

AirU

Machine Learning Scripts Manual

Applications to Ozone

Timothy Quah
8-19-2019

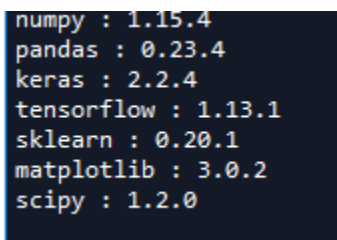
Contents

Introduction.....	3
Python Installation	4
Preprocessing Scripts and Functions.....	4
Tutorial.....	4
Cleaner_Loader_Functions List.....	9
indexall.....	9
find_in_list	10
sub_main_merge	10
DAQ_DATA_Filter	11
Main_Sep_Load.....	11
DAQ_Parser_Separator.....	12
Organize_Clean_DAQ.....	13
AIR_U_Sensor_Sep.....	13
Matchin_DAQ_AIR_U_time.....	14
null_code_DAQ_filter	15
Add_Missing_Data	15
Organize_Clean_All	16
Reorder_df	16
Reorder_df	17
Normalizer Functions List	18
extract_date	18
replace_header_id	18
Combine_All_Data	19
clean_dataframe	20
normalization	20
export_normalization	21
Training Machine Learning Algorithms	22
Introduction:.....	22
Background:.....	22
Methodology:.....	23
Tutorial:.....	24
Trainer Function List	26
model_neural_network	27
norm_divider.....	27

divider_XY	28
load_evaluate_neural_net.....	29
mse	29
R2.....	30
Analyzer Function List	31
export_graphs.....	31
Evaluating Neural Networks	33
References.....	34

Introduction

The following document is a manual to use the scripts that were developed in the summer of 2019 to predict Ozone using parameters found from sensors in the AirU network and from the Department of Air Quality (DAQ). The code base includes scripts/functions that first can preprocess the data prior to applying machine learning methods such as cleaning, merging, organizing, normalizing and splitting the dataset. Included is also scripts that can help aid in training, validating, and measuring the performance of neural networks. The machine learning method used in this code base is artificial neural networks (ANNs) and is based around Keras with a backend engine of TensorFlow. The individual sections will go through a tutorial along with some detail on the backend of custom made functions.



```
numpy : 1.15.4
pandas : 0.23.4
keras : 2.2.4
tensorflow : 1.13.1
sklearn : 0.20.1
matplotlib : 3.0.2
scipy : 1.2.0
```

Figure 1: Package List used in code base from Tim's Computer. Note that these do not have to be identical in order to run scripts, but if the code is performing differently then one should crosscheck with this list.

The main standard packages used in preprocessing scripts are Numpy, Sklearn, and Pandas. Numpy is a standard package for storing and manipulating data arrays. Sklearn is a standard package that is popular for machine learning, but other packages have since come out that have replaced it. However, Sklearn still has normalization packages used to normalize the data. Pandas is the main workhorse of preprocessing another packaged that is used to 1) load and export .csv files 2) data frame datatype 3) combining datasets.

The main standard packages used in the training are TensorFlow and Keras. For some of the optimization scripts Scipy is used. TensorFlow is a package created by google for deep learning and keras is a package that makes TensorFlow easier to use. These packages do not necessarily have to be installed on your computer, as the majority of trainings of neural networks and evaluation can be done elsewhere. However, I would recommend that one installs these packages. Another note is this project utilizes computational resources from Google Colab (Jupyter Notebook with GPU and TPU resources) and Kaggle (Scripting or Jupyter Notebook with GPU resources).

This code base is currently stored on a Github repository and access to the repository can be granted by Professor Kerry Kelly or Professor Butterfield.

Python Installation

If you are new to Python and are deciding which python to install among other details I would recommend going to Anaconda (<https://www.anaconda.com/distribution/>) and following the instructions there. Reiterating what was stated in the introduction numpy, pandas and sklearn are the packages needed for preprocessing, but are included in the Anaconda installation of python. For training and evaluating neural networks packages that are need are Keras and TensorFlow which can both be installed following instructions given in this article <https://inmachineswetrust.com/posts/deep-learning-setup/>

The article suggests:

- pip install tensorflow
 - **You can do this instead**
 - conda install tensorflow
- pip install keras
 - **You can do this instead**
 - conda install keras

Preprocessing Scripts and Functions

Preprocessing is the step that is done prior to training a neural network. This step is essential to ensure that neural networks are trained on acceptable data and while this step could be done by hand it is much easier to create scripts to do this for us.

Tutorial

Step 1:

Download or obtain the datasets from DAQ and AirU.

Step 2:

AirU dataset is stuck in a bunch of directories and subdirectories to get it all into one directory. The script that is used to accomplish this is in the script folder and is called: “**AirU_SingleDirectory_Script**”. All one needs to do is put the directory where the AirU data (Import) and where the script should dump all the files (Export). These two variables are highlighted in Figure 2 (shown on next page).

```

7 ## Import programs
8 import os #used to for operating system functionalities
9
10 ## Gets program to look in the directory
11 function_path = "../Functions" #path to Tim's function
12
13 ## Change directory to function directory allows script to use functions
14 os.chdir(function_path)
15
16 from Cleaner_Loader_Functions import sub_main_merge
17 import_path_AIR_U = \
18 r"D:\AirQuality_Research\Data\AirU_Data"
19 export_path_Air_U = \
20 r"D:\AirQuality_Research\Data\Full Year"
21 sub_main_merge(import_path_AIR_U,export_path_Air_U)
22

```

Figure 2: Snippet of AirU_SingleDirectory_Script-Red lines show the two places that should be altered for the script to work.

The result should look like the figure below:

Hawthorne	6/18/2019 4:57 PM	File folder	➔	Rose Park_Temperature	6/11/2019 7:18 PM	Microsoft Excel C...	570 KB
Rose Park	6/18/2019 4:57 PM	File folder		Rose Park_PM10	6/11/2019 7:16 PM	Microsoft Excel C...	567 KB
				Rose Park_PM2_5	6/11/2019 7:17 PM	Microsoft Excel C...	554 KB
				Rose Park_PM1	6/11/2019 7:15 PM	Microsoft Excel C...	529 KB
				Rose Park_NO	6/11/2019 7:17 PM	Microsoft Excel C...	538 KB
				Rose Park_MICS	6/11/2019 7:22 PM	Microsoft Excel C...	272 KB
				Rose Park_Humidity	6/11/2019 7:14 PM	Microsoft Excel C...	561 KB
				Rose Park_CO	6/11/2019 7:15 PM	Microsoft Excel C...	535 KB
				Hawthorne_Temperature	6/11/2019 7:13 PM	Microsoft Excel C...	573 KB
				Hawthorne_PM10	6/11/2019 7:11 PM	Microsoft Excel C...	569 KB
				Hawthorne_PM2_5	6/11/2019 7:12 PM	Microsoft Excel C...	553 KB
				Hawthorne_PM1	6/11/2019 7:10 PM	Microsoft Excel C...	526 KB
				Hawthorne_NO	6/11/2019 7:12 PM	Microsoft Excel C...	529 KB
				Hawthorne_MICS	6/11/2019 7:22 PM	Microsoft Excel C...	268 KB
				Hawthorne_Humidity	6/11/2019 7:09 PM	Microsoft Excel C...	565 KB
				Hawthorne_CO	6/11/2019 7:09 PM	Microsoft Excel C...	527 KB

Figure 3-Showing result of using AirU_SingleDirectory_Script

Step 3:

The next step is to use Data_Loading_Script if you want to use this script straight from the box you must meet the following criteria:

- 1) the DAQ dataset must have DAQ in the name of the file
- 2) No file can have a period ie. (2.5PM.csv is not ok but 2_5PM.csv is ok)
- 3) AirU and DAQ dataset are similar to the example files
 - Can have more or less data, but the format in which they are given should be similar.
- 4) Rose Park is missing SR data,
- 5) Agree that sensors with repetitive MAC addresses should become the other dataset (if they exist)

If you meet these criteria then running this section of the script is super simple and only need to specify the import path to the directory with all the files (both AirU and DAQ) as well as the desired export path where the .csv files will be dumped. In addition, one must specify the location name and the abbreviations that DAQ provides. This script does the following 1) loads the data into Python , 2) separates AirU data by sensors, 3) cleans DAQ and AirU datasets (also converts AirU to MST), 5) adds missing SR column to a DAQ dataset, 4) aligns and combines AirU and DAQ datasets, 5) reorders columns to be consistent, and 6) exports cleaned datasets as .csv files.

```
## Import all raw data this is a directory with all of the data both DAQ and AirU
import_path = \
r"D:\AirQuality_Research\Data\Full Year"
## Export the cleaned and organized csvs from this script
export_path = r"D:\AirQuality_Research\Data\Output_Clean"

## Locations Usually Hawthorne and RosePark
## Must put location with solar radiation first
Location_name = ['Hawthorne', 'Rose']
Location_abv = ['HW', 'RP']
```

Figure 4: Snippet of Data_Loading_Script showing the location of code that must be altered to use the code

The output should show csv files in the export directory with a name that includes both the location name with the sensor









 Hawthorne_8497	8/7/2019 1:26 PM	Microsoft Excel C...	1,013 KB
 Hawthorne_C274	8/7/2019 1:26 PM	Microsoft Excel C...	998 KB
 Hawthorne_F8CD	8/7/2019 1:26 PM	Microsoft Excel C...	916 KB
 Hawthorne_tsHW	8/7/2019 1:26 PM	Microsoft Excel C...	972 KB
 Rose_075B	8/7/2019 1:26 PM	Microsoft Excel C...	581 KB
 Rose_EA5D	8/7/2019 1:26 PM	Microsoft Excel C...	580 KB
 Rose_FD74	8/7/2019 1:26 PM	Microsoft Excel C...	590 KB
 Rose_tsRP	8/7/2019 1:26 PM	Microsoft Excel C...	572 KB

Figure 5: Sample output of Data_Loading_Script

If you do not meet the criteria specified above you may have to further alter this script and you should refer to the function documentation to either build a new script or alter this script for one's purposes. If you do not have repetitions in the MAC addresses one should remove the following piece of code.

```
#This is to create pure "other" set
air_dict_list = list(df_AirU_dict)

for i in range(0,len(air_dict_list),1):
    new_id = 'ts'+Location_abv[i] #replace ID
    header_air = list(df_AirU_dict[air_dict_list[i]]) #get current header
    header_change = find_in_list(header_air, '.1') # finds duplicates
    for j in range(0,len(header_change)):
        new_header = header_change[j][:-6]+new_id #changes id
        #saves and adds the duplicates into the dataset
        df_AirU_dict[Location_name[i]] = df_AirU_dict[Location_name[i]].rename(columns = {header_change
```

Figure 6: Code snippet from Data_Loading_Script that should be removed if there are no duplicate MAC addresses/sensor ids

Additionally, if SR value is not missing then the following line should also be removed.

```
72 #Adds missing values to DAQ-in the case it was written for rose park
73 df_DAQ_values_dict_miss,df_DAQ_NC_dict_miss |=\
74 Add_Missing_Data(df_DAQ_values_dict_unfilter,df_DAQ_Null_Code_dict,'SR Value')
75
```

Figure 7: Code snippet from Data_Loading_Script that should be removed if there are no missing values from DAQ

Step 4:

If one wants normalization to do the entire dataset and returns a single normalized file and add min and max values for each sensor. One should use Data_Normalization_Script.

```
#export combined data file
export_alldata_path =\
r"D:\AirQuality_Research\Data\Full_Redo_Data_Normalization\Remove_Params\PM1_SWD_PM10"
#Import sensor data files
import_alldata_path =\
"D:\AirQuality_Research\Data\Full_Redo_Data_Normalization\Raw_Data_Ordered"

Omit_list = ['date', 'Sensor']
delete_list = ['NOX Value', 'NO Value', 'MC Value']
delete_list+= ['CO Value', 'NO2 Value']
delete_list+= ['SWD Value']
delete_list+= ['PM10_AIR_U_Sensor', 'Min_PM10_AIR_U_Sensor', 'Max_PM10_AIR_U_Sensor']
delete_list+= ['PM1_AIR_U_Sensor', 'Min_PM1_AIR_U_Sensor', 'Max_PM1_AIR_U_Sensor']
```

Figure 8: Code snippet from Data_Normalization_Script to use the script

In Figure 8, we show the two variables that need to be altered to run the script. The export path is where the csv that is created by the script will be dumped. The import path is the path to the files cleaned and exported in **Step 3**. Then one needs to add variables such as date and sensor that cannot be normalized, but will be useful so these are added to the Omit_list and use this accordingly. Then use delete list to delete any variables that are unnecessary or unneeded. The “+=” value appends to the list. If one wants to keep some of these variables please remove them from delete list.

Step 5:

Now if we want to have an “other” dataset and a split dataset then this step is necessary, but if one just wants just one dataset then this step is unnecessary. If one wants to do it the script to use is the Data_Split_Script

```
import_list_files =\
[r"D:\AirQuality_Research\Data\Full_Redo_Data_Normalization\Remove_Params\PM1\All_Data_norm.csv",\
 r"D:\AirQuality_Research\Data\Full_Redo_Data_Normalization\Remove_Params\PM1_SWD_PMI0\All_Data_norm.csv",\
 r"D:\AirQuality_Research\Data\Full_Redo_Data_Normalization\Remove_Params\PM10\All_Data_norm.csv",\
 r"D:\AirQuality_Research\Data\Full_Redo_Data_Normalization\Remove_Params\SWD\All_Data_norm.csv"]

#List of all the directories to output to should be the same length as import_list_files
export_list =\
[r"D:\AirQuality_Research\Data\Full_Redo_Data_Normalization\Remove_Params\PM1",\
 r"D:\AirQuality_Research\Data\Full_Redo_Data_Normalization\Remove_Params\PM1_SWD_PMI0",\
 r"D:\AirQuality_Research\Data\Full_Redo_Data_Normalization\Remove_Params\PM10",\
 r"D:\AirQuality_Research\Data\Full_Redo_Data_Normalization\Remove_Params\SWD"]

#names of export file names
name_list = ['outsider_data.csv','train_validate_data.csv',\
            'outsider_data_extra.csv','train_validate_data_extra.csv']

#variables that should be omitted
omit_list = ['date','Sensor']
```

Figure 9: Code snippet from Data_Split_Script to use the script

Import_list_files can be a single file, export_list should be the same length as Import_list_files, name list is the list of names for the export csv files. Omit list is the list of variables that should be omitted from the exported file.

Cleaner_Loader_Functions List

Description: This the that allow Data_Loading_Script to Work

1. Separate the raw data from DAQ and AirU datasets
2. Remove Null data functions from DAQ
3. Add missing data to datasets (DAQ)
4. Cleans Air U dataset
5. Combines DAQ and AirU datasets
6. Final Data Ordering (Sorts headers by alphabetical order)
7. Export Data

Function Dependencies:

- panda
- numpy
- os
- copy

indexall

Description: Used to get index of value in a list. Example pass lst =[0,1,2] and value = 0

Function Dependencies:

- Python 3

Inputs:

- lst can be a list of int/float/str
- value can be a single int/float/str

Optional Inputs:

- None

Outputs:

- index of where value is in list is an int

Optional Outputs:

- None

`find_in_list`

Description: Used to match strings

Function Dependencies:

- Python 3

Inputs:

- `lst` can be a list of int/float/str
- `float` can be a single str

Optional Inputs:

- None

Outputs:

- Is a list of strings in the list that match the value

Optional Outputs:

- None

`sub_main_merge`

Description: Used to match strings

Function Dependencies:

- Python 3
- Os
- shutil

Inputs:

- `import_path` is a path to the main folder datatype is a string dtype = str
- `export_path` is a path to the folder that one wants to export datatype is a string dtype = str

Optional Inputs:

- None

Outputs:

- None

Optional Outputs:

- None

DAQ_DATA_Filter

Description: Used in Main_Sep_Load to Separate DAQ dataset

Function Dependencies:

- Python 3
- os
- pandas

Inputs:

- df_DAQ – dataframe from DAQ dtype=dataframe
- Location_abv - List of location abbreviations dtype=list

Optional Inputs:

- None

Outputs:

- df_DAQ_dict - dictionary of dataframes DAQ dtype = dict

Optional Outputs:

- None

Main_Sep_Load

Description: Used to load datafiles from DAQ and AirU and for DAQ Separates data into locations

for AirU it will Separate into data into locations and into sensors

Function Dependencies:

- Python 3
- os
- pandas
- DAQ_DATA_Filter

Inputs:

- import_path-path to directory with all the data dtype=str
- Location_keyword-list of locations dtype=list
- Location_abv-list of location abbreviation dtype=list

Optional Inputs:

- file_type-type of file loading '.csv' dtype = str NOTE: Only loads '.csv' files
- DAQ_keyword-keyword for DAQ csv files dtype = str

Outputs:

- df_DAQ_dict-dictionary of dataframes DAQ dtype = dict
- df_AirU_dict-dictionary of dataframes AirU dtype = dict

Optional Outputs:

- None

DAQ_Parser_Separator

Description: Used to Separate DAQ dataframe values from null code and flags

Function Dependencies:

- Python 3
- os
- pandas
- find_in_list

Inputs:

- df_DAQ_dict-DAQ dataframe dictionary dtype = dict
- Location_name - list of locations dtype = list
- Location_abv - list of locations abbreviations dtype = list

Optional Inputs:

- None

Outputs:

- df_DAQ_values_dict - DAQ dataframe dictionary with values dtype = dict
- df_DAQ_Flags_dict - DAQ dataframe dictionary with flags dtype = dict
- df_DAQ_Null_Code_dict - DAQ dataframe dictionary with Null Code dtype = dict
- Symbol_Flag_list_unique - List of unique flags dtype = list
- Symbol_Null_Code_list_unique - List of unique Null Code dtype = list

Optional Outputs:

- None

Organize_Clean_DAQ

Description: Used to delete/replace missing values if 2 NaN values are next to each other the entire row of data will be deleted

Function Dependencies:

- Python 3
- os
- pandas

Inputs:

- df_DAQ_values_dict - dictionary of DAQ values dtype = dict

Optional Inputs:

- None

Outputs:

- df_DAQ_values_dict - outputs a clean dictionary of DAQ values dtype = dict

Optional Outputs:

- None

AIR_U_Sensor_Sep

Description: Separates AirU datasets into specific sensors and converts time zone

Function Dependencies:

- Python 3

- os
- pandas

Inputs:

- df_AirU_dict - dictionary of AirU values dtype = dict

Optional Inputs:

- data_offset - convert GST to MST (7 hour difference)

Outputs:

- df_AirU_sensor_dict - outputs a clean dictionary of AirU values Separated by sensor and the date is converted to the correct time zone dtype = dict

Optional Outputs:

- None

[Matchin_DAQ_AIR_U_time](#)

Description: Matches DAQ and AIR_U datasets

Function Dependencies:

- Python 3
- os
- pandas

Inputs:

- df_AirU_sensor_dict-dictionary of AirU dataframes Separated by sensor dtype = dict
- df_DAQ_values_dict-dictionary of DAQ dataframes Separated by Location dtype = dict

Optional Inputs:

- None

Outputs:

- df_All_Data_dict - dictionary of combined AirU/DAQ dtype = dict

Optional Outputs:

- None

[null_code_DAQ_filter](#)

Description: Removes data that has been nulled by DAQ

Function Dependencies:

- Python 3
- os
- pandas

Inputs:

- df_DAQ_Null_Code_dict - dictionaries of dataframe of Null Code dtype = dict
- df_DAQ_values_dict - dictionaries of dataframe of values dtype = dict

Optional Inputs:

- None

Outputs:

- df_DAQ_values_dict - clean dictionary of data frame of values dtype = dict

Optional Outputs:

- None

[Add_Missing_Data](#)

Description: Adds data that is missing from the current DAQ dataset-First Location MUST have the data and the second/third/...ect can be added later

Function Dependencies:

- Python 3
- os
- pandas

Inputs:

- df_DAQ_Null_Code_dict - dictionaries of dataframe of Null Code dtype = dict
- df_DAQ_values_dict - dictionaries of dataframe of values dtype = dict

- missing_var - Variable that needs to be added to a dataset dtype = str

Optional Inputs:

- None

Outputs:

- df_DAQ_values_dict - dictionary of data frame of values with missing data dtype = dict
- df_DAQ_Null_Code_dict - dictionary of data frame of Null Code with missing data dtype = dict

Optional Outputs:

- None

[Organize_Clean_All](#)

Description: Used to delete/replace missing values if 2 NaN values are next to each other the entire row of data will be deleted for all data

Function Dependencies:

- Python 3
- os
- pandas

Inputs:

- df_All_Data_dict - dictionary of combined AirU/DAQ dtype = dict

Optional Inputs:

- None

Outputs:

- df_All_Data_dict - outputs a clean dictionary of all data values dtype = dict

Optional Outputs:

- None

[Reorder_df](#)

Description: Reorder dataframe to make sure that the columns allways match up

Function Dependencies:

- Python 3
- os
- pandas

Inputs:

- df_All_Data_dict_clean - clean dictionary of dataframe dtype = dict

Optional Inputs:

- None

Outputs:

- df_reorder - outputs a reordered version of df_All_Data_dict_clean dtype = dict

Optional Outputs:

- None

[Reorder_df](#)

Description: Export CSVs given the dictionary dataframe and the path

Function Dependencies:

- Python 3
- os
- pandas

Inputs:

- df_All_Data_dict_clean - Dictionary data frame that is to be exported dtype = dict
- Export_path - export csvs to file

Optional Inputs:

- None

Outputs:

- None

Optional Outputs:

- None

Normalizer Functions List

extract_date

Description: Used to extract the hour of day/day of the week/month of the year

Function Dependencies:

- Python 3
- pandas

Inputs:

- df - all data dataframe dtype = dict

Optional Inputs:

- None

Outputs:

- df_date - outputs a dataframe that has date/hour/month

Optional Outputs:

- None

replace_header_id

Description: replace sensor id and replace with a generic name

Function Dependencies:

- Python 3
- find_in_list
- indexall

Inputs:

- header- list of headers dtype = list
- sensor_id-the sensor id dtype = str

Optional Inputs:

- `replace_id`- what to replace the sensor id with `dtype = str`

Outputs:

- `header_new`-the new header stripped of the sensor id and replaced with a generic name

Optional Outputs:

- `None`

Combine_All_Data

Description: Combines all the data, adds min and max columns, and export a dataframe

with all the data

Function Dependencies:

- Python 3
- `os`
- `numpy`
- `replace_header_id`
- `extract_date`

Inputs:

- `import_path` - the path to the clean files from data cleaner script `dtype = str`
- `export_path` - the path to export the all data csv `dtype = str`
- `export_name` - name of the csv `dtype = str`

Optional Inputs:

- `export_lg` - export is an option `dtype = bool`
- `date_convert`- extract date is an option `dtype = bool`
- `minmax` - minmax addition is an option `dtype = bool`
- `minmax_num` - number to average to make min and max `dtype = int`

Outputs:

- `df_comb`- dataframe with all the data with additional additions

Optional Outputs:

- None

clean_dataframe

Description: Removes unwanted or unneeded variables

Function Dependencies:

- Python 3

Inputs:

- df- dataframe that is inputted dtype = dataframe
- delete_list - delete the list of variables dtype = list

Optional Inputs:

- None

Outputs:

- df_data_pre_norm - dataframe without the unwanted/unneeded variables dtype = dataframe

Optional Outputs:

- None

normalization

Description: Normalizes a dataset and returns the normalized dataset and the normalizing function

Function Dependencies:

- Python 3
- sklearn
- pandas
- numpy
- random

Inputs:

- df - dataframe of normalizable data NO DATES OR STRINGS dtype = dataframe

Optional Inputs:

- Min - Minimum value dtype = float
- Max - Maximum value dtype = float
- rnum -random seed dtype = int

Outputs:

- df_norm - normalized dataframe returned dtype = dataframe
- s - normalizing function can be saved an used to unnormalize the data dtype = object

Optional Outputs:

- None

[export_normalization](#)

Description: Does exactly what it says exports the normalization tool

Function Dependencies:

- Python 3
- sklearn

Inputs:

- s - normalizing function can be saved an used to unnormalize the data dtype = object
- export_path - path to export to dtype = str
- name - name that the normalizing function is to be saved as dtype = str
- Optional Inputs:
- replace_id- what to replace the sensor id with dtype = str

Optional Inputs:

- None

Outputs:

- None

Optional Outputs:

- None

Training Machine Learning Algorithms

Introduction:

Regression analysis have been extensively for many air quality applications such as calibrating sensors[1], improving measurements of low-quality sensors[1-5], and predicting future pollutant levels[6]. Within regression analysis for air quality applications there have been quite a range of algorithms/techniques used such as: multiple linear regression model (MLR) [4], principle component regression (PCR) [6], artificial neural networks (ANN) [1, 6, 7], random forest (RF) [5], geographical weighted regression (GWR) [7], and support vector regression (SVM) [7]. The main method to conduct regression in this document is using supervised ANNs.

Background:

Note the following background came from “Machine Learning Methods in the Environmental Sciences : Neural Networks and Kernels” by William Hsieh [8].

The earliest neural network model was developed in 1953 by McCulloch and Pitts. The paper had a few purposes to show a network made out of neurons are capable of performing the same computations as a digital computer. They gave the following equation in their paper:

$$y_i = H\left(\sum_i w_i x_i + b\right) \quad (1)$$

Where y are outputs, H is a Heaviside step function. w_i is a weight parameter, x_i are inputs, and b is some bias parameter. For one's reference the Heaviside function is defined in Equation (2) as:

$$H(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases} \quad (2)$$

These equations give us a simple idea of how neural networks work. Hsieh goes on to explain the next advances in neural networks which was the perceptron model of Rosenblatt (and similarly Widrow and Hoff). The perceptron model consists of just an input and output layer and like Equation (1).

$$y_j = f\left(\sum_i w_{ji} x_i + b_j\right) \quad (3)$$

Note that unlike Equation (1) f is a generic function and could be a sigmodal or a Heaviside function depending on the application. However, the perceptron model could only be applied to linearly separable problems. In the 80s, multi-layer perceptrons (MLP) were developed by Rumelhart. The equation is similar to what is seen above, but it is repeated twice for two layers.

$$h_j = f \left(\sum_i w_{ji} x_i + b_j \right) \quad (4)$$

$$y_k = g \left(\sum_j \hat{w}_{kj} h_j + \hat{b}_k \right) \quad (5)$$

Equations 4 and 5 can be visualized using Figure 10 assuming one hidden layer.

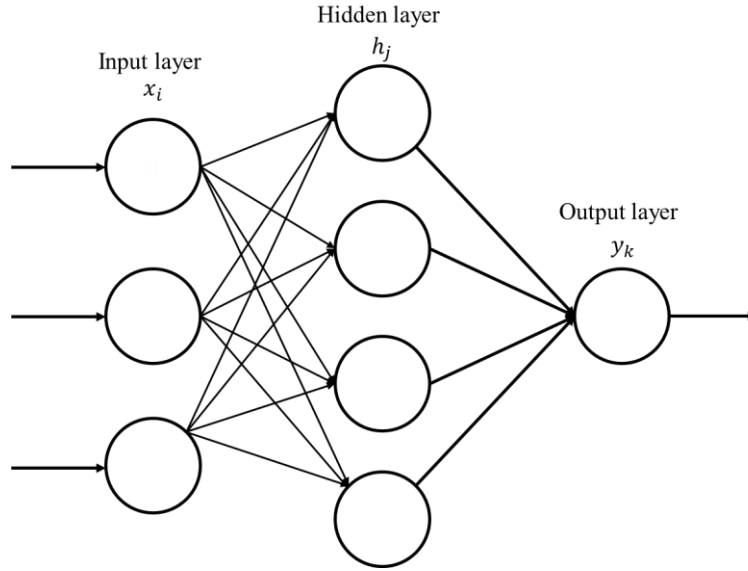


Figure 10: Scheme showing an example MLP neural network

Methodology:

Feed forward neural networks (FFNN) is a commonly used neural network where information moves in a single forward direction. FFNNs are composed of nodes, weights, and layers. Nodes are neuron like processors that are arranged in layers. These nodes are connected with other nodes in other layers, but all the connections between nodes are not weighted equally. The simplest FFNNs are composed of two layers (output and input layer) and nodes for each input and output. Multilayer perceptron (MLP) is a class of FFNNs which have a minimum of three layers. The additional layer is a hidden layer that is composed of a user specified number of nodes an example of this structure is shown in Figure 10.

This work uses an MLP model, with a Rectified Linear Unit (ReLU) activation function for the hidden layer nodes and a linear activation function for the output layer. A ReLU function is defined in

Equation 6 where γ is the saturation threshold, β is the threshold value, and α is the slope of the negative portion.

$$f(x) = \begin{cases} \gamma & x \geq \gamma \\ x & \gamma \geq x \geq \beta \\ \alpha(x - \beta) & \beta \geq x \end{cases} \quad (6)$$

The linear activation function is defined in Equation 7 as:

$$f(x) = x \quad (7)$$

The model was implemented in Python 3 using Keras as a frontend for TensorFlow. The parameter space for the network's hidden layer nodes and hidden layers were chosen to be between 6 and 160 nodes and between 1 to 4 layers. A neural network trained by picking up trends in a dataset and correlating them to the output. Training is done by taking an existing model of a network and evaluating the network on the dataset and comparing the model's prediction of a value vs the true value (loss). The network's weights are changed to minimize loss, and after several iterations the trained network is returned. To train the model a few hyperparameters had to be chosen such as the loss metric, optimizer, epoch, batch size, and drop rate. The loss metric chosen for this work was the mean squared error (MSE) (formula shown in Equation 8 where n is the number of data points, \hat{y} are the predicted output values, y are the true output values).

$$MSE = \frac{1}{n} \left(\sum_i^n (\hat{y}_i - y_i)^2 \right) \quad (8)$$

The optimization method chosen to minimize loss (or in this work MSE) by changing the weight matrix was Adam, a stochastic optimizer implemented in Keras. The parameters related to Adam optimizer followed the default settings except for the learning rate which was set to 0.05. The number of training cycles (epoch) was chosen to be 100. Both batch size and drop rate were added to prevent overfitting. The batch size determines how many data points are used is trained at a time was set to 32. The drop rate which is the percent of nodes that are randomly selected to be dropped out was set to be 20%. Training was done using computational resources provided on Google Collaboratory.

Tutorial:

For the purposes of this project, I have setup a Google Colab account that directly connects to Github repositories. The first step is to go to the Github repository and go to the Google_Colab_Jupyter_Notebook folder shown in Figure 11.

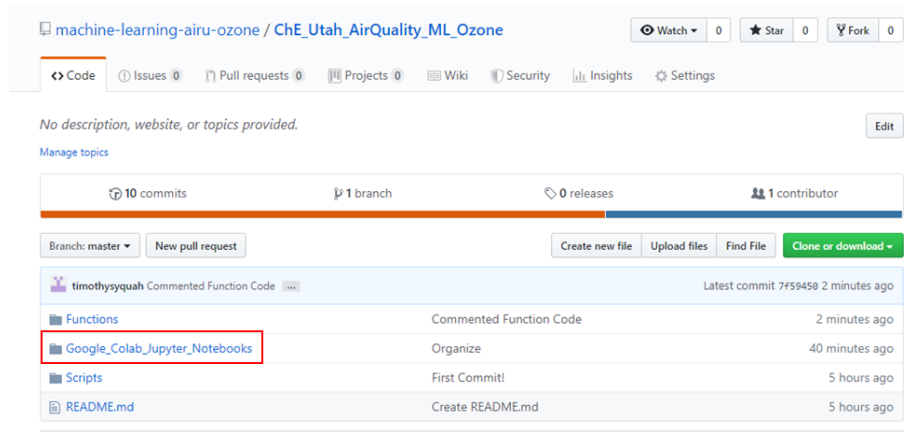


Figure 11: Snip of github repository shown the folder that should be opened.

Within that folder one should select the Neural Network Notebook shown in

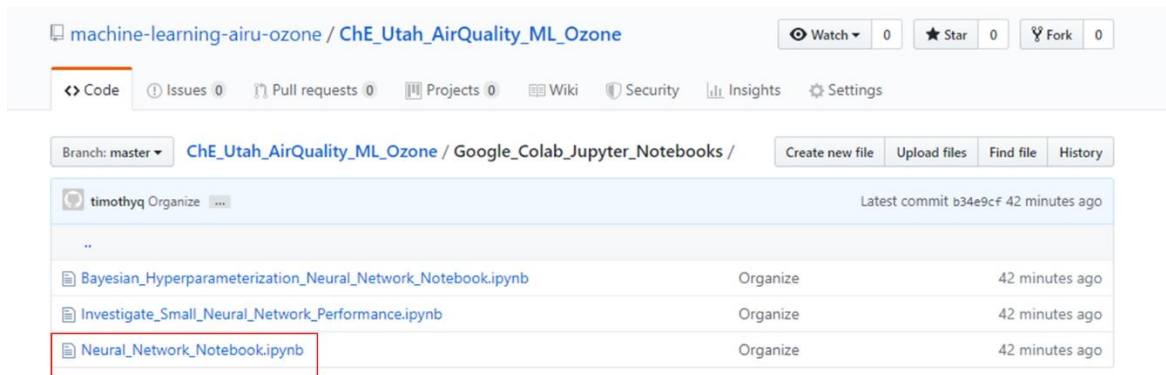


Figure 12: Snip of the Neural Network Notebook that should be selected.

If one opens the notebook one should open the notebook in google colab.

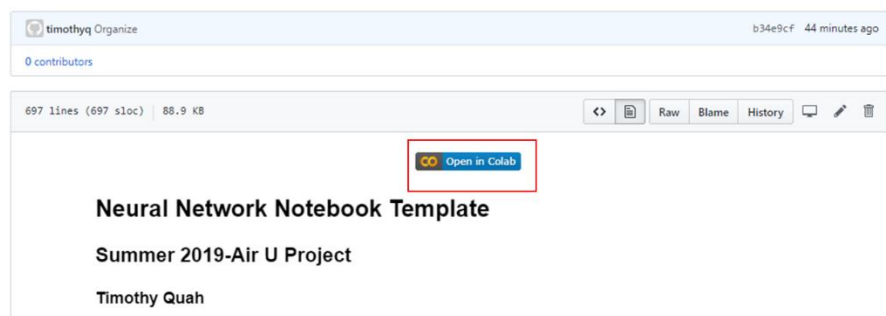


Figure 13: Showing how to open the notebook in google colab.

At this point just read through the notebook change the variables and make modifications as needed which is well documented and run the google notebook when you are ready. Usually the things that need to be changed are the inputs to the import path.

Trainer Function List

Description: This the functions that help make it a bit easier to use Keras and TensorFlow and divides your dataset for you

Function Dependencies:

- Python 3
- keras
- numpy
- random
- TensorFlow
- os

r2_keras

Description: Used to compute metric R^2

Function Dependencies:

- Python 3
- keras

Inputs:

- y_true- Actual output value dtype = keras object
- y_pred- Predicted output value dtype = keras object

Optional Inputs:

- None

Outputs:

- r^2 value dtype = keras object

Optional Outputs:

- None

[model_neural_network](#)

Description: Used to initialize neural network

Function Dependencies:

- Python 3
- keras
- TensorFlow

Inputs:

- layer - number of layers dtype = int
- nodes- number of nodes dtype = int
- input_dim_ - input dimensions dtype = int
- output_dim_ output dimensions dtype = int

Optional Inputs:

- hidden_layer - type of hidden layer activation function dtype = str
- activation_layer - type of output activation function dtype = str
- DropPercent - drop rate for drop out normalization dtype = float

Outputs:

- model - Neural network dtype-tensorflow object

Optional Outputs:

- None

[norm_divider](#)

Description: Randomly splits dataset into training and validation

Function Dependencies:

- Python 3
- numpy
- random

Inputs:

- Data_Array - data array that should be split dtype = numpy array

Optional Inputs:

- train_percent - percent training dtype = float
- rnum -random seed dtype = int

Outputs:

- train_list - the index of training data dtype = list
- valid_list - the index of validation data dtype = list

Optional Outputs:

- None

[divider_XY](#)

Description: Actually divides the dataset returning the training and validation datasets

Function Dependencies:

- Python 3
- numpy

Inputs:

- Data_Array - data array that should be split dtype = numpy array
- X_list - x locations of data dtype = list
- Y_list - y locations of data dtype = list
- train_list - the index of training data dtype = list
- valid_list - the index of validation data dtype = list

Optional Inputs:

- None

Outputs:

- X-Training data dtype = numpy array
- Y- Training data dtype = numpy array

- X_valid - validation data dtype = numpy array
- Y_valid - validation data dtype = numpy array

Optional Outputs:

- None

`load_evaluate_neural_net`

Description: Loads and evaluates a neural network

Function Dependencies:

- Python 3
- keras
- TensorFlow
- os
- r2_keras
- numpy
-

Inputs:

- path - path where the neural network is saved dtype = str
- name - name of the neural network dtype = str
- X_input - numpy array of X data dtype = numpy array

Optional Inputs:

- None

Outputs:

- Y_pred - numpy array of prediction dtype = numpy array

Optional Outputs:

- None

`mse`

Description: Used to compute loss MSE

Function Dependencies:

- Python 3
- numpy

Inputs:

- `y_true`- Actual output value dtype = numpy array
- `y_pred`- Predicted output value dtype = numpy array

Optional Inputs:

- None

Outputs:

- MSE value dtype = numpy array

Optional Outputs:

- None

R2

Description: Used to compute metric R^2

Function Dependencies:

- Python 3
- numpy

Inputs:

- `y_true`- Actual output value dtype = numpy array
- `y_pred`- Predicted output value dtype = numpy array

Optional Inputs:

- None

Outputs:

- MSE value dtype = numpy array

Optional Outputs:

None

Analyzer Function List

export_graphs

Description: Used to export figure

Function Dependencies:

- Python 3
- datetime
- matplotlib
- os

Inputs:

- plot_name - name of plot dtype = str
- fig - matplotlib figure dtype = matplotlib object

Optional Inputs:

- None

Outputs:

- export_path - path to export to dtype = str
- filetype- export file type dtype = str
- dpi_set - resolution dtype = int

Optional Outputs:

- None

plot_settings

Description: Converts plots to publication quality settings

Function Dependencies:

- Python 3
- matplotlib

Inputs:

- ax_size - axis size dtype = int
- x_tick_size - x label size dtype = int

- y_tick_size - y label size dtype = int
- figure_size - size of figure dtype = int

Optional Inputs:

- None

Outputs:

- None

Optional Outputs:

- None

[plot_parity](#)

Description: Converts plots to publication quality settings

Function Dependencies:

- Python 3
- matplotlib

Inputs:

- yvalid - Actual data dtype = numpy array
- ypred - Predicted data dtype = numpy array
- xlabel - label x axis dtype = str
- ylabel - label y axis dtype = str

Optional Inputs:

- c - color dtype = str
- s - size of dots dtype = float

Outputs:

- fig - matplotlib figure dtype matplotlib object

Optional Outputs:

- None

Parameter_Analysis

Description: Analyzes what each parameter does in the neural network

Function Dependencies:

- Python 3
- matplotlib
- keras
- TensorFlow

Inputs:

- model - Neural Network dtype = tensorflow object
- header - list of parameter names dtype = list
- X_header_list - list of parameter indexes dtype = list

Optional Inputs:

- None

Outputs:

- fig_save- dictionary of figures dtype = dict
- plt_name_save - list of names to export dtype = list
- range_save - numpy array of the range dtype = list

Optional Outputs:

- None

Evaluating Neural Networks

To evaluate neural networks the Jupyter notebooks graph parity plots, but if one wants to investigate other things such as the data in the time series context or the parameter's impact one should read and use `Analyzing_Neural_Network_Parameters` for parameter impact and `Time_Series_Analysis` for time series analysis.

References

- [1] D. B. Topalović, M. D. Davidović, M. Jovanović, A. Bartonova, Z. Ristovski, and M. Jovašević-Stojanović, "In search of an optimal in-field calibration method of low-cost gas sensors for ambient air pollutants: Comparison of linear, multilinear and artificial neural network approaches," *Atmospheric Environment*, 2019.
- [2] E. Esposito, S. De Vito, M. Salvato, V. Bright, R. L. Jones, and O. Popoola, "Dynamic neural network architectures for on field stochastic calibration of indicative low cost air quality sensing systems," *Sensors and Actuators B: Chemical*, vol. 231, pp. 701-713, 2016.
- [3] A. C. Lewis, J. D. Lee, P. M. Edwards, M. D. Shaw, M. J. Evans, S. J. Moller, *et al.*, "Evaluating the performance of low cost chemical sensors for air pollution research," *Faraday discussions*, vol. 189, pp. 85-103, 2016.
- [4] V. van Zoest, F. B. Osei, A. Stein, and G. Hoek, "Calibration of low-cost NO₂ sensors in an urban air quality network," *Atmospheric environment*, vol. 210, pp. 66-75, 2019.
- [5] N. Zimmerman, A. A. Presto, S. P. Kumar, J. Gu, A. Hauryliuk, E. S. Robinson, *et al.*, "A machine learning calibration model using random forests to improve sensor performance for lower-cost air quality monitoring," *Atmospheric Measurement Techniques*, vol. 11, 2018.
- [6] S. M. Al-Alawi, S. A. Abdul-Wahab, and C. S. Bakheit, "Combining principal component regression and artificial neural networks for more accurate predictions of ground-level ozone," *Environmental Modelling & Software*, vol. 23, pp. 396-403, 2008.
- [7] M. R. Delavar, A. Gholami, G. R. Shiran, Y. Rashidi, G. R. Nakhaeizadeh, K. Fedra, *et al.*, "A Novel Method for Improving Air Pollution Prediction Based on Machine Learning Approaches: A Case Study Applied to the Capital City of Tehran," *ISPRS International Journal of Geo-Information*, vol. 8, p. 99, 2019.
- [8] W. W. Hsieh, *Machine learning methods in the environmental sciences: Neural networks and kernels*: Cambridge university press, 2009.