Machine Learning Assignment #1

408410056 資工三 許庭涵

Execution description:

需在有**python & pip**的環境下執行，可利用以下指令檢查

python:

```
py --version
```

> 正常應該會顯示 Python 3.X.X

pip:

```
pip --version
```

> 正常(最新版)應該會顯示 pip 22.X.X from ~ (python 3.X)

接著需安裝函式庫

```
py -m pip install -U numpy matplotlib
```

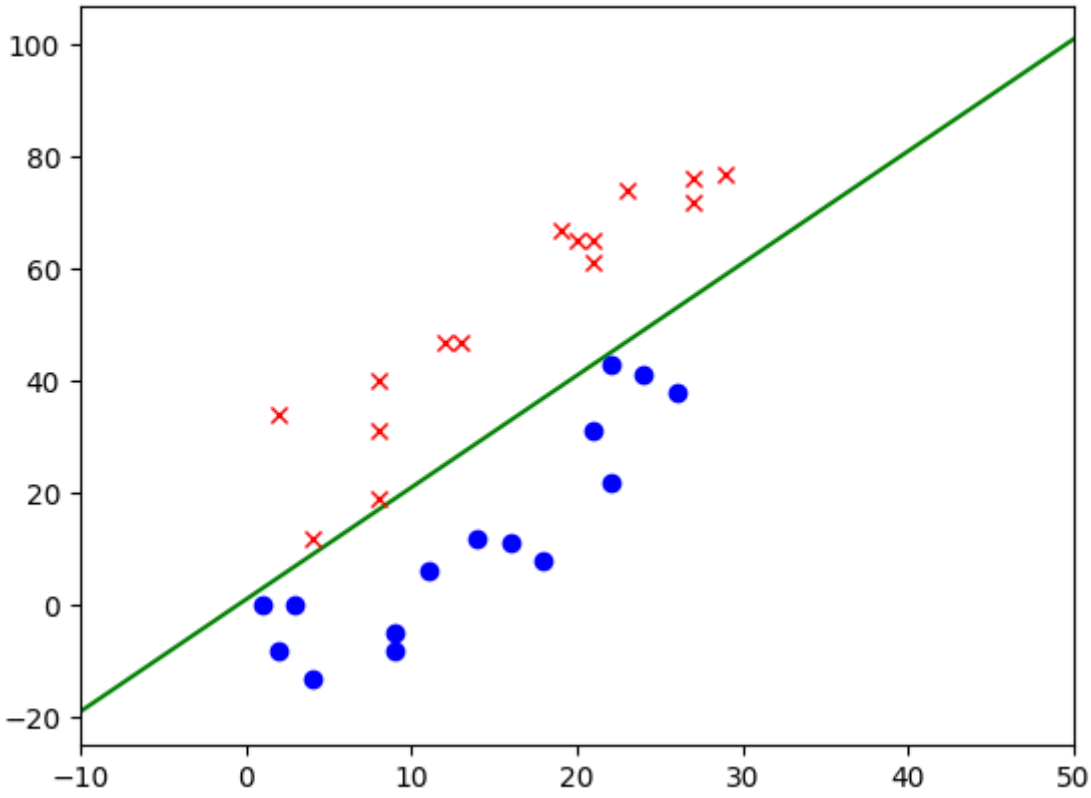
完成後即可開始執行以下程式了:

- **第一題 & 第二題** 執行 project1.py
- **第三題** 執行 project1_2.py
- **第四題** 執行 project1_3.py

Experimental results

1. 需依方程式y = mx + b ， 於此線左右產生負數據與正數據各15個，並不能壓在線上

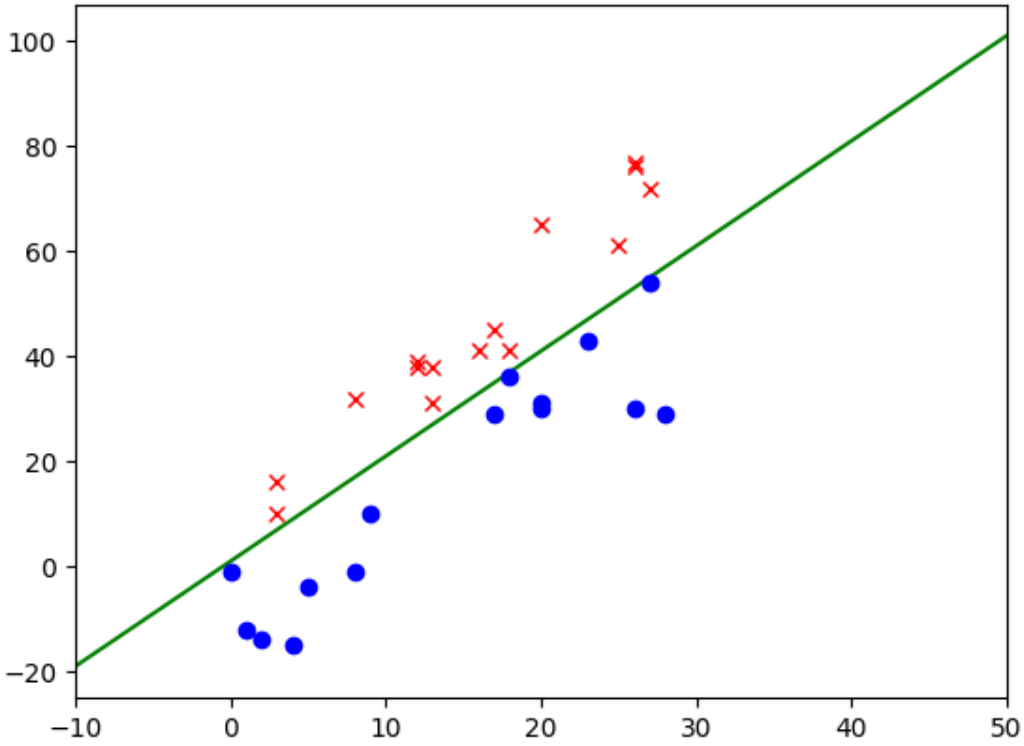
- **y = 2x + 1** 綠線為方程式 y = 2x + 1 ， 於其線左邊為15個負數據，以紅色 "X"做標記，右邊為15個正數據，以藍色 "O"做標記



2. 自訂初始 W 向量 *W0* 為(0,0,0)，並依據第一題產生數據的方法實驗三次PLA找出分類線，並計算三次PLA迭代次數的平均

- **實驗A**

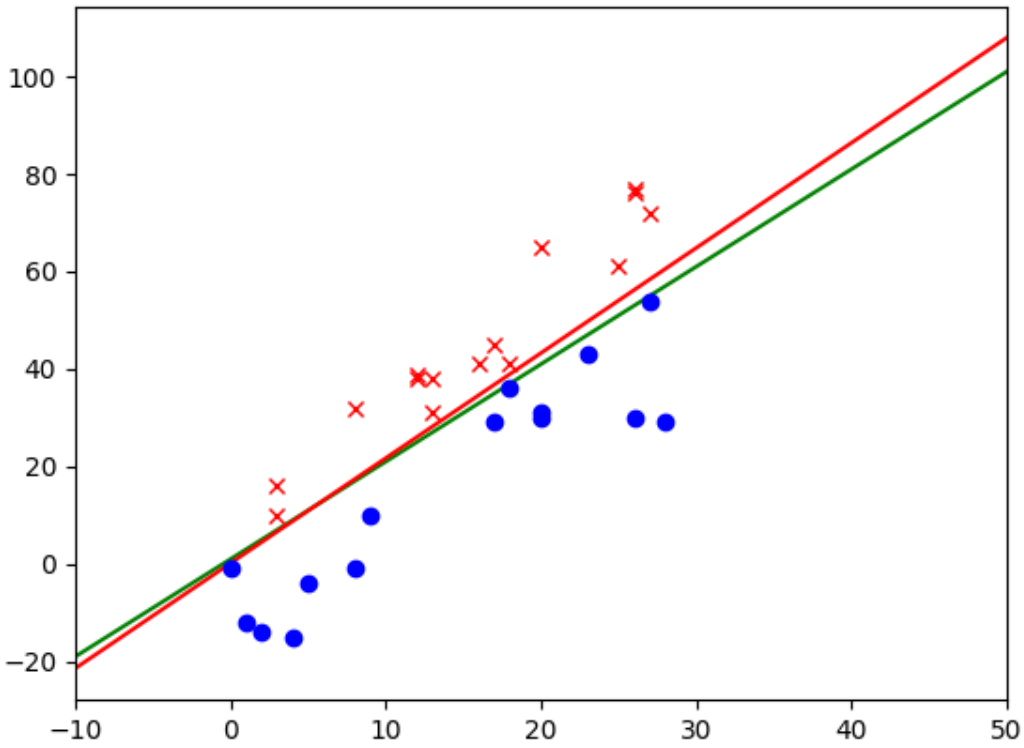
- 由 y = 2x + 1 產生samples



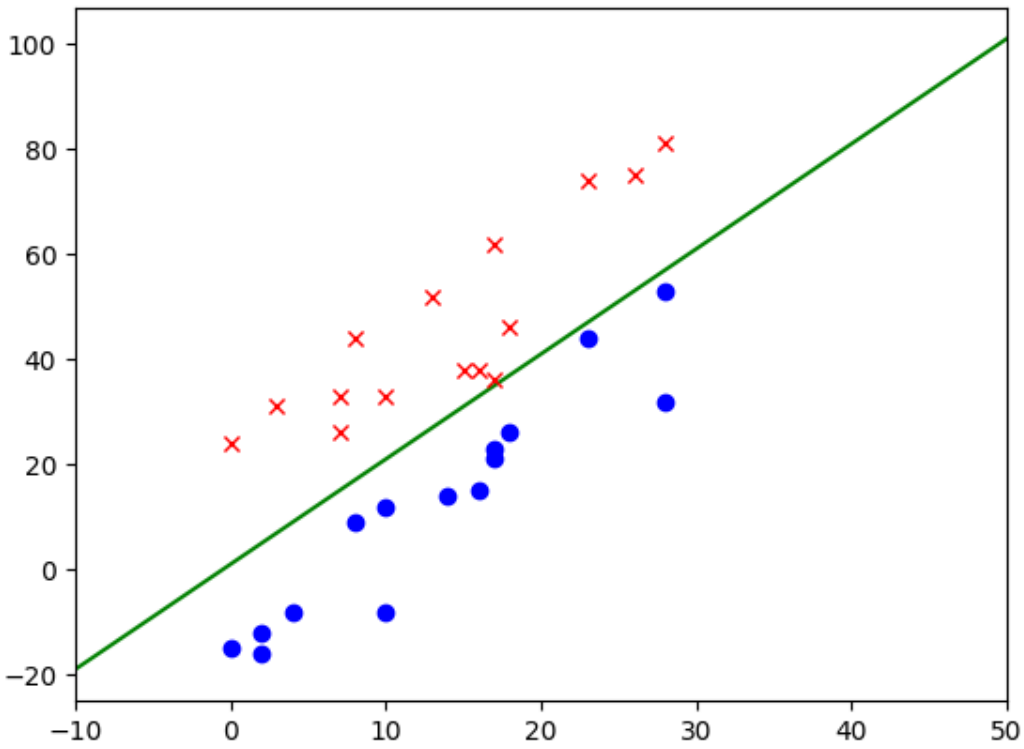
- 實驗結果(迭代次數、PLA找到的分類線)

```
PS D:\3_third_grade\ML_2021> python .\project1.py
original:
y = 2x + 1
iteration: 21
y = 2.15625x + 0.15625
```

- 紅線為PLA找到的分類線



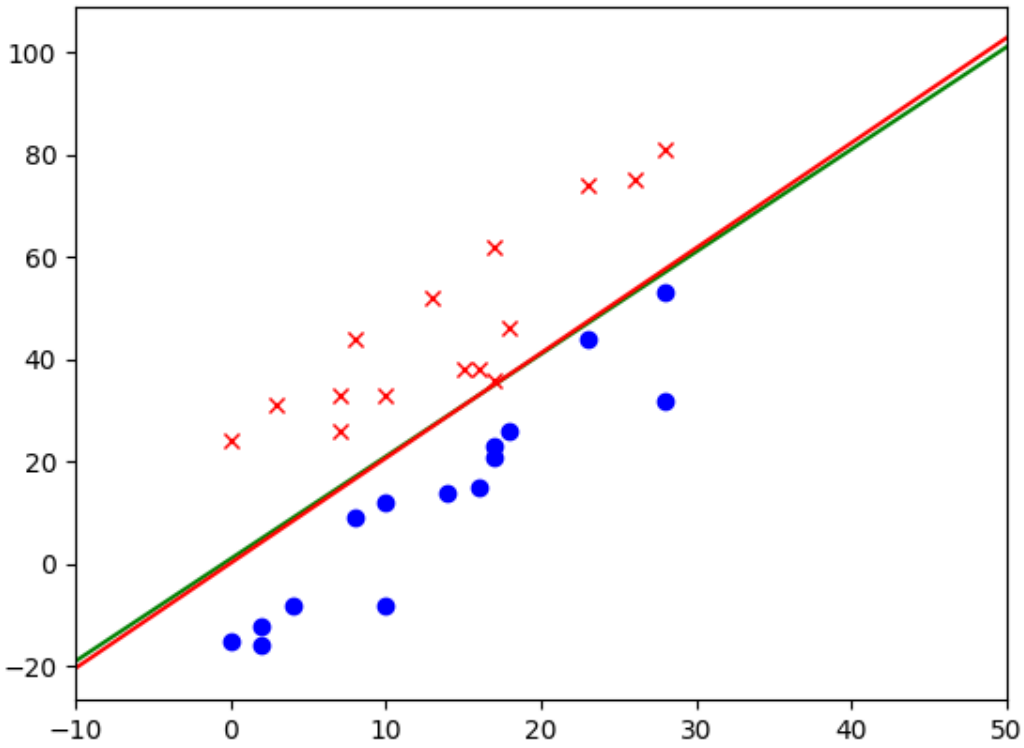
- **實驗B**
  - 由 y = 2x + 1 產生samples



- 實驗結果(迭代次數、PLA找到的分類線)

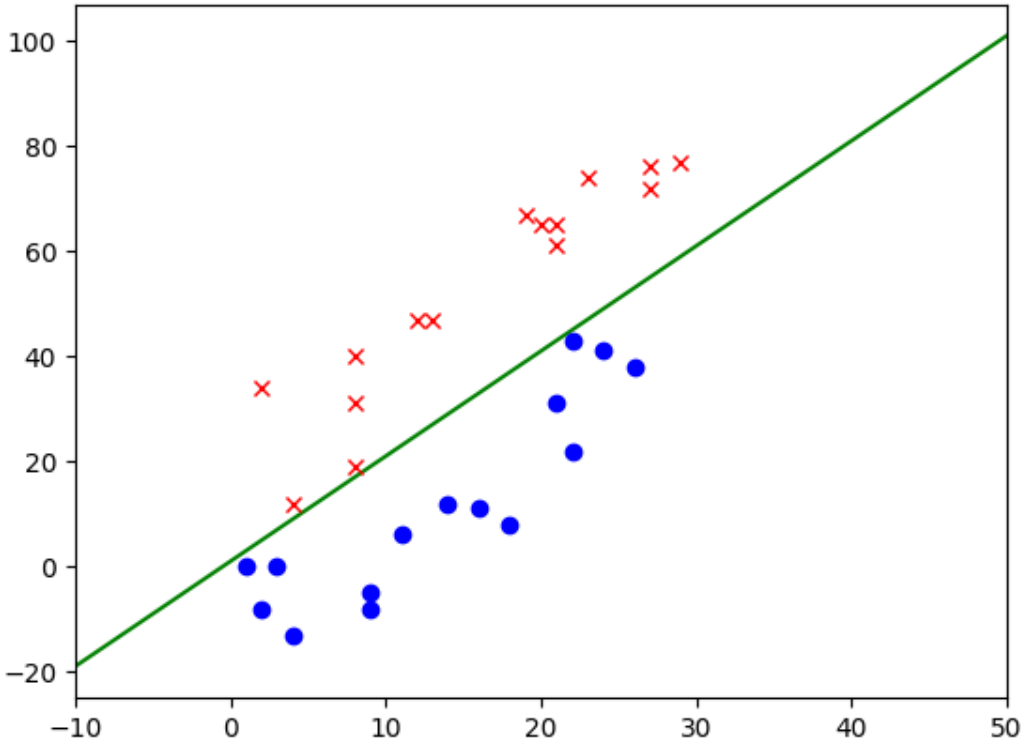
```
PS D:\3_third_grade\ML_2021> python .\project1.py
original:
y = 2x + 1
iteration: 7
y = 2.0526315789473686x + 0.15789473684210525
```

- 紅線為PLA找到的分類線



- **實驗C**

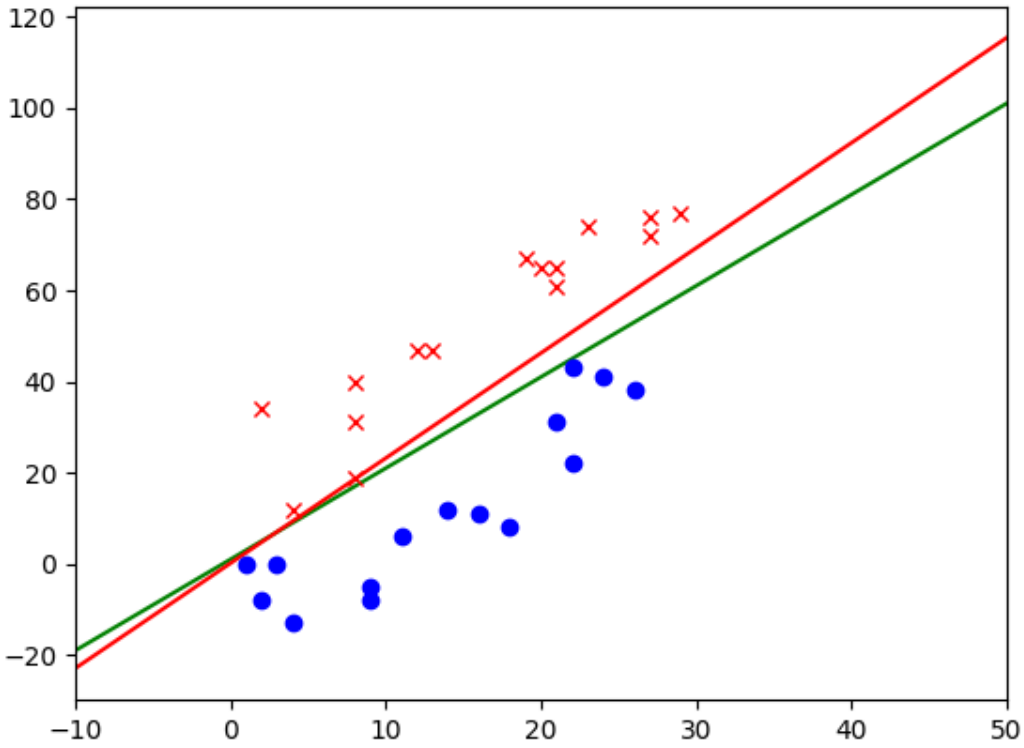
- 由 y = 2x + 1 產生samples



- 實驗結果(迭代次數、PLA找到的分類線)

```
PS D:\3_third_grade\ML_2021> python .\project1.py
original:
y = 2x + 1
iteration: 29
y = 2.305084745762712x + 0.15254237288135594
```

- 紅線為PLA找到的分類線



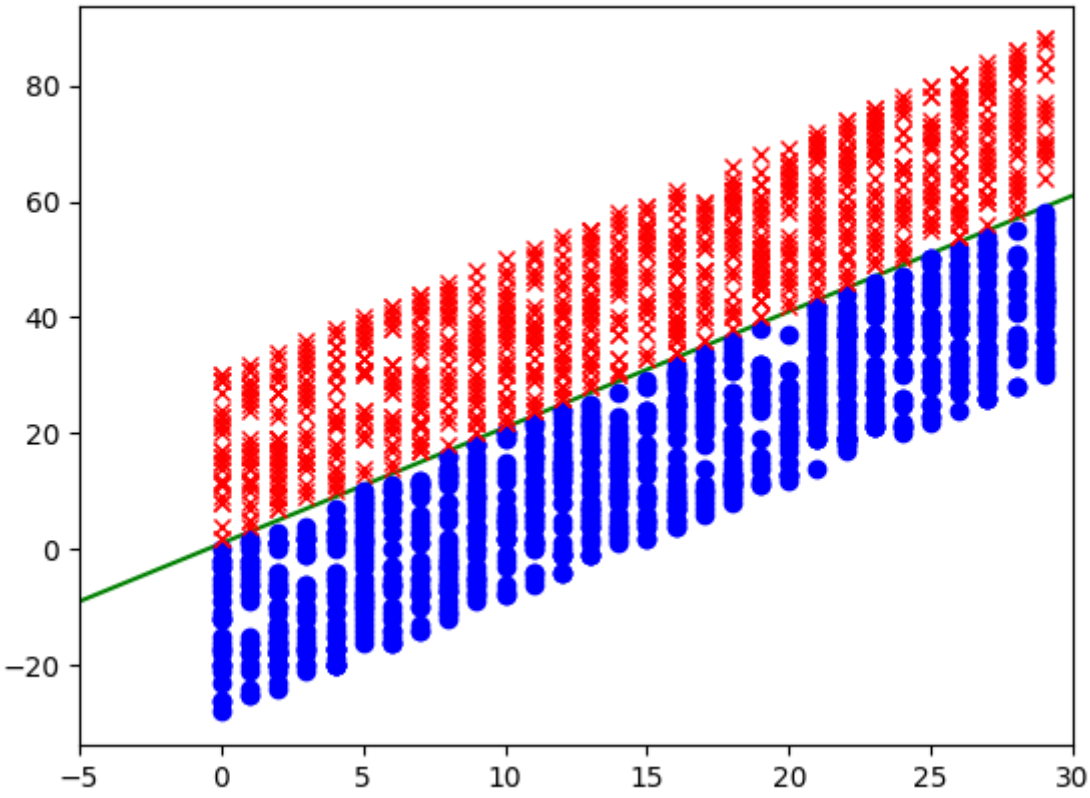
迭代次數的平均

```
Avg = ( 21( A )+7( B )+29( C ) ) / 3 = 19
```

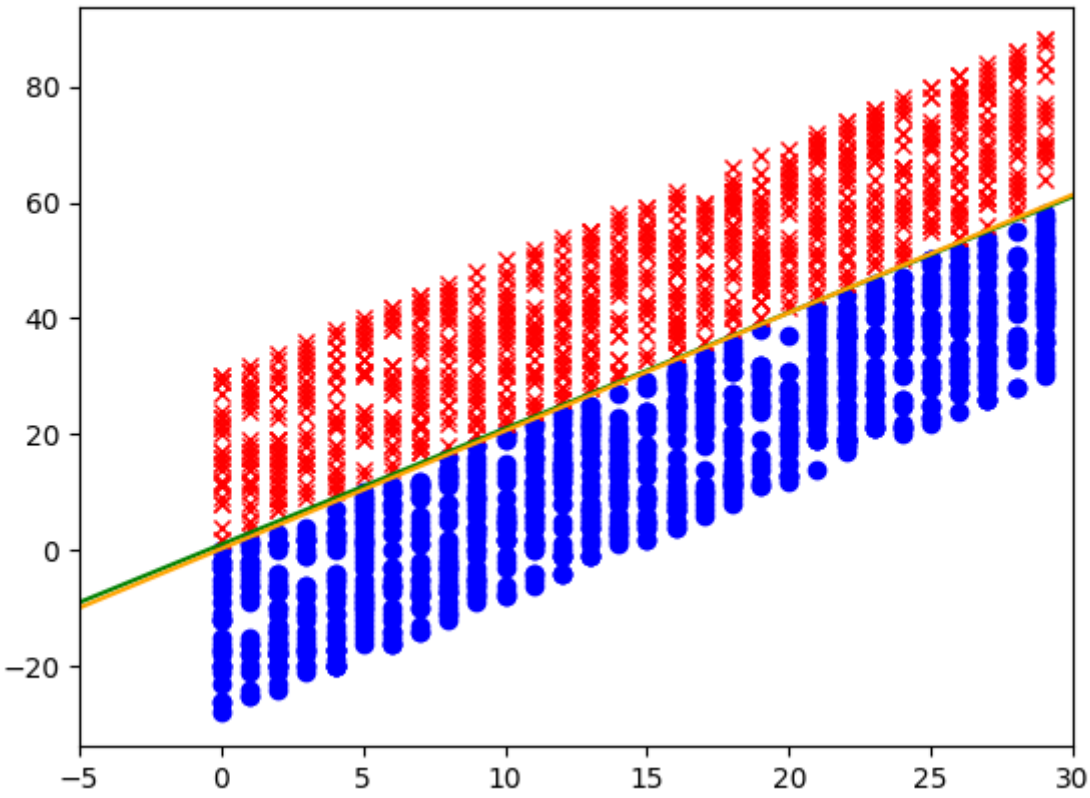
3. 參考第一題，產生1000個負數據與1000個正數據，並實驗PLA與POCKET Algorithm，比較兩執行時間差異

※由於POCKET Algorithm執行一定的iteration後會進入循環使W不會替換(於conclusion解釋)，因此以**1000 iterations為上限**

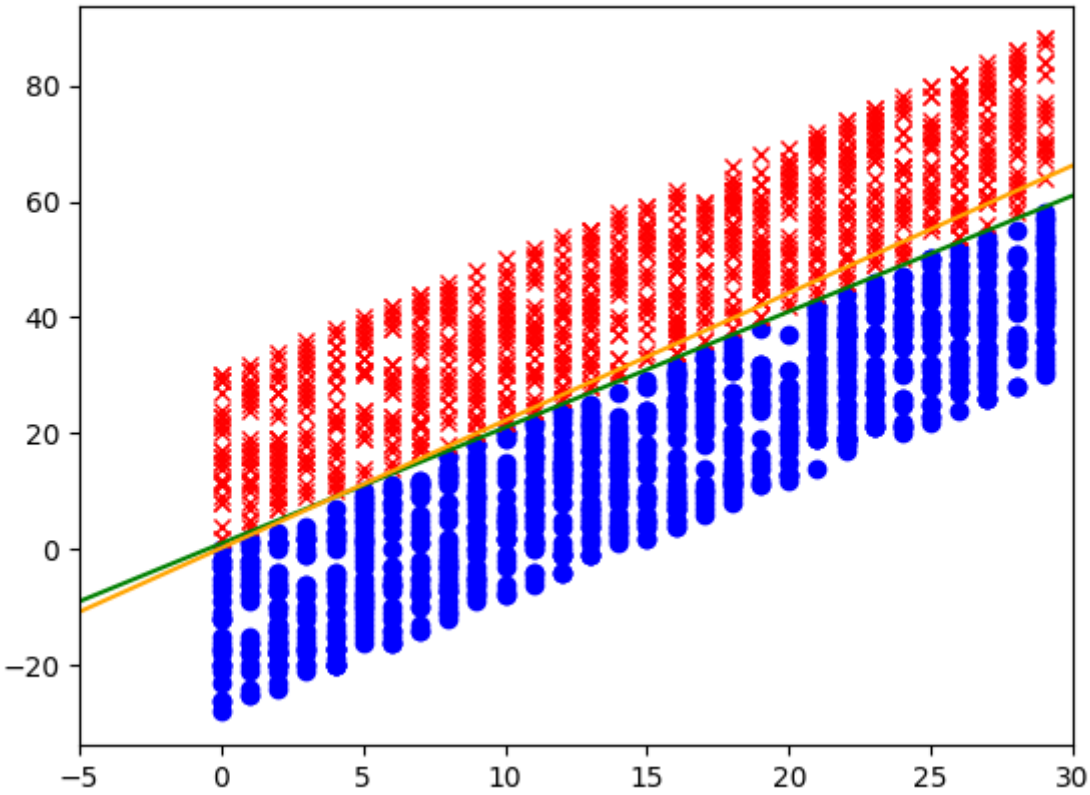
- **由 y = 2x + 1 產生samples**



- 黃線為**PLA** 找到的分界線



- 黃線為**POCKET 找到的分界線**



- **實驗結果**

```
PS D:\3_third_grade\ML_2021> python .\project1_2.py
original:
y = 2x + 1
PLA exe_time: 0.06101059913635254s
PLA_iteration: 23
y = 2.0357142857142856x + 0.25
POCKET exe_time: 12.290933609008789s
POCKET_iteration: 1022
POCKET_accuracy: 0.9755
y = 2.1999999999999997x + 0.19999999999999996
```

PLA與POCKET Algorithm差距:

```
12.229923秒
```

主因為iteration比PLA多上不少，其次為POCKET每經一次iteration皆須確認新的W'是否比舊的錯誤率更低，所以**需多跑一個for loop**確認所有點的label與sign(w•x)是否相同

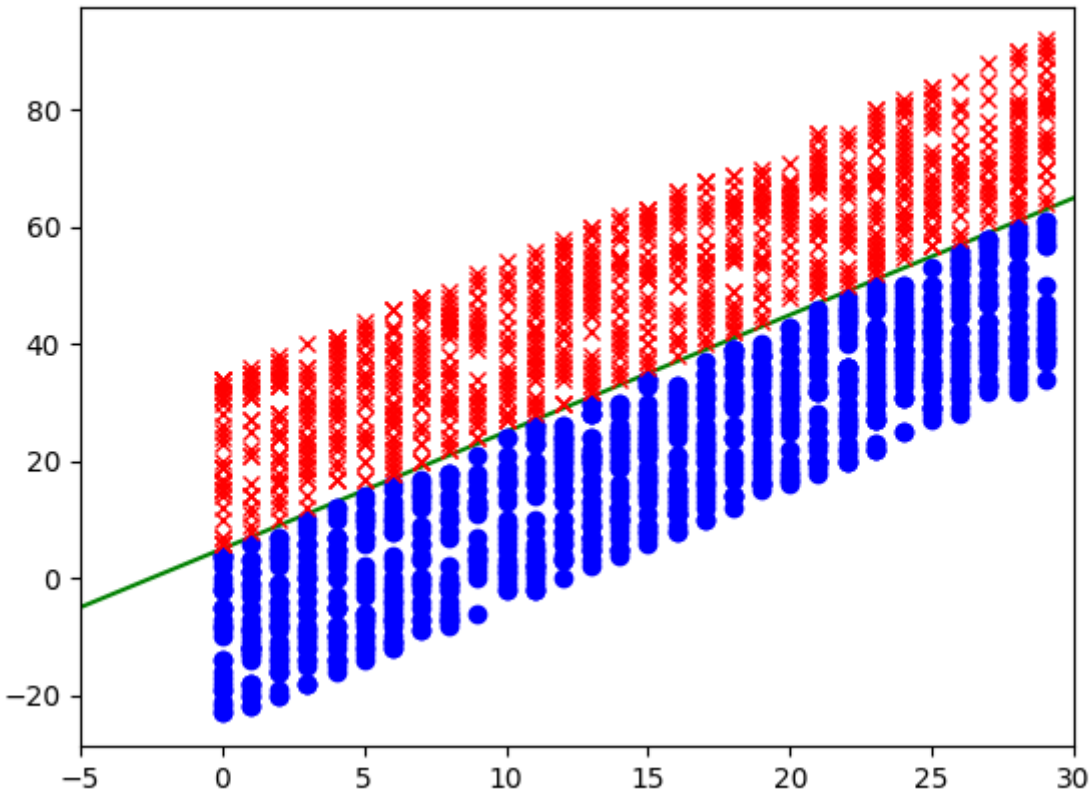
此外若無設定POCKET上限的話，POCKET會因找不到更低錯誤率的W'而跳不出迴圈，POCKET執行時間可能會無上限(永遠找不到分界線)

但也有例外，幸運的找到對的 W ，不過依舊花了較長的時間才完成，POCKET較PLA多花了1.65秒

```
PS D:\3_third_grade\ML_2021> python .\project1_2.py
original:
y = 2x + 1
PLA exe_time: 0.012965679168701172s
PLA_iteration: 8
y = 2.0384615384615383x + 0.23076923076923078
POCKET exe_time: 1.6603825092315674s
POCKET_iteration: 193
POCKET_accuracy: 1.0
y = 2.0526315789473686x + 0.15789473684210525
```

4. 參考第三題，並將產生的數據刻意標錯50個正數據與50個負數據，實驗POCKET Algorithm在標錯前與後正確率差異

- **產生samples**



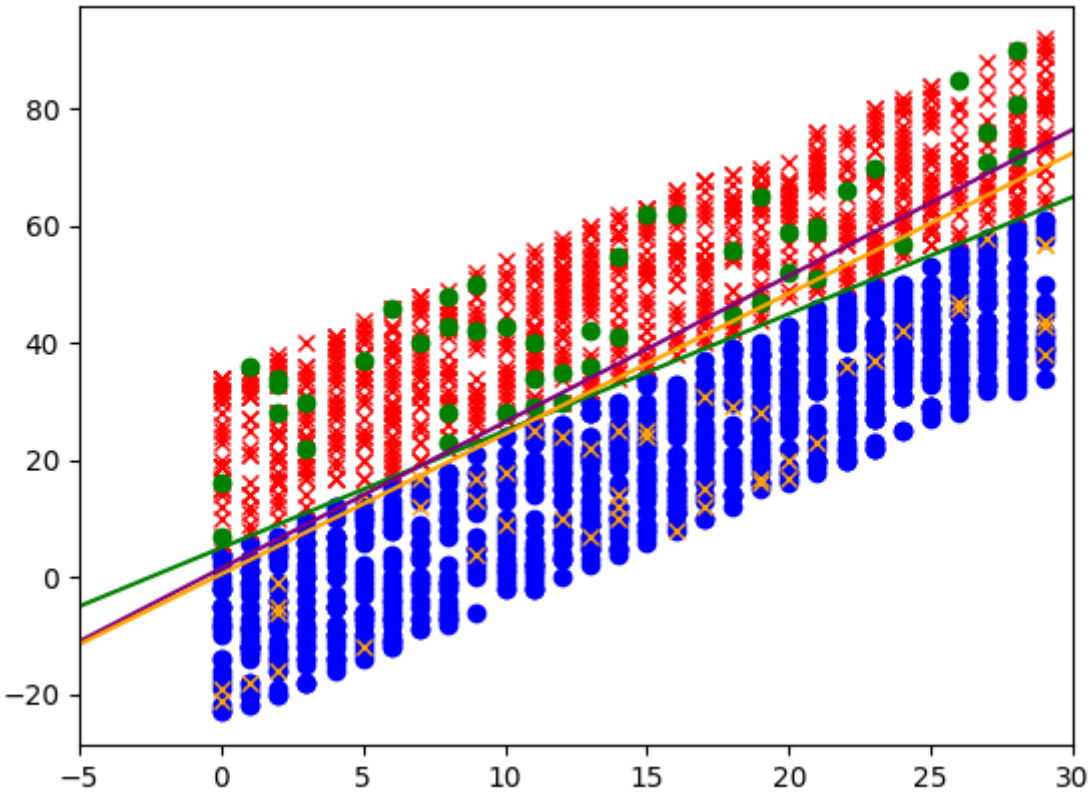
- **Correct Label & Mislabel產生分界線**

綠線 -- 產生samples的方程式 y = 2x + 5

黃線 -- Correct Label

紫線 -- Misabel

綠色"O"與黃色"X"為mislabel的點



- **實驗結果**

```
PS D:\3_third_grade\ML_2021> python .\project1_3.py
POCKET MISLABEL START
mislabel_loop: 1000
label_mistake: 144
label_accuracy: 0.928
mislabel_mistake: 232
mislabel_accuracy: 0.884
mislabel_equation:
y = 2.5x + 1.5
POCKET CORRECT LABEL START
Correct_label_loop: 1000
Correct_label_accuracy: 0.9575
Correct_label_equation:
y = 2.4x + 0.5
```

正確率比較

```
Mislabel 正確率為 0.928

Correct label 正確率為 0.9575
```

證實mislabel 100個samples後會影響POCKET Algorithm判斷，然而影響不大

值得一提的是，Mislabel端用標錯的label去計算正確率比用正確的來的低 (0.884 vs 0.928)

Conclusion

第二題

iteration次數會跳蠻多的，第一個找到label錯誤的點產生的W若與正確分界線接近的話，會非常快找到如第二題實驗B

不幸的話可能找非常久，以 y = 2x + 1 來說，最多曾遇過近百次

另外實驗了多次 m 與 b 的組合，發現PLA iteration次數"基本上"會與b成正比，如圖:

```
PS D:\3_third_grade\ML_2021> python .\project1.py
original:
y = 2x + 30
iteration: 3305
y = 2.0652173913043477x + 28.108695652173914
PS D:\3_third_grade\ML_2021> python .\project1.py
original:
y = 2x + 100
iteration: 547568
y = 2.6881533101045294x + 94.35540069686411
```

而會這樣的原因為W初始為(0,0,0)，若分界線未經(0,0)的話，W需移動較多次才會到分界線附近

另外產生的分界線看起來會很像跟點重疊，實際放大後發現沒有，僅是產生的圖小導致點與線看起來重疊

第三題

> Pocket Algorithm在每次更新wt = w + yx時，都會去比較wt跟w的正確率哪一個比較高。
>
> 若是wt的正確率較高，則 w = wt。

(from https://medium.com/@bob800530/pla-%E7%BA%8C-pla-pocket-92a178a9f0a4)

實驗後發現一個狀況，W有可能因找不到錯誤率更低的wt而一直無法更新導致無法找到分界線跳出迴圈

因此實驗時須加上限給POCKET Algorithm，否則會消耗非常多時間但結果因W早於前幾百次軼代後就無更新而與前幾百次iteration的結果相同

此外也因**強制跳出迴圈 & W無法更新** 使得POCKET並未找到分界線，所以POCKET的正確率會較低，且正確率差異極大，幸運的話可以來到90%以上不幸的話可能60%左右，與找到的W有關

以執行時間比較，POCKET Algorithm會較PLA執行時間多，原因除了POCKET會有無法更新W而跳不出迴圈的問題，導致iteration增加外，其每次計算wt皆須再檢視每個點所以會多一個迴圈時間

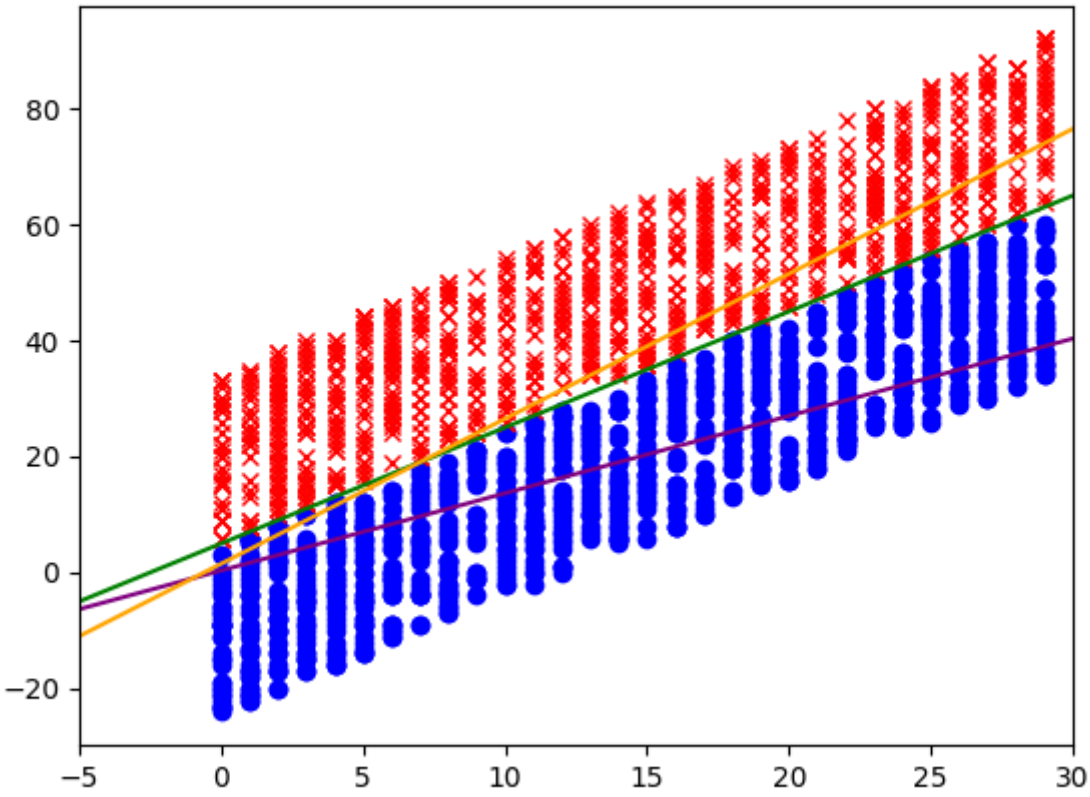
結論: POCKET較適合尋找非線性分類，以近似的方式正確分類大部分的samples

第四題

實驗為刻意標錯50個正數據，50個負數據，實驗後發現差距並不大，可能是以有2000個samples相比，100個標錯影響並不明顯，正確率只差近3%

因此另外實驗了若各標錯500個，如圖:

- **紫色線為Mislabel後的結果**



- **實驗結果圖:**

可以看出Correct_label比Mislabel正確率多上70%左右，另外Mislabel端用標錯的label去計算正確率會比正確的label高上近30%

```
PS D:\3_third_grade\ML_2021> python .\project1_3.py
original:
y = 2x + 5
POCKET MISLABEL START
mislabel_loop: 1000
label_mistake: 1534
label_accuracy: 0.23299999999999998
mislabel_mistake: 978
mislabel_accuracy: 0.511
mislabel_equation:
y = 1.3333333333333333x + 0.3333333333333333
POCKET CORRECT LABEL START
Correct_label_loop: 1000
Correct_label_mistakes: 146
Correct_label_accuracy: 0.927
Correct_label_equation:
y = 2.5x + 1.5
```

結論: 標錯會影響分界線的判斷，然而相對於samples的總數，標錯數量少的話影響不大，若數量多的話會嚴重影響判斷!!

Discussion:

首先最有障礙的即為Python，因為過去幾乎沒有寫過python code的經驗，所以花了一點時間在適應

另外就是變數型態的問題，中間有遇上numpy.array乘法的問題如下，因此找了解決方法

TypeError: can't multiply sequence by non-int of type 'numpy.float64' in Machine learning Non-linear regression

https://stackoverflow.com/questions/63220314/typeerror-cant-multiply-sequence-by-non-int-of-type-numpy-float64-in-machine

還有第三題的POCKET Algorithm由於無法正確找到分界線因此輸出會是錯誤的，然而會錯的非常嚴重直接歪到整張圖的另一邊，因此詢問同學 郭怡靚得出要加上Learning Rate校正，因此`wt = w + y*x*LR`有乘上Learning Rate = 0.001

注: 後來測試發現與b相關，POCKET Algorithm的結果y = 2x+1 較 y = 3x+8 準確率高非常多，甚至前者不需Learning Rate反而正確率較高

第四題的mislabel，一開始測試時發現怎麼對的跟標錯的結果一樣(如下圖)，以為是程式碼寫錯

後來刻意將標錯的加非常大才看到明顯差異，最後才得出標錯的量不夠多這個結論

```
PS D:\3_third_grade\ML_2021> python .\project1_3.py
original:
y = 2x + 5
POCKET MISLABEL START
mislabel_loop: 1000
label_mistake: 111
label_accuracy: 0.9445
mislabel_mistake: 201
mislabel_accuracy: 0.8995
mislabel_equation:
y = 2.0x + 1.3333333333333333
POCKET CORRECT LABEL START
Correct_label_loop: 1000
Correct_label_mistakes: 111
Correct_label_accuracy: 0.9445
Correct_label_equation:
y = 2.0x + 1.3333333333333333
```

總結來說此次作業算是接觸到機器學習的入門，挺有趣的，最大的困難應該是在放春假被家人拖出去玩還得在飯店房間寫這個作業如此辛酸的故事吧