# 深度學習

## Deep Learning

# 作業二

## HW #2

B053012010 蔡宗頴

# 作業二、Gradient descent

## Part 1. Linear regression with one variable

### 1.1 Plotting the Data

在使用資料之前，將資料以視覺化的方式呈現出來。但要能以視覺化的方式呈現的先決條件是:維數不會太多。若資料的維度超過 3 維，其實也非常難直接以此次作業的方式繪圖出來觀察。圖 1.1 是作業附檔 data 的 2D 圖，呈現的是人口與餐廳利潤的關係。

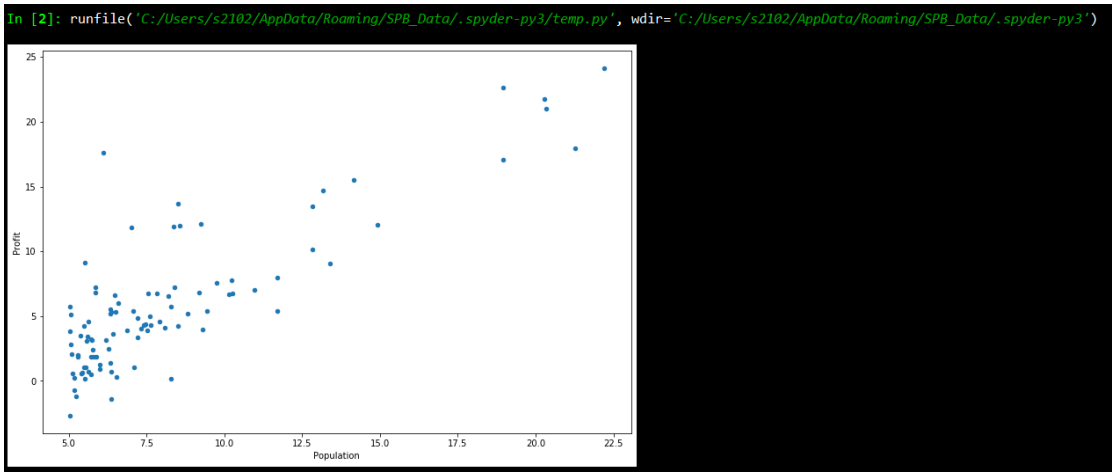


圖 1.1:作業附檔資料圖

### 1.2 Gradient Descent

#### 1.2.1 Update Equations

闡述 theta 的值如何在每一次的 iteration 中更新。由於 Gradient Descent 是要找出最小的斜率，而斜率即是一次微分，所以就計算 Cost Function 對 theta 的一次微分，再乘上一個常數 alpha 作為 theta 更替的變化值。

#### 1.2.2 Implementation

(1)Take a look at X and y.(註:由於做作業時發現沒有 pandas 這個 library，因此使用 my_array[:k]代替 my_array.head())

此處就是將 X 和 y 以矩陣的方式顯示出來，此次僅顯示到第 5 列。

```
In [10]: X[:5]
Out[10]:
matrix([[1.    , 6.1101],
        [1.    , 5.5277],
        [1.    , 8.5186],
        [1.    , 7.0032],
        [1.    , 5.8598]])

In [11]: y[:5]
Out[11]:
matrix([[17.592 ],
        [ 9.1302],
        [13.662 ],
        [11.854 ],
        [ 6.8233]])
```

(2) Convert X and y, initialize theta.

此處顯示 X, theta 和 y 的維度。(a, b)為 a 列 b 行，即 a*b 的矩陣。

```
In [12]: X.shape,theta.shape,y.shape
Out[12]: ((97, 2), (1, 2), (97, 1))
```

(3) Compute the cost for initial solution.

此處以 X, theta, y 值加上前面建立的 computeCost 函式來計算第一次的 Cost 值。

```
In [13]: computeCost(X,y,theta)
Out[13]: 32.072733877455676
```

(4) Get g and cost from "gradientDescent" and "computeCost".

此處的 g 即為 gradientDescent 函式中的 theta 值，並且經過遞迴運算之後計算出 Cost 為 4.52 左右。可以看見，相對於前面用 0 當作 theta 的初始值，經過遞迴後的 theta 值計算出的 cost 比一開始下降非常多。

```
In [47]: g
Out[47]: matrix([[-3.24140214,  1.1272942 ]])

In [48]: computeCost(X,y,g)
Out[48]: 4.515955503078912
```

(5) Plot the linear model along with the data and see how well it fits.

此處將 data 與計算出的預測值線進行疊圖比較，可發現仍然有一定的誤差。圖 1.2.1 是其疊圖。

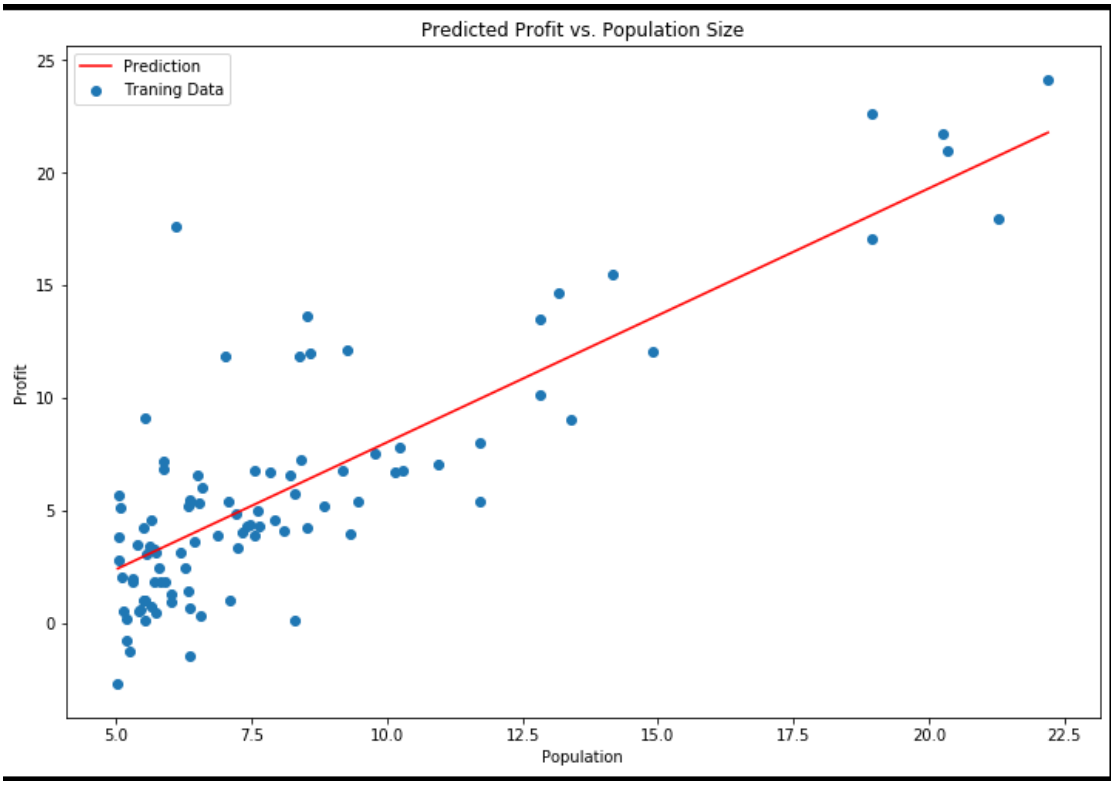


圖 1.2.1:Data with prediction

(6)Error function at each training iteration.

看看 error function 與遞迴數的關係。可以明顯看出，在遞迴前期下降速度非常快速，隨後慢慢趨緩，到 1000 次左右已經接近水平線。

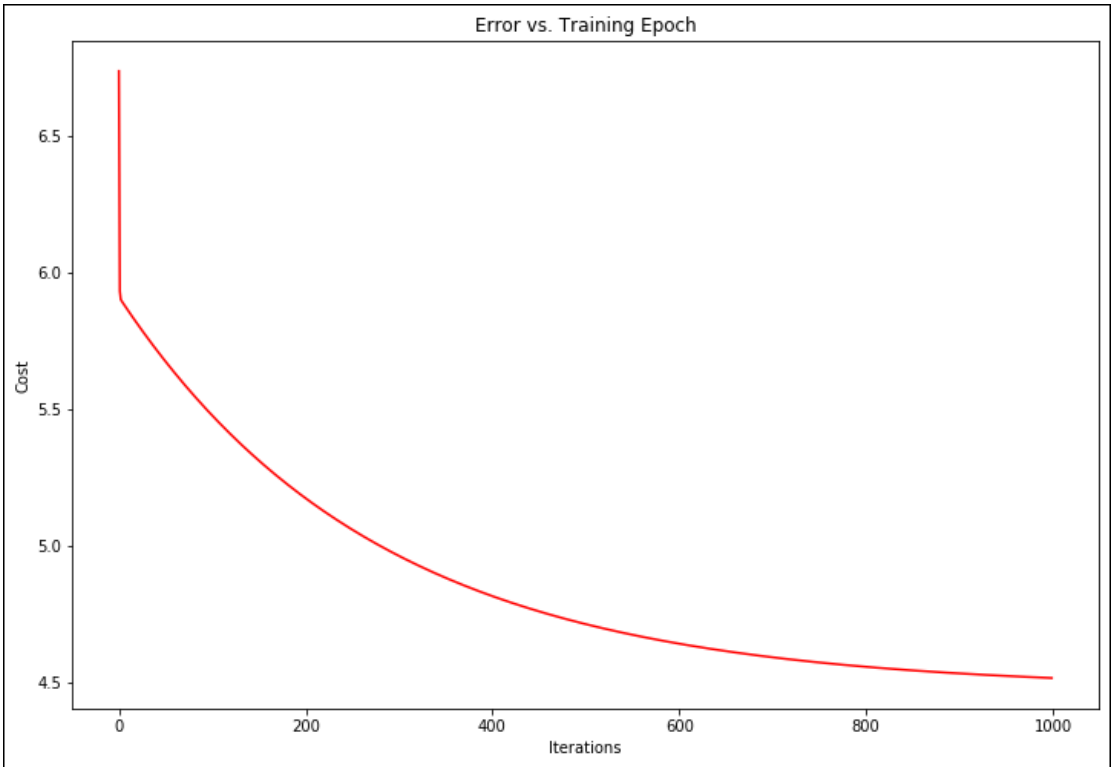


圖 1.2.2:Cost 值與 iteration 次數的關係

*討論 1:

從上圖可看出，在經過一定的遞迴次數之後，即使遞迴數繼續增加，對 cost 值的優化效果並不佳，所以我們可以設定當 cost 值小於某個值時可以停止遞迴，減少不必要的計算。

*討論 2:

雖然嘗試但無法直接另外寫一個函式來找出最適合的 iteration 次數，所以這邊我就用手動的方式改變 iteration 次數與 alpha 值來進行觀察。表 1.2.1 為同樣 1000 次時改變 alpha 值其 cost 值的變化；表 1.2.2 為 alpha 固定為 0.01 時 iteration 改變其 cost 值的變化。

| alpha | cost |
|---|---|
| 0.01(base) | 4.515955503078912 |
| 0.02 | 4.478020743321126(decrease) |
| 0.021 | 4.477702136928591(decrease) |
| 0.022 | 4.477480231721616(decrease) |
| 0.023 | 4.477325687748957(decrease) |
| 0.024 | 4.477218063976344(decrease) |
| 0.0241 | 4.477209291457483(decrease) |
| 0.0242 | 4.477200864169786(decrease) |
| 0.02425 | 4.477329826672877(decrease) |
| 0.0245 | 803101642789 56.2(increase) |
| 0.025 | 2.569410453446089e+48(increase) |
| 0.03 | N/A |
| 0.1 | N/A |
| 0.005 | 4.7140023495336 67(increase) |
| 0.001 | 5.480269332020322(increase) |

表 1.2.1: 同樣 1000 次時改變 alpha 值其 cost 值的變化

| iteration | cost |
|---|---|
| 1000(base) | 4.515955503078912 |
| 2000 | 4.4780276098799705 |
| 3000 | 4.476999993521461 |
| 4000 | 4.476972151337483 |
| 5000 | 4.476971396982805 |

表 1.2.2: alpha 固定為 0.01 時 iteration 改變其 cost 值的變化

由表 1.2.1 可知，cost 值確實是 converge 在 4.47 左右，且最低點位應可在 alpha 介於 0.0242~0.02425 之間的取值獲得；由表 1.2.2 可知，當 alpha 固定為 0.01 時，iteration 次數越多則 cost 值越小，但減小的幅度隨 iteration 越大而趨緩。因此若我們可以接受的容錯範圍較大，我們只需要做 1000 次即可；若我們要求精確度高，在時間及硬體允許下可做到 5000 次以上。並且，若 alpha 與 iteration 同時進行調整則 cost 值可能可以達到比單個調整更佳的效果，主要需要考量的是，時間與硬體成本是否符合自己的需求。

*討論 3:

在 Cost function 的選擇上，除了講義給的 MSE 外，常見的還有 MAE (Mean Average Error)和 MSLE(Mean Squared Logarithmic Error)兩種。其中，MAE 和 MSE 又稱為 L1 和 L2 Loss，這邊主要討論這兩者[1,2]。由於 MSE 有平方，因此對異常點會特別敏感。簡單地說，異常點本身就與 prediction 值相差較大，再平方就會更大，是故其會遠離異常點，計算出來的值偏向整體資料的平均數。而 MAE 就不太一樣，MAE 對異常點沒有 MSE 那樣敏感，會犧牲其他資料的準確度來進行預測，降低整體模型性能，所以計算出來的值會偏向整體資料的中位數。另外，由於中位數比平均數更穩定，MAE 的結果其實比 MSE 對於異常值更加穩定。

從上面的討論可以知道，如果要應用在能夠偵測出異常點的情況下，使用 MSE 效果較佳；若沒有要特別去偵測異常點的話，使用 MAE 即可，計算速度也較快。

(7)Apply scikit-learn's linear regression algorithm to the data from part 1 and see what the predictions look like.

此處是讓我們知道還有 sckit-learn 的套件可以直接拿來使用，這個 prediction 的圖與前面的 prediction 圖基本上相同。(因為 cost 都很接近 4.47。) 結果請見下頁圖 1.2.3。

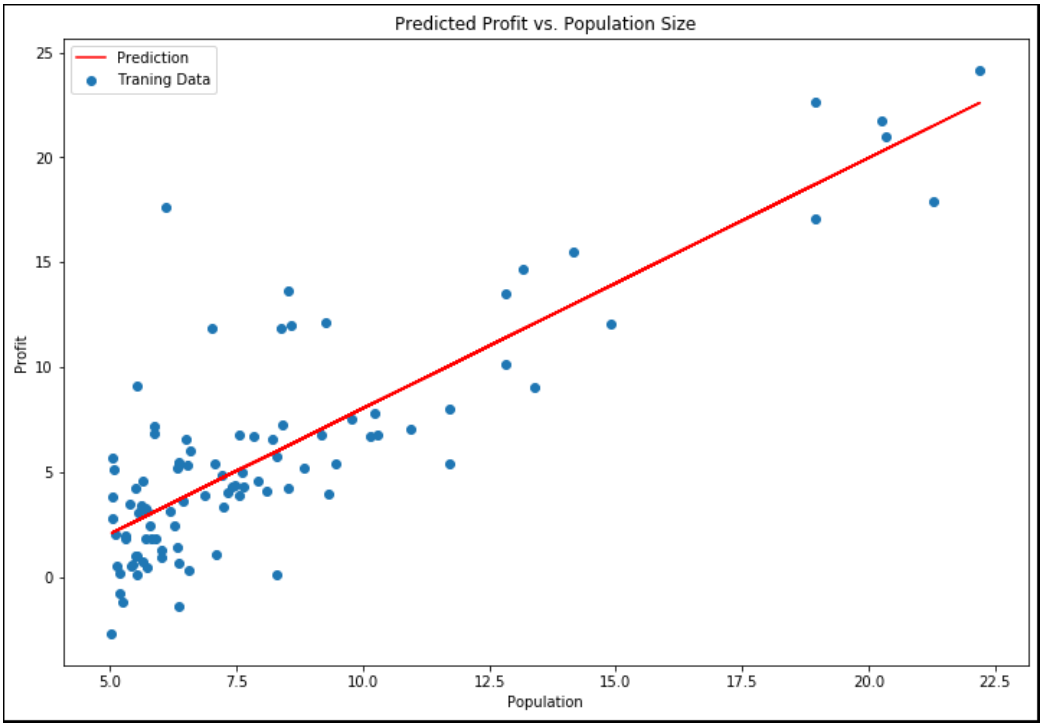


圖 1.2.3: Data with prediction, using scikit-learn's algorithm

## Part 2. Reference

[1] 機器學習\統計方法: 模型評估-驗證指標(validation index)
https://medium.com/@chih.sheng.huang821/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E7%B5%B1%E8%A8%88%E6%96%B9%E6%B3%95-%E6%A8%A1%E5%9E%8B%E8%A9%95%E4%BC%B0-%E9%A9%97%E8%AD%89%E6%8C%87%E6%A8%99-b03825ff0814
[2] L1 vs. L2 Loss function
http://rishy.github.io/ml/2015/07/28/l1-vs-l2-loss/