Mẫu đồ án báo cáo KNN về hoa Iris môn học máy

Machine Learning (Đại học Bách khoa Tphcm)

TRƯỜNG ĐẠI HỌC BÁCH KHOA TPHCM
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

TRIỂN KHAI GIẢI THUẬT K-NN VÀ ÁP DỤNG VÀO PHÂN LỚP TẬP DỮ LIỆU HOA IRIS

Môn học: Học Máy và Ứng Dụng

GVHD: PGS.TS Dương Tuấn Anh

Sinh viên thực hiện:

Huỳnh Gia Bảo	2070089
Nguyễn Thanh Quân	2070108

TPHCM, ngày 26 tháng 6 năm 2021

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

DANH MỤC TỪ VIẾT TẮT

K-NN	K-Nearest Neighbors

1. GIỚI THIỆU K-NEAREST NEIGHBORS (K-NN)

1.1. Thuật toán K-NN là gì

"Hãy cho tôi biết bạn của bạn là ai, tôi sẽ cho bạn biết bạn là người như thế nào." – đây là ý tưởng của K-NN.

K-NN là một trong những thuật toán supervised-learning đơn giản nhất (mà hiệu quả trong một vài trường hợp) trong Machine Learning. Thuật toán có mục đích phân loại lớp cho một mẫu mới (query point) dựa trên các thuộc tính và lớp của các mẫu sẵn có (training data), các mẫu này được nằm trong một hệ gọi là không gian mẫu. Khi training, thuật toán này không học một điều gì từ dữ liệu training (đây cũng là lý do thuật toán này được xếp vào loại lazy learning), mọi tính toán được thực hiện khi nó cần dự đoán kết quả của dữ liệu mới. K-NN có thể áp dụng được vào cả hai loại của bài toán Supervised learning là Classification và Regression. K-NN còn được gọi là một thuật toán Instance-based hay Memory-based learning.

Một đối tượng được phân lớp dựa vào k lân cận của nó với k là một số nguyên dương được xác định trước khi thực hiện thuật toán. Khoảng cách Euclidean thường được dùng để tính khoảng cách giữa các đối tượng trong K-NN.

1.2. Mô tả thuật toán K-NN

Các mẫu được mô tả bằng n – chiều thuộc tính số. Mỗi mẫu đại diện cho một điểm trong một không gian n – chiều. Theo cách này tất cả các mẫu được lưu trong một mô hình không gian n – chiều.

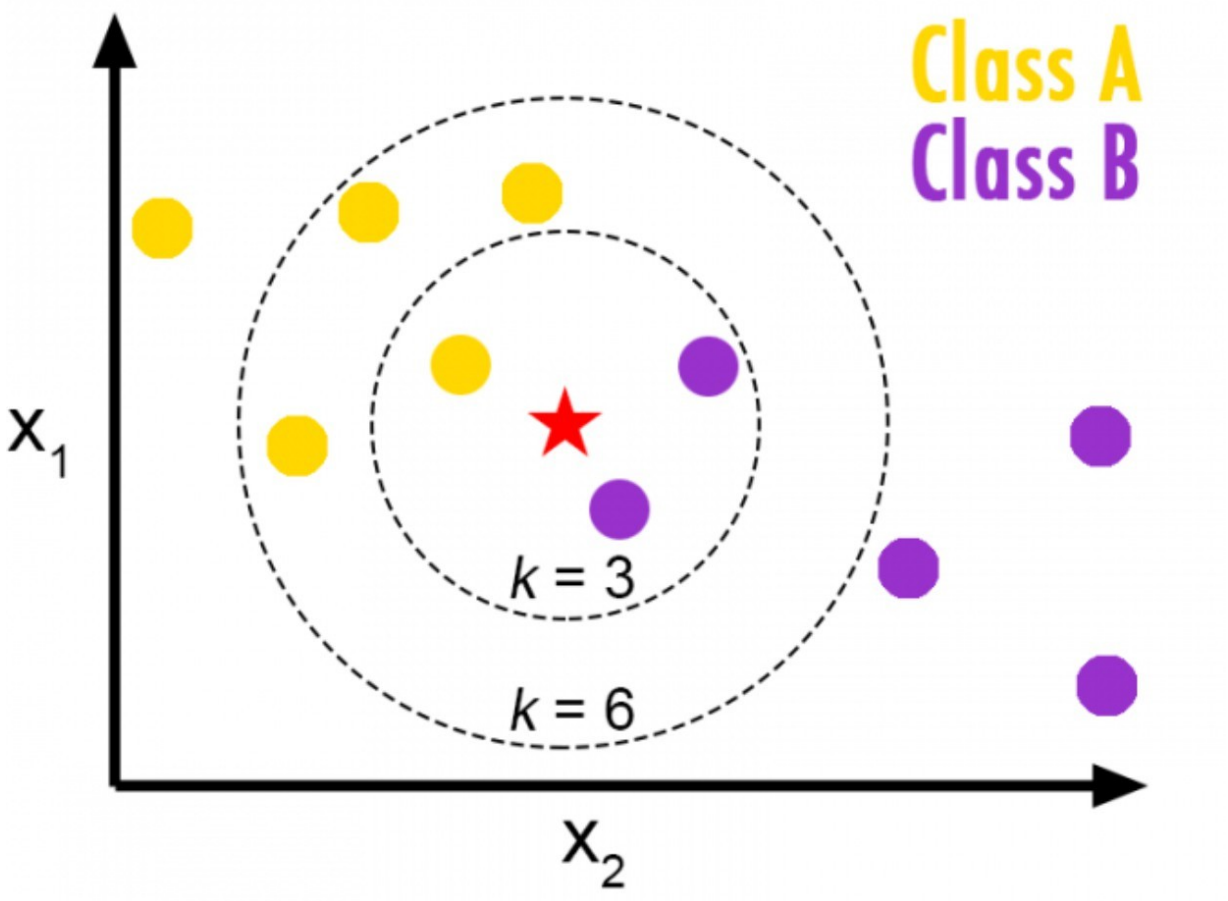
Các bước thực hiện của thuật toán K-NN được mô tả như sau:

- Xác định giá trị tham số k (số lân cận gần nhất)
- Tính khoảng cách giữa các đối tượng cần phần lớp (query point) với tất cả các đối tượng trong training data (thường sử dụng khoảng cách Euclidean)
- Sắp xếp khoảng cách theo thứ tự tăng dần và xác định k lân cận gần nhất với query point.
- Lấy tất cả các lớp của k lân cận gần nhất đã xác định.

- Dựa vào phần lớn lớp của lân cận gần nhất để xác định lớp cho mẫu mới (query point).

Để hiểu rõ K-NN được dùng để phân lớp như thế nào ta xem minh hoạ dưới đây:

Trong hình sau, training data được mô tả bởi hình tròn màu vàng và màu tím, đối tượng cần được xác định lớp cho nó (query point) là hình ngôi sao màu đỏ. Nhiệm vụ là dự đoán lớp của đối tượng dựa vào việc lựa chọn số lân cận gần nhất với nó. Nói cách khác chúng ta muốn biết liệu đối tượng được phân vào lớp hình tròn màu vàng hay màu tím.



Hình 1. Minh hoạ K-NN

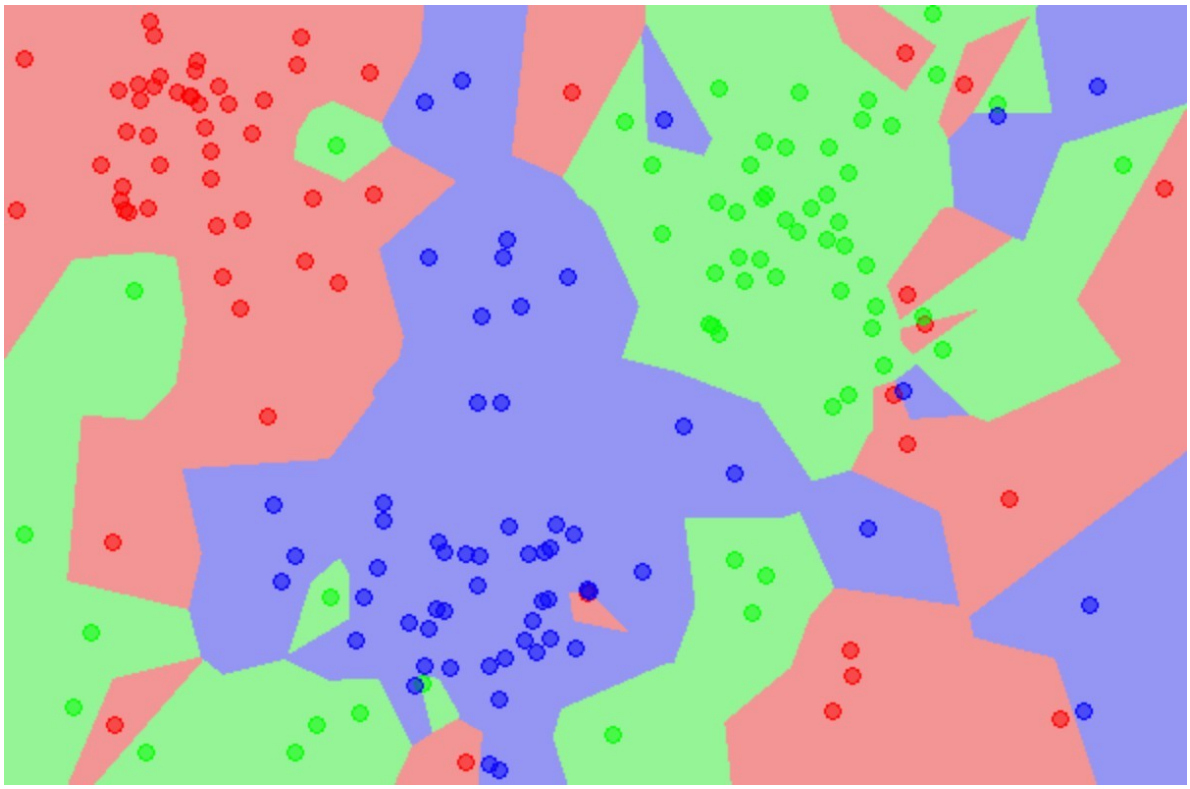
Dễ thấy rằng, với:

3 – lân cận gần nhất: kết quả là hình tròn màu tím.

4 – lân cận gần nhất (2 vàng, 2 tím): không xác định lớp cho đối tượng được vì số lân cận gần nhất với nó là 2 trong đó 2 là lớp hình tròn màu vàng và

2 là lớp hình tròn màu tím. Nói khác hơn là không có lớp nào có số đối tượng nhiều hơn lớp kia. Vì vậy số k thường được chọn sẽ là một số lẻ.

6 – lân cận gần nhất: kết quả là vàng vì có 4 đối tượng thuộc vàng và chỉ có 2 đối tượng thuộc tím.



Hình 2. K-NN với giá trị nhiễu

Hình trên là bài toán phân lớp với ba lớp: đỏ, lam lục. Mỗi điểm dữ liệu mới sẽ được gán nhãn theo màu của điểm đó mà nó thuộc về. Trong hình trên ta thấy rằng một số điểm thuộc về vùng khác màu, và đó là dữ liệu nhiễu dẫn đến việc dữ liệu kiểm tra nếu rơi vào vùng này sẽ có nhiều khả năng cho kết quả sai lệch và đây là K-NN với các giá trị nhiễu.

2. ĐỘ ĐO KHOẢNG CÁCH

2.1. Euclidean

Khoảng cách Euclidean của một điểm *P* trong không gian *n* chiều đến gốc tọa độ được tính bằng căn bậc hai của tổng bình phương các tọa độ thành phần:

$$d(O, P) = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

Đây cũng chính là độ dài của vector nối từ gốc tọa độ đến điểm *P*. Với hai điểm *P* và *Q*, ta có khoảng cách Euclidean giữa chúng như sau:

$$d(P, Q) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$$

Khoảng cách Euclidean đơn giản chỉ tính độ dài hình học giữa các điểm khác nhau trong không gian, coi mọi khoảng cách của các điểm có vai trò như nhau.

2.2. Manhattan

Khoảng cách Manhattan còn được gọi là khoảng cách L1, là 1 dạng khoảng cách giữa 2 điểm trong không gian Euclid và hệ tọa độ Descartes và được tính bằng tổng chiều dài của hình chiếu của đường thẳng nối 2 điểm này trong hệ trục tọa độ Descartes

Tọa độ 2 điểm $A(x_A, y_A)$ và $B(x_B, y_B)$ khi đó khoảng cách Manhattan giữa 2 điểm như sau:

$$d(A, B) = |x_A - x_B| + |y_A - y_B|$$

3. TÌM K LÂN CẬN GẦN NHẤT

Trong K-NN, việc xác định số k ban đầu phù hợp là điều không hề dễ dàng. Với giá trị nhỏ k, dữ liệu nhiễu có thể ảnh hưởng lớn đến kết quả trong khi với k giá trị lớn sẽ tốn chi phí cho việc tính toán. Các nhà khoa học dữ liệu thường chọn k là một số lẻ nếu số lớp là số chẵn và một cách đơn giản hơn là k bằng căn bậc hai của n, trong đó n là số mẫu dữ liệu.

4. TÍNH TOÁN ĐỘ CHÍNH XÁC

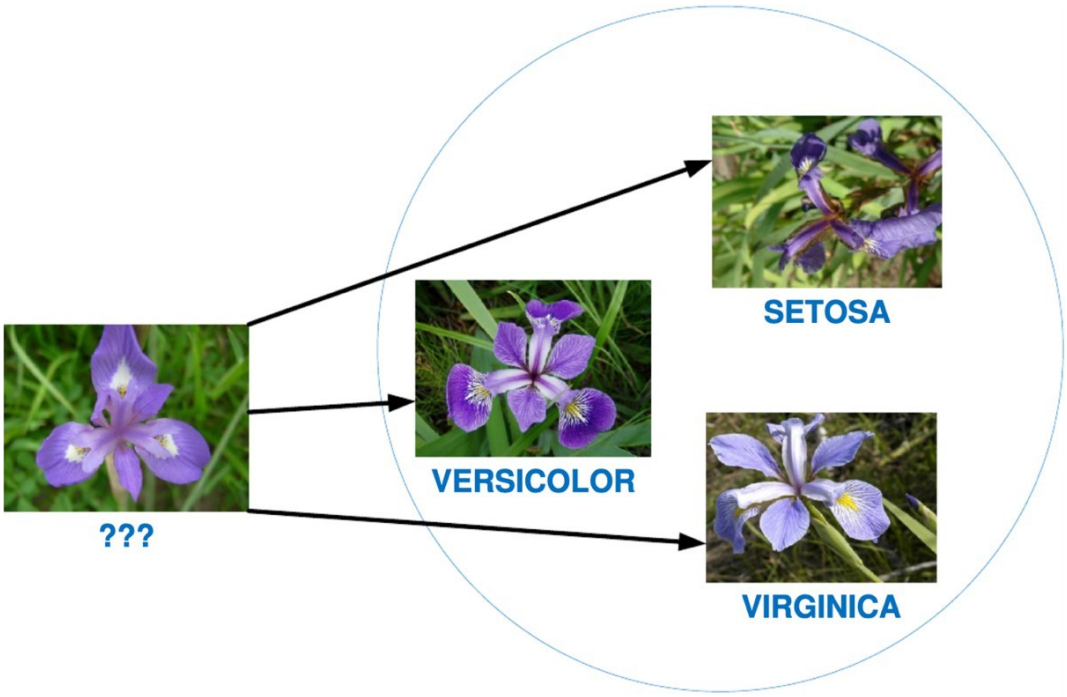
Để đánh giá độ chính xác của thuật toán KNN classifier này, chúng ta xem xem có bao nhiêu điểm trong test data được dự đoán đúng. Số lượng này chia cho tổng số lượng trong tập test data sẽ ra độ chính xác.

5. ÁP DỤNG K-NN VÀO PHÂN LỚP HOA IRIS

Iris setosa Iris versicolor Iris virginica

Hình 3. Ba loài hoa Iris (Nguồn: Wikipedia)

5.1. Định nghĩa bài toán



Hình 4. Bài toán phần lớp hoa Iris

Với một bộ dữ liệu về hoa Iris đã có về các cá thể hoa thuộc các loài Iris cụ thể, ta không cần phải phân tích về gen hay phân tích về sinh học phức tạp để nhận biết ra một cá thể hoa đang xét thuộc loài Iris nào, bằng các dữ liệu đã có sẵn và dựa vào các thuộc tính của cá thể, ta có thể đưa ra nhận biết nhanh chóng cá thể đó thuộc thoài hoa Iris nào bằng cách tính toán số học trên máy tính.

5.2. Thu thập và tiền xử lý dữ liệu

Bộ dữ liệu về hoa Iris được thu thập bởi Edgar Anderson – nhà thực vật học, sau đó được thống kê và rút gọn lai bởi Ronald Aylmer Fisher với các thuộc tính như sau:

- Bốn thuộc tính kiểu số:
  - Chiều dài đài hoa
  - Chiều rộng đài hoa
  - Chiều dài cánh hoa
  - Chiều rộng cánh hoa
- Một thuộc tính còn lại là tên của loài Iris
  - Iris Setosa
  - Iris Versicolor
  - Iris Virginica

Trong tập dữ liệu Iris, các giá trị nhỏ nhất, trung bình và lớ nhất ứng với các thuộc tính lần lượt như sau:

Bảng 1. Giá trị trong tập dữ liệu Iris

	Giá trị nhỏ nhất	**Giá trị lớn nhất**	**Giá trị trung bình**
Chiều dài đài hoa	4.3	7.9	5.84
Chiều rộng đài hoa	2.0	4.4	3.05
Chiều dài cánh hoa	1.0	6.9	3.76
Chiều rộng cánh hoa	0.1	2.5	1.20

Tỷ lệ phân chia cho mỗi loài trong ba loài Iris là 33.33%

5.2.1. Làm sạch dữ liệu (data cleaning)

Thiếu giá trị: khi xảy ra sự thiếu thông tin ở một thuộc tính nào đó trong một bộ dòng của bộ dữ liệu thu thập, khi mà tính đảm bảo số bộ dòng

chia đều

cho ba loài hoa trên đã có (33.33%) và số lượng bộ dòng thiếu là ít ta có thể áp dụng phương pháp loại bỏ. Trường hợp cần điền giá trị thiếu, ta có thể áp dụng các giá trị trung bình bên trên.

Dữ liệu nhiễu: khi xuất hiện một giá trị bất ngờ nào đó, đột nhiên vượt qua các giá trị biên đã được thống kê, ta có thể sửa lại giá trị đó thành các giá trị ở vùng biên.

5.2.2. Chọn lọc dữ liệu (data selection)

Tích hợp và dư thừa dữ liệu: do dữ liệu của Iris được thu thập từ một nguồn duy nhất nên việc tích hợp là không cần thiết trong trường hợp này, các thuộc tính của dữ liệu độc lập nhau, không có mối quan hệ tương quan nào, các thuộc tính đã được rút gọn chọn lọc nên không cần việc phân tích dư thừa dữ liệu đối với tập hoa Iris này.

5.2.3. Chuyển đổi dữ liệu (data transformation)

Với tập hoa Iris, chuẩn hoá dữ liệu về khoảng giá trị [0, 1] được sử dụng để đảm bảo việc cân bằng dữ liệu, không có thuộc tính mang giá trị lớn sẽ ảnh hưởng lớn đến đến các thuộc tính còn lại.

5.2.4. Chuẩn hoá dữ liệu (data normalization)

Khi có một thuộc tính trong dữ liệu (hay phần tử trong vector) lớn hơn các thuộc tính khác rất nhiều (nguyên nhân thường dẫn đến việc này là đơn vị đo các thuộc tính không giống nhau, ví dụ thay vì đo bằng cm thì một kết quả lại tính bằng mm), khoảng cách giữa các điểm sẽ phụ thuộc vào thuộc tính này rất nhiều. Để có được kết quả chính xác hơn, một kỹ thuật thường được dùng là Data Normalization (chuẩn hóa dữ liệu) để đưa các thuộc tính có đơn vị đo khác nhau về cùng một khoảng giá trị, thường là từ 0 đến 1, trước khi thực hiện KNN. Có nhiều kỹ thuật chuẩn hóa khác nhau và các kỹ thuật chuẩn hóa được áp dụng với không chỉ KNN mà còn với hầu hết các thuật toán khác. Ta cùng theo dõi bảng giá trị bên dưới để thấy rõ điều này.

Trích từ bộ dữ liệu hoa Iris, giá trị các thuộc tính trước và sau khi chuẩn hoá dữ liệu

Bảng 2. Dữ liệu minh hoạ trước khi chuẩn hoá

Chiều dài đài hoa	**Chiều rộng đài hoa**	**Chiều dài cánh hoa**	**Chiều rộng cánh hoa**	**Tên loài**
5.1	3.5	1.4	0.2	Setosa
4.7	3.2	1.3	0.2	Setosa
7.0	3.2	4.7	1.4	Versicolor
6.3	3.3	6.0	2.5	Virginica
5.9	3.0	4.2	1.5	Versicolor
5.1	3.8	1.6	0.2	??? (Điểm dữ liệu cần phân loại)

Bảng 3. Khoảng cách minh hoạ trước khi chuẩn hoá

Tên loài	**Khoảng cách**
Setosa	$\sqrt{(5.1-5.1)^2+(3.8-3.5)^2+(1.6-1.4)^2+(0.2-0.2)^2} = 0.361$
Setosa	$\sqrt{(5.1-4.7)^2+(3.8-3.2)^2+(1.6-1.3)^2+(0.2-0.2)^2} = 0.781$
Versicolor	$\sqrt{(5.1-7.0)^2+(3.8-3.2)^2+(1.6-4.7)^2+(0.2-1.4)^2} = 3.876$
Virginica	$\sqrt{(5.1-6.3)^2+(3.8-3.3)^2+(1.6-6.0)^2+(0.2-2.5)^2} = 5.132$
Versicolor	$\sqrt{(5.1-5.9)^2+(3.8-3.0)^2+(1.6-4.2)^2+(0.2-1.5)^2} = 3.120$

Với dữ liệu chưa được chuẩn hoá, dễ thấy khoảng cách sẽ phụ thuộc vào các điểm mang giá trị số lớn rất nhiều, điều này làm cho kết quả phân lớp có thể không chính xác.

Bảng 4. Dữ liệu minh hoạ sau khi chuẩn hoá

Chiều dài đài hoa	**Chiều rộng đài hoa**	**Chiều dài cánh hoa**	**Chiều rộng cánh hoa**	**Tên loài**
5.1/7.0 = 0.73	3.5/3.8 = 0.92	1.4/6.0 = 0.23	0.2/2.5 = 0.08	Setosa
4.7/7.0 = 0 .67	3.2/3.8 = 0.84	1.3/6.0 = 0.22	0.2/2.5 = 0.08	Setosa
7.0/7.0 = 1	3.2/3.8 = 0.84	4.7/6.0 = 0.78	1.4/2.5 = 0.56	Versicolor
6.3/7.0 = 0.9	3.3/3.8 = 0. 87	6.0/6.0 = 1	2.5/2.5 = 1	Virginica
5.9/7.0 = 0.84	3.0/3.8 = 0.79	4.2/6.0 = 0.7	1.5/2.5 = 0.6	Versicolor
5.1/7.0 = 0.73	3.8/3.8 = 1	1.6/6.0 = 0.27	0.2/2.5 = 0.08	???

Bảng 5. Khoảng cách minh hoạ sau khi chuẩn hoá

Tên loài	**Khoảng cách**
Setosa	$!\overline{(0.73-0.73)^2+(0.92-1)^2+(0.23-0.27)^2+(0.08-0.08)^2} = 0.008$
Setosa	$!\overline{(0.67-0.73)^2+(0.92-1)^2+(0.23-0.27)^2+(0.08-0.08)^2} = 0.178$
Versicolor	$!\overline{(1-0.73)^2+(0.84-1)^2+(0.78-0.27)^2+(0.56-0.08)^2} = 0.767$
Virginica	$!\overline{(0.9-0.73)^2+(0.87-1)^2+(1-0.27)^2+(1-0.08)^2} = 1.194$
Versicolor	$!\overline{(0.84-0.73)^2+(0.79-1)^2+(0.7-0.27)^2+(0.6-0.08)^2} = 0.715$

aChọn k = 3 thì khoảng cách nhỏ nhất lần lượt là 0.008, 0.178 và 0.715 tương ứng với tên loài là Setosa, Setosa và Versicolor. Trong đó có 2 lân cận gần nhất là Setosa và 1 thuộc về Versicolor nên ta kết luận loài cần phân lớp thuộc Setosa.

Chọn k = 4 tương tự như trên lần lượt ta có 2 lân cận gần nhất là Setosa và 2 lân cận gần nhất thuộc về Versicolor nên ta không thể kết luận loài cần phân lớp sẽ thuộc về lớp nào.

Với dữ liệu sau khi được chuẩn hoá, việc tính toán khoảng cách sẽ cho kết quả chính xác hơn khi không có sự phụ thuộc vào đơn vị hay giá trị thuộc tính

5.3 Triển khai phân lớp hoa Iris

Iris flower dataset là một bộ dữ liệu nhỏ. Bộ dữ liệu này bao gồm thông tin của ba loại hoa Iris (một loài hoa lan) khác nhau: Iris setosa, Iris virginica và Iris versicolor. Mỗi loại có 50 bông hoa được đo với dữ liệu là 4 thông tin: chiều dài (length), chiều rộng (width) đài hoa (sepal), và chiều dài, chiều rộng cánh hoa (petal).

```
[INFO] Dataset shape: (150, 6)
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

Hình 5. Giá trị mẫu 3 dòng dữ liệu Iris

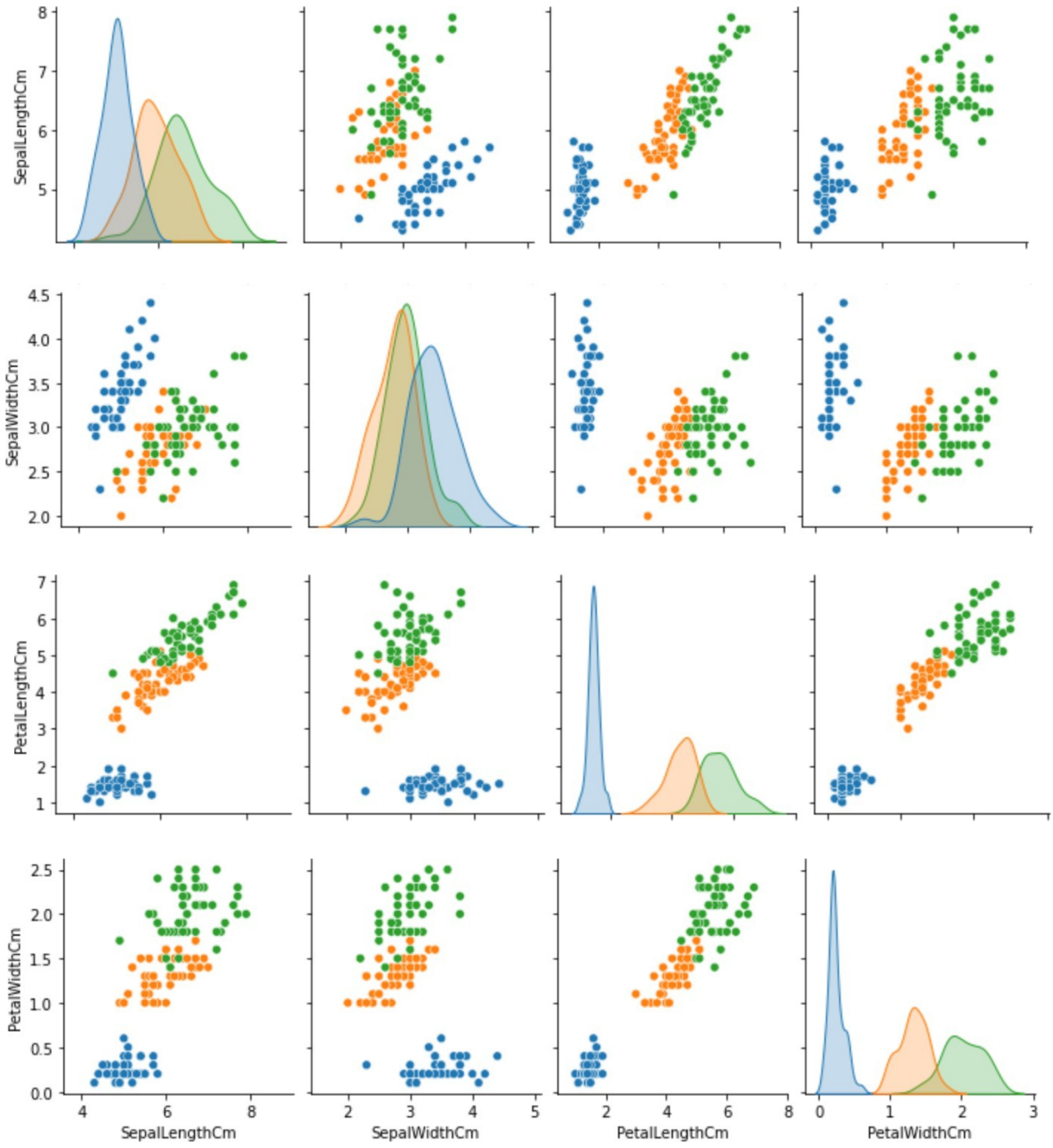
Dataset không có các dữ liệu missing hay null. Có 150 mẫu dữ liệu trong bộ dataset. Phân bố dữ liệu trong dataset là đều không có hiện tượng mất cân bằng dữ liệu.

```
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

Giá trị của các đặc trưng khá chênh lệch. Ví dụ như 'SepalLength' có giá trị max = 150, trong khi đó PetalWidth có giá trị max = 2.5. Nên chuẩn hoá (normalize) dữ liệu về tầm giá trị [0, 1]. Nhóm sẽ so sánh kết quả độ sự ảnh hưởng độ chính xác khi không chuẩn hoá và sau chuẩn hoá dữ liệu đầu vào.

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

Bảng 6. Giá trị đặc trưng chênh lệch



Hình 6. Sự tương quan phân bố dữ liệu theo thuộc tính

<u>Tiền xử lí dữ liệu</u>

Loài hoa (nhãn) là một dữ liệu rời rạc vì thế cần chuyển đổi về dạng số để có thể làm việc được với mô hình phân lớp KNN triển khai

- Setosa tương ứng với 0
- Versicolor tương ứng với 1
- Virginica tương ứng với 2

5.2.5. Cách thức đo độ chính xác của mô hình

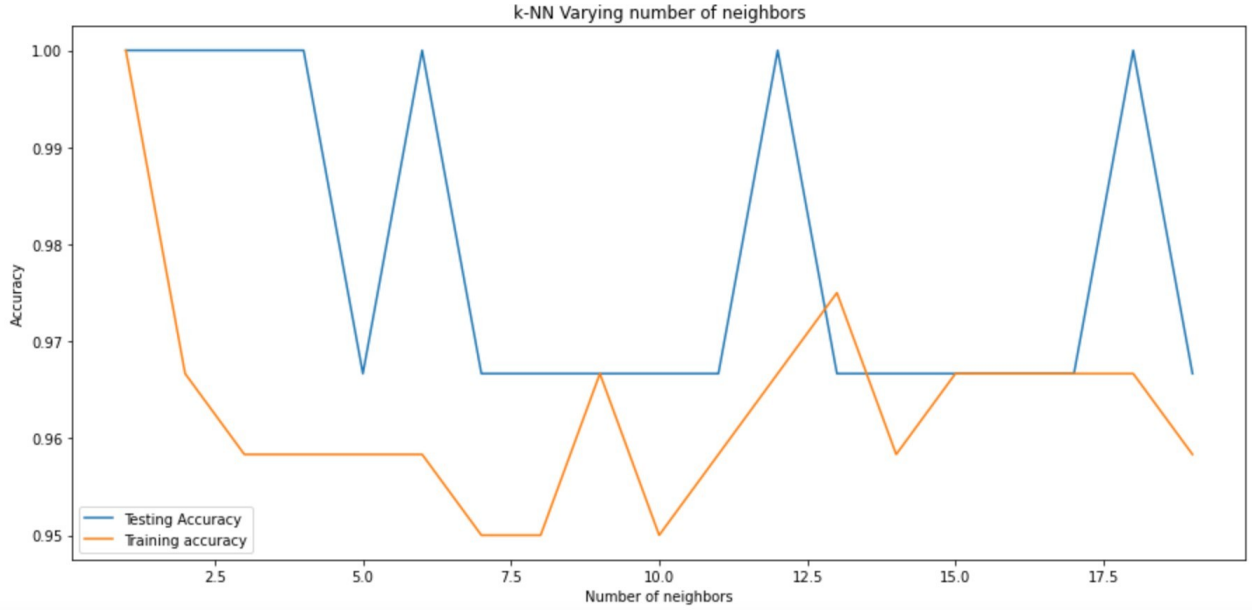
Bước 1: Phân dữ liệu ngẫu nhiên theo hai tập huấn luyện và kiểm tra

```
Training size: 120
Test size   : 30
```

Bước 2: Đánh giá mô hình qua tập kiểm tra

```
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 2 1 2 1 1 0 0 2 0 2]

Accuracy:  96.66666666666667 %
```

5.3. Phương pháp chọn số k thích hợp



Hình 7. Chọn số k thích hợp

6. K-NN VỚI TRỌNG SỐ

Kết quả bên trên vẫn có thể được tiếp tục cải thiện. Trong kỹ thuật major voting, mỗi trong 10 điểm gần nhất được coi là có vai trò như nhau và giá trị *lá phiếu* của mỗi điểm này là như nhau. Như thế là không công bằng, vì rõ ràng rằng những điểm gần hơn nên có trọng số cao hơn (*càng lân cận thì càng chính xác hơn*). Vậy nên chúng ta sẽ đánh trọng số khác nhau cho mỗi trong 10 điểm gần nhất này. Cách đánh trọng số phải thoải mãn điều kiện là một điểm càng gần điểm test data thì phải được đánh trọng số càng cao (tin tưởng hơn). Cách đơn giản nhất là lấy nghịch đảo của khoảng cách này. (Trong trường hợp test data trùng với 1 điểm dữ liệu trong training data, tức khoảng cách bằng 0, chúng ta lấy luôn label của điểm training data).

Ngoài ra, trong phần triển khai KNN của tài liệu này, một cách đánh trọng số khác còn được sử dụng với công thức sau

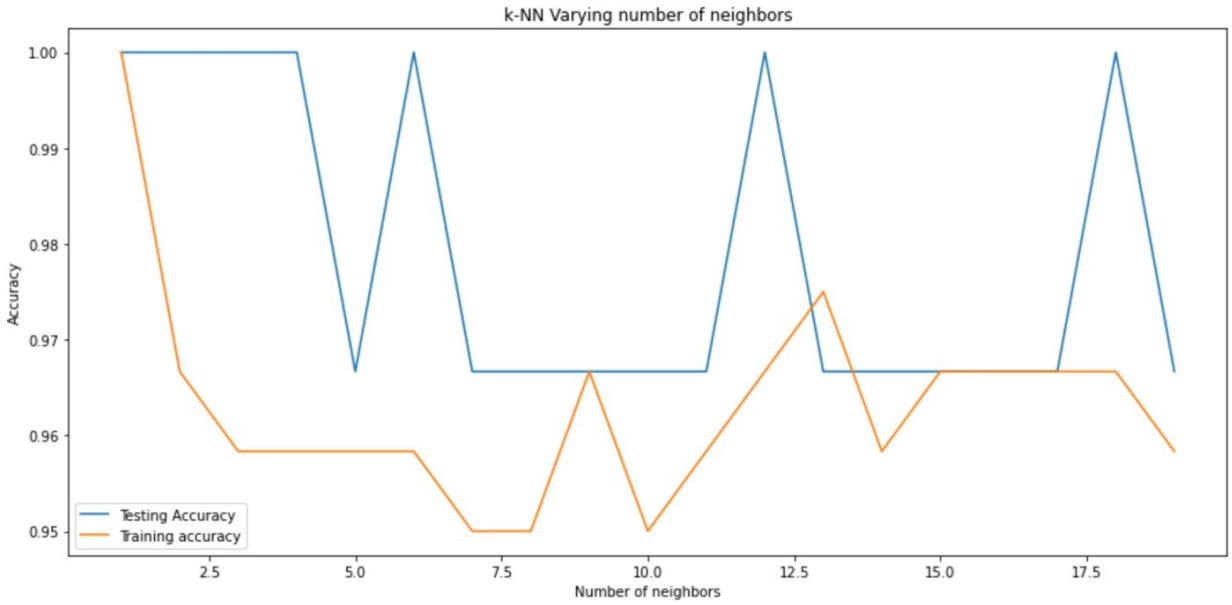
$$w_i = \exp\left(\frac{-||\mathbf{x} - \mathbf{x}_i||_2^2}{\sigma^2}\right)$$

trong đó x là test data, xi là một điểm trong K-lân cận của x, wi là trọng số của điểm đó (ứng với điểm dữ liệu đang xét x), σ là một số dương. Nhận thấy rằng hàm số này cũng thỏa mãn điều kiện: điểm càng gần x thì trọng số càng cao (cao nhất bằng 1).

7. SO SÁNH KẾT QUẢ VỚI THƯ VIỆN SKLEARN

Với k = 7, độ chính xác khi sử dụng thư viện sklearn đúng bằng với giải thuật triển khai K-NN bên trên

```
Labels:
[2 2 0 2 0 0 1 1 2 0 0 2 2 2 0 0 2 1 1 1 1 1 0 0 0 2 2 2 2 1]
Predictions:
[2 1 0 2 0 0 1 1 2 0 0 2 2 2 0 0 2 1 1 1 1 1 0 0 0 2 2 2 2 1]
Accuracy is: 96.66666666666667%
```



Hình 8. Tìm số k thích hợp theo thư viện sklearn

8. KẾT LUẬN

Ưu điểm của thuật toán K-NN

- Độ phức tạp tính toán của quá trình training là bằng 0.
- Dễ triển khai và sử dụng.
- Việc dự đoán kết quả cả dữ liệu mới dễ dàng.
- Không cần giả sử gì về phân phối của các

class. Nhược điểm của thuật toán K-NN

- KNN nhiễu dễ đưa ra kết quả không chính xác khi k được chọn nhỏ.
- KNN cần phải nhớ tất cả các điểm dữ liệu training dẫn đến cần không gian lưu trữ tập huấn luyện khi dữ liệu huấn luyện lớn.
- KNN sẽ tốn chi phí tính toán khi số lượng kiểm thử tăng lên nhiều.

9. MÃ NGUỒN

Giải thuật KNN được nhóm triển khai trên môi trường Google Colab.

Link mã nguồn: Tham khảo tại đây

9.1. Cài đặt các thư viện cần thiết

Một số thư viện cần thiết được dùng trong bài gồm:

- Thư viện pandas dùng để load dữ liệu dưới dạng bảng (csv).
- Thư viện numpy dùng để hỗ trợ tính toán ma trận.
- Thư viện sklearn dùng để kiểm tra so sánh kết quả với giải thuật hiện thực.
- Thư viện matplotlib và seaborn cho trực quan dữ liệu.

```
import pandas as pd
import numpy as np
import operator
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.utils import shuffle
import seaborn as sns
```

9.2. Xây dựng lớp mô hình K-NN

Nhóm đã xây dựng một lớp `KnearestNeighborsClassifier` với các tham số khởi tạo là số lượng điểm lân cận k (`n_neighbors`) và độ đo khoảng cách (`metric`) sẽ sử dụng (`'euclidean'`, `'manhattan'`), phương pháp đánh trọng số (`weights`) sẽ sử dụng (`'uniform'`, `'distance'`, `custum_weight_function`).

Việc xây dựng lớp mô hình sẽ giúp việc khởi tạo và tái sử dụng mô hình dễ dàng hơn.

- Hàm `fit(self, X, y)`: Giúp đưa dữ liệu huấn luyện vào mô hình. Với X là mảng các giá trị đặc trưng của điểm dữ liệu với mỗi hàng là một điểm dữ liệu. Với y là vector nhãn dữ liệu. Dữ liệu sẽ được chuyển đổi thành từng cặp giá trị đặc trưng và nhãn dữ liệu.

- Hàm `_calculate_distances(self, X)`: Tính toán khoảng cách của các điểm dữ liệu cần dự đoán với các điểm dữ liệu trong tập huấn luyện.
- Hàm `_get_k_neighbors(self, X)`: Trả về khoảng cách và chỉ số thứ tự trong tập dữ liệu huấn luyện của k điểm dữ liệu gần nhất với dữ liệu cần dự đoán.
- Hàm `_get_weights(self, dist)`: Tính trọng số của khoảng cách.
- Hàm `_weighted_mode(self, a, w, *, axis=0)`: Giúp bầu chọn nhãn cho dữ liệu cần dự đoán dựa theo phương pháp đánh trọng số lựa chọn.
- Hàm `score(self, X, y)`: Trả về trực tiếp giá trị độ chính xác khi đưa dữ liệu cần kiểm tra.

```
class  KNearestNeighborsClassifier():
    def __init__(self, n_neighbors=5, metric='euclidean',
weights='uniform'):
        self.n_neighbors = n_neighbors
        self.metric = metric
        self.weights = weights

    def fit(self, X, y):
        self.X_fit_ = np.asarray(X)
        self.y_fit_ = np.asarray(y)

    def _calculate_distances(self, X):
        if self.metric == 'euclidean':
            X = np.expand_dims(X, 1)
            distances = np.linalg.norm(X - self.X_fit_, axis=-1)
        elif self.metric == 'manhattan':
            distances = np.sum(np.abs(X - self.X_fit_), axis=-1)
        return distances

    def _get_k_neighbors(self, X):
        distances = self._calculate_distances(X)
        neigh_ind = np.argsort(distances)[:,:self.n_neighbors]
        neigh_dist = np.asarray([np.take(distances[i], neigh_ind[i])
for i in range(neigh_ind.shape[0])])
        return neigh_dist, neigh_ind

    def _get_weights(self, dist):
        if self.weights in (None, 'uniform'):
            return None
```

```
        elif self.weights == 'distance':
            with np.errstate(divide='ignore'):
                dist = 1. / dist
            inf_mask = np.isinf(dist)
            inf_row = np.any(inf_mask, axis=1)
            dist[inf_row] = inf_mask[inf_row]
            return dist
        elif callable(self.weights):
            return weights(dist)

    def _weighted_mode(self, a, w, *, axis=0):
        if axis is None:
            a = np.ravel(a)
            w = np.ravel(w)
            axis = 0
        else:
            a = np.asarray(a)
            w = np.asarray(w)

        if a.shape != w.shape:
            w = np.full(a.shape, w, dtype=w.dtype)

        scores = np.unique(np.ravel(a))       # get ALL unique values
        testshape = list(a.shape)
        testshape[axis] = 1
        oldmostfreq = np.zeros(testshape)
        oldcounts = np.zeros(testshape)
        for score in scores:
            template = np.zeros(a.shape)
            ind = (a == score)
            template[ind] = w[ind]
            counts = np.expand_dims(np.sum(template, axis), axis)
            mostfrequent = np.where(counts > oldcounts, score,
oldmostfreq)
            oldcounts = np.maximum(counts, oldcounts)
            oldmostfreq = mostfrequent
        return mostfrequent, oldcounts

    def predict(self, X):
        X_ = np.asarray(X)
        neigh_dist, neigh_ind = self._get_k_neighbors(X_)
        weights = self._get_weights(neigh_dist)

        if weights is None:
            mode, _ = stats.mode(self.y_fit_[neigh_ind], axis=1)
        else:
            mode, _ = self._weighted_mode(self.y_fit_[neigh_ind],
weights, axis=1)

        mode = np.asarray(mode.ravel(), dtype=np.intp)
```

```
        return mode

    def score(self, X, y):
        predictions = self.predict(X)
        correct = 0
        for i in range(len(y)):
            if y[i] == predictions[i]:
                correct = correct + 1
        return correct / float(len(y))
```

- Hàm hỗ trợ:
  - `get_accuracy` :hỗ trợ tính toán độ chính xác của mô hình bằng cách tính tỉ lệ dự đoán đúng của mô hình so với nhãn biết trước
  - `draw_varying_neighbor` : Giúp tính toán độ chính xác của tập dữ liệu huấn luyện và tập kiểm tra với sự thay đổi của giá trị k lân cận và vẽ biểu đồ sự thay đổi.

```
def get_accuracy(labels, predictions):
    correct = 0
    for i in range(len(labels)):
        if labels[i] == predictions[i]:
            correct = correct + 1
    return (correct / float(len(labels))) * 100.0

def draw_varying_neighbor(X_train, y_train, X_test, y_test,
model_class=KNearestNeighborsClassifier, weights='uniform'):
    #Setup arrays to store training and test accuracies
    neighbors = np.arange(1,20)
    train_accuracy =np.empty(len(neighbors))
    test_accuracy = np.empty(len(neighbors))

    for i,k in enumerate(neighbors):
        #Setup a knn classifier with k neighbors
        knn = model_class(n_neighbors=k, weights=weights)

        #Fit the model
        knn.fit(X_train, y_train)

        #Compute accuracy on the training set
        train_accuracy[i] = knn.score(X_train, y_train)

        #Compute accuracy on the test set
        test_accuracy[i] = knn.score(X_test, y_test)
```

```
    #Generate plot
    plt.figure(figsize=(15,7))
    plt.title('k-NN Varying number of neighbors')
    plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
    plt.plot(neighbors, train_accuracy, label='Training accuracy')
    plt.legend()
    plt.xlabel('Number of
    neighbors')
    plt.ylabel('Accuracy')
    plt.show()
```

9.3. Áp dụng mô hình K-NN vào bộ dữ liệu Iris

9.3.1. Tải bộ dữ liệu

```
dataset_filename = "/content/drive/MyDrive/MasterBK_2020/Machine
Learning/data/Iris.csv"
dataset = pd.read_csv(dataset_filename)
print(f"[INFO] Dataset shape: {dataset.shape}")
dataset.head()
```

```
dataset.info()
```

```
dataset['Species'].value_counts().plot(kind='bar')
```

```
dataset.describe()
```

```
sns.pairplot(dataset.drop(columns='Id'), hue = 'Species',
kind='scatter')
plt.show()
```

9.3.2. Tiền xử lý dữ liệu

```
X = dataset.drop(['Id', 'Species'], axis=1)
y = dataset['Species']
```

- Mã hoá dữ liệu

Nhãn dữ liệu thuộc loại kiểu dữ liệu phân loại. Do đó cần phải chuyển đổi nhãn từ kiểu kí tự sang số nguyên trước khi đưa vào mô hình.

Iris-setosa tương ứng 0

Iris-versicolor tương 1

Iris-virginica tương ứng 2

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

- Chia tập dữ liệu thành tập train và test tỉ lệ test_size = 20%

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.20, random_state=2)
```

- Chuẩn hoá dữ liệu

```
X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.fit_transform(X_test)
```

9.3.3. Xây dựng mô hình KNN cho tập dữ liệu Iris

- Không có chuẩn hoá dữ liệu

```
knn_model = KNearestNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
predictions = knn_model.predict(X_test)
print(y_test)
print(predictions)
print("Accuracy: ", get_accuracy(y_test, predictions), "%")
```

```
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]

[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]

Accuracy:  100.0 %
```

- Có chuẩn hoá dữ liệu

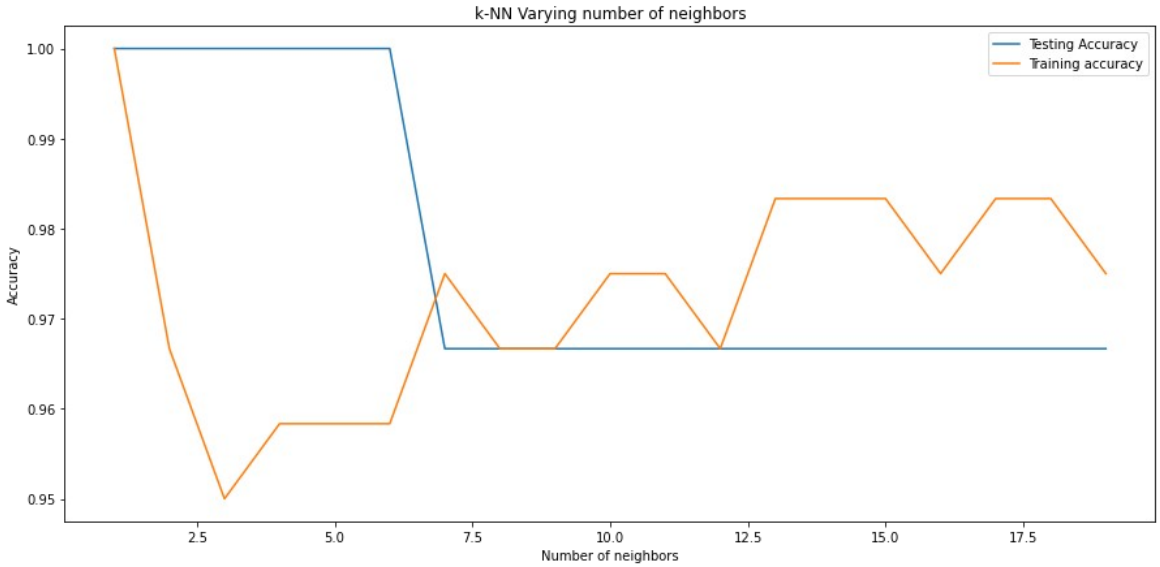
```
knn_model.fit(X_train_norm, y_train)
predictions = knn_model.predict(X_test_norm)
print(y_test)
print(predictions)
print("Accuracy: ", get_accuracy(y_test, predictions), "%")
```

```
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]

[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 2 1 2 1 1 0 0 2 0 2]

Accuracy:  96.66666666666667 %
```

9.3.4. Thay đổi giá trị k tìm giá trị tốt nhất

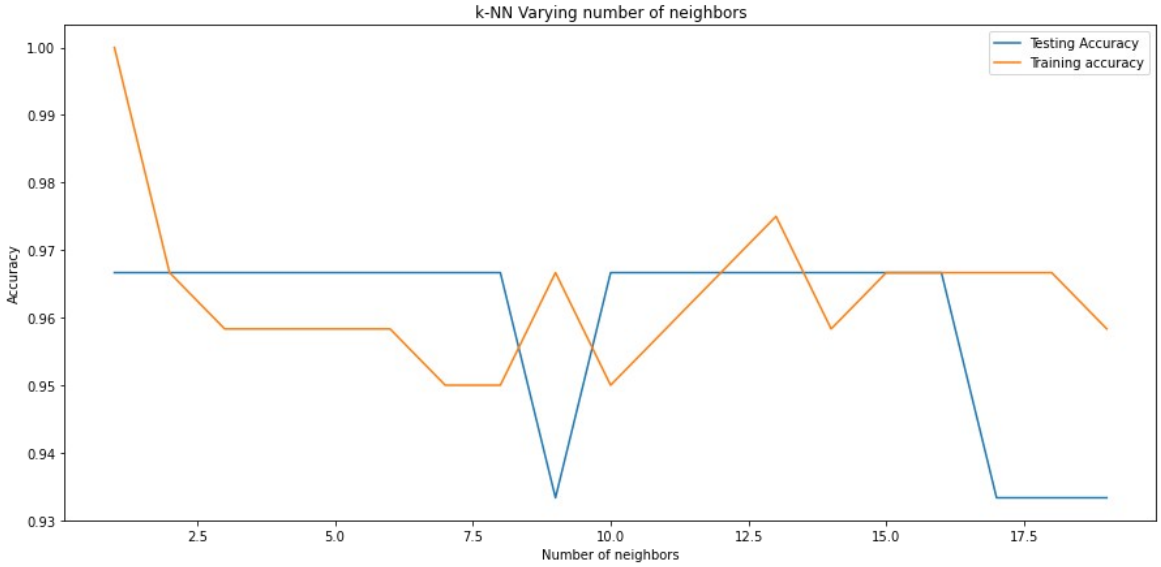
- Không có chuẩn hoá dữ liệu

```
draw_varying_neighbor(X_train, y_train, X_test, y_test)
```



- Có chuẩn hoá dữ liệu

```
draw_varying_neighbor(X_train_norm, y_train, X_test_norm, y_test)
```



- Chọn k tốt nhất k=1

```
# Tạo model (k = 1)
knn_best = KNearestNeighborsClassifier(n_neighbors=1)

# Fitting the model
knn_best.fit(X_train_norm, y_train)
predictions = knn_best.predict(X_test)
print(y_test)
print(predictions)
print("Accuracy: ", get_accuracy(y_test, predictions), "%")
```

```
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]
```

```
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 2 1 2 1 1 0 0 2 0 2]

Accuracy:  96.66666666666667 %
```

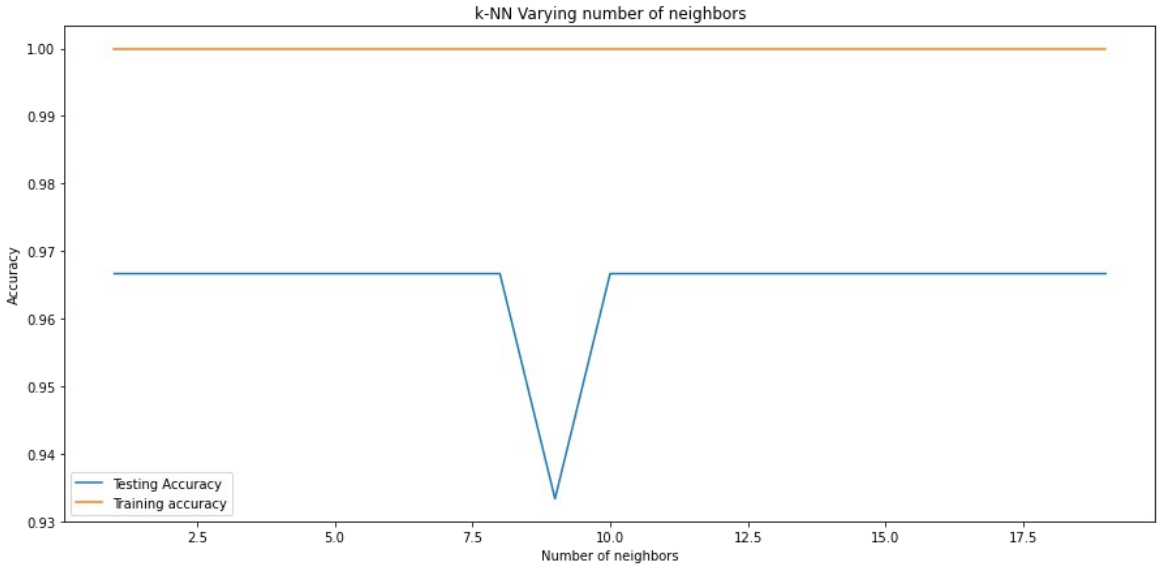
9.3.5. Dùng mô hình K-NN có đánh trọng số

- Đánh trọng số "distance" cho các điểm lân cận

```
knn_model = KNearestNeighborsClassifier(n_neighbors=3,
weights='distance')
knn_model.fit(X_train_norm, y_train)
predictions = knn_model.predict(X_test_norm)
print(y_test)
print(predictions)
print("Accuracy with weight sigma: ", get_accuracy(y_test,
predictions), "%")
```

```
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 2 1 2 1 1 0 0 2 0 2]
Accuracy with weight sigma:  96.66666666666667 %
```

```
draw_varying_neighbor(X_train_norm, y_train, X_test_norm, y_test,
weights='distance')
```



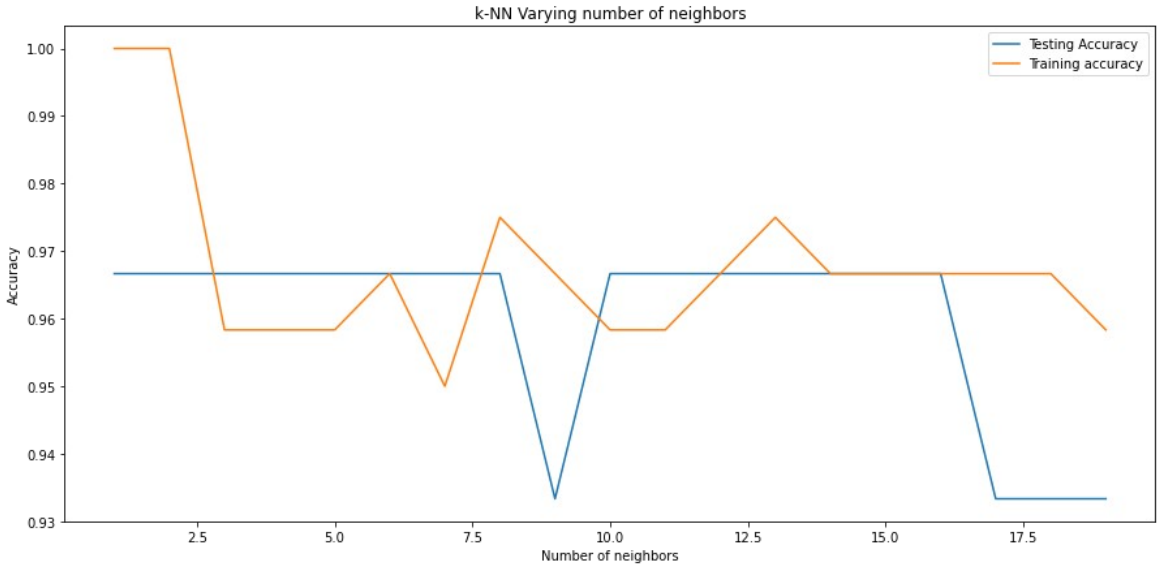
- Đánh trọng số "sigma" cho các điểm lân cận

```
def myweight(distances):
    sigma2 = .5 # we can change this number
    return np.exp(-distances**2/sigma2)

knn_model = KNearestNeighborsClassifier(n_neighbors=3,
weights=myweight)
```

```
knn_model.fit(X_train_norm, y_train)
predictions = knn_model.predict(X_test_norm)
print(y_test)
print(predictions)
print("Accuracy with weight sigma: ", get_accuracy(y_test,
predictions), "%")
```

```
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 2 1 2 1 1 0 0 2 0 2]
Accuracy with weight sigma:  96.66666666666667 %
```

```
draw_varying_neighbor(X_train_norm, y_train, X_test_norm, y_test,
weights=myweight)
```



9.4. So sánh với kết quả của thư viện sklearn

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, accuracy_score,
plot_confusion_matrix
```

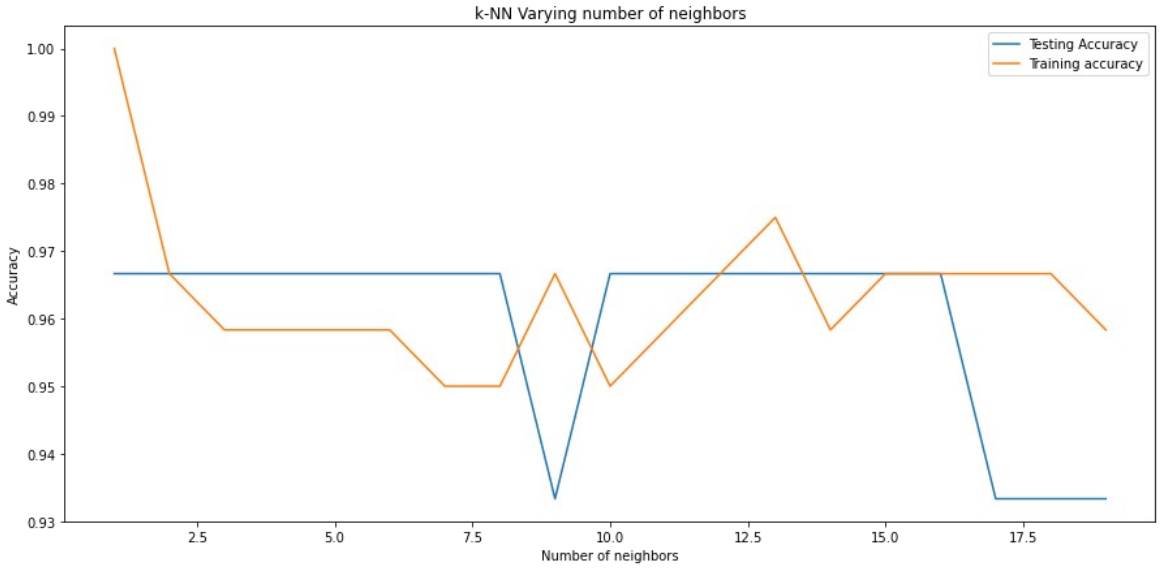
9.4.1. Dự đoán kết quả tập test dùng mô hình Knn

```
knn_model_sk = KNeighborsClassifier(n_neighbors=14)
knn_model_sk.fit(X_train_norm, y_train)
predictions_sk = knn_model_sk.predict(X_test)
print(y_test)
print(predictions_sk)
print("Accuracy: ", get_accuracy(y_test, predictions_sk), "%")
```

```
[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]

[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 2 1 2 1 1 0 0 2 0 2]

Accuracy:  96.66666666666667 %
```

9.4.2. Thay đổi k tìm giá trị tốt nhất

```
draw_varying_neighbor(X_train_norm, y_train, X_test_norm, y_test,
model_class=KNeighborsClassifier)
```



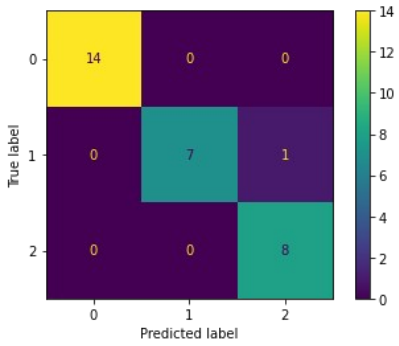
- Chọn k tốt nhất k = 1

```
# Tạo model (k = 1)
classifier = KNeighborsClassifier(n_neighbors=1)

# Fitting the model
classifier.fit(X_train_norm, y_train)

# Predicting on the test set
y_pred = classifier.predict(X_test)
print(f"Labels:      {y_test}")
print(f"Predictions: {y_pred}")
print(f"Accuracy is: {accuracy_score(y_test, y_pred)*100}%")
```

```
Labels:      [0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 1 1 2 1 1 0 0 2 0 2]
Predictions:[0 0 2 0 0 2 0 2 2 0 0 0 0 0 1 1 0 1 2 1 2 1 2 1 1 0 0 2 0 2]
Accuracy is: 96.66666666666667%
```

Confusion matrix

```
cm = plot_confusion_matrix(classifier, X_test_norm, y_test)
plt.show()
```



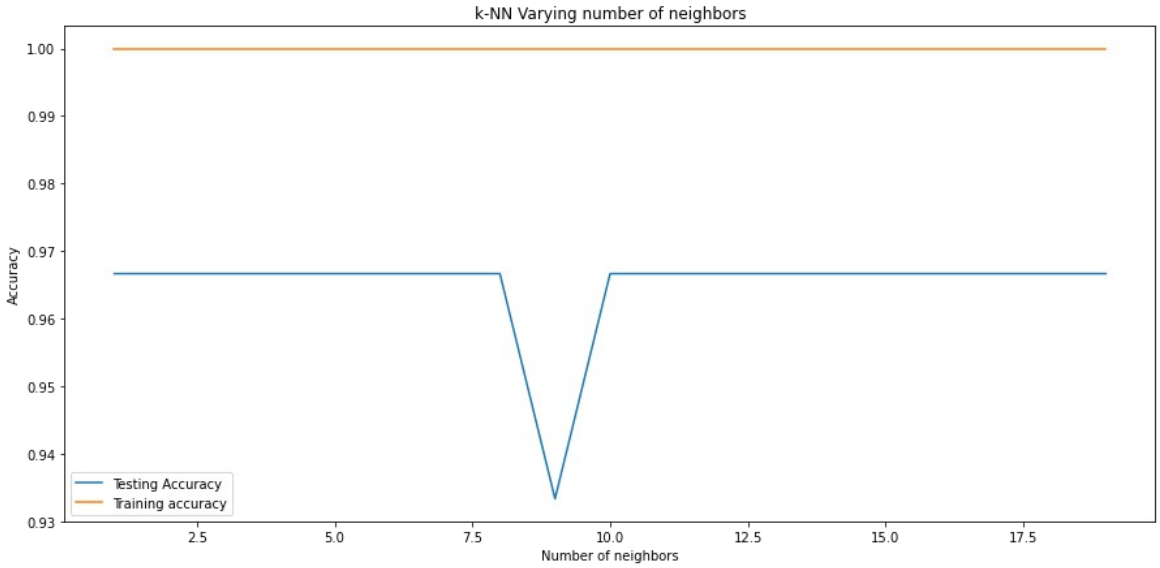
9.4.3. Đánh trọng số cho các điểm lân cận

- Đánh trọng số "distance" cho các điểm lân cận

```
knn_model_weight = KNeighborsClassifier(n_neighbors = 7, p = 2,
weights 'distance')
knn_model_weight.fit(X_train_norm, y_train)
y_pred = knn_model_weight.predict(X_test)

print("Accuracy of 10NN (1/distance weights): %.2f %%"
%(100*accuracy_score(y_test, y_pred)))
```

```
Accuracy of 10NN (1/distance weights): 96.67 %
```

```
draw_varying_neighbor(X_train_norm, y_train, X_test_norm, y_test,
model_class=KNeighborsClassifier, weights = 'distance')
```

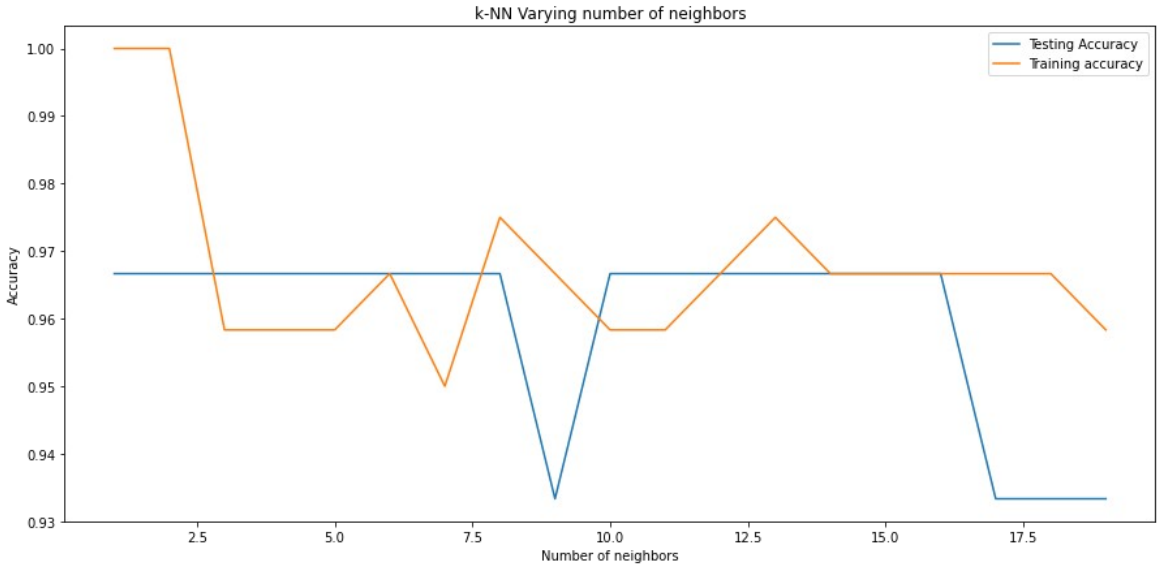


- Đánh trọng số tự chọn

```
def myweight(distances):
    sigma2 = .5 # we can change this number
    return np.exp(-distances**2/sigma2)

knn_model_custom_weight = KNeighborsClassifier(n_neighbors = 5, p =
2, weights = myweight)
knn_model_custom_weight.fit(X_train_norm, y_train)
y_pred = knn_model_custom_weight.predict(X_test_norm)

print("Accuracy of 10NN (customized weights): %.2f %%"
%(100*accuracy_score(y_test, y_pred)))
```

```
Accuracy of 10NN (customized weights): 96.67 %
```

```
draw_varying_neighbor(X_train_norm, y_train, X_test_norm, y_test,
model_class=KNeighborsClassifier, weights = myweight)
```



- **Nhận xét:**

Kết quả hiện thực của nhóm giống với kết quả của thư viện sklearn.

10. TÀI LIỆU THAM KHẢO

[1] PGS.TS Dương Tuấn Anh, Chương 3 – Phân lớp dựa trên lân cận gần nhất, Học Máy và Ứng Dụng, Khoa Khoa Học và Kỹ Thuật Máy Tính, Đại Học Bách Khoa TPHCM.

[2] Vũ Hữu Tiệp, Blog Machine Learning Cơ Bản, https://machinelearningcoban.com/2017/01/08/knn/

[3] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestNeighbors

[4] Tutorial to Implement k-Nearest Neighbors in Python from Scratch https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/