[Machine Learning HW1] – Linear Regression　R04921006 謝仲凱

- **(1%) Linear regression function by Gradient Descent.**

  Linear regression by Gradient Descent:

  基本上照著老師上課的公式打成 code，將 cost function 對參數做微分 當成一個 step, 並使用 step 更新參數值。

  $$\mathrm{L}(w,b) = \sum_{n=1}^{10} \left(\hat{y}^n - \left(b + w \cdot x_{cp}^n\right)\right)^2$$

  Parameters(t+1) = Parameter(t) – learning_rate*step.

```
def gradient_descent(X, y, X_validation, y_validation, theta, learning_rate, epochs):
        #---------------------------------------------
        # Performs gradient descent to learn theta
        # by taking epochs gradient steps with learning_rate
        #---------------------------------------------
        X_tmp = X
        n = X.shape[0]
        bias_ones = np.ones( (n,1) , dtype='float32')
        X_tmp = np.append(X, bias_ones, axis=1)

        X_validation_tmp = X_validation
        X_validation_tmp = np.append(X, bias_ones, axis=1)

        g_history = np.zeros( [len(theta), epochs] )
        cost_trainning_history = np.zeros(shape=(epochs, 1))
        cost_validation_history = np.zeros(shape=(epochs, 1))

        for i in range(epochs):
                predictions = X_tmp.dot(theta)
                theta_size = theta.shape[0]



                for it in range(theta_size):
                        temp = X_tmp[:,it]
                        temp.shape = (n,1)

                        error_x1 = ( y - predictions) * (-temp)
                        g = (1.0/n) * error_x1.sum()
                        g_history[it, i] = g
                        # print("g_histroy : " + str(g_history))
                        # print("rms         : " + str(rms( g_history[it,:], i+1)))
                        adagrad_coeff = rss( g_history[it,:])
                        # error_x1 = ( y - predictions) * (temp)
                        # theta[it] = theta[it] - learning_rate * g
                        theta[it] = theta[it] - learning_rate * g/adagrad_coeff



                print("iteration numbers : " + str(i))
                # print("theta : " + str(theta))
                print(" [cost] - trainning data   : " + str( compute_cost(X, y, theta)))
                print(" [cost] - validation data : " + str( compute_cost(X_validation, y_validation, theta)))
                cost_trainning_history[i, 0] = compute_cost(X, y, theta)
                cost_validation_history[i,0] = compute_cost(X_validation,y_validation,theta)

        return theta, cost_trainning_history, cost_validation_history
```

- **(1%) Describe your method.因為我們沒限制你該怎麼做，所以請詳述方法**

  ➢ **Training Data:**

  取法是以 10 小時為一單位來取，feature 是前 9 小時的資料, y_hat 是第 10 小時 PM2.5 得值。

  Training Data 分成：

  Training Set　= (7/8)筆 Training Data

  Validation Set = (1/8)筆 Training Data

A. feature:

這邊我想比較 feature 抽取的多寡與 kaggle 上的分數有沒有差（對 linear regression 的影響）

1. [PM2.5] ->最高分：5.84
2. [PM2.5,CO,NO2,O3,PM10,SO2] ->最高分：5.69
3. [其他全部 except RAINFALL] ->最高分：5.81

從這邊可以看出單純只取[PM2.5]是不夠的，需要更多的特徵會有更好的效果。上述第 2 中的特徵項目之所以會這樣取,是因為都是空氣品質 AQI 參與評價的污染物,從分數上看來這些污染物的確與 PM2.5 有著相關性,並且因為取的特徵增加分數也跟著上升。

B. y_hat:

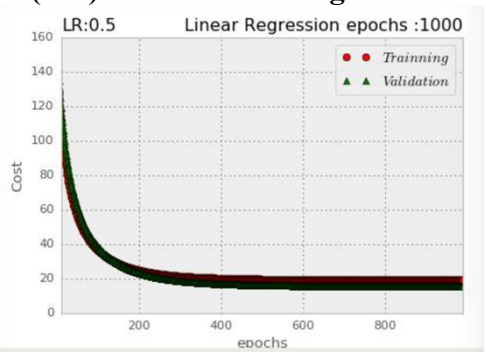
[PM2.5 第 10 小時的值]

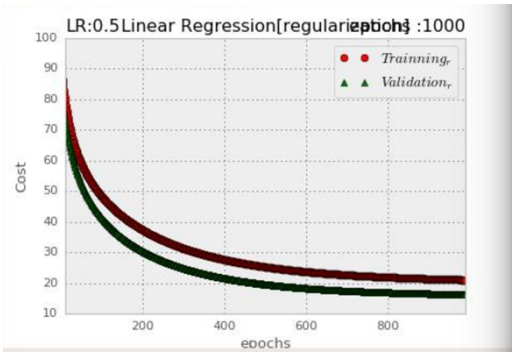
C. Parameters:

我認為參數值越接近 0 越好，因為這樣受到 Noise 的影響會越少因此我的初始參數的 mean 為 0。

Theta = 隨機的高斯函數值 (mu = 0, sigma=1.0)

- **(1%) Discussion on regularization.**



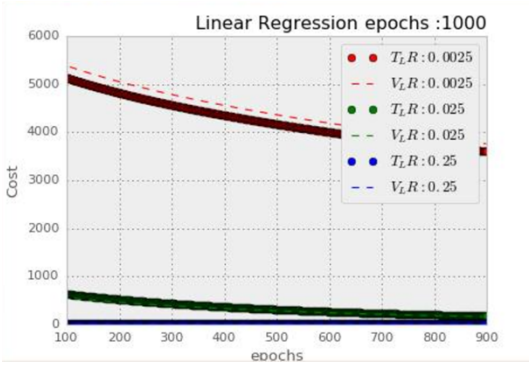


由上面兩張圖可以發現：

在所有其他條件一樣的情況下有加 regularization 的(右) cost history 線段會比較平滑,相對而言沒有加的(左)顯得非常尖銳。然而在我的實驗之中，regularization 並沒有讓我的 performance 更好，我認為是因為這次的 data 沒有特別怪異的 bias 因此沒有特別大的差別。

- **(1%) Discussion on learning rate.**

  在所有外在的條件情況之下，我們可以看到不同的 learning rate 所造成的 cost history 都不一樣。在我的實驗之中，initial 的 learning rate = 0.25 時 performance 最好，收斂最快 cost 最小。



- **(1%) TA depend on your other discussion and detail.**

在這次的功課之中我試過了不少種方式

(1)adagrad ： 收斂速度快非常多

(2)mini- batch ： 沒有感覺到差別，performance 也沒特別好

(3)momentum ： Linear regression 是一個 convex hull 因此也沒差別

(4)regularization ： cost history 比較滑順,較不容易受 bias 影響

(5)hybrid 1,2,3,4 ： …沒有比較好

最後從自己的實驗當中歸納了兩點：

初始參數：我是高斯函數隨機給的，因此每次跑的 performance 都不太一樣,因此嘗試跑很多次再把最好的 performance(看 validation cost) 上傳。

Validation set ：也是隨機從 Training Data 取的 因此如果 Validation set 與 testing data 越一致 validation 的 cost 越準 。