Tìm hiểu về thuật toán K-Means

Nguyễn Văn Huy & Lê Duy An

Tháng 7, 2020

Mục lục

1 Giới thiệu

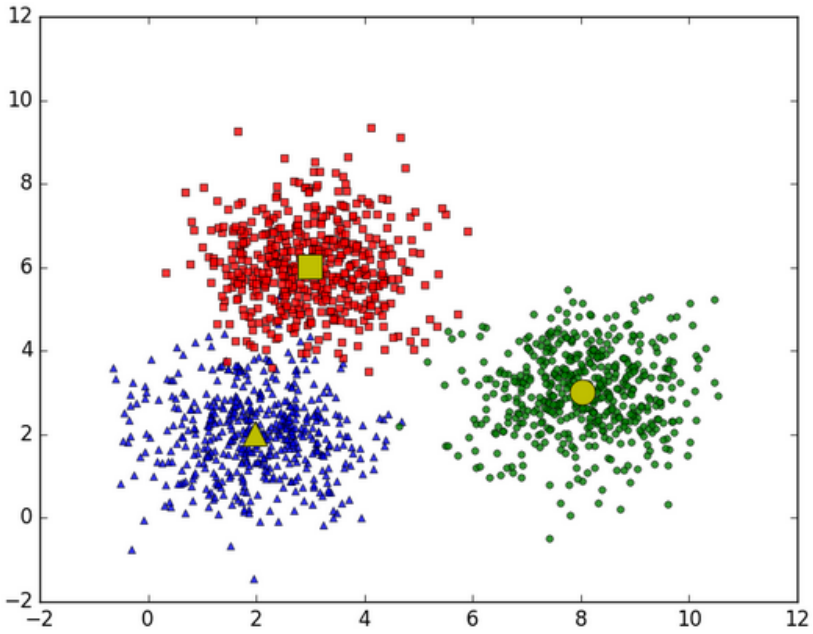
Phân cụm là kỹ thuật rất quan trọng trong việc khai thác dữ liệu. Phân cụm là các quy trình tìm cách nhóm các đối tượng đã cho vào các cụm (clusters), sao cho các đối tượng trong cùng 1 cụm thoả mãn các tính chất tương tự nhau và các đối tượng khác cụm thì không tương tự nhau.

Kỹ thuật phân cụm có thể được áp dụng trong đa dạng các lĩnh vực khác nhau như:

- Marketing: Xác định các nhóm khách hàng dựa trên sở thích, thói quen mua sắm,...
- Sinh học: Phận nhóm động vật và thực vật dựa vào các thuộc tính của chúng,...
- Quản lý thư viện: Theo dõi độc giả, sách, dự đoán nhu cầu của độc giả,...
- Giao thông: tìm nhóm đường có tốc độ giống nhau,...
- ...

Phương pháp k-means là một trong số những thuật toán phân cụm (clustering). Với đầu vào tập dữ liệu cần phân cụm và số cụm (cluster), đầu ra chúng ta sẽ được kết quả dữ liệu đã được phân về các cluster.

Trong thuật toán K-means clustering, chúng ta không biết nhãn (label) của từng điểm dữ liệu. Mục đích là làm thể nào để phân dữ liệu thành các cụm (cluster) khác nhau sao cho dữ liệu trong cùng một cụm có tính chất giống nhau.



Hình 1.1: Ví dụ về bài toán phân cụm dữ liệu

Xem xét ví dụ về các phân cụm dữ liệu như hình trên, ta quan sát được mỗi cụm có một điểm màu vàng gọi là điểm trung tâm (centroid), các điểm dữ liệu nằm gần điểm trung tâm nào nhất thì thuộc cùng một nhóm với điểm trung tâm đó. Như vậy, từ một bộ dữ liệu ban đầu, ta đã phân thành 3 cụm dữ liệu, mỗi cụm bao gồm các điểm dữ liệu có tính chất tương tự nhau.

2 Phân tích toán học

2.1 Giới thiệu dữ liệu của bài toán

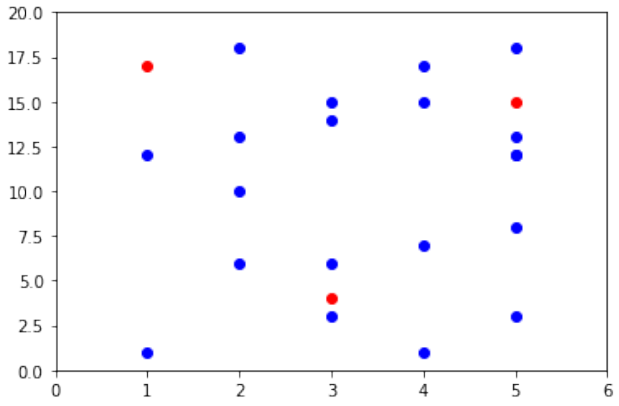
Với dữ liệu đầu vào của thuật toán là tập hợp các điểm dữ liệu $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N] \in \mathbb{R}^{d \times N}$ với $\mathbf{x}_i$ (có d phần tử) là một vector mang giá trị của mỗi điểm, N là số lượng các vector và số lượng K các nhóm cần phân loại từ các điểm dữ liệu đó với $K < N$ (vì số lượng nhóm cần phân loại không được lớn hơn số lượng các phần tử). Điều mà chúng ta cần phải làm là làm thế nào để xác định các điểm thuộc về nhóm nào một cách gắn kết nhất, ở đây để cho dễ gọi và tính toán thì chúng ta cho rằng K nhóm cần phần loại được gọi là nhóm $1, 2, 3, ..K$. Trong phần này chúng ta chỉ đề cập đến bài toán chỉ có một điểm dữ liệu thuộc vào một nhóm duy nhất.

Ban đầu chúng ta phải có được các điểm gốc ban đầu của các nhóm có thể chọn k điểm bất kì hoặc có thể lấy các điểm dữ liệu có sẵn trong tập dữ liệu ban đầu. Gọi các điểm gốc ban đầu là $\mathbf{m} = [\mathbf{m}_1, \mathbf{m}_2, \mathbf{m}_3, ..., \mathbf{m}_K]$ với mỗi điểm $\mathbf{m}_k$ cũng có có d các giá trị tương tự như các điểm dữ liệu $\mathbf{x}_i$. Dựa vào tập các điểm gốc $\mathbf{m}_k$ chúng ta phải xác định xem điểm $\mathbf{x}_i$ thuộc vào nhóm nào và gán nhãn cho các điểm đó bằng vector $\mathbf{y}$ có dạng:

$$\mathbf{y}_{ij} \in \{0, 1\}, \ \forall i, j; \ \sum_{j=1}^{K} = 1, \ \forall i$$

Trong đó $\mathbf{y}_{ij} = 0$ và $\mathbf{y}_{ik} = 1$, nghĩa là vector $\mathbf{y}$ có K giá trị và vị trí ở vị trí k có giá trị bằng 1 thì đồng nghĩa là vector $\mathbf{x}_i$ được gán vào nhóm k. Tập hợp các nhãn là $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3, ..., \mathbf{y}_N] \in \mathbb{R}^{K \times N}$.

Một ví dụ để hình dung rõ hơn về các tập dữ liệu. Ví dụ đề cập đến việc phân nhóm các điểm của tập $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, ..., \mathbf{x}_N]$ được biểu diễn trong Bảng 2 nằm trong không gian 2D gồm có số lượng các điểm là $N = 20$ được biểu diễn trong Hình 2.1 bằng các điểm màu xanh, vì trong không gian 2D nên $d = 2$ tương ứng là tọa độ x và y. Điều kiện của bài toán là các điểm đó thành ba cụm với ba điểm ban đầu là $\mathbf{m} = [(1; 17), (3; 4), (5; 15)]$ có $K = 3$ được biểu thị bằng các điểm màu đỏ trên Hình 2.1.



Hình 2.1: Hình các các dữ liệu đầu vào.

$\mathbf{X}$	x	y
$\mathbf{x}_1$	1	4
$\mathbf{x}_2$	5	12
$\mathbf{x}_3$	2	18
...	...	...
$\mathbf{x}_N$	5	13

Bảng 1: Bảng biểu diễn một vài giá trị của các điểm trong tập $\mathbf{X}$

$\mathbf{Y}$	1	2	3
$\mathbf{y}_1$	0	0	1
$\mathbf{y}_2$	1	0	0
$\mathbf{y}_3$	0	1	0
...	...	...	...
$\mathbf{y}_N$	0	0	1

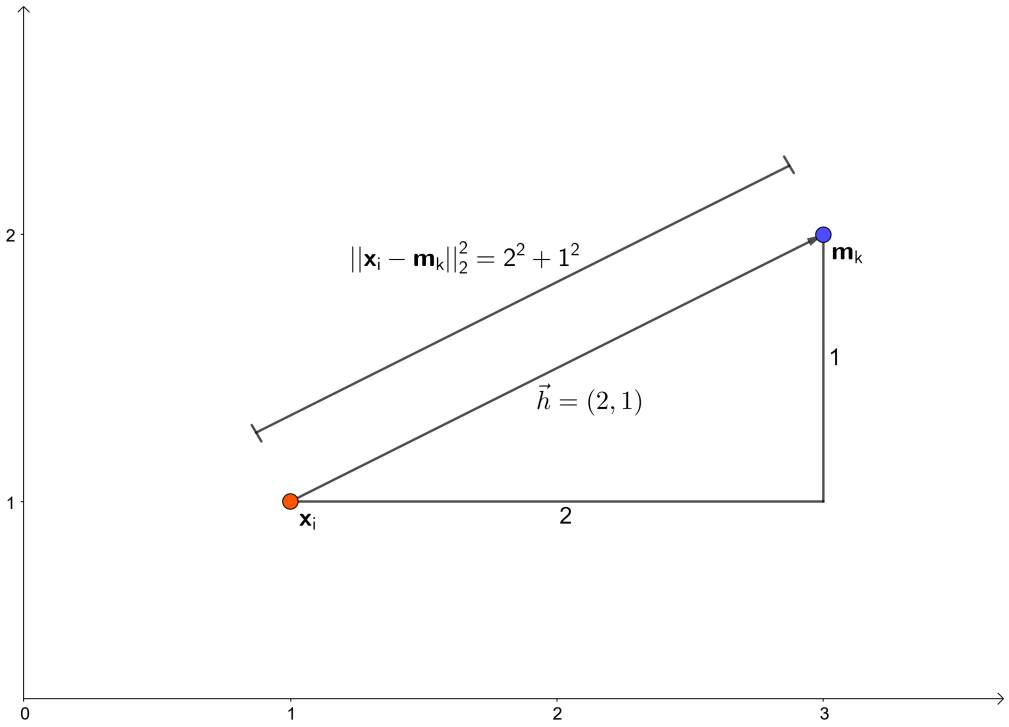
Bảng 2: Bảng biểu diễn mội vài nhãn $\mathbf{y}_i$ trong tập $\mathbf{Y}$

2.2 Hàm mất mát và bài toán tối ưu

Như đã đề cập phía trên $\mathbf{m}_k \in \mathbb{R}^d$ là điểm gốc ban đầu của các nhóm cần phân loại, từ các điểm gốc ban đầu để xác định ra các vector $\mathbf{x}_i$ cần được gom vào nhóm nào. Tức là một nhóm có quan hệ chặt chẽ với một nhóm nào đó được đại diện bằng điểm tâm của mỗi nhóm là $\mathbf{m}_k$. Ở đây để xét 2 điểm được xem là chặt chẽ (nhất) với nhau thì chúng ta sẽ xét xem khoảng cách Euclid của hai điểm đó. Nếu khoảng cách điểm $\mathbf{x}_i$ được gọi là liên kết chặt chẽ với điểm $\mathbf{m}_k$ khi và chỉ khi không có một điểm khác k nào có giá trị Euclid gần điểm k. Khi đã xác định một điểm có liên kết chặt chẽ nhất đối với một đại diện của một nhóm, thì tất nhiên điểm đó sẽ được ưu tiên gom nhóm vào nhóm đó. Do vậy, cũng để xét độ lỗi phân lớp của một điểm thì ta sẽ xét đến khoảng cách Euclid của điểm đó và điểm đại diện $\mathbf{m}_k$ của nhóm đó, được biểu diễn theo công thức

$$l = \sqrt{\sum_{j=1}^{d} (\mathbf{x}_{ij} - \mathbf{m}_{ij})^2}$$

Nhưng để tiện tính toán và tiết kiệm thời gian thực thi vì khi thực hiện dấu căn ở trên tốn rất nhiều tài nguyên để thực hiện, do đó đề xuất bỏ dấu căn trong công thức bằng cách bình phương khoảng cách Euclid $||\mathbf{x}_i - \mathbf{m}_k||_2^2$ nhưng vẫn không thay đổi tính chất của bài toán. Mặc dù xem là bình phương khoảng cách Euclid, nhưng ở đây thực nhất là không thực hiện bình phương mà chỉ đơn giản là bỏ dấu căn. Công thức độ lỗi được biểu diễn bằng hình dưới đây trong không gian 2𝔻.



Hình 2.2: Độ lỗi của một điểm với trung tâm nhóm.

Theo như đề xuất ở trên thì $\mathbf{y}_i$ biểu thị cho vùng k mà vector $\mathbf{x}_i$ được gom nhóm vào, do đó $\mathbf{y}_{ij} = 0$,

$\mathbf{y}_{ik} = 1\ \forall i \neq k$. Công thức trên có thể được viết thành

$$||\mathbf{x}_i - \mathbf{m}_k||_2^2 = y_{ik}||\mathbf{x}_i - \mathbf{m}_k||_2^2 = \sum_{j=1}^{K} y_{ij}||\mathbf{x}_i - \mathbf{m}_j||_2^2$$

Như vậy, sai số trung bình cho toàn bộ dữ liệu hay còn gọi là độ lỗi trung bình sẽ là tổng độ lỗi các vector $\mathbf{x}_i$ với điểm đại diện nhóm của chính vector đó và vì độ lỗi trung bình nên sẽ chia cho tổng số các vector ban đầu là N. Được biểu diễn bằng công thức dưới đây:

$$\mathcal{L}(\mathbf{Y}, \mathbf{M}) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij}||\mathbf{x}_i - \mathbf{m}_j||_2^2$$

Ta gọi hàm $\mathcal{L}(\mathbf{Y}, \mathbf{M})$ là hàm mất mát và giá trị của hàm này là độ lỗi trung bình của thuật toán K-means. Nhưng công việc của bài toán không thể dừng ở đây, chúng ta phải xác định rằng liệu các vector $\mathbf{x}_i$ được gom nhóm có liên kết chặt chẽ nhất có thể hay chưa. Điều này tương đương với việc giảm giá trị hàm mất mát, cần phải tìm ra cách để làm giá trị hàm mất mát $\mathcal{L}(\mathbf{Y}, \mathbf{M})$ về nhỏ nhất có thể. Do đó ta xác định bài toán cần tối ưu là tìm điểm tìm tập $\mathbf{Y}$ và $\mathbf{M}$ phù hợp nhất cho bài toán và được thể hiện bằng công thức sau

$$\mathbf{Y}, \mathbf{M} = \frac{1}{N} \underset{\mathbf{Y},\mathbf{M}}{argmin} \sum_{j=1}^{K} y_{ij}||\mathbf{x}_i - \mathbf{m}_j||_2^2$$

Với y_{ij} thõa mãn điều kiện

$$y_{ij} \in \{0, 1\},\ \forall i, j :\ \sum_{j=1}^{K} y_{ij} = 1,\ \forall i$$

2.3 Thuật toán tối ưu hàm mất mát

Như được nêu ở trên, sau khi đã có hàm sai số trung bình cho toàn dữ liệu, điều chúng ta cần thực hiện là làm sao để giá trị đó là nhỏ nhất có thể. Chúng ta sẽ tập trung vào phương pháp tìm $\mathbf{Y}$ và $\mathbf{M}$ bởi vì đơn giản là chỉ có hai tập đó là chúng ta có thể thay đổi được và nó liên quan trực tiếp đến hàm mất mát$\mathcal{L}(\mathbf{Y}, \mathbf{M})$. Nhưng vì không thể dùng một công thức trực tiếp hay cụ thể nào thể thực hiện điều đó cả. Nên do đó, việc cần làm là lần lượt thử các trường hợp đến khi chúng ta đạt được giá trị hàm mất mát không thay đổi hoặc chấp nhận được. Để làm được điều đó chúng ta sẽ lần lượt thực hiện xen kẽ hai phép tính để tìm ra $\mathbf{Y}$ và $\mathbf{M}$ thõa mãn. Hai phép toán cần được thực hiện là cố định $\mathbf{M}$ tìm $\mathbf{Y}$ và cố định $\mathbf{Y}$ tìm $\mathbf{M}$. Vì dữ liệu ban đầu đề bài có là tập các điểm đại cho mỗi nhóm $\mathbf{M}$ do đó chúng ta thực hiện phép tính cố định $\mathbf{M}$ tìm $\mathbf{Y}$ trước.

- Cố định $\mathbf{M}$ tìm $\mathbf{Y}$

  Dữ liệu đầu vào đã có tập hợp các điểm trung tâm $\mathbf{M}$ của mỗi nhóm hoặc chúng ta có thể tự chọn K điểm trong tập $\mathbf{X}$ để thay thế. Chúng ta phải tiến hành gom nhóm các điểm dữ liệu vào mỗi nhóm phù hợp với nó, tức là xác định label y_i của mỗi vector $\mathbf{x}_i$ để cho độ lỗi với điểm $\mathbf{m}_k$ là nhỏ nhất, được biểu diễn theo công thức dưới đây

  $$\mathbf{y}_i = \underset{\mathbf{y}_i}{argmin} \sum_{j=1}^{K} y_{ij}||\mathbf{x}_i - \mathbf{m}_j||_2^2$$

  Với y_{ij} thõa mãn điều kiện

  $$y_{ij} \in \{0, 1\},\ \forall i, j :\ \sum_{j=1}^{K} y_{ij} = 1,\ \forall i$$

  Đối với vị trí k mà $\mathbf{y}_{ik} = 1$ thì đó là vị trí nhóm mà $\mathbf{x}_i$ được gom vào nhóm đó. Hay còn có thể nói rằng khoảng cách của vector $\mathbf{x}_i$ với các điểm trong tập $\mathbf{M}$ thì chỉ có mỗi vector $\mathbf{m}_k$ có khoảng cách(độ lỗi) nhỏ nhất. Mặc dù trong công thức có dấu sigma($\sum$) nhưng thực chất chỉ có một giá trị độ lỗi được tính, còn lại vì giá trị $\mathbf{y}_{ij} = 0$ nên giá trị trong sigma luôn luôn bằng 0 khi $j \neq k$ do đó chỉ cần tính giá trị khi $j = k$ là được.

- Cố định $\mathbf{Y}$ tìm $\mathbf{M}$

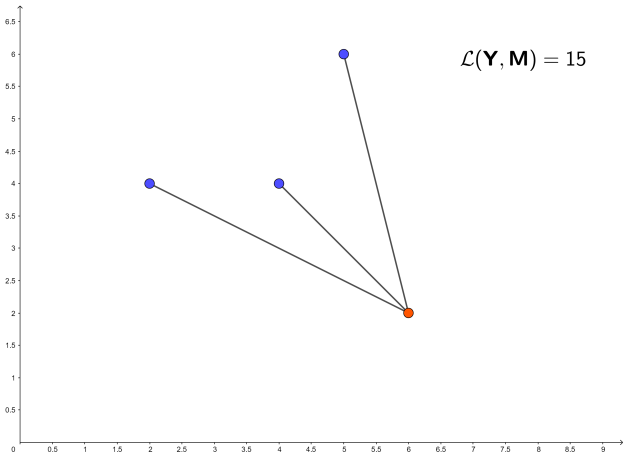
  Sau khi qua phép toán thứ nhất, chúng ta đã có được tập label $\mathbf{Y}$. Nhưng điều quan trọng là độ lỗi trung bình $\mathcal{L}(\mathbf{Y}, \mathbf{M})$ phải thõa mãn điều kiện nhỏ nhất. Để làm cho độ lỗi đó nhỏ hơn thì phải xác định lại điểm đại diện $\mathbf{m}_k$c ho cụm đó. Do đó, cần phải tiếp tục chọn lại các điểm trung tâm của mỗi cụm, làm cho độ lỗi trung bình nhỏ dần. Cũng gần giống với công thức tìm label $\mathbf{Y}$, ta có công thức sau đây.

$$\mathbf{m}_j = \underset{\mathbf{m}_j}{argmin} \sum_{i=1}^{K} y_{ij} ||\mathbf{x}_i - \mathbf{m}_j||_2^2$$

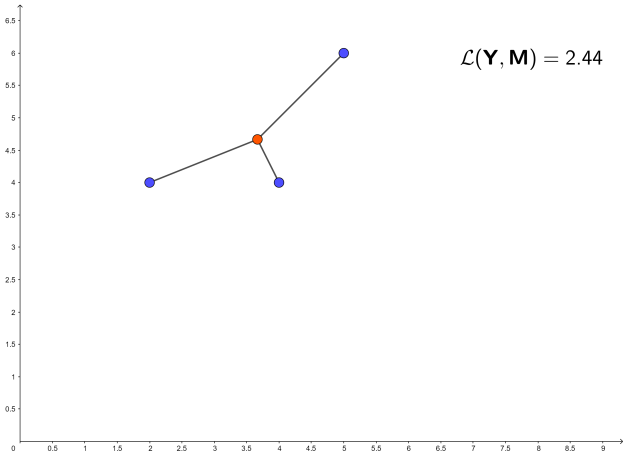
  Vì chúng ta cần tìm $\mathbf{m}_j$ để cho phương trình trên đạt giá trị nhỏ nhất (cực tiểu), mà hàm ở trên là hàm liên tục và có đạo hàm xác định tại mọi điểm. Do đó điều chúng ta nghỉ đến đầu tiên là phương pháp đạo hàm phương trình đó để tìm ra các điểm cực trị, ở đây là cực tiểu. Ta cần giải phương trình.

$$\nabla_{\mathbf{m}_j} l(\mathbf{m}_j) = \frac{2}{N} \sum_{i=1}^{N} y_{ij}(\mathbf{m}_j - \mathbf{x}_i) = 0 \Leftrightarrow \mathbf{m}_j \sum_{i=1}^{N} y_{ij} = \sum_{i=1}^{N} y_{ij}\mathbf{x}_i \Leftrightarrow \mathbf{m}_j = \frac{\sum_{i=1}^{N} y_{ij}\mathbf{x}_i}{\sum_{i=1}^{N} y_{ij}}$$

  Từ kết quả trên cho thấy, mẫu số là số lượng các điểm được phân lớp vào nhóm j, còn tử số chính là tổng các vector $\mathbf{x}_i$ được phân lớp và nhóm j đó. Nói cách khác, $\mathbf{m}_j$ là trung bình cộng (mean) của các điểm trong nhóm j. Hai hình bên dưới sẽ cho chúng ta dễ hình dung hơn



Hình 2.3: Độ lỗi trung bình của một cụm chưa thực hiện tìm $\mathbf{M}$ theo $\mathbf{Y}$.



Hình 2.4: Độ lỗi trung bình của một cụm sau khi thực hiện tìm $\mathbf{M}$ theo $\mathbf{Y}$.

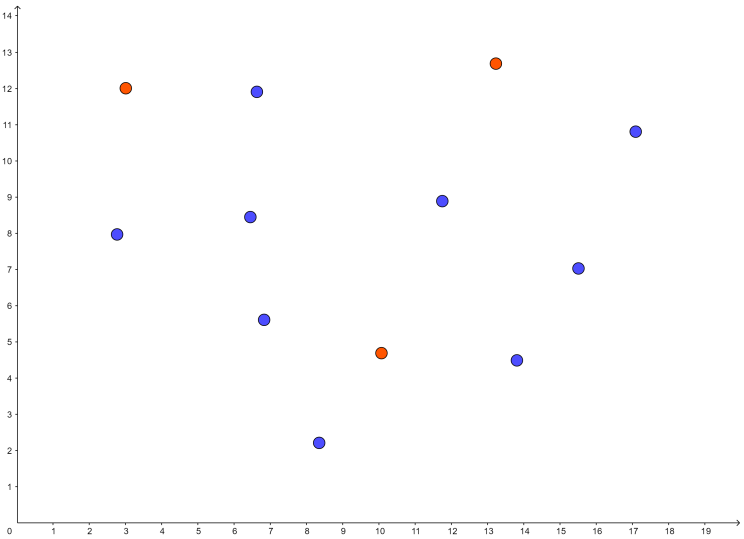
2.4 Tóm tắt thuật toán

- Đầu vào: Ma trận dữ liệu $\mathbf{X} \in \mathbb{R}^{d \times N}$ và số lượng cluster cần tìm $K < N$.
- Đầu ra: Ma trận K nhóm $\mathbf{M} \in \mathbb{R}^{d \times K}$ và ma trận label $\mathbf{Y} \in \mathbb{R}^{K \times N}$
- Thuật toán
  - Bước 1: Chuẩn bị tập các điểm ban đầu $\mathbf{M}$ hoặc có thể chọn K điểm bất kỳ trong tập dữ liệu ban đầu làm tâm của các nhóm ban đầu
  - Bước 2: Phân mỗi điểm dữ liệu vào nhóm có tâm $\mathbf{m}_j$ gần nó nhất. (Sử dụng phép tính cố định $\mathbf{M}$ tìm $\mathbf{Y}$)
  - Bước 3: Kiểm tra điều kiện bài toán đã thõa hay chưa. Nếu việc phân nhóm dữ liệu vào từng nhóm ở bước 2 không thay đổi so với vòng lặp trước nó thì ta dừng thuật toán.
  - Bước 4: Cập nhật tâm $\mathbf{m}_j$ của từng nhóm bằng cách lấy trung bình cộng của tất các các điểm dữ liệu đã được gán vào nhóm đó sau bước 2. (Sử dụng phép tính cố định $\mathbf{Y}$ tìm $\mathbf{M}$)
  - Bước 5: Quay trở lại bước 2.

Thuật toán này được đảm bảo sẽ hội tụ sau một số hữu hạn vòng lặp. Thật vậy, vì hàm mất mát là một số dương và sau mỗi bước 2 hoặc 3, giá trị của hàm mất mát bị giảm đi. Vậy, dãy số biểu diễn giá trị của hàm mất mát sau mỗi bước là một đại lượng không tăng và bị chặn trên, điều này chỉ ra rằng dãy số này phải hội tụ. Để ý thêm nữa, số lượng cách phân nhóm cho toàn bộ dữ liệu là hữu hạn (khi số cụm K là cố định) nên đến một lúc nào đó, hàm mất mát sẽ không thể thay đổi, và chúng ta có thể dừng thuật toán tại đây.

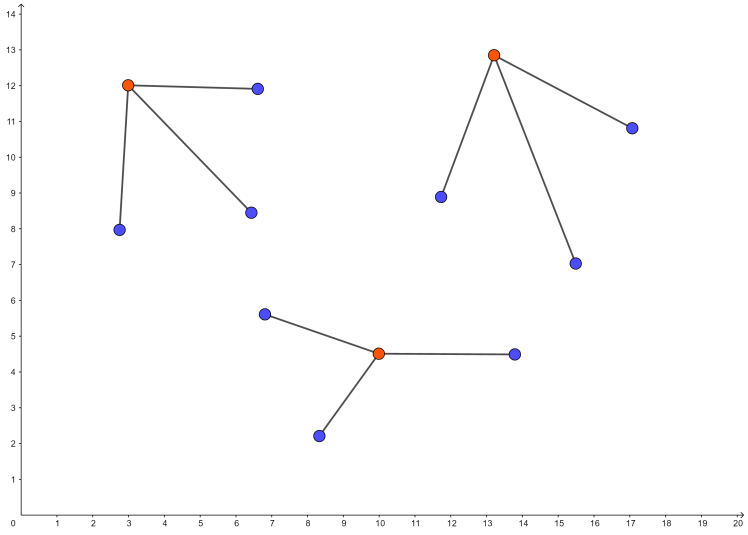
Nếu tồn tại một cụm không chứa điểm nào thì sẽ không thực hiện được phép toán trên vì mẫu số bằng 0. Vì vậy, K điểm bất kỳ trong tập dữ liệu đầu vào được chọn làm các trung tâm ban đầu ở Bước 1 phải đảm bảo mỗi cụm có ít nhất một điểm thành phần. Do đó có hai cách giải quyết vấn đề này, cách thứ nhất là bỏ đi điểm trung tâm đó và giảm K đi một hoặc là thay điểm trung tâm của nhóm đó bằng một điểm nằm trong tập dữ liệu đầu vào $\mathbf{X}$.

Chúng ta có thể hình dung thuật toán bằng các hình dưới đây với $N = 9$, $K = 3$ với tập $\mathbf{M}$ là các điểm màu đỏ và tập $\mathbf{X}$ là các điểm màu xanh.



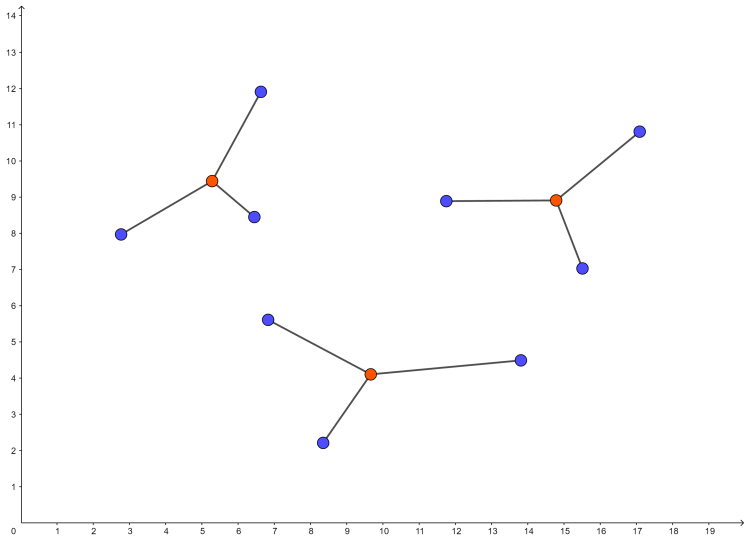
Hình 2.5: Các dữ liệu đầu vào của thuật toán K-means

Sau đó tiến hành tính độ lỗi của các vector $\mathbf{x}_i$ và các điểm $\mathbf{m}_k$ để tìm cụm phù hợp.



Hình 2.6: Các dữ liệu đầu vào đã được gom vào mỗi 3 cụm

Tiến hành tìm lại các điểm $\mathbf{M}$ để cho độ lỗi trung bình là nhỏ nhất bằng phương pháp trung bình các điểm trong cùng một cụm (cố định $\mathbf{Y}$ tìm $\mathbf{M}$)

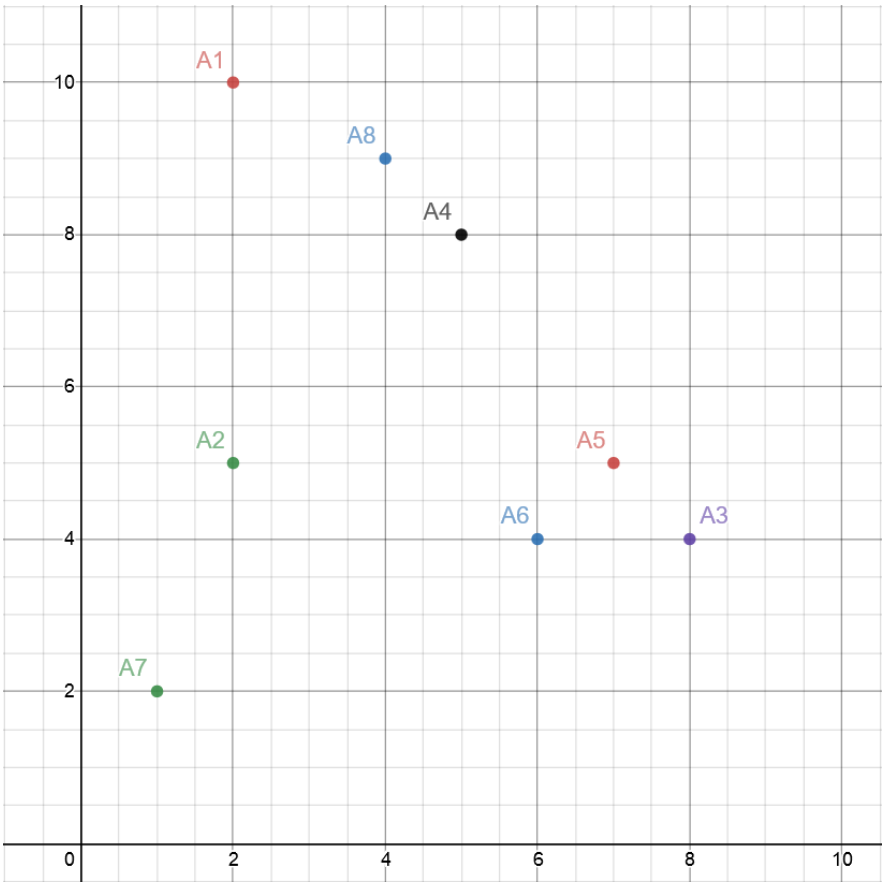


Hình 2.7: Tìm lại các điểm $\mathbf{M}$

Vì bài toán trên là bài toán mẫu nên khá đơn giản để tối ưu hóa độ lỗi trung bình. Đối với các bài toán lớn cần phải thiết lập thêm điều kiện để đừng các bài toán tránh trường hợp lặp vô hạn. Ngoài ra cần phải chú ý thêm số K cụm và các điểm trung tâm của các cụm $\mathbf{M}$, các vấn đề đó sẽ được trình bài sau.

2.5 Ví dụ phân cụm dữ liệu

Ta xét mẫu dữ liệu đầu vào gồm 8 điểm dưới đây, với tọa độ các điểm được cho như sau:



Hình 2.8: Ví dụ mẫu dữ liệu đầu vào

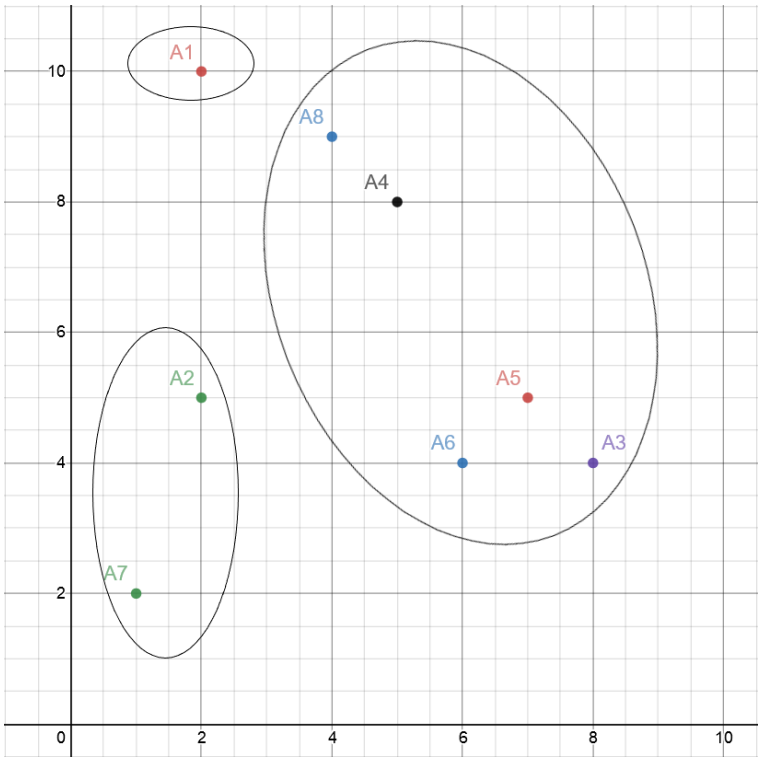
Điểm	A1	A2	A3	A4	A5	A6	A7	A8
Tọa độ	(2,10)	(2,5)	(8,4)	(5,8)	(7,5)	(6,4)	(1,2)	(4,9)

Bảng 3: Tọa độ các điểm trong mẫu dữ liệu

Giả sử ta chọn ba điểm điểm A1, A4 và A7 làm trung tâm của cụm, ta tính được khoảng cách của từng điểm trong mẫu đến hai điểm centroid.

	A2	A3	A5	A6	A8
A1	5	8.49	7.07	7.21	2.24
A4	4,24	**5**	**3.61**	**4.12**	**1.41**
A7	**3.16**	7.28	6.71	5.39	7.62

Khoảng cách từ một điểm đến đến centroid nào ngắn nhất được in đậm sẽ thuộc cụm của centroid đó.

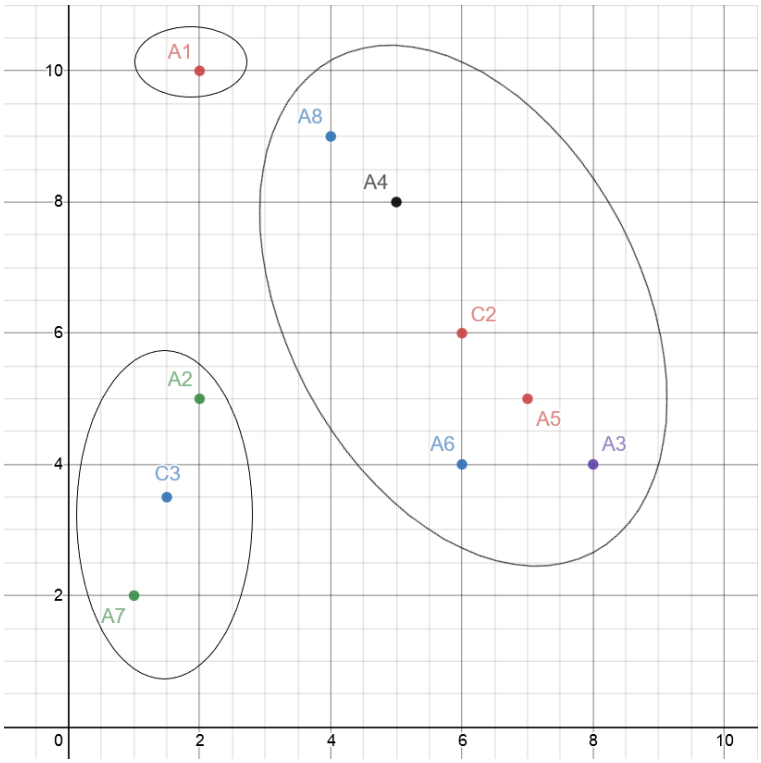


Hình 2.9: Phân cụm dữ liệu lần 1

Như vậy, sau lần phân cụm thứ nhất, ta có các cụm $C_1 = \{A_1\}, C_2 = \{A_4; A_8; A_6; A_5; A_3\}, C_3 = \{A2; A_7\}$.

Ta tính được tọa độ điểm centroid của ba cụm dữ liệu sau khi phân cụm:

$C_1 = (2, 10)$; $C_2 = \left(\frac{8+5+7+6+4}{5}, \frac{4+8+5+4+9}{5}\right) = (6, 6), C_3 = \left(\frac{2+1}{2}, \frac{5+2}{2}\right) = (1.5, 3.5)$

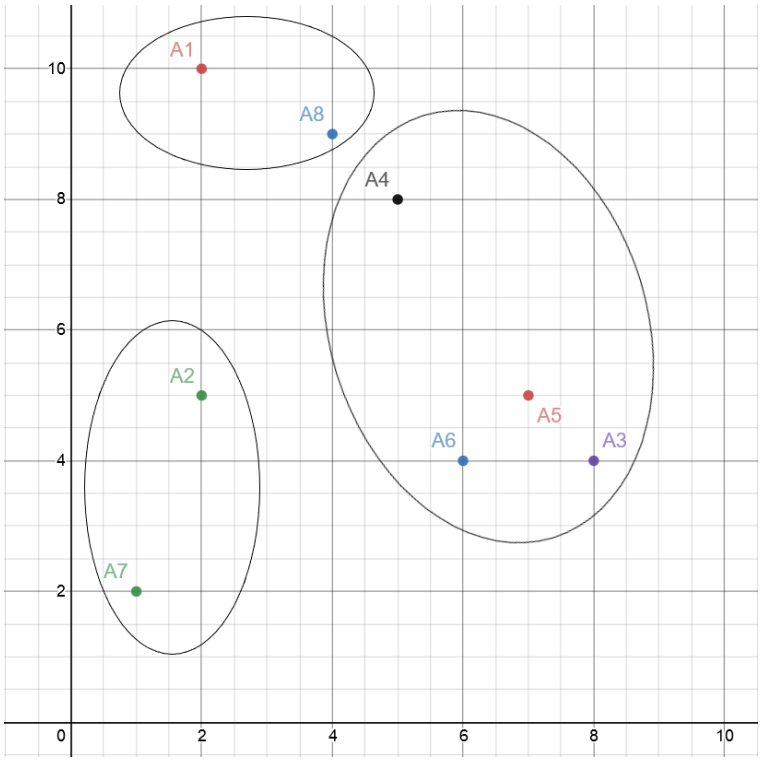


Hình 2.10: Tọa độ các điểm centroid sau khi chạy lần 1

Vì tọa độ của ba điểm centroid này khác với toạ độ ba điểm đã đưa ra ban đầu, nên ta tiếp tục tính khoảng cách của các điểm dữ liệu đến centroid mới rồi tiến hành phân nhóm như lần 1.

	A1	A2	A3	A4	A5	A6	A7	A8
C1	**0**	5	8.49	3.61	7.07	7.21	8.06	**2.24**
C2	5.66	4.12	**2.83**	**2.24**	**1.41**	**2.00**	6.40	3.61
C3	6.52	**1.58**	6.52	5.70	5.70	4.53	**1.58**	6.04

Tiến hành phân cụm lần 2:

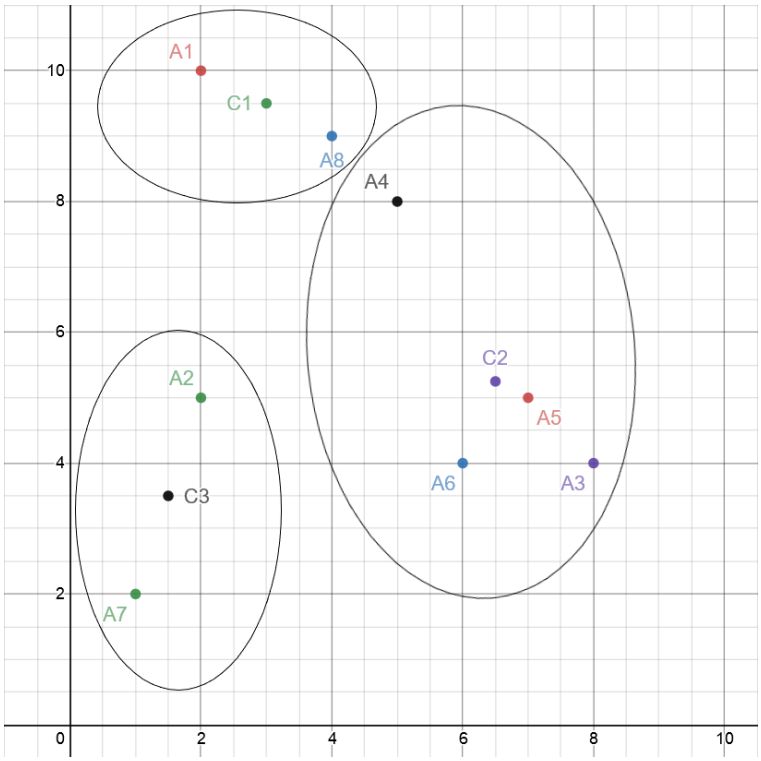


Hình 2.11: Phân cụm dữ liệu lần 2

Như vậy, sau lần phân cụm thứ hai, ta có các cụm $C_1 = \{A_1; A_8\}, C_2 = \{A_4; A_6; A_5; A_3\}, C_3 = \{A2; A_7\}$.

Ta tính được tọa độ điểm centroid của ba cụm dữ liệu sau khi phân cụm:

$C_1 = \left(\frac{2+4}{2}, \frac{10+9}{2}\right) = (3, 9.5)$; $C_2 = \left(\frac{8+5+7+6}{4}, \frac{4+8+5+4}{4}\right) = (6.5, 5.25), C_3 = \left(\frac{2+1}{2}, \frac{5+2}{2}\right) = (1.5, 3.5)$

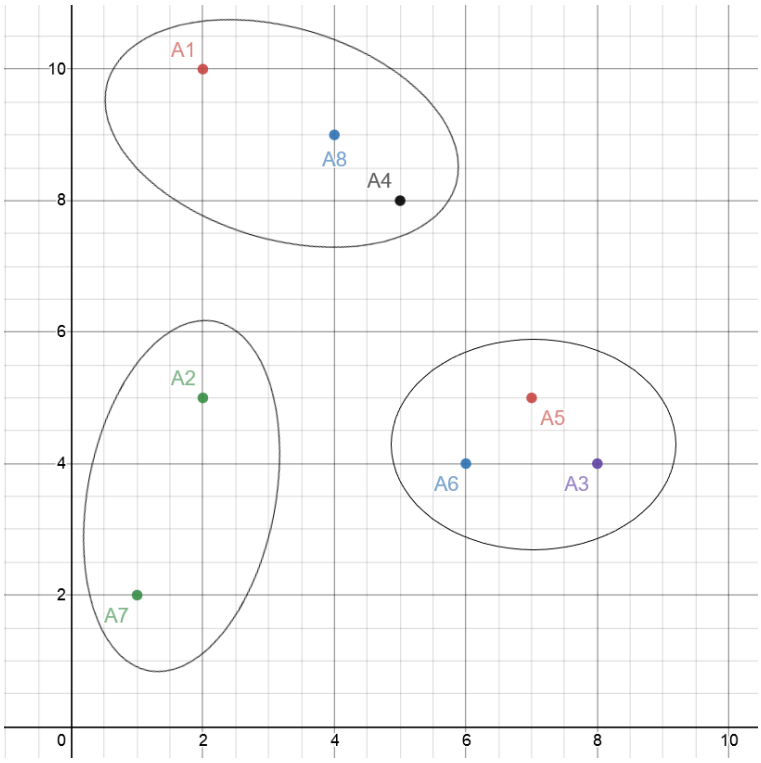


Hình 2.12: Tọa độ các điểm centroid sau khi chạy lần 2

Vì tọa độ của ba điểm centroid này khác với toạ độ ba điểm centroid ở lần phân cụm thứ nhất, nên ta tiếp tục tính khoảng cách của các điểm dữ liệu đến centroid mới rồi tiến hành phân nhóm.

	A1	A2	A3	A4	A5	A6	A7	A8
C1	**1.12**	4.61	7.43	**2.50**	6.02	6.26	7.76	**1.12**
C2	6.37	4.26	**2.15**	3.02	**0.79**	**1.27**	6.17	4.37
C3	6.52	**1.58**	6.52	5.70	5.70	4.53	**1.58**	6.04

Tiến hành phân cụm lần 3:

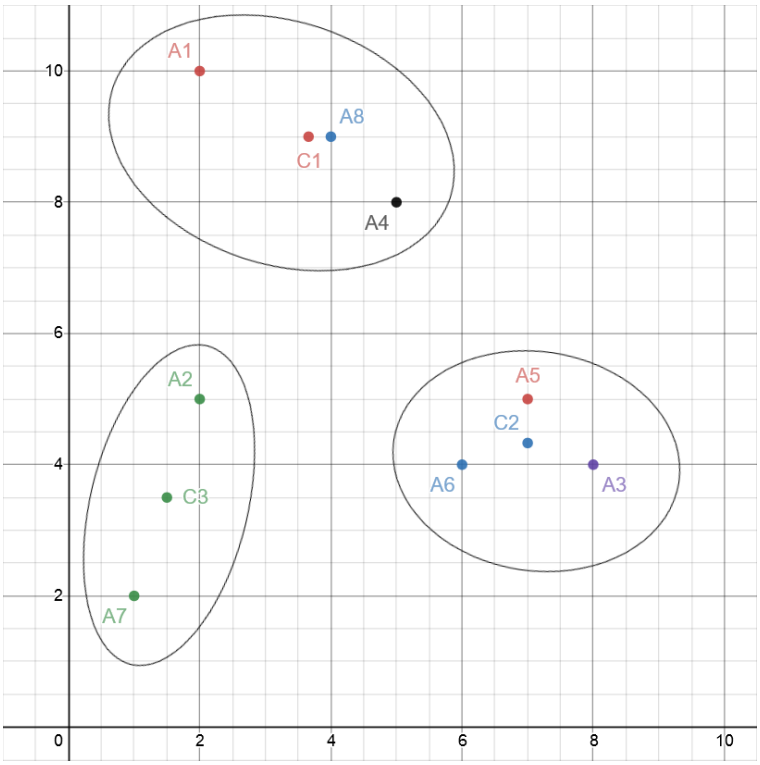


Hình 2.13: Phân cụm dữ liệu lần 3

Như vậy, sau lần phân cụm thứ ba, ta có các cụm $C_1 = \{A_1; A_8; A_4\}, C_2 = \{A_6; A_5; A_3\}, C_3 = \{A2; A_7\}$.

Ta tính được tọa độ điểm centroid của ba cụm dữ liệu sau khi phân cụm:

$C_1 = \left(\frac{2+4+5}{3}, \frac{10+9+8}{3}\right) = (3.66, 9)$; $C_2 = \left(\frac{8+5+7+6}{4}, \frac{4+8+5+4}{4}\right) = (7, 4.33), C_3 = \left(\frac{2+1}{2}, \frac{5+2}{2}\right) = (1.5, 3.5)$



Hình 2.14: Tọa độ các điểm centroid sau khi chạy lần 3

Vì tọa độ của ba điểm centroid này khác với toạ độ ba điểm centroid ở lần phân cụm thứ nhất, nên ta tiếp tục tính khoảng cách của các điểm dữ liệu đến centroid mới rồi tiến hành phân nhóm.

	A1	A2	A3	A4	A5	A6	A7	A8
C1	**1.94**	4.33	6.62	**1.67**	5.21	5.52	7.49	**0.34**
C2	7.56	5.04	**1.05**	4.18	**0.67**	**1.05**	6.44	5.55
C3	6.52	**1.58**	6.52	5.70	5.70	4.53	**1.58**	6.04

Ta nhận thấy, thành phần điểm dữ liệu trong các cluster không thay đổi so với lần chạy thứ 3 ở trên, dẫn đến toạ độ các điểm centroid không thay đổi. Do đó thuật toán kết thúc tại đây với ba cụm dữ liệu có các điểm dữ liệu như trên.

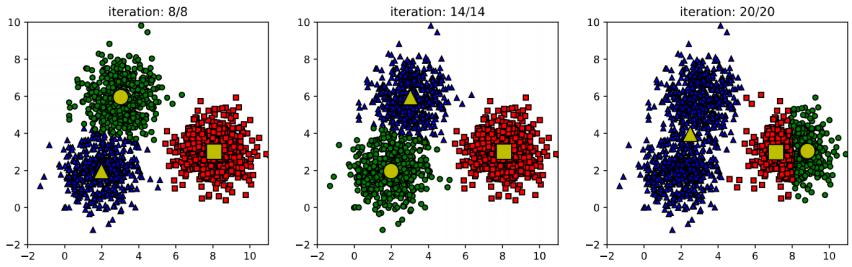
3 Ưu và nhược điểm

3.1 Ưu điểm

- Thuật toán đơn giản, hiệu quả với độ phức tạo là $O(tKn)$, t, k « n, với:
  - t: số lần lặp
  - K: số cluster
  - n: số mẫu
- Sử dụng được với bộ số liệu lớn

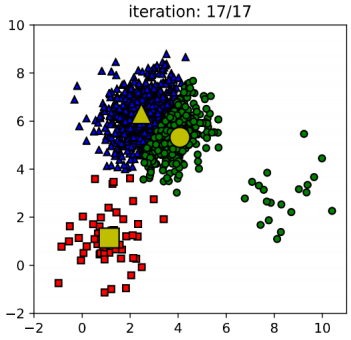
3.2 Nhược điểm

- Cần phải xác định trước số lượng cluster. Trong thực tế, cần phải sử dụng thêm một số biện pháp giúp xác định giá trị K, chẳng hạn như phương pháp Elbow.
- Thuật toán KMeans không đảm bảo tìm được nghiệm tối ưu toàn cục nên nghiệm cuối cùng phụ thuộc rất nhiều vào các centroid ban đầu. Hình 3.1 thể hiện các kết quả khác nhau khi các centroid khởi tạo khác nhau. Ngoài ra, việc chọn các centroid cũng ảnh hưởng đến hiệu suất làm việc của thuật toán. Với cùng một kết quả tối ưu như nhau, số lần chạy ở hình 2 lớn gần gấp đôi so với hình 1.



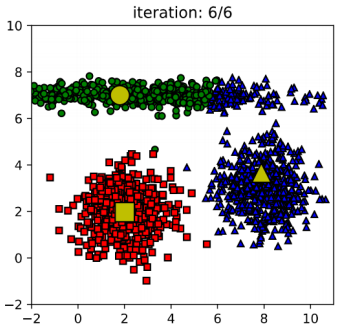
Hình 3.1: Các nghiệm khác nhau do khởi tạo ban đầu khác nhau

- Các cluster cần phải có số lượng điểm gần bằng nhau. Ở hình 3.2 minh hoạ kết quả thuật toán KMeans với bộ dữ liệu có các cluster có số điểm chênh lệch. Trong trường hợp này, nhiều điểm đáng lẽ thuộc vào cụm xanh lam đã bị nhầm vào cụm xanh lục.



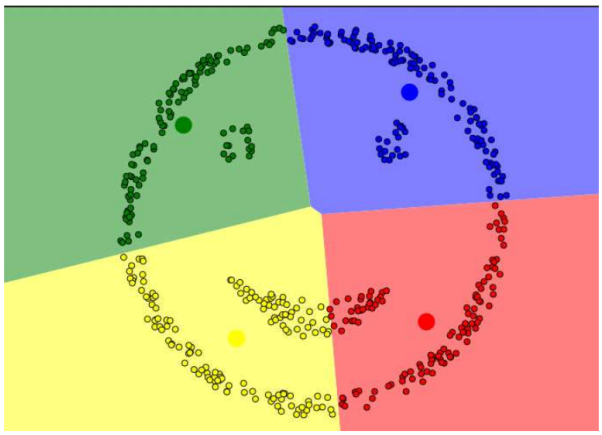
Hình 3.2: Các nghiệm trong cluster này bị nhầm vào cluster khác

- Các cluster cần có dạng hình tròn (cầu), nếu không thì KMeans sẽ hoạt động không hiệu quả. Lý do chính là vì Kmeans quyết định cluster của một điểm dữ liệu thông qua khoảng cách của nó đến centroid.



Hình 3.3: Các nghiệm trong cluster này bị nhầm vào cluster khác

- Centroid có thể bị xê dịch bởi các ngoại lệ, hoặc các ngoại lệ có thể có cụm riêng thay vì bị bỏ qua.
- Cho kết quả sai khi một cluster này bị bao bọc bằng một cluster khác. Hình 3.3 là một minh hoạ điển hình cho việc KMeans không thể phân cụm dữ liệu. Một cách tự nhiên, chúng ta chia hình mặt thành 4 cụm: mắt trái, mắt phải, miệng, hình bao quanh mặt. Nhưng vì các bộ phận mắt, miệng nằm bên trong mặt nên KMeans phân cụm không chính xác.



Hình 3.4: KMeans phân cụm hình mặt không chính xác

4 Cách tìm K cụm tối ưu nhất

Xác định số lượng cụm tối ưu trong một tập dữ liệu là vấn đề cơ bản trong phân cụm Kmeans, yêu cầu người dùng chỉ định số lượng cụm k được tạo. Ý tưởng đằng sau Kmeans bao gồm xác định các cụm k sao cho tổng biến thể trong cụm là tối thiểu. Đây được xem là một nhược điểm của thuật toán này. Phần dưới đây trình bày một vài phương pháp giúp xác định số cụm k hợp lý nhất.

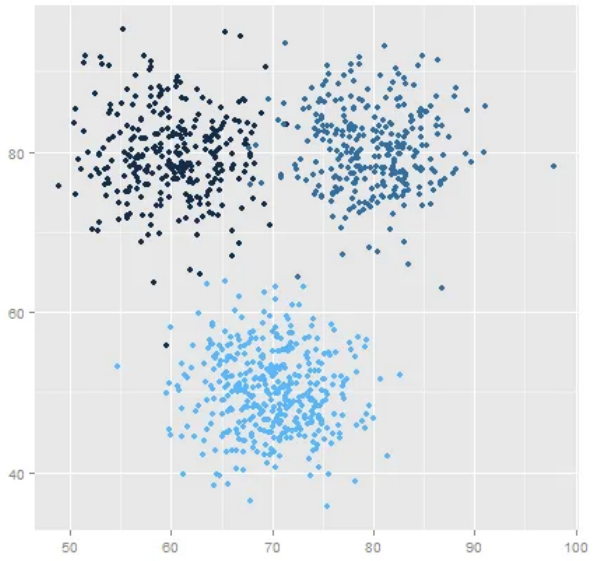
4.1 Thuật toán Elbow

Tư tưởng chính của phương pháp phân cụm phân hoạch (như KMeans) là định nghĩa 1 cụm sao cho tổng bình phương khoảng cách của tất cả các điểm đến đến trung tâm cụm là nhỏ nhất, tham số này là WSS (Within-cluster Sum of Square). Elbow method chọn số cụm k sao cho khi thêm vào một cụm khác thì không làm cho WSS thay đổi nhiều.

Quy trình triển khai thuật toán Elbow:

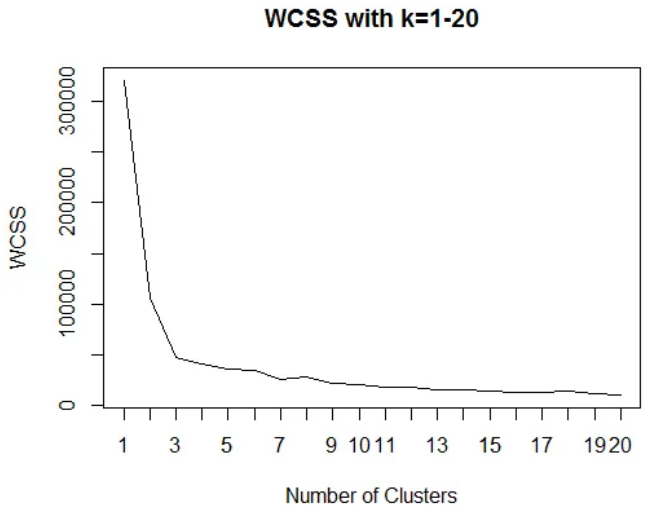
- Thực hiện phân cụm với số cụm thay đổi
- Với mỗi giá trị k, tính giá trị WSS
- Vẽ đường cong Elbow theo các giá trị k
- Dựa vào đường cong Elbow chọn số k thích hợp, là vị trí ở khúc cua

Xét ví dụ mẫu dữ liệu sau:



Hình 4.1: Mẫu dữ liệu

Tính toán giá trị WSS tương ứng với K từ 1 đến 20, ta thu được biểu đồ sau:



Hình 4.2: WSS tương ứng với k từ 1 đến 20

Ta chọn k = 3 làm số cụm cho mẫu dữ liệu trên.

4.2 Thuật toán Average Silhouette

Average silhouette dùng để đo chất lượng của một cụm. Giá trị Silhouette s(i) cho mỗi điểm dữ liệu i được xác định như sau:

$$s(i) = \frac{b(i)-a(i)}{max\{a(i),b(i)\}}(|C_i| > 1)$$

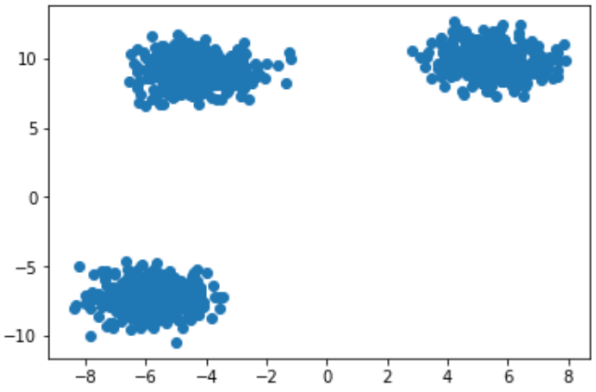
Trong đó:

- a(i) khoảng cách trung bình từ điểm i đến các điểm khác trong cụm. $a(i) = \frac{1}{|C_i|-1} \sum_{j \in C_i, i \neq j} d(i,j)$
- b(i) là khoảng cách trung bình giữa các cụm. $b(i) = \min_{i \neq j} \frac{1}{|C_j|} \sum_{j \in C_j} d(i,j)$

Với s(i) = 0 với những cụm có chỉ có 1 phần tử.
Quy trình triển khai thuật toán average silhouette:

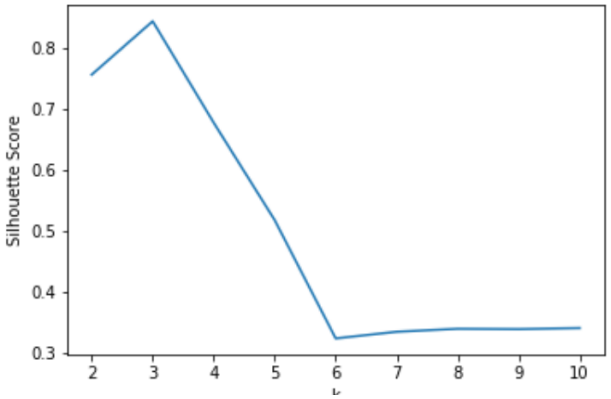
- Thực hiện phân cụm với số cụm thay đổi
- Với mỗi giá trị k, tính giá trị average silhouette
- Vẽ đường cong average silhouette theo các giá trị k
- Vị trí có average silhouette lớn nhất là số cụm cần tìm

Xét ví dụ về mẫu dữ liệu sau:



Hình 4.3: Mẫu dữ liệu

Vẽ đường cong average silhouette:



Ta chọn k = 3 làm số cụm cho mẫu dữ liệu trên.

5 Implement trên Python

5.1 Bài toán phân cụm dữ liệu

Import các thư viện cần thiết:

```
import numpy as np # math calculation libary
import matplotlib.pyplot as plt # visualize data on graph
from scipy.spatial.distance import cdist # calculate distance from point to point
```

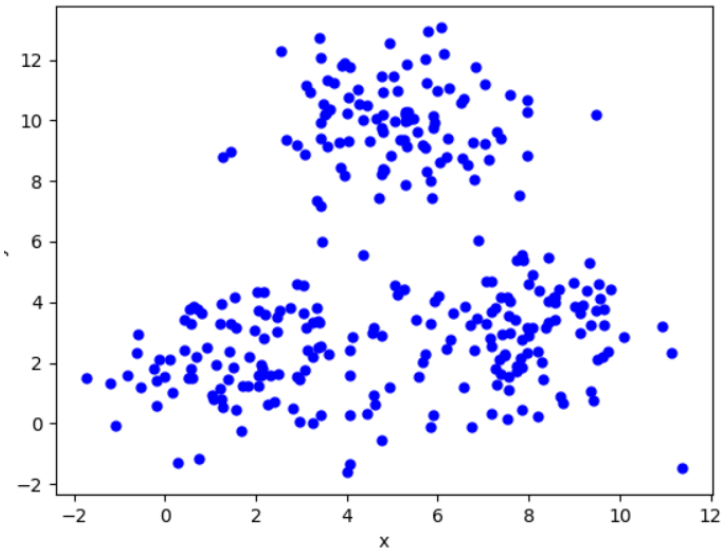
Khởi tạo ngẫu nhiên các điểm dữ liệu. Để thể hiện tính chất cụm của dữ liệu, các mẫu dữ liệu này được khởi tạo xung quanh 3 tâm cụm (2,2), (8,3) và (5,10). Ứng với mỗi tâm cụm có 100 điểm dữ liệu, cụ thể là các cụm X0, X1 và X2.

```
means = [[2, 2], [8, 3], [5, 10]]
cov = [[2, 0], [0, 2]]
n_samples = 100
n_cluster = 3 # number of cluster, for example, 3
X0 = np.random.multivariate_normal(means[0], cov, n_samples)
X1 = np.random.multivariate_normal(means[1], cov, n_samples)
X2 = np.random.multivariate_normal(means[2], cov, n_samples)
X = np.concatenate((X0, X1, X2), axis=0)
```

Hiển thị các mẫu dữ liệu được khởi tạo trên đồ thị:

```
plt.xlabel('x')
plt.ylabel('y')
plt.plot(X[:, 0], X[:, 1], 'bo', markersize=5)
plt.plot()
plt.show()
```

Kết quả cho ra ở Console:



Hình 5.1: Các mẫu dữ liệu được khởi tạo

Khởi tạo các điểm cluster của mẫu dữ liệu bằng cách lấy ngẫu nhiên các điểm dữ liệu. Số điểm dữ liệu được lấy bằng với số cluster.

```
def kmeans_init_centers(X, n_cluster):
    # random k index beetween 0 and shape(X) without duplicate index.
    # return X[index] as cluster
    return X[np.random.choice(X.shape[0], n_cluster, replace=False)]
```

Hàm xác định tâm cụm của từng điểm dữ liệu trong tập dữ liệu. Với mỗi điểm dữ liệu trong tập dữ liệu, tâm cụm của nó sẽ là 1 trong các tâm cụm gần với nó nhất.

```
def kmeans_predict_labels(X, centers):
    D = cdist(X, centers)
    # return index of the closest center
    return np.argmin(D, axis = 1)
```

Hàm cập nhật lại vị trí của các tâm cụm. Việc tính toán lại tọa độ của mỗi tâm cụm được thực hiện bằng cách lấy trung bình cộng tọa độ của tất cả các điểm dữ liệu của cụm.

```
def kmeans_update_centers(X, labels, n_cluster):
    centers = np.zeros((n_cluster, X.shape[1]))
    for k in range(n_cluster):
        # collect all points assigned to the k-th cluster
        Xk = X[labels == k, :]
        # take average
        centers[k,:] = np.mean(Xk, axis = 0)
    return centers
```

Hàm kiểm tra kết quả sau khi cập nhật vị trí tâm cụm. Nếu việc cập nhật lại vị trí của các tâm cụm không có thay đổi gì có nghĩa là việc phân cụm đã hoàn tất.

```
def kmeans_has_converged(centers, new_centers):
    # return True if two sets of centers are the same
    return (set([tuple(a) for a in centers]) == set([tuple(a) for a in new_centers]))
```

Hàm hiển thị đồ thị của các mẫu dữ liệu trong quá trình phân cụm và các tâm cụm của nó:

```
def kmeans_visualize(X, centers, labels, n_cluster, title):
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title(title)
    plt_colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
    for i in range(n_cluster):
        data = X[labels == i] # get data of cluster i
        plt.plot(data[:, 0], data[:, 1], plt_colors[i] + '^', markersize = 4, label = '
            cluster_' + str(i)) # draw cluster i
        plt.plot(centers[i][0], centers[i][1], plt_colors[i+4] + 'o', markersize = 10,
            label = 'center_' + str(i)) # draw centroid of cluster
    plt.legend()
    plt.show()
```

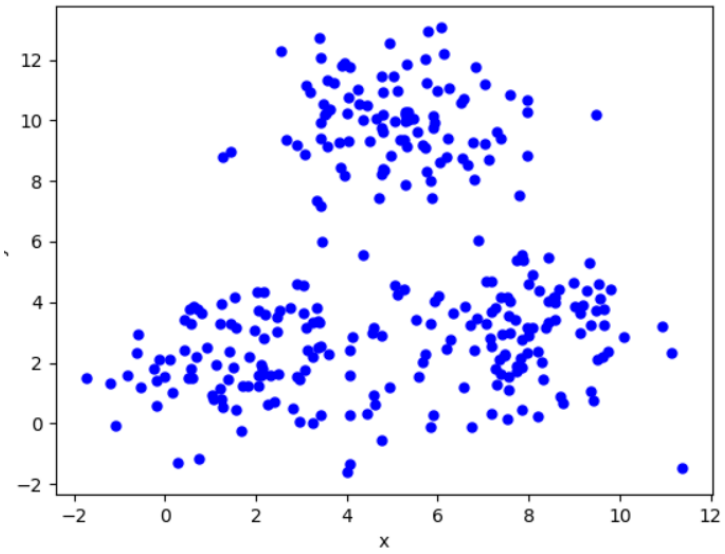
Hàm thực hiện Kmeans. Bạn đầu, chúng ta sẽ khởi tọa độ cho các tâm cụm. Sau đó lặp lại việc tìm tâm cụm cho các điểm dữ liệu, cập nhật tọa độ tâm cụm cho tới khi tọa độ của các tâm cụm không còn thay đổi nữa. Hàm `kmeans_visualize` được gọi sau mỗi lần gán lại tâm cụm cho từng điểm dữ liệu hoặc tính toán lại tọa độ các tâm cụm.

```
def kmeans(init_centes, init_labels, X, n_cluster):
    centers = init_centes
    labels = init_labels
    times = 0
    while True:
        labels = kmeans_predict_labels(X, centers)
        kmeans_visualize(X, centers, labels, n_cluster, 'Assigned label for data at time
            = ' + str(times + 1))
        new_centers = kmeans_update_centers(X, labels, n_cluster)
        if kmeans_has_converged(centers, new_centers):
            break
        centers = new_centers
        kmeans_visualize(X, centers, labels, n_cluster, 'Update center possition at time
            = ' + str(times + 1))
        times += 1
    return (centers, labels, times)
```

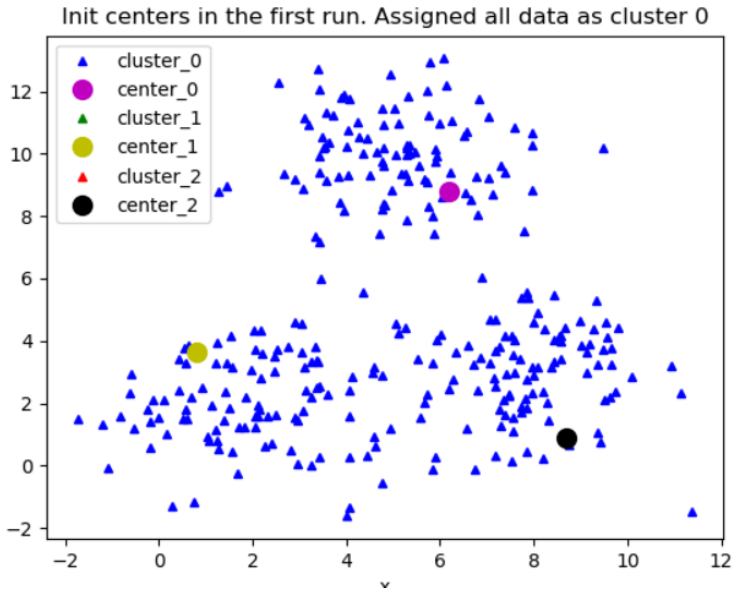
Khởi tạo toạ độ tâm các cụm, gọi hàm `kmeans` để thực thi.

```
init_centers = kmeans_init_centers(X, n_cluster)
print(init_centers)
init_labels = np.zeros(X.shape[0])
kmeans_visualize(X, init_centers, init_labels, n_cluster, 'Init centers in the first
    run. Assigned all data as cluster 0')
centers, labels, times = kmeans(init_centers, init_labels, X, n_cluster)
print('Kmeans has converged after', times, 'times')
print("Final centroid: ", centers)
```

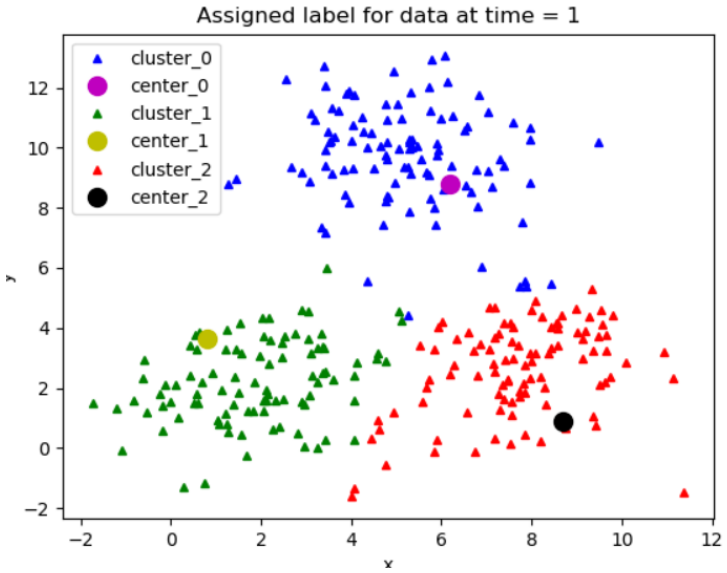
Quá trình phân cụm được thể hiện ở biểu đồ:



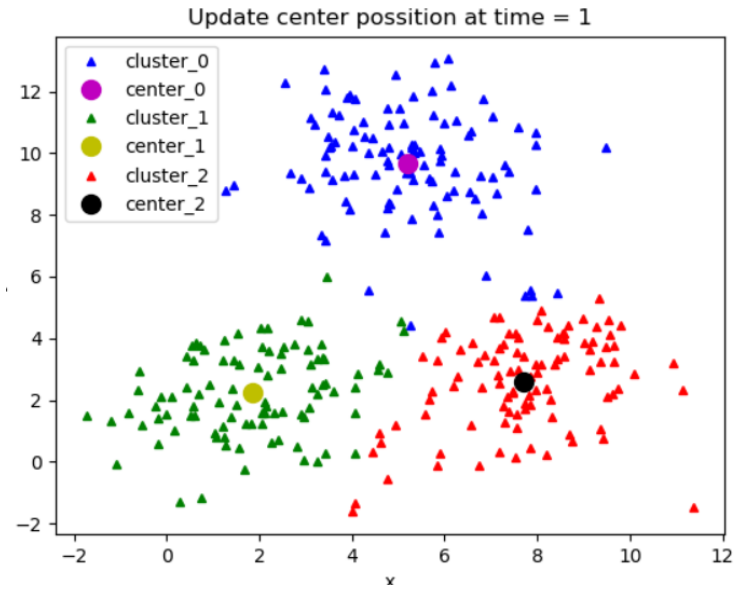
Hình 5.2: Khởi tạo mẫu dữ liệu



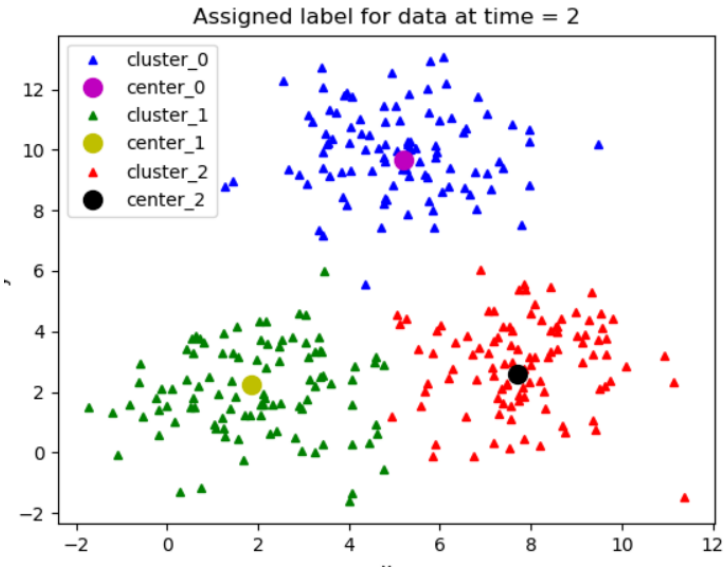
Hình 5.3: Khởi tạo ngẫu nhiên 3 trung tâm cụm



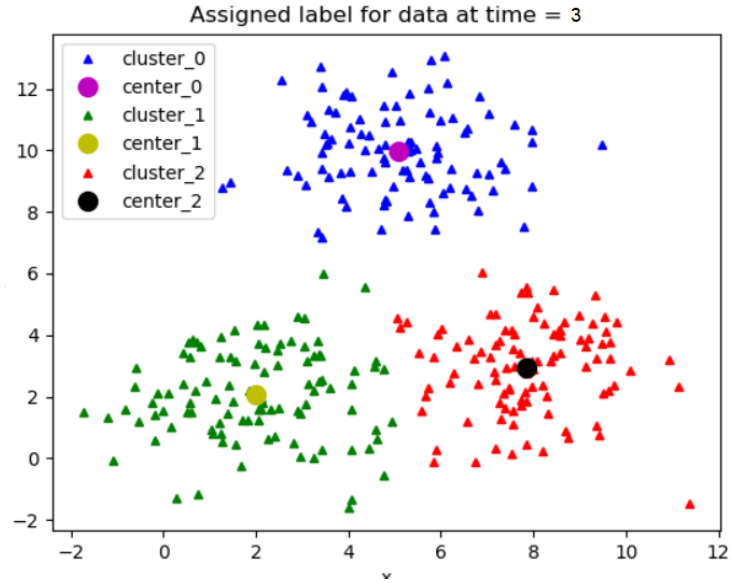
Hình 5.4: Phân cụm dữ liệu lần 1



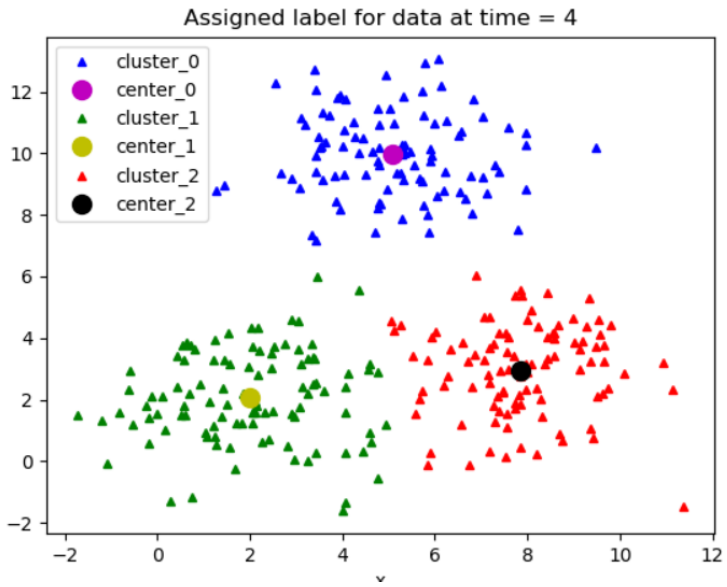
Hình 5.5: Cập nhật lại các điểm centroid



Hình 5.6: Phân cụm dữ liệu lần 2



Hình 5.7: Phân cụm dữ liệu lần 3



Hình 5.8: Phân cụm dữ liệu lần 4

Sau khi tiến hành xong việc phân cụm dữ liệu, ở Console cho ra kết quả toạ độ các centroid ban đầu và sau cùng:

```
[[6.18653382 8.81216618]
[0.81554069 3.64003336]
[8.70275363 0.91404576]]
Final centroid: [[5.08051201 9.96834598]
[2.00313276 2.06729414]
[7.86474658 2.96544294]]
```

Dễ thấy toạ độ các điểm centroid sau cùng gần như trùng khớp với 3 tâm cụm khởi tạo ban đầu là (2,2), (8,3) và (5,10).

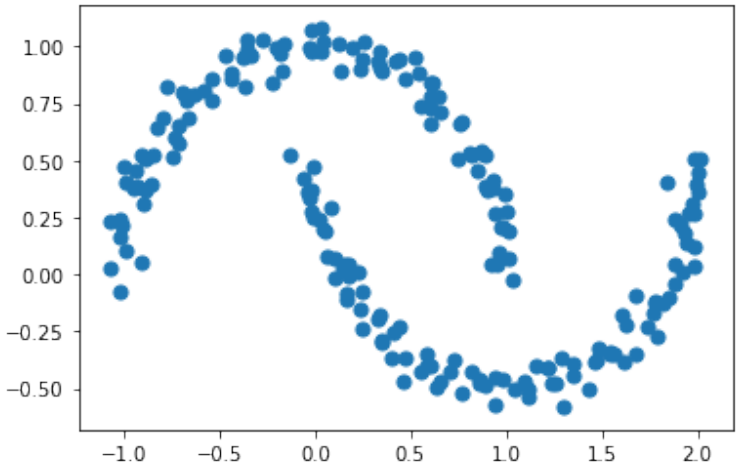
5.2 Trường hợp phân cụm sai

Theo như nhược điểm của thuật toán K-Means đã được nêu ở trên thì trong đó có một nhược điểm là các cluster cần có dạng hình tròn (cầu), nếu không thì K-Means sẽ hoạt động không hiệu quả. Do đó, chúng ta cùng thực nghiệm về vấn đề này bằng python, sử dụng tập dữ liệu của sklearn và hàm `make_moons` để tạo ra tập dữ liệu có hai cụm hình dạng trăng lưỡi liềm và số lượng các vector là $N = 200$, được trình bày như dưới đây.

```
from sklearn.datasets import make_moons
X, y = make_moons(200, noise = 0.05, random_state = 0)
```

Hiển thị tập dữ liệu và thiết lập số nhóm $K = 2$.

```
plt.scatter(X[:,0], X[:,1], s = 50)
K = 2
```

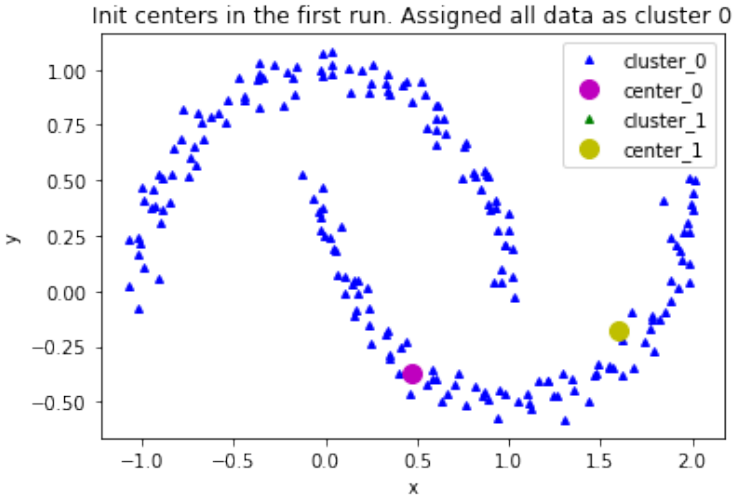


Hình 5.9: Tập dữ liệu không phải dạng tròn

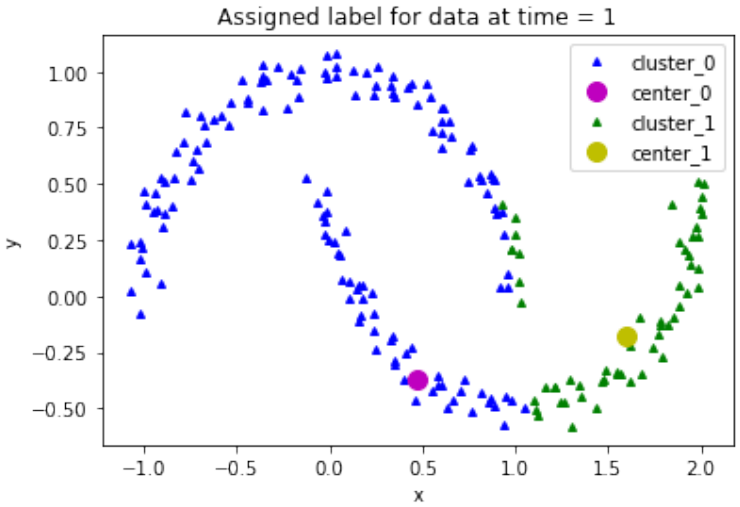
Thực hiện phân cụm một cách bình thường với các hàm có ở phía trên.

```
init_centers = kmeans_init_centers(X, K)
print(init_centers)
init_labels = np.zeros(X.shape[0])
kmeans_visualize(X, init_centers, init_labels, K, 'Init centers in the first run.
    Assigned all data as cluster 0')
centers, labels, times = kmeans(init_centers, init_labels, X, K)
print('Kmeans has converged after', times, 'times')
print("Final centroid: ", centers)
```

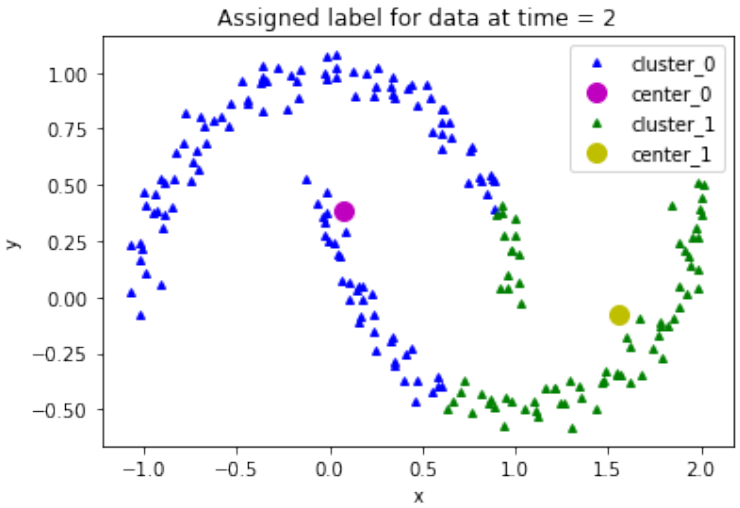
Quá trình phân cụm được biểu thị bằng các sơ đồ dưới đây.



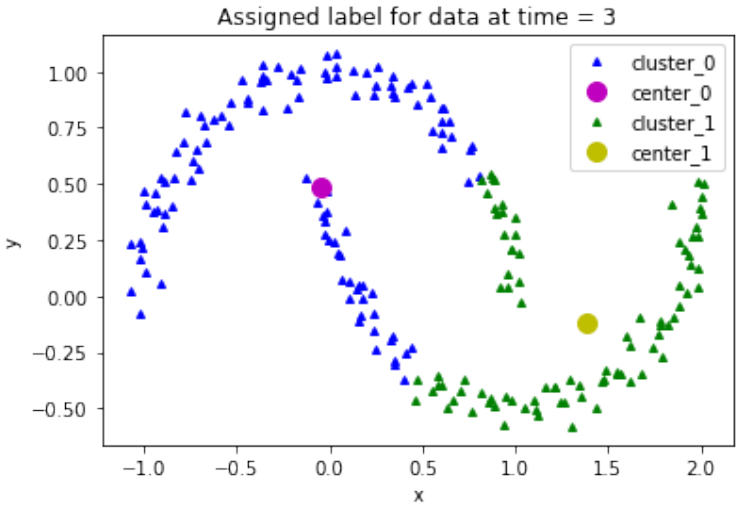
Hình 5.10: Thực hiện khởi tạo các điểm trung tâm



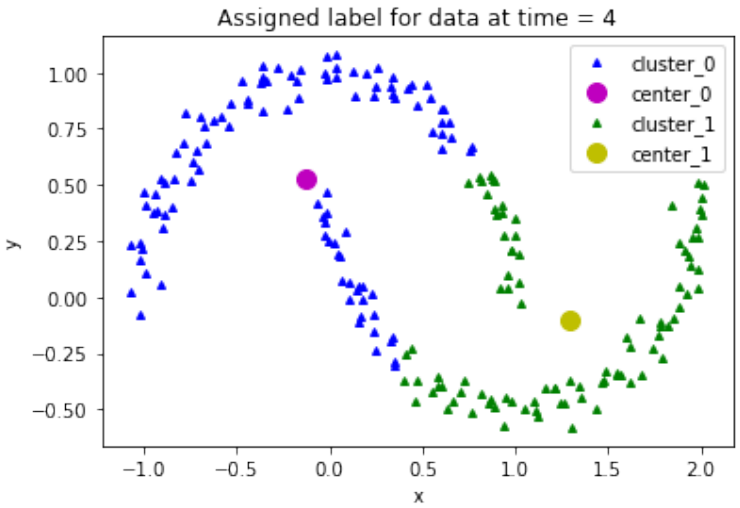
Hình 5.11: Phân cụm dữ liệu lần 1



Hình 5.12: Phân cụm dữ liệu lần 2

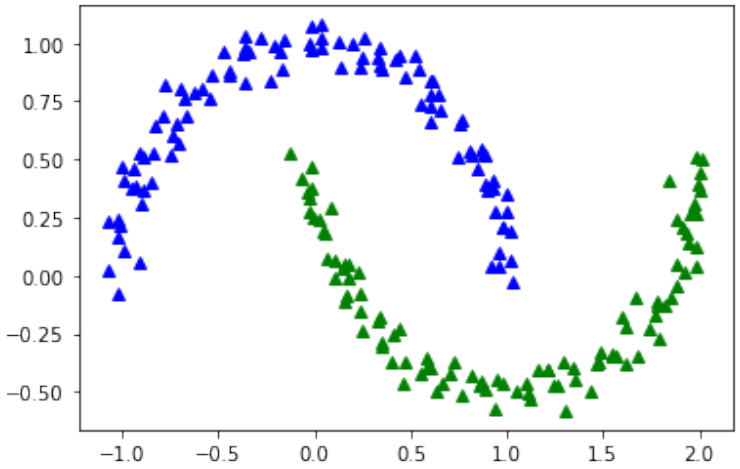


Hình 5.13: Phân cụm dữ liệu lần 3



Hình 5.14: Phân cụm dữ liệu lần 4

Như vậy chúng ta đã thấy được rằng, đối với các tập dữ liệu không có dạng hình tròn (ở trường hợp phía trên có dạng hình lưỡi liềm) thì thuật toán K means không còn chính xác nữa. Chúng ta có thể tham khảo phân loại đúng của tập dữ liệu trên thì nên được phân loại như sau.



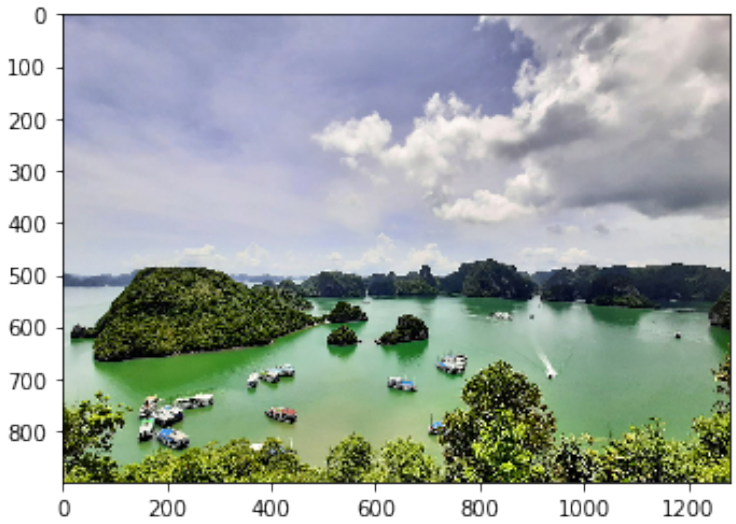
Hình 5.15: Dạng phân cụm đúng của tập dữ liệu trên.

5.3 Nén hình ảnh

Ngoài ra chúng ta có thể áp dụng thuật toán K-Means tách vật thể từ trong ảnh hoặc có thể dùng để nén ảnh và dữ liệu. Ở đây chúng ta sẽ bàn tới các sử dụng thuật toán K-means để nén ảnh. Trước tiên đi vào các khái niệm về ảnh, một ảnh là tập hợp $h \times w$ pixel, với mỗi pixel được gọi là các điểm ảnh mang 3 giá trị Red, Green và Blue, còn h và w lần lượt là chiều cao và chiều rộng của ảnh. Trong bài toán này chúng ta sẽ sử dụng thư viện có sẵn hỗ trợ thuật toán K-means là `sklearn`, chúng ta sẽ sử dụng hàm `KMeans` để phân cụm dữ liệu. Đầu tiên, thực hiện import thư viện và hình ảnh cần nén vào, ở đây sử dụng hình ảnh Vịnh Hạ Long (https://en.wikipedia.org/wiki/H%E1%BA%A1_Long_Bay).

```
1  import matplotlib.image as mpimg
2  import matplotlib.pyplot as plt
3  from sklearn.cluster import KMeans
4  img = mpimg.imread('halongbay.jpg')
5  plt.imshow(img)
```

Kết quả hiện thị trên màn hình.



Hình 5.16: Hình ảnh được thực hiện nén ảnh.

Tiếp tục biến đổi bức ảnh thành một ma trận với mỗi dòng tương ứng với mỗi pixel để phù hợp với định dạng dữ liệu đầu vào. Bên cạnh đó, sử dụng một biến khác để lưu giá trị các pixel này để lưu giá trị ảnh sau khi được nén.

```
X = np.copy(img)
X = X.reshape(X.shape[0]*X.shape[1], 3)
```

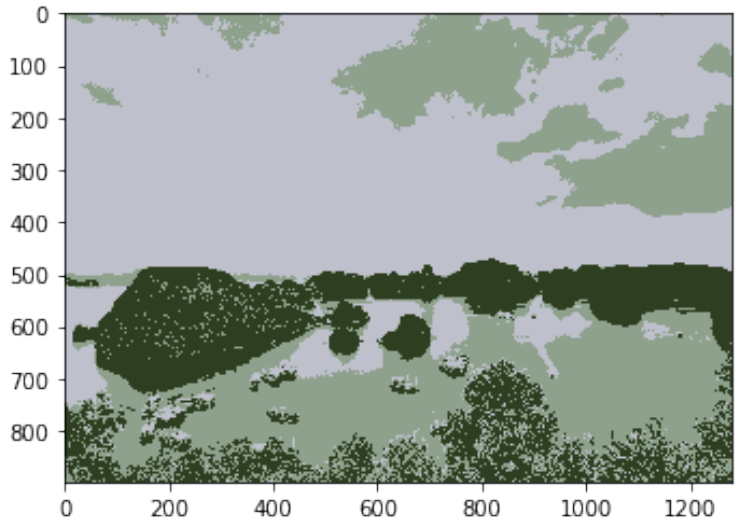
Áp dụng hàm `KMeans` vào tập dữ liệu pixel với số màu chúng ta mong muốn còn lại là 3, tức là gom tất cả các màu của ảnh về còn lại 3 màu.

```
kmeans = KMeans(n_clusters = 3, random_state = 0)
labels = kmeans.fit_predict(X)
```

Sau khi đã thực hiện gom cụm các pixel, tiến hành thay thế các pixel thành các pixel mới tương ứng với điểm trung tâm của cụm và nó được gom vào, sau đó chuyển đổi tập dữ liệu về lại định dạng ban đầu.

```
for i in range(3):
 X[labels==i] = kmeans.cluster_centers_[i]
X = X.reshape(img.shape[0],img.shape[1], 3)
plt.imshow(X)
```

Kết quả hình ảnh sau khi nén thu được.



Hình 5.17: Hình ảnh sau khi được nén.

Hình ảnh sau khi được nén chỉ còn lại 3 màu, nên dung lượng của hình ảnh sẽ bị giảm đi đáng kể. Đáp ứng cho nhu cầu lưu trữ mà không cần quá cao về chất lượng hình ảnh, hoặc chỉ cần hình ảnh chất lượng thấp để tiết kiệm bộ nhớ.