CHƯƠNG

9

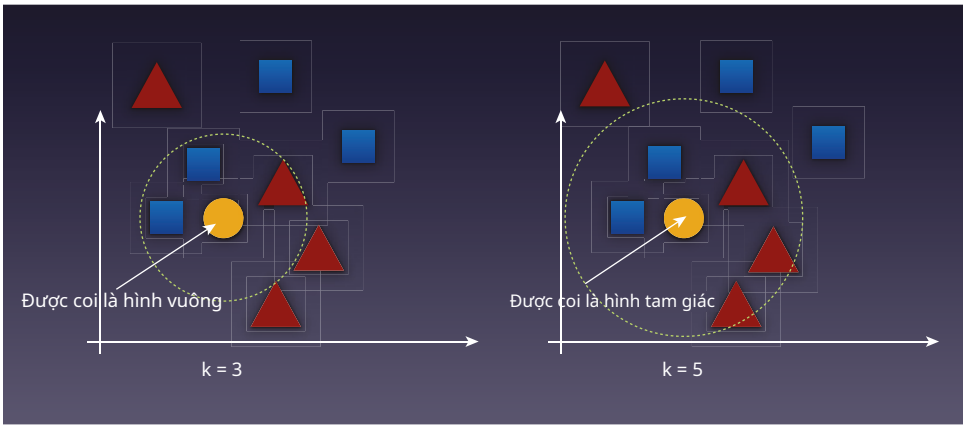
Lê giám sát arning— Phân loại sử dụng K - Gần nhất Hàng xóm rs (KNN)

K-Nearest Neighbors là gì?

Cho đến thời điểm này, chúng ta đã thảo luận về ba thuật toán học có giám sát: hồi quy tuyến tính, hồi quy hậu cần và máy vectơ hỗ trợ. Trong chương này, chúng ta sẽ đi sâu vào một thuật toán học máy được giám sát khác được gọi là *K-Hàng xóm gần nhất (KNN)*.

KNN là một thuật toán tương đối đơn giản so với các thuật toán khác mà chúng ta đã thảo luận trong các chương trước. Nó hoạt động bằng cách so sánh khoảng cách của cá thể truy vấn với các mẫu huấn luyện khác và chọn K-láng giềng gần nhất (do đó có tên là nó). Sau đó, phần lớn các lớp K-láng giềng này là dự đoán của cá thể truy vấn.

Hình 9.1 đã tổng kết điều này một cách độc đáo. Khi k = 3, ba lân cận gần nhất của hình tròn là hai hình vuông và một hình tam giác. Dựa trên quy tắc đơn giản của đa số, hình tròn được phân loại là hình vuông. Nếu k = 5, thì năm hàng xóm gần nhất là hai hình vuông và ba hình tam giác. Do đó, hình tròn được phân loại là hình tam giác.



Hình 9.1: Việc phân loại một điểm phụ thuộc vào phần lớn các nước láng giềng của nó

TIỀN BOA **KNN đôi khi cũng được sử dụng để hồi quy ngoài việc phân loại.**
Ví dụ, nó có thể được sử dụng để tính giá trị trung bình của mục tiêu số của K-láng giềng gần nhất. Tuy nhiên, đối với chương này, chúng tôi chỉ tập trung vào việc sử dụng phổ biến hơn của nó như một thuật toán phân loại.

Triển khai KNN bằng Python

Bây giờ bạn đã thấy KNN hoạt động như thế nào, hãy thử triển khai KNN từ đầu bằng Python. Như thường lệ, trước tiên hãy nhập các mô-đun mà chúng ta cần:

```
nhập gấu trúc dưới dạng
pd nhập numpy dưới
dạng toán tử nhập np
import seaborn as sns import
matplotlib.pyplot as plt
```

Vẽ các điểm

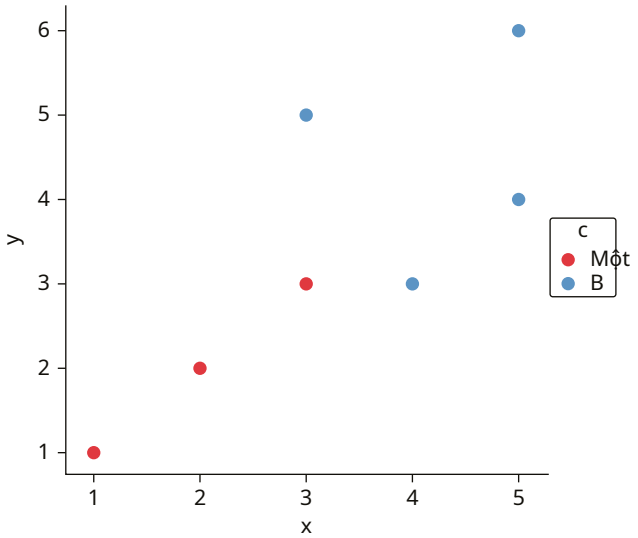
Đối với ví dụ này, bạn sẽ sử dụng một tệp có tênknn.csvchứa dữ liệu sau:

```
x, y, c
1,1, A
2,2, A
4,3, B
3,3, A
3,5, B
5,6, B
5,4, B
```

Như chúng ta đã làm trong các chương trước, một cách tốt là vẽ các điểm bằng cách sử dụng Seaborn:

```
data = pd.read_csv ("knn.csv") sns.lmplot
('x', 'y', data = data,
                    hue = 'c', Palette = 'Set1', fit_reg = False,
                    scatter_kws = {"s": 70})
plt.show ()
```

Hình 9.2 cho thấy sự phân bố của các điểm khác nhau. Điểm thuộc loại A được hiển thị bằng màu đỏ trong khi điểm thuộc loại B được hiển thị bằng màu xanh lam.



Hình 9.2:Vẽ các điểm một cách trực quan

Tính khoảng cách giữa các điểm

Để tìm lân cận gần nhất của một điểm cho trước, bạn cần tính khoảng cách Euclide giữa hai điểm.

TIỀN BOA **Trong hình học, không gian Euclide bao gồm mặt phẳng Euclid hai chiều, không gian ba chiều của hình học Euclid và các không gian tương tự có chiều cao hơn.**

Cho hai điểm, $P = (P_1, P_2, ,P_N$ và $) = (q_1 q_2,,.q_N,$ khoảng cách giữa P và được cho bởi công thức sau:

$$\sqrt{\left((q_1 - P_2)\ + (q_2 - P_2)^2 + \ldots + (q_N - P_N)^2\right)}$$

Dựa trên công thức này, bây giờ bạn có thể xác định một hàm có têneuclidean _ khoảng cách()như sau:

```
# - - - để tính khoảng cách giữa hai điểm --- def euclid_distance (pt1,
pt2, thứ nguyên):
    khoảng cách = 0
    cho x trong phạm vi (thứ nguyên):
        khoảng cách + = np.square (pt1 [x] - pt2 [x]) trả về
    np.sqrt (khoảng cách)
```

CácEuclidean _ khoảng cách ()hàm có thể tìm khoảng cách giữa hai điểm trong bất kỳ chiều nào. Đối với ví dụ này, các điểm mà chúng tôi đang xử lý là ở dạng 2D.

Thực hiện KNN

Tiếp theo, xác định một hàm có tênknn (),trong đó lấy điểm rèn luyện, điểm kiểm tra và giá trị của k:

```
# - - - mô hình KNN của riêng chúng tôi ---
def knn (training_points, test_point, k):
    khoảng cách = {}

    # - - - số trục chúng tôi đang xử lý --- kích thước =
    test_point.shape [1]

    # - - tính toán khoảng cách euclid giữa mỗi
    # point trong dữ liệu đào tạo và dữ liệu kiểm tra cho x
    trong phạm vi (len (training_points)):
        dist = euclidean_distance (test_point, training_points.iloc [x],
                                   kích thước)
        # - - - ghi lại khoảng cách cho mỗi điểm đào tạo --- khoảng cách [x] =
        dist [0]

    # - - - sắp xếp khoảng cách ---
    sorted_d = đã sắp xếp (khoảng cách.items (), key = operator.itemgetter (1))

    # - - - để lưu trữ hàng xóm --- hàng xóm
    = []

    # - - - trích xuất k lân cận hàng đầu --- cho x
    trong khoảng (k):
        Neighbor.append (sorted_d [x] [0])

    # - - - đối với mỗi hàng xóm được tìm thấy, hãy tìm lớp của nó ---
    class_counter = {}
    cho x trong phạm vi (len (hàng xóm)):
        # - - - tìm ra lớp cho điểm cụ thể đó --- cls = training_points.iloc
        [Neighbor [x]] [- 1]
```

```
            nếu cls trong class_counter:
                class_counter [cls] + = 1 khác:

                class_counter [cls] = 1

        # - - - sắp xếp class_counter theo thứ tự giảm dần ---
        sorted_counter = sorted (class_counter.items (),
                                 key = operator.itemgetter (1),
                                 reverse = True)

        # - - - trả về lớp có số lượng nhiều nhất, cũng như
        # hàng xóm được tìm thấy ---
        return (sorted_counter [0] [0], hàng xóm)
```

Hàm trả về lớp mà điểm kiểm tra thuộc về, cũng như các chỉ số của tất cả k lân cận gần nhất.

Đưa ra dự đoán

Vớiknn ()chức năng được xác định, bây giờ bạn có thể đưa ra một số dự đoán:

```
# - - - điểm kiểm tra ---
test_set = [[3,3,9]]
test = pd.DataFrame (test_set) cls, Neighbor =
knn (data, test, 5) print ("Predicted Class:" +
cls)
```

Đoạn mã trước đó sẽ in ra kết quả sau:

```
Lớp dự đoán: B
```

Hình dung các giá trị khác nhau của K

Sẽ rất hữu ích khi có thể hình dung tác động của việc áp dụng các giá trị khác nhau của k. Đoạn mã sau đây vẽ một loạt các vòng tròn đồng tâm xung quanh điểm kiểm tra dựa trên các giá trị của k, nằm trong khoảng từ 7 đến 1, với khoảng thời gian là –2:

```
# - - - tạo bản đồ màu cho biểu đồ phân tán ---
# - - - nếu cột 'c' là A thì sử dụng Màu đỏ, còn lại thì sử dụng Màu xanh lam
--- Colors = ['r' if i == 'A' else 'b' for i in data ['c']] ax = data.plot (kind =
'scatter', x = 'x', y = 'y', c = Colors) plt.xlim (0,7)

plt.ylim (0,7)

# - - - vẽ biểu đồ điểm kiểm tra --- plt.plot (test_set [0] [0], test_set [0] [1], "yo",
markersize = '9')

cho k trong phạm vi (7,0, -2):
    cls, hàng xóm = knn (dữ liệu, kiểm tra, k)
```

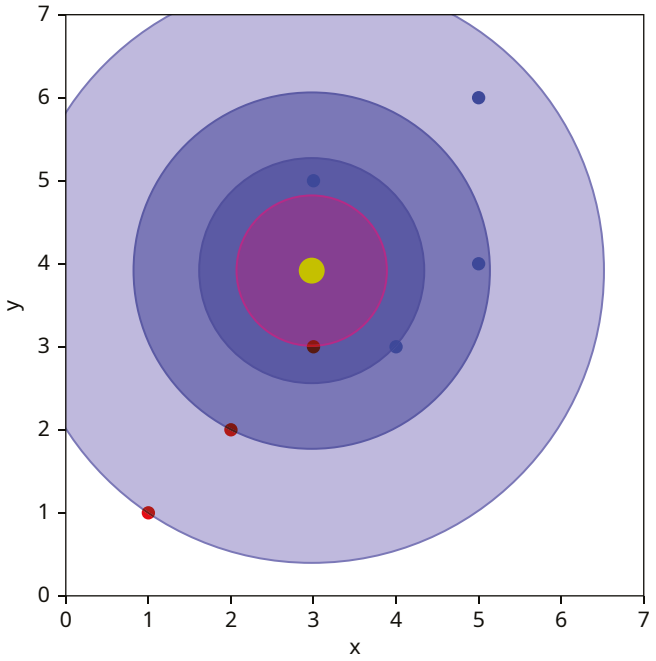
```
        print ("============")
        print ("k =", k) print ("Lớp",
        cls)
        print ("Hàng xóm")
        print (data.iloc [hàng xóm])

        furthest_point = data.iloc [hàng xóm] .tail (1)

        # - - - vẽ một vòng tròn nối điểm kiểm tra
        # và điểm xa nhất ---
        radius = euclidean_distance (kiểm tra, furthest_point.iloc [0], 2)

        # - - - hiển thị vòng tròn màu đỏ nếu phân loại là A,
        # else hiển thị vòng tròn màu xanh lam --- c
        = 'r' if cls == 'A' else 'b'
        circle = plt.Circle ((test_set [0] [0], test_set [0] [1]),
                                        bán kính, màu = c, alpha = 0,3)
        ax.add_patch (vòng tròn)

plt.gca (). set_aspect ('bằng', điều chỉnh = 'hộp') plt.show ()
```

Đoạn mã trước in ra kết quả sau:

```
============
k = 7
Hạng B
Người hàng xóm
     xyc
3 3 3 A
4 3 5 B
2 4 3 B
6 5 4 B
1 2 2 A
5 5 6 B
0 1 1 A
============
k = 5
Hạng B
Người hàng xóm
     xyc
3 3 3 A
4 3 5 B
2 4 3 B
6 5 4 B
1 2 2 A
============
k = 3
Hạng B
Người hàng xóm
```

```
    xyc
3 3 3 A
4 3 5 B
2 4 3 B
============
k = 1
Hạng A
Người hàng xóm
    xyc
3 3 3 A
```

Hình 9.3 cho thấy chuỗi các vòng tròn xoay quanh điểm thử nghiệm, với các giá trị khác nhau của k. Vòng trong cùng là k = 1, với vòng ngoài tiếp theo là k = 3, v.v. Như bạn có thể thấy, nếu k = 1, vòng tròn có màu đỏ, nghĩa là điểm màu vàng đã được phân loại là lớp A. Nếu vòng tròn màu xanh lam, điều này có nghĩa là điểm màu vàng đã được phân loại là lớp B. Điều này được thể hiện rõ ràng trong ba vòng tròn bên ngoài.



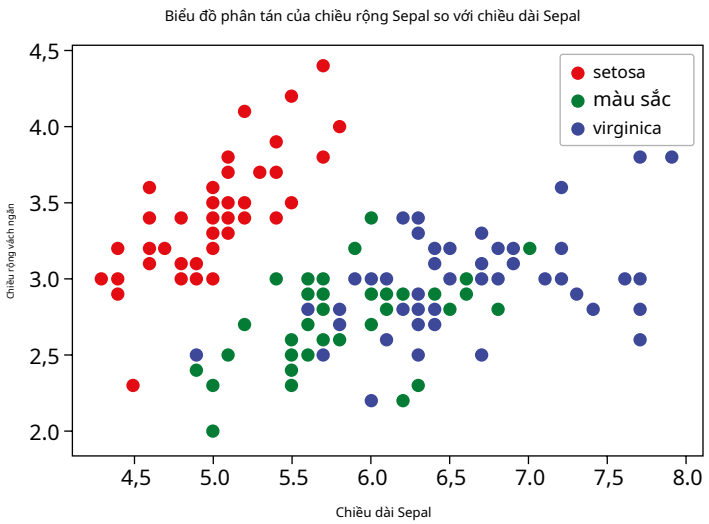
Hình 9.3: Việc phân loại điểm vàng dựa trên các giá trị khác nhau của k

Sử dụng Lớp học KNeighborsClassifier của Scikit-Learn cho KNN

Bây giờ bạn đã thấy cách KNN hoạt động và cách nó có thể được triển khai theo cách thủ công bằng Python, hãy sử dụng cách triển khai được cung cấp bởi Scikit-learning.

Đoạn mã sau tải tập dữ liệu Iris và vẽ nó ra bằng cách sử dụng biểu đồ phân tán:

```
% matplotlib inline
nhập gấu trúc dưới dạng pd
nhập numpy dưới dạng np
nhập matplotlib.patches dưới dạng khớp từ
sklearn nhập svm, bộ dữ liệu nhập
matplotlib.pyplot dưới dạng plt

iris = datasets.load_iris ()

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

# - - - 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 ()
```



Hình 9.4:Vẽ biểu đồ chiều rộng Sepal so với chiều dài Sepal trong một biểu đồ phân tán

Hình 9.4 cho thấy biểu đồ phân tán của chiều rộng Sepal so với chiều dài Sepal.

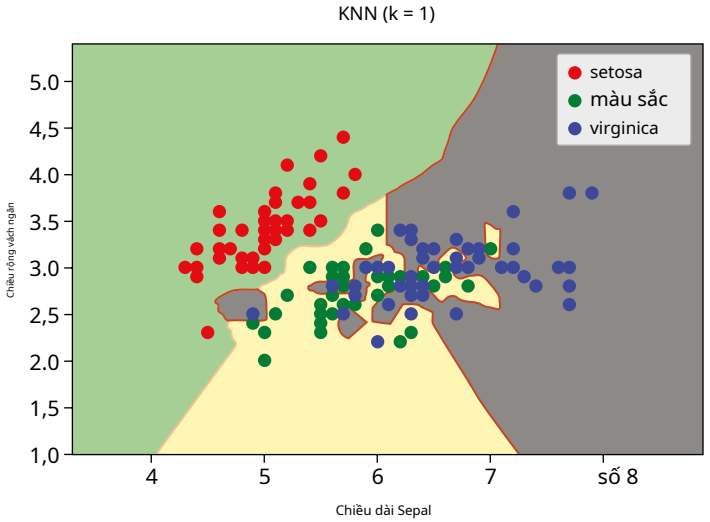
Khám phá các giá trị khác nhau của K

Bây giờ chúng ta có thể sử dụng Scikit-learning'sKNeighborsClassifierđể giúp chúng tôi đào tạo một mô hình trên tập dữ liệu Iris sử dụng KNN. Để bắt đầu, hãy sử dụng ak trong số 1:

```
từ sklearn.neighbors nhập KNeighborsClassifier

k = 1
# - - - mô hình học tức thời --- knn =
KNeighborsClassifier (n_neighbors = k)

# - - - phù hợp với mô hình ---
knn.fit (X, y)

# - - - 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 = knn.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 (f'KNN (k = {k}) ')
plt.legend (loc = 'best', shadow = False, scatterpoints = 1)

dự đoán = knn.p Dự đoán (X)

# - - phân loại dựa trên dự đoán --- print (np.unique (dự đoán,
return_counts = True))
```

Đoạn mã trước đó tạo ra một*lưới điện*(một lưới hình chữ nhật gồm các giá trị) của các điểm nằm rải rác trên các trục x và y. Mỗi điểm sau đó được sử dụng để dự đoán và kết quả được vẽ bằng biểu đồ màu.

Hình 9.5 cho thấy ranh giới phân loại sử dụng ak là 1. Lưu ý rằng đối với k = 1, bạn thực hiện dự đoán của mình chỉ dựa trên một mẫu duy nhất — hàng xóm gần nhất của bạn. Điều này làm cho dự đoán của bạn rất nhạy cảm với tất cả các loại sai lệch, chẳng hạn như ngoại lệ, ghi nhãn sai, v.v. Nói chung, đặt k = 1 thường dẫn đến*quá sức*, và kết quả là dự đoán của bạn thường không chính xác lắm.



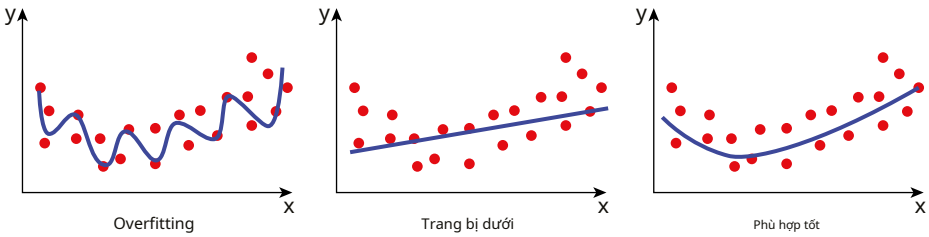
Hình 9.5: Ranh giới phân loại dựa trên k = 1

TIỀN BOA ***Overfitting*trong học máy có nghĩa là mô hình bạn đã đào tạo phù hợp**

dữ liệu đào tạo quá tốt. Điều này xảy ra khi tất cả các tiếng ồn và biến động trong dữ liệu đào tạo của bạn được thu thập trong quá trình đào tạo. Nói một cách dễ hiểu, điều này có nghĩa là mô hình của bạn đang rất cố gắng để phù hợp hoàn hảo với tất cả dữ liệu của bạn. Vấn đề quan trọng với một mô hình được trang bị quá nhiều là nó sẽ không hoạt động tốt với dữ liệu mới, chưa được nhìn thấy.

***Trang bị dưới*mặt khác, xảy ra khi một mô hình học máy không thể nắm bắt chính xác xu hướng cơ bản của dữ liệu. Cụ thể, mô hình không phù hợp với dữ liệu đủ tốt.**

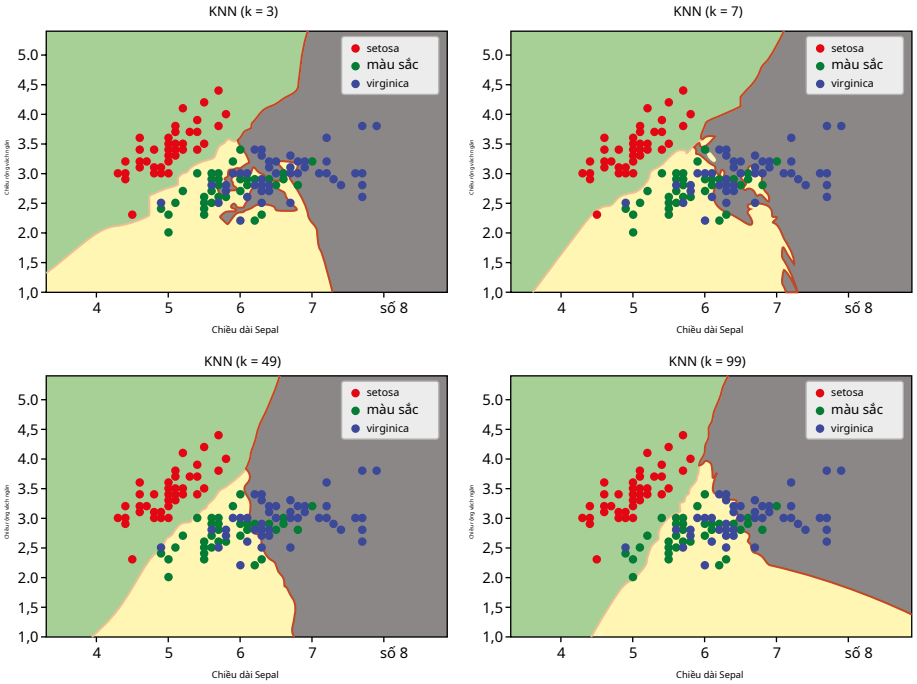
Hình 9.6 cho thấy một cách dễ hiểu về trang phục thừa, trang phục thiếu và nói chung là vừa vặn.



Hình 9.6: Hiểu khái niệm về trang phục thừa, trang phục thiếu và vừa vặn

Đối với KNN, việc đặt k thành giá trị cao hơn có xu hướng làm cho dự đoán của bạn chắc chắn hơn chống lại nhiễu trong dữ liệu của bạn.

Sử dụng cùng một đoạn mã, hãy thay đổi các giá trị của k. Hình 9.7 cho thấy các phân loại dựa trên bốn giá trị khác nhau của k.



Hình 9.7:Ảnh hưởng của việc thay đổi các giá trị của k

Lưu ý rằng khi k tăng, ranh giới trở nên mịn hơn. Nhưng nó cũng có nghĩa là nhiều điểm hơn sẽ được phân loại không chính xác. Khi k tăng đến một giá trị lớn, *thiếu trang bị*xảy ra.

Vấn đề quan trọng với KNN là làm thế nào để bạn tìm ra giá trị lý tưởng của k để sử dụng?

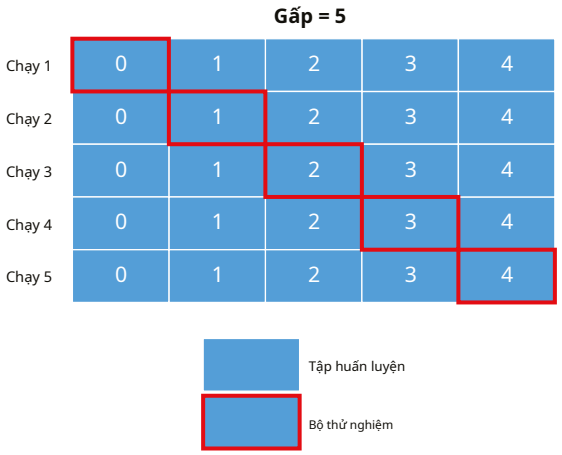
Xác thực chéo

Trong vài chương trước, bạn đã chứng kiến rằng chúng tôi chia bộ dữ liệu của mình thành hai bộ riêng lẻ — một bộ để đào tạo và một bộ để thử nghiệm. Tuy nhiên, dữ liệu trong tập dữ liệu của bạn có thể không được phân phối đồng đều và kết quả là tập thử nghiệm của bạn có thể quá đơn giản hoặc quá khó dự đoán, do đó rất khó để biết liệu mô hình của bạn có hoạt động tốt hay không.

Thay vì sử dụng một phần dữ liệu để đào tạo và một phần để thử nghiệm, bạn có thể chia dữ liệu thành*k-nếp gấp*và đào tạo các mô hình*k*lần, luân phiên các tập huấn luyện và kiểm tra. Bằng cách đó, mỗi điểm dữ liệu hiện đang được sử dụng để đào tạo và thử nghiệm.

TIỀN BOA **Đừng nhầm lẫn*k*trong k-nếp gấp với*k*trong KNN — chúng không liên quan.**

Hình 9.8 cho thấy một tập dữ liệu được chia thành năm nếp gấp (khối). Trong lần chạy đầu tiên, các khối 1, 2, 3 và 4 sẽ được sử dụng để huấn luyện mô hình. Khối 0 sẽ được sử dụng để kiểm tra mô hình. Trong lần chạy tiếp theo, các khối 0, 2, 3 và 4 sẽ được sử dụng để đào tạo và khối 1 sẽ được sử dụng để kiểm tra, v.v.



Hình 9.8:Cách xác thực chéo hoạt động

Vào cuối mỗi lần chạy, mô hình được tính điểm. Vào cuối k-run, điểm được tính trung bình. Điểm số trung bình này sẽ cung cấp cho bạn một dấu hiệu tốt về cách thuật toán của bạn hoạt động tốt như thế nào.

TIỀN BOA **Mục đích của*xác nhận chéo*không phải để đào tạo mô hình của bạn, mà nó là để kiểm tra mô hình. Xác thực chéo rất hữu ích khi bạn cần so sánh các thuật toán học máy khác nhau để xem chúng hoạt động như thế nào với tập dữ liệu đã cho. Khi thuật toán được chọn, bạn sẽ sử dụng tất cả dữ liệu để đào tạo mô hình.**

Tham số-Điều chỉnh K

Bây giờ bạn đã hiểu xác thực chéo, hãy sử dụng nó trên tập dữ liệu Iris của chúng tôi. Chúng tôi sẽ đào tạo mô hình bằng cách sử dụng tất cả bốn tính năng và đồng thời chúng tôi sẽ sử dụng xác nhận chéo trên tập dữ liệu bằng cách sử dụng 10 lần. Chúng tôi sẽ làm điều này cho mỗi giá trị của k:

```
từ sklearn.model_selection nhập cross_val_score

# - - - giữ điểm cv (xác thực chéo) --- cv_scores = []


# - - - sử dụng tất cả các tính
năng --- X = iris.data [:,: 4]
y = iris.target

# - - - số lần gấp --- lần gấp =
10

# - - - tạo danh sách K lẻ cho KNN ---
ks = danh sách (phạm vi (1, int (len (X) * ((nếp gấp - 1) / nếp gấp))))

# - - - loại bỏ tất cả các bội số của 3 --- ks = [k
với k tính bằng ks nếu k% 3! = 0]

# - - - thực hiện xác nhận chéo k-lần --- cho k trong
ks:
    knn = KNeighborsClassifier (n_neighbors = k)

    # - - - thực hiện xác thực chéo và trả về độ chính xác trung bình --- score = cross_val_score
    (knn, X, y, cv = fold, score = 'precision') mean = score.mean ()

    cv_scores.append (có nghĩa là)
    print (k, có nghĩa là)
```

Thư viện Scikit-learning cung cấpcross _ val _ score ()chức năng thực hiện xác thực chéo cho bạn một cách tự động và nó trả về các chỉ số mà bạn muốn (ví dụ: độ chính xác).

Khi sử dụng xác thực chéo, hãy lưu ý rằng tại bất kỳ thời điểm nào, sẽ có*((nếp gấp-1) / fold) * total_rows*có sẵn để đào tạo. Điều này là bởi vì*(1 / lần gấp) * tổng_số* sẽ được sử dụng để thử nghiệm.

Đối với KNN, có ba quy tắc mà bạn phải tuân thủ:

- Giá trị của k không được vượt quá số hàng để huấn luyện.
- Giá trị của k phải là một số lẻ (để bạn có thể tránh các trường hợp có sự ràng buộc giữa các lớp) đối với một bài toán hai lớp.
- Giá trị của k không được là bội số của số lớp (để tránh ràng buộc, tương tự như điểm trước đó).

Vì thế ***ks*** danh sách trong đoạn mã trước đó sẽ chứa các giá trị sau:

```
[1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19, 20, 22, 23, 25, 26, 28, 29, 31, 32, 34, 35, 37 , 38, 40, 41, 43,
44, 46, 47, 49, 50, 52, 53, 55,
56, 58, 59, 61, 62, 64, 65, 67, 68, 70, 71, 73, 74, 76, 77, 79, 80, 82,
83, 85, 86, 88, 89, 91, 92, 94, 95, 97, 98, 100, 101, 103, 104, 106,
107, 109, 110, 112, 113, 115, 116, 118, 119, 121, 122, 124, 125, 127,
128, 130, 131, 133, 134]
```

Sau khóa đào tạo, ***điểm cv*** _sẽ chứa một danh sách các độ chính xác dựa trên các giá trị khác nhau của k:

```
1 0,96
2 0,9533333333333334
4 0,9666666666666666
5 0,9666666666666668
7 0,9666666666666668
8 0,9666666666666668
10 0,9666666666666668
11 0,9666666666666668
13 0.9800000000000001
14 0,9733333333333334
. . .
128 0,6199999999999999
130 0,606666666666666667
131 0,5933333333333332
133 0,5666666666666667
134 0,5533333333333333
```

Tìm K tối ưu

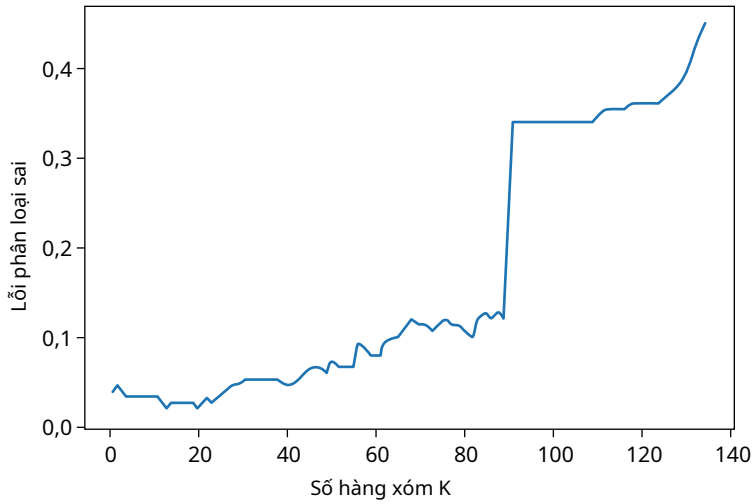
Để tìm k tối ưu, bạn chỉ cần tìm giá trị của k cho độ chính xác cao nhất. Hoặc, trong trường hợp này, bạn sẽ muốn tìm *lỗi phân loại sai (MSE)*.

Đoạn mã sau đây tìm MSE cho mỗi k, và sau đó tìm k có MSE thấp nhất. Sau đó, nó vẽ biểu đồ đường của MSE so với k (xem Hình 9.9):

```
# - - - tính toán lỗi phân loại sai cho mỗi k --- MSE = [1 - x cho x trong
cv_scores]

# - - - xác định k tốt nhất (tối thiểu MSE) --- tối
ưu_k = ks [MSE.index (min (MSE))]
print (f "Số láng giềng tối ưu là {tối ưu_k}")

# - - - lỗi phân loại sai âm mưu so với k --- plt.plot (ks,
MSE)
plt.xlabel ('Số hàng xóm K') plt.ylabel ('Lỗi phân
loại sai') plt.show ()
```

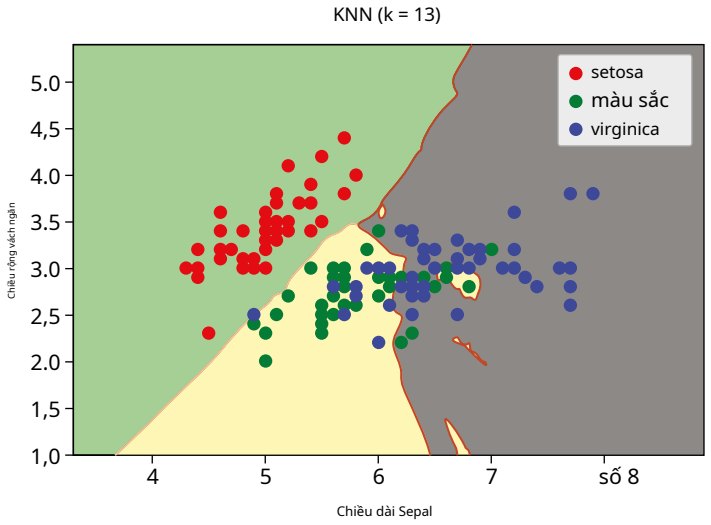
Đoạn mã trước in ra như sau:

```
Số láng giềng tối ưu là 13
```



Hình 9.9: Biểu đồ tính toán sai cho mỗi k

Hình 9.10 cho thấy sự phân loại khi k = 13.



Hình 9.10: Giá trị tối ưu của k tại 13

Bản tóm tắt

Trong số bốn thuật toán mà chúng ta đã thảo luận trong cuốn sách này, KNN được coi là một trong những thuật toán đơn giản nhất. Trong chương này, bạn đã học cách hoạt động của KNN và cách tính k tối ưu để giảm thiểu tính toán sai sót.

Trong chương tiếp theo, bạn sẽ học một loại thuật toán mới — học không giám sát. Bạn sẽ học cách khám phá các cấu trúc trong dữ liệu của mình bằng cách thực hiện phân nhóm bằng K-Means.