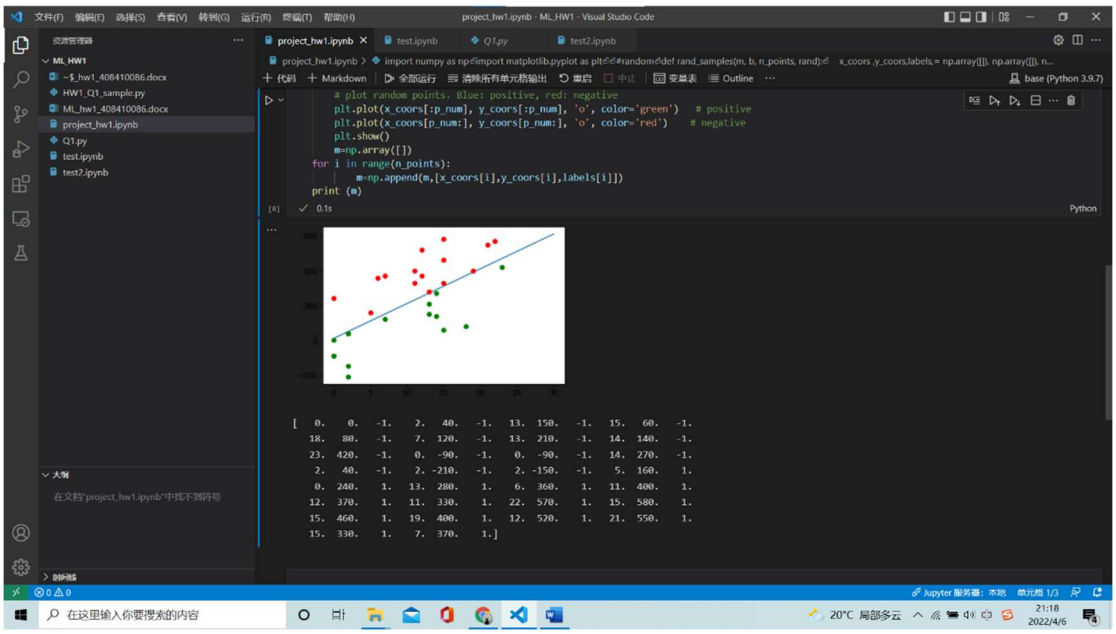
Report

常博愛 408410086 資工三

1. 執行描述：先設置 random sample（），然後畫出已知的直線 y=mx+b，之後再根據直線 y 的參數代入 random 函式，最後 output 每個點的坐標與對應的 label 值。

   執行結果：如圖

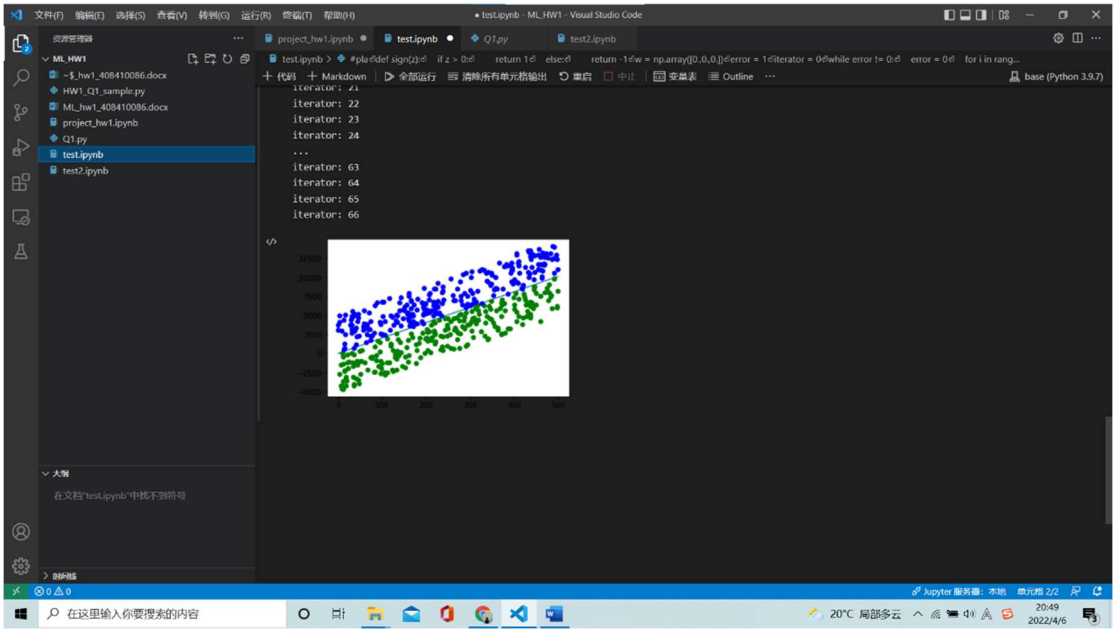


2. **執行描述**：

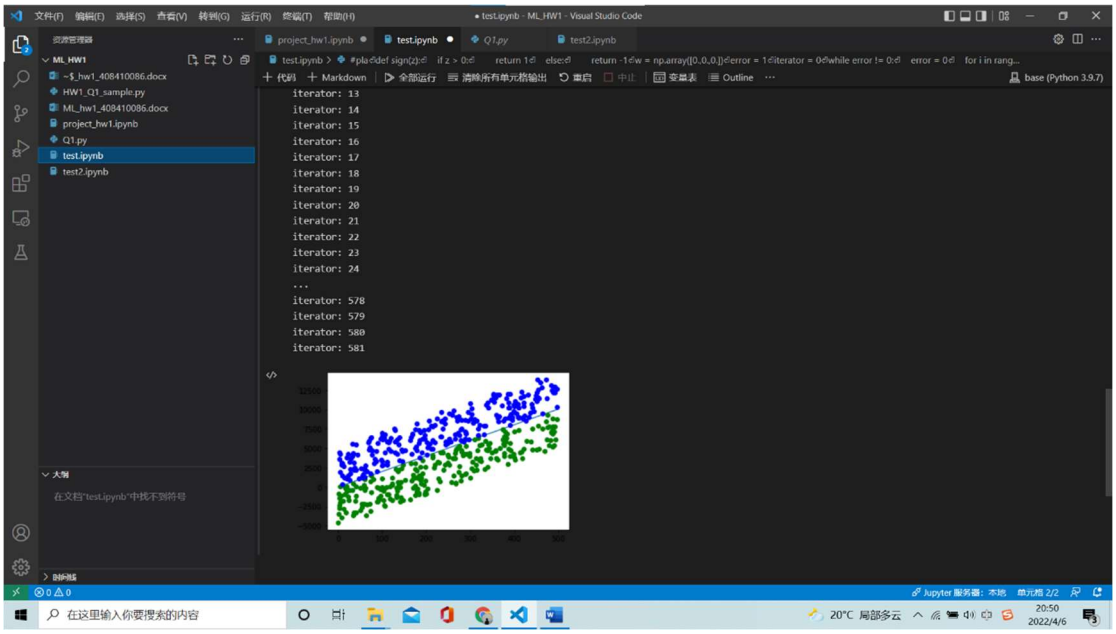
   首先產生隨機亂數點共 500（pos250,neg250）,然後撰寫激勵函數 sign（），如果 sign（z）>0;return 1,else -1.之後開始迴圈由 w0=[0,0,0]開始遍歷每個點 x 計算 w0*x 的內積，如果 y 是 positive 就相加，否則就相減。如果 sign 的結果與 y 相同，跳出迴圈，結束程式，畫出圖像。

   執行結果：以 500sample 為例：average=（66+581+581）/3=409 次

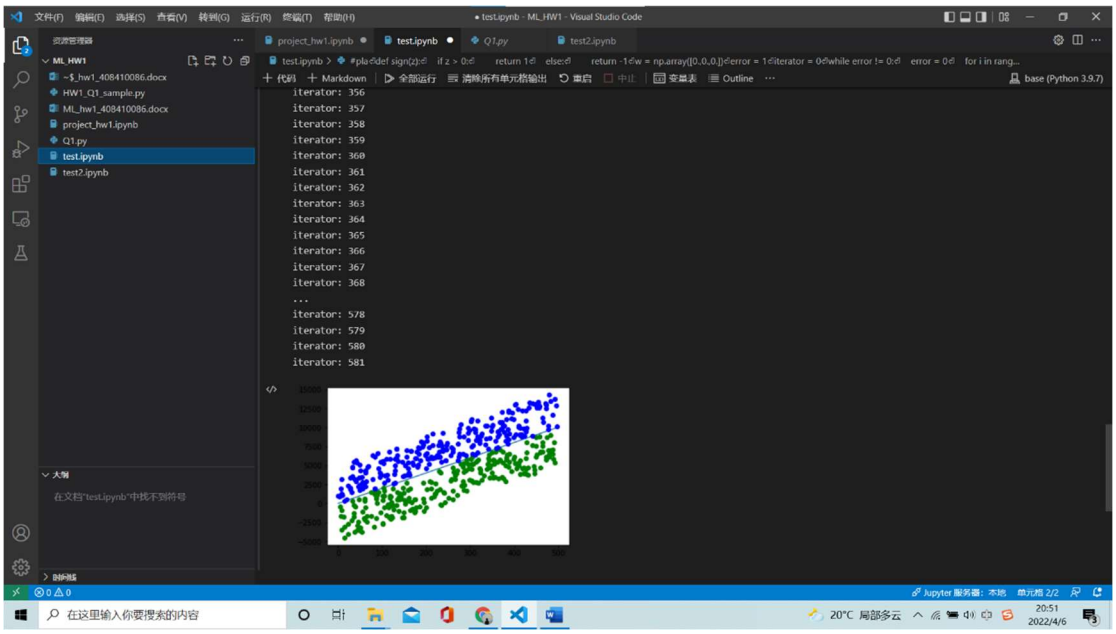
   第 一 次 ：



第　　二　　次　　：



第　　三　　次　　：

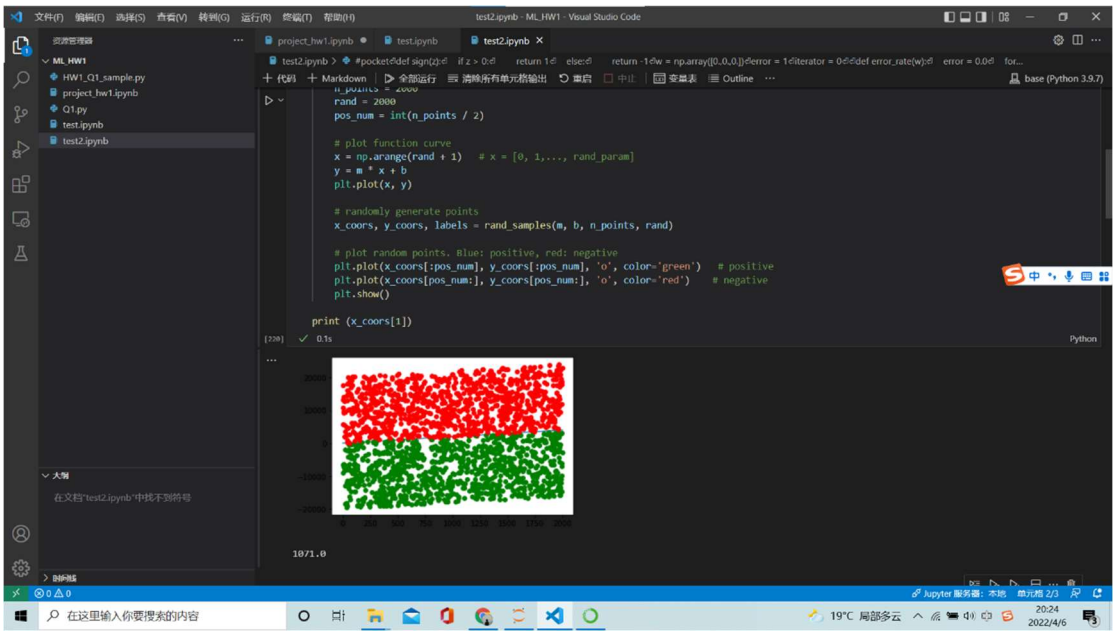


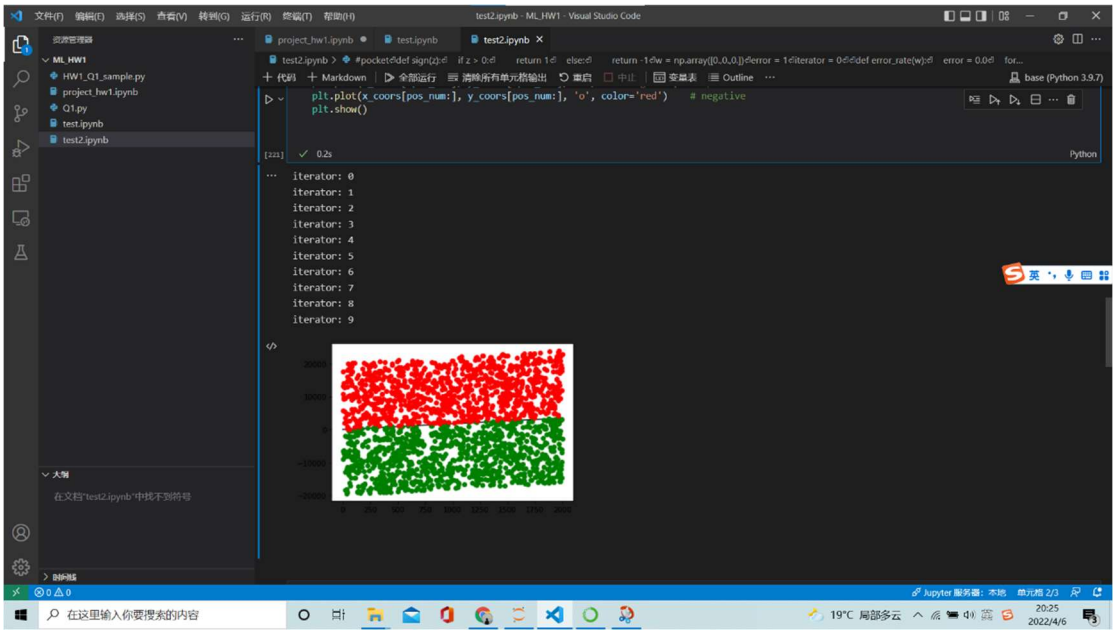
總結與討論：PLA 每次迭代結果都不會對後面產生影響，即 train 結果不是收斂的，可能需要多次迭代才能實現分類，當 data 量過大時，不適用。

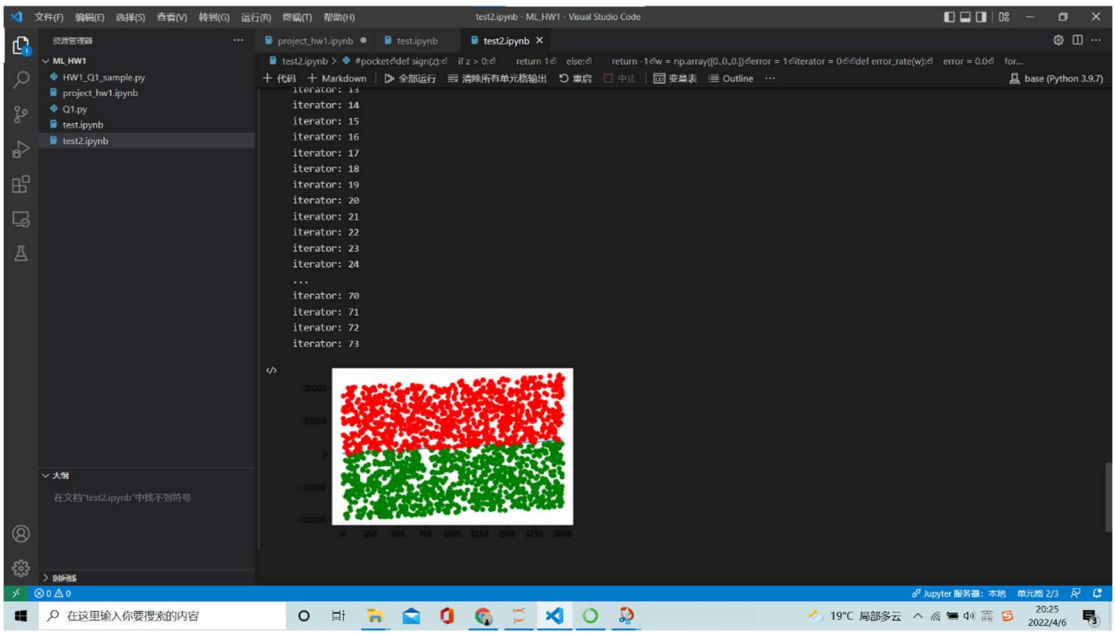
3. 執行描述：

框架基本與 PLA 一致，只是多撰寫了一個 error () function，如果 weight (new) 的 error 小於 weight（old），就更新 weight，並跳出 function。

執行結果：如圖可知，該次實驗 生成隨機點時間為 0.1s；pocket 執行時間為 0.2s,迭代 9 次；PLA 執行時間為 0.3s,迭代 73 次。


test2.ipynb - ML_HW1 - Visual Studio Code
ML_HW1
HW1_Q1_sample.py
project_hw1.ipynb
Q1.py
test.ipynb
test2.ipynb
rand = 2000
pos_num = int(n_points / 2)
# plot function curve
x = np.arange(rand + 1) # x = [0, 1,..., rand_param]
y = m * x + b
plt.plot(x, y)
# randomly generate points
x_coors, y_coors, labels = rand_samples(m, b, n_points, rand)
plot random points. Blue: positive, red: negative
plt.plot(x_coors[:pos_num], y_coors[:pos_num], 'o', color='green') # positive
plt.plot(x_coors[pos_num:], y_coors[pos_num:], 'o', color='red') # negative
plt.show()
print (x_coors[1])
1071.0


test2.ipynb - ML_HW1 - Visual Studio Code
ML_HW1
HW1_Q1_sample.py
project_hw1.ipynb
Q1.py
test.ipynb
test2.ipynb
plt.plot(x_coors[pos_num:], y_coors[pos_num:], 'o', color='red') # negative
plt.show()
iterator: 0
iterator: 1
iterator: 2
iterator: 3
iterator: 4
iterator: 5
iterator: 6
iterator: 7
iterator: 8
iterator: 9



總結與討論：由於我本次實驗只顯示出最後一次分類的圖像，所以計算量要簡化許多，時間也減少很多，因此在共 2000 個 sample 的情況下時間依舊控制在很小的範圍；兩種演算法的時間差異也很小，但是可以根據迭代次數推斷，pocket 的效率是高於 PLA 的，如果把每次迭代的結果都顯示出來，勢必 PLA 要花費更多時間，但無奈電腦算力不夠，無法執行題述的資料量。

Difficulty：公式的實現，可視化的處理，形態格式的轉換，套件不熟悉
Note：code 之前為 ipynb 檔，後改為.py