全部展開
回到頂部
移至底部

HW3: Cross Validation

309554001 劉雨恩

1. Data Preprocessing

1.1 Read Data

- 由於資料中有中文字，因此以 `gb18030` 來 encode 開啟資料
- 總共有 6481 筆資料，並都沒有缺失值

```
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   代□        6481 non-null   object
 1   ら戳        6481 non-null   object
 2   代兜        6481 non-null   object
 3   00         6481 non-null   object
 4   01         6481 non-null   object
 5   02         6481 non-null   object
 6   03         6481 non-null   object
 7   04         6481 non-null   object
 8   05         6481 non-null   object
 9   06         6481 non-null   object
 10  07         6481 non-null   object
 11  08         6481 non-null   object
 12  09         6481 non-null   object
 13  10         6481 non-null   object
 14  11         6481 non-null   object
 15  12         6481 non-null   object
 16  13         6481 non-null   object
 17  14         6481 non-null   object
 18  15         6481 non-null   object
 19  16         6481 non-null   object
 20  17         6481 non-null   object
 21  18         6481 non-null   object
 22  19         6481 non-null   object
 23  20         6481 non-null   object
 24  21         6481 non-null   object
 25  22         6481 non-null   object
 26  23         6481 non-null   object
```

```
# --- read data --- #
fp = open(r'./新竹_2019.csv', encoding='gb18030')
data = pd.read_csv(fp)
data.info()
fp.close()
```

1.2 Process Unused Data

- 丟掉第一列 (測站) 和第二行 (分隔符號)

```
# --- data preprocessing --- #

# drop unused data
data.drop(data.columns[0], axis=1, inplace=True)
data.drop([0], axis=0, inplace=True)
```

- 去除屬性和資料裡多餘的空格

```
# drop blank
data.columns = data.columns.str.strip()
for label in data.columns:
    data[label] = data[label].str.strip()
```

- 將非英文的文字轉成英文
  - `日期` → `date`
  - `測項` → `test_item`

```
# rename Chinese to English
data.rename(columns={data.columns[0]: 'date', data.columns[1]: 'test_item'}, i
data = data.reset_index(drop=True)
```

- 將 1~9 月的資料去除

```
# drop month 1~9
data = data.loc[data['date'] >= '2019/10/01']
print(data.head(10))
```

```
                     date test_item    00    01  ...    20    21    22    23
4824  2019/10/01 00:00:00  AMB_TEMP  24.7  25.1  ...    28  27.4    27  26.5
4825  2019/10/01 00:00:00       CH4  1.66  1.66  ...  1.82  1.83  1.93  1.96
4826  2019/10/01 00:00:00        CO  0.05  0.13  ...  0.58   0.6  0.69  0.49
4827  2019/10/01 00:00:00      NMHC     0     0  ...  0.22  0.21  0.22  0.17
4828  2019/10/01 00:00:00        NO     0   0.3  ...   0.6     2   5.1   3.8
4829  2019/10/01 00:00:00       NO2   1.3   1.1  ...  22.2  26.7    24  15.7
4830  2019/10/01 00:00:00       NOx   1.2   1.2  ...  22.8  28.6  29.1  19.5
4831  2019/10/01 00:00:00        O3  16.7  17.3  ...  23.3  13.1   7.5   5.5
4832  2019/10/01 00:00:00      PM10    18    18  ...    33    21    22    14
4833  2019/10/01 00:00:00     PM2.5     2     4  ...    19    14    17     8

[10 rows x 26 columns]
```

1.3 Process Missing Data

- 將資料中的缺失值利用 regular expression (`r'.*[0-9]+(#|\*|x|A)$'`) 尋找，並用 nan 取代
- 用 pandas 內建的 df.interpolate 將缺失值做前後差值處理

```
# process missing value
data = data.replace(to_replace=r'.*[0-9]+(#|\*|x|A)$', \
                    value=np.nan, regex=True)
for i, label in enumerate(data.columns):
    if i > 1:
        data[label] = data[label].astype(float)

data_num = data[data.columns[2:]]
data_num.interpolate(method='linear', axis=1, inplace=True, \
                     limit_direction='both')
data[data.columns[2:]] = data_num
```

1.4 Split to Training & Testing data

- 使用 10~11 月資料當作訓練集，12 月資料當作測試集
- 將資料集先以屬性做分類，再新增至新的 dataframe；並將形式轉換為行 (row) 代表18種屬性，欄 (column) 代表逐時數據資料
- training data 有 1464 筆資料，而 testing data 有 744 筆資料

Training

```
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   AMB_TEMP     1464 non-null   float64
 1   CH4          1464 non-null   float64
 2   CO           1464 non-null   float64
 3   NMHC         1464 non-null   float64
 4   NO           1464 non-null   float64
 5   NO2          1464 non-null   float64
 6   NOx          1464 non-null   float64
 7   O3           1464 non-null   float64
 8   PM10         1464 non-null   float64
 9   PM2.5        1464 non-null   float64
 10  RAINFALL     1464 non-null   float64
 11  RH           1464 non-null   float64
 12  SO2          1464 non-null   float64
 13  THC          1464 non-null   float64
 14  WD_HR        1464 non-null   float64
 15  WIND_DIREC   1464 non-null   float64
 16  WIND_SPEED   1464 non-null   float64
 17  WS_HR        1464 non-null   float64
```

Testing

```
Data columns (total 18 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   AMB_TEMP     744 non-null    float64
 1   CH4          744 non-null    float64
 2   CO           744 non-null    float64
 3   NMHC         744 non-null    float64
 4   NO           744 non-null    float64
 5   NO2          744 non-null    float64
 6   NOx          744 non-null    float64
 7   O3           744 non-null    float64
 8   PM10         744 non-null    float64
 9   PM2.5        744 non-null    float64
 10  RAINFALL     744 non-null    float64
 11  RH           744 non-null    float64
 12  SO2          744 non-null    float64
 13  THC          744 non-null    float64
 14  WD_HR        744 non-null    float64
 15  WIND_DIREC   744 non-null    float64
 16  WIND_SPEED   744 non-null    float64
 17  WS_HR        744 non-null    float64
```

```
# 10, 11 -> train data, 12 -> test data
train_data = data.loc[data['date'] < '2019/12/01']
train_data.drop(data.columns[0], axis=1, inplace=True)
test_data = data.loc[data['date'] >= '2019/12/01']
test_data.drop(data.columns[0], axis=1, inplace=True)

# group by 18 classes
new_train_data = pd.DataFrame()
for name, group in train_data.groupby(['test_item']):
    new_train_data[name] = np.array(group.iloc[:, 1:]).reshape(-1)

new_test_data = pd.DataFrame()
for name, group in test_data.groupby(['test_item']):
    new_test_data[name] = np.array(group.iloc[:, 1:]).reshape(-1)

new_train_data.to_csv('train_data.csv', index=False)
new_test_data.to_csv('test_data.csv', index=False)
```

2. Model Prediction

2.1 Predict Target/Attribute

- 分成兩種預測目標：將未來第一個小時當預測目標 & 將未來第六個小時當預測目標
- 分成兩種預測屬性：只有 `PM2.5` & 所有 18 種屬性

```
train_len, test_len = len(new_train_data), len(new_test_data)
train_np, test_np = np.array(new_train_data), np.array(new_test_data)
train25_np = np.array(new_train_data['PM2.5'])
test25_np = np.array(new_test_data['PM2.5'])
slice_num = 6

for future_i in [1, 6]:
    # training data
    all_train_x, all_train_y = [], []
    PM25_train_x, PM25_train_y = [], []
    for row_i in range(train_len - slice_num - future_i + 1):
        all_train_x.append(train_np[row_i:row_i + slice_num].T)
        all_train_y.append(train_np[row_i + slice_num])
        PM25_train_x.append(train25_np[row_i:row_i + slice_num])
        PM25_train_y.append(train25_np[row_i + slice_num])
    all_train_x = np.array(all_train_x).reshape(-1, 6)
    all_train_y = np.array(all_train_y).reshape(-1)
    PM25_train_x = np.array(PM25_train_x).reshape(-1, 6)
    PM25_train_y = np.array(PM25_train_y).reshape(-1)

    # testing data
    all_test_x, all_test_y = [], []
    PM25_test_x, PM25_test_y = [], []
    for row_i in range(test_len - slice_num - future_i + 1):
        all_test_x.append(test_np[row_i:row_i + slice_num].T)
        all_test_y.append(test_np[row_i + slice_num])
        PM25_test_x.append(test25_np[row_i:row_i + slice_num])
        PM25_test_y.append(test25_np[row_i + slice_num])
    all_test_x = np.array(all_test_x).reshape(-1, 6)
    all_test_y = np.array(all_test_y).reshape(-1)
    PM25_test_x = np.array(PM25_test_x).reshape(-1, 6)
    PM25_test_y = np.array(PM25_test_y).reshape(-1)

    for PM25_i in [True, False]:

        if PM25_i:
            train_x = PM25_train_x
            train_y = PM25_train_y
            test_x = PM25_test_x
            test_y = PM25_test_y
        else:
            train_x = all_train_x
            train_y = all_train_y
            test_x = all_test_x
            test_y = all_test_y
```

2.2 Model

兩種模型 Linear Regression 和 Random Forest Regression 建模，並計算 training score 和 MAE

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Linear Regression
Lreg = LinearRegression().fit(train_x, train_y)
Lreg_score = Lreg.score(train_x, train_y)
Lreg_y = Lreg.predict(test_x)
Lreg_MAE = mean_absolute_error(test_y, Lreg_y)

# Random Forest Regression
RFreg = RandomForestRegressor(oob_score=True)
RFreg.fit(train_x, train_y)
RFreg_score = RFreg.oob_score_
RFreg_y = RFreg.predict(test_x)
RFreg_MAE = mean_absolute_error(test_y, RFreg_y)

print('\nFuture {} hour after & PM2.5 only {}'.format(future_i, PM25_i))
print('Linear Regression: {}(score), {}(MAE)'.format(Lreg_score, Lreg_MAE))
print('Random Forest Regression: {}(score), {}(MAE)'.format(RFreg_score, RFreg_
```

3. Results

1. 明顯看出只觀測 `PM2.5` 比觀測所有屬性來的準確
2. 不管將未來 1 小時或 6 小時當預測目標，並無明顯的差別
3. Linear Regression 的 training score 都比 Random Forest Regression 好，但在觀測所有屬性 testing MAE 會大於 Random Forest Regression，有 overfitting 之嫌

3.1 Future 1 hour after & PM2.5 only

- Linear Regression: 0.8419723564256099(score), 2.613339309443991(MAE)
- Random Forest Regression: 0.8276903555445739(score), 2.9323471093044264(MAE)

3.2 Future 1 hour after & all data

- Linear Regression: 0.8835539916655512(score), 4.283428323635154(MAE)
- Random Forest Regression: 0.8710938204704244(score), 4.138888893681493(MAE)

3.3 Future 6 hour after & PM2.5 only

- Linear Regression: 0.8424668586263278(score), 2.6135923764317277(MAE)
- Random Forest Regression: 0.825896834961566(score), 2.973818781264211(MAE)

3.4 Future 6 hour after & all data

- Linear Regression: 0.8830716929572059(score), 4.295016890461818(MAE)
- Random Forest Regression: 0.8752463186626617(score), 4.136094758306926(MAE)