機器學習< Assignment #1 - Perceptron>

姓名: 翁佳煌　　學號: 409430030

要求 1:

下圖 1.1 為要求 1 的 code。
1. 設定一條線的斜率 m 和截距 b，用來區分正負標籤。

2. 隨機生成 30 個二維的數據點，每個數據點的 x 和 y 坐標都是在-20 到 20 之間的隨機值。

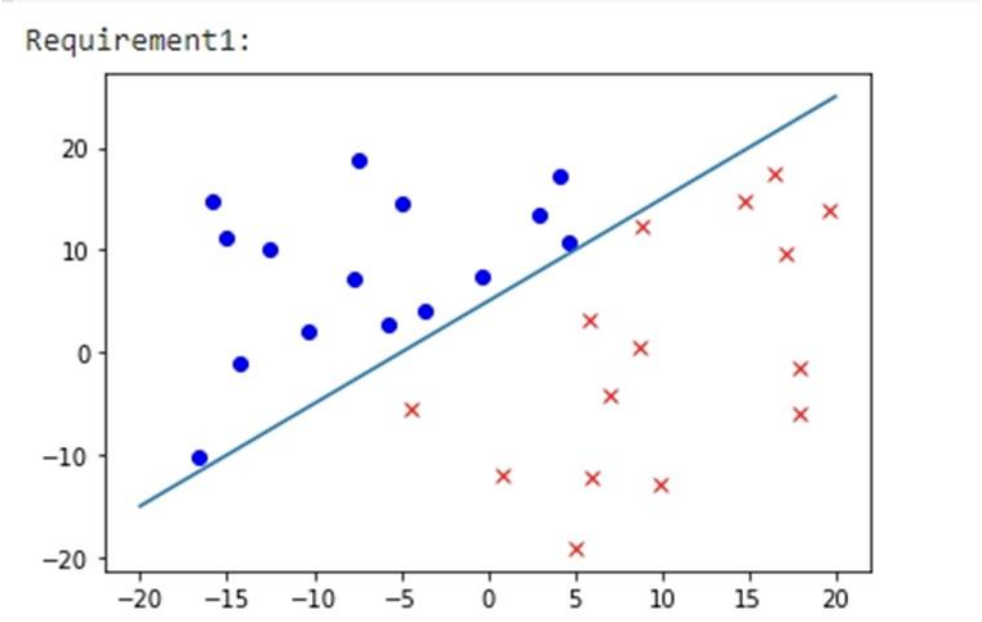
3. 判斷每個數據點是否在線的上方或下方，如果在線的上方且正標籤的數量還不足 15 個，則將其標籤設為 1，並添加到正標籤的列表中；如果在線的下方且負標籤的數量還不足 15 個，則將其標籤設為-1，並添加到負標籤的列表中。

4. 最後將正負標籤的數據合併成一個數據集，並繪製出來。其中，正標籤的數據用藍色圓圈表示，負標籤的數據用紅色叉號表示，並且製出了原來的線圖。

```python
import random
import numpy as np
import matplotlib.pyplot as plt
import time
#Requirement1
m = 1
b = 5
# 隨機生成 30 個 2D 點，其中線的右邊 15 個標記為-1，左邊 15 個標記為+1,不能落在線上
positive_samples = []
negative_samples = []
while len(positive_samples) < 15 or len(negative_samples) < 15:
        x = random.uniform(-20, 20)
        y = random.uniform(-20, 20)
        if (y > m * x + b and len(positive_samples) < 15):
                if len(positive_samples) < 15:
                        positive_samples.append((x, y ,1))

        elif (y < m * x + b and len(negative_samples) < 15):
                if len(negative_samples) < 15:
                    negative_samples.append((x, y ,-1))
        else:
            continue

data = positive_samples + negative_samples

for x,y,label in data:
    if label==1:
        plt.plot(x,y,'bo')
    else:
        plt.plot(x,y,'rx')

# 繪製生成的線
print("Requirement1:")
x = [-20, 20]
y = [m * xi + b for xi in x]
plt.plot(x, y)
plt.show()
print("\n")

```

▲圖 1.1

下圖 1.2 為要求 1 的執行結果:



▲圖 1.2

要求 1 問題討論:

在要求 1 上並無遇到太大題目上的問題，但由於我之前對 Python 不太熟悉，所以困難的點在於需要另外花時間搞懂其 random 和 matplotlib 的用法。

要求 2:

下圖 2.1 是實現 PLA(Perceptron Learning Algorithm)的訓練過程。

參數 data 是包含訓練資料的清單，其中每個元素都是一個三元組（x, y, label），代表一個二維點（x, y）和它的類別標籤 label。
w 是一個包含三個權重值的 NumPy 陣列，初始化為全零當作初始點。

1. 逐個檢查訓練資料中的每個點，計算其預測類別，若預測結果錯誤，則更新權重向量 w，並將錯誤計數器 error_count 加一。

2. 如果所有訓練點都被正確分類，那麼 error_count 保持為零，表示現有的權重向量 w 已經可以完美地分類所有資料，此時函式返回 w 和迭代次數 iteration。

3. 否則，繼續執行 PLA，直到所有資料都被正確分類為止。在這個過程中，迭代次數 iteration 會隨著每一次權重向量的更新而增加。

```
#Requirement2
print("Requirement2:")
#  定義  PLA  函數
def  PLA(data,  w):
        iteration  =  0
        while  True:
                error_count  =  0
                for  x,  y,  label  in  data:
                        if  np.sign(np.dot(w,  [x,  y,  1]))  !=  label:
                                w  +=  label  *  np.array([x,  y,  1])
                                error_count  +=  1
                if  error_count  ==  0:
                        return  w,iteration
                iteration  +=  1

```

▲圖 2.1

再來，下圖 2.2 是執行三個實驗。每個實驗產生一組隨機的資料，其中都各有 15 個正樣本和 15 個負樣本。接著使用 PLA 演算法訓練一個分類器，直到演算法收斂為止，並紀錄所需的迭代次數。最後，將產生的三個模型顯示在三個子圖中，並顯示每次實驗收斂時所需的迭代次數。最後，顯示三次實驗的平均迭代次數。

1. total_iteration 設為 0，以便計算三個實驗的平均迭代次數。

2. 在 for 迴圈中，進行三個實驗。在每次實驗中，開始產生隨機資料。

3. 初始化 positive_samples 和 negative_samples 空列表，並使用 while 迴圈生成正樣本和負樣本。並使用要求 1 的方法，如果正樣本數少於 15 個或負樣本數少於 15 個，則產生一個新的隨機資料點 (x, y)，並將其添加到正樣本或負樣本中，具體取決於它是否在隨機產生的直線上方或下方。

4. 然後，將正樣本和負樣本合併為一個列表，以供 PLA 演算法使用。初始化權重向量 w0 為 0.0、0.0、0.0。

5. 執行 PLA 演算法，直到演算法收斂為止。對於每個資料點 (x, y, label)，如果 PLA 的預測值不等於 label，則更新權重向量 w。

6. 如果錯誤次數等於 0，則表示 PLA 收斂，並將所需的迭代次數返回，以及更新的權重向量 w。如果錯誤次數不等於 0，則繼續迭代。

7. 計算每個實驗所需的迭代次數，並將其加到總迭代次數中，以便計算平均迭代次數。

8. 繪製每個實驗的結果，其中包括點和所學習的直線。

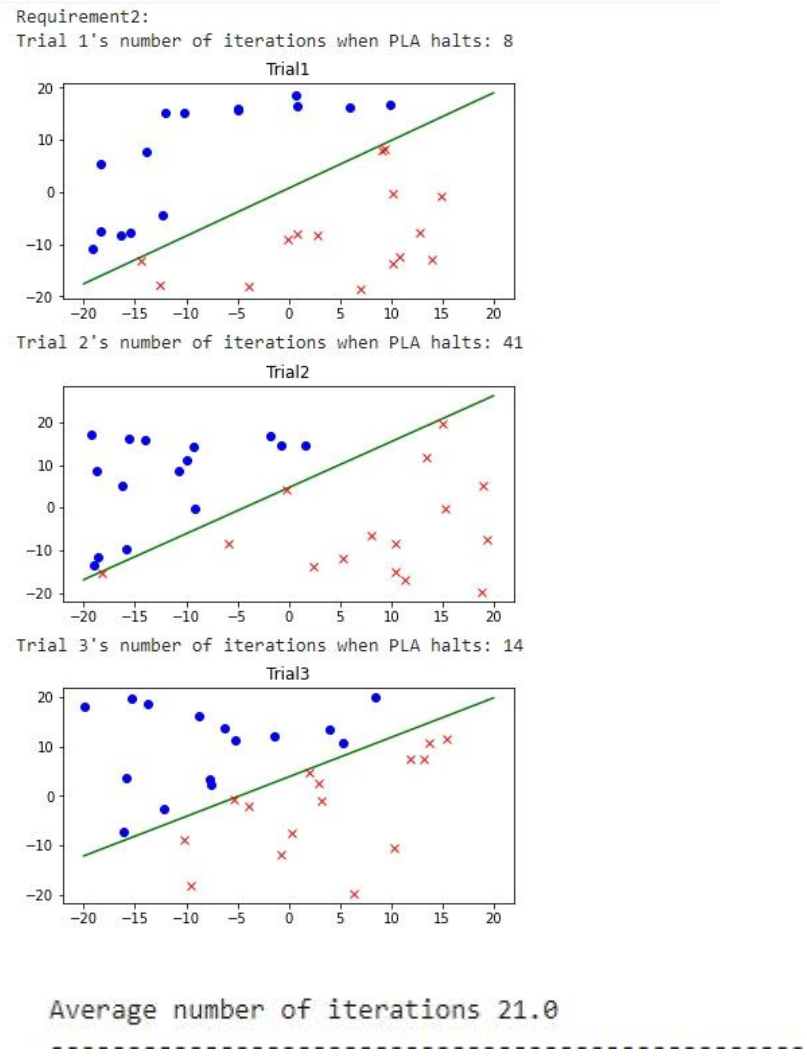
9. 顯示三次實驗的平均迭代次數。

```
60
61 total_iteration=0
62 for i in range(3):
63     positive_samples = []
64     negative_samples = []
65     while len(positive_samples) < 15 or len(negative_samples) < 15:
66             x = random.uniform(-20, 20)
67             y = random.uniform(-20, 20)
68             if (y > m * x + b and len(positive_samples) < 15):
69                     if len(positive_samples) < 15:
70                             positive_samples.append((x, y ,1))
71
72             elif (y < m * x + b and len(negative_samples) < 15):
73                     if len(negative_samples) < 15:
74                             negative_samples.append((x, y ,-1))
75             else:
76                     continue
77     data = positive_samples + negative_samples
78     # 初始化權重向量
79     w0= np.array([0.0, 0.0, 0.0])
80     # 執行 PLA，並將結果加入列表中
81     w,iteration = (PLA(data, w0))
82     total_iteration+=iteration
83     print("Trial {}'s number of iterations when PLA halts: {}".format(i+1,iteration))
84     x=[-20,20]
85     y=[-(w[0] * xi + w[2]) / w[1] for xi in x]      # w[0]x + w[1]y + w[2] = 0 的直線 ==> y = -(w[0]/w[1])x - (w[2]/w[1])
86     plt.figure(figsize=(6, 10))
87     plt.subplot(3, 1, i+1)
88     plt.plot(x, y, color='green')
89     plt.title("Trial{}".format(i+1))
90     for x,y,label in data:
91         if label == 1:
92             plt.plot(x, y, 'bo')
93         else:
94             plt.plot(x, y, 'rx')
95     plt.plot(x, y, color='green')
96     plt.show()
97
98 print("\nAverage number of iterations {}".format(total_iteration/3))
99
```

▲圖 2.2

下圖 2.3 為要求 2 的結果:

三次試驗下都有收斂，在試驗 1 迭代了 8 次，試驗 2 為 41 次，試驗 3 則是 14 次。總平均為 21 次。



▲圖 2.3

要求 2 問題討論:

在一開始困難的點為實現 PLA 演算法的時候還不太清楚運作方法，自己查了許多文章才了解其設計方式。此外，我發現每次收斂的迭代次數都不同，我認為收斂的迭代次數會因為每次隨機生成的樣本不同而有差異。每次生成的樣本是隨機取樣，因此每次取樣得到的樣本可能會有所不同，導致 PLA 算法的收斂速度也有所不同。此外，初始權重向量也可能對收斂次數產生影響。在這個例子中，初始權重向量都是[0, 0, 0]，因此可能會出現一些情況下 PLA 算法的收斂速度較慢。

要求 3:

下圖 3.1 為利用**要求 1** 的方法是隨機生成 1000 個正樣本和 1000 個負樣本，並將這些樣本畫在座標平面上，正樣本用藍色圓點表示，負樣本用紅色叉號表示。程式碼首先建立了空的正樣本列表和負樣本列表，然後進行 while 循環，直到正樣本和負樣本列表都至少包含 1000 個樣本。在每次迭代中，程式碼使用 random.uniform()函數生成一對隨機坐標 x 和 y，然後判斷它們是否位於一條直線上，如果是，就將它們添加到正樣本列表或負樣本列表中。如果正樣本和負樣本列表都不足 1000 個樣本，則繼續迭代。最後，程式碼將所有樣本畫在座標平面上，並使用藍色圓點和紅色叉號表示正樣本和負樣本。

```
#Requirement3
#PLA目的在找出一個絕對好的分類器，但Pocket則是找出相對好的即可
print("————————————————————————————————————————————————————————————")
print("Requirement3:")

positive_samples = []
negative_samples = []
while len(positive_samples) < 1000 or len(negative_samples) < 1000:
        x = random.uniform(-20, 20)
        y = random.uniform(-20, 20)
        if (y > m * x + b and len(positive_samples) < 1000):
                if len(positive_samples) < 1000:
                        positive_samples.append((x, y ,1))
        elif (y < m * x + b and len(negative_samples) < 1000):
                if len(negative_samples) < 1000:
                        negative_samples.append((x, y ,-1))
        else:
                continue

data = positive_samples + negative_samples

for x,y,label in data:
    if label==1:
        plt.plot(x,y,'bo')
    else:
        plt.plot(x,y,'rx')


```

▲圖 3.1

下圖 3.2 為實作 Pocket Algorithm ，其輸入參數為一組數據集和一個權重向量 w0，輸出結果為在訓練過程中錯誤率最小的權重向量和對應的錯誤次數。

1. 首先初始化 Pocket Algorithm 中的權重向量 w_pocket 和一個用於計算最小錯誤率的變量 min_error，以及錯誤次數 error_count，迭代次數 iteration。
2. 在每一次迭代中，從數據集 data 中隨機選取一個樣本 (x, y, label)，計算其在當前權重向量 w 下的預測結果，如果與 label 不一致，則更新權重向量 w_new = w + label * [x, y, 1]，並計算更新後的錯誤次數 error_count。
3. 然後檢查錯誤次數 error_count 是否小於最小錯誤次數 min_error，如果是，就更新 Pocket 中的權重向量 w_pocket，以及最小錯誤次數 min_error。
4. 最後更新迭代次數 iteration，如果迭代次數超過了我自己設定的最大值 2000，則停止算法並返回最小錯誤率對應的權重向量 w_pocket。

```
# Define Pocket Algorithm function
def pocket(data,w0):
    w_pocket=w0
    w=w0
    min_pocket=np.array([0.0, 0.0, 0.0])
    iteration=0
    min_error=len(data) # 初始化錯誤次數為數據點個數
    error_count=0
    while iteration < 2000:
        i = random.randint(0,len(data)-1)
        x,y,label=data[i]
        if np.sign(np.dot(w,[x,y,1]))!= label:
            w_new = w + label * np.array([x,y,1])
            error_count = sum([np.sign(np.dot(w_new, [x, y, 1])) != label for x, y, label in data])
            # 檢查錯誤率是否下降，若是就更新 Pocket 中的權重向量
            #print("error_count:",error_count,"min_error:",min_error)
            #print("The w_new:",w_new)
            #print("Now w_pocket:",w_pocket)
            if error_count < min_error:
                w_pocket=w_new.copy()
                min_error=error_count
                #print("update w_pocket",w_pocket)
        w=w_new.copy()
        iteration += 1
    return w_pocket,min_error
```

▲圖 3.2

下圖 3.3，定義了一個起始權重向量 w0，並使用 PLA 和 Pocket Algorithm 兩個演算法來分別訓練線性分類器，使用同一組數據集 data。接著，按照**要求 3** 計算並印出 PLA 和 Pocket Algorithm 的執行時間，並把 pocket 的結果畫出。

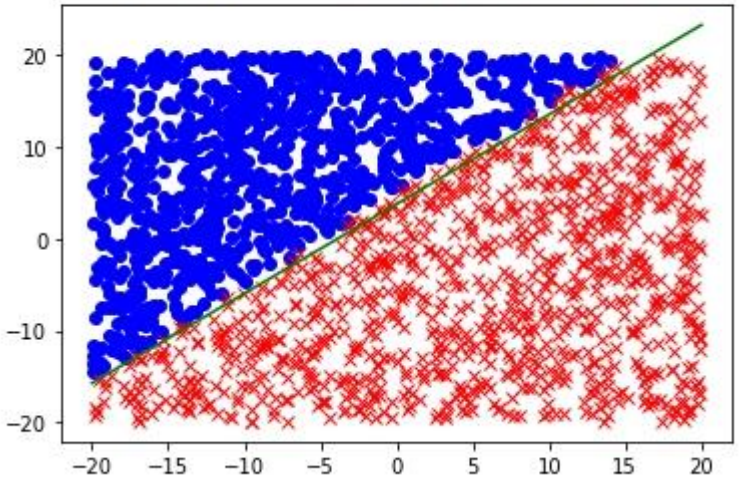
```
w0= np.array([0.0, 0.0, 0.0])
# Run PLA and measure the execution time
start_time = time.time()
w_pla,iteration = PLA(data,w0)
end_time = time.time()
print("PLA execution time:", end_time-start_time)

w0= np.array([0.0, 0.0, 0.0])
# Run Pocket Algorithm and measure the execution time
start_time = time.time()
w_pocket3,error3 = pocket(data,w0)
end_time = time.time()
print("Pocket Algorithm execution time:", end_time-start_time)


x=[-20,20]
y=[-(w_pocket3[0] * xi + w_pocket3[2]) / w_pocket3[1] for xi in x]
plt.plot(x, y, color='green')
plt.show()

```

▲圖 3.3

下圖 3.4.為要求 3 的結果:

按照**要求 3**，從下圖 3.4 可發現，Pocket 演算法的執行時間比 PLA 長，這是因為 Pocket 演算法在每次更新權重時都要先計算錯誤率，而且要與目前 Pocket 中最小錯誤率進行比較，這樣才能確定是否要更新 Pocket 中的權重向量。這些額外的計算使得 Pocket 比 PLA 更加複雜，因此執行時間更長。

```
Requirement3:
PLA execution time: 0.24556899070739746
Pocket Algorithm execution time: 2.6519930362701416
```



▲圖 3.4

要求 3 問題討論:

在這遇到最大的問題就是實作 Pocket 演算法，雖然與 PLA 非常相似，但還是需要額外花時間搞懂，其關鍵核心為 PLA 目的在找出一個絕對好的分類器，但 Pocket 則是找出相對好的即可。另外，在實作時，我一直在猶豫最大迭代次數應該要設多少才合適，這問題解法就是需要根據數據的大小等因素來慢慢調整和實驗。

要求 4:

下圖 4.1 的程式碼為對數據集進行了一些修改以創建一些錯誤的標籤。我們隨機選擇了 50 個原來是正樣本的樣本，將它們標記為負樣本，同時隨機選擇了 50 個原來是負樣本的樣本，將它們標記為正樣本，這樣我們就創建了一些具有錯誤標籤的樣本。然後，我們將這些樣本與原始數據集合併，並使用**要求 3** 中定義的 POCKET 算法訓練模型。最後，我們印出在需求 3 和需求 4 中的準確率，並繪製出分類平面，藍色圓圈代表正樣本，紅色叉代表負樣本，綠色線條代表 POCKET 算法找到的分類平面。

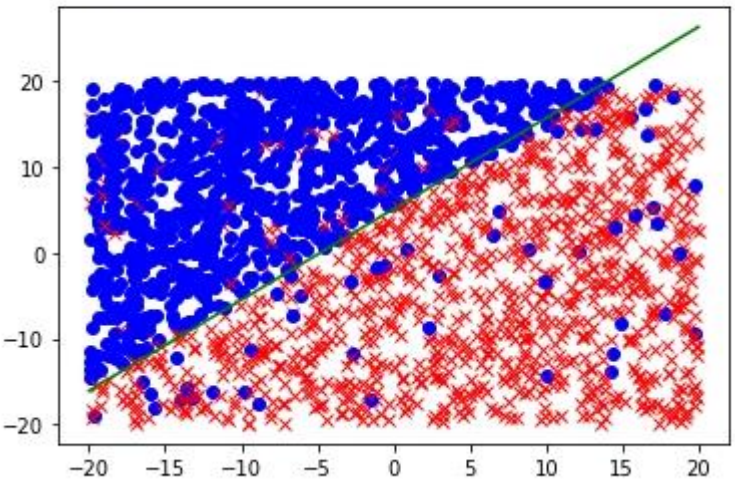
```
#Requirement4
print("Requirement4:")
#  Randomly  mislabel  50  positive  and  50  negative  samples
positive_samples_mislabeled  =  random.sample([sample  for  sample  in  positive_samples  if  sample[2]==1],  k=50)
for  sample  in  positive_samples_mislabeled:
        index  =  positive_samples.index(sample)
        positive_samples[index]  =  (sample[0],  sample[1],  -1)

negative_samples_mislabeled  =  random.sample([sample  for  sample  in  negative_samples  if  sample[2]==-1],  k=50)
for  sample  in  negative_samples_mislabeled:
        index  =  negative_samples.index(sample)
        negative_samples[index]  =  (sample[0],  sample[1],  1)

#  Merge  the  mislabeled  samples  with  the  original  dataset
data  =  positive_samples  +  negative_samples
w0=  np.array([0.0,  0.0,  0.0])
w_pocket4,error4  =  pocket(data,w0)
print("Accurary  in  Problem3:",1  -  error3/len(data))
print("Accurary  in  Problem4:",1  -  error4/len(data))

for  x,y,label  in  data:
    if  label==1:
        plt.plot(x,y,'bo')
    else:
        plt.plot(x,y,'rx')

x  =  np.linspace(-20,  20,  40)
y=[-(w_pocket4[0]  *  xi  +  w_pocket4[2])  /  w_pocket4[1]  for  xi  in  x]
plt.plot(x,  y,  color='green')
plt.show()
```

▲圖 4.1

下圖 4.2 為要求 4 的結果:

從下圖 4.2 的結果圖可知，在 Problem3 中的 Pocket Algorithm 的情況下，最終權重向量的準確度為 0.9825，而在 Problem4 由於把 50 個正樣本和 50 個負樣本標記錯誤，因此可知準確度比 Problem3 來的小。

```
Requirement4:
Accurary in Problem3: 0.9825
Accurary in Problem4: 0.9385
```



▲圖 4.2

要求 4 問題討論:

實作問題 4 沒什麼大問題，只要各挑 50 個點標記錯誤就好，隨機選擇 50 個正樣本和 50 個負樣本標記錯誤可能導致一些偏差。在實際應用中，標記錯誤的樣本可能分佈在不同的區域，並且可能與原始數據分佈的特徵不同。另外我查到許多不同對於錯誤樣本處理的機器學習演算法，例如 Boosting 和 Random Forest 等，該選擇使用哪種演算法，需要考慮多個因素，包括資料的特性、資料的大小、計算資源的限制、以及對準確度和解釋性的需求等等。