Machine Learning Final Project Report (Group 24)

source : https://github.com/WarClans612/machine_learning/tree/master/final

- Working environment

張翔中	劉昱劭	彭敬樺	周才錢
Mac OS	Mac OS/Ubuntu 16.04	Ubuntu 16.04	Windows

IDE	Jupyter Notebook

- Our files

logfile

Directory structure

```
logfile
|
|
└──json
|   |   fetch.sh
|   |   toTable.ipynb
|   |   toTable.py (`ipython toTable.py` to run this script)
|   |   README.md
|   |
|   |   2018-12-08-0309.json
|   |   2018-12-08-0340.json
|   |   2018-12-08-0345.json
|   |
|   |    ...
|
└──table
    |   2018-12-08-0309.csv
    |   2018-12-08-0340.csv
    |   2018-12-08-0345.csv
    |
    |    ...
```

- Run `ipython toTable.py` to transfer json to csv

- Data Fetching (Fetch.sh file)
  - Data fetched using shell script and in json file format.
  - Fetch Airbox json file to format like 2018-12-09-0334.json.

- Data Pre-Processing (Testdata_Grouping.ipynb file)
  - Grouping data (SiteName > ddate > hour).
  - Multiple rows averaged become groups.
  - Json data processed to csv.

```python
import warnings; warnings.simplefilter('ignore')
import numpy as np
import json
import pandas as pd
from pandas.io.json import json_normalize
```

```python
# list all processed table
processedList = !(ls data/*.csv)

for i, c in enumerate(processedList):
    processedList[i] = "./data/" + c.split('/')[-1]
print('processed json:\n')
#print(processedList.nlstr)
print('-'*20)
```

```
processed json:

--------------------
```

```python
#processedList = processedList[:]
```

```python
processedList[0]
```

```
'./data/2018-12-08-0309.csv'
```

- Csv file before pre-processing

	SiteName	date	gps_lat	gps_lon	s_d0	s_d1	s_d2	s_h0	s_t0	time
0	74DA38EBF78E	2018-12-07	24.765	120.969	0.0	0.0	0.0	70.0	24.62	20:29:58
1	苗栗縣縣立南埔國小	2018-12-07	24.635	120.973	0.0	0.0	0.0	75.0	21.50	20:46:40
2	苗栗縣縣立田美國小	2018-12-07	24.629	121.008	0.0	0.0	0.0	82.0	19.75	20:32:28
3	74DA38B0538C	2018-12-07	0.000	0.000	0.0	0.0	0.0	60.0	29.00	20:44:11
4	changhua18	2018-12-07	23.879	120.362	13.0	13.0	12.0	100.0	24.25	20:36:37

- Csv file after dropped useless columns (example below)

	SiteName	date	s_h0	s_t0	time	hour	s_d0	s_d1	s_d2
0	74DA38EBF78E	2018-12-07	70.0	24.62	20:29:58	20	0.0	0.0	0.0
1	苗栗縣縣立南埔國小	2018-12-07	75.0	21.50	20:46:40	20	0.0	0.0	0.0
2	苗栗縣縣立田美國小	2018-12-07	82.0	19.75	20:32:28	20	0.0	0.0	0.0
3	74DA38B0538C	2018-12-07	60.0	29.00	20:44:11	20	0.0	0.0	0.0
4	changhua18	2018-12-07	100.0	24.25	20:36:37	20	13.0	13.0	12.0

- Example after shifting/filtering

	hour	s_h0	s_t0	s_d0_b1	s_d0_b2	s_d0_b3	s_d0	s_d1	s_d2
3	18	100.0	20.584000	1.857143	1.588235	1.642857	0.300000	0.0	0.0
4	19	100.0	20.575000	0.300000	1.857143	1.588235	2.000000	0.0	0.0
5	20	100.0	20.367368	2.000000	0.300000	1.857143	0.315789	0.0	0.0
6	21	100.0	20.250000	0.315789	2.000000	0.300000	1.923077	0.0	0.0
7	23	100.0	20.397857	1.923077	0.315789	2.000000	5.357143	0.0	0.0

- Data preprocessing cleans datasets from undefined data and junk.

- Dataset Attribute (Final_dataset.ipynb file)

	0
FAKE_GPS	GPS is provided manually (1: yes, 0: no)
gps_alt	GPS altitude (might be fake value if FAKE_GPS=1)
gps_lat	GPS latitude (decimal degrees)
gps_lon	GPS longitude (decimal degrees)
gps_num	number of satellite seen in GPS fix (might be ...
lat	GPS latitude (decimal degrees)
lon	GPS longitude (decimal degrees)
s_b0	barometric pressure (mmHg)
s_d0	PM2.5 (ug/m3)
s_d1	PM10 (ug/m3)
s_d2	PM1 (ug/m3)
s_g8	TVOC (ppb)
s_g8e	CO2 equivalent (ppb)
s_gg	CO2 (ppm)
s_h0	Relative humidity (%)
s_t0	Temperature (degree, C)

- Visualization (Testdata_Visualization.ipynb file)
  - Normalize data first.
  - Plot all attributes with the target

```
import seaborn as sns
import matplotlib.pyplot as plt
```

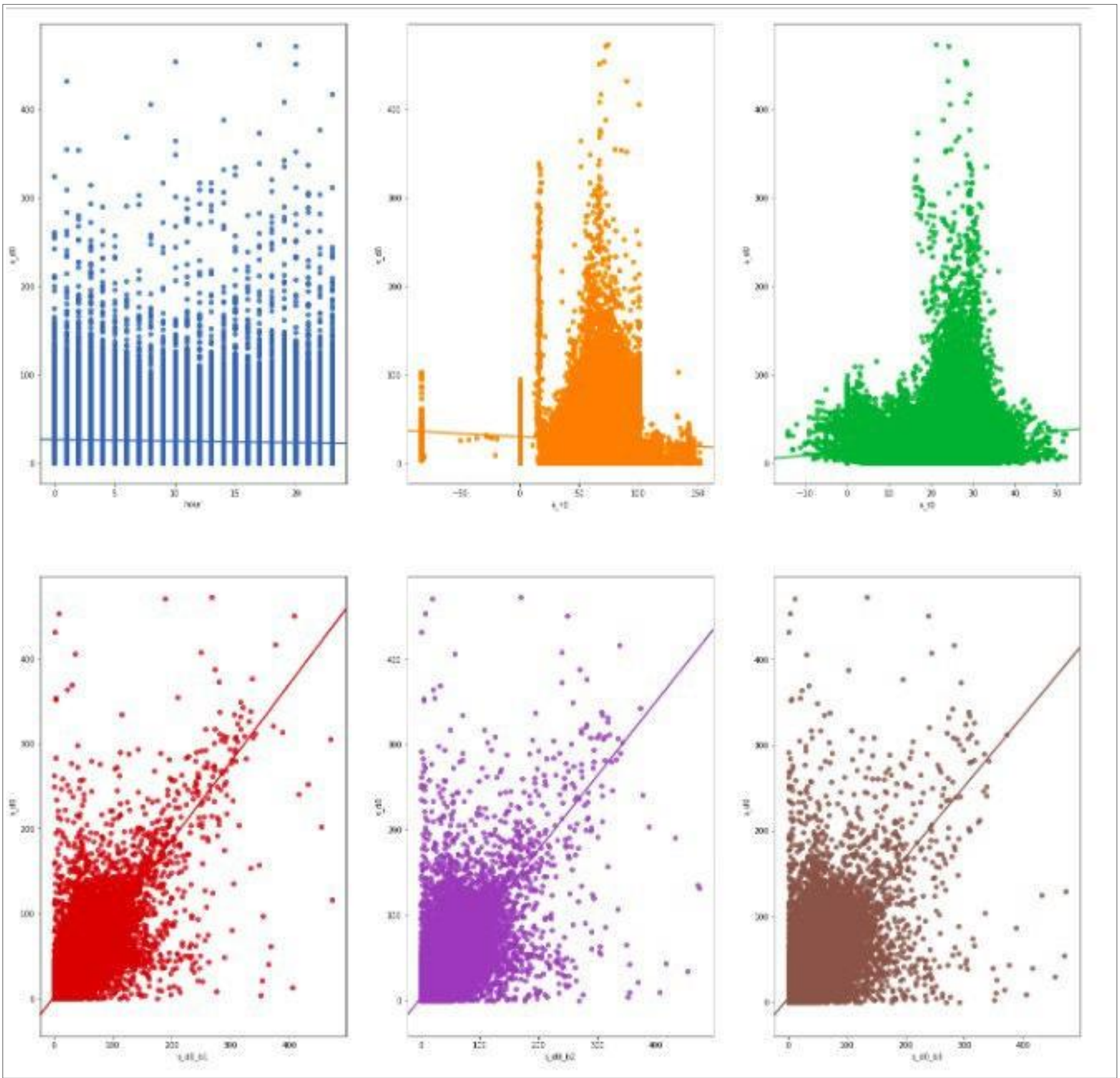
```
# Create a figure instance, and the two subplots
inputNum = len(df.columns)-3

#sns.set(font_scale=2)
fig, axes = plt.subplots(2, 3, figsize=(24, 24))


for i in range(0, 2):
    for j in range(0, 3):
        sns.regplot(x=df.columns[i*3+j], y=df.columns[inputNum], data=df, ax=axes[i][j])

#plt.show()
```

  - Using scatter plots with regression line

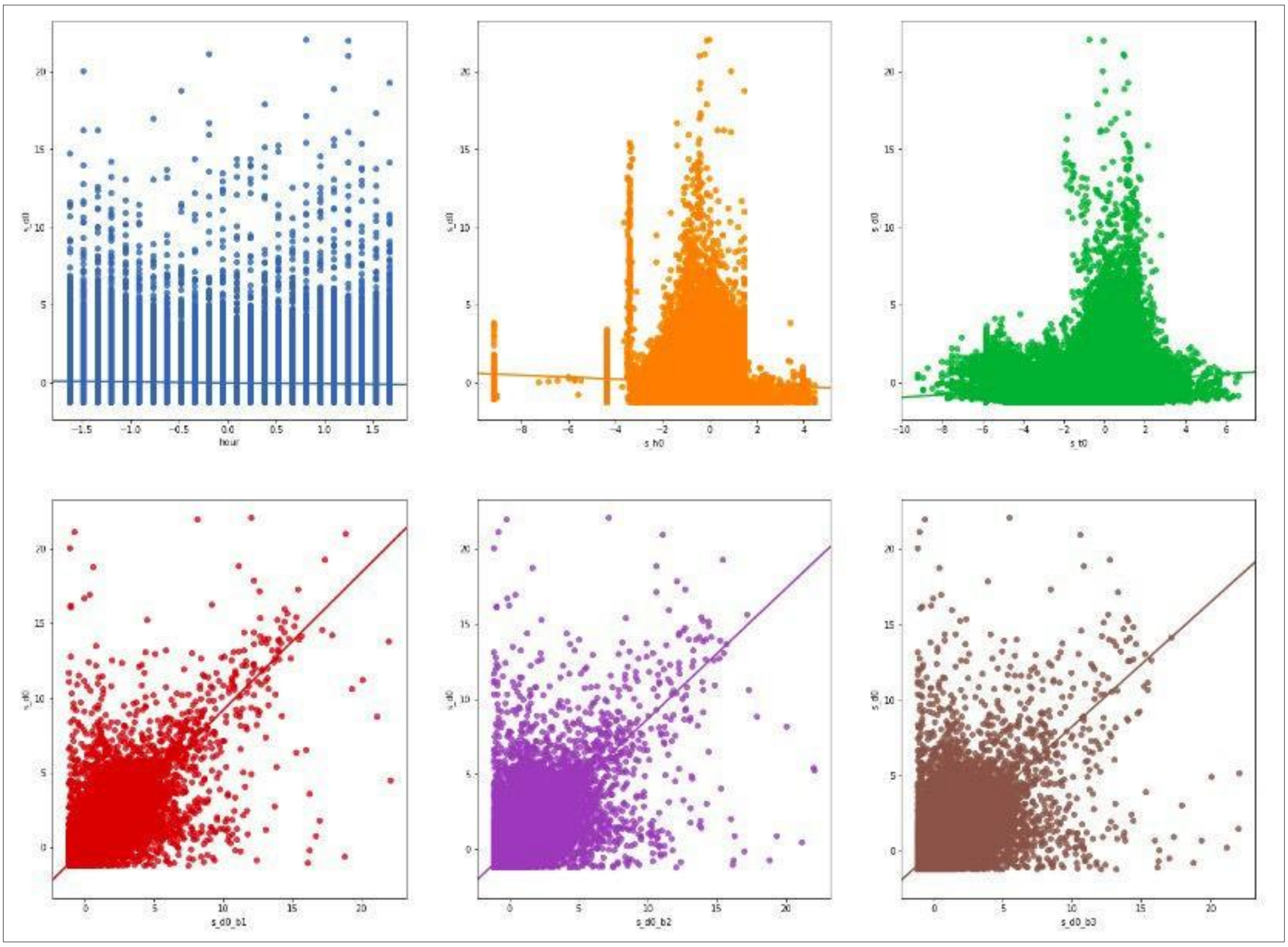


- Standardization (in Testdata_Visualization.ipynb file)

```
from sklearn.preprocessing import StandardScaler

np_scaled = StandardScaler().fit_transform(df)
df = pd.DataFrame(np_scaled, columns=df.columns)
df.head()
```

	hour	s_h0	s_t0	s_d0_b1	s_d0_b2	s_d0_b3	s_d0	s_d1	s_d2
0	0.947272	1.478326	-0.925826	-1.104995	-1.113292	-1.105243	-1.186842	-1.159696	-1.208202
1	1.090625	1.478326	-0.927982	-1.181576	-1.100068	-1.107927	-1.103216	-1.159696	-1.208202
2	1.233978	1.478326	-0.977711	-1.097969	-1.176645	-1.094711	-1.186066	-1.159696	-1.208202
3	1.377330	1.478326	-1.005821	-1.180800	-1.093042	-1.171240	-1.107000	-1.159696	-1.208202
4	1.664035	1.478326	-0.970408	-1.101753	-1.175869	-1.087690	-0.938071	-1.159696	-1.208202



- Data Partition (in Testdata_Grouping.ipynb file)

Data Partition

- For each input attribute
  - 80% data for training
  - 20% data for testing

```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split

row = ['lm1', 'lm2', 'lm3', 'lm4', 'lm5', 'lm6', 'lm7']
col = ['MSE', 'R2', 'bias', 'weight']
regResult = pd.DataFrame(index=row, columns=col)

inputNum = len(df.columns)-1

X, y = df.iloc[:, 0:inputNum], df.iloc[:, inputNum:inputNum+1]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

- Train Model & Predict (in Regressor_bad.ipynb & Regressor_good.ipynb)
  - To find most relative attribute to the target, we do regression for each input attribute.
  - Regressor_bad uses 3 training parameter (hour, s_h0, s_t0) to predict.

```
X = df[['hour', 's_h0', 's_t0']].values
Y = df[['s_d0']].values
airdata = (X, Y)
```

- Regressor_bad result

```
SGD score:  0.011176323587041459
Nearest Neighbors score:  -0.1266302470255749
Decision Tree score:  0.08527322154965467
Random Forest score:  0.09556509722326512

Neural Net score:  0.03374064543080868
Ridge1 score:  0.012018110742621602
Ridge2 score:  0.02718915382505693
Ridge3 score:  0.03959113788413816
Ridge4 score:  0.04767690182757811
Ridge5 score:  0.062160423316576896
```

model	MSE	R2
SGD	416.134126	0.011176
Nearest Neighbors	474.128305	-0.126630
Decision Tree	384.951370	0.085273
Random Forest	380.620162	0.095565
Neural Net	406.638213	0.033741
Ridge1	415.779871	0.012018
Ridge2	409.395326	0.027189
Ridge3	404.176106	0.039591
Ridge4	400.773313	0.047677
Ridge5	394.678103	0.062160

Lowest R2 score Regressor_bad can get.

Best/Highest R2 score Regressor_bad can get.

- Low correlation between input and output causes the result become bad.
- Refining results to become better with better model is futile.
- Best result would be the highest R2 score which is 0.095565.

- Regressor_good uses 6 training parameter (hour, s_h0, s_t0, s_d0_b1, s_d0_b2, s_d0_b3) to predict.

```
X = df[['hour', 's_h0', 's_t0', 's_d0_b1', 's_d0_b2', 's_d0_b3']].values
Y = df[['s_d0']].values
airdata = (X, Y)
```

- Regressor_good result

```
SGD score:  0.8516607927833313
Nearest Neighbors score:  0.83675460802985
Decision Tree score:  0.8489854245555569
Random Forest score:  0.8364578888121388

Neural Net score:  0.8598278227642228
Ridge1 score:  0.8522184523262684
Ridge2 score:  0.858562887588963
Ridge3 score:  0.8602036454090165
Ridge4 score:  0.8494075719473396
Ridge5 score:  0.7352178785247238
```

model	MSE	R2
SGD	62.426707	0.851661
Nearest Neighbors	68.699790	0.836755
Decision Tree	63.552603	0.848985
Random Forest	68.824660	0.836458
Neural Net	58.989715	0.859828
Ridge1	62.192023	0.852218
Ridge2	59.522047	0.858563
Ridge3	58.831554	0.860204
Ridge4	63.374947	0.849408
Ridge5	111.430257	0.735218

Best/Highest R2 score Regressor_good can get

Lowest R2 score Regressor_good can get

- Results between Regressor_bad and Regressor_good does not far from each other.
- Due to preprocessing, the results on both has been better.

- Conclusion
  - Time and Weather does not have much apparent relationship with PM 2.5
  - Derivative attributes that we created make data more fit to the regression models.