; T; RH
2017-09-24T04:30:48;69.8937;62.3520;59.6113;47.8217;370.3207;16.03;75.27
2017-09-24T05:00:49;86.5677;89.2967;98.8177;76.0747;575.8170;15.63;76.00
2017-09-24T05:30:51;100.3943;114.8940;133.6520;90.8980;1090.0657;15.00;76.00
2017-09-24T06:00:52;110.8643;131.2980;157.1250;103.0783;3017.3080;15.00;76.00
2017-09-24T06:30:54;109.9107;130.9013;148.3290;96.0383;5246.5073;15.00;75.23
```

3.2 Data format for the raptor calibration software

The software needs that the data is provided as a CSV file with data in the following format:

date; RefSt_NO2; RefSt_O3; S_NO2; S_O3; T; RH

where:

- **Date:** is the date in which the sample was taken. The date is in UTC (Coordinated Universal Time) format, e.g. dd-mm-yyyyThh:mm:ss or dd/mm/yy hh:mm:ss.
- **RefSt_NO2 and RefSt_O3:** is the value of NO_2 and Ozone of the reference station in which the node has been placed
- **S_NO2; S_O3:** are the values taken by the electrochemical sensor for NO2 and Ozone sensors.
- **T:** is the value of the temperature sensor.
- **RH:** is the value of the Relative Humidity sensor.

For example, here there are five samples taken from a node:

```
date; RefSt_NO2; RefSt_O3; S_NO2; S_O3; T; R
06/07/2017 15:00;7;174;-2.7;24.6;34.8;29.6
06/07/2017 16:00;6;174;-2;24.9;35.4;29.2
06/07/2017 17:00;6;175;-2.1;25.4;35.5;28.5
06/07/2017 18:00;7;176;-1.8;26.4;35.7;27.6
06/07/2017 19:00;9;179;-1.2;27.2;35.1;28.6
```

3.3 Modules and output of the captor sensor nodes calibration software

The python software obtains the calibration coefficients and plots the data, the normalized data, and scatterplots. A library with several functions has been created for plotting the data. The code for all the process can be found in appendix A.

The input of the software is a file in CSV format, section 3.1.

The output of the software calibration is a set of files that is written in a specified folder:

- **Scatterplots** in which in the x-axes is the normalized reference station data and in the y-axes is the normalized Ozone sensor data. The scatterplot allows us to see how linear is the data.

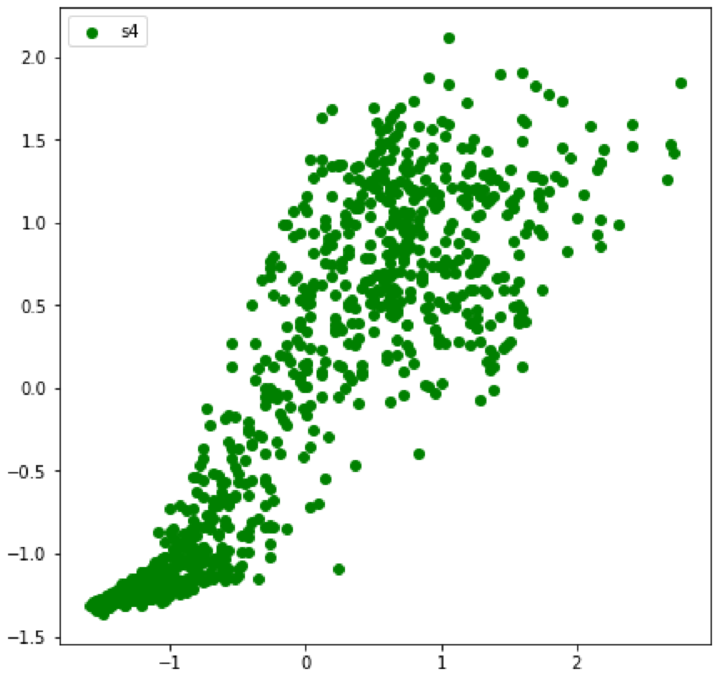

Figure 1. Scatterplot of sensor S4

- **Plots with the raw data** of every uncalibrated Ozone sensor (resistances) and the reference station. This kind of plot allows us to see whether the sensor data follows the same patterns that the reference station. For example, these plots allow us to identified change of scales, gaps in the data or peaks that show a malfunction in some of the sensors.

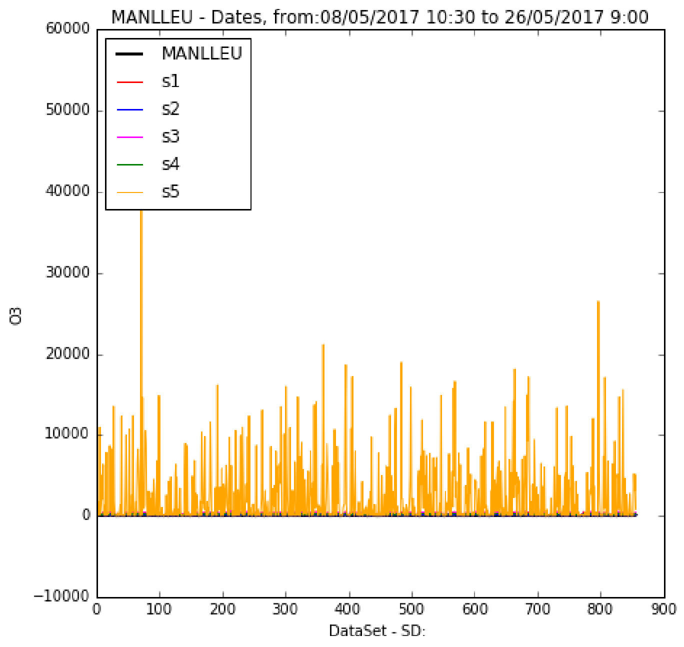

Figure 2. Data Set of Captor 17001.

- **Plots with the normalize data** of every Ozone sensor and the reference station can be drawn. What it is done, is to normalize all the data (sensors and reference station) with respect the mean and variance. For example, for an Ozone sensor whose data is stored in vector x=(x_1,…,x_N), with N the number of samples, the normalized variable x_{norm} of x is defined as x_{norm} = (x-μ)/σ, where μ is the mean of vector x and σ is the standard deviation of vector x. These plots allow us to better visualize the temporal behaviour of the uncalibrated data with respect the reference station data. The plot is very similar to the previous one, but all the data is in similar scale, which allows a better understanding of the data.

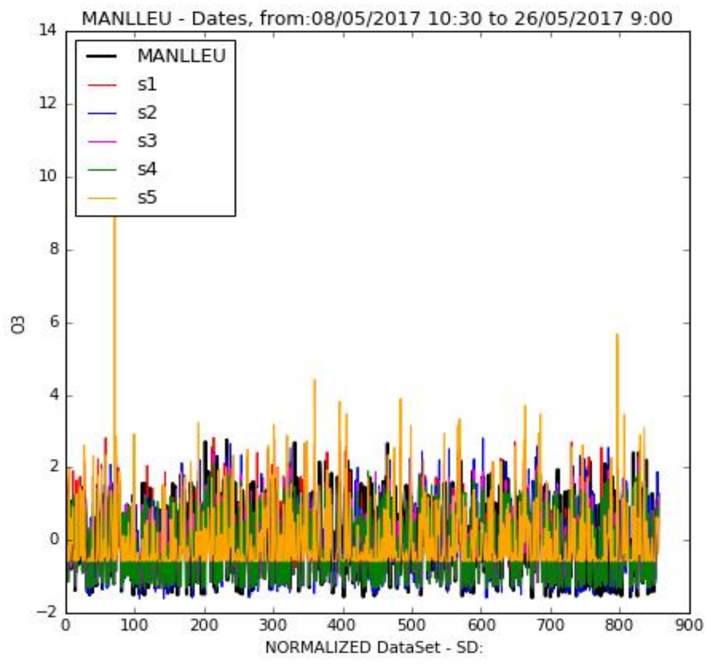

Figure 3. Data Set Normalized of Captor 17001.

- **Plots with the calibrated data of every sensor:** it is plotted the calibrated ozone concentrations for the training, validation and whole data. The training and validation data sets are shuffle, it is to say, the samples of the training and validation are randomly taken from the data set. For this reason, in these plots, the data is shown with these peaks. On the other hand, in the plot with the whole data set, the samples are re-ordered, and then plotted in the correct order.

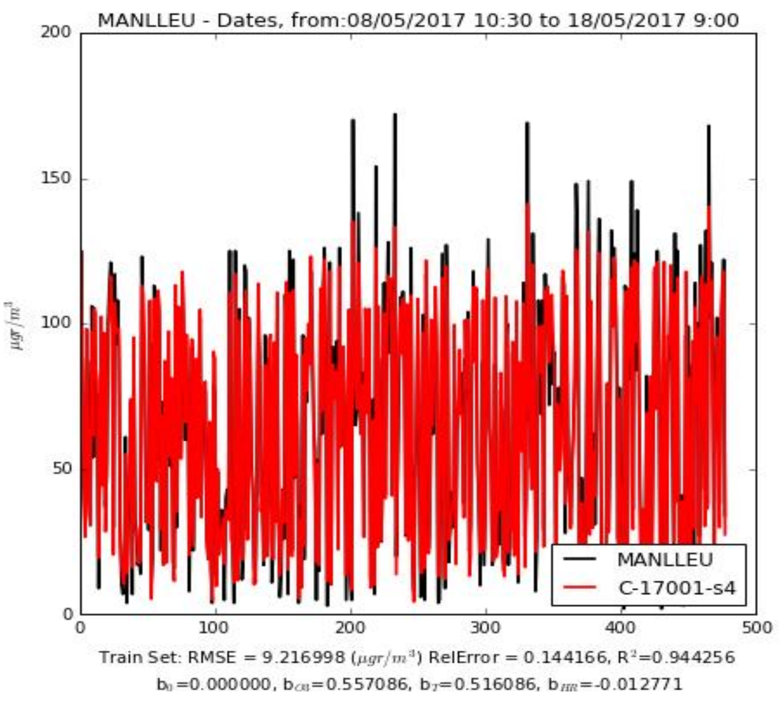

Figure 4. Calibrated Ozone for Training of sensor S4 Captor 17001.

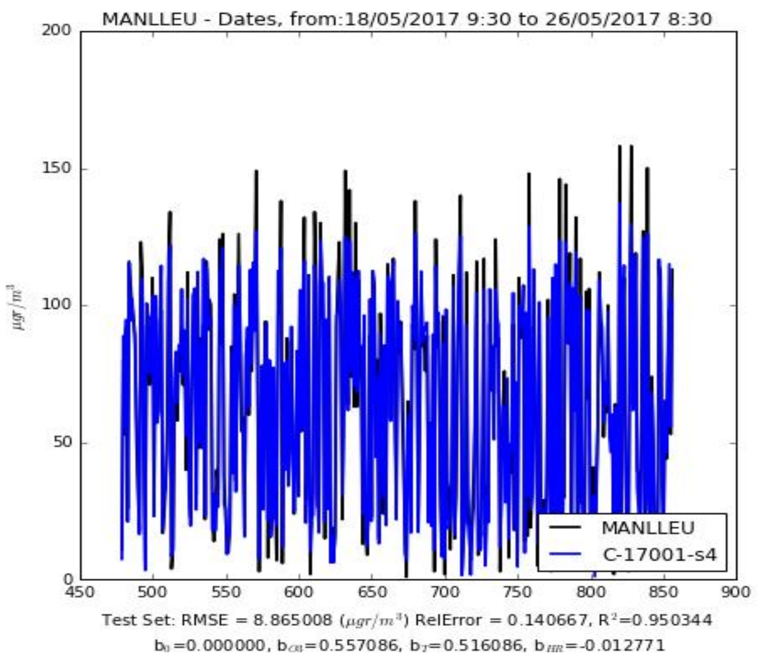

Figure 5. Calibrated Ozone for Validation of sensor S4 Captor 17001.

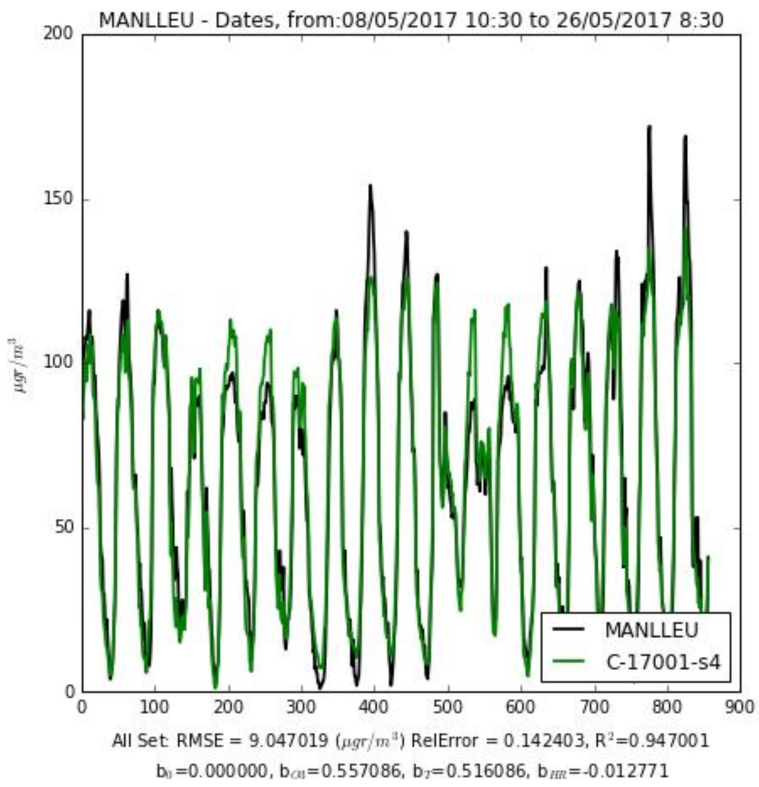

Figure 6. Calibrated Ozone for the whole data set of sensor S4 Captor 17001.

- **File with the RSE of all the sensors:** this file allows us in the case of the captor to choose which of the 5 sensors performs better. In the case of raptor there is only one electrochemical sensor. The file stores the size of the data set, the RSE for the training, the validation and the whole data for a captor node. For example:

```
17001
[857, 479, 378]
[ 14.33349493  12.63236135  10.81323125  10.10156376  16.32232404]
[ 14.41200886  12.80010649  10.99627148  10.10755182  16.36424206]
[ 14.34305967  12.68454743  10.8754736   10.08647158  16.31219212]
```

  where the 1st row identifies the captor node, the 2nd row is the data size and the size of the training and validation set, the 3rd row gives the training RSE, the 4th row gives the validation RSE and the 5th row gives the RSE of the whole set. In this case, we could choose S4 as the best sensor, the one that gives the less RSE among the 5 metal-oxide sensors,

- **Normalized coefficients of the Multivariate Linear Regression (MLR) algorithm**: The regression algorithm works with normalize data. Then the coefficients are calculated to give a normalized calibrated data. This data has to be denormalized using the mean and the standard deviation calculated previously and stored in a file, see below.

For a captor node:

```
# Each row has the betas of each sensor (S1 to S5). columns are offset, beta_O3, beta_HR,
beta_T
0.000000 0.689965 0.426913 0.159453
0.000000 0.739328 0.352376 0.136598
```

```
0.000000 0.724662 0.439091 0.225301
0.000000 0.678176 0.398302 0.148486
0.000000 0.200369 0.662802 -0.046232
```

It is to say, for sensor S1, β_0'=0.0000, β_1'=0.689965, β_2'=0.426913, β_3'=0.159453, and so on.

For a raptor node:

```
# the row has the betas of each sensor. columns are offset, beta_NO2, beta_O3, beta_T,
beta_HR
0.000000 -0.932400 0.771140 -0.250111 -0.050442
```

- **A file with the values of the mean and standard deviation of each sensor.** These data is necessary for denormalize the calibrated sensor data.

For a captor node:

```
1st row have the means of the Ref Station, S1 to S5, T and RH
91.400990 310.376556 232.357651 12.320598 275.475542 1774.450494 25.174752 38.079059
2nd row have the std of the Ref Station, S1 to S5, T and RH
48.014360 141.201602 112.268139 3.328627 135.665164 2609.624402 5.231312 21.591499
```

The values of the RefStat, S4 (the sensor chosen), T and RH are marked in blue. These values are the ones for later predict the ozone. Now, for obtaining the new calibrated data, we have to perform the following operations:

1. Normalize the new data using the vector of means and std stored. Note that we have to normalize, e.g. for captor nodes, the sensor chosen (S4 in the previous example) we normalize the values of the ozone for sensor S4, T and RH.
2. Predict the normalize calibrated ozone using formula (7): $X_{CalNorm} = \beta'_0 + \beta'_1 X_{1new} + \beta'_2 X_{2new} + \beta'_3 X_{3new}$.
3. Denormalize the normalize calibrated value using the RefStat media and std:

$$X_{Cal} = X_{CalNorm} \cdot \sigma_{RefStatO3} + \mu_{RefStatO3} \qquad (10)$$

For a raptor node:

```
1st row have the means of the RefStat_NO2, RefStat_O3, S_NO2, S_O3, T and RH
11.801887 91.266509 2.998042 11.871297 23.410849 57.795755
7.607618 44.845618 2.909707 6.361106 6.874692 21.973817
```

Now, for obtaining the new calibrated data for a raptor node, we have to perform the following operations:

1. Normalize the new data using the vector of means and std stored. Note that we have to normalize the values of the NO_2, O_3, T and RH.
2. Predict the normalize calibrated ozone using formula (7): $X_{cal} = \beta'_0 + \beta'_1 X_{1new} + \beta'_2 X_{2new} + \beta'_3 X_{3new} + \beta'_4 X_{4new}$.
3. Denormalize the normalize calibrated value using the RefStat media and std of the O_3 reference station:

$$X_{Cal} = X_{CalNorm} \cdot \sigma_{RefStatO3} + \mu_{RefStatO3} \qquad (11)$$

4. Final delivery of the software

For the final delivery of the software, it will be delivered via GitHub that allows an open-free account. GitHub is a Web-based Git version control repository hosting service. It is mostly used for computer code. The code will be openly accessed by anyone interested and there will be a link from both the H2020 CAPTOR website and from UPC research group SANS.

Appendix A: Code for calibrating a captor node

The software performs the following operations (in python language).

Main code:

```python
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
#%%
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn.utils import shuffle
import csv
from CaptorLib import CaptorLib
from sklearn import linear_model

##%%
################ SOME CONSTANTS FOR THE WHOLE PROCESS
CAPTOR_SERIES = 17000 # Fill with the CAPTOR number series
TOTAL_SEN = 5  #  Total Number of Ozone sensor devices in a CAPTOR node
nsen=5   #  Number of Ozone sensor devices that we want to regress, from 1 to TOTAL_SEN
p_s=3     # number of features (O3, HR, Temp)
TRAIN_PERCENTAGE = 0.56
THRES_ERROR=15.0

##%%
############# LOAD THE DATA FROM THE .txt INTO THE DATASET list

dir_rd = "/Users/joseb/Dropbox/CAPTOR_Data_2017/Data_CAPTORS/"
dir_wr = "/Users/joseb/Dropbox/CAPTOR_Data_2017/tmp/"

cal=1
n_captors = [1,2,3,4]   ## captors to calibrate
Places=['MANLLEU','MANLLEU','MANLLEU','TONA'] ## Ref Station in which the captors were placed

n_captors = [1]   ## captors to calibrate
Places=['MANLLEU'] ## Ref Station in which the captors were placed

colors = ['red','blue','green','magenta','orange']
results_list = []
badsensors=[["THRES_ERROR",THRES_ERROR]]
mpl.rcParams['figure.figsize'] = 7, 7
captor_utils = CaptorLib()

#%%
counter = -1
for cpt in n_captors:
    counter = counter + 1
    res_list = []
    captor_number = CAPTOR_SERIES+cpt
    res_list.append(captor_number)
    xx="_Cal" + str(cal)
    #### BEGIN Specify the CAPTOR DATA SET name file
    rd_file = dir_rd + "CAP-" + str(captor_number) + "_Cal"+str(cal)+".txt"
    #### END Specify the CAPTOR DATA SET name file
    with open(rd_file) as f:
        ncols = len(f.readline().split(';'))
    print(ncols)

    Dates_DataSet = []
    Places_DataSet = []
    with open(rd_file) as f:
        reader=csv.reader(f,delimiter=';')
        for row in reader:
            Dates_DataSet.append(row[ncols-3])
            Places_DataSet=[Places[counter]]

#################  CARE: doesn't take 'ncols', it takes (ER + nsen + HR + Temp) columns
    DataSetAll = np.genfromtxt(rd_file, delimiter=";",skip_header=1,usecols=range(1,nsen+4))

##############################################################################################
###########
#   GIVE A SHIFT in case the RefData is shifted with respect the UTC time
    HOUR_SHIFT = 0  ### time to shift in hours
    DataSetAll[:,0]=captor_utils.shift(DataSetAll[:,0],HOUR_SHIFT)
##############################################################################################
###########
    DataSet = shuffle(DataSetAll, random_state=0)   # shuffle the data set
##############################################################################################
###########
    ### Normalize the Data Set
    DataSetMean=np.mean(DataSet,axis=0)
    DataSetSd=np.std(DataSet,axis=0)
    NormDataSet=np.zeros(DataSet.shape)
    for i in range(0,DataSet.shape[1]):
        NormDataSet[:,i]=(DataSet[:,i]-DataSetMean[i])/DataSetSd[i]

    ############ DEFINE TRAIN and TEST data SIZES
    TRAIN = int(len(DataSet)*TRAIN_PERCENTAGE)
    TEST=20000   ## a large number

    PR = 0            ## Ref Station
    Temp = TOTAL_SEN +1    ## Temp sensors
    HR = TOTAL_SEN +2      ## RH sensor

    timesSigma=np.sqrt(2) # Marging of confidence = timesSigma*sigma

    #################   NAMES of the FILES

    cap_name=dir_wr+"captor-"+str(captor_number)
    cap_param_s=cap_name+".betas-s" + xx
    cap_NormStats=cap_name+".NormStats" + xx
    cap_trainerrors_s=cap_name+".Train-RelErr-s" + xx

    #### Choose the Training Set and Normalize it

    trainLength=min(TRAIN,len(DataSet))
    lower_limit_training=0
    upper_limit_training=lower_limit_training+trainLength
    trainData=np.copy(DataSet[lower_limit_training:upper_limit_training,])

    # Normalize Training set
    trainMean=np.mean(trainData,axis=0)
    trainSd=np.std(trainData,axis=0)

    trainNorm=np.zeros(trainData.shape)
    for i in range(0,trainData.shape[1]):
        if trainSd[i]!=0.:
            trainNorm[:,i]=(trainData[:,i]-trainMean[i])/trainSd[i]
        else:
            trainNorm[:,i]=trainData[:,i]

    ############  Plot the Data Sensors
    file_wr = dir_wr + "Data-plot" + str(captor_number) + xx + "-all"

captor_utils.PLOT_NORM_DATA_SENSORS_5(DataSet,Places_DataSet[0],Dates_DataSet[0],Dates_DataSet
[len(Dates_DataSet)-1],file_wr)

    ############  Plot the Normalized Data Sensors
    file_wr = dir_wr + "DataNorm-plot" + str(captor_number) + xx + "-all"
```

```
captor_utils.PLOT_NORM_DATA_SENSORS_5(NormDataSet,Places_DataSet[0],Dates_DataSet[0],Dates_DataSet[len(Dates_DataSet)-1],file_wr)

    ########### TRAINING variables
    y_train_OzoneData=trainData[:,PR]
    y_train_OzoneNorm=trainNorm[:,PR]

    train_errorMSE_PD_s=np.zeros(nsen)
    train_RMSE_s=np.zeros(nsen)
    train_R2_s=np.zeros(nsen)
    train_relativeMSE_PD_s=np.zeros(nsen)
    x_train_PD=np.zeros(len(trainNorm))

    meanTrain_PR=trainMean[PR]

    #############    VALIDATION SET
    lower_limit_test=TRAIN
    upper_limit_test=min(len(DataSet),lower_limit_test+TEST)
    testLength=upper_limit_test-lower_limit_test+1

    testData=np.copy(DataSet[lower_limit_test:upper_limit_test,])
    testNorm=np.zeros(testData.shape)
    for i in range(0,testData.shape[1]):
        if trainSd[i]!=0.:
            testNorm[:,i]=(testData[:,i]-trainMean[i])/trainSd[i]
        else:
            testNorm[:,i]=testData[:,i]

    y_test_OzoneData=testData[:,PR]
    meanTest_PR=np.mean(y_test_OzoneData)

    test_errorMSE_PD_s=np.zeros(nsen)
    test_RMSE_s=np.zeros(nsen)
    test_R2_s=np.zeros(nsen)
    test_relativeMSE_PD_s=np.zeros(nsen)

    ########### ALL DATA
    VolNorm=np.zeros(DataSetAll.shape)
    for i in range(0,DataSetAll.shape[1]):       # x.shape gives a vector of dim=2,
[nrows,ncolumns],with the size of the array
        if trainSd[i]!=0.:
            VolNorm[:,i]=(DataSetAll[:,i]-trainMean[i])/trainSd[i]
        else:
            VolNorm[:,i]=DataSetAll[:,i]

    res_list.append([len(DataSet),len(trainNorm),len(testData)])

    tot_errorMSE_PD_s=np.zeros(nsen)
    tot_RMSE_s=np.zeros(nsen)
    tot_R2_s=np.zeros(nsen)
    tot_relativeMSE_PD_s=np.zeros(nsen)

    ######
    betas=np.zeros((nsen,p_s+1))
    data_Norm=dict()
    data_Norm["O_PR"]=trainNorm[:,PR]  # vector with O3_PR
    data_Norm["HR_Captor"]=trainNorm[:,HR]  # vector with HR captor node
    data_Norm["Temp_Captor"]=trainNorm[:,Temp]  # vector with Temp captor node

    for i in range(1,nsen+1):
       captor="O_Captor_"+str(i)
       data_Norm[captor]=trainNorm[:,i]

    ###################    Regression for INDIVIDUAL Sensors

    train_betas_s_list=[]
    for i in range(1,nsen+1):
        Ofit="O_Captor_"+str(i)
        s=i-1

captor_utils.SCATTERPLOT_NORM_DATA_SENSORS_5(data_Norm["O_PR"],data_Norm[Ofit],colors[s],'s'+s
tr(i))
        ############  MLR

X_des=np.matrix([np.ones(len(trainNorm)),data_Norm[Ofit],data_Norm["Temp_Captor"],data_Norm["H
R_Captor"]])
        X_des=np.transpose(X_des)
        Y_des=np.matrix(trainNorm[:,PR])
        Y_des=np.transpose(Y_des)

        regr = linear_model.LinearRegression()
        regr.fit(X_des, Y_des)
        gamma=regr.coef_
        betas[s,0]=gamma[0][0]
        betas[s,1]=gamma[0][1]
        betas[s,2]=gamma[0][2]
        betas[s,3]=gamma[0][3]

        x_train_PD=betas[s,0] + betas[s,1]*data_Norm[Ofit]+
betas[s,2]*data_Norm["Temp_Captor"]+ betas[s,3]*data_Norm["HR_Captor"]
        x_train_PD=x_train_PD*trainSd[PR]+trainMean[PR]

        x_train_PD_tmp = regr.predict(X_des)
        x_train_PD_tmp=x_train_PD_tmp*trainSd[PR]+trainMean[PR]

        train_error_vector_PD=(x_train_PD-y_train_OzoneData)**2
        train_errorMSE_PD_s[s] = np.sum(train_error_vector_PD)
        train_RMSE_s[s]=np.sqrt(train_errorMSE_PD_s[s]/(trainLength-p_s-1))
        train_relativeMSE_PD_s[s]=train_RMSE_s[s]/meanTrain_PR

        train_avg_vector_PD=(trainMean[0]-y_train_OzoneData)**2
        train_avg = np.sum(train_avg_vector_PD)
        train_R2_s[s] = 1.0-train_errorMSE_PD_s[s]/train_avg
        #########################   Writing the trainMean and trainSD to a file
        train_stats=np.matrix([trainMean,trainSd])
        header="First row is trainMean, second row is trainSd"
        np.savetxt(cap_NormStats,train_stats,fmt='%f',header=header)

        #########################   Writing the betas and the relative errors to files
        header="Each row is the betas of each sensor. columns are intercept, beta_O3,
beta_Temp, beta_HR"
        np.savetxt(cap_param_s,betas,fmt='%f',header=header)

        ll= 0
        ul= TRAIN
        text='C-'+str(captor_number)+"-s"+str(i)
        x = np.arange(ll, ul-1, 1)
        plt.plot(x,y_train_OzoneData[ll:ul-1],color='black',linewidth=1.75, linestyle="-
",label=Places_DataSet[0])
        plt.plot(x,x_train_PD_tmp[ll:ul-1],color='red', linewidth=1.75, linestyle="-
",label=text)
        plt.ylabel('$\mu gr/m^3$')
        plt.ylim(0,200)
        plt.xlabel('Train Set: RMSE = %f ($\mu gr/m^3$) RelError = %f, R$^2$=%f \n b$_0$=%f,
b$_{O3}$=%f, b$_{T}$=%f, b$_{HR}$=%f \n\n' %
(train_RMSE_s[s],train_relativeMSE_PD_s[s],train_R2_s[s],betas[s,0],betas[s,1],betas[s,2],beta
s[s,3]))
        plt.legend(loc='lower right')
        plt.title(Places_DataSet[0]+" - Dates, from:"+Dates_DataSet[ll]+" to
"+Dates_DataSet[ul-1])
        file_wr = dir_wr + "Training-C-" + str(captor_number) + xx + "-s" + str(s) + ".png"
        plt.savefig(file_wr,bbox_inches='tight')
        plt.show()

        ###################  MLR Prediction for INDIVIDUAL Sensors
        x_test_PD_s=betas[s,0] + betas[s,1]*testNorm[:,i]+ betas[s,2]*testNorm[:,Temp]+
betas[s,3]*testNorm[:,HR]
```

```
        x_test_PD_s=x_test_PD_s*trainSd[PR]+trainMean[PR]


X_des_val=np.matrix([np.ones(len(testNorm)),testNorm[:,i],testNorm[:,Temp],testNorm[:,HR]])
        X_des_val=np.transpose(X_des_val)
        Y_des_val=np.matrix(testNorm[:,0])
        Y_des_val=np.transpose(Y_des_val)

        x_test_PD_tmp = regr.predict(X_des_val)
        x_test_PD_tmp = x_test_PD_tmp*trainSd[PR]+trainMean[PR]

        test_error_vector_PD=(x_test_PD_s-y_test_OzoneData)**2
        test_errorMSE_PD_s[s] = np.sum(test_error_vector_PD)
        test_RMSE_s[s]=np.sqrt(test_errorMSE_PD_s[s]/(testLength-p_s+1))
        test_relativeMSE_PD_s[s]=test_RMSE_s[s]/meanTest_PR

        test_avg_vector_PD=(trainMean[0]-y_test_OzoneData)**2
        test_avg = np.sum(test_avg_vector_PD)
        test_R2_s[s] = 1.0-test_errorMSE_PD_s[s]/test_avg

        if train_RMSE_s[s] >= THRES_ERROR or test_RMSE_s[s] >= THRES_ERROR:
            badsensors.append([captor_number,i,train_RMSE_s[s],test_RMSE_s[s]])


        ll= TRAIN
        ul=  upper_limit_test
        x = np.arange(ll, ul, 1)
        plt.plot(x,y_test_OzoneData,color='black',linewidth=1.75, linestyle="-
",label=Places_DataSet[0])
        plt.plot(x,x_test_PD_tmp,color='blue', linewidth=1.75, linestyle="-",label=text)
        plt.ylabel('$\mu gr/m^3$')
        plt.xlabel('Test Set: RMSE = %f ($\mu gr/m^3$) RelError = %f, R$^2$=%f \n b$_0$=%f,
b$_{O3}$=%f, b$_{T}$=%f, b$_{HR}$=%f \n\n' %
(test_RMSE_s[s],test_relativeMSE_PD_s[s],test_R2_s[s],betas[s,0],betas[s,1],betas[s,2],betas[s
,3]))
        plt.legend(loc='lower right')
        plt.ylim(0,200)
        plt.title(Places_DataSet[0]+" - Dates, from:"+Dates_DataSet[ll]+" to
"+Dates_DataSet[ul-1])
        file_wr = dir_wr + "Testing-C-" + str(captor_number) + xx  + "-s" + str(s) + ".png"
        plt.savefig(file_wr,bbox_inches='tight')
        plt.show()

        x_Vol_PDNorm = np.zeros(len(DataSetAll))
        x_Vol_PDNorm = betas[s,0] + betas[s,1]*VolNorm[:,i]+ betas[s,2]*VolNorm[:,Temp]+
betas[s,3]*VolNorm[:,HR]
        x_Vol_PD = np.zeros(len(DataSetAll))
        x_Vol_PD = x_Vol_PDNorm*trainSd[PR]+trainMean[PR]
        All_RefST_Mean = np.mean(DataSetAll[:,0])
        tot_error_vector_PD=(x_Vol_PD-DataSetAll[:,0])**2
        tot_errorMSE_PD_s[s] = np.sum(tot_error_vector_PD)
        tot_RMSE_s[s]=np.sqrt(tot_errorMSE_PD_s[s]/(len(DataSetAll)-p_s+1))
        tot_relativeMSE_PD_s[s]=tot_RMSE_s[s]/All_RefST_Mean

        tot_avg_vector_PD=(All_RefST_Mean-DataSetAll[:,0])**2
        tot_avg = np.sum(tot_avg_vector_PD)
        tot_R2_s[s] = 1.0-tot_errorMSE_PD_s[s]/tot_avg

        ll= 0
        ul=  len(DataSetAll)

        x = np.arange(ll, ul, 1)
        plt.plot(x,DataSetAll[:,0],color='black',linewidth=1.75, linestyle="-
",label=Places_DataSet[0])
        plt.plot(x,x_Vol_PD,color='green', linewidth=1.75, linestyle="-",label=text)
        plt.ylabel('$\mu gr/m^3$')
        plt.xlabel('All Set: RMSE = %f ($\mu gr/m^3$) RelError = %f, R$^2$=%f \n b$_0$=%f,
b$_{O3}$=%f, b$_{T}$=%f, b$_{HR}$=%f \n\n' %
(tot_RMSE_s[s],tot_relativeMSE_PD_s[s],tot_R2_s[s],betas[s,0],betas[s,1],betas[s,2],betas[s,3]
))
        plt.legend(loc='lower right')
        plt.ylim(0,200)
        plt.title(Places_DataSet[0]+" - Dates, from:"+Dates_DataSet[ll]+" to
"+Dates_DataSet[ul-1])
        file_wr = dir_wr + "AllData-C-" + str(captor_number) + xx  + "-s" + str(s) + ".png"
        plt.savefig(file_wr,bbox_inches='tight')
        plt.show()

    res_list.append(train_RMSE_s)
    res_list.append(test_RMSE_s)
    res_list.append(tot_RMSE_s)
    results_list.append(res_list)


if len(n_captors)!=1:
    LIST = "_ALL"
else:
    LIST= str(captor_number)


file_w = dir_wr + "Res"+LIST+xx+"-4rth-set.csv"
with open(file_w,"w") as fw:
    writer=csv.writer(fw,delimiter=';')
    for j in range(0,len(results_list)):
        writer.writerow(results_list[j])

file_w = dir_wr + "BadSensors"+LIST+xx+"-4rth-set.csv"
with open(file_w,"w") as fw:
    writer=csv.writer(fw,delimiter=';')
    for j in range(0,len(badsensors)):
        writer.writerow(badsensors[j])
```

The code uses the following library:

```
"""
Created on Wed Nov  8 12:13:28 2017

    Captor Library

@author: pauTE
"""

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


class CaptorLib:

    def PLOT_NORM_DATA_SENSORS_5(self,NormDataSet,place,date_in,date_end,file_wr):
        ll=0
        ul=len(NormDataSet)
        x = np.arange(ll, ul, 1)
        plt.figure(1)
        plt.plot(x,NormDataSet[:ul,0],color='black',linewidth=2.0, linestyle="-",label=place)
        plt.plot(x,NormDataSet[:ul,1],color='red',linewidth=1.0, linestyle="-",label="s1")
        plt.plot(x,NormDataSet[:ul,2],color='blue',linewidth=1.0, linestyle="-",label="s2")
        plt.plot(x,NormDataSet[:ul,3],color='magenta',linewidth=1.0, linestyle="-
",label="s3")
        plt.plot(x,NormDataSet[:ul,4],color='green',linewidth=1.0, linestyle="-",label="s4")
        plt.plot(x,NormDataSet[:ul,5],color='orange',linewidth=1.0, linestyle="-",label="s5")
        plt.ylabel('O3')
        text = 'NORMALIZED DataSet'
        plt.legend(loc='upper left')
        plt.xlabel(text)
        plt.title(place+" - Dates, from:"+date_in+" to "+date_end)
        wr_file = file_wr+'.png'
        plt.savefig(wr_file)
        plt.show()

    def PLOT_DATA_SENSORS_5(self,DataSet,place,date_in,date_end,file_wr):
        ll=0
        ul=len(DataSet)
        x = np.arange(ll, ul, 1)
        plt.figure(2)
        plt.plot(x,DataSet[:ul,0],color='black',linewidth=2.0, linestyle="-",label=place)
        plt.plot(x,DataSet[:ul,1],color='red',linewidth=1.0, linestyle="-",label="s1")
        plt.plot(x,DataSet[:ul,2],color='blue',linewidth=1.0, linestyle="-",label="s2")
        plt.plot(x,DataSet[:ul,3],color='magenta',linewidth=1.0, linestyle="-",label="s3")
        plt.plot(x,DataSet[:ul,4],color='green',linewidth=1.0, linestyle="-",label="s4")
        plt.plot(x,DataSet[:ul,5],color='orange',linewidth=1.0, linestyle="-",label="s5")
        plt.ylabel('O3')
        text = 'DataSet'
        plt.legend(loc='upper left')
        plt.xlabel(text)
        plt.title(place+" - Dates, from:"+date_in+" to "+date_end)
        wr_file = file_wr+'.png'
        plt.savefig(wr_file)
        plt.show()

    def SCATTERPLOT_NORM_DATA_SENSORS_5(self,DataS_RS,DataS_Sn,color,sensor):
        plt.figure(3)
        plt.scatter(DataS_RS,DataS_Sn,color=color,label=sensor)
        plt.legend(loc='upper left')
        plt.show()


    def removeBadObservations(self,dataset):
        '''
        This function removes rows from the datset having -1 on a sensor
        '''

        aux = dataset.copy()
        aux = dataset[dataset[:,1]!=-1.0,]
        aux = aux[aux[:,2]!=-1.0,]
        aux = aux[aux[:,3]!=-1.0,]
        aux = aux[aux[:,4]!=-1.0,]
        aux = aux[aux[:,5]!=-1.0,]

        return aux

    def selectLocation(self,dataset,refs,place):
        '''
        This function select the captor data from a palce and removes bad obsevrations
        (i.e. data from PR, then place ='PR' and refs is all the labels)
        '''
        indexes = np.where(refs==place)[0]
        dataset = dataset[indexes,:]
        dataset = self.removeBadObservations(dataset)

        return dataset

    def shift(self,a,step):
        ee=[0.0]*len(a)
        for t in range(len(a)):
            ee[t]=a[t]
        if step>0:
            for t in range(step,len(a)):
                ee[t]=a[t-step]
        if step<0:
            for t in range(len(a)+step):
                ee[t]=a[t-step]
        return ee


    def adjustedRsquared(self,rsquared,n,k):
        '''
        This function computes the adjusted rsquared given the number
        of observations and the numbe rof features used in the fit
        '''

        intermidiate = (float(k-1.0)/float(n-k))*(1.0-rsquared)
        return  rsquared - intermidiate

    def plot3D(self,dataset,o3_sensor = 1):
        '''
        This functions plots a 3D plot of an ozone sensor against
        the HR and Temp
        '''

        fig = plt.figure()
        ax = fig.add_subplot(111, projection='3d')
        ax.scatter(dataset[:,o3_sensor],dataset[:,5],dataset[:,6])
        ax.set_xlabel('O3')
        ax.set_ylabel('Temp')
        ax.set_zlabel('HR')
        plt.show()
```

Appendix B: Code for calibrating a raptor node

The software performs the following operations (in python language):

```python
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
#%%
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from sklearn import datasets, linear_model
import csv
from sklearn.utils import shuffle

################ SOME CONSTANTS FOR THE WHOLE PROCESS
CAPTOR_SERIES = 0 # Fill with the CAPTOR number,
TOTAL_SEN = 2
nsen=2   #  Number of sensor devices
p_s=4     # number of features (O3, HR, Temp)
TRAIN_PERCENTAGE = 0.65
THRES_ERROR=15.0

############# LOAD THE DATA FROM THE .txt INTO THE DATASET list

dir_rd = "/home/CAPTOR_Data_2017/Data_RAPTORS/"
dir_wr = "/home/CAPTOR_Data_2017/Raptors-Cal/"

data_meaning = "End_local;ref_NO2;ref_O3;raw_no2;raw_o3;avg_temp;avg_humi"

LIST= "-R70-Tona"
LIST1 = '_Tona'
Cal = 2
n_raptors = [70]
Places_DataSet = ['TONA']

results_list = []
results_list_resta = []
badsensors=[["THRES_ERROR",THRES_ERROR]]
mpl.rcParams['figure.figsize'] = 7, 7

def shift(a,step):
    ee=[0.0]*len(a)
    for t in range(len(a)):
        ee[t]=a[t]
    if step>0:
        for t in range(step,len(a)):
            ee[t]=a[t-step]
    if step<0:
        for t in range(len(a)+step):
            ee[t]=a[t-step]
    return ee

def PLOT_NORM_DATA_SENSORS_5(NormDataSet,place,date_in,date_end,file_wr,sd,polutant):
    ll=0                  ## Lower Limit
    ul=len(NormDataSet)       ## upper Limit
    x = np.arange(ll, ul, 1)
    plt.plot(x,NormDataSet[:ul,0],color='black',linewidth=2.0, linestyle="-",label=place)
    plt.plot(x,NormDataSet[:ul,1],color='red',linewidth=1.0, linestyle="-",label="s1")
    plt.ylabel(polutant)
    text = 'DataSet - SD: ' + str(sd[1:])
    plt.legend(loc='upper left')
    plt.xlabel(text)
    plt.savefig(file_wr,bbox_inches='tight')
    plt.title(place+" - Dates, from:"+date_in+" to "+date_end)
    plt.show()

def SCATTERPLOT_NORM_DATA_SENSORS_2(NormDataSet):
    plt.scatter(NormDataSet[:,0],NormDataSet[:,1],color='red',label="NO2")
    plt.legend(loc='upper left')
    plt.show()
    plt.scatter(NormDataSet[:,1],NormDataSet[:,3],color='blue',label="O3")
    plt.legend(loc='upper left')
    plt.show()

#%%
for cpt in n_raptors:
    res_list = []
    res_list_resta = []
    captor_number = CAPTOR_SERIES+cpt
    res_list.append(captor_number)
    print 'RAPTOR',captor_number
    if captor_number < 100:
        rd_file = dir_rd + "RAP-0" + str(captor_number) + LIST1 + "Cal"+str(Cal)+".txt"
    else:
        rd_file = dir_rd + "RAP-0" + str(captor_number) + LIST1 + "Cal"+str(Cal)+".txt"
    with open(rd_file) as f:
        ncols = len(f.readline().split(';'))
    print(ncols)

    Dates_DataSet = []
    rownumber = 0
    with open(rd_file) as f:
        reader=csv.reader(f,delimiter=';')
        for row in reader:
            if rownumber >0:
                Dates_DataSet.append(row[0])
            else:
                rownumber = rownumber +1

    DataSetAll = np.genfromtxt(rd_file, delimiter=";",skip_header=1,usecols=range(1,7))
    DataSetAll = DataSetAll[~np.isnan(DataSetAll).any(axis=1)]
    DataSet = shuffle(DataSetAll, random_state=0)

    DataSetMean=np.mean(DataSet,axis=0)
    DataSetSd=np.std(DataSet,axis=0)
    NormDataSet=np.zeros(DataSet.shape)
    for i in range(0,DataSet.shape[1]):
        NormDataSet[:,i]=(DataSet[:,i]-DataSetMean[i])/DataSetSd[i]

    ############ DEFINE TRAIN and TEST data SIZES
    TRAIN = int(len(DataSet)*TRAIN_PERCENTAGE)
    TEST=20000

    PR_NO2 = 0
    PR_O3 = 1
    polutant = ['NO2','O3']
    PR = [PR_NO2, PR_O3]
    NO2 = 2
    O3 = 3
    Temp = 4
    HR = 5
    NO2_O3 = 6

    timesSigma=np.sqrt(2) # Marging of confidence = timesSigma*sigma

    #################   NAMES of the FILES

    cap_name=dir_wr+"raptor-"+ str(captor_number) + LIST1
    cap_param_s=cap_name+".betas-s" + '_Cal' + str(Cal)
    cap_param_s_resta=cap_name+".betas-s-Resta" + '_Cal' + str(Cal)
    cap_NormStats=cap_name+".NormStats" + '_Cal' + str(Cal)
    cap_trainerrors_s=cap_name+".Train-RelErr-s" + '_Cal' + str(Cal)
    cap_trainerrors_s_resta=cap_name+".Train-RelErr-s_Resta" + '_Cal' + str(Cal)

    #### Choose the Training Set and Normalize it

    trainLength=min(TRAIN,len(DataSet))
    lower_limit_training=0
    upper_limit_training=lower_limit_training+trainLength
trainData=np.copy(DataSet[lower_limit_training:upper_limit_training,])

    # Normalize Training set
    trainMean=np.mean(trainData,axis=0)
    trainSd=np.std(trainData,axis=0)

    trainNorm=np.zeros(trainData.shape)
    for i in range(0,trainData.shape[1]):              if trainSd[i]!=0.:
            trainNorm[:,i]=(trainData[:,i]-trainMean[i])/trainSd[i]
        else:
            trainNorm[:,i]=trainData[:,i]

    file_wr = dir_wr + "DataNorm-plot" + str(captor_number) + "-" + polutant[0] + LIST1 +
'_Cal' +str(Cal) +  ".png"
    ll=0                   ## Lower Limit
    ul=len(NormDataSet)        ## upper Limit
    x = np.arange(ll, ul, 1)
    plt.plot(x,DataSetAll[:ul,0],color='black',linewidth=2.0, linestyle="-
",label=Places_DataSet[0])
    plt.plot(x,DataSetAll[:ul,2],color='green',linewidth=1.0, linestyle="-",label="s1")
    plt.ylabel(polutant[0])
    text = 'DataSet - SD: ' + str(trainSd[1:])
    plt.legend(loc='upper left')
    plt.xlabel(text)
    plt.savefig(file_wr,bbox_inches='tight')
    plt.title(Places_DataSet[0]+" - Dates, from:"+Dates_DataSet[0]+" to
"+Dates_DataSet[len(Dates_DataSet)-1])
    plt.show()

    file_wr = dir_wr + "DataNorm-plot" + str(captor_number) + "-" + polutant[1] + LIST1 +
'_Cal' +str(Cal) + ".png"
    ll=0                   ## Lower Limit
    ul=len(NormDataSet)        ## upper Limit
    x = np.arange(ll, ul, 1)
    plt.plot(x,DataSetAll[:ul,1],color='black',linewidth=2.0, linestyle="-
",label=Places_DataSet[0])
    plt.plot(x,DataSetAll[:ul,3],color='orange',linewidth=1.0, linestyle="-",label="s1")
    plt.ylabel(polutant[1])
    text = 'DataSet - SD: ' + str(trainSd[1:])
    plt.legend(loc='upper left')
    plt.xlabel(text)
    plt.savefig(file_wr,bbox_inches='tight')
    plt.title(Places_DataSet[0]+" - Dates, from:"+Dates_DataSet[0]+" to
"+Dates_DataSet[len(Dates_DataSet)-1])
    plt.show()

    ############  SCATTER-PLOTs of the Normalized Data Sensors
    SCATTERPLOT_NORM_DATA_SENSORS_2(NormDataSet)

    ########### TRAINING variables
    train_errorMSE_PD_s=np.zeros(nsen)
    train_RMSE_s=np.zeros(nsen)
    train_R2_s=np.zeros(nsen)
    train_relativeMSE_PD_s=np.zeros(nsen)
    x_train_PD=np.zeros(len(trainNorm))

    #############    VALIDATION SET
    lower_limit_test=TRAIN
    upper_limit_test=min(len(DataSet),lower_limit_test+TEST)
    testLength=upper_limit_test-lower_limit_test+1

    testData=np.copy(DataSet[lower_limit_test:upper_limit_test,])
    testNorm=np.zeros(testData.shape)
    for i in range(0,testData.shape[1]):
        if trainSd[i]!=0.:
            testNorm[:,i]=(testData[:,i]-trainMean[i])/trainSd[i]
        else:
            testNorm[:,i]=testData[:,i]

    test_errorMSE_PD_s=np.zeros(nsen)
    test_RMSE_s=np.zeros(nsen)
    test_R2_s=np.zeros(nsen)
    test_relativeMSE_PD_s=np.zeros(nsen)

    test_errorMSE_PD_s_tmp=np.zeros(nsen)
    test_RMSE_s_tmp=np.zeros(nsen)
    test_R2_s_tmp=np.zeros(nsen)
    test_relativeMSE_PD_s_tmp=np.zeros(nsen)

    ########### ALL DATA
    VolNorm=np.zeros(DataSetAll.shape)
    for i in range(0,DataSetAll.shape[1]):
        if trainSd[i]!=0.:
            VolNorm[:,i]=(DataSetAll[:,i]-trainMean[i])/trainSd[i]
        else:
            VolNorm[:,i]=DataSetAll[:,i]

    tot_errorMSE_PD_s=np.zeros(nsen)
    tot_RMSE_s=np.zeros(nsen)
    tot_R2_s=np.zeros(nsen)
    tot_relativeMSE_PD_s=np.zeros(nsen)

    res_list.append([len(DataSet),len(trainNorm),len(testData)])
    res_list_resta.append([len(DataSet),len(trainNorm),len(testData)])

    #### organize the data in a dictionary and regress NO2 and O3
    betas=np.zeros((nsen,5))
    data_Norm=dict()
    data_Norm["O3_PR"]=trainNorm[:,PR_O3]  # vector with O3_PR
    data_Norm["NO2_PR"]=trainNorm[:,PR_NO2]  # vector with O3_PR
    data_Norm["HR_Raptor"]=trainNorm[:,HR]  # vector with HR captor node
    data_Norm["Temp_Raptor"]=trainNorm[:,Temp]  # vector with Temp captor node
    data_Norm["O3_raw"]=trainNorm[:,O3]  # vector with O3 raw captor node
    data_Norm["NO2_raw"]=trainNorm[:,NO2]  # vector with NO2_raw captor node

    Ofit = ["NO2_raw","O3_raw"]
    train_betas_s_list=[]
    for i in range(0,nsen):
        y_train_OzoneData=trainData[:,PR[i]]  # vector with O3_PR
        y_train_OzoneNorm=trainNorm[:,PR[i]]  # vector with O3_PR
        y_test_OzoneData=testData[:,PR[i]]  # vector with O3_PR
        meanTest_PR=np.mean(y_test_OzoneData)

X_des=np.matrix([np.ones(len(trainNorm)),trainNorm[:,NO2],trainNorm[:,O3],trainNorm[:,Temp],tr
ainNorm[:,HR]])
        X_des=np.transpose(X_des)
        Y_des=np.matrix(trainNorm[:,PR[i]])
        Y_des=np.transpose(Y_des)

        regr = linear_model.LinearRegression()
        regr.fit(X_des, Y_des)
        gamma=regr.coef_
        betas[i,0]=gamma[0][0]  ## offset
        betas[i,1]=gamma[0][1]  ## NO2
        betas[i,2]=gamma[0][2]  ## O3
        betas[i,3]=gamma[0][3]  ## Temp
        betas[i,4]=gamma[0][4]  ## HR

        x_train_PD=betas[i,0] + betas[i,1]*data_Norm[Ofit[0]]+ betas[i,2]*data_Norm[Ofit[1]]+
betas[i,3]*data_Norm["Temp_Raptor"]+ betas[i,4]*data_Norm["HR_Raptor"]
        x_train_PD=x_train_PD*trainSd[PR[i]]+trainMean[PR[i]]
        x_train_PD_tmp = regr.predict(X_des)
        x_train_PD_tmp=x_train_PD_tmp*trainSd[PR[i]]+trainMean[PR[i]]

        train_error_vector_PD=(x_train_PD-y_train_OzoneData)**2
        train_errorMSE_PD_s[i] = np.sum(train_error_vector_PD)
        train_RMSE_s[i]=np.sqrt(train_errorMSE_PD_s[i]/(trainLength-p_s+1))
        train_relativeMSE_PD_s[i]=train_RMSE_s[i]/trainMean[i]

        train_avg_vector_PD=(trainMean[0]-y_train_OzoneData)**2
        train_avg = np.sum(train_avg_vector_PD)
        train_R2_s[i] = 1.0-train_errorMSE_PD_s[i]/train_avg
        #########################   Writing the trainMean and trainSD to a file
        train_stats=np.matrix([trainMean,trainSd])
        header="First row is trainMean, second row is trainSd"
        np.savetxt(cap_NormStats,train_stats,fmt='%f',header=header)

        #########################   Writing the betas and the relative errors to files
        header="Each row is the betas of each sensor. columns are intercept, beta_O3, beta_HR,
beta_CT"
        np.savetxt(cap_param_s,betas,fmt='%f',header=header)

        ll= 0             ## Lower Limit
        ul= TRAIN     ## Upper Limit
        wr_file = dir_wr + "Training-R-" + str(captor_number) + "-s" + polutant[i] + LIST1 +
'_Cal' +str(Cal) + ".png"

        text='R-'+str(captor_number)+"-s"+str(i)
        x = np.arange(ll, ul-1, 1)
        plt.plot(x,y_train_OzoneData[ll:ul-1],color='black',linewidth=1.75, linestyle="-
",label=Places_DataSet[0])
        plt.plot(x,x_train_PD_tmp[ll:ul-1],color='red', linewidth=1.75, linestyle="-
",label=text)
        plt.ylabel(polutant[i])
        if i==0:
            plt.ylim(0,80)
        else:
            plt.ylim(0,240)
        plt.xlabel('Train Set: RMSE = %f ($\mu gr/m^3$) RelError = %f, R$^2$=%f \n b$_0$=%f,
b$_{NO2}$=%f, b$_{O3}$=%f, b$_{T}$=%f, b$_{HR}$=%f \n\n' %
(train_RMSE_s[i],train_relativeMSE_PD_s[i],train_R2_s[i],betas[i,0],betas[i,1],betas[i,2],beta
s[i,3],betas[i,4]))
        plt.legend(loc='lower right')
        plt.savefig(wr_file,bbox_inches='tight')
        plt.title(Places_DataSet[0]+" - Dates, from:"+Dates_DataSet[ll]+" to
"+Dates_DataSet[ul-1])
        plt.show()

  ##############  TEST  DATA
  ###################  MLR Prediction for INDIVIDUAL Sensors
        x_test_PD_s=betas[i,0] + betas[i,1]*testNorm[:,NO2] + betas[i,2]*testNorm[:,O3]+
betas[i,3]*testNorm[:,Temp]+ betas[i,4]*testNorm[:,HR]
        x_test_PD_s=x_test_PD_s*trainSd[PR[i]]+trainMean[PR[i]]


X_des_val=np.matrix([np.ones(len(testNorm)),testNorm[:,PR[0]],testNorm[:,PR[1]],testNorm[:,Tem
p],testNorm[:,HR]])
        X_des_val=np.transpose(X_des_val)
        Y_des_val=np.matrix(testData[:,PR[i]])
        Y_des_val=np.transpose(Y_des_val)

        x_test_PD_tmp = regr.predict(X_des_val)
        x_test_PD_tmp = x_test_PD_tmp*trainSd[PR[i]]+trainMean[i]

        test_error_vector_PD=(x_test_PD_s-y_test_OzoneData)**2
        test_errorMSE_PD_s[i] = np.sum(test_error_vector_PD)
        test_RMSE_s[i]=np.sqrt(test_errorMSE_PD_s[i]/(testLength-p_s+1))
        test_relativeMSE_PD_s[i]=test_RMSE_s[i]/meanTest_PR
        test_avg_vector_PD=(trainMean[0]-y_test_OzoneData)**2
        test_avg = np.sum(test_avg_vector_PD)
        test_R2_s[i] = 1.0-test_errorMSE_PD_s[i]/test_avg

        test_error_vector_PD_tmp=(x_test_PD_tmp[0]-y_test_OzoneData)**2
        test_errorMSE_PD_s_tmp[i] = np.sum(test_error_vector_PD_tmp)
        test_RMSE_s_tmp[i]=np.sqrt(test_errorMSE_PD_s_tmp[i]/(testLength-p_s+1))
        test_relativeMSE_PD_s_tmp[i]=test_RMSE_s_tmp[i]/meanTest_PR

        test_avg_vector_PD=(trainMean[0]-y_test_OzoneData)**2
        test_avg = np.sum(test_avg_vector_PD)
        test_R2_s[i] = 1.0-test_errorMSE_PD_s[i]/test_avg

        if train_RMSE_s[i] >= THRES_ERROR or test_RMSE_s[i] >= THRES_ERROR:
            badsensors.append([captor_number,i,train_RMSE_s[i],test_RMSE_s[i]])


        ll= TRAIN             ## Lower Limit
        ul=  upper_limit_test   ## Upper Limit

        wr_file = dir_wr + "Testing-R-" + str(captor_number) + "-s" + polutant[i] + LIST1 +
'_Cal' +str(Cal) +".png"
        x = np.arange(ll, ul, 1)
        plt.plot(x,y_test_OzoneData,color='black',linewidth=1.75, linestyle="-
",label=Places_DataSet[0])
        plt.plot(x,x_test_PD_s,color='blue', linewidth=1.75, linestyle="-",label=text)
        plt.ylabel(polutant[i])
        plt.xlabel('Test Set: RMSE = %f ($\mu gr/m^3$) RelError = %f, R$^2$=%f \n b$_0$=%f,
b$_{NO2}$=%f, b$_{O3}$=%f, b$_{T}$=%f, b$_{HR}$=%f \n\n' %
(test_RMSE_s[i],test_relativeMSE_PD_s[i],test_R2_s[i],betas[i,0],betas[i,1],betas[i,2],betas[i
,3],betas[i,4]))
        plt.legend(loc='lower right')
        if i==0:
            plt.ylim(0,80)
        else:
            plt.ylim(-40,240)
        plt.savefig(wr_file,bbox_inches='tight')
        plt.title(Places_DataSet[0]+" - Dates, from:"+Dates_DataSet[ll]+" to
"+Dates_DataSet[ul-1])
        plt.show()

##############  ALL DATA

        x_Vol_PDNorm = np.zeros(len(DataSetAll))
        x_Vol_PDNorm=betas[i,0] + betas[i,1]*VolNorm[:,NO2] + betas[i,2]*VolNorm[:,O3]+
betas[i,3]*VolNorm[:,Temp]+ betas[i,4]*VolNorm[:,HR]
        x_Vol_PD = np.zeros(len(DataSetAll))
        x_Vol_PD = x_Vol_PDNorm*trainSd[PR[i]]+trainMean[PR[i]]

        All_RefST_Mean = np.mean(DataSetAll[:,PR[i]])
        tot_error_vector_PD=(x_Vol_PD-DataSetAll[:,PR[i]])**2
        tot_errorMSE_PD_s[i] = np.sum(tot_error_vector_PD)
        tot_RMSE_s[i]=np.sqrt(tot_errorMSE_PD_s[i]/(len(DataSetAll)-p_s+1))
        tot_relativeMSE_PD_s[i]=tot_RMSE_s[i]/All_RefST_Mean

        tot_avg_vector_PD=(All_RefST_Mean-DataSetAll[:,0])**2
        tot_avg = np.sum(tot_avg_vector_PD)
        tot_R2_s[i] = 1.0-tot_errorMSE_PD_s[i]/tot_avg


        ll= 0              ## Lower Limit
        ul=  len(DataSetAll)   ## Upper Limit
        file_wr = dir_wr + "AllData-R-" + str(captor_number) + "-s" + polutant[i] + LIST1 +
'_Cal' +str(Cal) + ".png"
        x = np.arange(ll, ul, 1)
        plt.plot(x,DataSetAll[:,PR[i]],color='black',linewidth=1.75, linestyle="-
",label=Places_DataSet[0])
#        plt.plot(x,x_test_PD_s,color='red', linewidth=1.75, linestyle="-",label=text)
        plt.plot(x,x_Vol_PD,color='green', linewidth=1.75, linestyle="-",label=text)
        plt.ylabel(polutant[i])
        plt.xlabel('All Set: RMSE = %f ($\mu gr/m^3$) RelError = %f, R$^2$=%f \n b$_0$=%f,
b$_{NO2}$=%f, b$_{O3}$=%f, b$_{T}$=%f, b$_{HR}$=%f \n\n' %
(tot_RMSE_s[i],tot_relativeMSE_PD_s[i],tot_R2_s[i],betas[i,0],betas[i,1],betas[i,2],betas[i,3]
,betas[i,4]))
        plt.legend(loc='lower right')
        if i==0:
            plt.ylim(0,80)
        else:
            plt.ylim(0,240)
        plt.title(Places_DataSet[0]+" - Dates, from:"+Dates_DataSet[ll]+" to
"+Dates_DataSet[ul-1])
        plt.savefig(file_wr,bbox_inches='tight')
        plt.show()

    res_list.append(train_RMSE_s)
    res_list.append(test_RMSE_s)
    res_list.append(tot_RMSE_s)
    results_list.append(res_list)


file_w = dir_wr + "Res"+ LIST +'_Cal' +str(Cal) +".csv"
with open(file_w,"w") as fw:
    writer=csv.writer(fw,delimiter=';')
    for j in range(0,len(results_list)):
        writer.writerow(results_list[j])


file_w = dir_wr + "BadSensors"+ LIST +'_Cal' +str(Cal) + ".csv"
with open(file_w,"w") as fw:
    writer=csv.writer(fw,delimiter=';')
    for j in range(0,len(badsensors)):
        writer.writerow(badsensors[j])
```