ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

KHAI PHÁ DỮ LIỆU

BÀI TẬP LỚN

PHÂN CỤM DỮ LIỆU VỚI GIẢI THUẬT DBSCAN

GV:	Lê Hồng Trang	
SV:	Nguyễn Huỳnh Minh	1813085
	Lê Long	1812881
	Nguyễn Lê Tiến Đạt	1710959
	Nguyễn Trần Phương Khoa	1711790

TP. Ho Chi Minh, 12/2020

Mục lục

1 Giới thiệu

Chúng ta đang sống trong thời đại dữ liệu. Mỗi ngày, có đến hàng chục, hàng trăm petabyte dữ liệu đổ vào mạng máy tính và các thiết bị lưu trữ dữ liệu khác nhau từ xã hội, kinh doanh, khoa học - kỹ thuật, y học và hầu hết mọi khía cạnh khác của cuộc sống hàng ngày. Các mạng viễn thông đường trục toàn cầu mang hàng chục petabyte lưu lượng dữ liệu mỗi ngày. Hàng tỷ lượt tìm kiếm trên Web được hỗ trợ bởi các công cụ tìm kiếm xử lý hàng chục petabyte dữ liệu mỗi ngày. Cộng đồng và phương tiện truyền thông xã hội ngày càng trở thành nguồn dữ liệu quan trọng, tạo ra hình ảnh và video kỹ thuật số, blog, cộng đồng Web và nhiều loại mạng xã hội khác nhau. Danh sách các nguồn tạo ra lượng dữ liệu khổng lồ là vô tận. Khối dữ liệu khổng lồ đang phát triển bùng nổ, có sẵn rộng rãi và khổng lồ này khiến thời đại của chúng ta thực sự là thời đại của dữ liệu. Rất cần các công cụ mạnh mẽ và linh hoạt để tự động khám phá thông tin có giá trị từ lượng dữ liệu khổng lồ và chuyển đổi dữ liệu đó thành kiến thức có tổ chức. Sự cần thiết này đã dẫn đến sự ra đời của khai phá dữ liệu. Lĩnh vực này trẻ trung, năng động và đầy hứa hẹn. Khai phá dữ liệu đã và sẽ tiếp tục đạt được những bước tiến lớn trong hành trình của chúng ta từ thời đại dữ liệu sang thời đại thông tin sắp tới.

Phân tích cụm là một tác vụ chính của khai phá dữ liệu, và là một kỹ thuật phổ biến trong thống kê phân tích dữ liệu. Nó được ứng dụng trong nhiều lĩnh vực như phân tích ảnh, truy hồi thông tin, tin sinh học, nén dữ liệu, đồ họa máy tính, học máy,...

Một trong những giải thuật phân cụm phổ biến hiện nay là DBSCAN. Giải thuật này cho phép phát hiện cụm có hình dạng bất kỳ và có khả năng phát hiện nhiễu tốt. Song, giải thuật này gặp vấn đề ở việc tìm kiếm các điểm láng giềng, nên độ phức tạp của nó tỷ lệ với hàm bậc hai theo số lượng phần tử của tập dữ liệu. Trong bài viết này, nhóm chúng em đề xuất giải pháp tìm kiếm các điểm láng giềng sử dụng chỉ mục không gian KDTree nhằm tăng tốc quá trình tìm kiếm các điểm láng giềng, cải thiện tốc độ thực thi của giải thuật DBSCAN đối với các tập dữ liệu lớn.

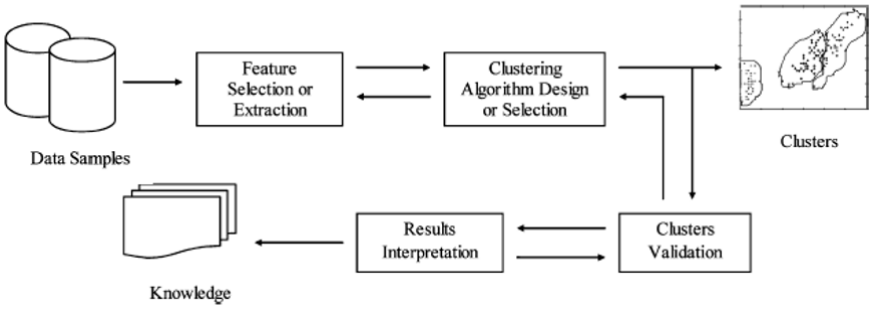
2 Cơ sở lý thuyết

2.1 Phân cụm dữ liệu

2.1.1 Phân cụm

Phân tích cụm hay đơn giản là phân cụm là quá trình phân vùng một tập hợp các đối tượng dữ liệu (hoặc các quan sát) thành các tập con. Mỗi tập hợp con là một cụm, sao cho các đối tượng trong một cụm tương tự với nhau, nhưng không giống với các đối tượng trong các cụm khác. Tập hợp các cụm kết quả từ một phân tích cụm có thể được gọi là phân cụm. Trong bối cảnh này, các phương pháp phân cụm khác nhau có thể tạo ra các chuỗi khác nhau trên cùng một tập dữ liệu. Việc phân vùng không phải do con người thực hiện mà do thuật toán phân cụm. Do đó, phân cụm hữu ích ở chỗ nó có thể dẫn đến việc phát hiện ra các nhóm chưa biết trước đó trong dữ liệu.

2.1.2 Quá trình phân cụm



Hình 1: Quá trình gom cụm (nguồn [4])

2.1.3 Các yêu cầu tiêu biểu về việc phân cụm dữ liệu

- Khả năng co giãn về tập dữ liệu (scalability)
- Khả năng xử lý nhiều kiểu thuộc tính khác nhau (different types of attributes)
- Khả năng khám phá các cụm với hình dạng tùy ý (clusters with arbitrary shape)
- Tối thiểu hóa yêu cầu về tri thức miền trong việc xác định các thông số nhập (domain knowledge for input parameters)
- Khả năng xử lý dữ liệu có nhiễu (noisy data)
- Khả năng gom cụm tăng dần và độc lập với thứ tự của dữ liệu nhập (incremental clustering and insensitivity to the order of input records)
- Khả năng xử lý dữ liệu đa chiều (high dimensionality)
- Khả năng gom cụm dựa trên ràng buộc (constraint-based clustering)
- Khả diễn và khả dụng (interpretability and usability)

2.1.4 Một số phương pháp phân cụm tiêu biểu

- Phân hoạch (partitioning): các phân hoạch được tạo ra và đánh giá theo một tiêu chí nào đó.
- Phân cấp (hierarchical): phân rã tập dữ liệu/đối tượng có thứ tự phân cấp theo một tiêu chí nào đó.
- Dựa trên mật độ (density-based): dựa trên connectivity and density functions.
- Dựa trên lưới (grid-based): dựa trên a multiple-level granularity structure.
- Dựa trên mô hình (model-based): một mô hình giả thuyết được đưa ra cho mỗi cụm; sau đó hiệu chỉnh các thông số để mô hình phù hợp với cụm dữ liệu/đối tượng nhất.

2.2 Giải thuật DBSCAN

2.2.1 Phân cụm dựa trên mật độ

Phân cụm dựa trên mật độ đề cập đến các phương pháp học tập không giám sát nhằm xác định các nhóm/ cụm khác biệt trong dữ liệu, dựa trên ý tưởng rằng một cụm trong không gian dữ liệu là một vùng không gian dày đặc (dense region), được phân tách bằng các cụm thưa thớt. Các điểm dữ liệu trong vùng phân cách có mật độ điểm thấp thường được coi là nhiễu/ ngoại lệ. Một số giải thuật phân cụm dựa trên mật độ tiêu biểu như:

- DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
- OPTICS (Ordering Points To Identify the Clustering Structure)
- DENCLUE (DENsity-based CLUstEring)

Trong báo cáo này, chúng ta tập trung vào giải thuật DBSCAN cải tiến sử dụng chỉ mục không gian (spatial index) KDTree.

2.2.2 Các khái niệm cơ bản

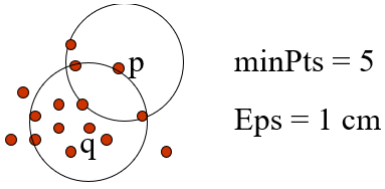
a) Tham số (Parameter)
Giải thuật DBSCAN nhận input là hai tham số là ϵ và minPts.
Trong đó:

- ϵ: bán kính của vùng lân cận của một đối tượng, gọi là ϵ-neighborhood.
- minPts: số điểm tối thiểu trong vùng lân cận (ϵ-neighborhood) của một đối tượng.

b) Vùng lân cận ngưỡng ϵ (ϵ-neighborhood)
Gọi D là tập cơ sở dữ liệu điểm. ϵ-neighborhood của điểm p, ký hiệu là $N_\epsilon(p)$, được xác định bởi $N_\epsilon(p) = \{q \in D | dist(p,q) \leq \epsilon\}$. Số lượng các phần tử của $N_\epsilon(p)$, ký hiệu là $|N_\epsilon(q)|$.

c) Đến được trực tiếp theo mật độ (Directly density – reachable)
Một điểm $p \in$ D là đến được trực tiếp theo mật độ từ một điểm $q \in$ D nếu:

- $p \in N_\epsilon(q)$
- $|N_\epsilon(q)| \geq minPts$ (điều kiện điểm lõi)

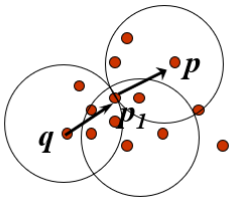


Hình 2: Đến được trực tiếp theo mật độ (nguồn [4])

d) Đến được theo mật độ (Density - reachable)

- Cho trước tập đối tượng D, ϵ và minPts.

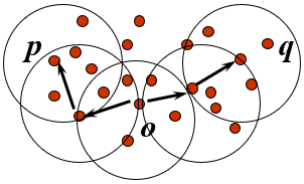
- Một điểm p được gọi là đến được từ một điểm q theo hai tham số ϵ và minPts nếu tồn tại một dãy các điểm p_1, p_2,...$p_n \in$ D với $p_1 = q$, $p_n = p$ sao cho p_{i+1} có thể truy cập trực tiếp theo mật độ từ p_i.
- Bao đóng truyền (transitive closure) của directly density-reachable.
- Quan hệ bất đối xứng (asymmetric relation).



Hình 3: Đến được theo mật độ(nguồn [4])

e) Liên thông mật độ (Density-connected)

- Cho trước tập đối tượng D, ϵ và minPts.
- $p, q \in D$
- Điểm p được gọi là điểm liên thông với q theo hai tham số ϵ và minPts nếu tồn tại một điểm o sao cho cả hai điểm p và q đều có thể đạt được theo mật độ từ o theo hai tham số ϵ và minPts.
- Quan hệ đối xứng

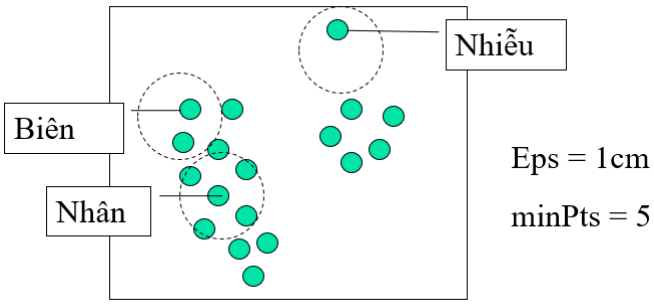


Hình 4: Liên thông mật độ (nguồn [4])

f) Cụm (Cluster)

Dựa trên khái niệm cụm dựa trên mật độ: Một cụm được định nghĩa là một tập hợp tối đa các điểm liên thông mật độ (density-connected points).
Đối tượng thuộc về cụm có thể là core object. Nếu đối tượng đó không là core object thì gọi là đối tượng biên (border object).

g) Nhiễu (Noise)
Gọi C_1, C_2,...C_k là các cụm của tập dữ liệu D. Nhiễu là tập hợp tất cả các điểm không thuộc bất kỳ cụm C_i nào với $i = 1, ..., k$.



Hình 5: Dữ liệu nhiễu (nguồn [4])

2.2.3 Giải thuật DBSCAN

- Input: tập đối tượng D, ϵ, minPts
- Output: density-based clusters (và noise/outliers)
- Giải thuật:

1. Xác định ϵ–neighborhood của mỗi đối tượng p $\in$ D.
2. Nếu p là đối tượng lõi, tạo được một cluster.
3. Từ bất kỳ đối tượng lõi p, tìm tất cả các đối tượng density-reachable và đưa các đối tượng này (hoặc các cluster) vào cùng cluster ứng với p.
   3.1 Các cluster đạt được (density-reachable cluster) có thể được trộn lại với nhau.
   3.2 Dừng khi không có đối tượng mới nào được thêm vào.

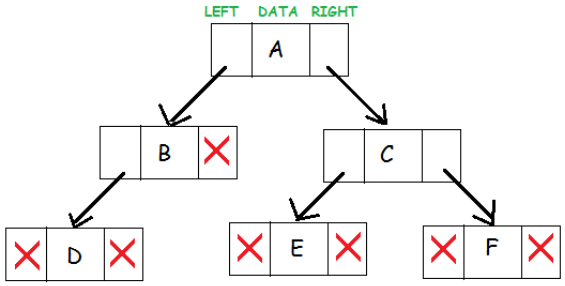
2.3 KD Tree

2.3.1 Cây tìm kiếm nhị phân

a) Cây nhị phân
Cây nhị phân là một cấu trúc dữ liệu cây, mà mỗi node có nhiều nhất 2 node con, được gọi là *con trái*, *con phải*.
Mỗi node chứa các thành phần:

- Dữ liệu
- Con trỏ trỏ đến cây con trái
- Con trỏ trỏ đến cây con phải

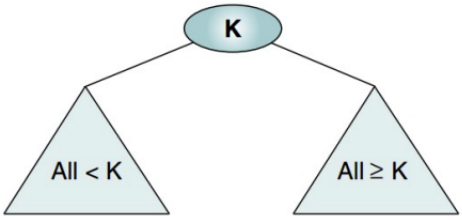
Node trên cùng trong cây gọi là *root*. Mỗi cây trống có giá trị là *null*.



Hình 6: Cây nhị phân (nguồn [6])

b) Cây tìm kiếm nhị phân
Cây tìm kiếm nhị phân là cây nhị phân với những tính chất:

- Cây con trái của một node chỉ chứa các node với *key* nhỏ hơn *key* của node đó.
- Cây con phải của một node chỉ chứa các node với *key* không nhỏ hơn *key* của node đó.
- Cây con trái và cây con phải là các cây tìm kiếm nhị phân.



Hình 7: Cây tìm kiếm nhị phân

c) Ưu, nhược điểm của cấu trúc cây

- Ưu điểm:
  - Phản ảnh mối quan hệ cấu trúc trong dữ liệu.
  - Được sử dụng để thể hiện cho hệ thống phân cấp.
  - Thao tác chèn, tìm kiếm hiệu quả.
- Nhược điểm:
  - Tốn chi phí *overhead*.
  - Độ phức tạp lớn hơn để khởi tạo và duy trì.

2.3.2 KD Tree

a) Khái niệm
KD Tree (viết đầy đủ là K-Dimensional Tree) là cây tìm kiếm nhị phân, trong đó dữ

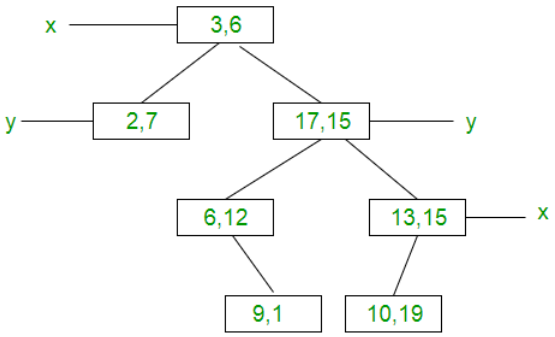
liệu trong mỗi node có k *chiều* trong không gian, là cấu trúc dữ liệu phân vùng không gian để tổ chức các điểm trong không gian k *chiều*.

Mỗi node trong của K-D Tree chia không gian thành hai phần, gọi là nửa không gian. Các điểm ở bên trái của không gian này được biểu thị bằng cây con trái của node đó, các điểm ở bên phải của không gian này được thể hiện bằng cây con phải.

b) Ý tưởng hiện thực

Mỗi level của cây (tương ứng với độ sâu) sẽ được so sánh trên 1 chiều. Khái quát, với không gian k *chiều*, đánh số các chiều phân chia lần lượt là 0, 1, 2,..., $k-1$. Một node có độ sâu d sẽ được phân chia ở chiều thứ d *mod* k.

Để đơn giản, lấy ví dụ là K-D Tree với số chiều là 2, có thể dễ dàng biểu diễn các điểm dữ liệu trên mặt phẳng Oxy. Theo ý tưởng trên, ban đầu *root* được so sánh theo trục x, các node con của node root được chia theo trục y, node cháu của root được chia theo trục x,... Vì node root được căn chỉnh theo trục x, nên các node của cây con trái sẽ có giá trị tọa độ theo trục x nhỏ hơn root. Tương tự, cây con phải chứa các node có giá trị tọa độ theo trục x lớn hơn bằng root.



Hình 8: Minh họa về KD Tree với 2 chiều (nguồn [1])

c) Giải thuật

Với đầu vào là danh sách n điểm, giải thuật với mã giả sau đây (xem [9]) sẽ tạo ra kd-tree cân bằng chỉ chứa các điểm đã cho.

Algorithm 1 Build a balanced KD Tree

```
1: function CREATE_KDTREE(pointList: list of points, depth: int)
2:     if pointList is empty then
|          return NULL
       end
3:
4:     axis ← depth mod k
5:     value ← find_median(pointList, axis)
6:     node.value ← value
7:     node.leftChild ← create_kdtree(points in pointList before median, depth + 1)
8:     node.rightChild ← create_kdtree(points in pointList after median, depth + 1)
9:     return node
```

Với giải thuật trên, tất cả các nodes của cây con trái nằm một phía của mặt phẳng phân chia, và tất cả các nodes của cây con phải nằm ở phía còn lại của mặt phẳng phân chia. Mặt phẳng phân chia sẽ đi qua điểm của node hiện tại (cụ thể là $node.value$). Có thể có những điểm khác cũng nằm trên mặt phẳng phân chia sẽ thuộc về 1 trong 2 phía.

d) Đánh giá giải thuật

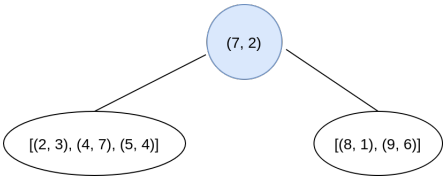
- Độ phức tạp thời gian tốt nhất của giải thuật trên là $O(nlog(n))$. Nếu danh sách n điểm được sắp xếp trước theo mỗi chiều của k chiều bằng cách sử dụng các thuật toán có độ phức tạp $O(nlog(n))$ như heapsort, mergesort, quicksort,... Tuy nhiên, với cách này sẽ tốn bộ nhớ vì phải lưu lại danh sách đã được sắp xếp cho mỗi chiều.
- Độ phức tạp thời gian xấu nhất là $O(knlog(n))$. Nếu cứ mỗi chiều trong k chiều, chúng ta lại thực hiện sắp xếp lại danh sách. Nhưng cách này sẽ ít hơn bộ nhớ hơn cách trên.

e) Minh họa với dữ liệu 2 chiều

Để hiểu hơn về giải thuật trên, chúng ta sẽ áp dụng vào một tập dữ liệu nhỏ với 2 chiều như sau: (2, 3), (5, 4), (9, 6), (4, 7), (8, 1), (7, 2).

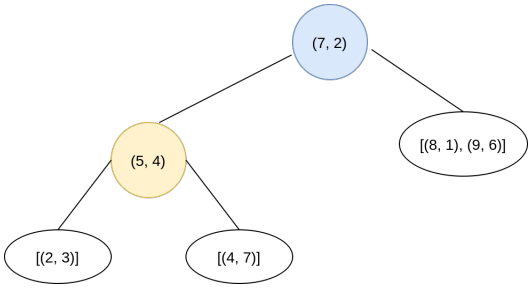
Các bước thực hiện của giải thuật như bên dưới, sau khi đã làm vắn tắt 1 số bước không cần thiết để lời giải được gọn hơn. Sau mỗi bước đều có 1 ảnh để minh họa cho kết quả. Với các node mà giá trị được so sánh theo trục x sẽ có màu danh dương, còn các node mà giá trị được so sánh theo trục y có màu vàng. Các node đã được $visit$ qua có hình tròn, còn các cây con là hình oval chứa danh sách các phần tử.

1. Ban đầu, vì cây đang rỗng nên $depth = 0$, suy ra $axis = 0$, do đó các điểm được so sánh theo trục x. Phần tử trung vị của danh sách ban đầu theo x là (7, 2). Các phần tử ở cây con trái là (2, 3), (4, 7), (5, 4). Các phần tử ở cây con phải là (8, 1), (9, 6). Sau đó, đệ quy xuống cây con trái, rồi đệ quy xuống cây con bên phải.



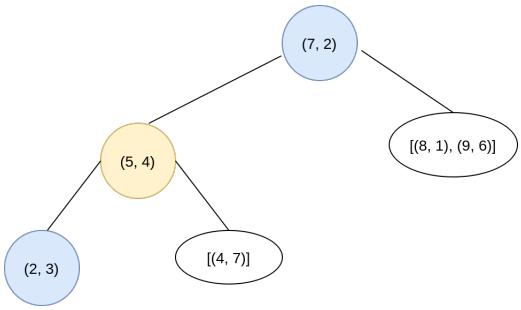
Hình 9: Kết quả sau bước 1

2. Danh sách phần tử hiện tại là (2, 3), (4, 7), (5, 4), giá trị $depth = 1$, suy ra $axis = 1$, nên các điểm được so sánh theo trục y. Phần tử trung vị là (5, 4). Các phần tử ở cây con trái là (2, 3). Các phần tử ở cây con phải là (4, 7). Sau đó, đệ quy xuống cây con trái, rồi đệ quy xuống cây con bên phải.



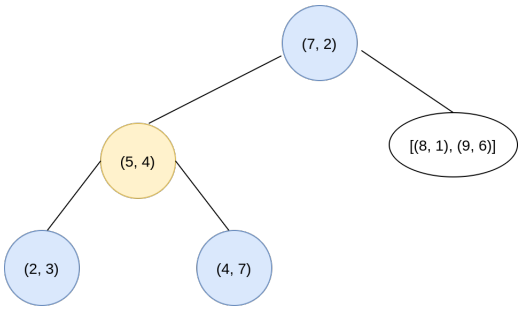
Hình 10: Kết quả sau bước 2

3. Danh sách phần tử hiện tại là (2, 3), giá trị $depth = 2$, suy ra $axis = 0$, nên các điểm được so sánh theo trục x. Phần tử trung vị là (2, 3). Danh sách các phần tử của cây con bên trái và phải đều rỗng nên cả 2 cây con đều bị *null*. Sau đó, quay lên node cha của node hiện tại. Vì ở node cha có giá trị *value* là (5, 4) vẫn chưa đệ quy xuống cây con phải nên sẽ tiếp tục.



Hình 11: Kết quả sau bước 3

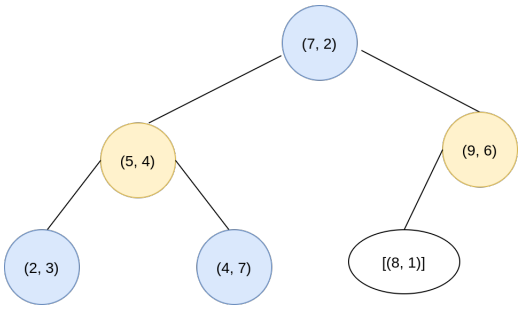
4. Danh sách phần tử hiện tại là (4, 7). Giá trị $depth = 2$, suy ra $axis = 0$, nên các điểm được so sánh theo trục x. Phần tử trung vị là (4, 7). Danh sách các phần tử của cây con bên trái và phải đều rỗng nên cả 2 cây con đều bị *null*. Sau đó, sẽ quay lên node có giá trị *value* là (7, 2), tiếp tục đệ quy xuống cây con phải.



Hình 12: Kết quả sau bước 4

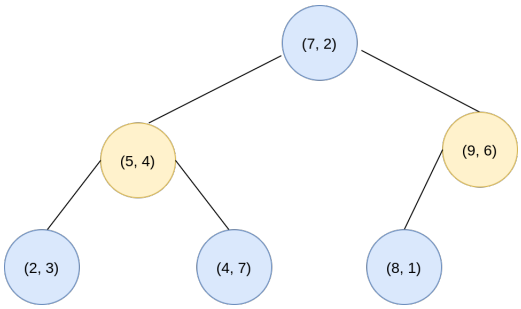
5. Danh sách các phần tử hiện tại là (8, 1), (9, 6). Giá trị $depth = 1$, suy ra $axis = 1$, nên các điểm được so sánh theo trục y. Phần tử trung vị là (9, 6). Các phần tử cây

con trái là (8, 1). Danh sách các phần tử của cây con phải rỗng nên cây con phải có giá trị *null*. Sau đó, đệ quy xuống cây con trái.



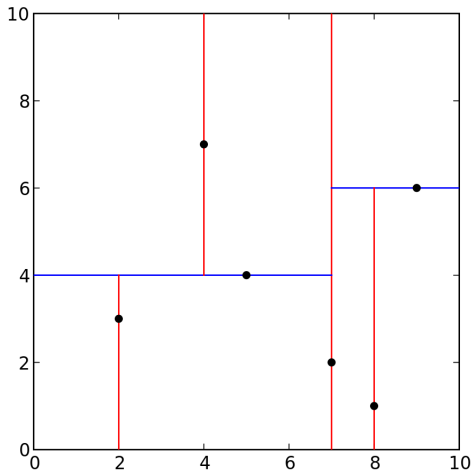
Hình 13: Kết quả sau bước 5

6. Danh sách phần tử hiện tại là (8, 1). Giá trị $depth = 2$, suy ra $axis = 0$, nên các điểm được so sánh theo trục x. Phần tử trung vị là (8, 1). Danh sách các phần tử của cây con bên trái và phải đều rỗng nên cả 2 cây con đều bị *null*. Tới đây thì đã hoàn thành xong việc xây dựng kd tree.



Hình 14: Kết quả sau bước 6

Vì tập dữ liệu chỉ có 2 chiều, nên chúng ta có thể dễ dàng biểu diễn các điểm lên mặt phẳng tọa độ như hình 15. Các đường màu đỏ và xanh chính là các đường phẳng phân chia, cụ thể đường màu đỏ là chia theo trục x, còn màu xanh là chia theo trục y.



Hình 15: Minh họa kết quả trên mặt phẳng tọa độ (nguồn [9])

f) Ứng dụng
Trong thực tế, KD Tree có khá nhiều ứng dụng như

- Nearest neighbor search
- Truy vấn cơ sở dữ liệu có search key nhiều chiều
- Giải bài toán N-body

2.4 Radius neighbor search

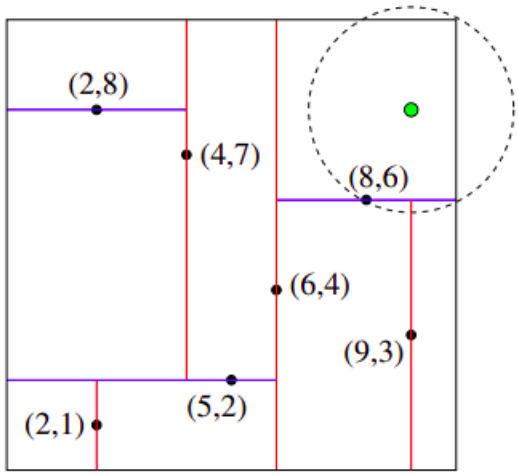
Bài toán đặt ra là chúng ta có danh sách n điểm, mỗi điểm có d chiều ($d > 1$). Với một điểm p trong danh sách đã có và 1 số ϵ cho trước, tìm tất cả các điểm trong danh sách có khoảng cách Euler với p không quá ϵ.

2.4.1 Giải thuật vét cạn

Đây là một giải thuật cơ bản để giải quyết bài toán trên. Chúng ta chỉ cần duyệt qua toàn bộ danh sách, và tính khoảng cách của từng điểm với p, chỉ lấy các điểm thỏa mãn điều kiện. Với cách này, độ phức tạp thời gian luôn là $O(n)$, nếu chúng ta có rất nhiều truy vấn tương tự thì sẽ tốn nhiều thời gian để thực thi.

2.4.2 Giải thuật tìm kiếm với KD Tree

Để khắc phục sự hạn chế về mặt thời gian của giải thuật vét cạn, nhóm áp dụng 1 giải thuật tìm kiếm với cấu trúc dữ liệu KD Tree đã được trình bày ở 2.3. Để dễ giải thích, chúng ta sẽ dùng tập dữ liệu 2 chiều.



Hình 16: Minh họa giải thuật tìm kiếm

Theo hình 16, node root có giá trị là (6, 4), điểm p là điểm màu xanh thuộc cây con bên phải của node root, ϵ là bán kính của đường tròn nét gạch đứt. Gọi đường thẳng phân chia đi qua điểm (6, 4) là đường thẳng s. Dễ dàng nhận thấy khoảng cách từ điểm p tới đường thẳng s lớn hơn ϵ, nên chúng ta không cần phải duyệt qua cây con bên trái của node root để tìm kiếm, nhờ vậy mà đã bỏ qua được một nửa các điểm của node root. Do đó, thời gian tìm kiếm cũng được cải thiện so với giải thuật vét cạn.
Tổng quát, với điểm p cho trước chúng ta dễ dàng xác định được nó thuộc cây con bên trái hay bên phải của node hiện tại. Chúng ta chỉ duyệt qua cây con còn lại nếu thỏa mãn điều kiện

$$d(p, splitting\_plane(node), axis) \leq \epsilon$$

Trong đó:

- d: là hàm đo khoảng cách
- $splitting\_plane(node)$: mặt phẳng phân chia đi qua node hiện tại
- $axis$: chiều dữ liệu của node hiện tại dùng để so sánh

3 Hiện thực

3.1 Giải thuật DBSCAN

Chúng ta có mã giả của giải thuật DBSCAN như sau (xem [8]).

```
DBSCAN(D, eps, MinPts)
C = 0
for each unvisited  point P in dataset D
    mark P as visited
    NeighborPts = regionQuery(P, eps)
    if sizeof(NeighborPts) < MinPts
        mark  P as NOISE
    else
        C = next cluster
        expandCluster(P, NeighborPts, C, eps, Minpts)

expandCluster(P, NeighborPts, C, eps, MinPts)
    add P to cluster C
    for each point P' in NeighborPts
        if P' is not visited
            mark P' as visited
            NeighborPts' = regionQuery(P', eps)
            if sizeof(NeighborPts') >= MinPts
                NeighborPts = NeighborPts join with NeighborPts'
        if P' is not yet member of any cluster
            add P' to cluster C

regionQuery(P, eps)
    return all points within P's eps-neighborhood (including P)
```

Trong đó:

- D: Tập các điểm dữ liệu.
- eps: Khoảng cách quy định tính từ tọa độ của một điểm dữ liệu cần xét.
  Xem lại khái niệm (1).
- MinPts: Số lượng điểm dữ liệu tối thiểu phải có trong vùng lân cận của một điểm dữ liệu.
  Xem lại khái niệm (2).
- Hàm expandCluster: tìm các điểm dữ liệu trong tập dữ liệu cùng nhóm với điểm dữ liệu truyền vào và gán các điểm dữ liệu vào nhóm tương ứng.
- Hàm regionQuery: Trả về các điểm láng giềng của điểm dữ liệu truyền vào, tính cả điểm dữ liệu đó.

3.2 Giải thuật DBSCAN cải tiến

Với mã giả ở 3.1, chúng ta dễ dàng nhận thấy hàm *regionQuery* tốn khá nhiều thời gian thực thi của chương trình. Do đó, chúng ta sẽ xây dựng 2 phiên bản hiện thực của giải thuật DBSCAN (chỉ khác nhau hàm regionQuery) để so sánh độ hiệu quả về thời gian thực thi, cụ thể:

- Phiên bản 1: hàm regionQuery tìm kiếm tuyến tính, với giải thuật vét cạn đã đề cập ở 2.4.1. Mã giả của hàm tìm kiếm regionQuery như sau.

```
Result: empty array
regionQuery(P, eps)
    for each point P' in D
        if distance(P, P') <= eps
            add P' to Result
    return Result
```

  Trong đó:

  - D: tập các điểm dữ liệu
  - P: điểm dữ liệu cần tìm các điểm láng giềng
  - eps: khoảng cách để xác định một điểm có phải láng giếng của P

- Phiên bản 2: hàm regionQuery hiện thực giải thuật tìm kiếm với KD Tree ở 2.4.2. Mã giả chi tiết như sau.

```
Result: empty array
regionQuery(root, P, eps, depth)
    if root is null
        Return Result
    axis = depth % 2
    split_point = root.value
    if distance(split_point, P) <= eps
        add split_point to Result
    if split_point[axis] < P[axis]
        nearer_branch = root.rightChild
        further_branch = root.leftChild
    else
        nearer_branch = root.leftChild
        further_branch = root.rightChild
    regionQuery(nearer_branch, P, eps, depth + 1)
    if distance(split_point, P, axis) <= eps
        regionQuery(further_branch, P, eps, depth + 1)
    Return Result
```

  Trong đó:

  - root: cây KD Tree được xây dựng từ tập dữ liệu ban đầu
  - P: điểm dữ liệu cần tìm các điểm láng giềng
  - eps: khoảng cách để xác định một điểm có phải láng giếng của P
  - depth: độ sâu của node hiện tại, giá trị ban đầu là 0

3.3 Mã nguồn

Nhóm đã hiện thực toàn bộ giải thuật DBSCAN và KD Tree bằng ngôn ngữ lập trình Pascal. Mã nguồn chi tiết được gửi kèm cùng với bài báo cáo này.

4 Kết quả

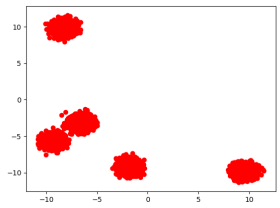
4.1 Môi trường thực thi

Nhóm đã biên dịch và thực thi với Free Pascal compiler phiên bản 3.2.0. Tất cả kết quả ghi nhận đều được thực hiện trên máy tính cá nhân (chip Intel 2.8 GHz x 8GB RAM) chạy hệ điều hành Windows 10 64-bit (phiên bản 10.0 build 18362)

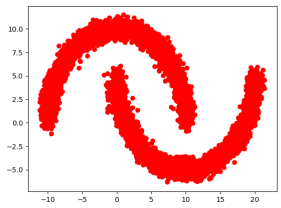
4.2 Các tập dữ liệu kiểm thử

Trong báo cáo bài tập lớn này, nhóm sử dụng các tập dữ liệu 2 chiều không nhãn với hình dạng đặc trưng được sinh bởi thư viện scikit-learn của Python (xem [5]). Các tập dữ liệu này được lấy nguồn https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html.

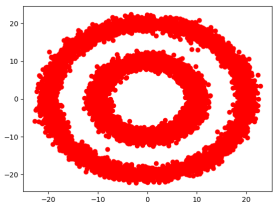
Phân bố của các tập dữ liệu được hiển thị bởi các hình dưới đây:



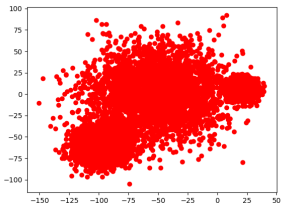
Tập dữ liệu 1



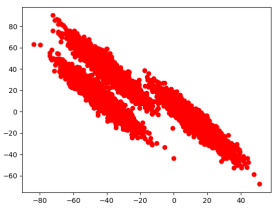
Tập dữ liệu 2



Tập dữ liệu 3



Tập dữ liệu 4

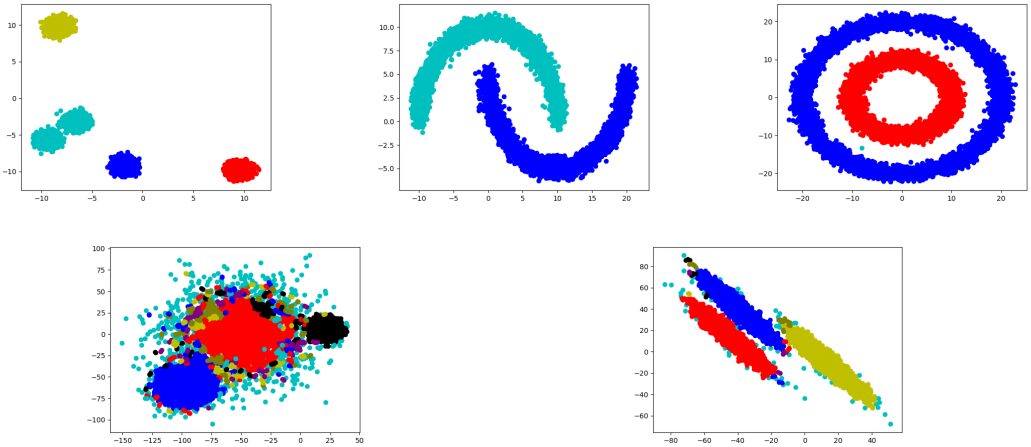


Tập dữ liệu 5

Hình 17: Các tập dữ liệu kiểm thử phần hiện thực

4.3 Kết quả hiện thực

Kết quả phân cụm các tập dữ liệu nói trên bằng giải thuật DBSCAN kết hợp với giải thuật tìm kiếm láng giềng KD Tree được minh họa bởi các hình dưới đây. Tất cả các tập dữ liệu đều sử dụng bộ tham số $\epsilon = 2$ và minPts $= 2$, với số mẫu là 10000.

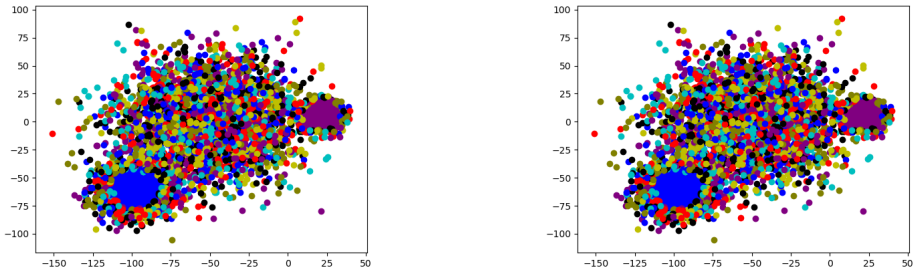


Hình 18: Kết quả phân cụm các tập dữ liệu nêu trên

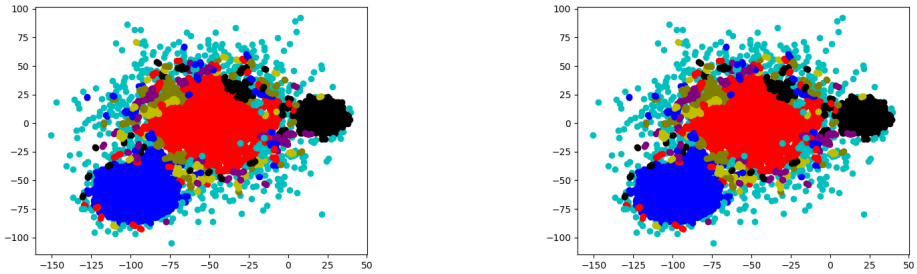
4.4 So sánh

4.4.1 Độ chính xác

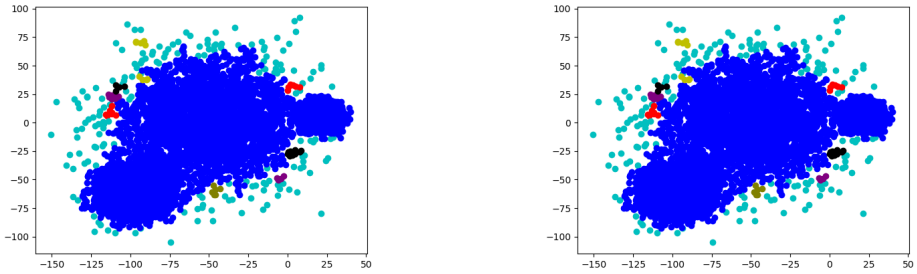
Trong phần này, với số mẫu cố định là 10000, các tham số ϵ và minPts sẽ được thay đổi để đánh giá kết quả phân cụm của giải thuật DBSCAN kết hợp với KD Tree được nhóm hiện thực, dựa trên kết quả phân cụm từ giải thuật DBSCAN được hiện thực từ thư viện scikit-learn. Kết quả phân cụm của 2 giải thuật được minh họa bởi các cặp hình vẽ bên dưới. Tập dữ liệu 4 được sử dụng ở đây.



Hình 19: $\epsilon = 1$, minPts $= 1$, độ chính xác 100%



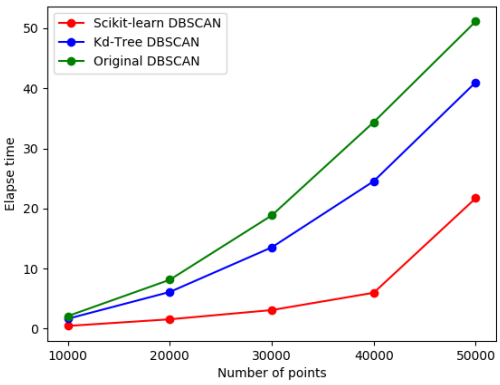
Hình 20: $\epsilon = 2$, minPts $= 2$, độ chính xác 100%

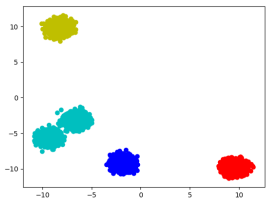


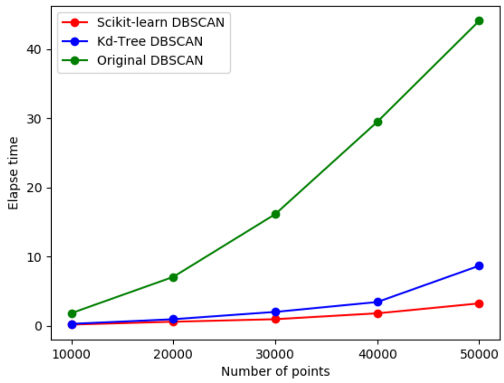
Hình 21: $\epsilon = 4$, minPts $= 4$, độ chính xác 100%

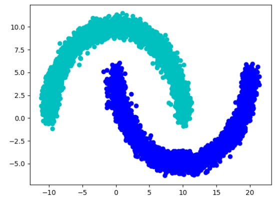
4.4.2 Hiệu suất

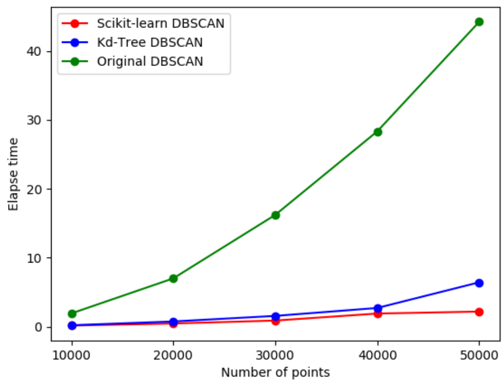
Để đánh giá hiệu suất của các giải thuật được hiện thực (DBSCAN sử dụng tìm kiếm tuần tự và DBSCAN sử dụng tìm kiếm với KD Tree), nhóm tiến hành so sánh thời gian phân cụm của 2 giải thuật này với thời gian phân cụm của giải thuật được hiện thực trong thư viện scikit-learn. Trong đó, các tập dữ liệu với hình dạng đã được giới thiệu ở phần trước với độ lớn của tập dữ liệu điểm chạy từ 10000 đến 50000 và các thông số không đổi $\epsilon = 2$, minPts $= 2$. Kết quả đo và dạng phân bố dữ liệu tương ứng được minh họa bởi các hình dưới đây:

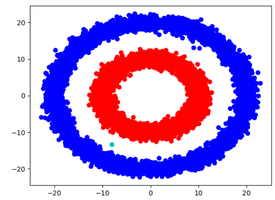


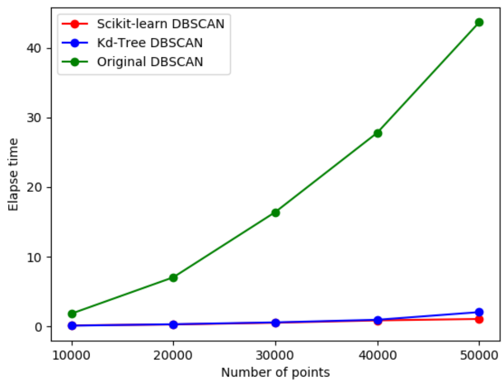


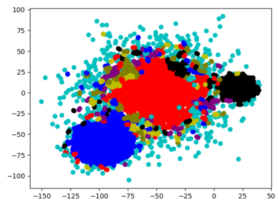

Scikit-learn DBSCAN
Kd-Tree DBSCAN
Original DBSCAN
Elapse time
Number of points

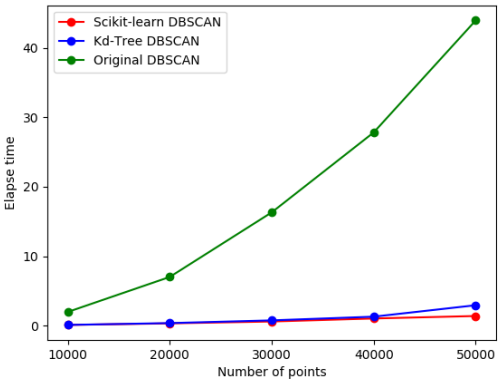


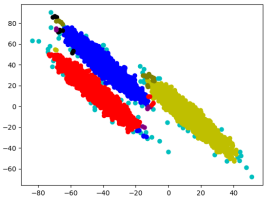

Scikit-learn DBSCAN
Kd-Tree DBSCAN
Original DBSCAN
Elapse time
Number of points




Scikit-learn DBSCAN
Kd-Tree DBSCAN
Original DBSCAN
Elapse time
Number of points







4.5 Đánh giá kết quả

4.5.1 Độ chính xác

Trong phần so sánh nêu trên, nhóm đã tiến hành kiểm tra độ chính xác của giải thuật DBSCAN sử dụng KD Tree dựa trên kết quả phân cụm có được từ việc sử dụng thư viện scikit-learn của Python. Kết quả phân cụm thu được hoàn toàn giống với kết quả mẫu khi thay đổi các thông số đầu vào ϵ và minPts.

4.5.2 Hiệu suất

- Giải thuật DBSCAN sử dụng tìm kiếm bằng KD Tree cho hiệu suất cải thiện rõ rệt so với việc sử dụng chiến lược tìm kiếm tuần tự (kết quả đã được minh họa trong phần trước), mặc dù vẫn chậm hơn so với thư viện sklearn.
- Đối với các tập dữ liệu có độ nhiễu lớn (như ở các dataset 4 và 5), việc tìm kiếm bằng KD Tree cho thấy hiệu quả rõ rệt, thể hiện qua thời gian thực thi gần như tương đương so với thư viện của Python.

5 So sánh giải thuật DBSCAN và KMeans

5.1 Giải thuật KMeans

5.1.1 Ưu điểm

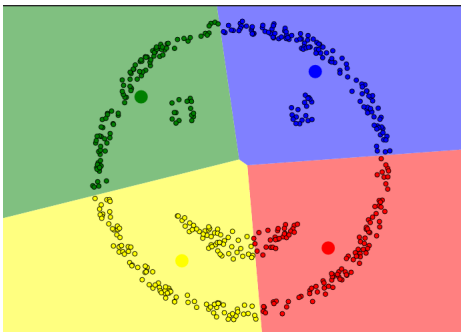
- Tương đối đơn giản khi hiện thực.
- Mở rộng (scale) được đối với tập dữ liệu lớn.
- Đảm bảo việc hội tụ.
- Có thể khởi tạo tâm các cụm.
- Có thể dễ dàng áp dụng đối với các vấn đề mới.

5.1.2 Nhược điểm

- Chọn số cụm thủ công.
- Kết quả phân cụm cũng phụ thuộc vào các điểm khởi tạo
- Gặp vấn đề trong việc phân cụm các điểm dữ liệu với nhiều kích thước và mật độ khác nhau.
- Phân cụm không hiệu quả vì nhạy cảm với nhiễu.
- Các cụm phải có số lượng các điểm dữ liệu gần bằng nhau.
- Các cụm cần có dạng hình tròn.
- Khi một cụm nằm phía trong 1 cụm khác.

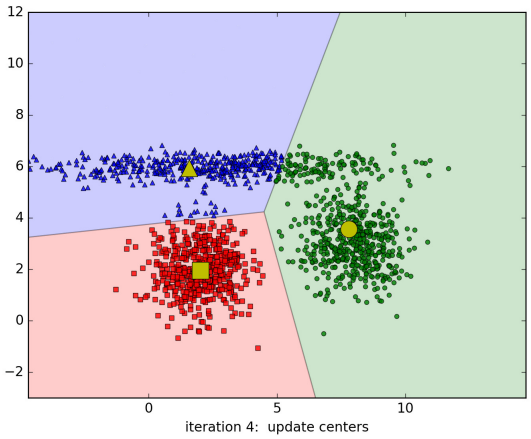
5.1.3 Ví dụ

1. Đây là ví dụ kinh điển về việc K-means clustering không thể phân cụm dữ liệu. Một cách tự nhiên, chúng ta sẽ phân ra thành 4 cụm: mắt trái, mắt phải, miệng, xunh quanh mặt. Nhưng vì mắt và miệng nằm trong khuôn mặt nên K-means clustering không thực hiện được.



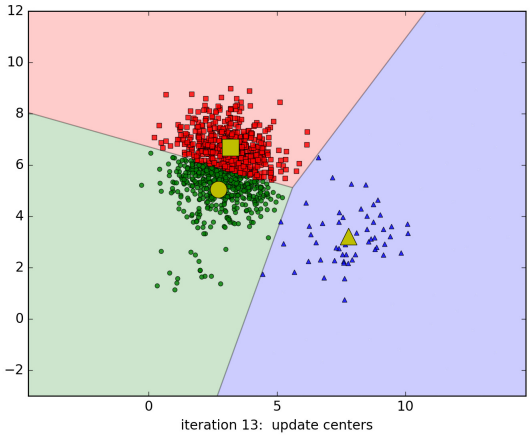
Hình 22: Cụm nằm trong một cụm khác.

2. Các cluster cần có dạng hình cầu (tròn).
Dưới đây là 1 ví dụ khi 1 cluster có dạng hình dẹt.



Hình 23: Một cụm có dạng hình dẹt (nguồn [7])

3. Các cluster cần có só lượng điểm gần bằng nhau, khi ta khởi tạo 3 cluster với số lượng phần tử lần lượt là 20, 50, 1000 điểm thì kết quả phân cụm sẽ sai.



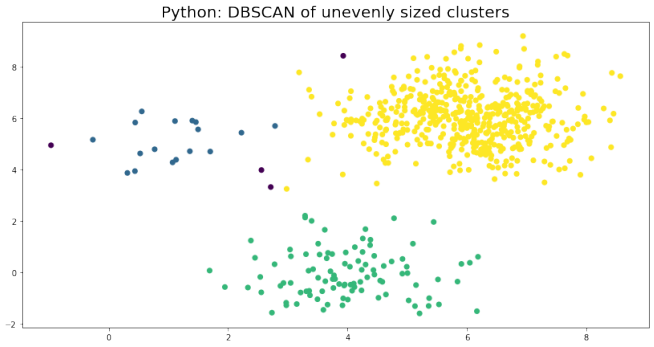
Hình 24: Số lượng điểm dữ liệu trong cụm khác nhau (nguồn [7])

5.2 Giải thuật DBSCAN

5.2.1 Ưu điểm

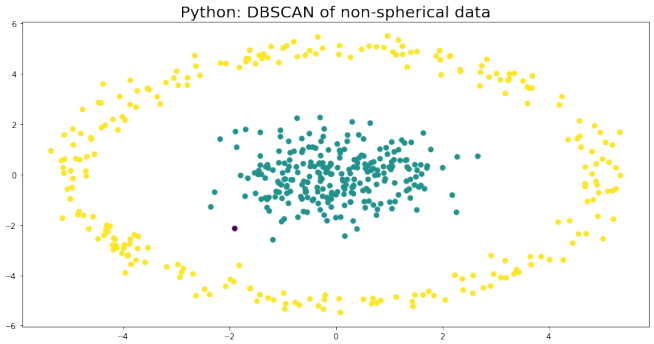
- Không cần phải xác định trước số cụm cần phân cụm trước khi phân cụm.
- Kết quả tốt đối với hình dạng cụm dữ liệu tùy ý.
- DBSCAN có thể phát hiện và xử lý tốt các ngoại lệ (outlier).

Khi sử dụng DBSCAN thì ta có thể khắc phục được một số nhược điểm của KMean như sau:
1. Giải quyết được vấn đề số lượng điểm dữ liệu mỗi cụm khác nhau nhiều.



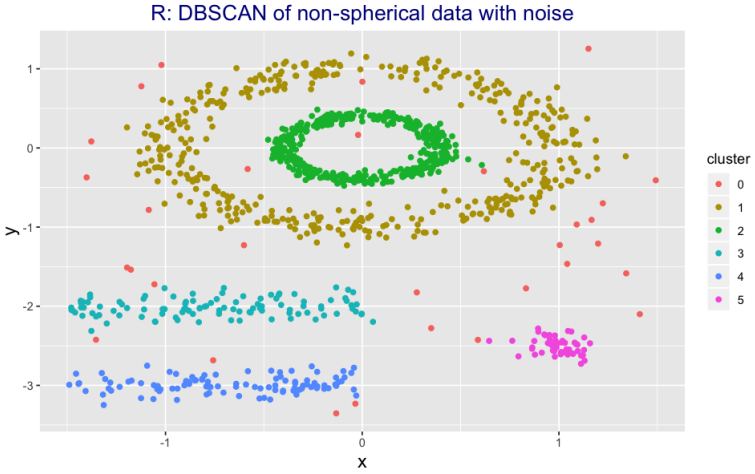
Hình 25: Số lượng điểm dữ liệu trong cụm khác nhau

2. Dữ liệu của một cụm nằm trong cụm khác vẫn phân cụm đúng.



Hình 26: Cụm dữ liệu này nằm trong cụm dữ liệu khác

3. Giải quyết được vấn đề các cụm cần phải có hình tròn, hình cầu và có thể phát hiện và xử lý nhiễu và ngoại lệ.



Hình 27: Cụm dữ liệu hình dạng khác nhau và có nhiễu

5.2.2 Nhược điểm

- Trong một số trường hợp việc xác định khoảng cách cho phép đến các điểm láng giềng (ϵ) đôi khi không dễ và cần có kiến thức chuyên môn liên quan tới vấn đề.
- Nếu các cụm khác nhau về mật độ các điểm trong cụm, DBSCAN khó xác định các cụm dữ liệu. Vì mỗi lần truyền tham số thì ta chỉ truyền một bộ (ϵ, minPts) nên không thể khái quát được các cụm với mật độ khác nhau.
- Khó xác định nhóm khi các điểm dữ liệu quá thưa.

6 Tổng kết

Qua bài tập lớn này, chúng ta đã hiểu về ý tưởng và cách hiện thực của thuật toán phân cụm DBSCAN. Thuật toán DBSCAN có một số điểm mạnh như không cần phải xác định trước số cụm, có khả năng phát hiện và xử lý dữ liệu nhiễu, kết quả tốt đối với hình dạng cụm dữ liệu bất kỳ.

Nhóm cũng đã hiện thực bằng ngôn ngữ lập trình Pascal theo yêu cầu của bài tập, mã nguồn được gửi kèm cùng với báo cáo này. Đồng thời, nhóm cũng đề xuất cải tiến thời gian thực thi bằng cách sử dụng cấu trúc dữ liệu KDTree để giúp tìm kiếm nhanh hơn. Thông qua các thí nghiệm cho thấy thời gian thực thi từ việc cải tiến cho thấy giảm rõ rệt so với phiên bản ban đầu khi tập dữ liệu có kích thước càng lớn và kết quả phân cụm đạt được giống với kết quả từ thư viện scikit-learn.

Tài liệu tham khảo

[1] Aashish Barnwal. *K Dimensional Tree | Set 1 (Search and Insert)*. URL: https://www.geeksforgeeks.org/k-dimensional-tree/ (visited on 12/01/2020).

[2] J. Behley, V. Steinhage, and A. B. Cremers. "Efficient radius neighbor search in three-dimensional point clouds". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 3625–3630. DOI: 10.1109/ICRA.2015.7139702.

[3] Shilpa Dang. "Performance Evaluation of Clustering Algorithm Using Different Datasets". In: *IJARCSMS* 3 (Jan. 2015), pp. 167–173.

[4] Micheline Kamber Jiawei Han and Jian Pei. "Data Mining: Concepts and Techniques (3rd ed.) Chapter 10. Cluster Analysis: Basic Concepts and Methods". In: (2011).

[5] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[6] Studytonight. *Introduction To Binary Trees*. URL: https://www.studytonight.com/data-structures/introduction-to-binary-trees (visited on 12/01/2020).

[7] Vu Huu Tiep. *K-means Clustering*. URL: https://machinelearningcoban.com/2017/01/01/kmeans/ (visited on 12/05/2020).

[8] Wikipedia. *DBSCAN*. URL: https://en.wikipedia.org/wiki/DBSCAN (visited on 12/01/2020).

[9] Wikipedia. *k-d tree*. URL: https://en.wikipedia.org/wiki/K-d_tree (visited on 12/01/2020).