

tags: Machine Learning

Assignment 1 Readme

File Description

- `functions.py`
 - 撰寫所有題目中用到的函式
- `main.py`
 - 依次執行第一小題至第四小題
 - 每執行完一小題，會在螢幕上印出提示文字，以及顯示結果圖
 - 結果圖會存到同一個目錄底下
- `Q1_random_generate.py`
 - 第一小題的程式碼，可單獨執行第一小題
 - 執行完後會產生 `HW1-1.png`
- `Q2_pla.py`
 - 第二小題的程式碼，可單獨執行第二小題
 - 執行完後會產生三個圖檔：`HW1-2_1.png` `HW1-2_2.png` `HW1-2_3.png`
- `Q3_pocket_pla.py`
 - 第三小題的程式碼，可單獨執行第三小題
 - 執行完後會產生 `HW1-3.png`
- `Q4_mislabel.py`
 - 第四小題的程式碼，可單獨執行第四小題
 - 執行完後會產生 `HW1-4.png`

Execution

Environment

- Python 3.10.6
- numpy: 1.23.5
- matplotlib: 3.6.2

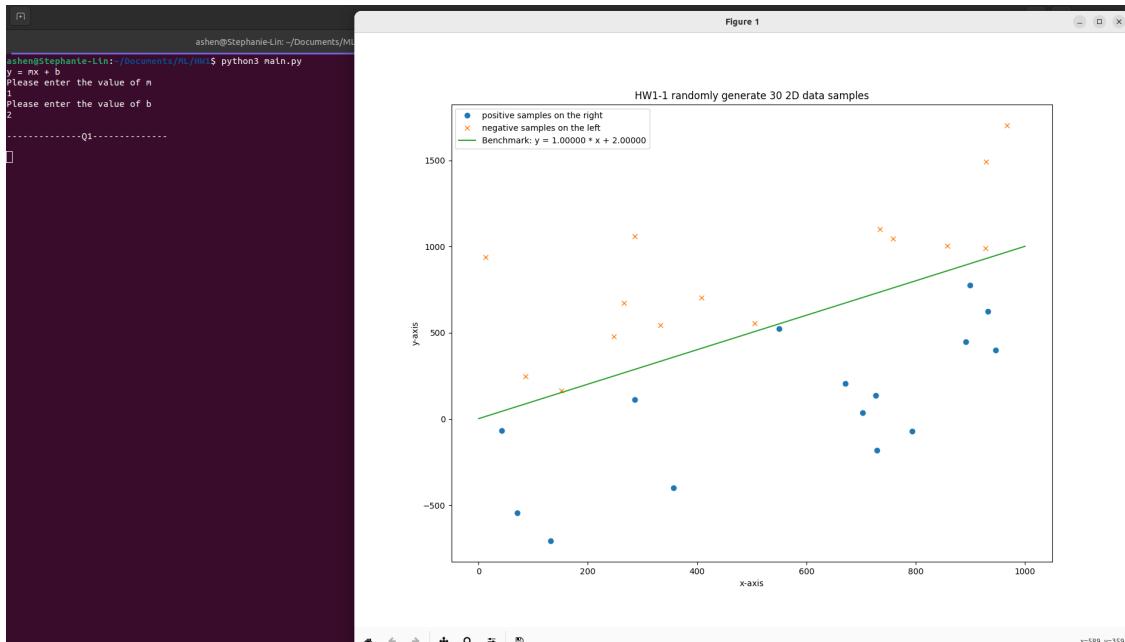
Execute All

- 執行 `main.py`
- 一開始會讓使用者輸入 $y=mx+b$ 中的 m 和 b 的數值

```
ashen@Stephanie-Lin:~/Documents/ML/HW1$ python3 main.py
y = mx + b
Please enter the value of m
1
Please enter the value of b
2
```

- 四個小題都將依照上述輸入的 m 和 b 的值進行計算

- 每執行完一小題，螢幕上會顯示一些提示文字，以及跳出顯示結果圖



- 關閉圖檔之後，顯示 Q[num] is done!，並繼續執行下一小題

```
ashen@Stephanie-Lin:~/Documents/ML/HW1$ python3 main.py
y = mx + b
Please enter the value of m
1
Please enter the value of b
2

-----Q1-----
Q1 is done!

-----Q2-----
y = -0.00441 * x + 0.00409
iteration = 28
Line Correct!
Average iteration is = 9.333
```

Execute Individual

- 預設每一小題的 m, b 為 1, 2

Q1

- 執行 `Q1_random_generate.py`
- 利用 `functions.py` 中的 `random_generate` 函式，來隨機產生並建立資料
- 建立 `samples` 陣列，為 $(1, x, y)$
- 將 x 的大小固定在 0~1000 的隨機亂數（透過 `np.random.uniform`）
- 與 $y = mx+b$ (之後稱之為 benchmark) 的間隔，設定在 1~1000 的隨機亂數

Q2

- 執行 `Q2_pla.py`
 - 利用 `functions.py` 中的 `PLA` 函式來對數據做 PLA
 - 先產生一個亂數生成的整數陣列，陣列大小為數據的筆數（依照題目為 30）
 - 對該陣列做檢查
 - `get_sign()` 取得 $\text{sign}(W^T \cdot x)$
 - 如果 `get_sign() != y[i]`
 - 則更新 $w = w + y[i] \cdot x[i]$
 - 利用 `functions.py` 中的 `verification` 函式來驗證答案
 - 如果是對的，那會在螢幕上輸出 Line Correct
 - 更新迭代次數，輸出圖片和平均迭代次數
 - `Q2_pla.py` 中的 `RUN_PLA` 函式接參數：`m, b, num_sample, times`
 - `m` 和 `b` 為 1, 2
 - `num_samples` 為數據的個數，依照題目為 30，`num_samples = 30`
 - `times` 為做幾次，依照題目會做 3 次，`times = 3`
-

Q3

- 執行 `Q3_pocket_pla.py`
 - 利用 `functions.py` 中的 `PLA` 和 `Pocket` 函式分別對數據做兩種不同的分類
 - 計算時間的部份採用 `time.time()` 取得當下的秒數
 - `Pocket` 函式的操作：
 - `threshold` 為要跑幾次，在此設定跑 10,000 次
 - `con_threshold` 為如果連續第幾次沒有更新 `w` 的值，就停止遞迴，在此設定為 1000 次
 - 每一輪都會產生新的 random 數值 (`rand`)
 - 利用 `get_sign()` 判斷是否正確
 - 利用 `verification()` 分別取的新舊參數的錯誤率
-

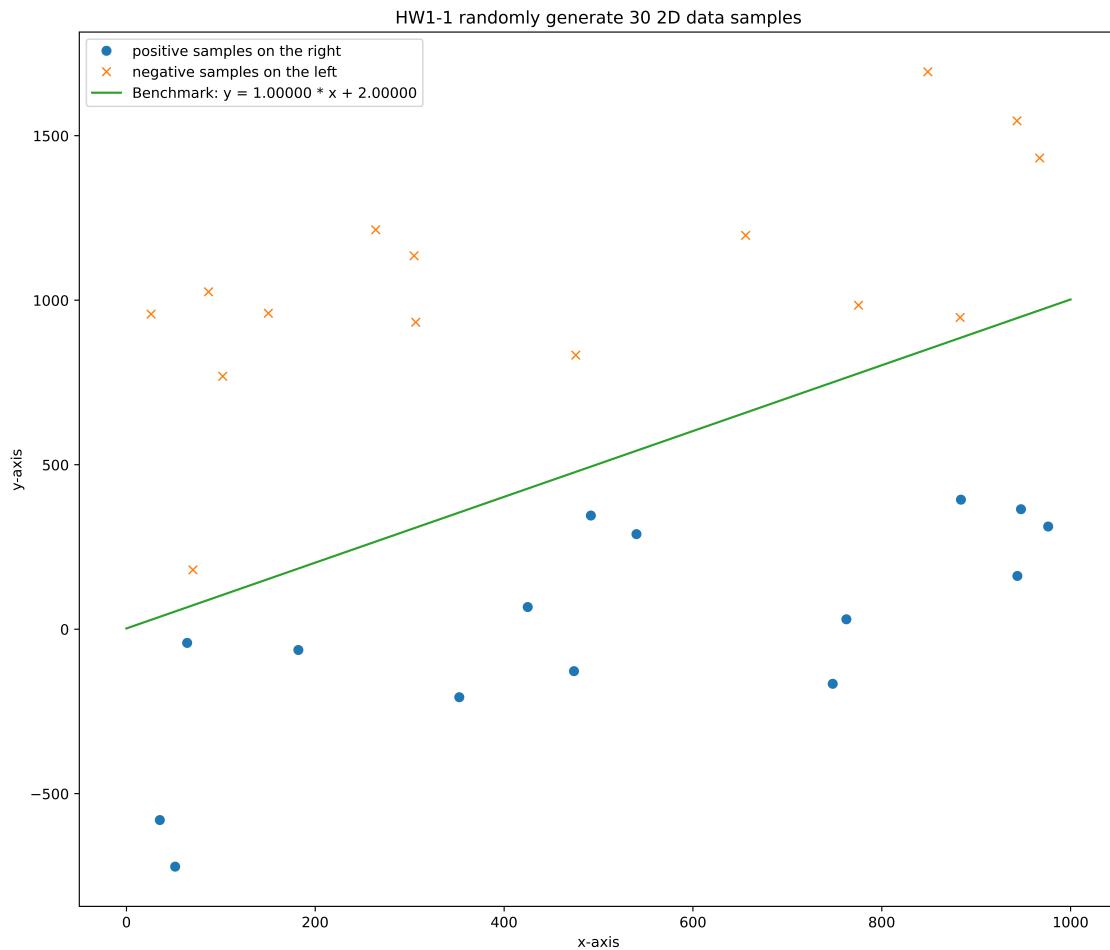
Q4

- 執行 `Q4_mislabel.py`
 - 利用 `Q4_mislabel.py` 中的 `Pocket_with_mislabel` 函式來執行
 - 產生完數據後，利用 `np.random.choice()` 隨機選擇要 mislabel 的 index 值
 - 其餘皆跟 Q3 中執行 Pocket 的步驟相同
 - 最後進行比較，如果 $W_{best} - W_t < W_{best} - W_{best-1}$ ，更新 $W_{best} = W_t$
-

Results & Conclusion

Q1

- 隨機產生數據
- 並分 benchmark 的左邊為負；右邊為正
- 結果圖



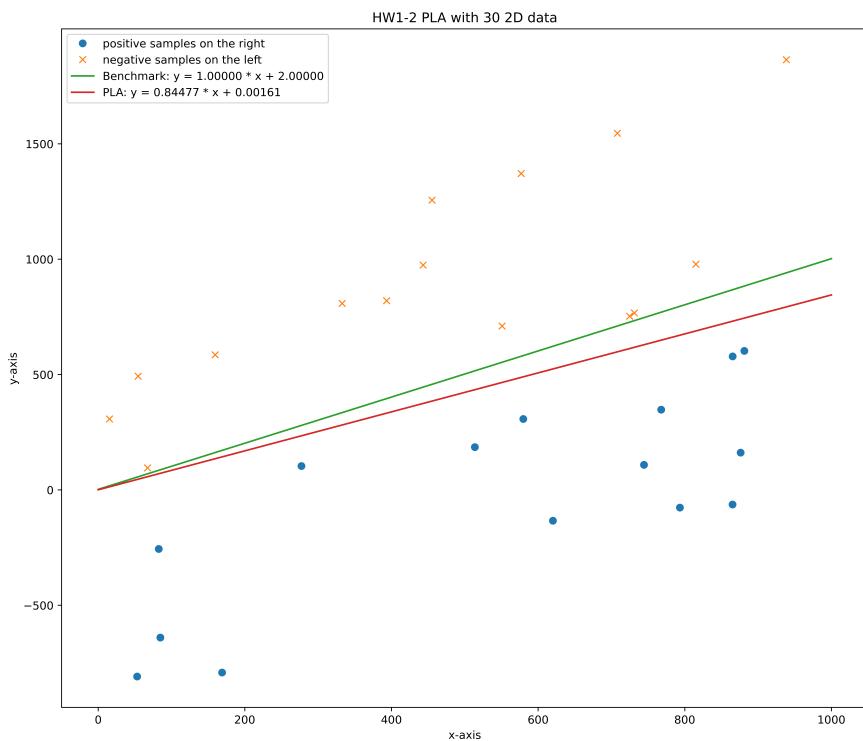
Q2

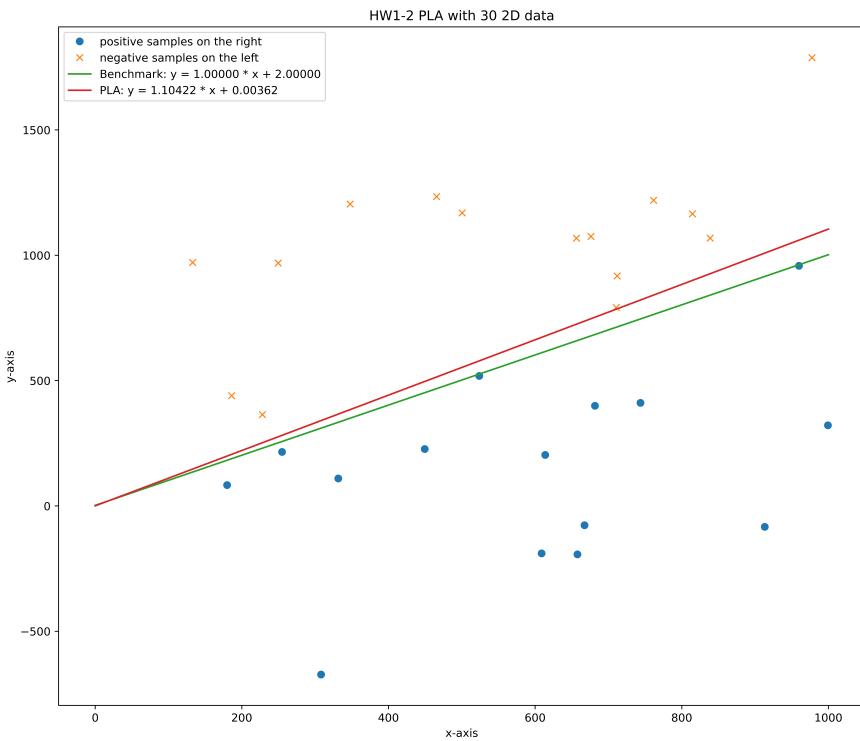
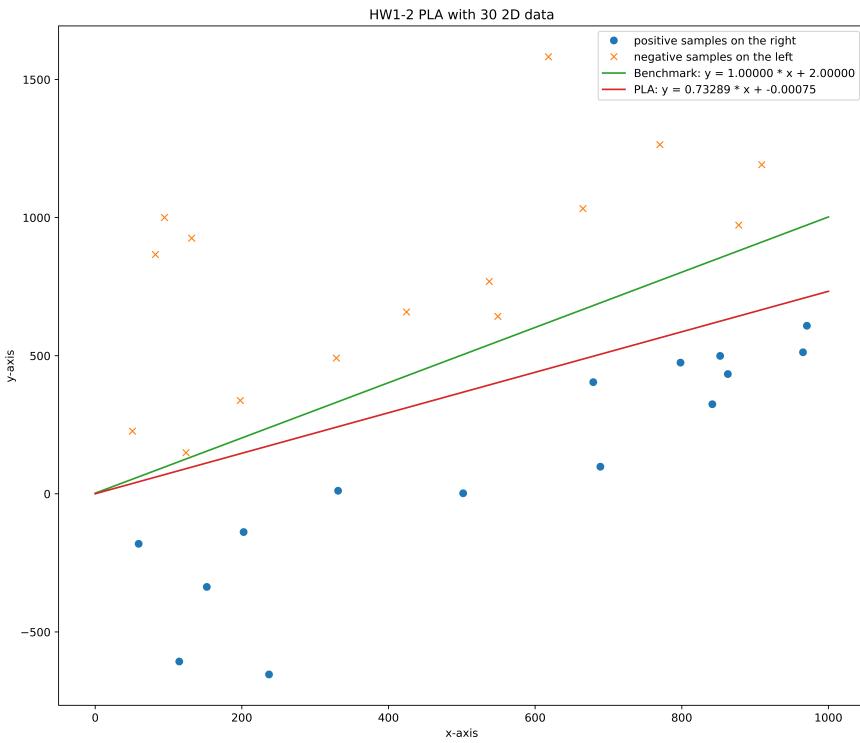
- 利用 PLA 對數據做分割
- 可以拿來和原本的 benchmark 來做比較

- 執行結果

```
-----Q2-----  
  
y = 0.84477 * x + 0.00161  
iteration = 11  
Line Correct!  
Average iteration is = 3.667  
  
y = 0.73289 * x + -0.00075  
iteration = 11  
Line Correct!  
Average iteration is = 7.333  
  
y = 1.10422 * x + 0.00362  
iteration = 15  
Line Correct!  
Average iteration is = 12.333  
  
Q2 is done!
```

- 結果圖





Q3

- 利用 Pocket 對數據做分割
- 和第二題的 PLA 進行比較
 - 執行時間
 - 錯誤率

- 執行結果

```

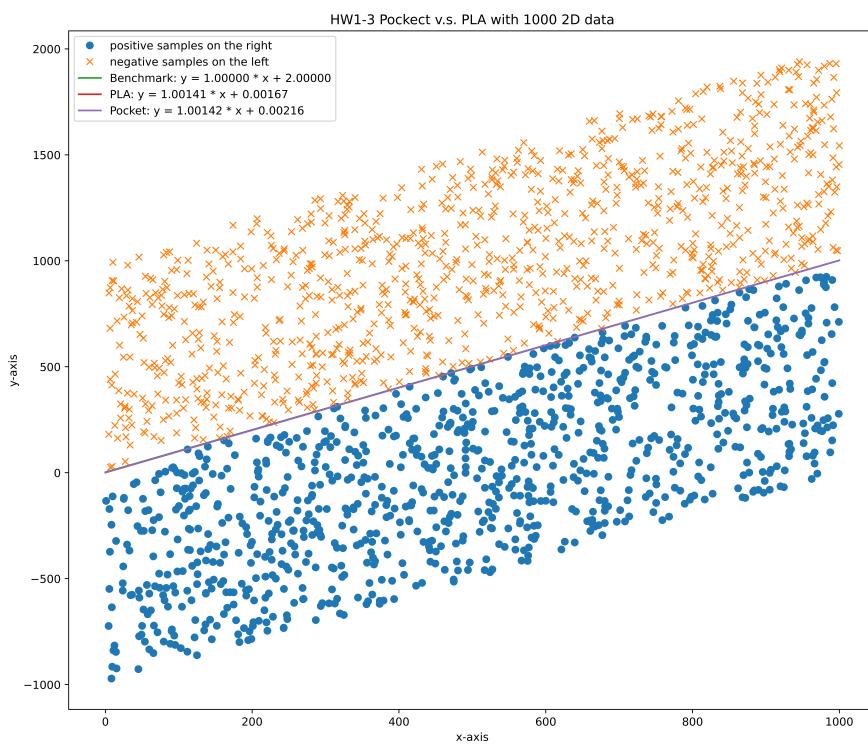
-----Q3-----
data generate
initial figure
PLA done
Pocket done
PLA execution time = 0.02761 sec
PLA iteration = 112
PLA executin time per iteration = 0.00025 sec
PLA error rate = 0.00000

Pocket execution time = 0.84173 sec
Pocket iteration = 120
Pocket executin time per iteration = 0.00701 sec
Pocket error rate = 0.00000

Q3 is done!

```

- 可以看到 PLA 和 Pocket 演算法的執行時間有著明顯的差距
- 推測造成 PLA 和 Pocket 值時間上的差異是因為，Pocket 演算法停止迭代的條件相較 PLA 較為嚴格，Pocket 演算法只要沒滿足條件，就會繼續迴圈遞迴下去
- 結果圖

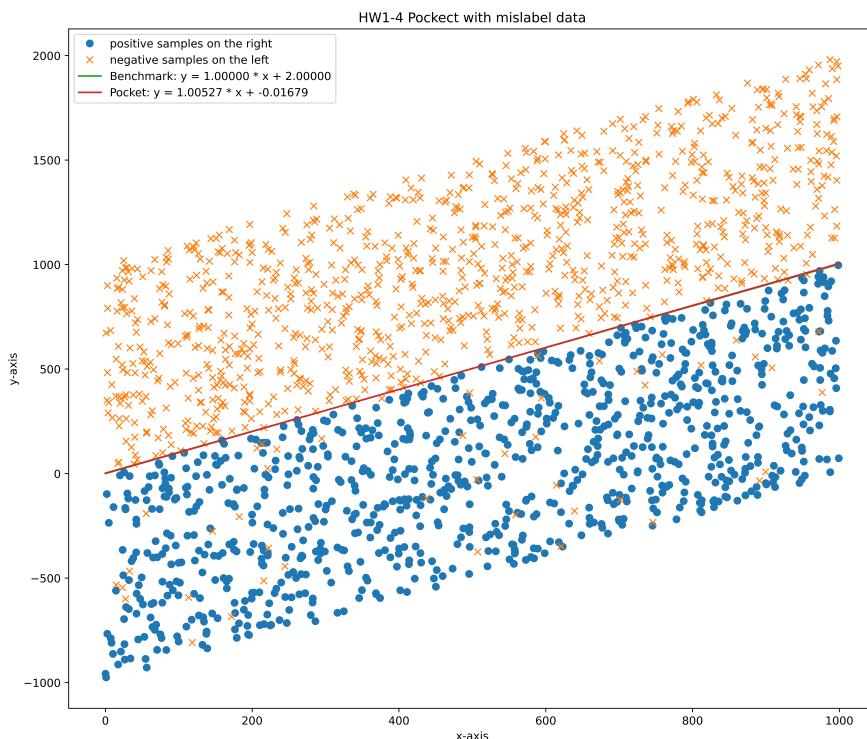


Q4

- 故意標記錯誤的 50 組數據，並再次利用 Pocket 來分割
- 執行結果

```
----- Q4 -----  
data generate  
initial figure  
Pocket execution time = 5.92447 sec  
Pocket iteration = 864  
Pocket executin time per iteration = 0.00686 sec  
Pocket error rate = 0.02400  
  
Q4 is done!
```

- 結果圖



Discussion

- 看到題目的時候，稍微花了一點時間去想整個程式的執行架構要長怎麼樣
 - 考量一：每個檔案都獨立，functions 都寫在各自的檔案中
 - e.g. 取名 q1.py, q2.py, q3.py, q4.py
 - 再寫一個 makefile 執行 python3 filename
 - 考量二：寫一個函式檔案，將所有用到的函式寫進去，再寫一個 main 函式，可以統整所有題目
 - 後來覺得考量二似乎會比較貼近大型專案時的規劃需求，因此採用考量二的方式去實現
- 整體來說，我覺得程式部份比較難的地方是視覺化數據的部份
 - 因為我只打算寫一個製圖的函式，可以套用到所有小題的圖
 - 想了一下要怎麼去定義 class 的架構，還有分析需要用到的參數