Machine Learning HW1: Perceptron

tags: `Machine Learning`

Files Description

- `hw1-1.py` : part 1 的 code，執行這份以執行 part 1
- `hw1-2.py` : part 2 的 code，執行這份以執行 part 2
- `hw1-1.png` : part 1 的輸出結果
- `hw1-2.png` : part 2 的輸出結果
- `utils.py` : 所有程式的相關工具程式
- `requirement.txt` : 執行程式所需額外安裝之函式庫
- `hw1-report.pdf` : 作業報告書

Execution description

Part 1. PLA

1. 用 `PLA_3_times_with_30_data` 跑所有程式
2. `build_data` 來建立資料集
   - ***x*** 為 (1, x, y) 的組合
   - ***y*** 為對應 label 之組合
3. 預設 ***w*** 為對應 (1, x, y) 之參數，初始為 (0, 0, 0)
4. `plt_proc` 為畫圖的相關程式，故不多解釋
5. 總共跑3次
   1. `PLA()` 為執行 PLA algorithm 的程式
      1. 先產生一條亂序、無重複、從0到資料數量的整數序列
      2. 對該條序列依序檢測
      3. `get_sign()` 為取得 $\text{sign}(w^T * x)$
      4. 如果 $\text{sign}(w^T * x) \neq y_i$
         - 更新 $w = w + y_i * x_i$
         - 重新開始遞迴
   2. 利用 `verification()` 驗證結果
6. 輸出圖片

Part 2. PLA v.s. Pocket

1. 用 `pocket_vs_pla` 跑所有程式

2.~4.與 part 1 一樣

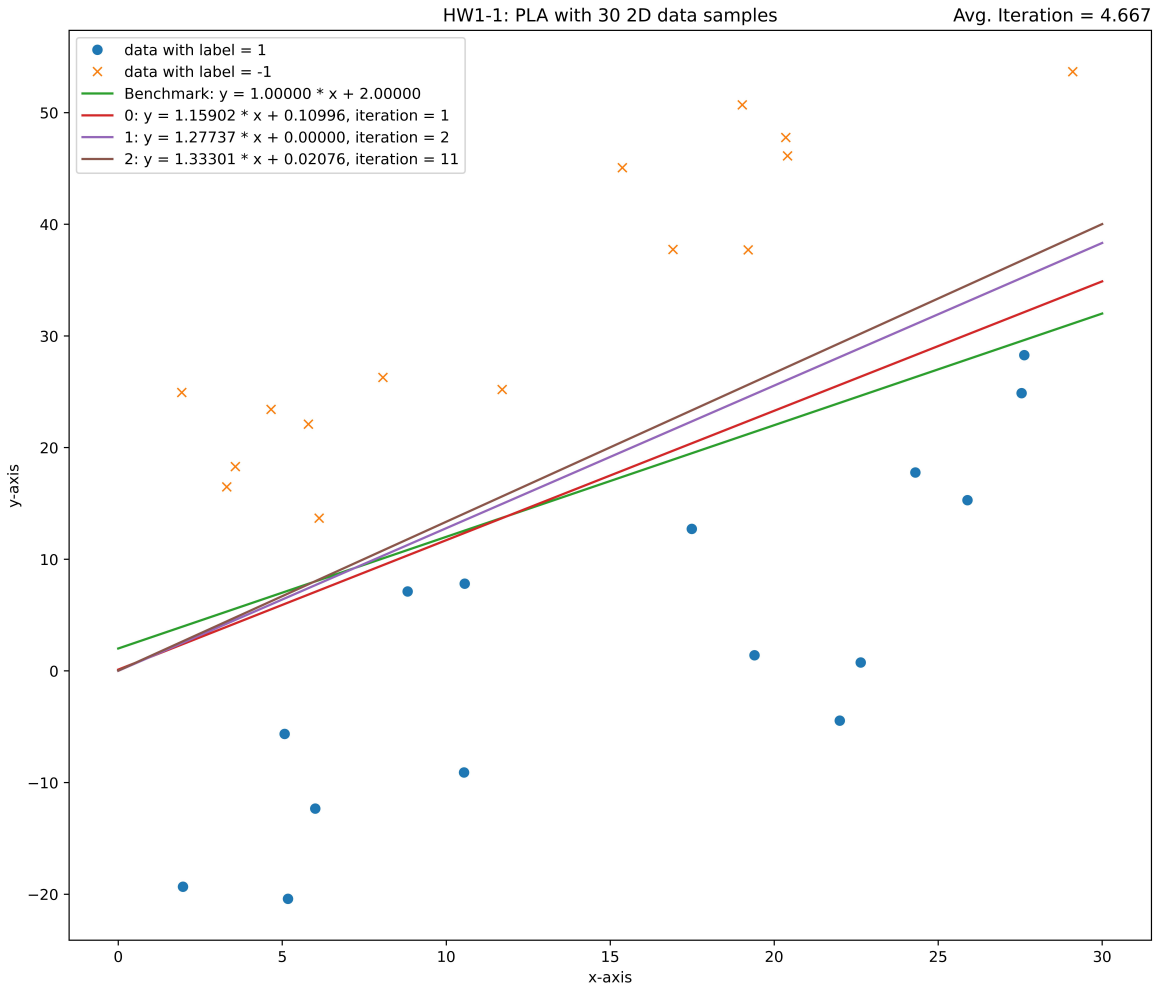
5. 利用 `time.time()` 取得當下秒數
6. 利用 `PLA()` 執行 PLA
7. 利用 `verification()` 驗證結果及計算錯誤率
8. 利用 `Pocket()` 執行 Pocket
   1. `iteration` 為設定總共要跑幾回，預設為 1000

2. `continuous_threshold` 為設定如果**連續**沒有更新 w 幾次要停止繼續遞迴，預設為 100
3. 每回合都隨機產生一個 index
4. 用 `get_sign()` 判斷是否正確
5. 分別利用 `verification()` 取得舊參數與新參數的錯誤率
6. 進行比較並更新參數

9. 最後輸出結果

Experimental results

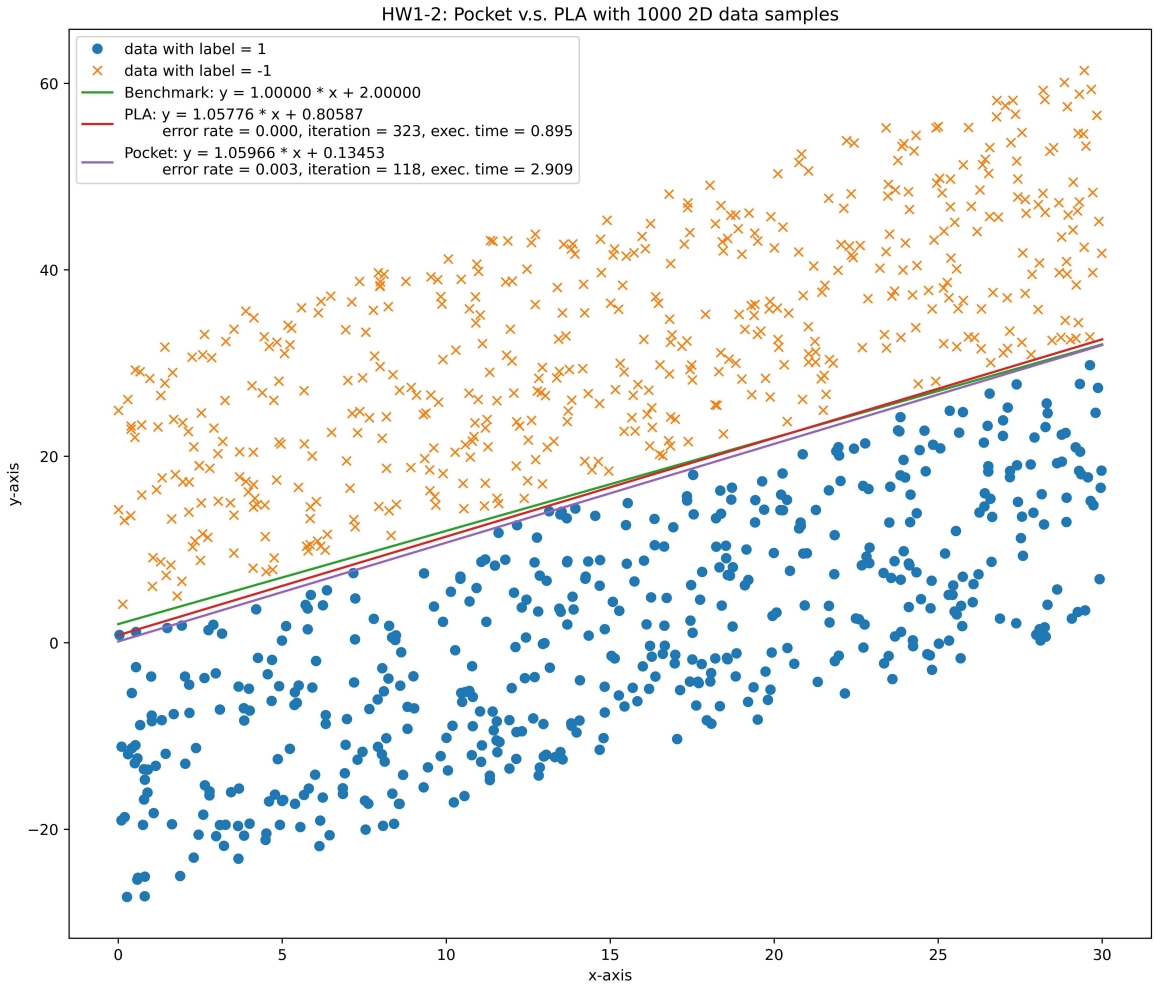
Part 1 PLA



1. y = 1.15902 * x + 0.10996
   - iteration = 1
2. y = 1.27737 * x + 0.00000
   - iteration = 2
3. y = 1.33301 * x + 0.02076
   - iteration = 11

Avg. Iteration = 4.667

Part 2 PLA v.s. Pocket



PLA execution time = 0.89505 seconds
PLA Iteration = 323
PLA error rate = 0.0

Pocket execution time = 2.90895 seconds
Pocket Iteration = 118
Pocket error rate = 0.003

Conclusion

Part 1 PLA

- 多數可以在遞迴10次內產生正確結果
- 三次的方程式都蠻接近預設標準

Part 2 PLA v.s. Pocket

- Pocket alg. 的錯誤率浮動蠻大的，但都可以保持在錯誤100個以內
- Pocket 執行時間慢 PLA 很多，原因是每一遞迴都要檢查全部點的錯誤率，而 PLA 並不用

Discussion

- 在實作 Pocket 時，發現在更新參數幾十次後，就會因錯誤率較高而不再更新參數，所以其實多跑了100多回合也等同沒跑，推測可能跟實作時是以亂數且可能會重複的方式取點，所以並不會把那1000點都跑過

- Pocket的實驗結果，有時候錯誤率會高達十幾個點，畫出來的圖會有明顯的錯誤，所以我是在執行多次後才取得錯誤率為3的結果