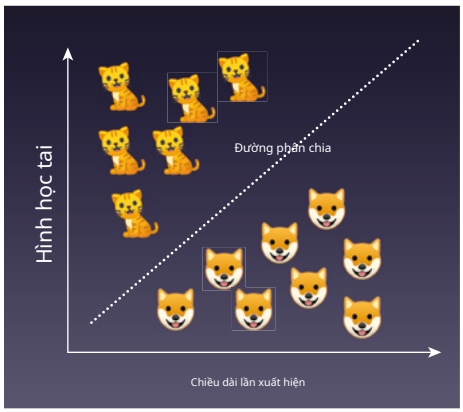
CHƯƠNG SỐ 8

Lê giám sát arning— Phân loại Sử dụng Ủng hộ Véc tơ M achines

Máy hỗ trợ Vector là gì?

Trong chương trước, bạn đã biết cách thực hiện phân loại bằng cách sử dụng hồi quy hậu cần. Trong chương này, bạn sẽ tìm hiểu một thuật toán học máy có giám sát khác cũng rất phổ biến trong giới khoa học dữ liệu—*Máy Véc tơ hỗ trợ (SVM)*. Giống như hồi quy hậu cần, SVM cũng là một thuật toán phân loại.

Ý tưởng chính đằng sau SVM là vẽ một đường thẳng giữa hai hoặc nhiều lớp theo cách tốt nhất có thể (xem Hình 8.1).



Hình 8.1:Sử dụng SVM để tách hai lớp động vật

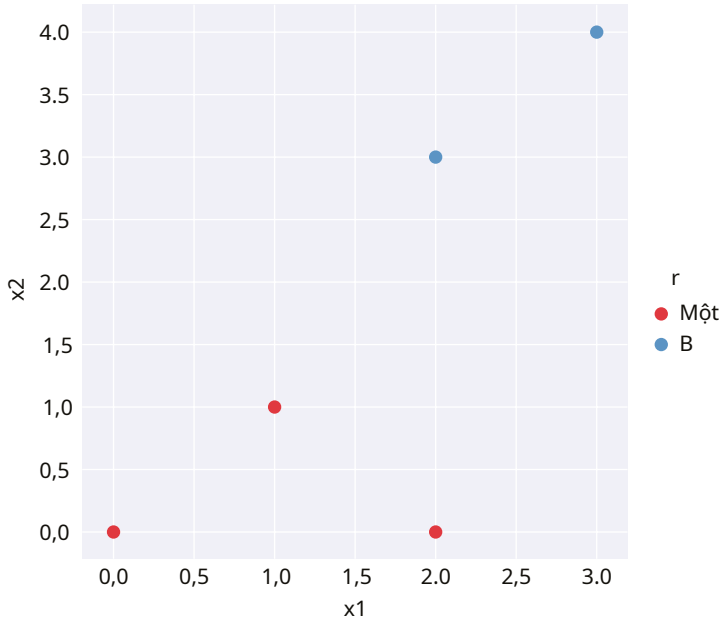
Học máy Python®, Ấn bản đầu tiên. Wei-Meng Lee.

Khi đường được vẽ để tách các lớp, bạn có thể sử dụng nó để dự đoán dữ liệu trong tương lai. Ví dụ: với chiều dài mõm và hình dạng tai của một con vật mới chưa biết, giờ đây bạn có thể sử dụng đường phân chia làm công cụ phân loại để dự đoán xem con vật đó là chó hay mèo.

Trong chương này, bạn sẽ tìm hiểu cách thức hoạt động của SVM và các kỹ thuật khác nhau mà bạn có thể sử dụng để điều chỉnh SVM nhằm giải quyết các tập dữ liệu có thể phân tách phi tuyến tính.

Khả năng tách biệt tối đa

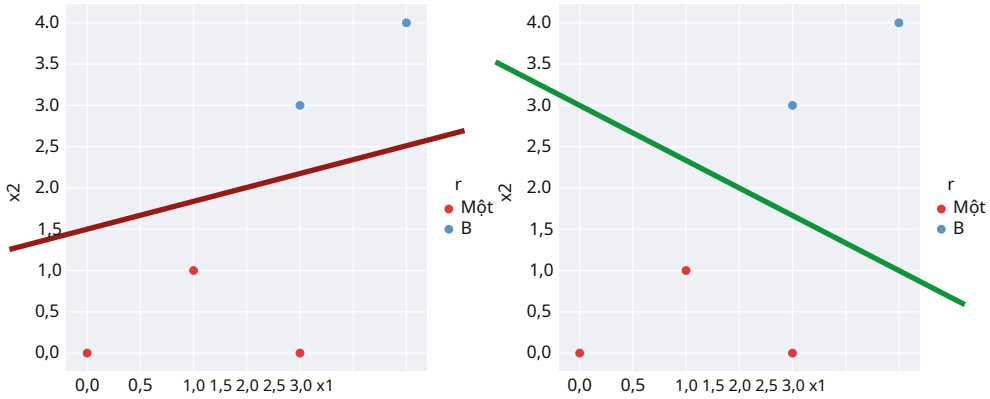
Làm thế nào để SVM tách hai hoặc nhiều lớp? Xem xét tập hợp các điểm trong Hình 8.2. Trước khi bạn nhìn vào hình tiếp theo, hãy liên tưởng trực quan đến một đường thẳng chia các điểm thành hai nhóm.



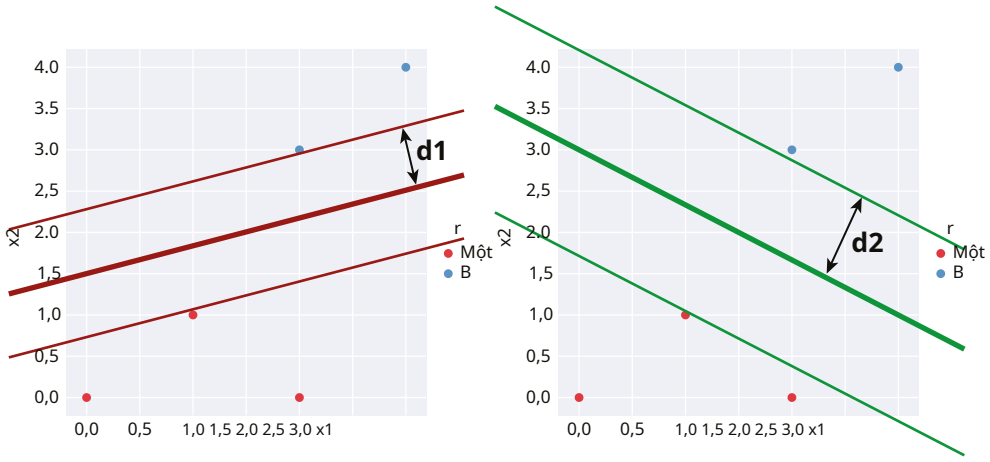
Hình 8.2:Một tập hợp các điểm có thể được phân tách bằng SVM

Bây giờ hãy nhìn vào Hình 8.3, nó cho thấy hai đường có thể phân cách hai nhóm điểm. Đây có phải là những gì bạn có trong tâm trí?

Mặc dù cả hai đường đều tách các điểm thành hai nhóm riêng biệt, nhưng cái nào là đúng? Đối với SVM, dòng bên phải là dòng có lề rộng nhất (với mỗi lề chạm ít nhất một điểm trong mỗi lớp), như thể hiện trong Hình 8.4. Trong hình đó, d1 và d2 là chiều rộng của các lề. Mục đích là tìm độ rộng lớn nhất có thể cho lề có thể tách hai nhóm. Do đó, trong trường hợp này, d2 là lớn nhất. Vì vậy, dòng được chọn là dòng bên phải.



Hình 8.3: Hai cách có thể để chia điểm thành hai lớp



d2> d1

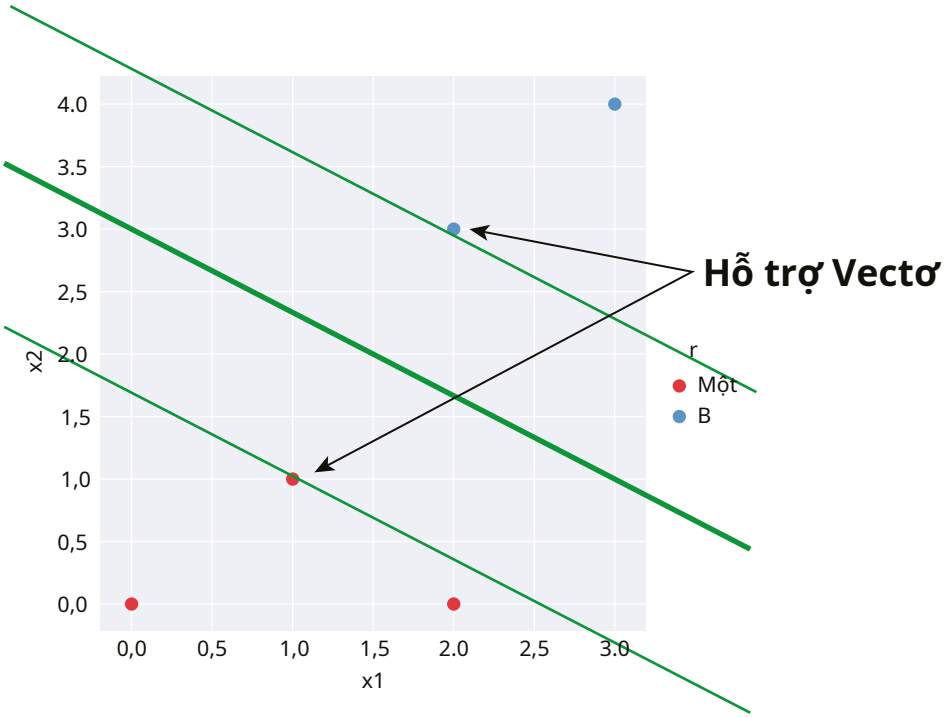
Hình 8.4:SVM tìm cách tách hai lớp với biên độ rộng nhất

Mỗi trong số hai lề tiếp xúc với (các) điểm gần nhất với mỗi nhóm điểm và tâm của hai lề được gọi là*siêu phẳng*. Siêu phẳng là đường phân cách hai nhóm điểm. Chúng tôi sử dụng thuật ngữ "siêu phẳng" thay vì "đường" vì trong SVM, chúng tôi thường xử lý nhiều hơn hai chiều và việc sử dụng từ "siêu phẳng" sẽ truyền tải chính xác hơn ý tưởng về một mặt phẳng trong không gian đa chiều.

Hỗ trợ Vectơ

Một thuật ngữ chính trong SVM là*vectơ hỗ trợ*. Các vectơ hỗ trợ là các điểm nằm trên hai biên. Sử dụng ví dụ từ phần trước, Hình 8.5 cho thấy hai vectơ hỗ trợ nằm trên hai biên.

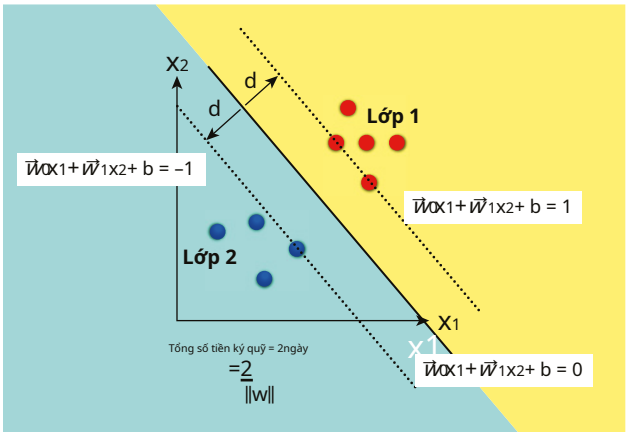
Trong trường hợp này, chúng ta nói rằng có hai vectơ hỗ trợ — một vectơ cho mỗi lớp.



Hình 8.5:Các vectơ hỗ trợ là các điểm nằm trên lề

Công thức cho Siêu phẳng

Với một loạt các điểm, câu hỏi tiếp theo sẽ là tìm công thức cho siêu phẳng, cùng với hai lề. Không cần đi quá sâu vào toán học đằng sau điều này, Hình 8.6 cho thấy công thức để nhận được siêu phẳng.



Hình 8.6:Công thức cho siêu phẳng và hai lề đi kèm của nó

Như bạn có thể thấy từ Hình 8.6, công thức cho siêu phẳng (g) được đưa ra như sau:

$$(x) = W_0x_1 + W_1x_2 + b$$

nơi $x_1$và $x_2$là đầu vào,$_0$và$_1$là các vectơ trọng số, và b là độ lệch. Nếu giá trị của g là >= 1, thì điểm được chỉ định nằm trong Lớp 1 và nếu giá trị của g là <= -1, thì điểm được chỉ định là trong Lớp 2. Như đã đề cập, mục tiêu của SVM là tìm các biên rộng nhất để phân chia các lớp và tổng biên (2d) được xác định bởi:

$$/ \|w \|$$

trong đó ||w|| là vectơ trọng số chuẩn hóa ($_0$và $_1$). Sử dụng khóa đào tạo đặt, mục tiêu là giảm thiểu giá trị của w để bạn có thể nhận được sự phân tách tối đa giữa các lớp. Khi điều này được thực hiện, bạn sẽ có thể nhận được các giá trị của $_0$, $_1$, và B.

Tìm lề là một*Hạn chế tối ưu*vấn đề có thể được giải quyết bằng cách sử dụng*Hệ số Larange*kĩ thuật. Cuốn sách này nằm ngoài phạm vi thảo luận về cách tìm lề dựa trên tập dữ liệu, nhưng cũng đủ để nói rằng chúng tôi sẽ sử dụng thư viện Scikit-learning để tìm chúng.

Sử dụng Scikit-learning cho SVM

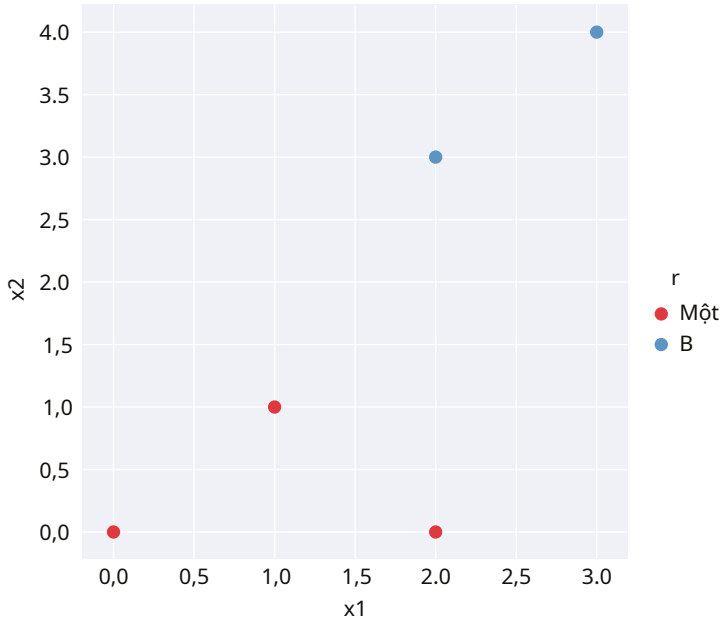
Bây giờ chúng ta hãy làm một ví dụ để xem cách SVM hoạt động và cách triển khai nó bằng Scikit-learning. Đối với ví dụ này, chúng tôi có một tệp có tênsvm.csv chứa dữ liệu sau:

```
x1, x2, r
0,0, A
1,1, A
2,3, B
2,0, A
3,4, B
```

Điều đầu tiên mà chúng tôi sẽ làm là lập biểu đồ các điểm bằng cách sử dụng Seaborn:

```
% matplotlib inline
nhập gấu trúc dưới dạng pd
nhập numpy dưới dạng np
nhập seaborn dưới dạng sns; sns.set (font_scale = 1.2) nhập
matplotlib.pyplot dưới dạng plt

data = pd.read_csv ('svm.csv')
sns.lmplot ('x1', 'x2',
                    data = dữ liệu,
                    hue = 'r',
                    Palette = 'Set1',
                    fit_reg = Sai,
                    scatter_kws = {"s": 50});
```

Hình 8.7 cho thấy các điểm được vẽ bằng Seaborn.



Hình 8.7:Vẽ các điểm bằng Seaborn

Sử dụng các điểm dữ liệu mà chúng ta đã tải trước đó, bây giờ hãy sử dụng Scikitlearn's svmmô-đun củaSVClớp để giúp chúng tôi lấy giá trị cho các biến khác nhau mà chúng tôi cần tính toán theo cách khác. Đoạn mã sau sử dụng *hạt nhân tuyến tính*để giải quyết vấn đề. Hạt nhân tuyến tính giả định rằng tập dữ liệu có thể được phân tách một cách tuyến tính.

```
từ sklearn nhập svm
# - - - Chuyển đổi Cột thành Ma trận --- points = data
[['x1', 'x2']]. Giá trị result = data ['r']


clf = svm.SVC (kernel = 'linear') clf.fit (điểm,
kết quả)

print ('Vectơ trọng số (w) =', clf.coef_ [0]) print ('b =',
clf.intercept_ [0])
print ('Các chỉ số của vectơ hỗ trợ =', clf.support_) print ('Các vectơ hỗ
trợ =', clf.support_vectors_)
print ('Số vectơ hỗ trợ cho mỗi lớp =', clf.n_support_) print ('Hệ số của vectơ hỗ trợ trong hàm
quyết định =',
            np.abs (clf.dual_coef_))
```

CácSVCviết tắt của*Hỗ trợ phân loại vectơ*. Cácsvmmô-đun chứa một loạt các lớp triển khai SVM cho các mục đích khác nhau:

svm.LinearSVC:Phân loại véc tơ hỗ trợ tuyến tính

svm.LinearSVR:Hồi quy vectơ hỗ trợ tuyến tính

svm.NuSVC:Phân loại vectơ hỗ trợ Nu

svm.NuSVR:Hồi quy vectơ Nu-Hỗ trợ

svm.OneClassSVM:Phát hiện ngoại lệ không được giám sát

svm.SVC:Phân loại vectơ hỗ trợ C

svm.SVR:Hồi quy vectơ hỗ trợ Epsilon

TIỀN BOA **Đối với chương này, chúng tôi tập trung vào việc sử dụng SVM để phân loại, mặc dù SVM cũng có thể được sử dụng để hồi quy.**

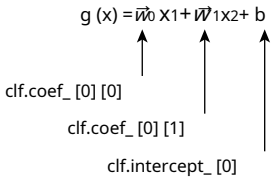
Đoạn mã trước tạo ra kết quả sau:

```
Véc tơ trọng lượng (w) = [0,4 0,8] b = -2,2

Chỉ số của vectơ hỗ trợ = [1 2] Các vectơ hỗ trợ =
[[1. 1.]
 [2. 3.]]
Số vectơ hỗ trợ cho mỗi lớp =                    [1 1]
Hệ số của vectơ hỗ trợ trong hàm quyết định =                    [[0,4 0,4]]
```

Như bạn có thể thấy, hệ số cân nặng đã được tìm thấy là [0,4 0,8], có nghĩa là $_0$bây giờ là 0,4 và $_1$bây giờ là 0,8. Giá trị của b là –2,2 và có hai vectơ hỗ trợ. Chỉ số của các vectơ hỗ trợ là 1 và 2, có nghĩa là các điểm là những điểm được in đậm:

x1	x2	r	
0	0	0	Một
1	**1**	**1**	**Một**
2	**2**	**3**	**B**
3	2	0	Một
4	3	4	B

Hình 8.8 cho thấy mối quan hệ giữa các biến khác nhau trong công thức và các biến trong đoạn mã.



Hình 8.8:Mối quan hệ giữa các biến trong công thức và các biến trong đoạn mã

Vẽ siêu phẳng và lề

Với tất cả các giá trị của các biến thu được, bây giờ là lúc để vẽ đồ thị siêu phẳng và hai biên đi kèm của nó. Bạn có nhớ công thức cho siêu phẳng? Nó như sau:

$$(x) = W_{0X_1} + W_{1X_2} + b,$$

Để vẽ siêu phẳng, hãy đặt $_{0X_1} + W_{1X_2} + b$ đến 0, như t của anh:

$_{0X_1} + W_{1X_2} + b$ θ

Để vẽ siêu phẳng (trong trường hợp này là một đường thẳng), chúng ta cần hai điểm: một trên trục x và một trên trục y.

Sử dụng công thức trước, khi $_1 = 0$, chúng ta có thể giải cho x_2 như sau:

0(0) + *1x2*+b = 0

1x2= -b

x_2= -b / *1*

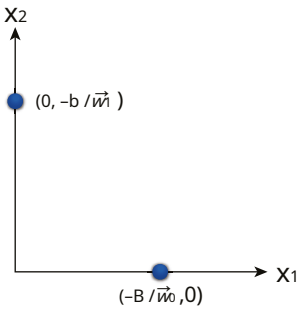
Khi x_2= 0, chúng ta có thể giải cho x_1 như sau:

0x1+ *1*(0) + b = 0

0x1= -b

x_1= -b / *0*

Điểm (0, -b / *1*) là *chặn y* của đường thẳng. Hình 8.9 cho thấy hai điểm trên hai trục.



Hình 8.9: Hai điểm chặn đối với siêu phẳng

Khi các điểm trên mỗi trục được tìm thấy, bây giờ bạn có thể tính toán *dốc* như sau:

Độ dốc = (-b / *1*) / (b / *0*)

Độ dốc = - (*0* / *1*)

Với độ dốc và điểm giao nhau của đường thẳng được tìm thấy, bây giờ bạn có thể tiếp tục và vẽ siêu phẳng. Đoạn mã sau thực hiện điều đó:

```
# - - - w là vectơ trọng số --- w = clf.coef_ [0]


# - - - tìm độ dốc của siêu phẳng --- Slides = -w [0] / w
[1]

b = clf.intercept_ [0]

# - - - tìm tọa độ cho siêu phẳng --- xx = np.linspace (0, 4)

yy = độ dốc * xx - (b / w [1])

# - - - vẽ lề --- s = clf.support_vectors_
[0]                                    # - - - vector hỗ trợ đầu tiên ---
yy_down = dốc * xx + (s [1] - độ dốc * s [0])

s = clf.support_vectors _ [- 1]        # - - - vector hỗ trợ cuối cùng ---
yy_up = dốc * xx + (s [1] - độ dốc * s [0])

# - - - vẽ các điểm ---
sns.lmplot ('x1', 'x2', data = data, hue = 'r', Palette = 'Set1', fit_reg = False,
scatter_kws = {"s": 70})

# - - - vẽ siêu phẳng ---
plt.plot (xx, yy, linewidth = 2, color = 'green');

# - - - vẽ 2 lề --- plt.plot (xx, yy_down,
'k--') plt.plot (xx, yy_up, 'k--')
```

Hình 8.10 cho thấy siêu phẳng và hai lề.

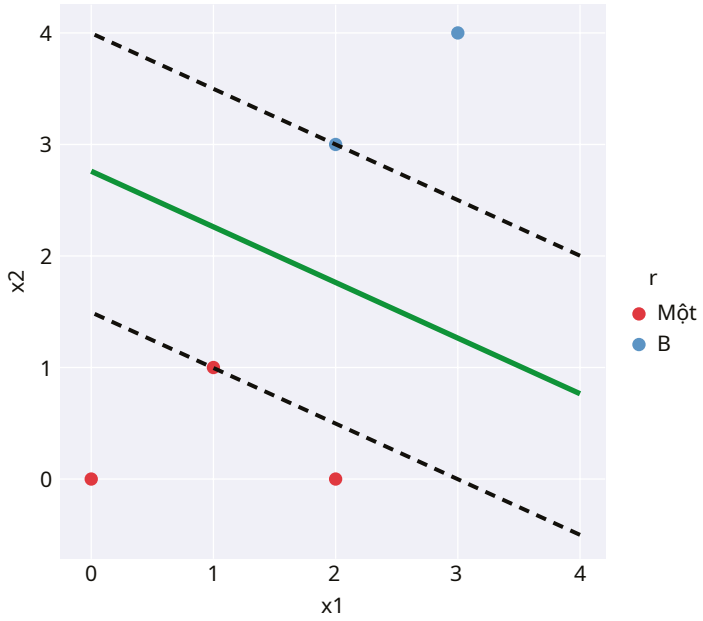
Đưa ra dự đoán

Hãy nhớ rằng, mục tiêu của SVM là tách các điểm thành hai hoặc nhiều lớp, để bạn có thể sử dụng nó để dự đoán các lớp của điểm trong tương lai. Sau khi đào tạo mô hình của bạn bằng cách sử dụng SVM, bây giờ bạn có thể thực hiện một số dự đoán bằng cách sử dụng mô hình.

Đoạn mã sau sử dụng mô hình mà bạn đã đào tạo để thực hiện một số dự đoán:

```
print (clf.p Dự đoán ([[3,3]]) [0]) in (clf.p      # 'B'
Dự đoán ([[4,0]]) [0]) in (clf.p Dự đoán           # 'MỘT'
([[2,2]]) [0]) print (clf.p Dự đoán ([[1,2]])      # 'B'
[0])                                               # 'MỘT'
```

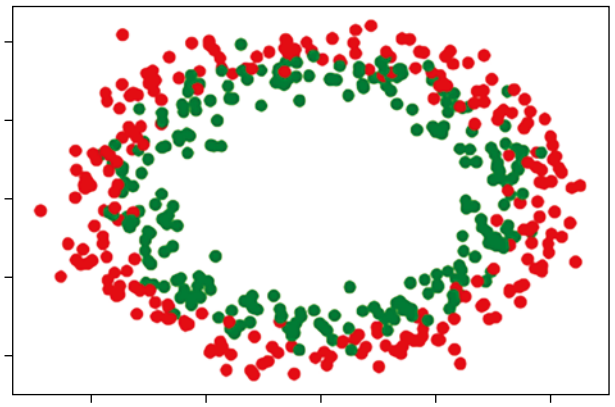
Kiểm tra các điểm so với biểu đồ trong Hình 8.10 và xem nó có hợp lý với bạn không.



Hình 8.10: Siêu phẳng và hai lề

Kernel Trick

Đôi khi, các điểm trong tập dữ liệu không phải lúc nào cũng có thể phân tách tuyến tính. Hãy xem xét các điểm được chỉ ra trong Hình 8.11.



Hình 8.11: Biểu đồ phân tán của hai nhóm điểm được phân bố theo kiểu hình tròn

Bạn có thể thấy rằng không thể vẽ một đường thẳng để ngăn cách hai tập hợp điểm. Tuy nhiên, với một số thao tác, bạn có thể làm cho tập hợp các điểm này có thể phân tách một cách tuyến tính. Kỹ thuật này được gọi là*lừa hạt nhân*. Thủ thuật hạt nhân là một kỹ thuật trong học máy biến dữ liệu thành một không gian có chiều cao hơn để sau khi chuyển đổi, nó có một ranh giới phân chia rõ ràng giữa các lớp dữ liệu.

Thêm thứ nguyên thứ ba

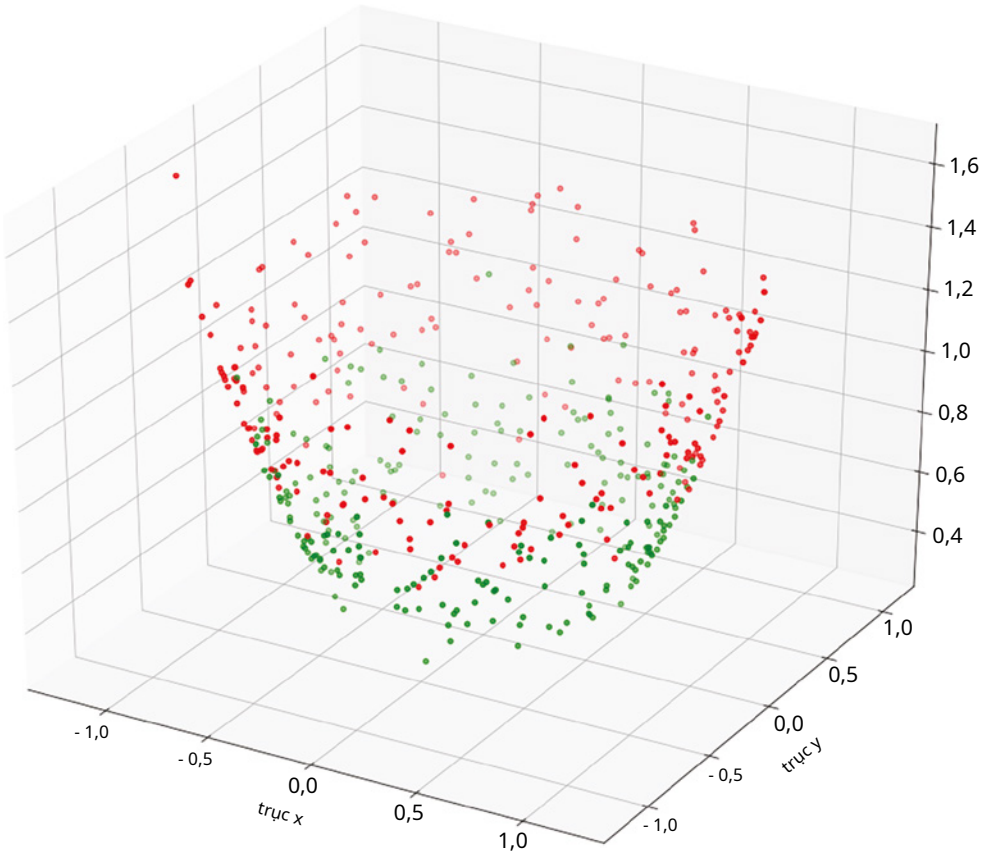
Để làm như vậy, chúng ta có thể thêm một chiều thứ ba, chẳng hạn như trục z và xác định z là:

$$= x_2 + y_2$$

Khi chúng tôi vẽ biểu đồ các điểm bằng biểu đồ 3D, các điểm bây giờ có thể phân tách một cách tuyến tính. Rất khó để hình dung điều này trừ khi bạn vạch ra các điểm. Đoạn mã sau thực hiện điều đó:

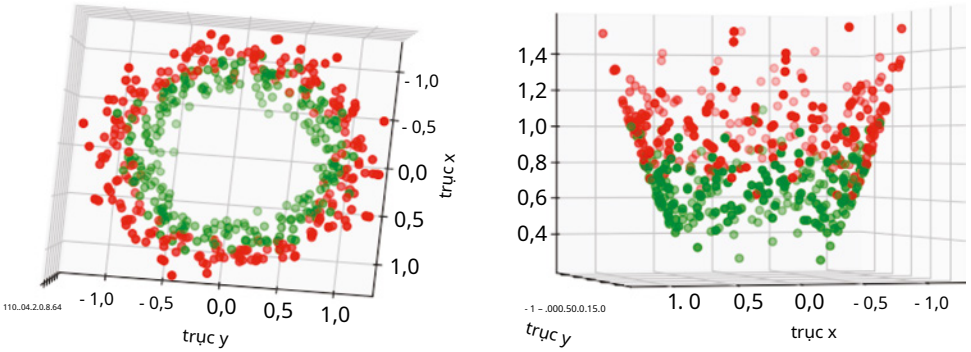
```
% matplotlib inline

từ mpl_toolkits.mplot3d nhập Axes3D nhập
matplotlib.pyplot dưới dạng plt
nhập numpy dưới dạng np
từ sklearn.datasets nhập make_circles

# - - - X là các tính năng và c là các nhãn lớp ---
X, c = make_circles (n_samples = 500, noise = 0.09)

rgb = np.array (['r', 'g']) plt.scatter (X [:, 0], X [:, 1], color =
rgb [c]) plt.show ()


fig = plt.figure (figsize = (18,15))
ax = fig.add_subplot (111, projector = '3d') z = X [:, 0] ** 2
+ X [:, 1] ** 2
ax.scatter (X [:, 0], X [:, 1], z, color = rgb [c]) plt.xlabel ("x-axis")

plt.ylabel ("trục y")
plt.show ()
```

Đầu tiên, chúng tôi tạo hai tập hợp các điểm ngẫu nhiên (tổng cộng 500 điểm) được phân phối theo kiểu vòng tròn bằng cách sử dụng`make_circles ()`hàm số. Sau đó, chúng tôi vẽ chúng trên biểu đồ 2D (như những gì được thể hiện trong Hình 8.11). Sau đó, chúng tôi thêm trục thứ ba, trục z và vẽ biểu đồ ở dạng 3D (xem Hình 8.12).



Hình 8.12:Vẽ các điểm trong không gian ba chiều

TIỀN BOA **Nếu bạn chạy mã trước đó trong Terminal (chỉ cần xóa**%matplotlib inline**ở đầu đoạn mã) bằng cách sử dụng**con trăn**, bạn sẽ có thể xoay và tương tác với biểu đồ. Hình 8.13 cho thấy các góc nhìn khác nhau của biểu đồ 3D.**



Hình 8.13:Các góc nhìn khác nhau trên cùng một tập dữ liệu trong 3D

Vẽ siêu phẳng 3D

Với các điểm được vẽ trong biểu đồ 3D, bây giờ chúng ta hãy đào tạo mô hình bằng cách sử dụng thứ nguyên thứ ba:

```
# - - - kết hợp X (trục x, trục y) và z thành một mảng duy nhất --- features =
np.concatenate ((X, z.reshape (-1,1)), axis = 1)

# - - - sử dụng SVM để đào tạo --- từ
sklearn nhập svm

clf = svm.SVC (kernel = 'linear') clf.fit
(features, c)
```

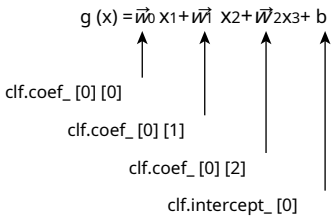
Đầu tiên, chúng tôi kết hợp ba trục thành mộtndarraysử dụngnp.concatenate ()hàm số. Sau đó chúng tôi đào tạo người mẫu như bình thường. Đối với một phân tách tuyến tính tập hợp các điểm trong hai chiều, công thức cho siêu phẳng như sau:

$$g(x) = \vec{w}_0 x_1 + \vec{w}_1 x_2 + b$$

Đối với tập hợp các điểm bây giờ trong ba chiều, công thức bây giờ trở thành tiếp theo:

$$g(x) = \vec{w}_0 x_1 + \vec{w}_1 x_2 + \vec{w}_2 x_3 + b$$

Đặc biệt, $\vec{w}_2$bây giờ được đại diện bởiclf.coef_ [0] [2],như hình 8.14.



Hình 8.14:Công thức cho siêu phẳng trong 3D và các biến tương ứng của nó trong đoạn mã

Bước tiếp theo là vẽ siêu phẳng ở dạng 3D. Để làm được điều đó, bạn cần tìm giá trị củax3,có thể được suy ra, như trong Hình 8.15.

$$\vec{w}_0 x_1 + \vec{w}_1 x_2 + \vec{w}_2 x_3 + b = 0$$
$$\vec{w}_2 x_3 = -\vec{w}_0 x_1 - w_1 x_2 - b$$
$$x_3 = \frac{-\vec{w}_0 x_1 - \vec{w}_1 x_2 - b}{\vec{w}_2}$$

Hình 8.15:Công thức tìm siêu phẳng trong 3D

Điều này có thể được thể hiện trong mã như sau:

```
x3 = lambda x, y: (-clf.intercept_ [0] - clf.coef_ [0] [0] * x-clf.coef_ [0] [1] * y) /
                        clf.coef_ [0] [2]
```

Để vẽ siêu phẳng trong 3D, hãy sử dụngplot_surface ()hàm số:

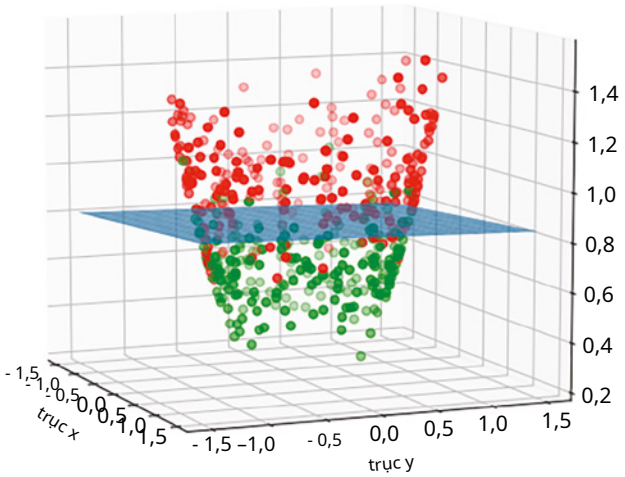
```
tmp = np.linspace (-1,5,1.5,100) x, y =
np.meshgrid (tmp, tmp)

ax.plot_surface (x, y, x3 (x, y)) plt.show ()
```

Toàn bộ đoạn mã như sau:

```
từ mpl_toolkits.mplot3d nhập Axes3D nhập
matplotlib.pyplot dưới dạng plt
nhập numpy dưới dạng np
từ sklearn.datasets nhập make_circles

# - - - X là các tính năng và c là các nhãn lớp ---
X, c = make_circles (n_samples = 500, noise = 0.09) z = X [:, 0] **
2 + X [:, 1] ** 2

rgb = np.array (['r', 'g'])

fig = plt.figure (figsize = (18,15))
ax = fig.add_subplot (111, projector = '3d') ax.scatter (X [:, 0], X
[:, 1], z, color = rgb [c]) plt.xlabel ("x-axis" )

plt.ylabel ("trục y")
#  plt.show ()

# - - - kết hợp X (trục x, trục y) và z thành một mảng duy nhất --- features =
np.concatenate ((X, z.reshape (-1,1)), axis = 1)

# - - - sử dụng SVM để đào tạo --- từ
sklearn nhập svm

clf = svm.SVC (kernel = 'linear') clf.fit
(features, c)
x3 = lambda x, y: (-clf.intercept_ [0] - clf.coef_ [0] [0] * x-clf.coef_ [0] [1]
                        * y) / clf.coef_ [0] [2]
```

```
tmp = np.linspace (-1,5,1.5,100) x, y =
np.meshgrid (tmp, tmp)

ax.plot_surface (x, y, x3 (x, y)) plt.show
()
```

Hình 8.16 cho thấy siêu phẳng, cũng như các điểm, được vẽ dưới dạng 3D.



Hình 8.16: Siêu phẳng trong 3D cắt qua hai tập hợp điểm

Các loại nhân

Cho đến thời điểm này, chúng ta chỉ thảo luận về một loại SVM — SVM tuyến tính. Như tên của nó, SVM tuyến tính sử dụng một đường thẳng để tách các điểm. Trong phần trước, bạn cũng đã tìm hiểu về việc sử dụng các thủ thuật hạt nhân để tách hai tập dữ liệu được phân phối theo kiểu vòng tròn và sau đó sử dụng SVM tuyến tính để tách chúng.

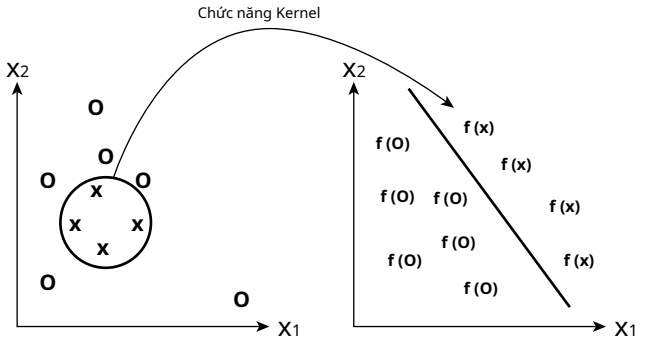
Đôi khi, không phải tất cả các điểm đều có thể được phân tách một cách tuyến tính, cũng như không thể tách chúng ra bằng cách sử dụng các thủ thuật hạt nhân mà bạn đã quan sát trong phần trước. Đối với loại dữ liệu này, bạn cần phải "uốn cong" các đường để tách chúng ra. Trong học máy, *hạt nhân*là các hàm biến đổi dữ liệu của bạn từ không gian phi tuyến thành không gian tuyến tính (xem Hình 8.17).

Để hiểu cách hạt nhân hoạt động, hãy sử dụng tập dữ liệu Iris làm ví dụ. Đoạn mã sau tải tập dữ liệu Iris và in ra các tính năng, mục tiêu và tên mục tiêu:

```
% matplotlib inline
nhập gấu trúc dưới dạng pd
nhập numpy dưới dạng np
```

```
từ sklearn import svm, datasets import
matplotlib.pyplot as plt

iris = datasets.load_iris () print
(iris.data [0: 5])                  # in 5 hàng đầu tiên
print (iris.feature_names)          # ['chiều dài đài hoa (cm)', 'chiều rộng đài hoa (cm)',
                                    # 'chiều dài cánh hoa (cm)', 'chiều rộng cánh hoa (cm)']
print (iris.target [0: 5]) print    # in 5 hàng đầu tiên
(iris.target_names)                 # ['setosa' 'versicolor' 'virginica']
```

Để minh họa, chúng tôi sẽ chỉ sử dụng hai tính năng đầu tiên của tập dữ liệu Iris:



Hình 8.17:Một hàm nhân chuyển đổi dữ liệu của bạn từ không gian phi tuyến thành không gian tuyến tính

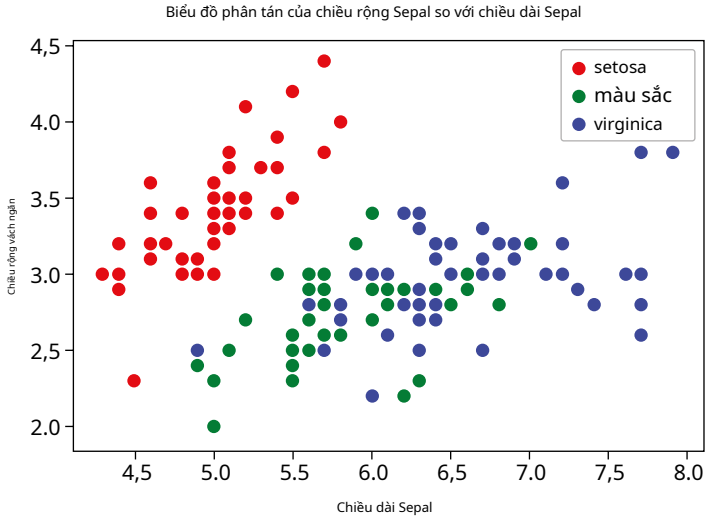
```
X = iris.data [:,: 2] y =           # lấy hai tính năng đầu tiên
iris.target
```

Chúng tôi sẽ vẽ biểu đồ các điểm bằng biểu đồ phân tán (xem Hình 8.18):

```
# - - - vẽ biểu đồ các điểm --- color = ['red',
'green', 'blue']
đối với màu, i, đích ở dạng zip (màu, [0, 1, 2], iris.target_names):
        plt.scatter (X [y == i, 0], X [y == i, 1], color = color, label = target)

plt.xlabel ('Chiều dài phần riêng')
plt.ylabel ('Chiều rộng phần riêng')
plt.legend (loc = 'best', shadow = False, scatterpoints = 1)

plt.title ('Biểu đồ phân tán của chiều rộng Sepal so với chiều dài Sepal') plt.show ()
```

Tiếp theo, chúng ta sẽ sử dụng lớp SVC với hạt nhân tuyến tính:

```
C = 1 # Tham số chính quy SVM clf = svm.SVC (kernel = 'linear',
C = C) .fit (X, y) title = 'SVC với kernel tuyến tính'
```



Hình 8.18: Biểu đồ phân tán của hai tính năng đầu tiên của tập dữ liệu Iris

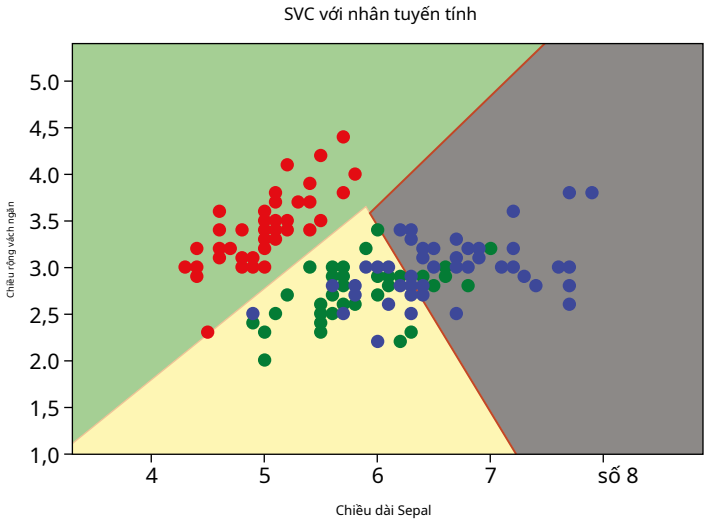
TIỀN BOA **Lưu ý rằng lần này, chúng ta có một tham số mới C. Chúng ta sẽ thảo luận về điều này trong một khoảnh khắc.**

Thay vì vẽ các đường để tách ba nhóm hoa Iris, lần này chúng ta sẽ vẽ các nhóm bằng màu bằng cách sử dụngcontourf ()hàm số:

```
# - - - tối thiểu và tối đa cho tính năng đầu tiên --- x_min, x_max = X [:,
0] .min () - 1, X [:, 0] .max () + 1

# - - - tối thiểu và tối đa cho tính năng thứ hai --- y_min, y_max = X [:,
1] .min () - 1, X [:, 1] .max () + 1

# - - - kích thước bước trong lưới --- h =
(x_max / x_min) / 100

# - - - đưa ra dự đoán cho từng điểm trong xx, yy ---
xx, yy = np.meshgrid (np.arange (x_min, x_max, h),
                      np.arange (y_min, y_max, h))

Z = clf.p Dự đoán (np.c_ [xx.ravel (), yy.ravel ()])

# - - - vẽ kết quả bằng biểu đồ màu --- Z = Z.reshape
(xx.shape)
plt.contourf (xx, yy, Z, cmap = plt.cm.Accent, alpha = 0,8)

# - - - vẽ biểu đồ các điểm rèn luyện --- color =
['red', 'green', 'blue']
đối với màu, i, đích ở dạng zip (màu, [0, 1, 2], iris.target_names):
        plt.scatter (X [y == i, 0], X [y == i, 1], color = color, label = target)
```

```
plt.xlabel ('Chiều dài phần riêng')
plt.ylabel ('Chiều rộng phần riêng')
plt.title (tiêu đề)
plt.legend (loc = 'best', shadow = False, scatterpoints = 1)
```

Hình 8.19 cho thấy các đồ thị phân tán cũng như các nhóm được xác định bởi hạt nhân tuyến tính SVM.



Hình 8.19:Sử dụng hạt nhân tuyến tính SVM

Sau khi đào tạo xong, chúng tôi sẽ thực hiện một số dự đoán:

```
dự đoán = clf.posystem (X) print (np.unique (dự đoán,
return_counts = True))
```

Đoạn mã trước trả về như sau:

```
(mảng ([0, 1, 2]), mảng ([50, 53, 47]))
```

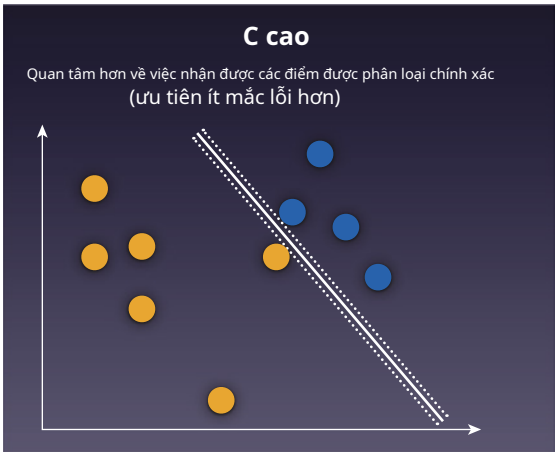
Điều này có nghĩa là sau khi cung cấp cho mô hình với tập dữ liệu Iris, 50 con được phân loại là “setosa”, 53 được phân loại là “versicolor” và 47 được phân loại là “virginica.”

C

Trong phần trước, bạn đã thấy việc sử dụng tham số C:

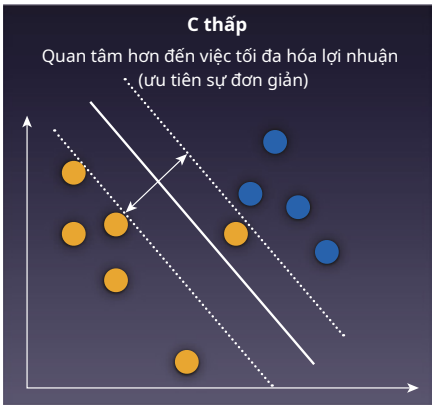
```
C = 1
clf = svm.SVC (kernel = 'linear',C = C) .fit (X, y)
```

C được gọi là *tham số phạt của thuật ngữ lỗi*. Nó kiểm soát sự cân bằng giữa ranh giới quyết định suôn sẻ và phân loại các điểm đào tạo một cách chính xác. Ví dụ: nếu giá trị của C cao, thì thuật toán SVM sẽ tìm cách đảm bảo rằng tất cả các điểm đều được phân loại chính xác. Nhược điểm của điều này là nó có thể dẫn đến biên lợi nhuận hẹp hơn, như trong Hình 8.20.



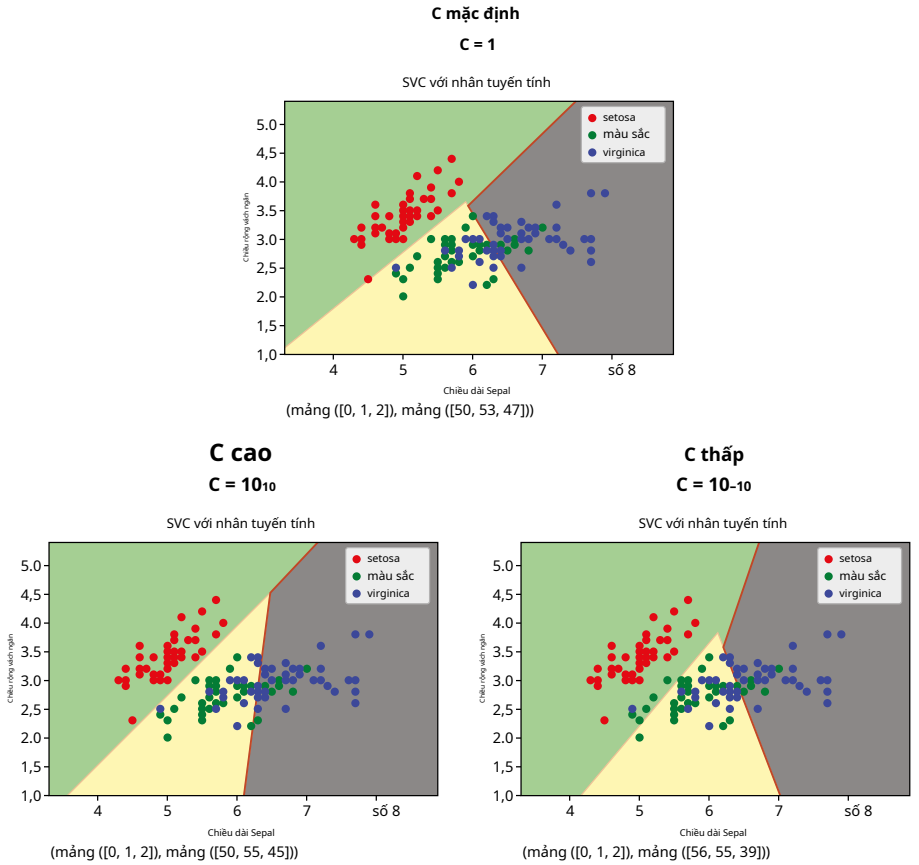
Hình 8.20:Điểm C cao tập trung nhiều hơn vào việc nhận được các điểm được phân loại chính xác

Ngược lại, điểm C thấp hơn sẽ hướng đến biên độ rộng nhất có thể, nhưng nó sẽ dẫn đến một số điểm bị phân loại không chính xác (xem Hình 8.21).



Hình 8.21:C thấp nhằm mục đích cho lợi nhuận rộng nhất, nhưng có thể phân loại một số điểm không chính xác

Hình 8.22 cho thấy ảnh hưởng của việc thay đổi giá trị của C khi áp dụng thuật toán hạt nhân tuyến tính SVM. Kết quả phân loại xuất hiện ở cuối mỗi biểu đồ.



Hình 8.22:Sử dụng SVM với các giá trị khác nhau của C

Lưu ý rằng khi C là 1 hoặc 10_{10}, không có quá nhiều khác biệt giữa các kết quả phân loại. Tuy nhiên, khi C nhỏ (10_{-10}), bạn có thể thấy rằng một số điểm (thuộc về "versicolor" và "virginica") hiện bị phân loại nhầm thành "setosa."

TIỀN BOA **Tóm lại, C thấp làm cho bề mặt quyết định mịn trong khi cố gắng phân loại *phần lớn*điểm, trong khi điểm C cao cố gắng phân loại*tất cả các*của các điểm một cách chính xác.**

Hạt nhân hàm cơ sở hướng tâm (RBF)

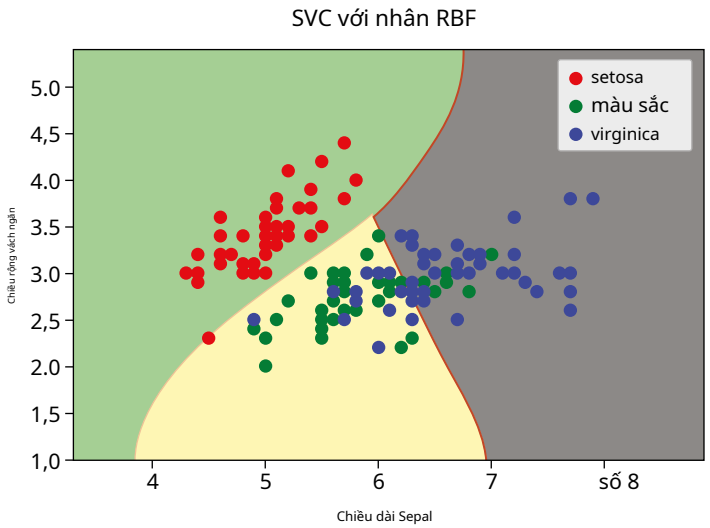
Bên cạnh hạt nhân tuyến tính mà chúng ta đã thấy cho đến nay, có một số hạt nhân phi tuyến thường được sử dụng:

- *Chức năng Radial Basis (RBF)*, còn được biết là*Hạt nhân Gaussian*
- Đa thức

Đầu tiên, RBF, cung cấp giá trị cho mỗi điểm dựa trên khoảng cách của nó từ điểm gốc hoặc *đã sửa*trung tâm, thường là trên không gian Euclide. Hãy sử dụng cùng một ví dụ mà chúng ta đã sử dụng trong phần trước, nhưng lần này sửa đổi hạt nhân để sử dụngrbf:

```
C = 1
clf = svm.SVC (kernel ='rbf', gamma = 'auto', C = C) .fit (X, y) title = 'SVC with
RBF kernel'
```

Hình 8.23 cho thấy cùng một mẫu được đào tạo bằng cách sử dụng nhân RBF.



Hình 8.23:Tập dữ liệu Iris được đào tạo bằng cách sử dụng nhân RBF

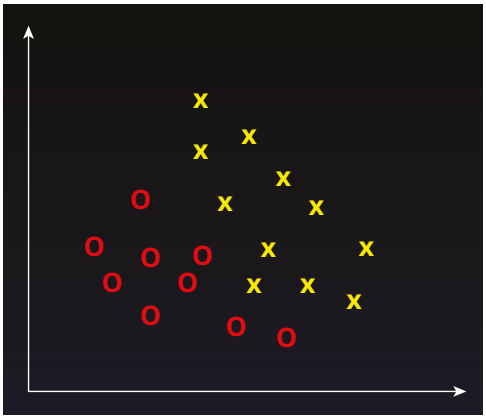
Gamma

Nếu bạn xem kỹ đoạn mã, bạn sẽ phát hiện ra một thông số mới—gamma. *Gamma*xác định mức độ ảnh hưởng của một ví dụ đào tạo. Xem xét tập hợp các điểm được thể hiện trong Hình 8.24. Có hai loại điểm - x và o.

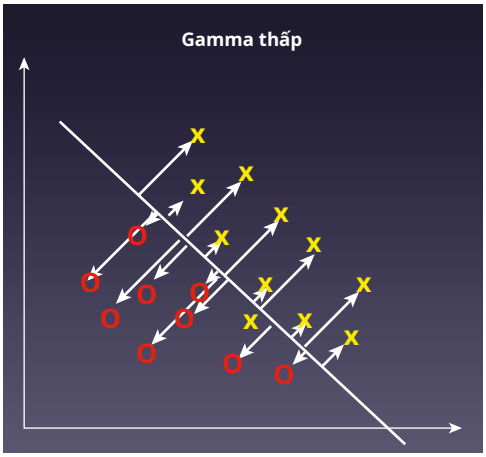
Giá trị Gamma thấp chỉ ra rằng mọi điểm đều có khoảng cách xa (xem Hình 8.25). Mặt khác, Gamma cao có nghĩa là các điểm gần ranh giới quyết định nhất có phạm vi tiếp cận gần nhau. Giá trị của Gamma càng cao, nó sẽ càng cố gắng điều chỉnh chính xác tập dữ liệu huấn luyện, dẫn đến việc trang bị quá mức (xem Hình 8.26).

Hình 8.27 cho thấy sự phân loại của các điểm sử dụng RBF, với các giá trị khác nhau của C và Gamma.

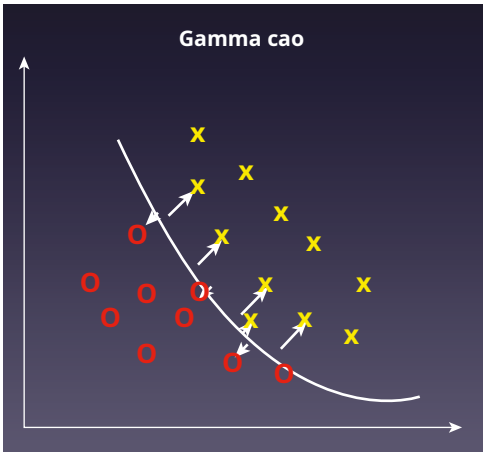
Lưu ý rằng nếu Gamma cao (10), hiện tượng overfitting xảy ra. Bạn cũng có thể thấy từ hình này rằng giá trị của C điều khiển độ trơn của đường cong.



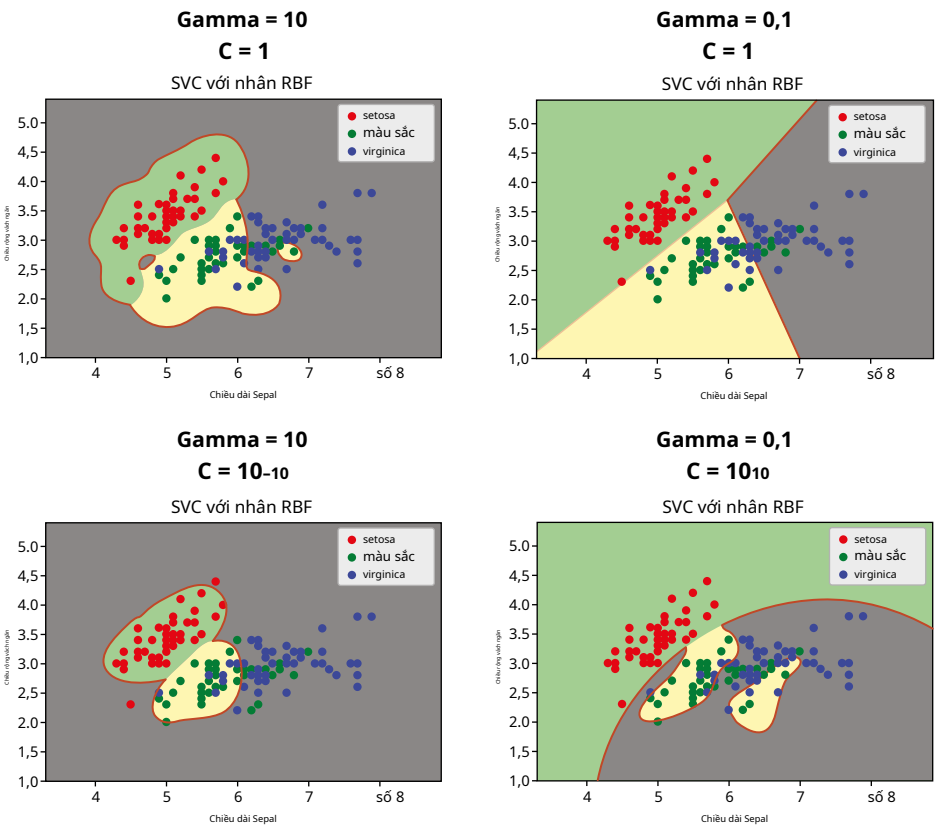
Hình 8.24:Tập hợp các điểm thuộc hai lớp



Hình 8.25: Giá trị Gamma thấp cho phép mọi điểm có phạm vi tiếp cận bằng nhau



Hình 8.26: Giá trị Gamma cao tập trung nhiều hơn vào các điểm gần ranh giới



Hình 8.27:Tác động của việc phân loại các điểm bằng cách sử dụng các giá trị khác nhau của C và Gamma

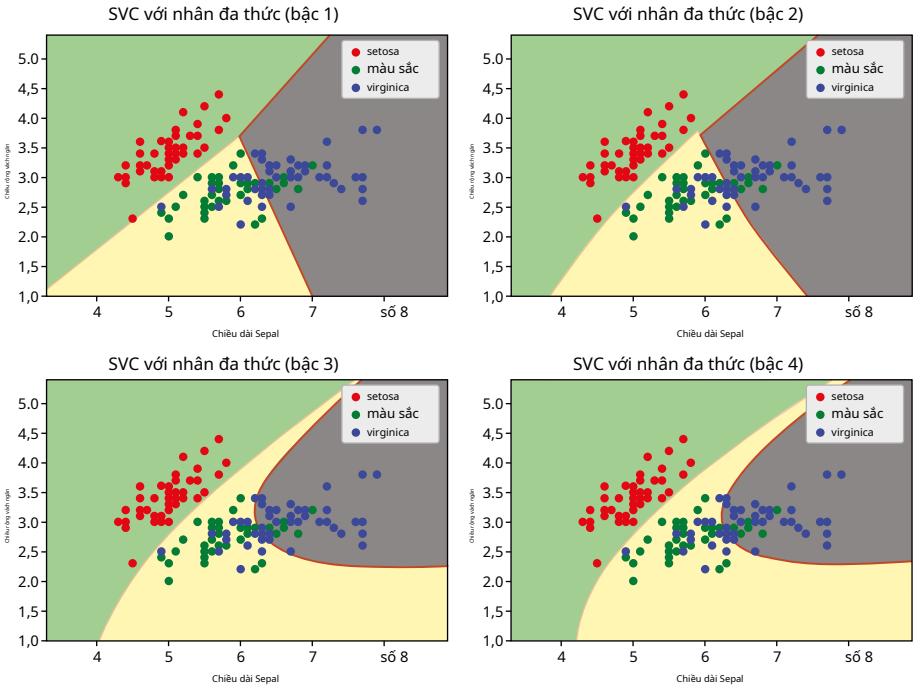
TIỀN BOA **Tóm lại, C kiểm soát độ mịn của đường biên và Gamma xác định xem các điểm có được trang bị quá mức hay không.**

Nhân đa thức

Một loại nhân khác được gọi là*nhân đa thức*. Nhân đa thức bậc 1 tương tự như nhân tuyến tính. Các nhân đa thức bậc cao hơn tạo ra một ranh giới quyết định linh hoạt hơn. Đoạn mã sau đây cho thấy tập dữ liệu Iris được đào tạo bằng cách sử dụng nhân đa thức với mức độ 4:

```
C = 1 # thông số chính quy SVM
clf = svm.SVC (kernel = 'poly', Deg = 4, C = C, gamma = 'auto'). fit (X, y) title = 'SVC with
polynomial (level 4) kernel'
```

Hình 8.28 cho thấy tập dữ liệu được phân tách với các nhân đa thức từ mức 1 đến 4.



Hình 8.28: Việc phân loại tập dữ liệu Iris sử dụng nhân đa thức ở các mức độ khác nhau

Sử dụng SVM cho các sự cố trong đời thực

Chúng ta sẽ kết thúc chương này bằng cách áp dụng SVM vào một vấn đề phổ biến trong cuộc sống hàng ngày của chúng ta. Hãy xem xét tập dữ liệu sau (được lưu trong một tệp có tên house_size_prices_svm. csv)bao gồm quy mô của các ngôi nhà và giá chào bán của chúng (tính bằng nghìn) cho một khu vực cụ thể:

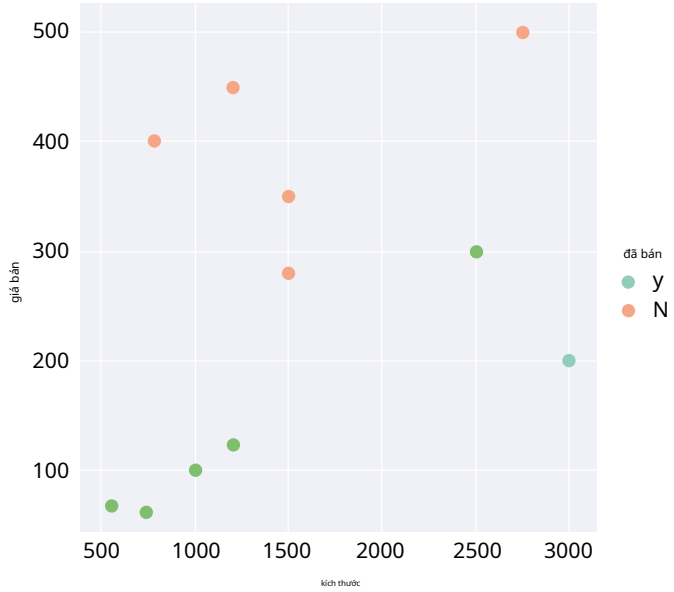
```
kích thước, giá cả, đã bán
550,50, y
1000,100, y
1200,123, y
1500,350, n
3000.200, y
2500.300, y
750, 45, y
1500,280, n
780,400, n
1200, 450, n
2750, 500, n
```

Cột thứ ba cho biết ngôi nhà đã được bán hay chưa. Sử dụng tập dữ liệu này, bạn muốn biết liệu một ngôi nhà có giá chào bán cụ thể có thể bán được hay không.

Đầu tiên, hãy vạch ra các điểm:

```
% matplotlib inline

nhập gấu trúc dưới dạng pd nhập numpy
dưới dạng np từ sklearn nhập svm nhập
matplotlib.pyplot dưới dạng plt

nhập seaborn dưới dạng sns; sns.set (font_scale = 1.2)

data = pd.read_csv ('house_sizes_prices_svm.csv')

sns.lmplot ('size', 'price',
                    data = dữ liệu,
                    hue = 'đã bán',
                    Palette = 'Set2',
                    fit_reg = Sai,
                    scatter_kws = {"s": 50});
```

Hình 8.29 cho thấy các điểm được vẽ dưới dạng biểu đồ phân tán.



Hình 8.29:Vẽ các điểm trên một biểu đồ phân tán

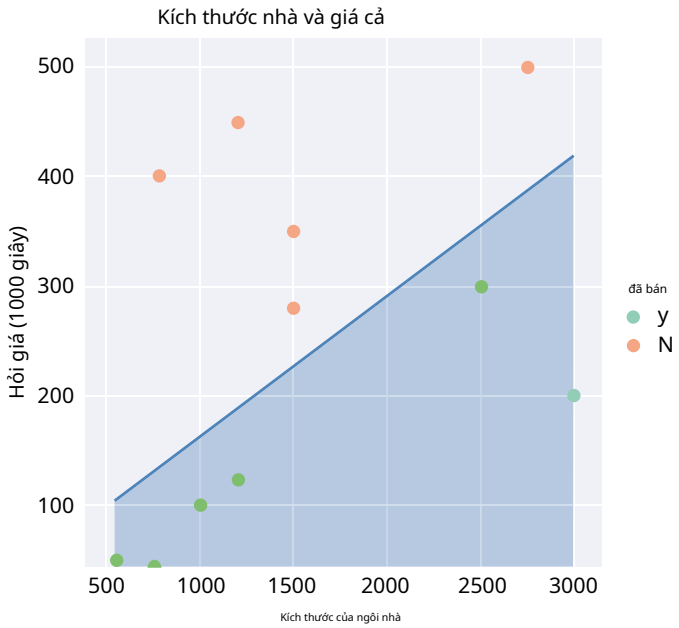
Bằng mắt thường, bạn có thể thấy rằng đây là một vấn đề có thể được giải quyết với hạt nhân tuyến tính của SVM:

```
X = data [['size', 'price']]. Giá trị
y = np.where (data ['sell'] == 'y', 1, 0) # - 1 cho Y và 0 cho N --- model = svm.SVC (kernel
= 'linear'). fit ( X, y)
```

Với mô hình được đào tạo, bây giờ bạn có thể thực hiện các dự đoán và vẽ hai lớp:

```
# - - - tối thiểu và tối đa cho tính năng đầu tiên --- x_min, x_max = X [:,
0] .min () - 1, X [:, 0] .max () + 1

# - - - tối thiểu và tối đa cho tính năng thứ hai --- y_min, y_max = X [:,
1] .min () - 1, X [:, 1] .max () + 1

# - - - kích thước bước trong lưới --- h =
(x_max / x_min) / 20

# - - - đưa ra dự đoán cho từng điểm trong xx, yy ---
xx, yy = np.meshgrid (np.arange (x_min, x_max, h),
                      np.arange (y_min, y_max, h))

Z = model.p Dự đoán (np.c_ [xx.ravel (), yy.ravel ()])

# - - - vẽ kết quả bằng biểu đồ màu --- Z = Z.reshape
(xx.shape)
plt.contourf (xx, yy, Z, cmap = plt.cm.Blues, alpha = 0,3)

plt.xlabel ('Kích thước ngôi nhà') plt.ylabel ('Giá bán (1000 giây)')
plt.title ("Kích thước nhà và giá bán")
```

Hình 8.30 cho thấy các điểm và các lớp mà chúng thuộc về.



Hình 8.30:Tách các điểm thành hai lớp

Bây giờ, bạn có thể thử dự đoán xem một ngôi nhà có quy mô nhất định với giá bán cụ thể sẽ có thể bán được hay không:

```
def will_it_sell (kích thước, giá cả):
        if (model.p Dự đoán ([[size, price]])) == 0:
                print ('Sẽ không bán!') else:

                print ('Sẽ bán!')

# - - - thực hiện một số dự đoán ---
will_it_sell (2500, 400) will_it_sell    # Sẽ không bán!
(2500, 200)                              # Sẽ bán!
```

Bản tóm tắt

Trong chương này, bạn đã học về cách Máy hỗ trợ hỗ trợ trong các vấn đề phân loại. Bạn đã học về công thức tìm siêu phẳng, cũng như hai lề đi kèm. May mắn thay, Scikit-learning cung cấp các lớp cần thiết cho các mô hình đào tạo sử dụng SVM và với các tham số được trả về, bạn có thể vẽ siêu phẳng và lề một cách trực quan để bạn có thể hiểu cách SVM hoạt động. Bạn cũng đã học về các hạt nhân khác nhau mà bạn có thể áp dụng cho các thuật toán SVM của mình để tập dữ liệu có thể được phân tách một cách tuyến tính.