

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN



BÁO CÁO BÀI TẬP 1
TOÁN CHO KHOA HỌC MÁY TÍNH
CS115.012

Giảng viên hướng dẫn: Dương Việt Hằng

Sinh viên thực hiện:

Nguyễn Trọng Nhân 22521005

TP. Hồ Chí Minh, tháng 10 năm 2023

Mục lục

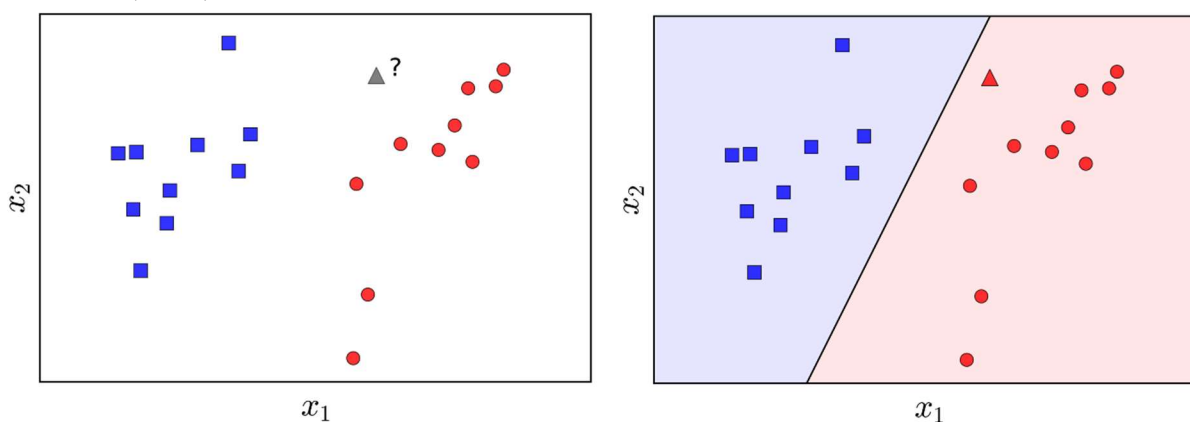
I.	Giới thiệu bài toán Perceptron	2
1.	Giới thiệu.....	2
2.	Phát biểu bài toán	2
II.	Thuật toán Perceptron (Perceptron Learning Algorithm)	3
1.	Xây dựng hàm mất mát	3
2.	Tóm tắt thuật toán Perceptron	6
3.	Chứng minh hội tụ.....	6
III.	Minh hoạ với Python	6
1.	Tải bộ dữ liệu.....	6
2.	Minh hoạ PLA	8
	Nguồn tham khảo.....	11

I. Giới thiệu bài toán Perceptron

1. Giới thiệu

Perceptron là một thuật toán phân lớp (classification) cho trường hợp đơn giản nhất: chỉ có hai lớp (bài toán với chỉ hai lớp được gọi là binary classification) và cũng chỉ hoạt động được trong một trường hợp rất cụ thể. Tuy nhiên, nó là nền tảng cho một mảng lớn quan trọng của Machine Learning là Neural Networks và sau này là Deep Learning.

Giả sử chúng ta có hai tập hợp dữ liệu đã được gán nhãn được minh họa trong hình 1 bên trái dưới đây. Hai lớp của chúng ta là tập các điểm màu xanh và tập các điểm màu đỏ. Bài toán đặt ra là: từ dữ liệu của hai tập được gán nhãn cho trước, hãy xây dựng một bộ phân lớp để khi có một điểm dữ liệu hình tam giác màu xám mới, ta có thể dự đoán được màu (nhãn) của nó.



Hình 1-1. Bài toán Perceptron

Hiểu theo một cách khác, chúng ta cần tìm lãnh thổ của mỗi lớp sao cho, với mỗi một điểm mới, ta chỉ cần xác định xem nó nằm vào lãnh thổ của lớp nào rồi quyết định nó thuộc lớp đó. Để tìm lãnh thổ của mỗi lớp, chúng ta cần đi tìm biên (boundary) giữa hai lãnh thổ này. Vậy bài toán phân lớp có thể coi là bài toán đi tìm biên giữa các lớp. Và biên đơn giản nhất trong không gian hai chiều là một đường thẳng, trong không gian ba chiều là một mặt phẳng, trong không gian nhiều chiều là một siêu mặt phẳng (hyperplane). Những biên phẳng này được coi là đơn giản vì nó có thể biểu diễn dưới dạng toán học bằng một hàm số đơn giản có dạng tuyến tính (linear). Tất nhiên, chúng ta đang giả sử rằng tồn tại một đường phẳng để có thể phân định lãnh thổ của hai lớp. Hình 1 bên phải minh họa một đường thẳng phân chia hai phần trong mặt phẳng. Phần có nền màu xanh được coi là lãnh thổ của lớp xanh, phần có nền màu đỏ được coi là lãnh thổ của lớp đỏ. Trong trường hợp này, điểm dữ liệu mới hình tam giác được phân vào lớp đỏ.

2. Phát biểu bài toán

Bài toán Perceptron được phát biểu như sau: Cho hai class được gán nhãn, hãy tìm một đường phẳng sao cho toàn bộ các điểm thuộc class 1 nằm về 1 phía, toàn bộ các điểm thuộc class 2 nằm về phía còn lại của đường phẳng đó. Với giả định rằng tồn tại một đường phẳng như thế.

Nếu tồn tại một đường phẳng phân chia hai lớp thì ta gọi hai lớp đó là linearly separable. Các thuật toán phân lớp tạo ra các biên là các siêu mặt phẳng được gọi chung là Linear Classifier.

II. Thuật toán Perceptron (Perceptron Learning Algorithm)

Ý tưởng cơ bản của PLA là xuất phát từ một nghiệm dự đoán nào đó, qua mỗi vòng lặp, nghiệm sẽ được cập nhật tới một vị trí tốt hơn. Việc cập nhật này dựa trên việc giảm giá trị của một hàm mất mát nào đó.

1. Xây dựng hàm mất mát

Giả sử $X = [x_1, x_2, \dots, x_N] \in \mathbb{R}^{d \times N}$ là ma trận chứa các điểm dữ liệu mà mỗi cột $x_i \in \mathbb{R}^{d \times 1}$ là một điểm dữ liệu trong không gian d chiều.

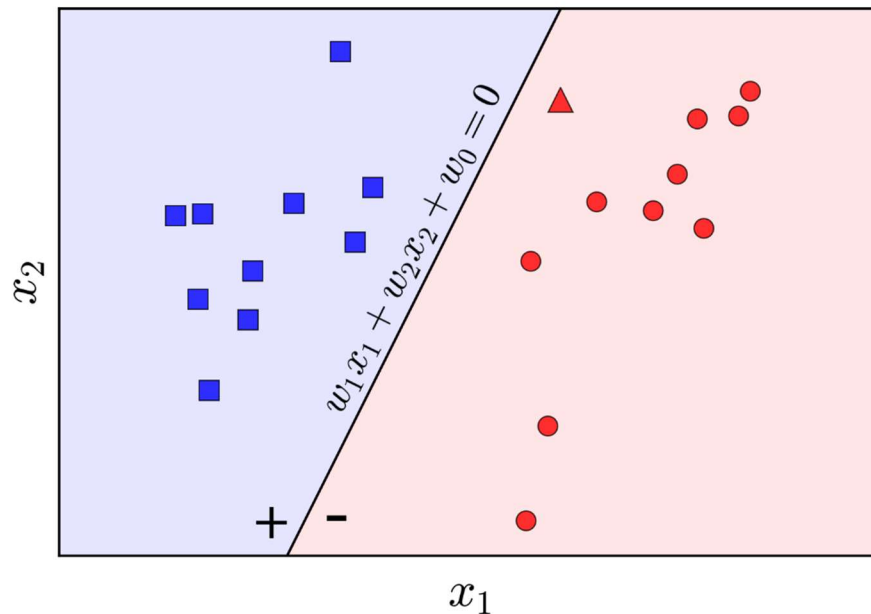
Các nhãn tương ứng với từng điểm dữ liệu được lưu trong một vector hàng $Y = [y_1, y_2, \dots, y_N] \in \mathbb{R}^{1 \times N}$, với $y_i = 1$ (xanh) nếu x_i thuộc lớp 1 và $y_i = 0$ nếu x_i thuộc lớp 2 (đỏ).

Tại một thời điểm, giả sử ta tìm được biên là đường phẳng có phương trình:

$$\begin{aligned} f_w(x) &= w_1x_1 + \dots + w_dx_d + w_0 \\ &= W^T \bar{x} = 0 \end{aligned}$$

với \bar{x} là điểm dữ liệu mở rộng bằng cách thêm phần tử $x_0 = 1$ lên trước vector X tương tự như trong hồi quy tuyến tính. Gọi X là điểm dữ liệu mở rộng.

Với trường hợp $d = 2$, giả sử đường thẳng $w_1x_1 + w_2x_2 + w_0 = 0$ chính là nghiệm cần tìm như hình dưới đây:



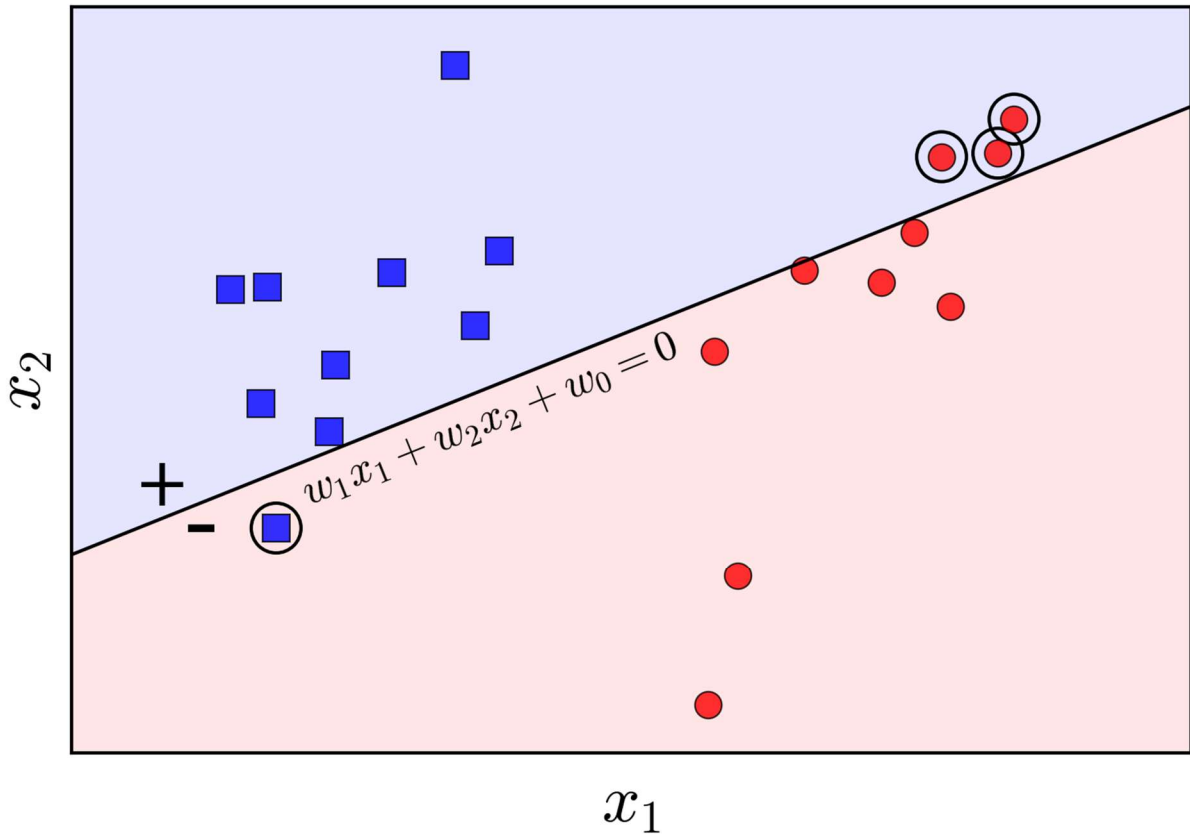
Hình II-1. Phương trình đường thẳng biên

Nhận xét: Các điểm nằm về cùng một phía so với đường thẳng này sẽ làm cho hàm số $f_w(x)$ mang cùng dấu. Giả sử các điểm nằm trong nửa mặt phẳng nền xanh mang dấu (+), còn lại mang dấu (-). Các dấu này cũng tương đương với nhãn y của mỗi lớp. Vậy nếu W là một nghiệm của bài toán Perceptron, với một điểm dữ liệu mới X chưa được gán nhãn, ta có thể xác định lớp của nó bằng phép toán:

$$label(x) = sgn(W^T X)$$

Trong đó, sgn là hàm xác định dấu, với giả sử $sgn(0) = 1$.

Tiếp theo ta cần xây dựng hàm mất mát với tham số W bất kỳ. Vẫn trong không gian hai chiều, giả sử đường thẳng $w_1x_1 + w_2x_2 + w_0 = 0$ được cho như hình dưới đây:



Hình II-2. Đường thẳng bất kỳ và các điểm phân lớp sai được khoanh tròn.

Các điểm được khoanh tròn là các điểm bị phân lớp sai. Hàm mất mát đơn giản nhất mà ta nghĩ đến là hàm đếm số lượng các điểm vi phân lớp sai và tìm các cực tiểu hoá hàm này:

$$J_1(W) = \sum_{x_i \in \mathcal{M}} (-y_i sgn(W^T X_i))$$

trong đó \mathcal{M} là tập hợp các điểm bị phân lớp sai. Với mỗi $x_i \in \mathcal{M}$, vì điểm này bị đánh dấu sai (ngược với y_i) nên $-y_i \text{sgn}(W^T X_i) = 1$. Vậy $J_1(W)$ chính là hàm đếm số lượng các điểm bị phân lớp sai. Khi hàm này bằng 0 thì ta không còn điểm nào bị phân lớp sai.

Hàm này là một hàm rời rạc, không tính được đạo hàm theo W nên rất khó tối ưu. Chúng ta cần tìm một hàm mất mát khác để việc tối ưu hoá khả thi hơn.

Xét hàm mất mát sau:

$$J(W) = \sum_{x_i \in \mathcal{M}} (-y_i W^T X_i)$$

Với việc bỏ đi hàm sgn , một điểm bị phân lớp sai x_i càng nằm xa biên thì giá trị $-y_i W^T X_i$ càng lớn. Giá trị nhỏ nhất của hàm này là 0 nếu không có điểm nào bị phân lớp sai. Hàm mất mát này cũng được cho là tốt hơn J_1 vì nó trừng phạt các điểm lấn sâu trong lãnh thổ của lớp kia. Trong khi J_1 trừng phạt các điểm bị phân lớp sai như nhau, bất kể xa hay gần biên.

Tại một thời điểm, nếu chúng ta chỉ quan tâm tới các điểm bị misclassified thì hàm số khả vi, vậy chúng ta có thể sử dụng Gradient Descent hoặc Stochastic Gradient Descent (SGD) để tối ưu hàm mất mát này. Với ưu điểm của SGD cho các bài toán large-scale, chúng ta sẽ làm theo thuật toán này. Với mỗi điểm dữ liệu x_i bị phân lớp sai, hàm mất mát trở thành:

$$J(W; X_i; y_i) = -y_i W^T X_i$$

Đạo hàm tương ứng:

$$\nabla_W J(W; X_i; y_i) = -y_i X_i$$

Vậy quy tắc cập nhật là:

$$W = W + \eta y_i x_i$$

với η là tốc độ học (learning rate) được chọn bằng 1. Ta có quy tắc cập nhật:

$$W_i = W_i + y_i x_i$$

Nói cách khác, với mỗi điểm x_i bị phân lớp sai, ta chỉ cần nhân nó với nhãn y_i của nó, cộng vào W để được W mới.

Ta có:

$$\begin{aligned} W_{t+1}^T X_i &= (W_t + y_i X_i)^T X_i \\ &= W_t^T X_i + y_i \|X_i\|_2^2 \end{aligned}$$

Nếu $y_i = 1$, vì X_i bị phân lớp sai nên $W_{t+1}^T X_i < 0$. Cũng vì $y_i = 1$ nên $y_i \|X_i\|_2^2 = \|X_i\|_2^2 \geq 1$, nghĩa là $W_{t+1}^T X_i > W_t^T X_i$, hay W_{t+1} tiến về phía làm cho X_i được phân lớp đúng. Điều tương tự xảy ra nếu $y_i = -1$.

2. Tóm tắt thuật toán Perceptron

1. Chọn ngẫu nhiên một vector hệ số W với các phần tử gần 0
2. Duyệt ngẫu nhiên qua từng điểm dữ liệu X_i
 - Nếu X_i được phân lớp đúng, chúng ta không cần làm gì
 - Nếu X_i bị phân lớp sai, cập nhật W theo công thức:

$$W = W + y_i X_i$$
3. Kiểm tra xem còn bao nhiêu điểm bị phân lớp sai. Nếu không còn, ta kết thúc thuật toán. Nếu còn, quay lại bước 2.

3. Chứng minh hội tụ

Giả sử rằng W^* là một nghiệm của bài toán (ta có thể giả sử việc này được vì chúng ta đã có giả thiết hai lớp là linearly separable - tức tồn tại nghiệm). Có thể thấy rằng, với mọi $\alpha > 0$, nếu W^* là nghiệm, αW^* cũng là nghiệm của bài toán. Xét dãy số không âm $u_\alpha(t) = \|W_t - \alpha W^*\|_2^2$. Với X_i là một điểm bị phân lớp sai, nếu dùng nghiệm W_t ta có:

$$\begin{aligned} u_\alpha(t+1) &= \|W_{t+1} - \alpha W^*\|_2^2 \\ &= \|W_t + y_i X_i - \alpha W^*\|_2^2 \\ &= \|W_t - \alpha W^*\|_2^2 + y_i^2 \|X_i\|_2^2 + 2y_i X_i^T (W_t - \alpha W^*) \\ &< u_\alpha(t) + \|X_i\|_2^2 - 2\alpha y_i X_i^T W^* \end{aligned}$$

Dấu nhỏ hơn ở dòng cuối là vì $y^2 = 1$ và $2y_i X_i^T W_t < 0$. Nếu ta đặt:

$$\begin{aligned} \beta^2 &= \max_{i=1,2,\dots,N} \|X_i\|_2^2 \\ \gamma &= \min_{i=1,2,\dots,N} y_i X_i^T W^* \end{aligned}$$

và chọn $\alpha = \frac{\beta^2}{\gamma}$, ta có:

$$0 \leq u_\alpha(t+1) < u_\alpha(t) + \beta^2 - 2\alpha\gamma = u_\alpha(t) - \beta^2$$

Điều này nghĩa là: nếu luôn luôn có các điểm bị misclassified thì dãy $u_\alpha(t)$ là dãy giảm, bị chặn dưới bởi 0, và phần tử sau kém phần tử trước ít nhất một lượng là $\beta^2 > 0$. Điều vô lý này chứng tỏ đến một lúc nào đó sẽ không còn điểm nào bị misclassified. Nói cách khác, thuật toán PLA hội tụ sau một số hữu hạn bước.

III. Minh hoạ với Python

1. Tải bộ dữ liệu

Dùng bộ dữ liệu iris dataset của thư viện scikit-learn phân loại hoa diên vĩ dựa trên chiều dài và chiều rộng của đài hoa (perceptron_test.py):

```

#import thu vien numpy
import numpy as np
#import datasets tu thu vien scikit_Learn
from sklearn import datasets
#import ham chia du lieu train-test
from sklearn.model_selection import train_test_split
#import thu vien matplotlib
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
#map mau cho data
cmap = ListedColormap(['#FF0000', '#0000FF'])

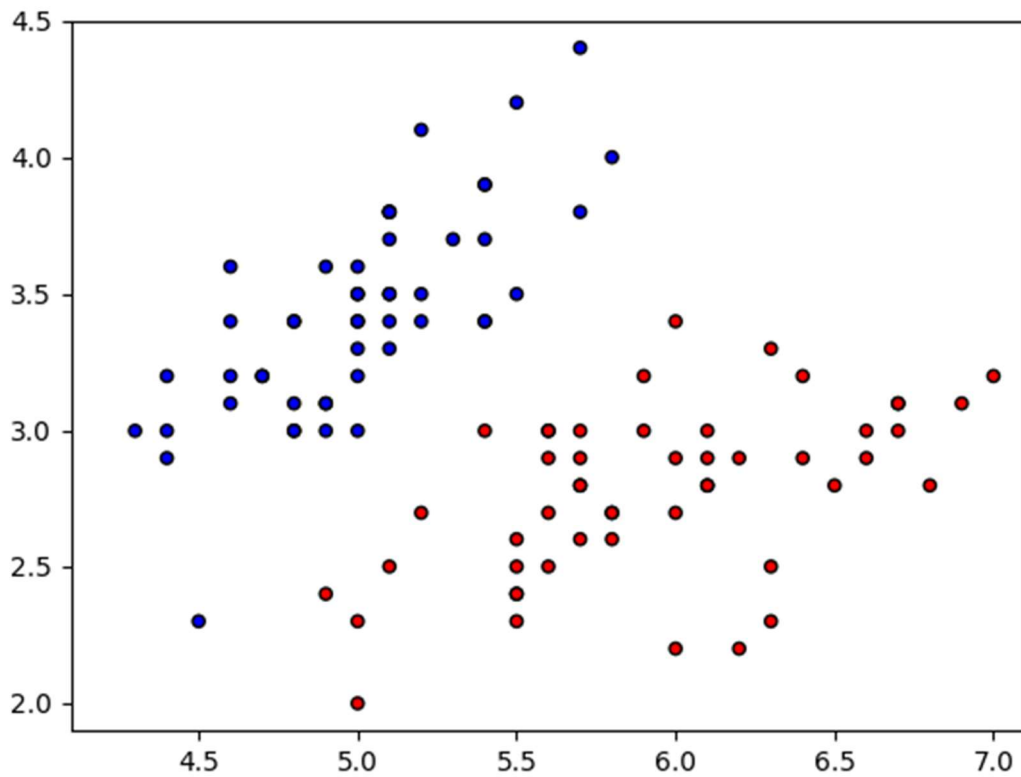
#Load iris dataset
iris = datasets.load_iris()
x, y = iris.data, iris.target

#chi lay 2 lop de dam bao linearly separable
x = x[:100]
y = y[:100]

#chi lay 2 feature sepal length va sepal width
x = x[:, 0:2]

```

Bộ dữ liệu gồm 3 lớp, mỗi lớp có 50 mục, dữ liệu mỗi mục gồm 4 số: chiều dài, chiều rộng của đài hoa và chiều dài, chiều rộng của cánh hoa. Ở bài toán này, ta chỉ cần 2 thông tin của đài hoa và 2 lớp được đánh nhãn 0 và 1 (loài setosa và versicolor) được đánh nhãn lại tương ứng là 1 và -1. Bộ dữ liệu được minh họa trong hình dưới đây:



Hình III-1. Minh hoạ bộ dữ liệu. Hai nhãn 1 và -1 lần lượt được tô màu xanh và đỏ.

Các điểm xanh được đánh nhãn 1, các điểm đỏ được đánh nhãn -1. Do chỉ lấy 2 trong số 4 đặc trưng dữ liệu, lớp 1 có 40 điểm dữ liệu, lớp -1 có 44 điểm dữ liệu được minh hoạ do trùng lặp, không phải 50 điểm mỗi lớp như trong bộ dữ liệu.

2. Minh hoạ PLA

Dựa vào tóm tắt thuật toán đã nêu trên, ta có mã cho thuật toán PLA (perceptron.py):

```

import numpy as np

def signf(w, x):
    if np.sign(np.dot(w.T, x)) >= 0:
        return 1
    return -1

def isConverged(x, y, w):
    for i in range(x.shape[0]):
        xi = x[i].reshape(x.shape[1],1)
        if signf(w, xi) != y[i]:
            return 0
    return 1

class perceptron:
    def fit(X, y):
        #so vector n + so chieu d
        n = X.shape[0]
        xtemp = np.c_[np.ones(n), X]
        d = xtemp.shape[1]
        #khởi tạo vector w
        w = [np.random.randn(d,1)]
        ytemp = y.copy()
        for i in range(n):
            if ytemp[i] == 0:
                ytemp[i] = -1
            ytemp[i] *= -1
        misclassifiedPoints = []
        while 1:
            #xếp dữ liệu ngẫu nhiên
            randomId = np.random.permutation(n)
            #duyet qua dữ liệu
            for i in range(n):
                xi = xtemp[randomId[i], :].reshape(d,1) #dua vector xi về
                yi = ytemp[randomId[i]] #nhân
                if signf(w[-1], xi) != yi: #tích trong khác nhân
                    misclassifiedPoints.append(randomId[i])
                    w_star = w[-1]+yi*xi #cập nhật
                    w.append(w_star)
            if isConverged(xtemp, ytemp, w[-1]):
                break
        return w

```

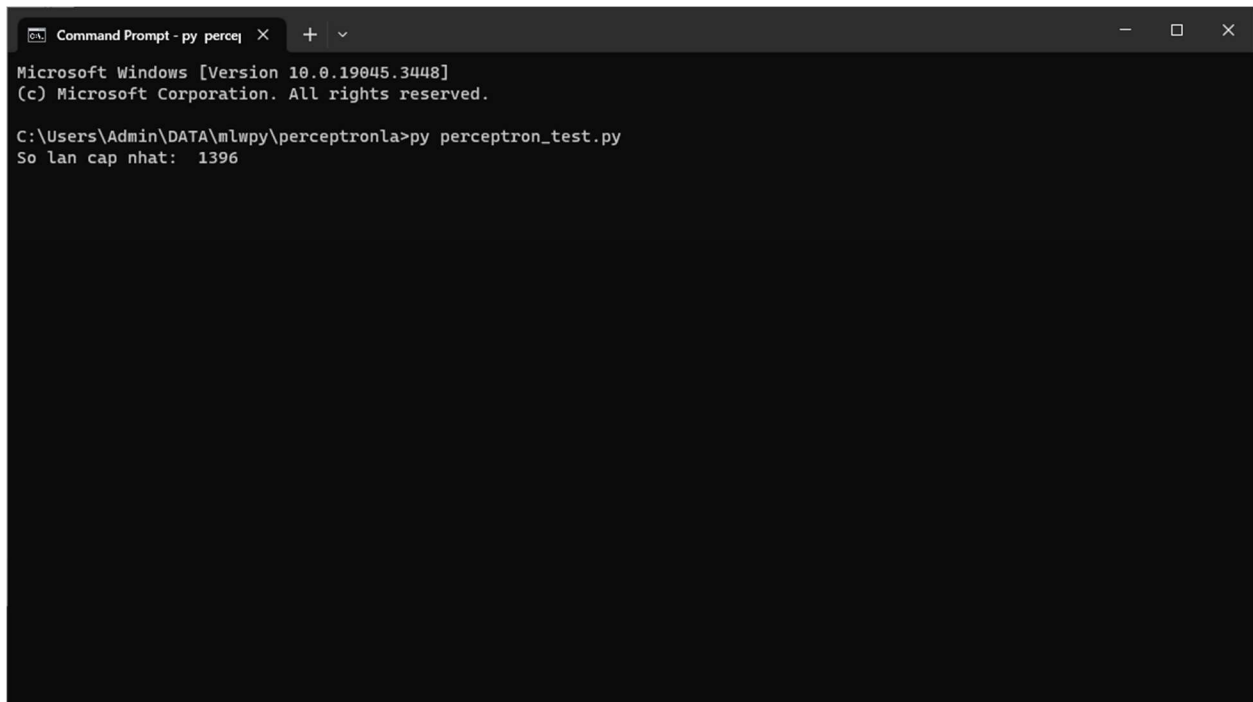
Thực thi thuật toán và vẽ biểu đồ với bộ dữ liệu trong file perceptron_test.py:

```
#import perceptron
from perceptron import perceptron
w = perceptron.fit(x, y)

#ve duong thang
def drawLines(w):
    w0, w1, w2 = w[0], w[1], w[2]
    if w2 != 0:
        x11, x12 = -100, 100
        return plt.plot([x11, x12], [-(w1*x11+w0)/w2, -(w1*x12+w0)/w2], 'k')
    else:
        x10 = -w0/w1
        return plt.plot([-100, 100], [x10, x10], 'k')

plt.figure()
plt.axis([4.1, 7.1, 1.9, 4.5])
plt.scatter(x[:, 0], x[:, 1], c = y, cmap = cmap, edgecolor = 'k', s = 20)
print("So lan cap nhat: ",len(w))
drawLines(w[-1])
plt.show()
```

Kết quả thực thi:

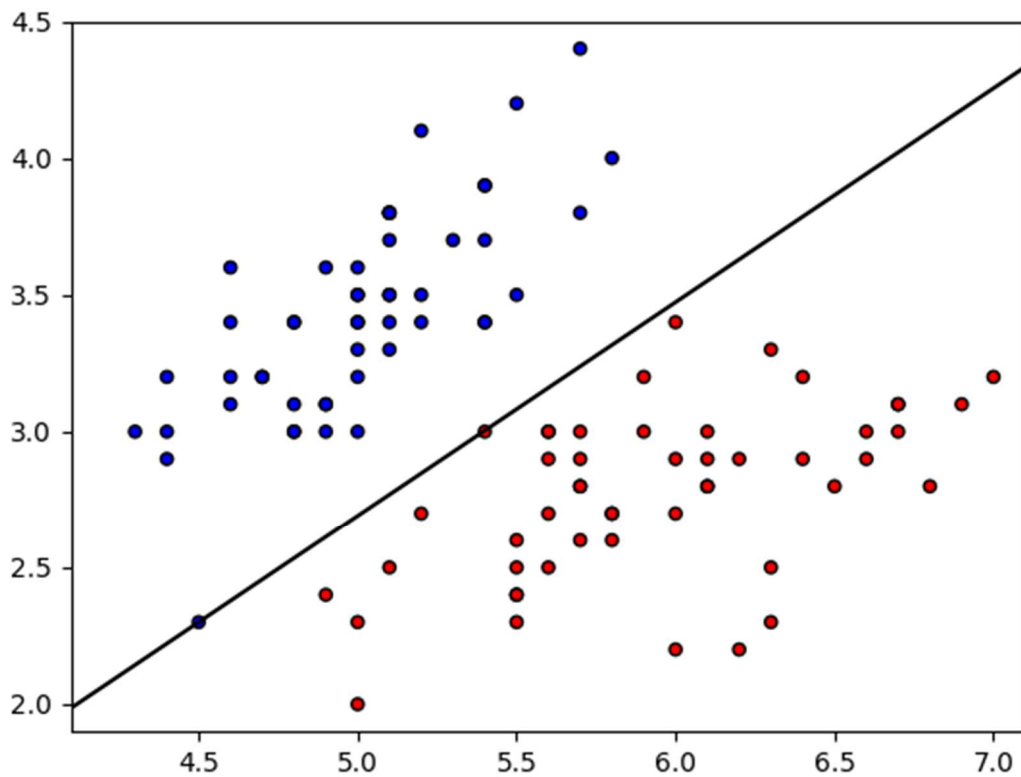


```
Command Prompt - py perce
Microsoft Windows [Version 10.0.19045.3448]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin\DATA\mlwp\perceptronla>py perceptron_test.py
So lan cap nhat: 1396
```

Hình III-2. Kết quả thực thi

Biểu đồ với dữ liệu được phân loại:



Hình III-3. Dữ liệu đã được phân loại.

Nguồn tham khảo

MARAVEGIAS, K. (2020). *Exploring the iris data set - scikit-learn*. Được truy lục từ kaggle: <https://www.kaggle.com/code/kostasmar/exploring-the-iris-data-set-scikit-learn>

Tiếp, V. H. (2017, Jan 21). *Bài 9: Perceptron Learning Algorithm*. Được truy lục từ Machine Learning cơ bản: <https://machinelearningcoban.com/2017/01/21/perceptron/>