Hw1 linear regression by gradient descent

b02502108
陶昇永

1.Linear regression function by Gradient Descent:

```
#------------setting initial point---------------------------------
X = np.array(x)
b,dLdb = 0,0
w = []
for i in range(162):
        w.append(0)
lambdaa = 0.5
dLdw = []
for i in range(162):
        dLdw.append(0)
totalm = 0
ypredict = 0
learningrate = 1
Gt = []
Gtb = 0
for i in range(162):
        Gt.append(0)
count = 0
#------------------------linear regression-----------------------------------------
while True:
        RMSE = 0
        count = count + 1
        for i in range(471):
                for j in range(12):
#-------------------------numbers of x,y data---------------------------------
                        yhat = float(x[18*j+9][i+9])    #yhat 為第 10hr 之 pm2.5
                        xx = []
                        for k in range(18):
                                xx = xx+x[18*j+k][i:i+9]
                        X = np.array(xx)                #做成 X, W 2 個 array 做內積
                        W = np.array(w)
                        ypredict = np.dot(X,W)+b
                        RMSE = RMSE + (yhat-ypredict)**2
#------------------------ gradient descent + adagrad ---------------------------------
                        dLdb = dLdb + (-2)*(yhat-ypredict)
                        Gtb = Gtb + dLdb**2
                        for k in range(162):
                                dLdw[k] = dLdw[k] + (-2)*(yhat-ypredict)*xx[k]
                                Gt[k] = Gt[k] + dLdw[k]**2
#-------------------------stop condition-----------------------------------------------
        if (count == 5000):
                print b,w
                break
#------------------------update w, b, dLdw, dLdb ------------------------------------
        RMSE = (float(RMSE/5640))**0.5                  #計算 train set 的 RMSE
        for i in range(162):
                if Gt[i]!= 0:
                        w[i] = w[i] - learningrate*dLdw[i]/(Gt[i]**0.5)
                b = b - learningrate*dLdb/(Gtb**0.5)
                dLdw[i] = 0
        dLdb = 0
```

2.Describe your method:

我剛開始是使用前一天的 pm2.5 當參數，配上常數項，共 2 個參數去做 linear regression，但是分數卻無法過 baseline，因此我開始增加參數。我找了很多跟 pm2.5 有關的參數，也上網去找 pm2.5 和哪些指標最為相關，可惜分數不增反減。最後我直接使用 18*9 = 162 維的 w 來預估，雖然這樣電腦要跑很久，但是意外的，成效相當的好，在加入 adagrad 和 regularization 的方法 modify 之後，已經可以超過 baseline，並隨著 train 的次

數增加，而將誤差愈降愈低。

3. Discussion on regularization:

在 regularization 方面，由於用 163 維的參數去跑成效不大，而且要跑很久，因此我用 2 維的參數來討論並比較。在老師上課的例子當中，當 lambda = 100 時可以得到最好效果，因此我剛開始也這樣設定，不料誤差反而飆升而無法降下來。後來我發現因為我們這次作業的參數，其 weight 都很小，因此太大的 lambda 可能會因此取代了真正 w 的主導地位，因為 regularization 只是輔助而已。於是我逐步調整，最後在 2 維參數時，發現套用 0.5~1 的 lambda 可以得到最小的誤差，而在 163 維的情況下，lambda 要更小，估計是在 10^-3~10^-4 左右會最好。

4.Discussion on learning rate:

在 learning rate 的部份，可以分為兩部份。第 1 部份為沒加 adagrad 的時候，為了避免 gradient descent 的移動距離太長，因此 learning rate 要非常小，大約在 10^-7 左右才可以避免掉取樣點亂衝的情況。不過為了加快 gradient descent 的腳步，我後的程式都有加 adagrad, 他利用持續增加的 Gt，放在變量的分母，可以在加快剛開始學習的腳步的同時，也讓後期的變動幅度降的很低，避免錯過最低點。由於這個緣故，若 learning rate 跟之前一樣設 10^-7 的話，整個改變就會趨近於 0 了，因此要將 learning rate 調大，我大概調到 10~100 之間，會讓整個 gradient descent 的過程最為平順。

5.Other discussions:

在做這次作業的過程中，我遇到了不少問題。剛開始的時候，我認為不必取太多不必要的 data，因為我認為那些 data 並不會幫助我預測 pm2.5，但是我後來卻發現用全部的 data 可以得到誤差最小的評分。於是我開始了解到一些機器學習跟我們人思考不一樣的地方，所有我們幫電腦設想，會幫助預測的方法，最後都會成為絆腳石，把所有的 data 餵給電腦，電腦就會在其中默默的找到一些規律。

再來就是對 adagrad 的修正，由於我是寫 python code，其速度遠不及 c++，因此我不想讓 adagrad 在我 train 太多次時，讓 descent 的幅度愈降愈低，所以我調整了程式，讓 Gt 在一段時間後固定不再增加。不過因為 gradient descent 的特性，愈接近斜率 0 處，移動的本來就會愈慢，所以總結來說效果還是有限。