Time Series Regression
資料探勘 HW3

Department : IMM, 洪翊誠, ID : 310653005

November 2021

Introduction

本次作業主題，做回歸分析 (使用線性回歸模型) 問題: 新竹地區 2020 年 10 12 月之空氣品質資料，進行時間序列分析 & 迴歸預測 pm2.5 之值。
資料來源: https://airtw.epa.gov.tw/CHT/Query/His_Data.aspx

Load data

一般測站資料註記說明：

表示儀器檢核為無效值

* 表示程式檢核為無效值

x 表示人工檢核為無效值

A 係指因儀器疑似故障警報所產生的無效值

空白表示缺值

流程:

1. 閱覽資料:
   整理表格，處理讀檔問題 ($utf8$) 編碼，(可能在不同界面上讀檔會有問題，colab 無法順利執行，在 jupyter lab 可以成功)
2. 處理上述符號型資料:
   因為是時間序列資料，利用前後小時的資料補齊缺失值，若前一小時也是缺失值，則在往前找非缺失值的數值取平均 (需要非常小心，要先把表格重新排序表示)
3. 整理資料架構:
   因為是要利用前 6 小時該第 17 種欄位屬性去預測第 18 種資料 PM2.5，故需要先調整資料架構。

4. 放入模型:

   - LinearRegression 訓練線性回歸模型
   - XGBoost：XGBRegressor

5. 預測結果: 計算 MAE(mean absolute error)

閱覽資料

- 讀資料

	測站	日期	測項	00	01	02	03	04	05	06	...	14	15	16	17	18	19	20	21	22	23
1	新竹	2020/01/01 00:00:00	AMB_TEMP	15.2	15.2	15.3	15.3	15.3	15.4	15.5	...	18.1	18.2	17.9	17.3	16.7	16.4	16.2	16.1	16	15.8
2	新竹	2020/01/01 00:00:00	CH4	1.74	1.74	1.77	1.78	1.77	1.77	1.77	...	1.78	1.78	1.77	1.8	1.81	1.82	1.85	1.83	1.92	1.94
3	新竹	2020/01/01 00:00:00	CO	0.28	0.25	0.24	0.22	0.2	0.19	0.2	...	0.28	0.29	0.28	0.34	0.39	0.41	0.46	0.49	0.58	0.52
4	新竹	2020/01/01 00:00:00	NMHC	0.06	0.07	0.05	0.05	0.05	0.05	0.07	...	0.09	0.09	0.07	0.08	0.12	0.12	0.16	0.14	0.17	0.2
5	新竹	2020/01/01 00:00:00	NO	0.3	0.6	0.6	0.6	0.3	0.3	0.5	...	1.6	1.6	1.2	0.7	0.9	1.1	1.1	1.7	1.8	1.4

Figure 1: DataFrame

- 取出我們所需保留資料並且拉直

```
# df_train, df_test 都純取值
month = ['10', '11']
tmp = df['日期'].apply(lambda x: True if x[5:7] in month else
    False)
df_train = df[tmp]
df_train = df_train.iloc[:,3:]

month = ['12']
tmp = df['日期'].apply(lambda x: True if x[5:7] in month else
    False)
df_test = df[tmp]
df_test = df_test.iloc[:,3:]
```

	0	1	2	3	4	5	6	7	8	9	...	1454	1455	1456	1457	1458	1459	1460	1461	1462	1463
0	23.7	23.8	23.8	23.9	23.9	23.8	24.1	24.7	26	27.2	...	21.6	21.5	20.4	20	20.1	19.9	19.4	18.9	18.9	18.7
1	1.97	1.95	1.96	1.96	1.95	1.96	1.97	1.97	1.96	1.98	...	1.93	1.94	1.93	1.94	1.94	1.95	1.95	1.95	1.95	1.95
2	0.23	0.22	0.21	0.2	0.2	0.22	0.24	0.29	0.27	0.33	...	0.26	0.27	0.27	0.29	0.29	0.31	0.25	0.22	0.2	0.18
3	0.06	0.05	0.03	0.03	0.03	0.04	0.04	0.05	0.06	0.07	...	0.05	0.06	0.06	0.09	0.07	0.09	0.07	0.07	0.07	0.06
4	1.2	0.7	0.5	0.7	0.5	0.3	0.7	0.9	1	1.8	...	2.5	2.4	2	1.8	1.6	1.6	1.8	1.7	1.6	1.6

Figure 2: DataFrame(FLATTEN)

Function

- **Fill the average value (FILLAVE)**
  將符號位置填入數直，利用前後一小時取平均，若前後是符號，就在往更前面或更後面搜尋 (若末項是符號，則取與前項相同。

```
symbol_list = ['#', '*', 'x', 'A']
def FILLAVE(df_FLATTEN):
    for a in range(0,len(df_FLATTEN)):
        for i in range(1,len(df_FLATTEN.columns)-1):
            j = i-1
            k = i+1
            if sum([str(df_FLATTEN.iloc[a,i]).rfind(sym) for sym in
                symbol_list])!=-4:
                while sum([str(df_FLATTEN.iloc[a,j]).rfind(sym) for
                    sym in symbol_list])!=-4:
                    j = j-1
                while sum([str(df_FLATTEN.iloc[a,k]).rfind(sym) for
                    sym in symbol_list])!=-4:
                    k = k+1
                    if k>=len(df_FLATTEN.columns)-1:
                        k=j
                df_FLATTEN.iloc[a,i] =
                    str((float(df_FLATTEN.iloc[a,j]) +
                    float(df_FLATTEN.iloc[a,k]))/2)
        if sum([1 for sym in symbol_list if sym in
            df_FLATTEN.iloc[a,-1]])!=0:
            df_FLATTEN.iloc[a,-1] = df_FLATTEN.iloc[a,-2]
    return df_FLATTEN

df_Train = FILLAVE(df_Tr)
df_Test = FILLAVE(df_Te)
```

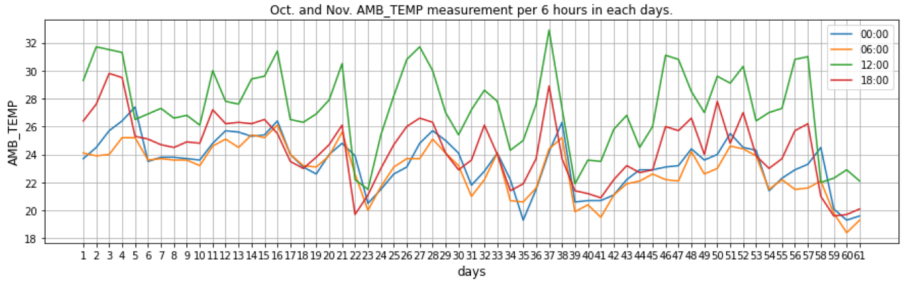
	0	1	2	3	4	5	6	7	8	9	...	734	735	736	737	738	739	740	741	742	743
11	85	86	87	87	86	86	87	86	85	80	...	58	60	61	62	62	61	61	60	60	62
12	0.5	1.9	1.8	1.8	1.8	2	1.8	2.5	3.1	3.3	...	#	#	#	#	#	#	#	#	#	#
13	2.01	2	2.01	2.01	1.99	2.01	1.97	2.03	2.04	2.06	...	2.03	2.07	2.07	2.1	2.1	2.07	2.07	2.05	2.04	2.07
14	43	45	49	43	50	35	36	34	34	37	...	54	50	52	45	47	42	42	47	45	44
15	31	38	40	47	55	37	30	41	35	35	...	48	43	44	33	50	40	46	46	51	38
16	4.3	4.2	3.8	3.9	4.7	5.3	4.3	4	4.6	5.3	...	4.5	4.4	4.2	3.8	3.7	4.7	4.5	4.4	3.9	3.9
17	3.5	3	3	3.1	3	4	3.6	3.3	3.7	4	...	3.7	3.1	3.3	3.1	2.9	3.3	3.1	2.9	2.8	2.6

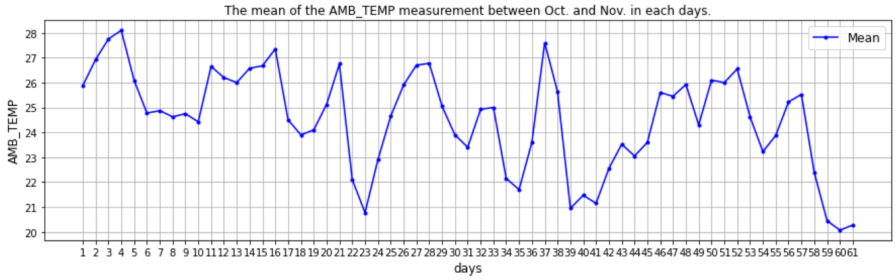
Figure 3: DataFrame(goal:Filled it)

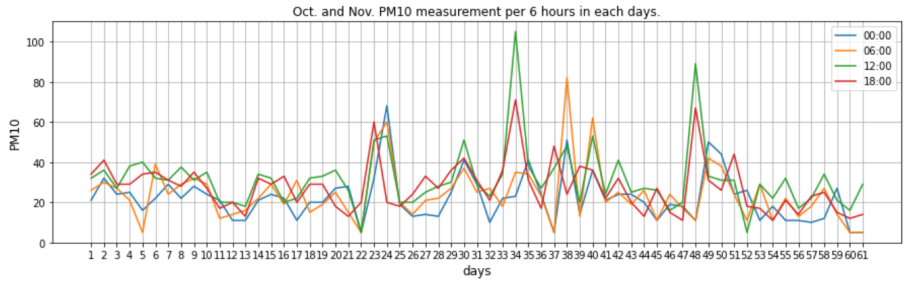
- Plot the Oct. and Nov. data figure
  藉由視覺化了解數值變化在每天的特定時間走向是如何。

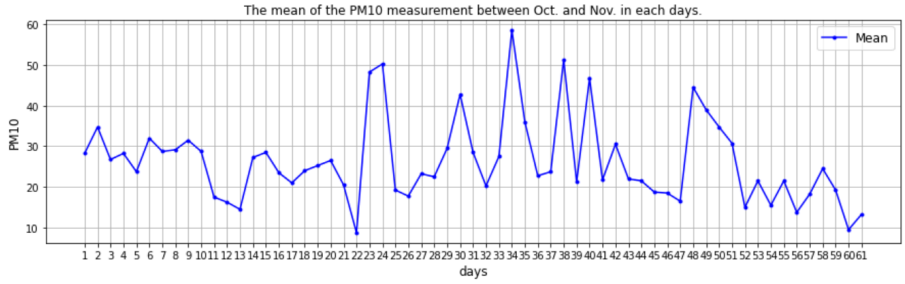
```
num = 9
Mean = 0
per_hour = 6
figsize = (15,4)
fontsize = 12

for num in range(18):
    fig = plt.figure(figsize=figsize)
    for hours in range(0, 24, per_hour):
        x1 = np.arange(hours, df_Train.shape[1],24)
        x = np.arange(1,len(df_Train.iloc[num,x1])+1)

        plt.plot(x, np.float64(df_Tr.iloc[num,x1]))
        Mean += np.float64(df_Tr.iloc[num,x1])
    plt.title(f'Oct. and Nov. {indexlist[num]} measurement per
        {per_hour} hours in each days.',{"fontsize" : fontsize})
    plt.ylabel(indexlist[num],{"fontsize" : fontsize})
    plt.xlabel('days',{"fontsize" : fontsize})
    plt.xticks(x)
    plt.legend([f'{int(e):02d}:00' for e in range(0, 24, per_hour)])
    plt.grid()
    Mean = Mean/(24//per_hour)

    fig = plt.figure(figsize=figsize)
    plt.title(f'The mean of the {indexlist[num]} measurement
        between Oct. and Nov. in each days.',{"fontsize" :
        fontsize})
    plt.ylabel(indexlist[num],{"fontsize" : fontsize})
    plt.xlabel('days',{"fontsize" : fontsize})
    plt.plot(x, Mean,'.-', color='blue')
    plt.xticks(x)
    plt.grid()
    plt.legend(['Mean'],fontsize=12)
    Mean=0
```

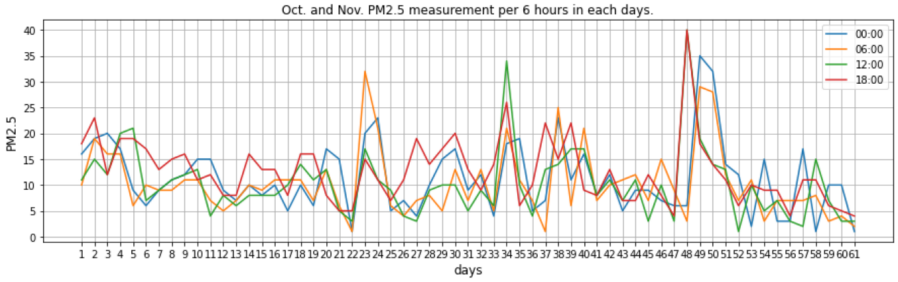
下方列幾張圖示描述，取每天的 00:00、06:00、12:00、18:00 時刻的對應屬性資料，連續繪製觀察其每日變化。有圖中可以看出在一日當中 PM2.5 波動變化其實不大，且在天空氣很糟的情況下，附近天數的空氣也是很糟的。(要看更多圖片可以點選附件的.html)


Oct. and Nov. AMB_TEMP measurement per 6 hours in each days.
AMB_TEMP
days
00:00
06:00
12:00
18:00


The mean of the AMB_TEMP measurement between Oct. and Nov. in each days.
AMB_TEMP
days
Mean


Oct. and Nov. PM10 measurement per 6 hours in each days.
PM10
days
00:00
06:00
12:00
18:00


The mean of the PM10 measurement between Oct. and Nov. in each days.
PM10
days
Mean


Oct. and Nov. PM2.5 measurement per 6 hours in each days.
PM2.5
days
00:00
06:00
12:00
18:00

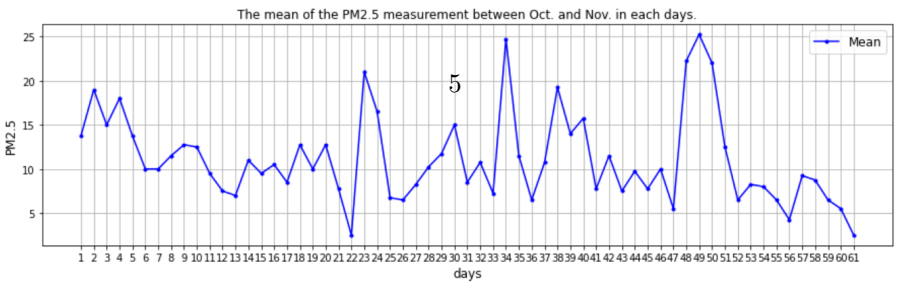

The mean of the PM2.5 measurement between Oct. and Nov. in each days.
PM2.5
days
Mean

Figure 4

- **Splitting Data** 分割訓練集 10∼11 月和測試集 12 月

```
def SPDA(df_Train, period, pred, feature=0):
    length = len(df_Train.columns) - pred
    train_x = [ [] for i in range(length)]
    if feature!=1:
        Index = df_Train.index
    else:
        Index = [9]
    for i in range(length):
        for j in Index:
            train_x[i].append(df_Train.loc[j, i:i+period-1])
    train_x = np.float64(train_x).reshape((len(train_x),-1))
    train_y =
        np.float64(list(df_Train.loc[9,pred:])).reshape(len(df_Train.loc[9,pred:]),1)
    return train_x, train_y
```

- **Training and Predict**

```
namelst = ['predict 6-th(1 feature)','predict 6-th(18 features)',
    'predict 11-th(1 feature)','predict 11-th(18 features)']
loss = []
for idx,lst in enumerate(data_list):
    train_X, train_Y, test_X, test_Y = lst
    lr_model = LinearRegression()
    lr_model.fit(train_X, train_Y)
    pred_Y = lr_model.predict(test_X)
    MAE = mean_absolute_error(test_Y, pred_Y)
    Plot_the_result(pred_Y, test_Y, hour=10, name=namelst[idx])
    loss.append(MAE)
print('The MAE(mean absolute error) is')
print(loss)
```

```
import xgboost as xgb
loss = []
for idx,lst in enumerate(data_list):
    train_X, train_Y, test_X, test_Y = lst
    xgb_model = xgb.XGBRegressor(objective='reg:squarederror',
        verbosity=2)
    xgb_model.fit(train_X, train_Y)
    pred_Y = xgb_model.predict(test_X)
    MAE = mean_absolute_error(test_Y, pred_Y)
    Plot_the_result(pred_Y, test_Y, hour=10, name=namelst[idx])
    loss.append(MAE)
print('The MAE(mean absolute error) is')
print(loss)
```

MAE (Mean Absolute Error) Table:
第一橫列說明在使用幾個特徵下預測第幾時刻時間
(6: 往後推 1 小時，11: 往後推 6 小時)

(feature(s), time)	1-6	1-11	18-6	18-11
LinearRegression	2.5224	4.5794	2.6959	6.0882
XGBoost	3.0083	5.0085	2.9761	4.6741

- **Show the Result Function Plot**

```
def Plot_the_result(pred_Y, test_Y, hour=10,
    name='LinearRegression'):
    figsize = (12,4)
    fontsize = 12
    mean_pred_Y = [np.mean(pred_Y[24*i:24*(i+1)]) for i in
        range(31)]
    mean_test_Y = [np.mean(test_Y[24*i:24*(i+1)]) for i in
        range(31)]
    x = np.arange(1,len(pred_Y[hour:-1:24])+1)
    fig = plt.figure(figsize=figsize)
    plt.title(f'The prediction of the PM2.5 measurement in Dec. in
        each days at time {hour:02d}:00.',{"fontsize" : fontsize})
    plt.ylabel('Measure of PM2.5',{"fontsize" : fontsize})
    plt.xlabel(f'Method : {name}' ,{"fontsize" : fontsize})
    plt.xticks(x)
    plt.plot(x, pred_Y[hour:-1:24], color='blue', marker='*',
        linestyle = 'dashed')
    plt.plot(x, test_Y[hour:-1:24], color='red', marker='o')
    plt.plot(x, mean_pred_Y, color='green', marker='*', linestyle =
        'dashed')
    plt.plot(x, mean_test_Y, color='orange', marker='o')
    plt.legend(['pred','real', 'M_pred','M_real'],fontsize=10)
    plt.grid()
```

Conclusion

從下方預測結果 (每日取 10:00 時刻) 和平均 (每天計算平均) 結果來看，用越多欄位反而導致跟實際資料差異大；而預測的時間越久越遠，也會導致預測偏差比較大。在做預測的過程中，理應要透過預測的數值疊加進輸入值，進一步預測後面項目 (模擬沒有未來資料的情況下，預測後面資料的過程中不會出現沒有資料能輸入的問題，但這就會如同颱風軌跡預測一樣，後面會隨之誤差越來越大)

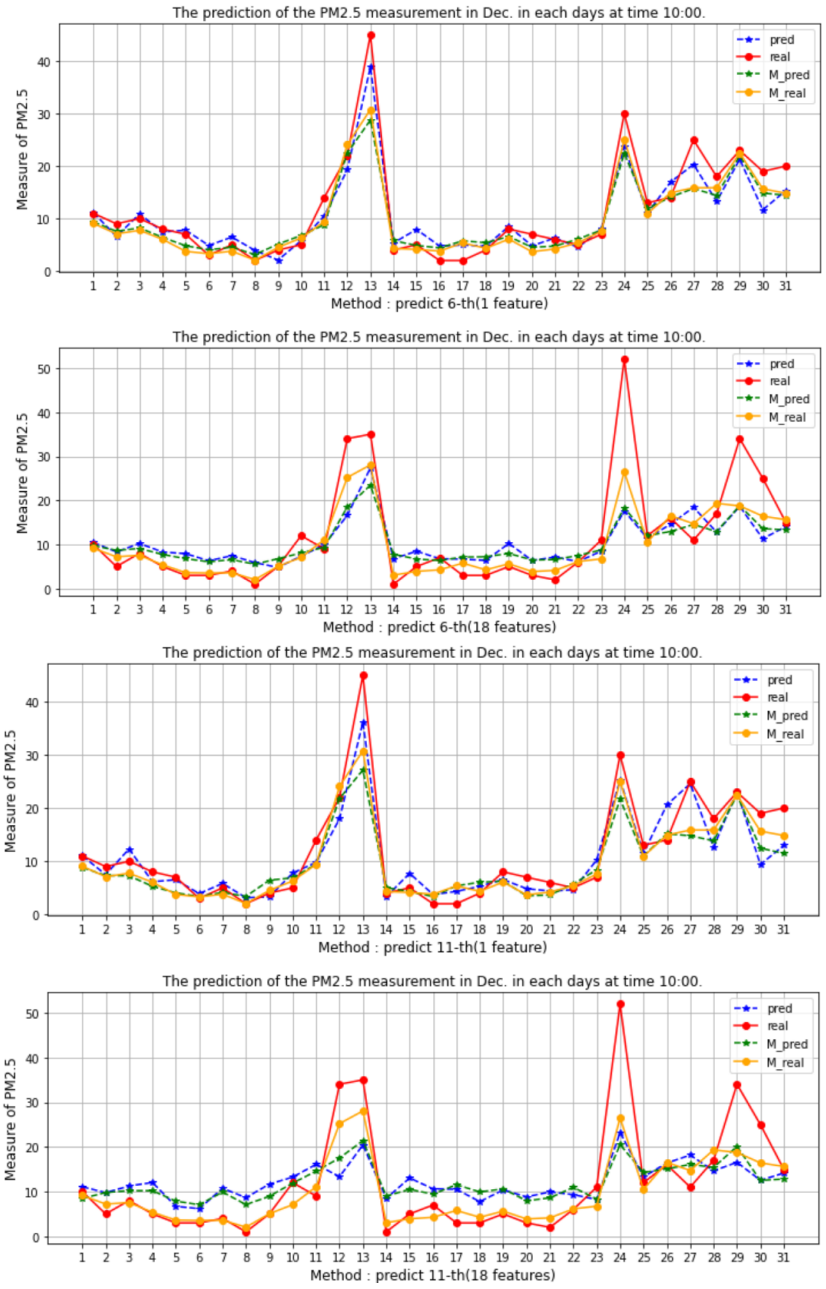


Figure 5: Result (Linear Regression)

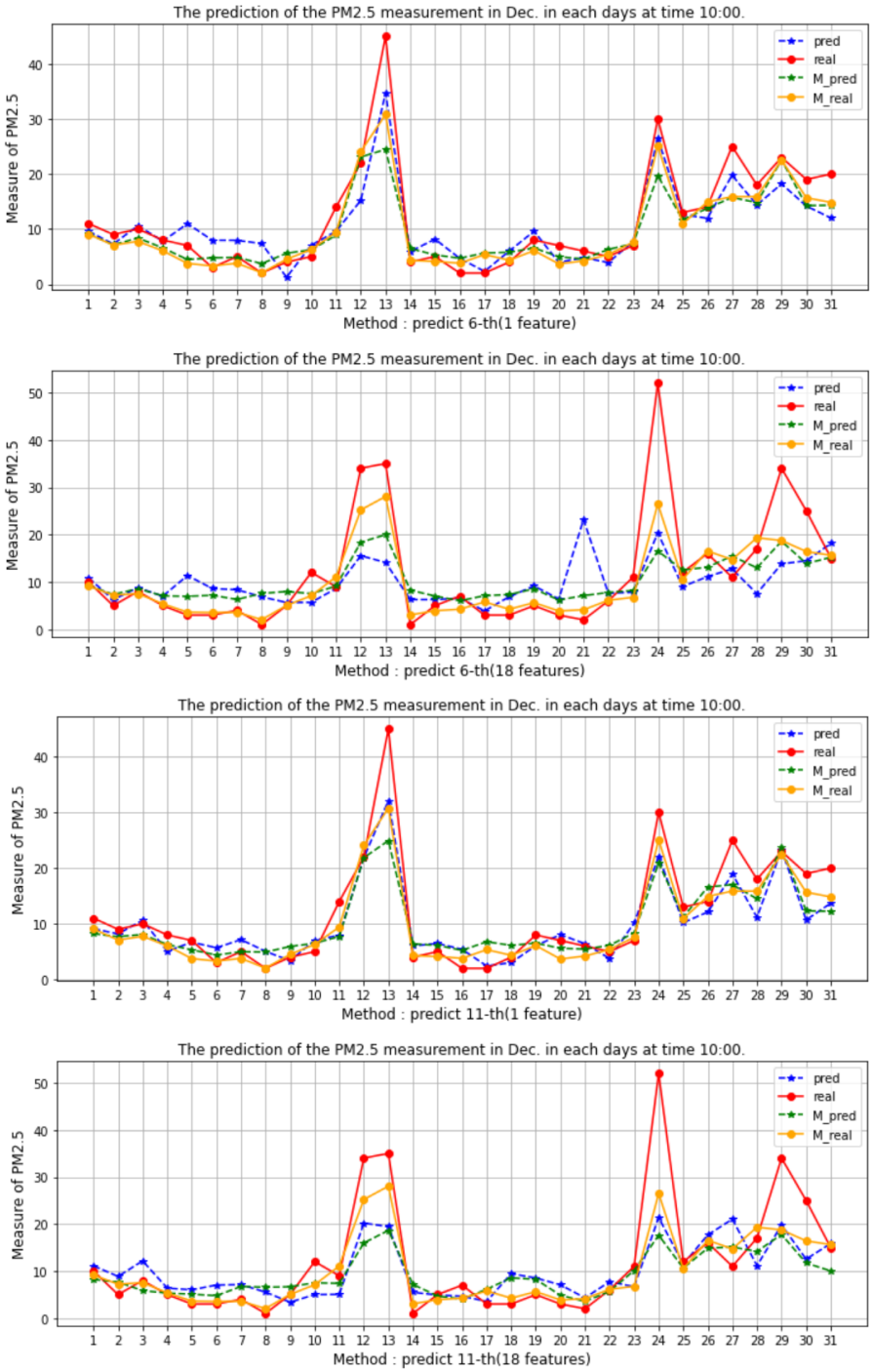


Figure 6: Result (Linear Regression)