

# Machine Learning Homework#1

## --prediction of PM2.5

b02901120 羅志軒

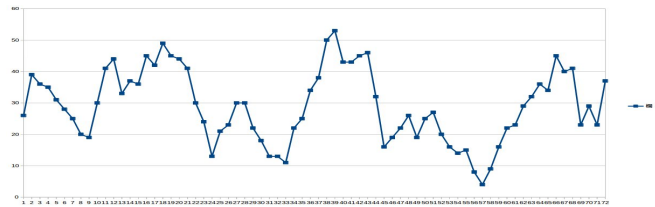
### 實驗流程

本次作業以豐原觀測站每個月前20天（共12個月）的觀測資料為training data，利用linear regression、gradient descent和regularization來訓練模型，目標為在testing data已知前九個小時PM2.5值的情況下，預測第十個小時的PM2.5值

### 實驗分析

#### 1.

為確定PM2.5值隨時間變化的關係，以連續72小時為例，將PM2.5值的變化畫出來，如圖一：

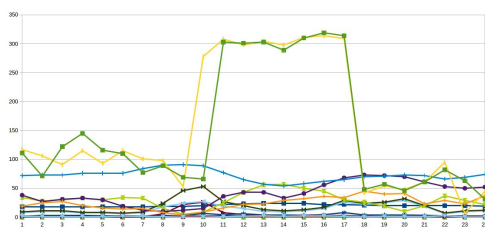


圖一

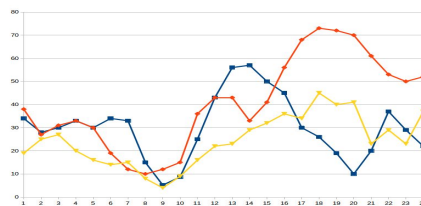
可以發現雖然PM2.5值經常會發生轉折，但大致上會符合前面九點的趨勢，且現在的值通常會和前面一兩點的值相近，可以推測，training data必須包含前九個小時的PM2.5值，且越接近現在的PM2.5值可能越需要重視。

#### 2.

若將PM2.5值和其他的觀測數據做比較，以連續24小時為例，如圖二：



圖二



圖三

可以發現O3（45列）、PM10（46列）、PM2.5（47列）的值和趨勢大致相近，將其獨立並放大來觀察（如圖三）可以推測，現在的PM2.5值可能會和PM10及O3的值相關，甚至還有其他SO2、NO2、CO等數據也會對其產生影響，training data必須將這些資訊納入考量。

### 實驗結果

#### 1.

以下分別就learning rate、lambda of regularization、iteration、training data差異及模型的差異做比較，如下：（以下的test error為自行測試的結果）

learning rate	lambda	iteration	training data	model	test error
0.000001	100	10000	O3、PM10、PM2.5	線性	7.12
0.000001	10	10000	O3、PM10、PM2.5	線性	6.58
0.000001	0	10000	O3、PM10、PM2.5	線性	6.52
0.000001	0	10000	O3、PM10、PM2.5、SO2、NO2、NOX、CO	線性	6.538
0.00001(batchsize=565)	0	10000	O3、PM10、PM2.5、SO2、NO2、NOX、CO	線性	6.420
0.00001(batchsize=565)	0	100000	O3、PM10、PM2.5、SO2、NO2、NOX、CO	線性	6.410
0.00001(batchsize=565)	0	100000	O3、PM10、PM2.5、SO2、NO2、NOX、CO	高次	6.456

就以上表格可以發現增加高次項並不會有利於貼近資料分佈，regularization在這效果也並不佳，可能是因為線性函式並不需要特別去壓低weight參數，另外，增加一些和PM2.5相關的資料有助於電腦分析及預測其趨勢，但也不是越多資料越好，過多不相關的資訊可能會造成overfitting。值得一提的是，使用Mini-Batch Gradient Descent將training data切割並random排序，不僅讓訓練速度提昇，更有助於機器以隨機的順序學習並提昇結果，最終調整learning rate到約0.00001會達到最佳值，過大會造成error震盪並發散，過小會需要較多iteration且不容易到達目標值

## 2.Gradient Descent

以下是Gradient Descent的推導公式：

$$\begin{aligned}
 L &= \sum (\hat{y} - X \cdot w)^2 = \|(\hat{y} - X \cdot w)\|^2 \\
 &= (\hat{y} - X \cdot w)^T \cdot (\hat{y} - X \cdot w) \\
 &= \hat{y}^T \cdot \hat{y} - 2(Xw)^T \cdot \hat{y} + (Xw)^T \cdot (Xw) \\
 \nabla L &= -2X^T \cdot \hat{y} + 2X^T (Xw) \\
 &= 2X^T (Xw - \hat{y})
 \end{aligned}$$

若實作在code上大約是以下的形式：

```

14 def gradient_linear(x_list, w_list, yn_list, b, lamda): #regularization, x_list, y_list-> hole batch list
15     power_arr = np.append(np.ones(62), np.array([2]), axis=0)
16     #np.power(x_list, power_arr)
17     _sum= np.dot(x_list.T, (yn_list- (np.dot(x_list, w_list)+ b))*(-1)) / len(x_list)
18     _sum_b= np.sum((yn_list- (np.dot(x_list, w_list)+ b))* (-1)) / len(x_list)
19     gradient= 2*_sum+ 2*w_list*lamda
20     gradient_b= 2*_sum_b
21
22     return gradient, gradient_b

```

將training data和weight存成numpy array以內積的方式計算gradient，計算速度將會提昇許多(“\_sum”為以上公式的 $X^T(Xw - \hat{y})$ ，“\_sum\_b”為L對b的微分值除以2)

