**Bài 9: Unsupervised Learning: Clustering phần 2 - Lập trình AI bằng Python**

**4. Đo lường hiệu suất Clustering :**

Dữ liệu thế giới thực không tự nhiên được tổ chức thành một số cụm riêng biệt. Do đó, không dễ hình dung và rút ra suy luận. Đó là lý do tại sao chúng ta cần đo hiệu suất phân cụm cũng như chất lượng của nó. Nó có thể được thực hiện với sự trợ giúp của phân tích silhouette.

**a. Phân tích silhouette :**

Phương pháp này có thể được sử dụng để kiểm tra chất lượng phân cụm bằng cách đo khoảng cách giữa các cụm. Về cơ bản, nó cung cấp một cách để đánh giá các thông số như số lượng cụm bằng cách cho điểm silhouette. Điểm số này là một số liệu đo lường mức độ gần của mỗi điểm trong một cụm với các điểm trong các cụm lân cận.

**b. Phân tích điểm silhouette :**

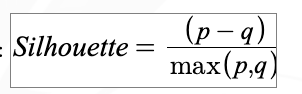
Điểm số có phạm vi là [-1, 1] như sau ;

* **Điểm +1**:  Điểm gần +1 cho biết mẫu ở xa cụm lân cận.
* **Điểm 0** : Điểm 0 cho biết mẫu nằm trên hoặc rất gần ranh giới quyết định giữa hai cụm lân cận.
* **Điểm -1 :** Điểm âm chỉ ra rằng các mẫu đã được chỉ định vào các cụm sai.

**5. Tính toán điểm Silhouette :**

Trong mục này, chúng ta sẽ học cách tính điểm **silhouette**.

Điểm Silhouette có thể được tính bằng cách sử dụng công thức sau:



Ở đây, 𝑝 là khoảng cách trung bình đến các điểm trong cụm gần nhất mà điểm dữ liệu không phải là một phần của nó. Và, 𝑞 là khoảng cách trung bình trong cụm đến tất cả các điểm trong cụm của chính nó.

Để tìm số lượng cụm tối ưu, chúng ta cần chạy lại thuật toán phân cụm bằng cách nhập mô-đun số liệu từ gói sklearn. Trong ví dụ sau, chúng tôi sẽ chạy thuật toán phân cụm K-mean để tìm số lượng cụm tối ưu -

Ở đây, 𝑝 là khoảng cách trung bình đến các điểm trong cụm gần nhất mà điểm dữ liệu không phải là một phần của nó. Và, 𝑞 là khoảng cách trung bình trong cụm đến tất cả các điểm trong cụm của chính nó.

Để tìm số lượng cụm tối ưu, chúng ta cần chạy lại thuật toán phân cụm bằng cách nhập mô-đun số liệu từ gói sklearn. Trong ví dụ sau, chúng tôi sẽ chạy thuật toán phân cụm K-mean để tìm số lượng cụm tối ưu :

**import** matplotlib.pyplot as plt

**import** seaborn as sns; sns.set()

**import** numpy as np

from sklearn.cluster **import** KMeans

Sau khi dùng đonaj code sau , ta sẽ tạo tập dữ liêuk 2 chiều, với 4 đốm màu, sử dụng make\_blob từ package sklearn.dataset

from sklearn.datasets.samples\_generator import make\_blobs

X, y\_true = make\_blobs(n\_samples = 500, centers = 4, cluster\_std = 0.40, random\_state = 0)

Khởi tạo :

scores = []

values = np.arange(2, 10)

Ta cần lặp lại mô hình K-mean qua các giá trị và cần huấn luyện nó với dữ liệu đầu vào

for num\_clusters in values:

kmeans = KMeans(init = 'k-means++', n\_clusters = num\_clusters, n\_init = 10)

kmeans.fit(X)

Bây giờ, hãy ước tính điểm **silhouette** cho mô hình phân cụm hiện tại bằng cách sử dụng số liệu khoảng cách Euclide -

score = metrics.silhouette\_score(X, kmeans.labels\_,

metric = 'euclidean', sample\_size = len(X))

Dòng mã sau đây sẽ giúp hiển thị số lượng các cụm cũng như điểm số Silhouette.

print("\nNumber of clusters =", num\_clusters)

print("Silhouette score =", score)

scores.append(score)

Number of clusters = 9

Silhouette score = 0.340391138371

num\_clusters = np.argmax(scores) + **values**[0]

**print**('\nOptimal number of clusters =', num\_clusters)

Output :

Optimal number of clusters = 2

**6. Thuật toán Finding Nearest Neighbors :**

Nếu chúng ta muốn xây dựng hệ thống recommender chẳng hạn như hệ thống giới thiệu phim thì chúng ta cần hiểu khái niệm finding the nearest neighbors. Đó là bởi vì hệ thống khuyến nghị sử dụng khái niệm về những người hàng xóm gần nhất.

Khái niệm tìm kiếm láng giềng gần nhất có thể được định nghĩa là quá trình tìm điểm gần nhất với điểm đầu vào từ tập dữ liệu đã cho. Việc sử dụng chính của thuật toán KNN ( K-nearest neighbors ) là xây dựng các hệ thống phân loại phân loại một điểm dữ liệu về mức độ lân cận của điểm dữ liệu đầu vào với các lớp khác nhau.

Đoạn code sau đưa ra dưới đây giúp tìm K-lân cận gần nhất của một tập dữ liệu nhất định -

**import** numpy as np

**import** matplotlib.pyplot as plt

from sklearn.neighbors **import** NearestNeighbors

Định nghĩa dự liệu input :

A = np.array([[3.1, 2.3], [2.3, 4.2], [3.9, 3.5], [3.7, 6.4], [4.8, 1.9],

[8.3, 3.1], [5.2, 7.5], [4.8, 4.7], [3.5, 5.1], [4.4, 2.9],])

Định nghĩa  nearest neighbors :

k = 3

Ta cần cung cấp dữ liệu test từ đó tìm thấy hàng xóm gần nhất

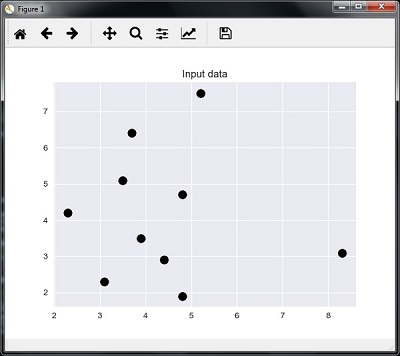
test\_data = [3.3, 2.9]

Đoạn code sau có thể trực quan hóa và vẽ biểu đồ dữ liệu đầu vào :

**plt**.figure()

**plt**.title('Input data')

**plt**.scatter(A[:,0], A[:,1], marker = 'o', s = 100, color = 'black')



Ta tiến hành xây dựng mô hình KNN

**knn\_model** = NearestNeighbors(n\_neighbors = k, algorithm = 'auto').fit(X)

distances, indices = knn\_model.kneighbors([test\_data])

Bây giờ, chúng ta có thể in K gần nhất như sau

print("\nK Nearest Neighbors:")

for rank, index in enumerate(indices[0][:k], start = 1):

print(str(rank) + " is", A[index])

Visualize các hàng xóm gần nhất cùng với điểm dữ liệu thử nghiệm

plt.figure()

plt.title('Nearest neighbors')

plt.scatter(A[:, 0], X[:, 1], marker = 'o', s = 100, color = 'k')

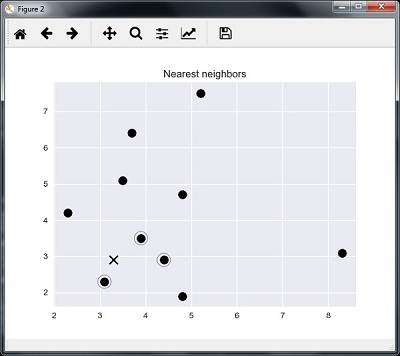
plt.scatter(A[indices][0][:][:, 0], A[indices][0][:][:, 1],

marker = 'o', s = 250, color = 'k', facecolors = 'none')

plt.scatter(test*\_data[0], test\_*data[1],

marker = 'x', s = 100, color = 'k')

plt.show()



Output :

1 is [ 3.1 2.3]

2 is [ 3.9 3.5]

3 is [ 4.4 2.9]

**7. K-Nearest Neighbors Classifier**

Bộ phân loại K-Nearest Neighbors (KNN) là một mô hình phân loại sử dụng thuật toán láng giềng gần nhất để phân loại một điểm dữ liệu nhất định. Chúng ta đã thực hiện thuật toán KNN trong phần trước, bây giờ chúng ta sẽ xây dựng bộ phân loại KNN bằng thuật toán đó.

**1. Khái niệm KNN Classifier :**

Khái niệm cơ bản của **KNN**classification là tìm một số được xác định trước, tức là 'k' - của các mẫu huấn luyện gần nhất trong khoảng cách với một mẫu mới, mà mẫu này phải được phân loại. Các mẫu mới sẽ được dán nhãn từ chính những người hàng xóm. Các bộ phân loại KNN có một hằng số do người dùng xác định cố định cho số lượng hàng xóm phải được xác định. Đối với khoảng cách, khoảng cách Euclid tiêu chuẩn là lựa chọn phổ biến nhất. KNN Classifier hoạt động trực tiếp trên các mẫu đã học thay vì tạo ra các quy tắc để học. Thuật toán KNN là một trong những thuật toán đơn giản nhất trong tất cả các thuật toán học máy. Nó đã khá thành công trong một số lượng lớn các bài toán phân loại và hồi quy, ví dụ, nhận dạng ký tự hoặc phân tích hình ảnh.

**2. Ví dụ**

Ta xây dựng bộ phân loại KNN để nhận dạng các chữ số. Đối với điều này, mình sẽ sử dụng tập dữ liệu MNIST. Mình sẽ viết mã này trong Jupyter Notebook.

Nhập các gói cần thiết như hình dưới đây.

Ở đây chúng tôi đang sử dụng mô-đun **KNeighborsClassifier**từ package **sklearn.neighbors**

from sklearn.datasets **import** \*

**import** pandas as pd

%matplotlib inline

from sklearn.neighbors **import** KNeighborsClassifier

**import** matplotlib.pyplot as plt

**import** numpy as np

Đoạn code sau sẽ hiển thị hình ảnh của chữ số để xác minh hình ảnh mà ta sẽi kiểm tra :

def Image\_display(i):

plt.imshow(digit['images'][i],cmap = 'Greys\_r')

plt.show()

Bây giờ, chúng ta cần tải tập dữ liệu MNIST. Trên thực tế có tổng cộng 1797 hình ảnh nhưng ta đang sử dụng 1600 hình ảnh đầu tiên làm mẫu đào tạo và 197 hình ảnh còn lại sẽ được giữ cho mục đích thử nghiệm.

digit = load\_digits()

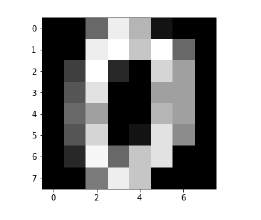
digit\_d = pd.DataFrame(digit['data'][0:1600])

Bây giờ, khi hiển thị hình ảnh, chúng ta có thể thấy kết quả như sau:

**Image\_display**(0)

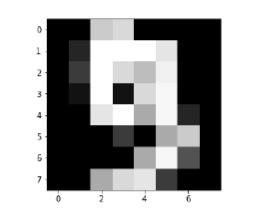
**Image\_display(0)**

image số 0 hiển thị như sau :



**Image\_display(9)**

image số 9 hiển thị như sau :



**digit.keys()**

Bây giờ, ta cần tạo tập dữ liệu đào tạo và thử nghiệm và cung cấp tập dữ liệu thử nghiệm cho các bộ phân loại KNN.

train\_x = digit['data'][:1600]

train\_y = digit['target'][:1600]

KNN = KNeighborsClassifier(20)

KNN.fit(train*\_x,train\_*y)

Đầu ra sau đây sẽ tạo ra K phương thức khởi tạo bộ KNN classifier:

KNeighborsClassifier(algorithm = 'auto', leaf\_size = 30, metric = 'minkowski',

metric\_params = None, n\_jobs = 1, n\_neighbors = 20, p = 2,

weights = 'uniform')

Ta cần tạo mẫu thử nghiệm bằng cách cung cấp bất kỳ số tùy ý nào lớn hơn 1600, là các mẫu huấn luyện.

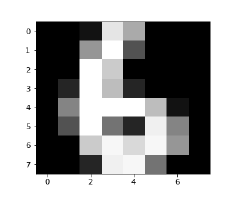
test = np.array(digit['data'][1725])

test1 = test.reshape(1,-1)

Image\_display(1725)

**Image\_display(6)**

Ảnh số 6 hiển thị như sau :



Ta nhận diện dữ liệu test như sau :

**KNN**.predict(test1)

Đoạn code trên sẽ tạo ra kết quả sau:

array([6])

Xem xét đoạn sau :

**digit**['target\_names']

Đoạn code trên sẽ tạo ra kết quả sau:

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])