

Machine Learning HW1 report

姓名：顏修溫

學號：R05922034

系級：資工所 碩一

1. Linear regression function by Gradient Descent. (藍色為程式碼 灰色為註解)

首先要說明一下我使用的 feature 為下列幾個 "PM2.5", "CH4", "CO", "NMHC", "NO", "NO2", "NOx", "O3", "PM10", "RAINFALL", "SO2", "THC", "WD_HR", "WS_HR", 所定義的 model 為一次式 $y = b + wx$ ，x 就是把 5 個小時的上述幾個 feature 之數值丟進去 train by gradient descent，這個方法最後會回傳 train 之後的 bias 與 weight。接著開始說明 gradient descent function 的程式碼。

def gradient(train_data):

```
# 主程式會先讀入 train_set.csv 並將原始資料存在名為 train_data 的 list 裡頭，接著當作參數傳入此方法中。
learning_rate = 1.      # 我有使用 adagrad 方法，所以 learning rate 初始值設為 1 即可
lambda_rate = 1.        # 我有使用 regularization，這個變數就是 regularization 的 lambda 值
b = random.uniform(-2, 2)    # 隨機產生一個 bias 值
w = np.array([random.uniform(-2, 2) for _ in range(factor_size * model_houramount)])
#隨機產生 5*14 個 weight 值。因為會把 5 小時的各 14 個 feature 當作 input 餵進去 train，此 70 個 input 各要有一個 weight
adagrad_b = 0.            # 這是給 adagrad 用的變數，用來累計往後每個 epoch 的 gradient 值
adagrad_w = np.array([0. for _ in range(factor_size * model_houramount)])
# 這是給 adagrad 用的變數，用來累計往後每個 epoch 的 gradient 值
for t in range(100001):    # 執行 10 萬次 epoch，每經過幾個 epoch 會 validate 看看效能是否有持續進步
    gradient_val_b = 0.    # 用來存每個 epoch 所算出來的 gradient 值
    gradient_val_w = np.array([0. for _ in range(factor_size * model_houramount)])
    # 用來存每個 epoch 所算出來的 gradient 值
    for i in range(train_hour - model_houramount):
        # 為了計算 gradient value，所以需要用此 loop 進行 total summation
        x = [entry[i] for entry in train_data] # 從 train_set 中取得第 i 小時到第 i+4 小時的 data(共 5hour)
        for p in range(model_houramount - 1): # 從 train_set 中取得第 i 小時到第 i+4 小時的 data(共 5hour)
            for q in range(factor_size):      # 且只會取出我們指定的那 14 種 feature 的 data
                x.append(train_data[q][i+1+p])
            predict_pm25 = b + np.sum(w * x) #使用第 i~i+4 小時的資料當作 input(簡稱 x)，接著計算 b + wx
            real_pm25 = train_data[0][i+model_houramount] #取得第 i+5 小時的真正 pm2.5 數值：y-bar
            gradient_val_b += 2 * (real_pm25 - predict_pm25) * (-1) #計算 b 對 L 偏微分數值，(partial L/partial b)
            gradient_val_w += [2 * (real_pm25 - predict_pm25) * (-m) for m in x] + 2 * lambda_rate * w
            # 計算 w 對 L 偏微分的數值，也就是(partial L/partial w)，然後要加上 regularization
        adagrad_b += gradient_val_b**2      # 由於使用 adagrad，所以要把每個 epoch 的 gradient 累積起來
```

```

adagrad_w += gradient_val_w**2 # 由於使用 adagrad，所以要把每個 epoch 的 gradient 累積起來
b = b - gradient_val_b * learning_rate * (1./(adagrad_b**0.5)) # 更新 bias
w = w - gradient_val_w * learning_rate * (1./(adagrad_w**0.5)) # 更新 weight
return (b, w)

```

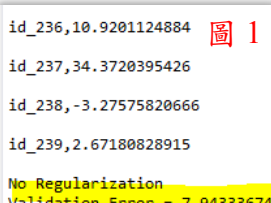
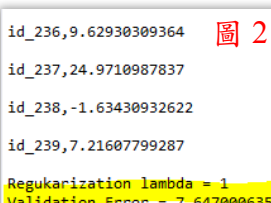
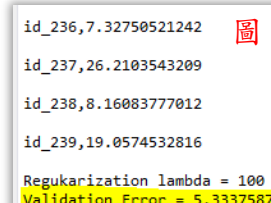
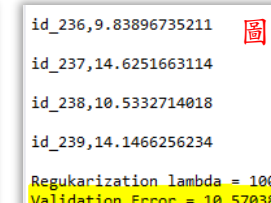
2. Describe your method. 因為我們沒限制你該怎麼做，所以請詳述方法 ex:怎麼取 training feature (X,y).

→ 關於取 training feature 的部分。一開始我是把 18 個維度都拿來當作 feature，並且取 n 個小時的數值餵進去 train(就好像一個 window 在 train set 從頭到尾移動，每次餵 18*n 個數值當作 input)。接著開始增加或減少一些 feature 看效能如何，比如說我發現 WD_HR、WS_HR 這兩個 feature 對結果有一定的影響(猜想是因為風力作用會揚起灰塵微粒)，所以我就會把影響度高的 feature 留著。每每收斂得到一個 model 後，就分別丟上去 Kaggle 測試效果如何。但是後來發現這樣的作法並不恰當，因為 Kaggle 每天的上傳上限只有 5 次，在這樣的限制底下無法持續地改進我的程式。

直到後一堂課老師提供了非常棒的做法，就是從 train set 裡頭去切 validation set，這個作法讓我有個可以獨自測試的依據，最重要的是我可以每過幾個 epoch 就去測試現在這個 model 是否有持續地增進效能，這樣不僅能避免 overfitting 的情況、也能很有效率地調整我的參數(learning rate、regularization lambda)與 feature，當自己測試出來的 Avg error 優於自定義的標準，我才會丟到 kaggle 去測試。後來我繼續加上 Adagrad 與 Regularization，並且嘗試二次式與三次式，讓 training 更完整一些。

關於程式的實作。初始化時我會對 train_set 進行字串處理，只從中取出我需要的 feature 的數值，接著將資料集丟入 gradient descent 方法中進行收斂並寫出結果。每個 feature 都會對應一個 weight，然而考慮到將來的 weight 個數可能會持續變動，所以為了彈性一點我就把所有 weight 放在一個 list 裡，而 list 大小就會在初始時根據 feature 數量進行調整，讓寫法較具彈性。

3. Discussion on regularization.

 <p>圖 1</p>	 <p>圖 2</p>	 <p>圖 3</p>	 <p>圖 4</p>
--	--	---	--

→ 我從 train_set 中切出 validation_set，並且分別用 without regularization、lambda = 1、lambda = 100、lambda = 10000 的 model 去跑 validation_set，發現有非常顯著的效果差異。首先比較有使用 regularization(圖 2)的結果和沒有使用 regularization(圖 1)的結果，發現有使用 regularization 可以得到較佳的 Avg error(圖 2)，而且如果持續增加 lambda 值的話，有機會得到更優的 Avg error(圖 3 得到最佳結果)，這是因為 regularization 讓 function 趨於平滑，output 就比較不會 sensitive to input noise。但是也不能無限度地增加 lambda，若 lambda 太大反而會造成此 function 太少考慮 training error 的部分(圖 4 顯示 lambda 太大反而 Avg error 變得很差)。

4. Discussion on learning rate.

→ 一開始在設定 learning rate 時，時常因為 rate 太大而一直來回震盪無法收斂到函數的最小值、甚至發生 overflow 的情況；接著慢慢地 trial and error，把 learning rate 調小後漸漸得到理想的數值。雖然設得小一點可以避免走過頭的情況，但是收斂所要花費得時間真得太長了，而且固定 learning rate 的方式可能會發生卡在 saddle point 的情況。所以後來我使用 Adagrad 去改進，讓 learning rate 的調整更具彈性，rate 會隨著時間減小就比較能避免走過頭的情況，而且函數中每個參數都有各自的 learning rate，方向上的調整變得更靈活，減少卡在 saddle point 的機會。

5. Other discussion and detail.

TA ML2016

10月10日 (4 天前) ☆

寄給我 ▾

同學你好，

我在想你是不是每一個model都一路train到training set 收斂，再去validation set上測試結果？我建議你可以嘗試每train數個epoch（也就是更新model參數數次），就將你的model在validation上看一下結果，只要你的model在validation set上不再進步，或者是退步，你就可以停止training，然後將這個時候的model拿去做test，這樣才是避免overfit的正確作法。

→ 原本我的做法是 train 一個 model 直到某個 epoch 次數後才進行測試，但是後來和助教討論後才發現這樣的作法並不好，不僅沒效率、也無法得知我現在這個 model 是否有持續地進步。經助教建議後，我每幾個 epoch 就會印出該 model 當下的 bias 與 weight，然後用這個 b, w 去跑 validation set，檢驗看看這個 model 的效能是否有在進步，若沒有的話我就能停止了。感謝助教幫助我釐清觀念並建議我這個好方法。

在這次的作業中，我深有體會每一次 train 都要花費大量的時間，而且成效是無法預測的，所以需要足夠的耐心毅力。除了學到一些 Adagrad、切 validation set 這方面的技巧，我想了各種方法去優化程式碼並減少執行時間。比如說使用 python 中的一些資料結構時，要選擇比較有效率的用法；另一方面我也使用系上的工作站，同時去跑各種參數的程式，比如說一個 task 跑取 5 種 feature、learning = 0.1 的情境；另一個 task 跑取 10 種 feature、learning rate=0.0001 的等等。這樣可以更有效率地測試各種參數，選出最佳結果。

然而經過各種嘗試，我發現取 5 個小時的數值進去 train 可以得到最佳結果，合理猜想是因為前幾個小時的數值對於要預測的那個時間點來說，已經算是太久遠，所以這些太久遠的資料反而沒辦法提供有效的資訊，而會造成我的 model 發生 overfitting 的情況。