**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)

1. **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。

1. **Discussion on regularization.**

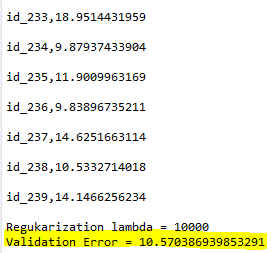
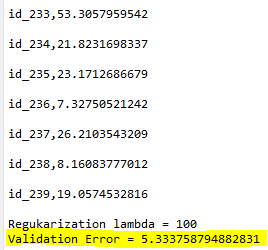
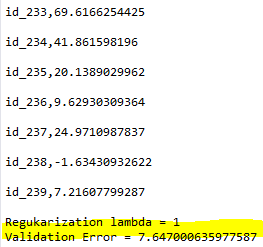
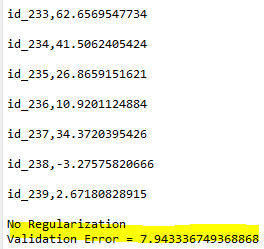


圖4

圖3

圖2

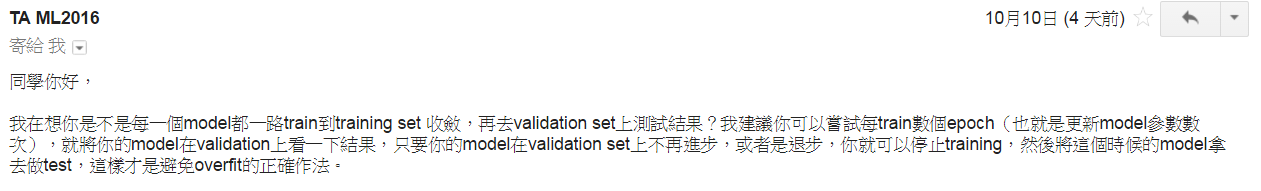
圖1

**→** 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)。

1. **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。

1. **Other discussion and detail.**



**→** 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。