Machine Learning HW#2

0510822 陳紀愷

Problem 1. Python code Exercise

(a) Please solve (0.3) by gradient descent where η = 1e-10 . What is sse and k? Does this algorithm converge?

在學習率 η = 1e-10 下做 gradient descent 可以在 k(iter) = 13671 時達到

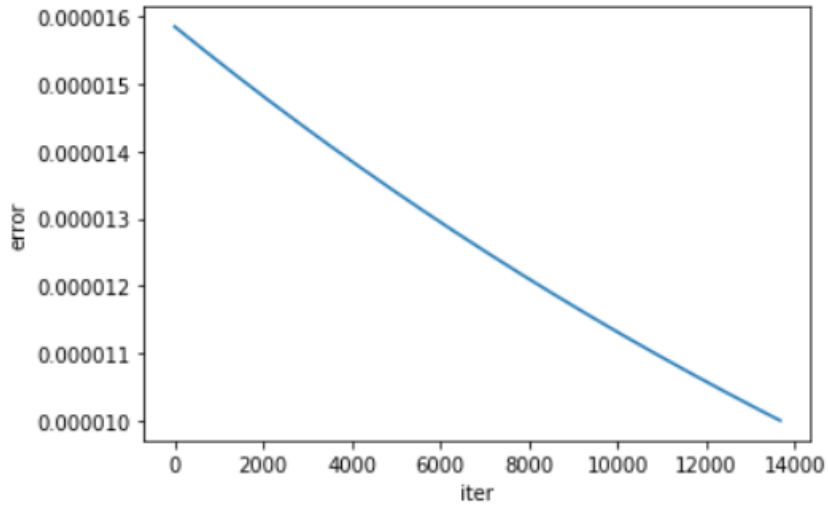
$$\text{error} = 9.9\text{e}-06 < \text{tolerance} = 1\text{e}-05$$

且 w = [b0 b1 b2] = [0.998 0993 0.826]

sse = 136560.5

而畫出 error 迭代的曲線圖也可以發現其持續穩定下降的特徵，error 持續穩定的減小→可推斷在 η = 1e-10 下 gradient descent 會 converge

```
iter=13671 error=9.999836073692512e-06
```




```
-----------------------------------------------
[b0 b1 b2]=[[0.99812504 0.99334528 0.8265406 ]]
sse=136560.54384644926
```

(b) Please solve (0.3) by gradient descent where η = 1e-5 . What is sse

and k? Does this algorithm converge?

而 b 小題則是要改用比較大的學習率 η = 1e−5 來做學習，經過 train 以後可以發現這樣的情況下 gradient descent 會完全的爆走，在 k=1 的時候error 就已經 = 1.5

```
iter=1 error=1.5850732832152075
[b0 b1 b2]=[[ 0.9766227   0.91626818 -0.58268752]]
```

而每經過一次的梯度計算以後，誤差還是不斷上升，在經過僅僅 27 次計算後 error 就已經達到 10^{10} 的數量級，因此把他強制 shutdown，在 $k(iter) = 27$ 的時候 $error = 1.09 * 10^{10}$ 且 $sse = 2 * 10^{25}$

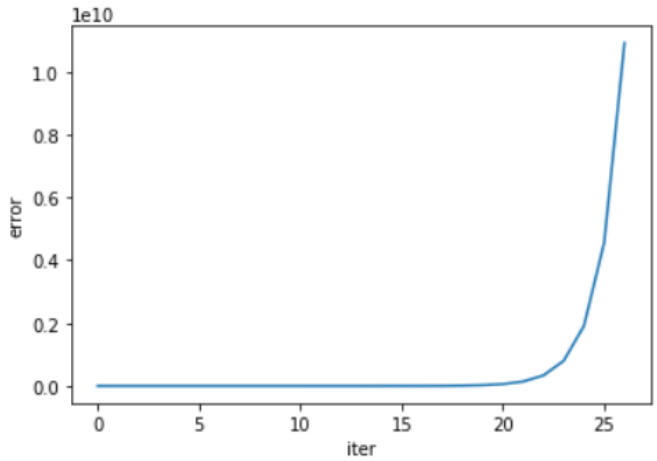
畫出曲線圖加上數據後可判斷→不會收斂。

```
iter=25 error=1910886369.8759706
[b0 b1 b2]=[[-4.12936412e+07 -1.49812058e+08 -1.33822727e+09]]

iter=26 error=4567210050.0967
[b0 b1 b2]=[[9.86959486e+07 3.58065853e+08 3.19849738e+09]]

iter=27 error=10916090025.3102
[b0 b1 b2]=[[-2.35893209e+08 -8.55813285e+08 -7.64472949e+09]]

shutdown!
```




```
-------------------------------------
[b0 b1 b2]=[[-2.35893209e+08 -8.55813285e+08 -7.64472949e+09]]
sse=2.007954734270278e+25
error=10916090025.3102
```

(c) Compare your result in (a) and (b). If your result in (a) is different from result in (b), please explain the possible reason.

(a)和(b)一個會收斂而另外一個則是發散，主要對大的原因可能是因為學習率的不同，(a)的學習率 η = 1e-10 而 b 則是 η = 1e-5，由於$w = w - \eta * \nabla f$ ，(b)過大的 η 可能會造成一次 w 變動得太大而無法找到 minimum，梯度最佳法搜尋的過程就像爬山一樣，想要走到山頂(最大或最小值)，而學習率則是一步要走多大步，學習率小的情況就像走小小步的前進，雖然花費的時間可能比較久但是最終還是會找到一個最小值(可能是區域最小)，但是當學習率過大，一步跨太大可能會一不小心就走過頭，導致無法收斂也無法求到好的解。

(d) Please calculate sse in Problem 2 (d) in HW1; compare it with sse you got in (a). What is your observation?

HW1 求到的$sse = 36753.3$，而這次(a)求到的$sse = 136560.5$，可以發現這次求出來的值很明顯的大於上次所得到的，也就是誤差比較大，我認為主要的原因是用 gradient descent 這種慢慢改進的算法可能需要經過還要經過很多次重覆計算才會比較精確，加上 gradient descent 他可能會有陷入區域最佳解的情況，因此如果要讓(a)更加精確的話可能要適當地去調整他的學習率來看

看能不能擺脫區域最佳的困境(學習率用動態之類的)。

Problem 2: Python code Exercise

利用讀取 minst 資料集訓練集中的 20000 筆資料來做訓練，再對測試集中的 1000 筆資料做預測，並記錄下來計算時間去做比較

(a)

```
KNN k= 1
Train time(sec): 10
Test time(sec): 39
Accuracy =  0.939
KNN k= 5
Train time(sec): 9
Test time(sec): 38
Accuracy =  0.937
```

(b)

```
SVM Linear
Train time(sec): 76
Test time(sec): 5
Accuracy =  0.91
```

(c)

```
KDE :
bandwidth =  0.0007856749081851164
Train time(sec): 3
Test time(sec): 25
Accuracy =  0.421
```

(d) 比較 KNN、SVM 和 KDE 三個學習方法來做比較，可以發現精確度最高的是 KNN，再來則是 SVM，最低則是 KDE。KNN 和 SVM 明顯

比 KDE 好的原因是因為他們都有直接考慮並和過去的 training 資料去做比較(最近的點或是被歸類在同一區)，因此如果兩組資料如果是在同一類別的話其實位置也會相對很接近，因此這種方法的準確性較高，而 KDE 則是用機率的方式去把很多的高斯線去做疊加，比較他們的機率，模型做起來叫為模糊沒有清楚的定義去做分類，因此準確率較低。

紀錄並比較訓練時間可以發現 training 的複雜度上 SVM>>KNN>KDE，原因是因為 SVM 是需要一筆一筆的加入新資料然後去做比較並更新分界線，尤其在這題上面維度又比較高(28*28)，加上還要分成 10 區間，因此訓練時間的複雜度很明顯的高過另外兩者，畢竟另外兩個需要的就只是一筆筆的載入資料然後簡單計算。

最後比較預測所花的時間可以發現 KNN>KDE>>SVM，KNN 需要花最久的時間也就是最複雜，主要是因為在預測的時候 KNN 需要去比較原先的所有資料，計算他們的距離並找出最近的 K 個點,來做比較,因此在這個階段他還必須回去遍歷所有的資料，因此要花的時間明顯的比其他長，而 SVM 則是把資料丟進訓練好分類集,看它所在區塊並給予他對應的標籤,整體較為簡單，因此時間也最少。

經過上面比較以後可以發現如果training的時間較長的話，通常在 predict 的部分所花的時間就比較短，相反如果 training 比較簡潔的話，通常在預測的部分就比較需要下功夫，也就是所謂的有一好沒兩好啊。