Hierarchical Methods

Group 5

Khanh Le Tran Quoc Khoa Tran Nhat Khoa Huynh Tong Dang

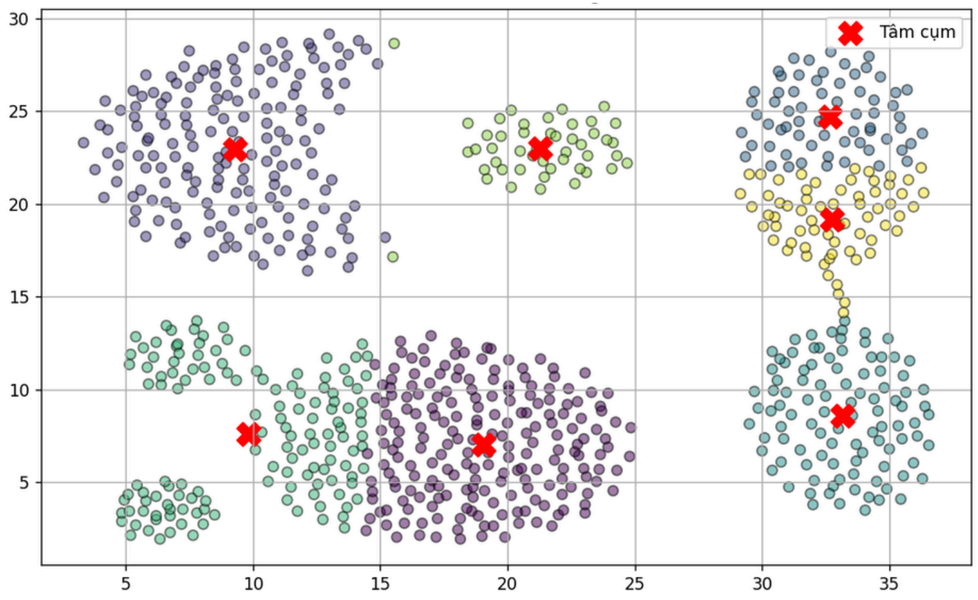
University of Information Technology

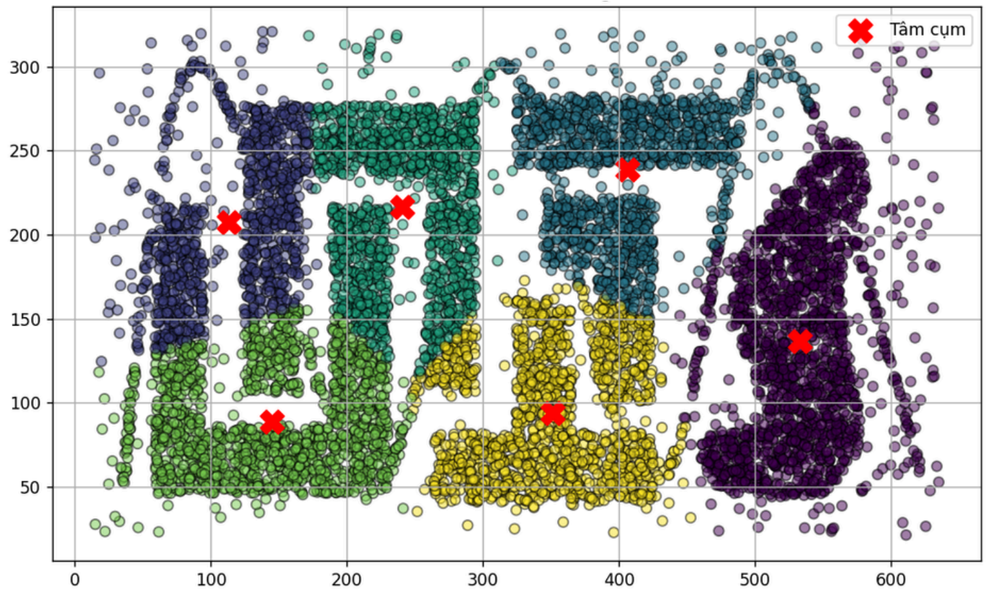
4th November 2024

Link Github

Motivation

- Partitioning methods are widely used due to their simplicity, but they come with certain limitations:
  - **Require the number of clusters.**
  - **Struggle with data that has complex struture.**



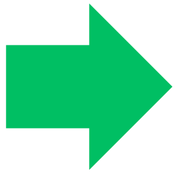


Motivation

- Partitioning methods are widely used due to their simplicity, but they come with certain limitations:
  - **Require the number of clusters.**
  - **Struggle with data that has complex struture.**
- We want to understand why the model decides to form specific clusters.

Motivation

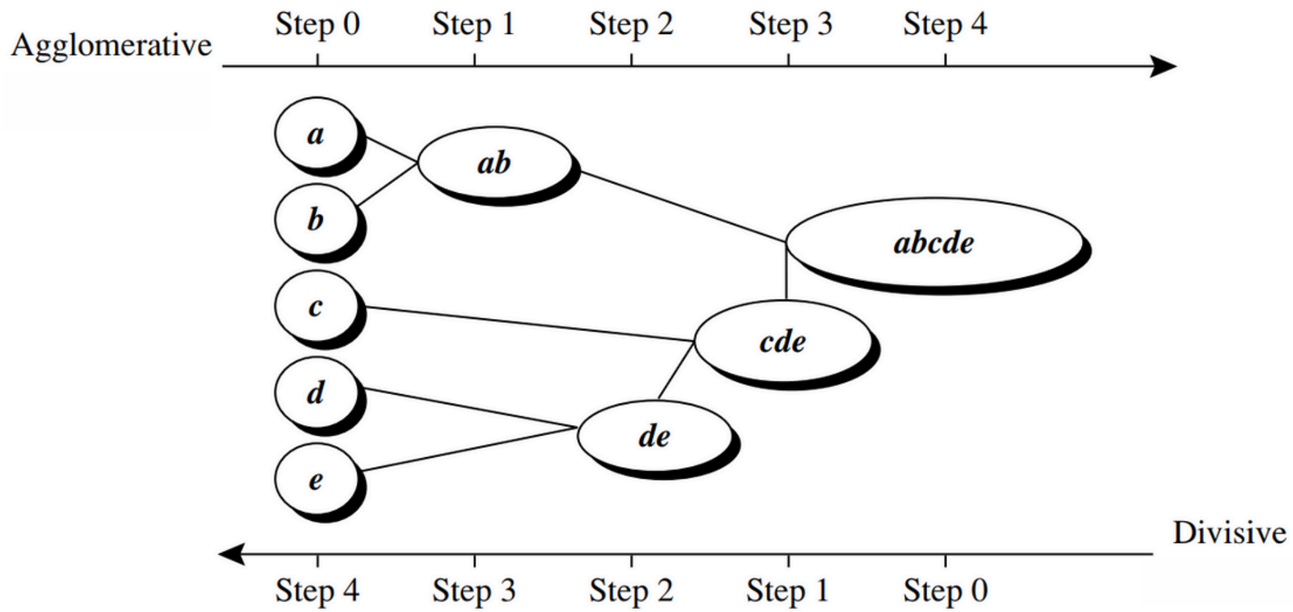
- Partitioning methods are widely used due to their simplicity, but they come with certain limitations:
  - **Require the number of clusters.**
  - **Struggle with data that has complex struture.**
- We want to understand why the model decides to form specific clusters.



Hierachical Clustering Methods

Hierarchical Clustering Introduction

- **Hierarchical clustering** is a clustering method in which data objects are **arranged into a hierarchical structure**. It can be performed in two main ways:
  - **Agglomerative (bottom-up)**
  - **Divisive (top-down)**



Agglomerative Hierarchical Clustering

- A **bottom-up** approach to clustering
- Begin with each the data point is a cluster and progressively **merge** them into **bigger clusters**.
- This process continues recursively until there is only **a cluster**.
- How we decide to merge two clusters of each iter?

→ How **“close”** or **“similarity”** between two clusters

Agglomerative Hierarchical Clustering

- Ward method: how much the **sum of squares will increase when we merge them.**

$$D(C_i, C_j) = \sum_{x \in C_i \cup C_j} \|x - \mu_{C_i \cup C_j}\|^2 - \sum_{x \in C_i} \|x - \mu_{C_i}\|^2 - \sum_{x \in C_j} \|x - \mu_{C_j}\|^2$$

- Complete method: **maximum distance** beetween points in two clusters.

$$D(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y)$$

- Average method: **Average distance** beetween points in two clusters.

$$D(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i} \sum_{y \in C_j} d(x, y)$$

- Single method: **minimum distance** beetween points in two clusters.

$$D(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y)$$

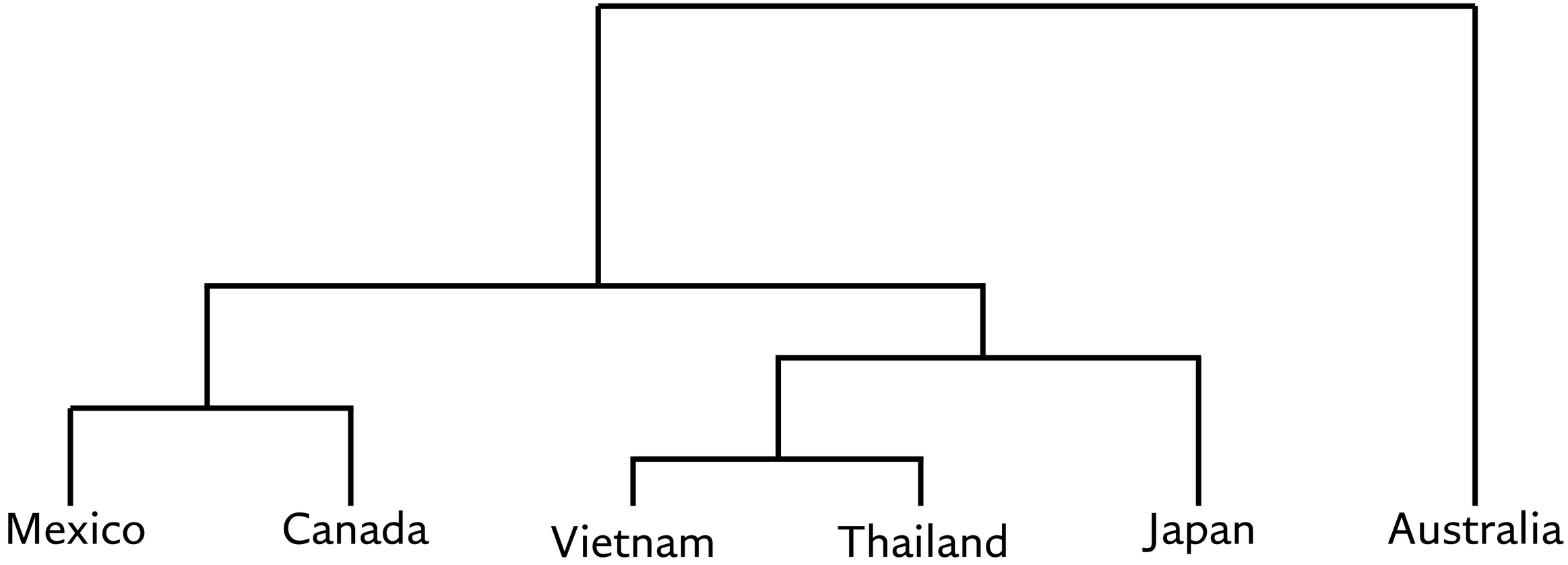
Agglomerative Hierarchical Clustering

Algorithm 1 Agglomerative Clustering

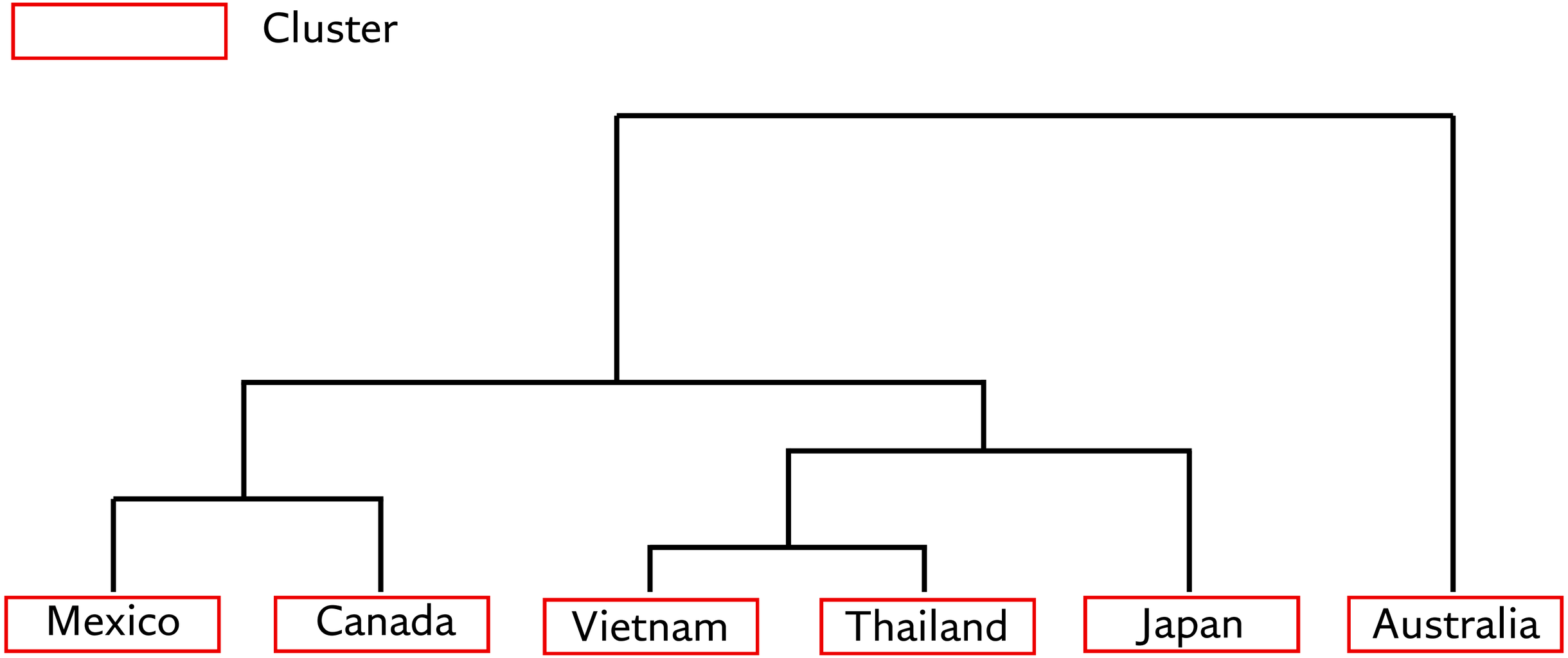
Require: Set of data points $X = \{x_1, x_2, \ldots, x_n\}$, number of clusters k
Ensure: k clusters containing all data points

1: Initialize each data point as its own cluster: $C = \{\{x_1\}, \{x_2\}, \ldots, \{x_n\}\}$
2: **while** $|C| > k$ **do**
3: Calculate **the distance** between each pair of clusters in C
4: Select the pair of clusters (C_i, C_j) with the minimum distance
5: Merge clusters C_i and C_j to form a new cluster C_{ij}
6: Update the set of clusters: $C = (C \setminus \{C_i, C_j\}) \cup \{C_{ij}\}$
7: **end while**
8: **return** C as the final set of clusters

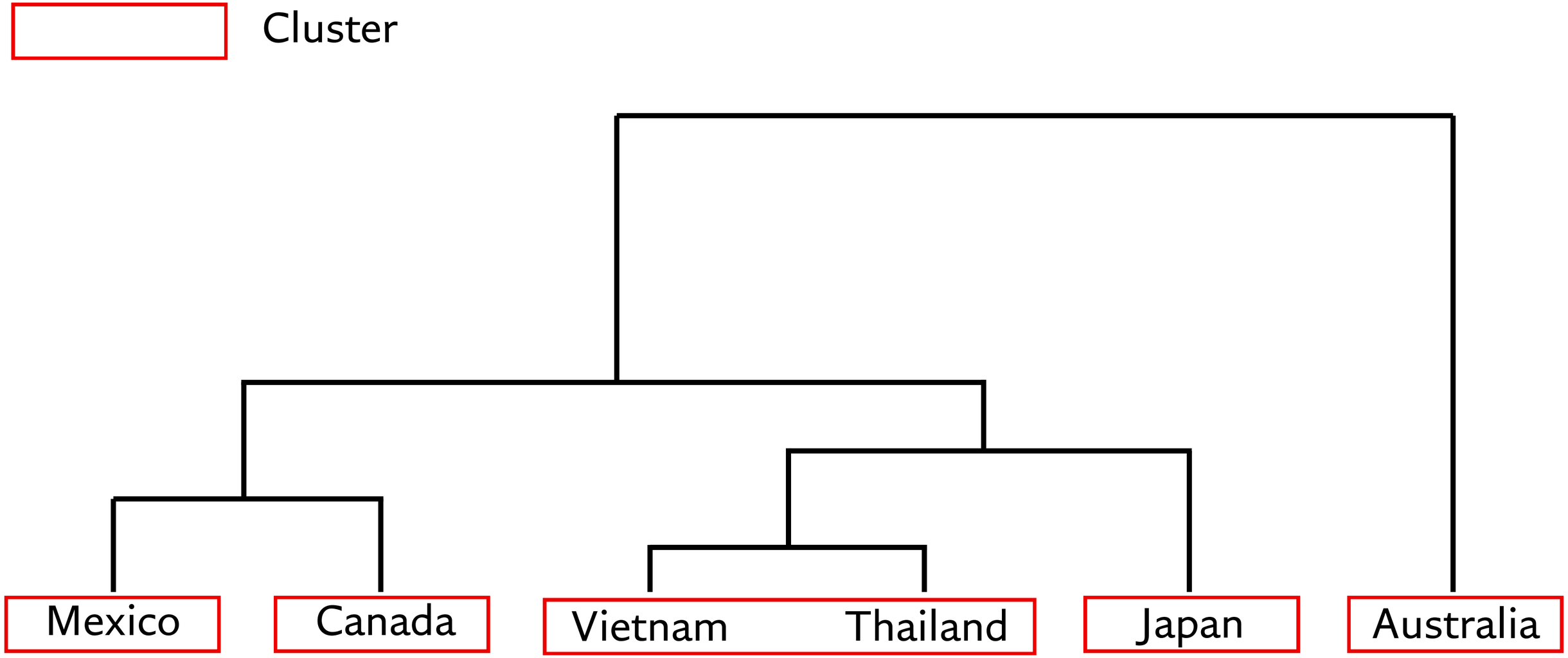
Dendrogram



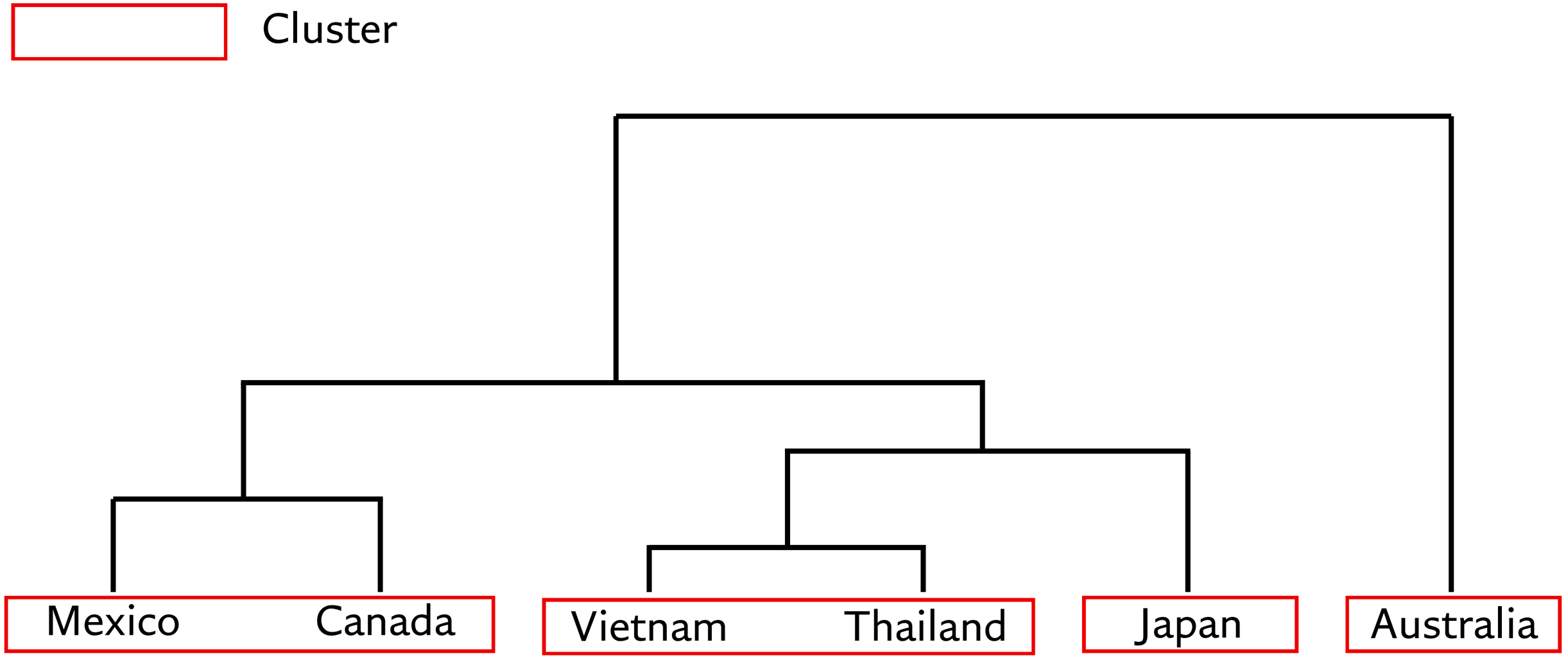
Dendrogram



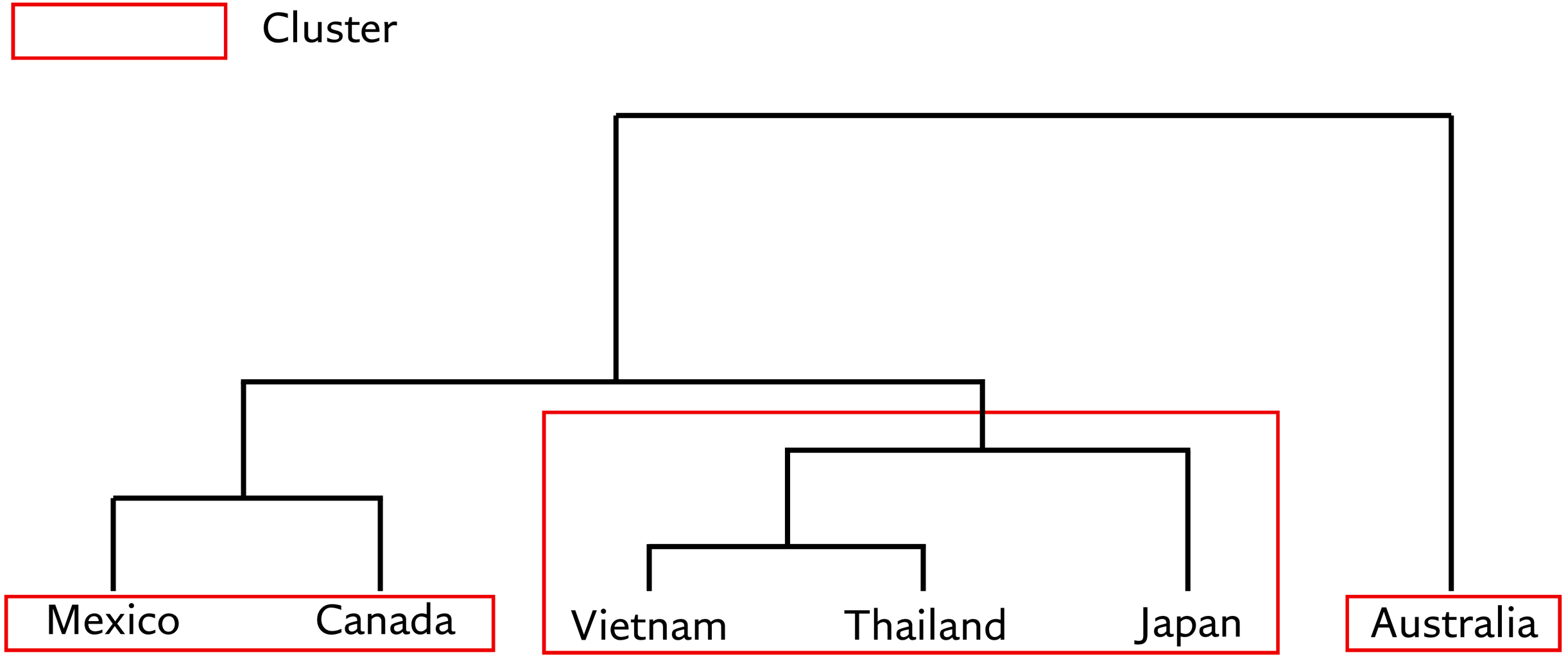
Dendrogram



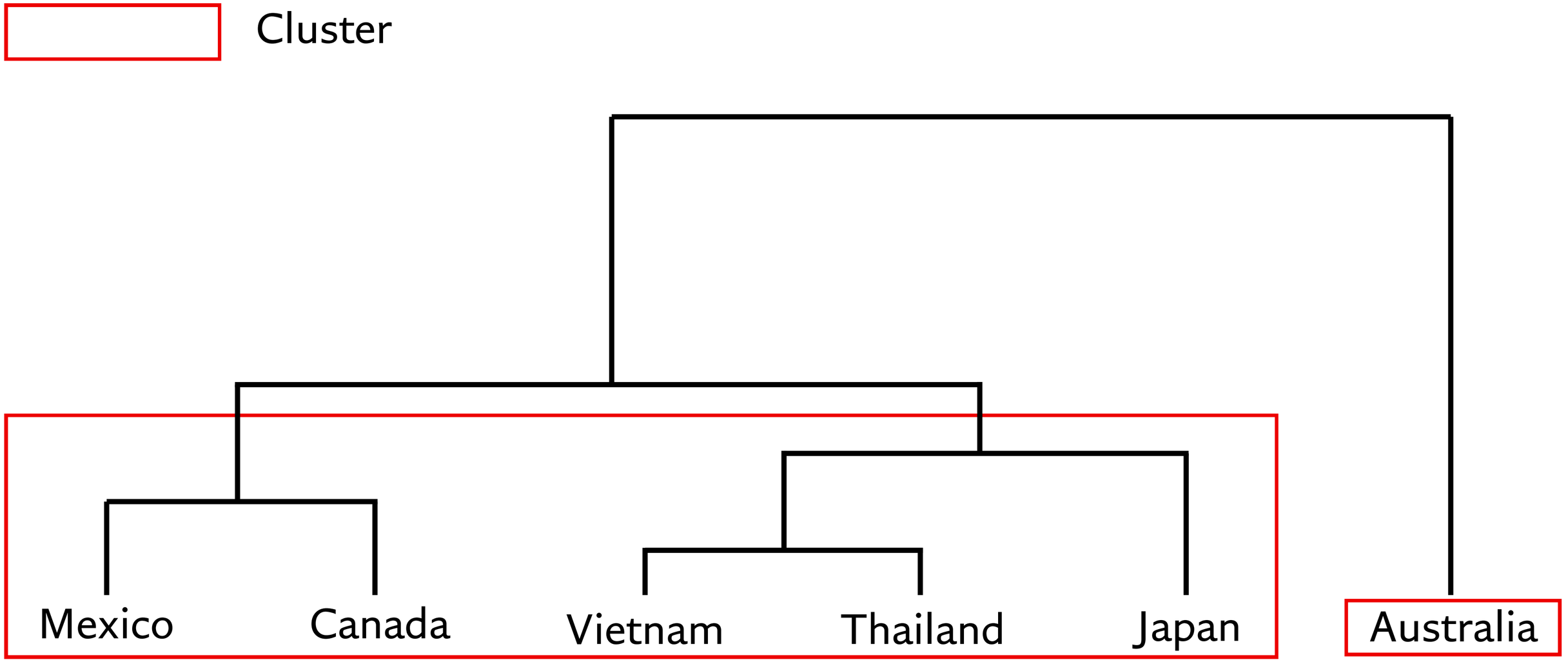
Dendrogram



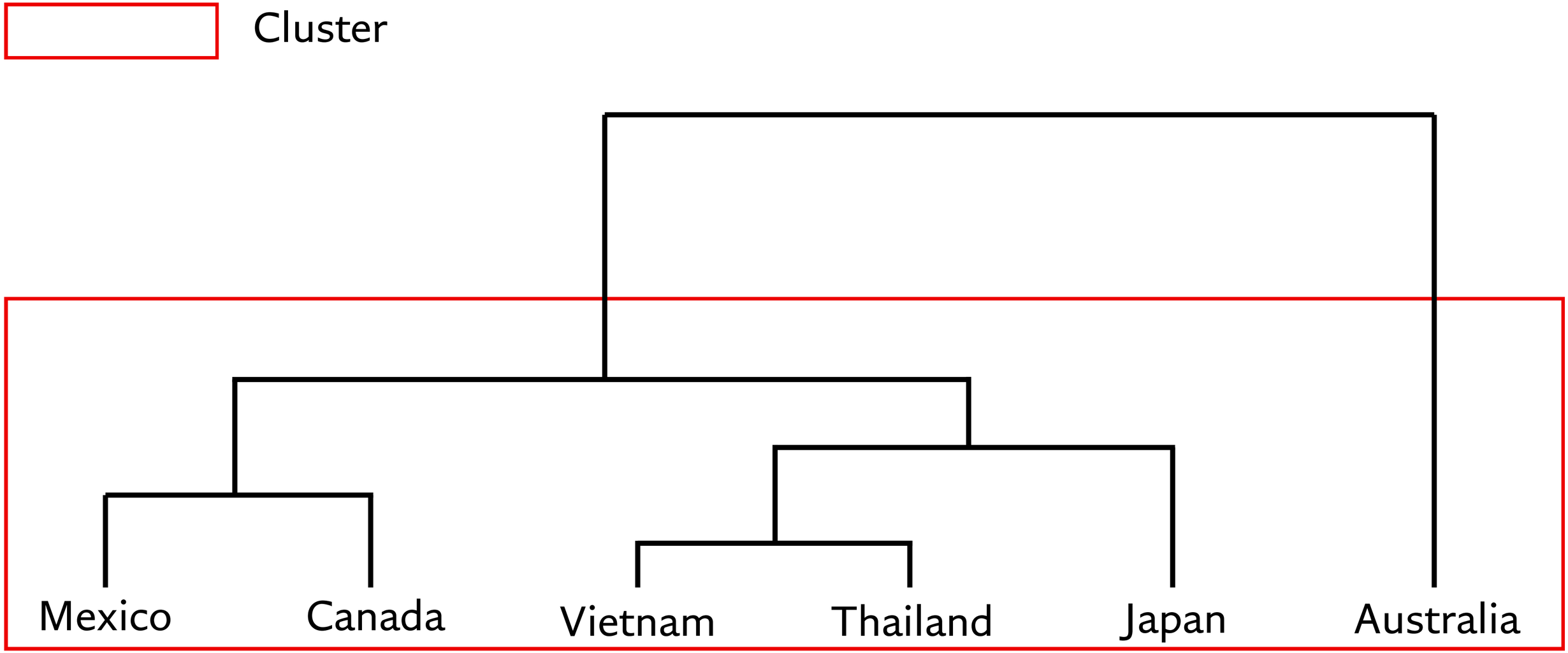
Dendrogram



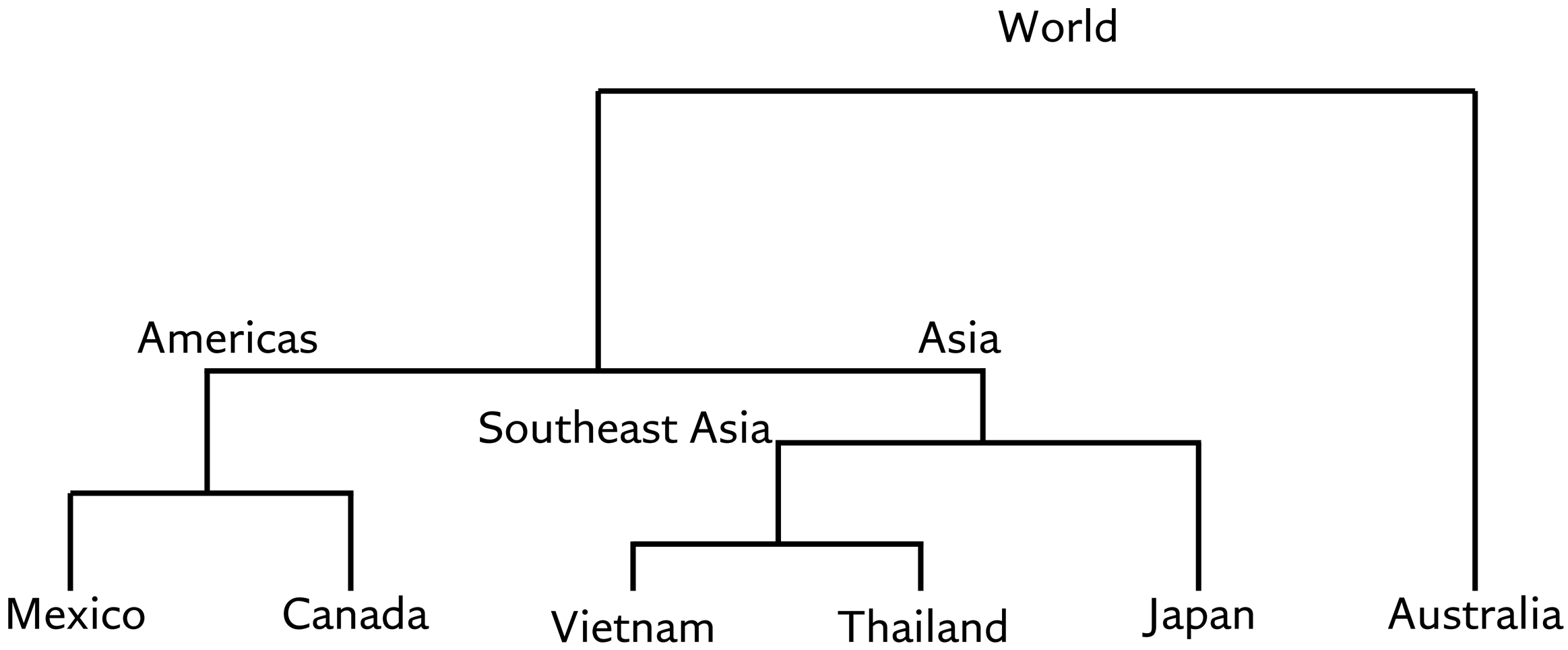
Dendrogram



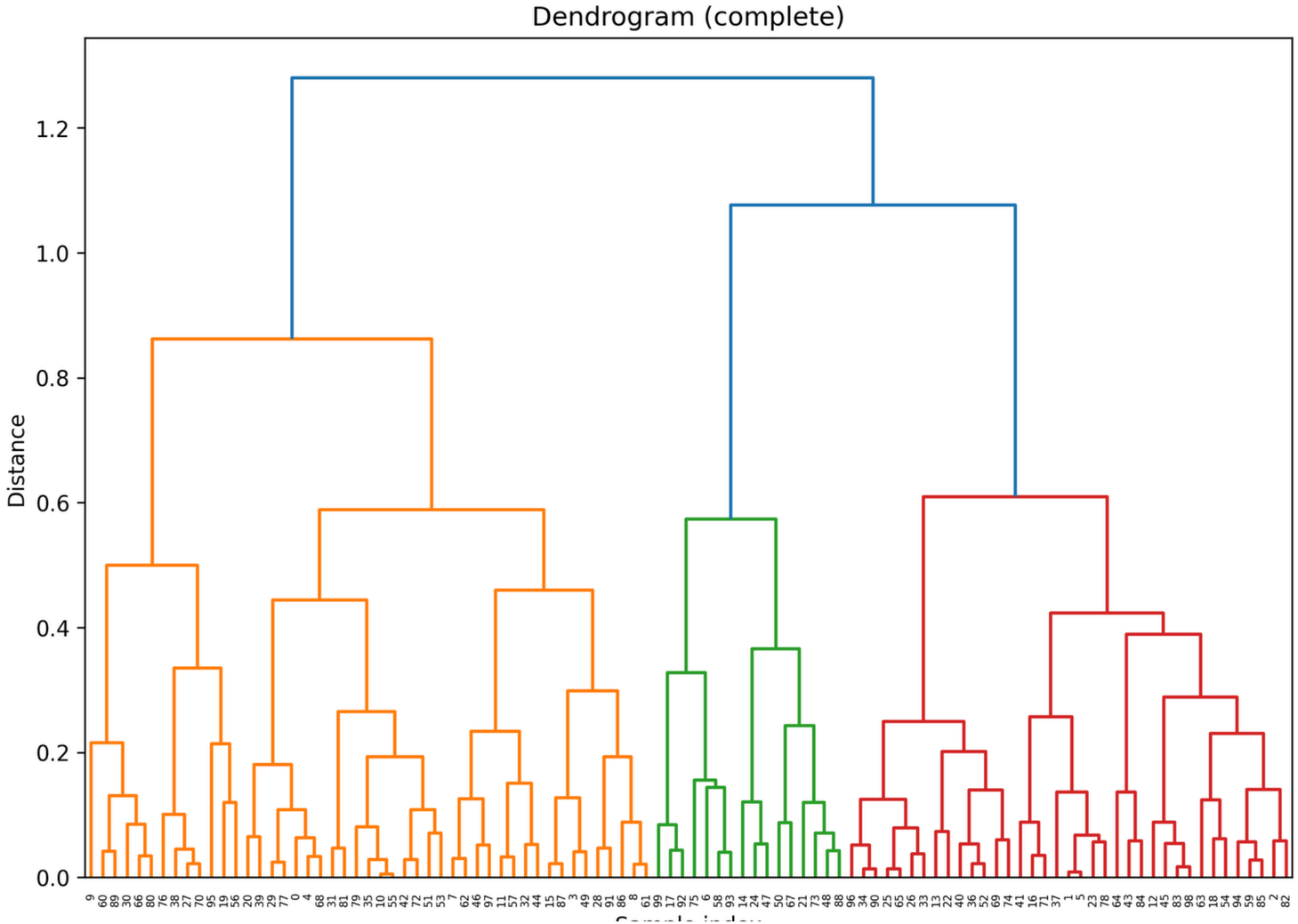
Dendrogram



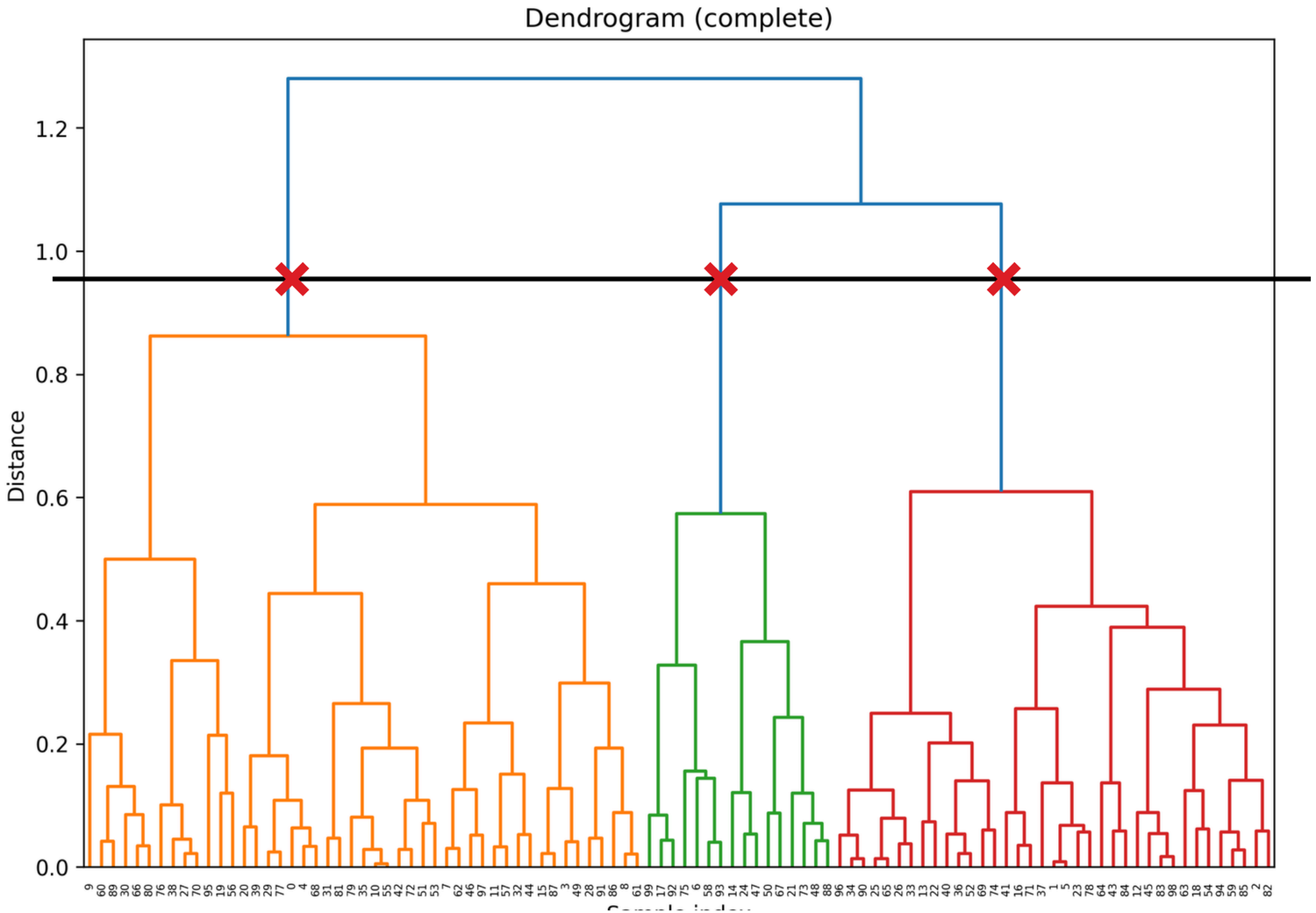
Dendrogram


World
Americas
Asia
Southeast Asia
Mexico
Canada
Vietnam
Thailand
Japan
Australia

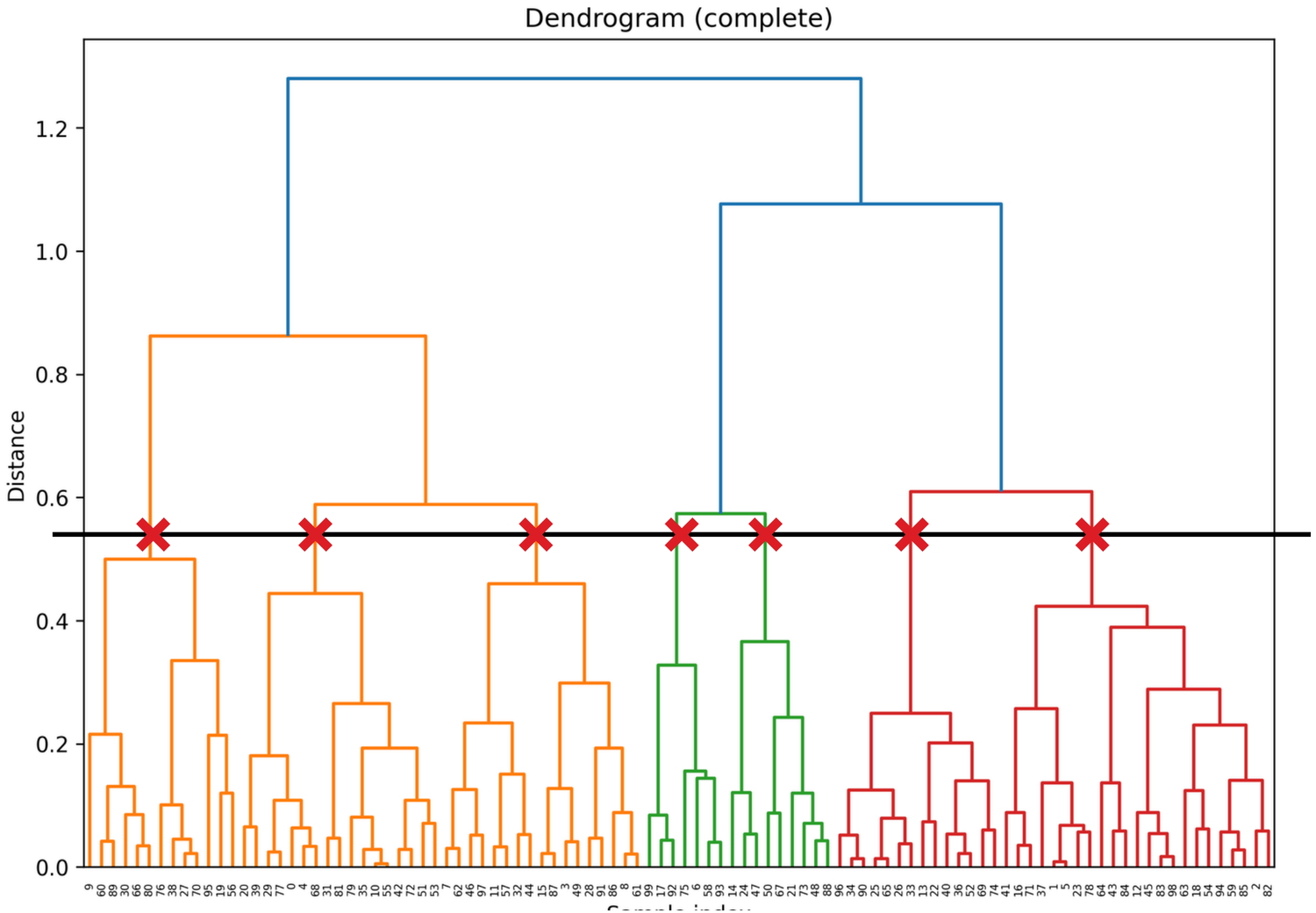
Dendrogram



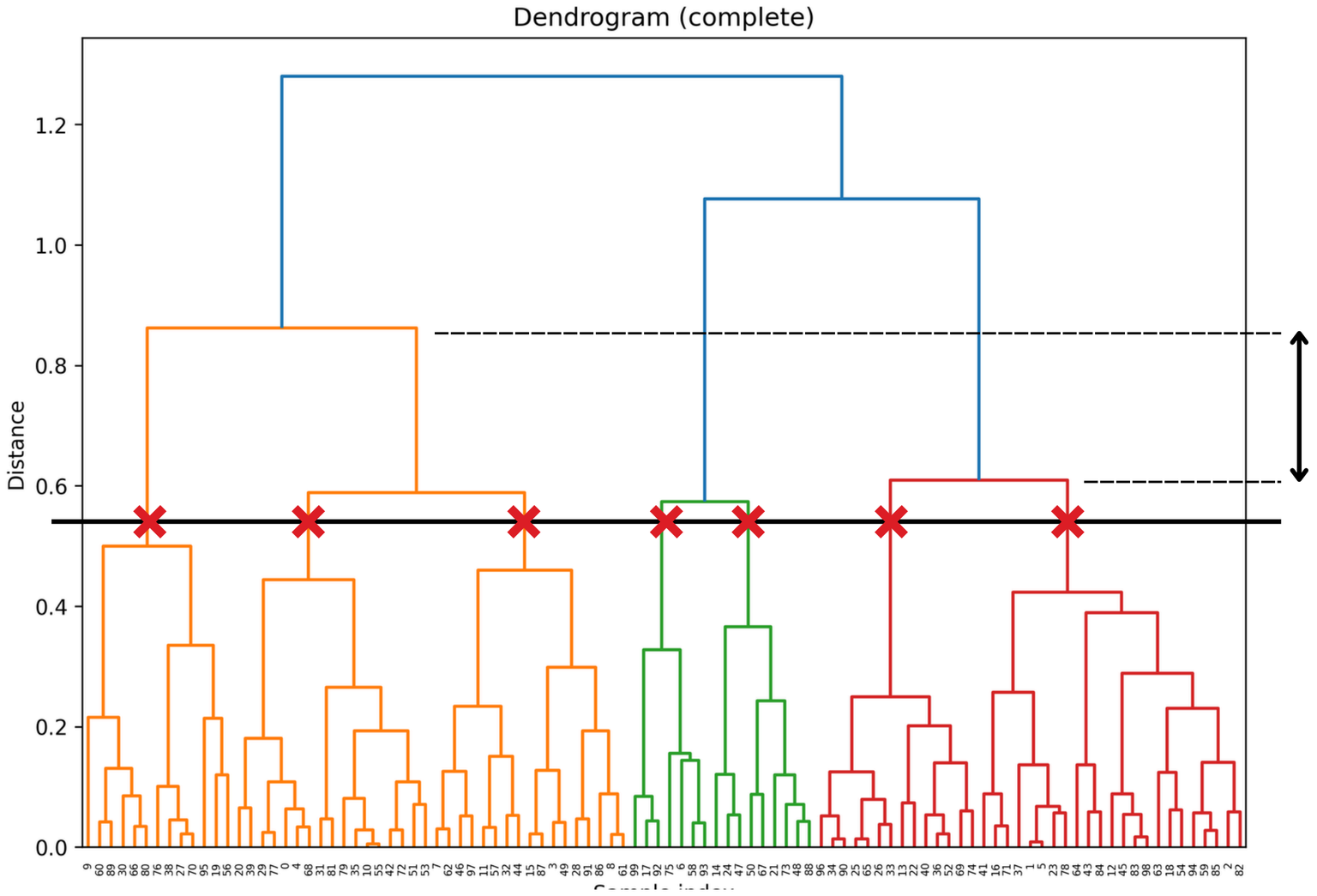
Dendrogram

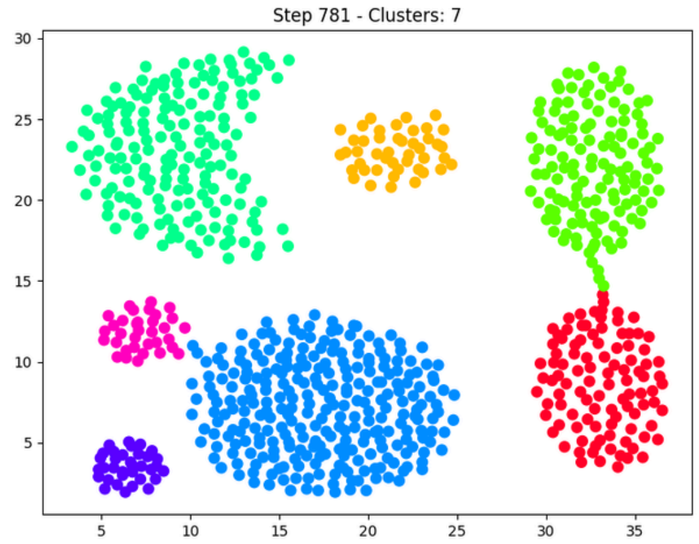


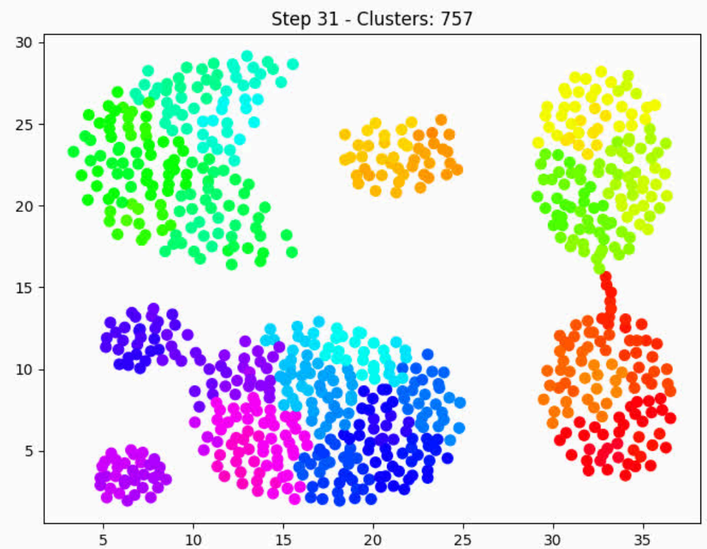
Dendrogram



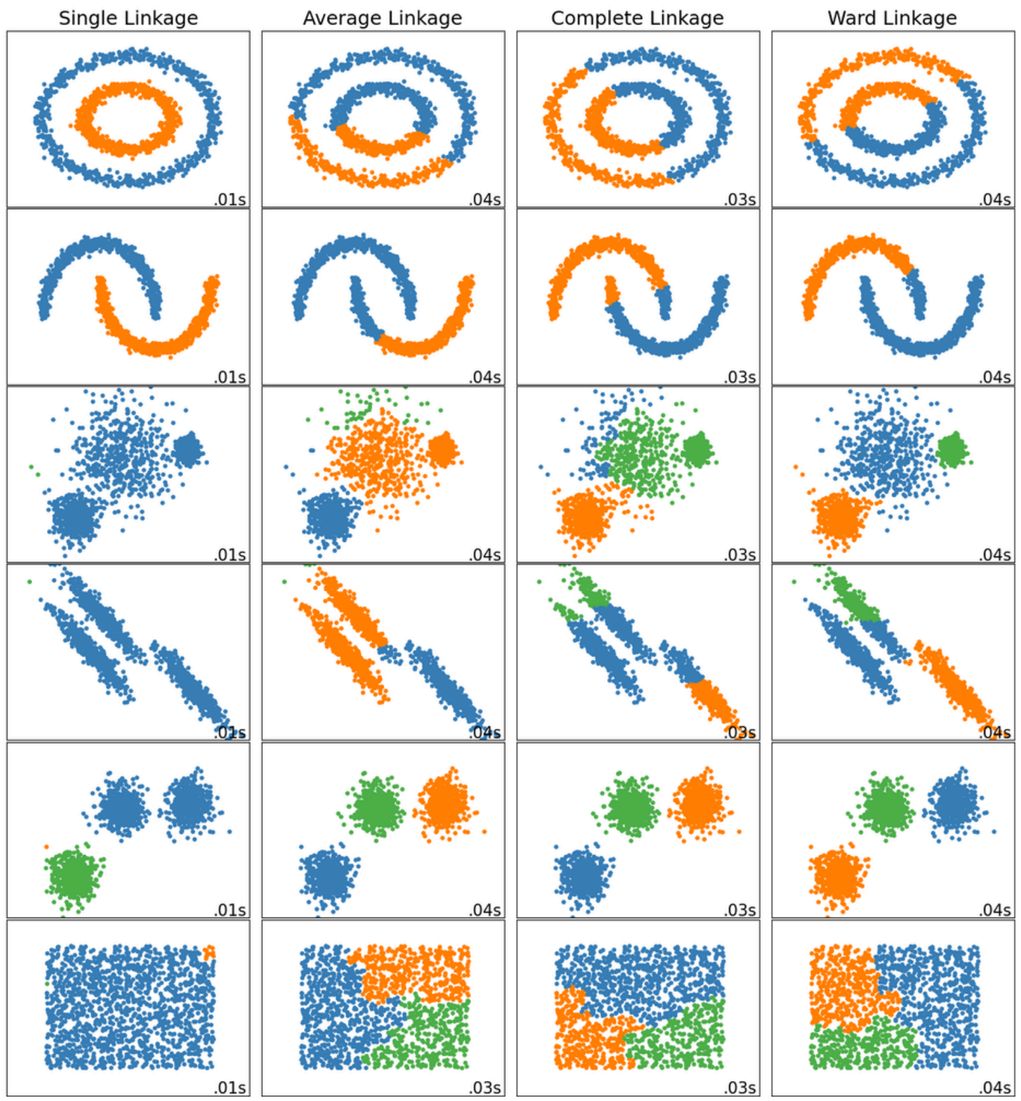
Dendrogram


Dendrogram (complete)
Distance
1.2
1.0
0.8
0.6
0.4
0.2
0.0


Step 781 - Clusters: 7


Step 31 - Clusters: 757

Experiment



- **Single**: fast, and can **perform well on non-globular data**, but it performs **poorly in the presence of noise**.
- **Average**, **Complete**: perform well on cleanly separated globular clusters, but have mixed results otherwise.
- **Ward**: the most effective method for noisy data.

Divisive Hierarchical Clustering

- A **top-down** approach to clustering, which is the opposite of the more common **agglomerative** clustering.
- Begin with all the data points in **one single cluster** and progressively split them into **smaller clusters**.
- This process continues recursively until each data point is its own cluster.
- The challenge in divisive hierarchical clustering is deciding **the best way to split** clusters at each step.

Divisive Hierarchical Clustering

Algorithm 2 Divisive Clustering

Require: Set of data points $X = \{x_1, x_2, \ldots, x_n\}$, number of clusters k
Ensure: k clusters containing all data points

1: Initialize the set of clusters: $C = \{X\}$
2: **while** $|C| < k$ **do**
3: Select the cluster C_i in C with **the lowest connection**
4: Split the cluster C_i into two sub-clusters (C_{i1}, C_{i2})
5: Update the set of clusters: $C = (C \setminus \{C_i\}) \cup \{C_{i1}, C_{i2}\}$
6: **end while**
7: **return** C as the final set of clusters

Divisive Hierarchical Clustering

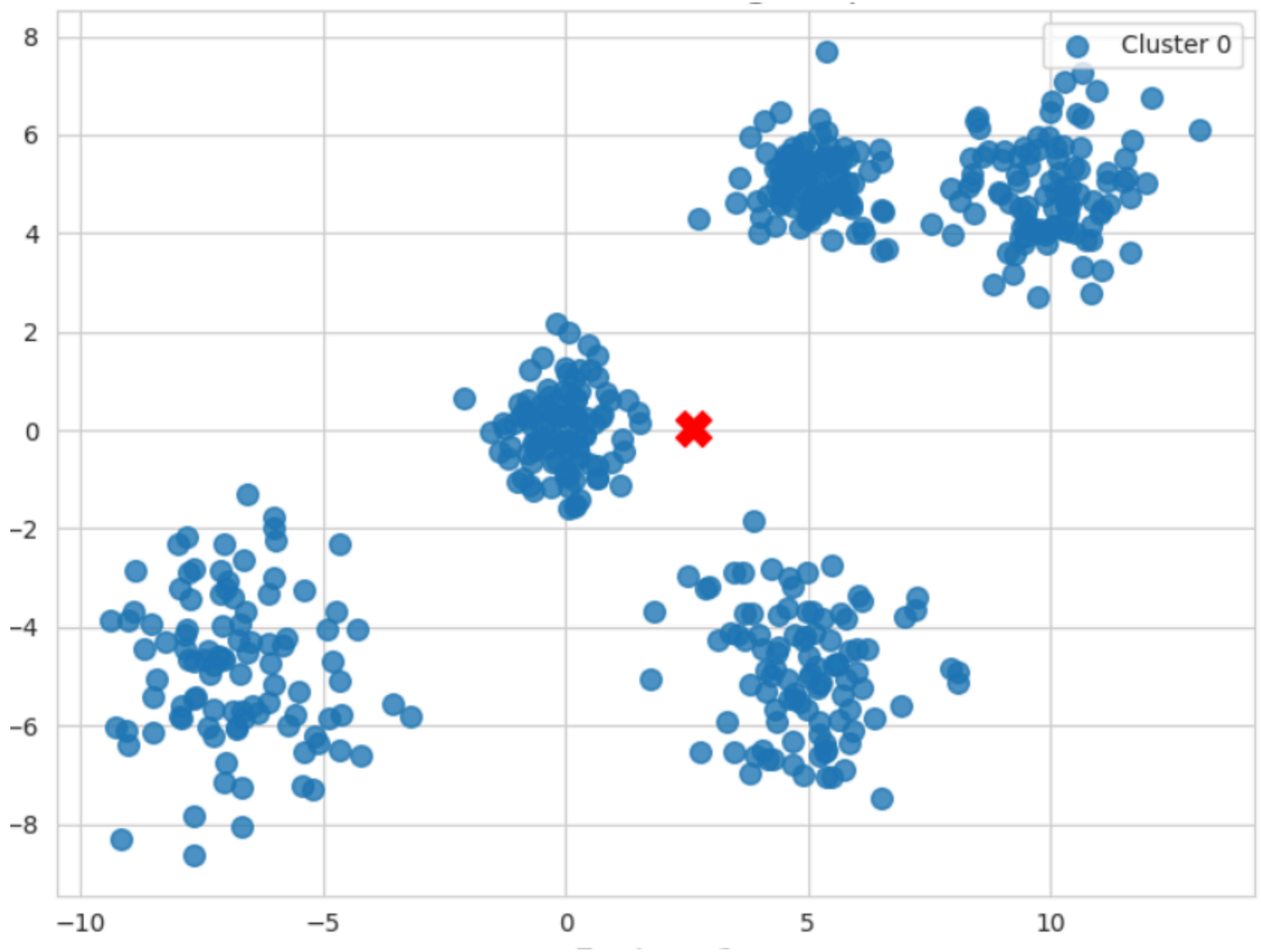
- There are **three key aspects** that must be addressed when designing a divisive hierarchical clustering algorithm:

  Splitting Criterion:
  - A common criterion for evaluating splits is the **Sum of Squared Errors (SSE),** which helps to minimize the within-cluster variance.

  Splitting Method:
  - Evaluate every possible way of dividing the data points **into two clusters**.
  - One practical and popular heuristic is to use the **bisecting k-means** method, where **k = 2.**

  Cluster Selection for Splitting:
  - Which cluster should we split next?
  - A simple yet effective approach is to choose the **loosest cluster**, i.e., the cluster with the **largest average SSE**.

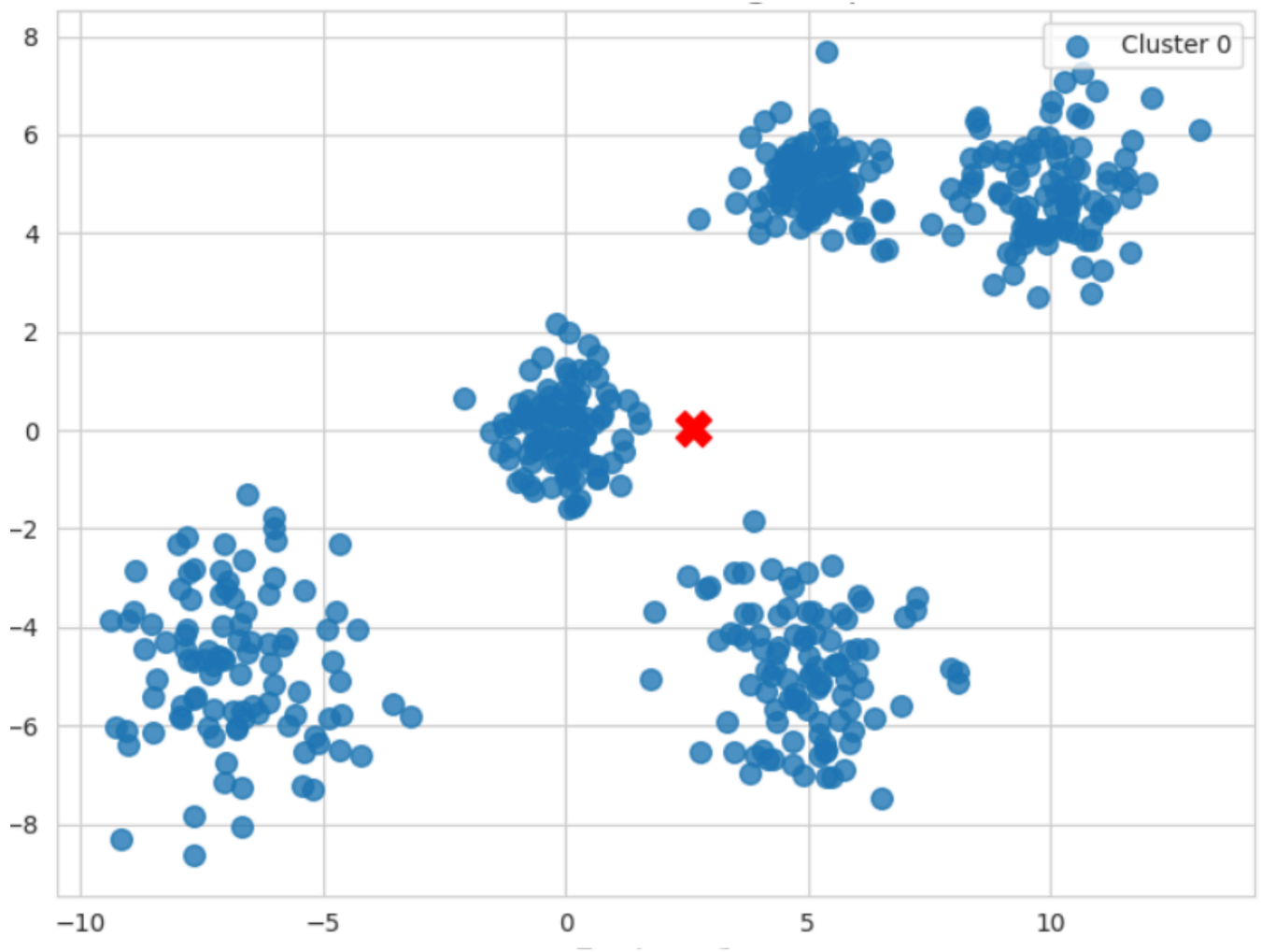
Divisive Hierarchical Clustering



The SSE for a cluster **C** is defined as:

$$SSE = \sum_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{m}_C\|^2$$

Divisive Hierarchical Clustering

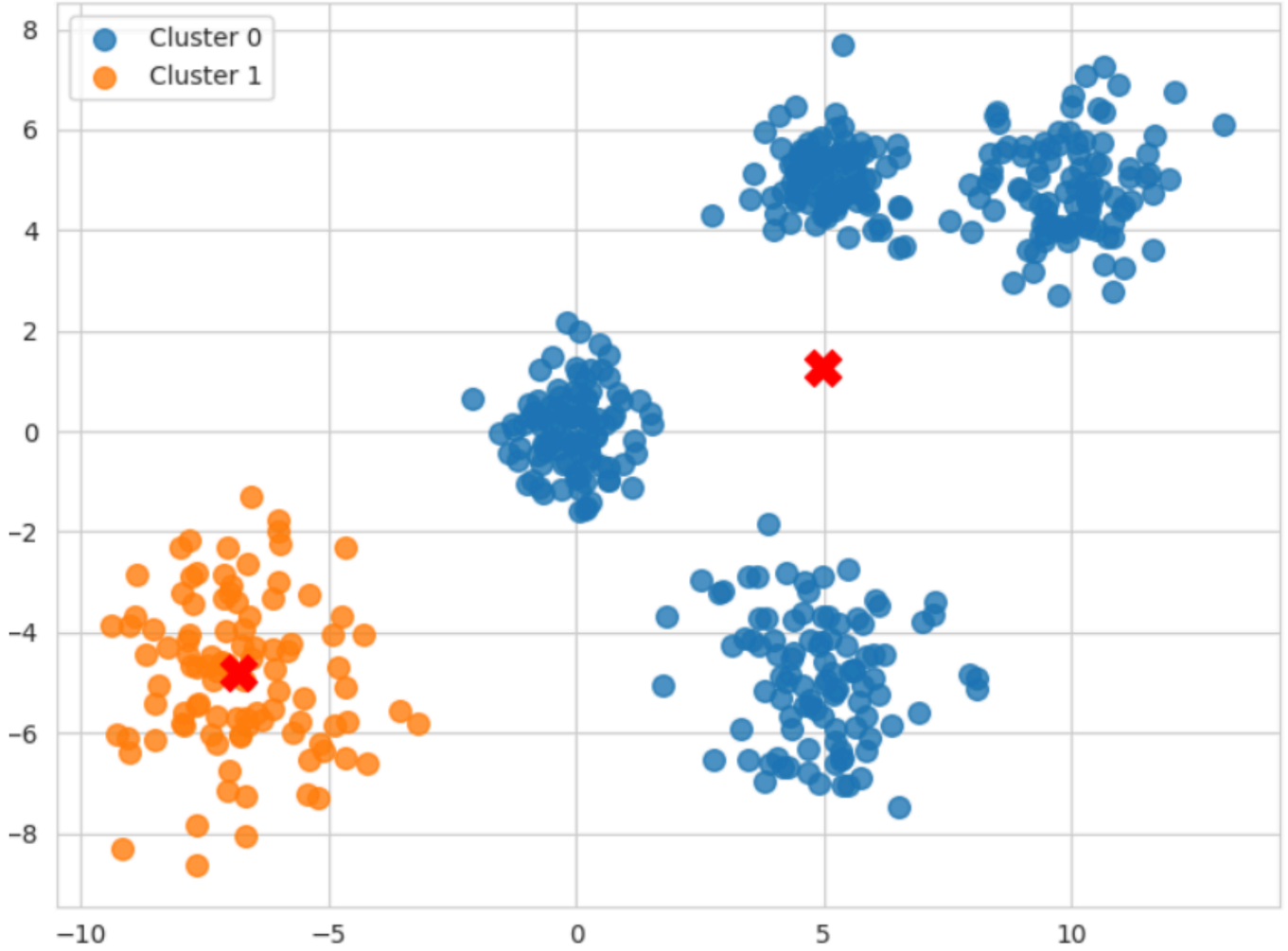


Formally, for each cluster C_i, we compute the average SSE:

$$E_{C_i} = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{m}_i\|^2$$

Cluster SSE scores: 53.69

Divisive Hierarchical Clustering



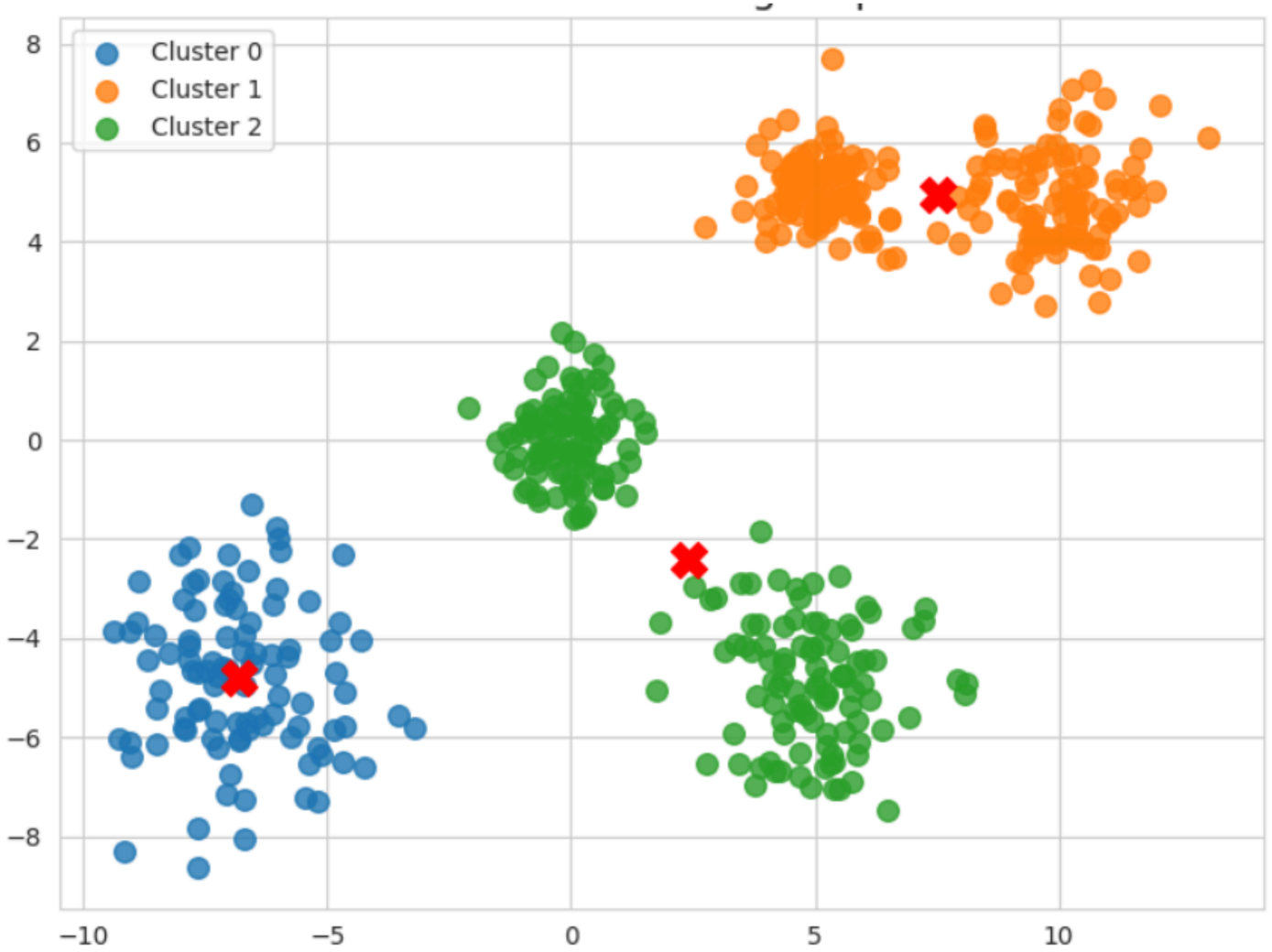
Use the bisecting k-means method, where k = 2.

$$E_{C_i} = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{m}_i\|^2$$

Cluster SSE scores (0): 31.0

Cluster SSE scores (1): 4.09

Divisive Hierarchical Clustering



Use the bisecting k-means method, where k = 2.

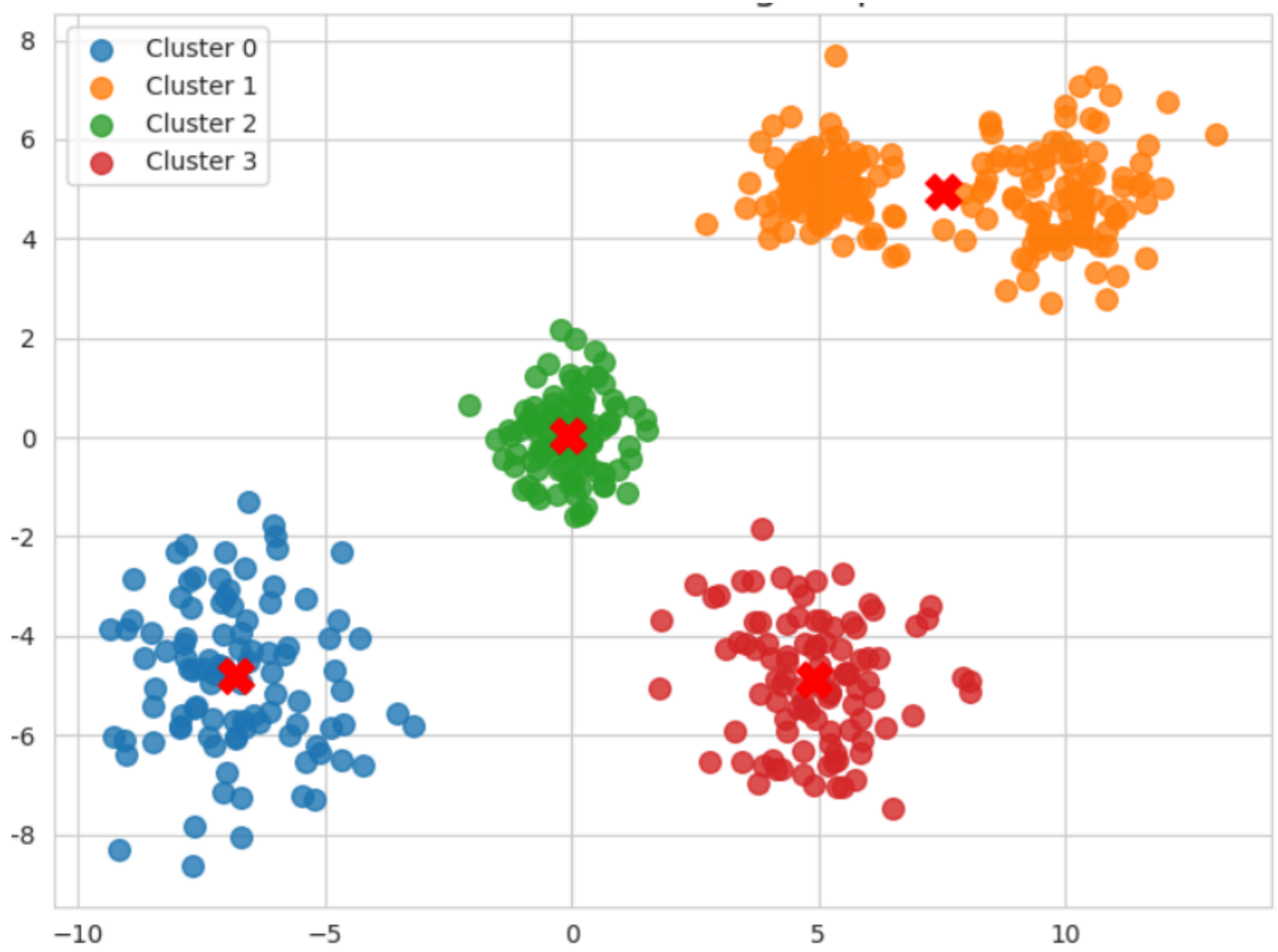
$$E_{C_i} = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mathbf{m}_i\|^2$$

Cluster SSE scores (0): 4.09

Cluster SSE scores (1): 7.38

Cluster SSE scores (2): 14.31

Divisive Hierarchical Clustering



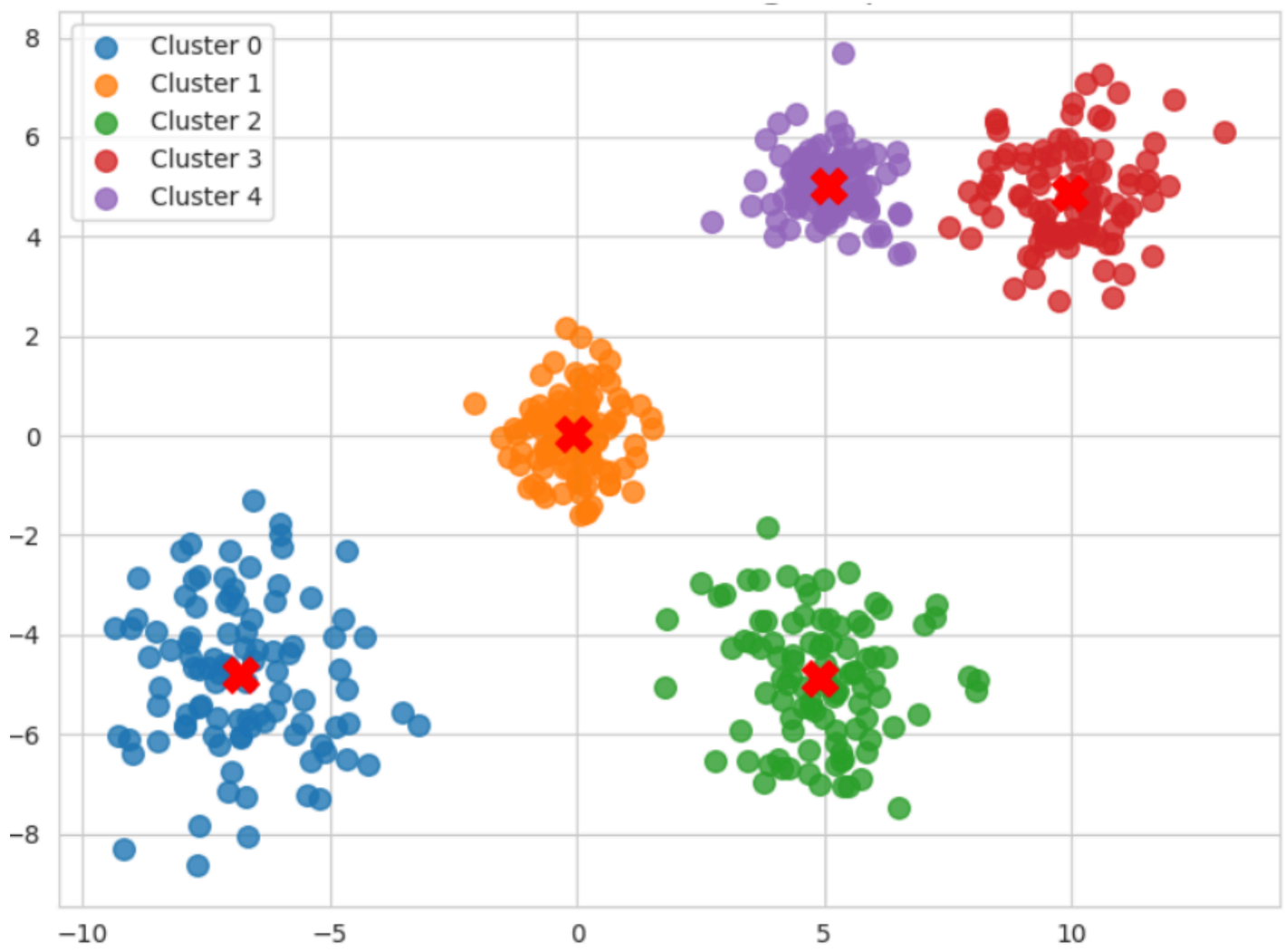
Cluster SSE scores (0): 4.09

Cluster SSE scores (1): 7.38

Cluster SSE scores (2): 1.09

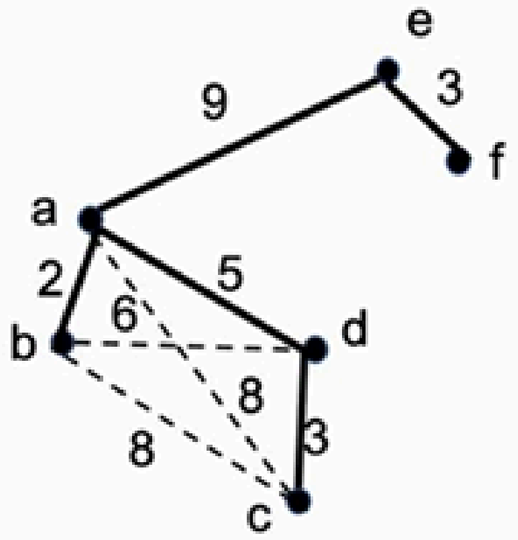
Cluster SSE scores (3): 2.96

Divisive Hierarchical Clustering



Minimum Spanning Tree-based Approach

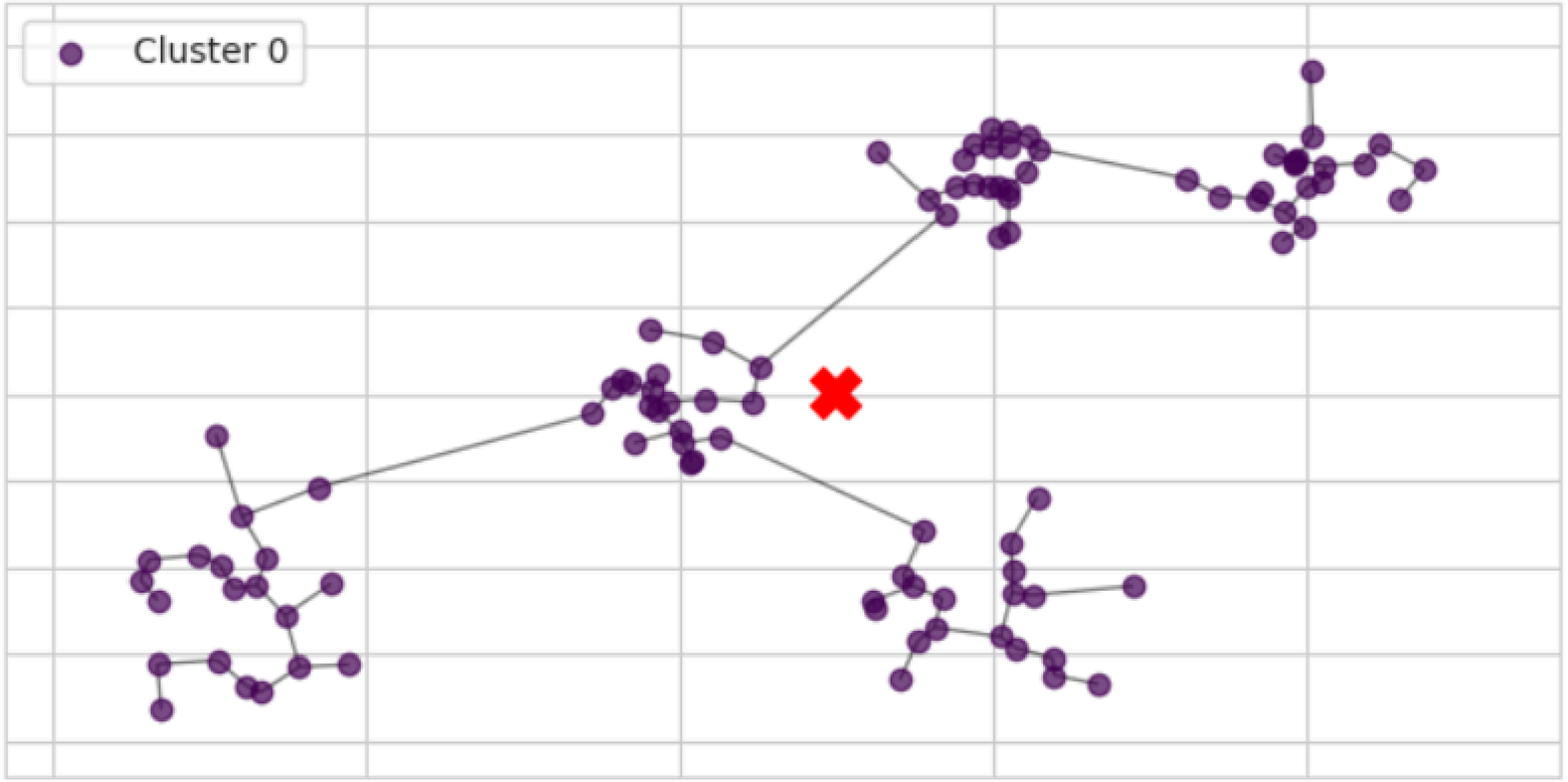
- Based on the concept of the **minimum spanning tree (MST)**
- In a weighted graph G, a minimum spanning tree is an acyclic subgraph of G that contains all nodes in G, and the sum of edge weights of the tree is minimized.



Minimum Spanning Tree-based Approach

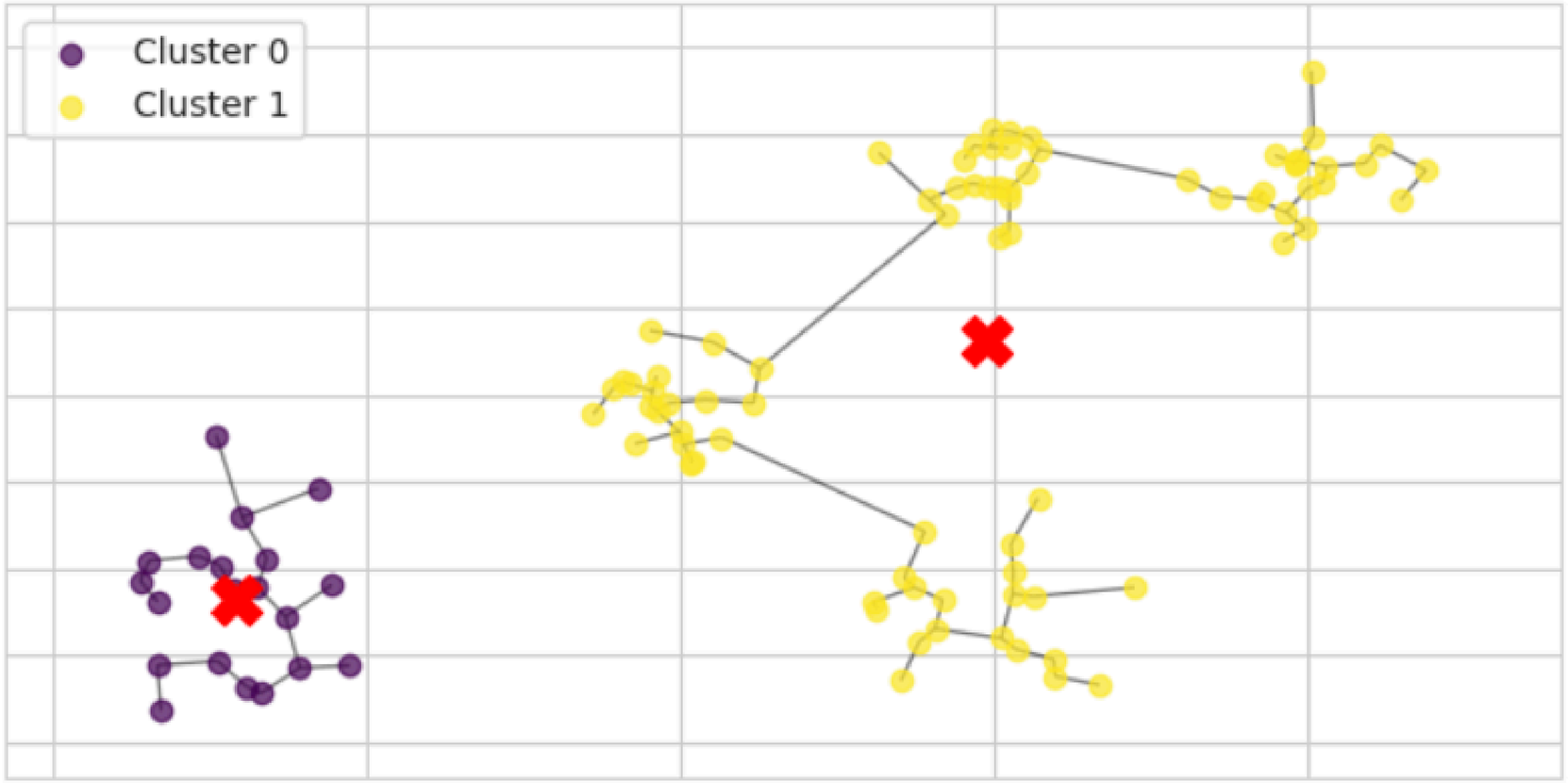
- To perform divisive hierarchical clustering using the MST, we follow these steps:
  - Construct a weighted graph where each node represents a data point, and the edge weights correspond to the distance between points
  - Compute the MST of the graph.
  - At each step, delete the edge with the largest weight in the MST. This will split the cluster into two sub-clusters.
  - Repeat the process of deleting the largest edge until each point is its own cluster or the desired number of clusters is reached.

Minimum Spanning Tree-based Approach



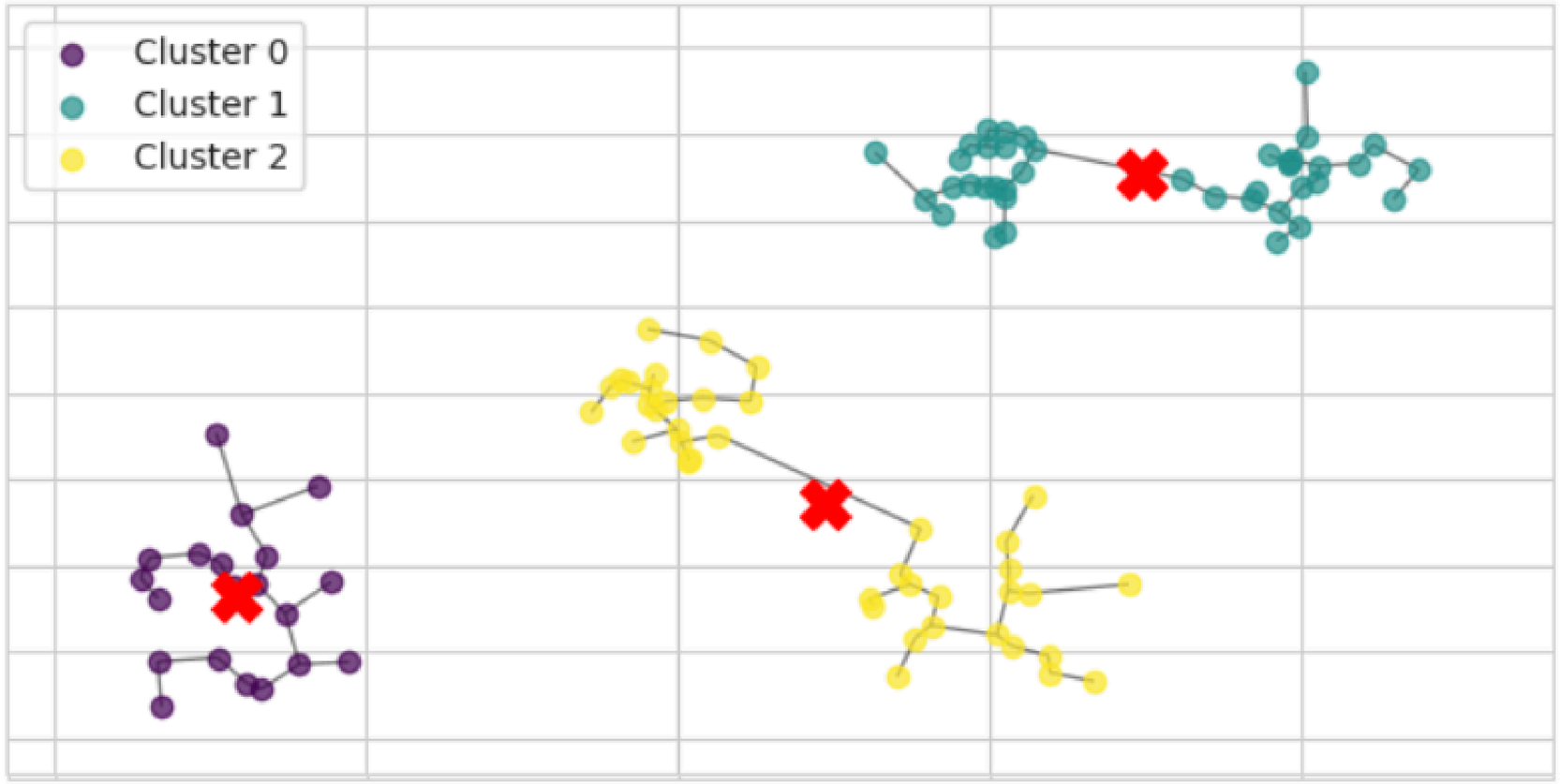
Compute the MST of the graph.

Minimum Spanning Tree-based Approach



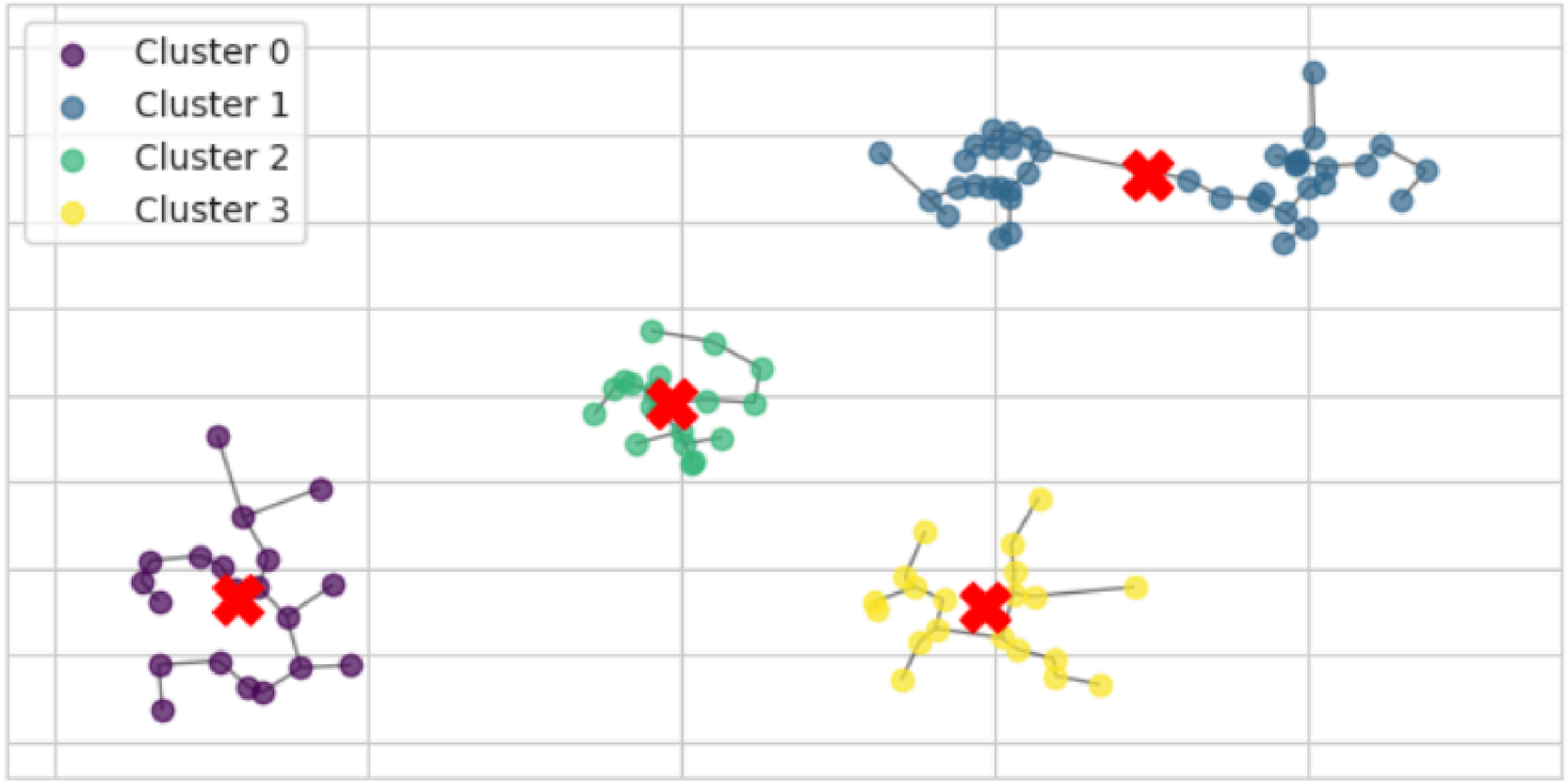
At each step, delete the edge with the largest weight in the MST. This will split the cluster into two sub-clusters.

Minimum Spanning Tree-based Approach



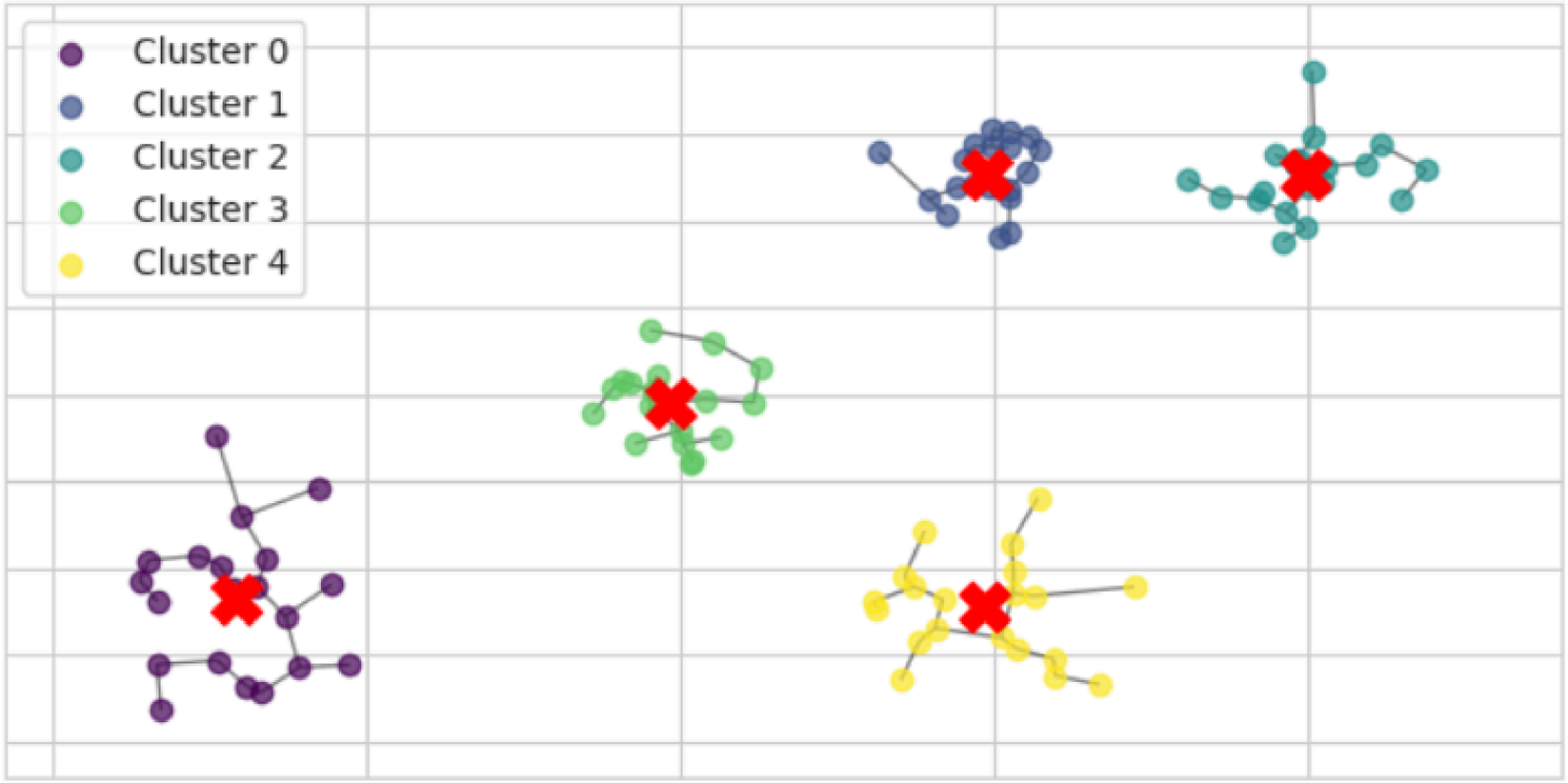
Repeat the process of deleting the largest edge until each point is its own cluster or the desired number of clusters is reached (5).

Minimum Spanning Tree-based Approach



Repeat the process of deleting the largest edge until each point is its own cluster or the desired number of clusters is reached (5).

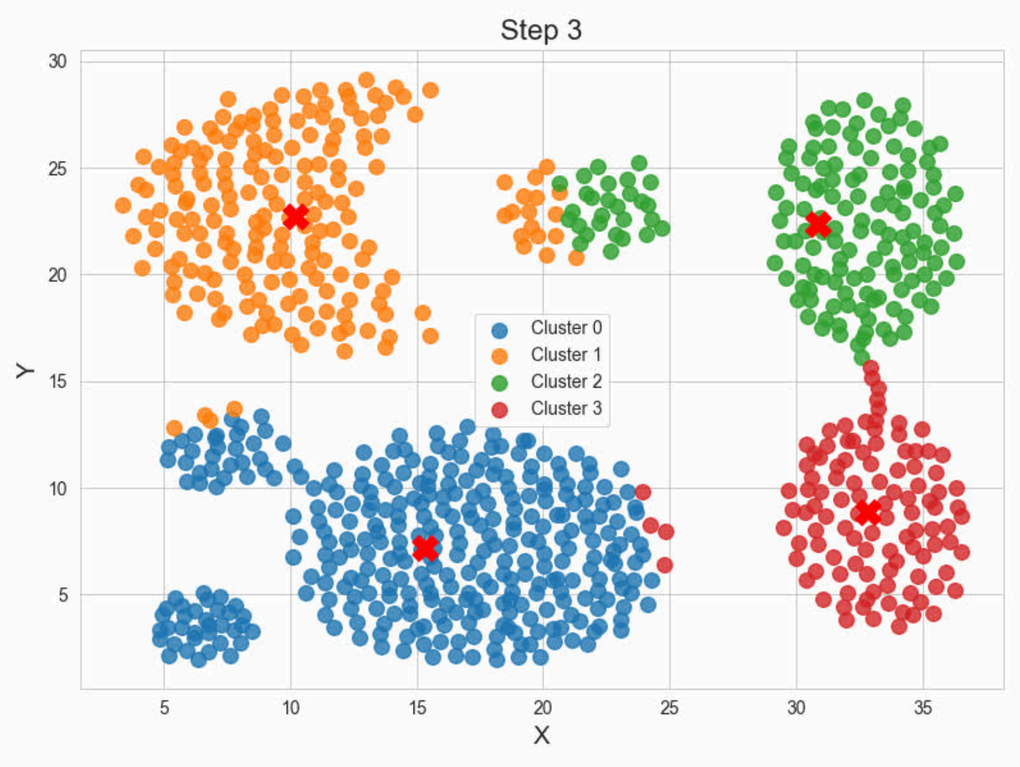
Minimum Spanning Tree-based Approach



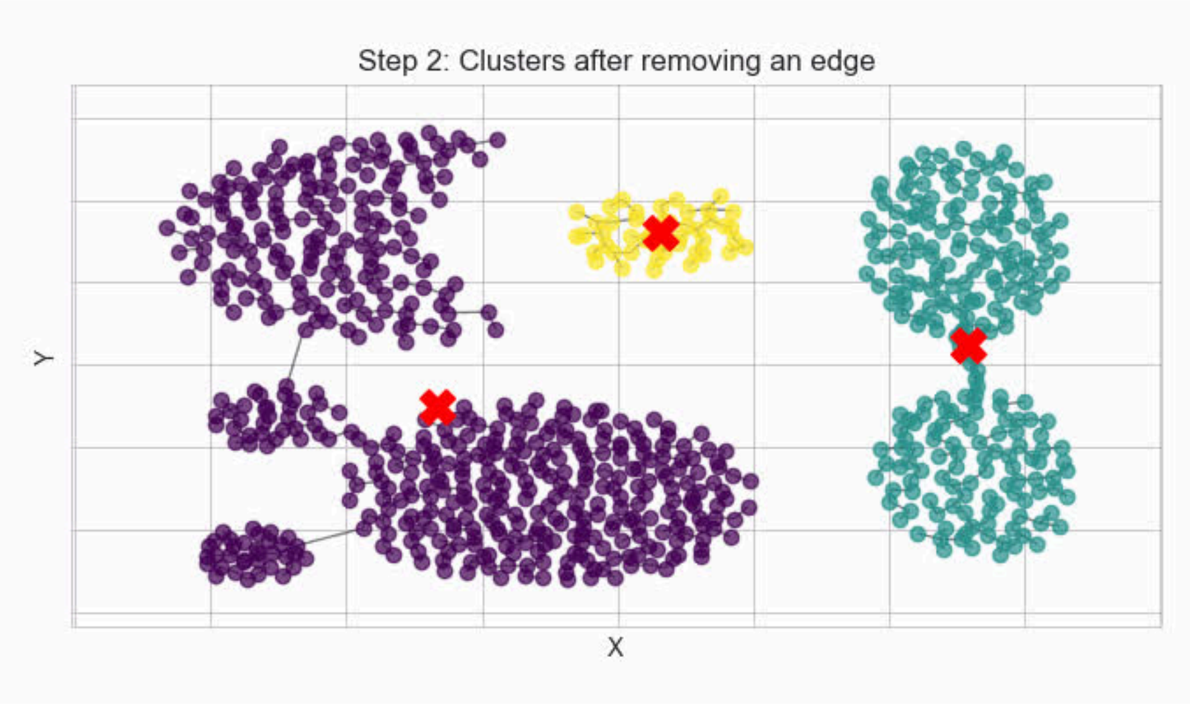
Repeat the process of deleting the largest edge until each point is its own cluster or the desired number of clusters is reached (5).

Experiments

BisectingKMeans

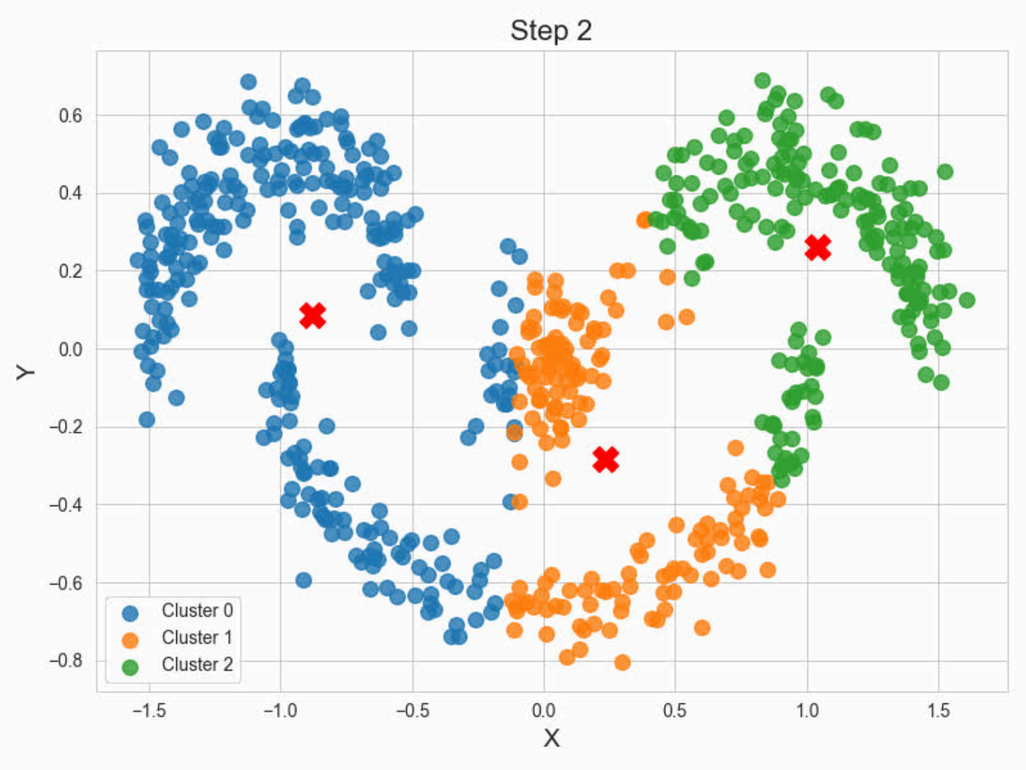


MST

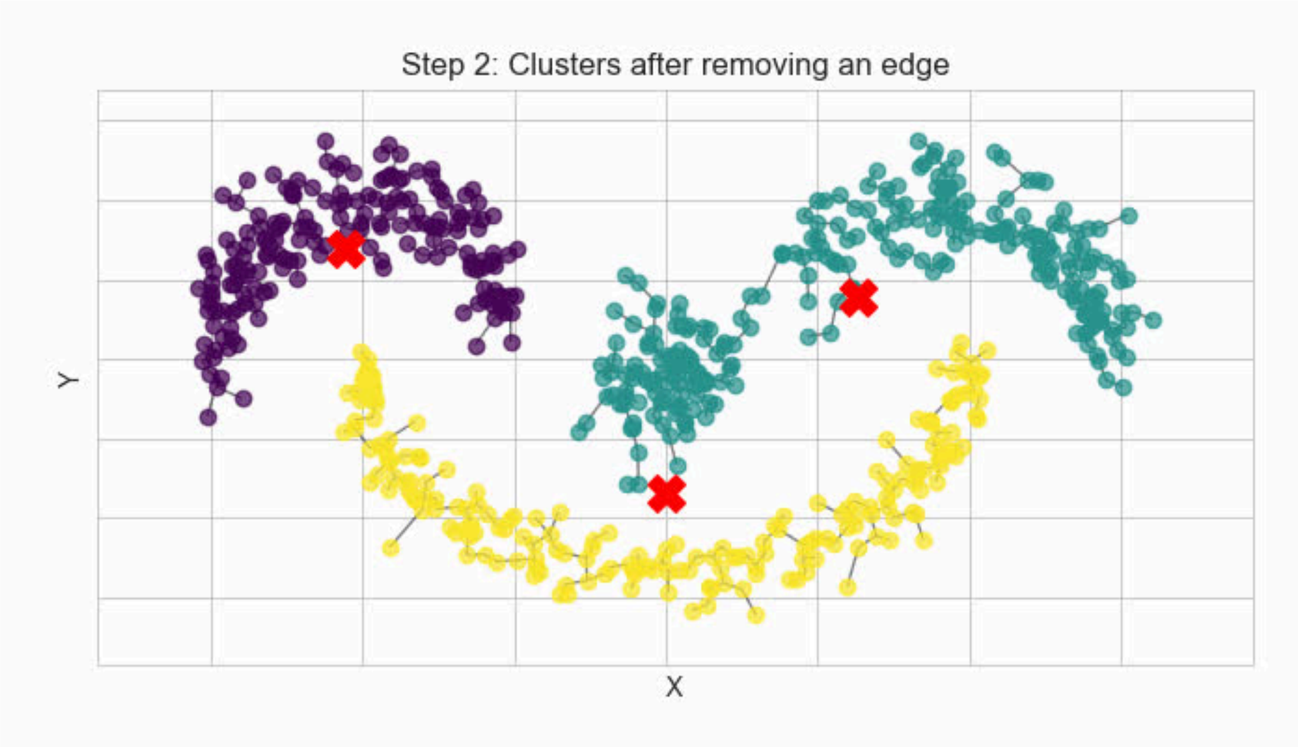


Experiments

BisectingKMeans

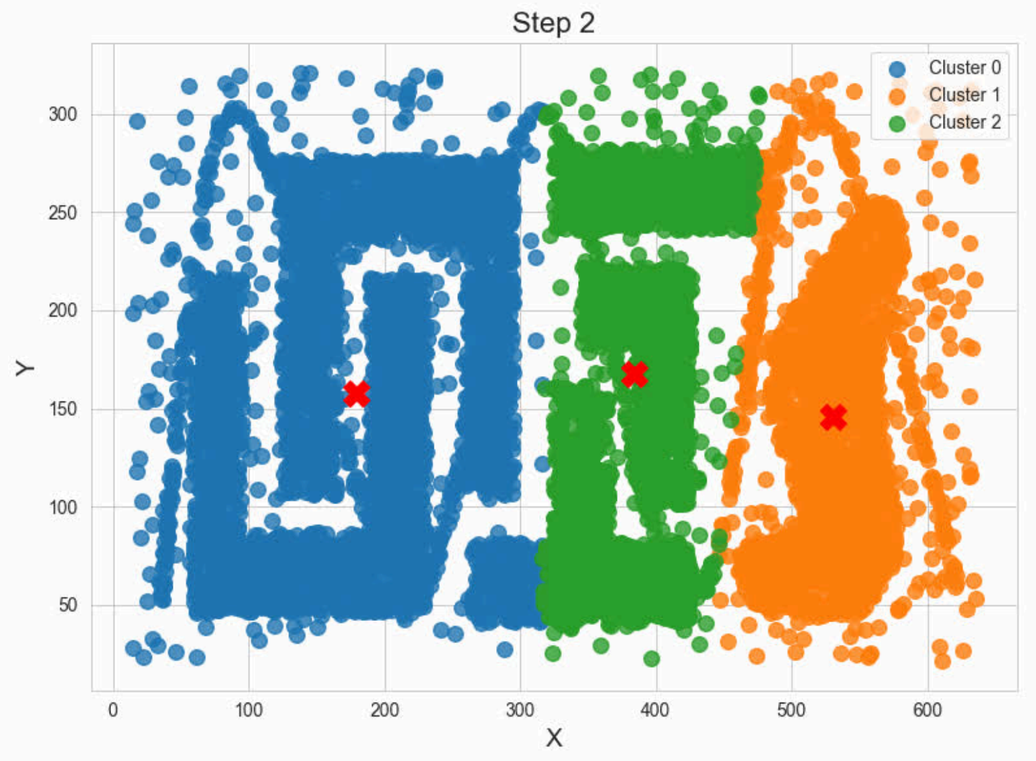


MST

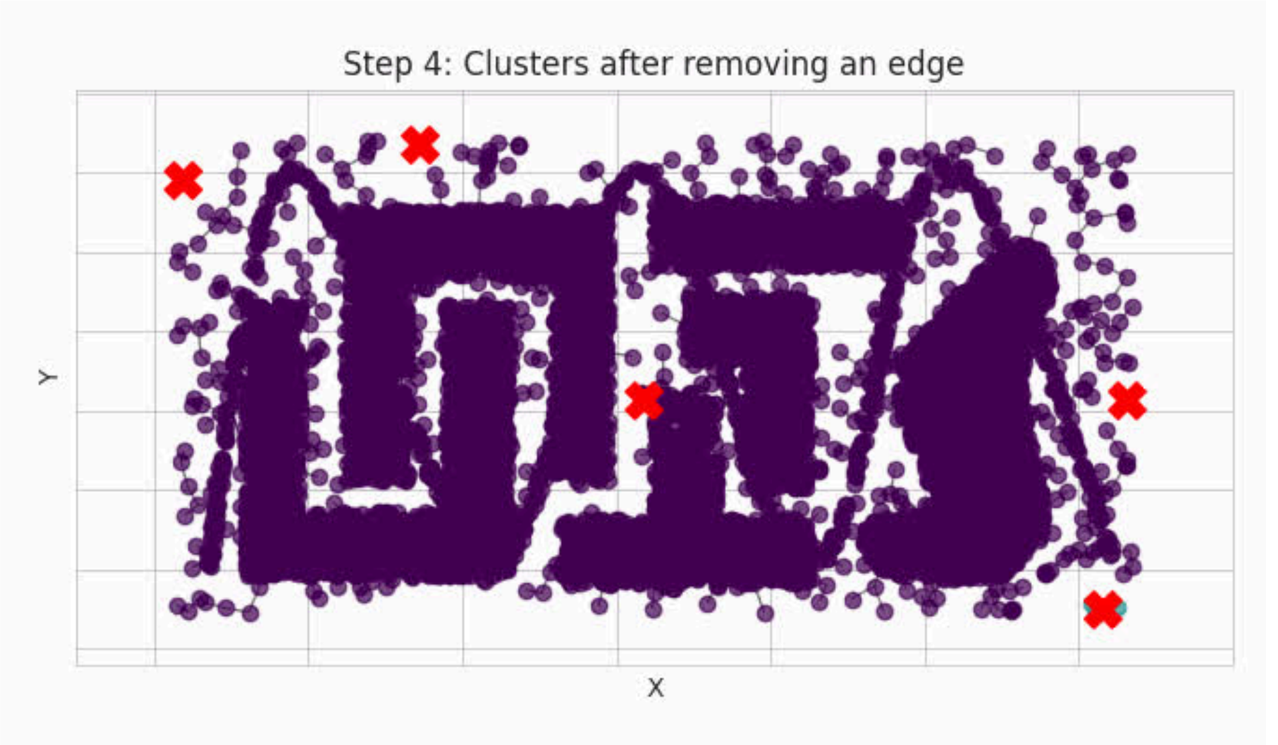


Experiments

BisectingKMeans



MST



BIRCH: Clustering Using Clustering Feature Trees

- BIRCH, developed by Tian Zhang, Raghu Ramakrishnan and Miron Livny in the year 1996.
- BIRCH is designed for clustering a **large amount** of **numeric data** by integrating hierarchical clustering and other clustering methods.
- BIRCH is base on the notation of CF (Clustering Feature) a CF Tree

BIRCH: Clustering Using Clustering Feature Trees

- The clustering feature (CF) of the cluster is a 3-D vector summarizing information about clusters of objects. It is defined as:

$$CF = (n, LS, SS)$$

  - n: Number of data points (d-dimensional data objects)
  - LS: Linear sum of points.
  - SS: Squared sum of points.

$$LS = \sum_{i=1}^{n} x_i \qquad SS = \sum_{i=1}^{n} x_i^2$$

BIRCH: Clustering Using Clustering Feature Trees

- Using a clustering feature, we can easily derive many useful statistics of a cluster. Some key statistic derivations:
  - Centroid: $x_0 = \frac{\sum_{i=1}^{n} x_i}{n} = \frac{LS}{n}$
  - Radius: $R = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x_0)^2}{n}} = \sqrt{\frac{SS}{n} - \frac{LS^2}{n^2}}$
  - Diameter: $D = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2}{n(n-1)}} = \sqrt{\frac{2nSS - 2LS^2}{n(n-1)}}$
- Here, R is the average distance from member objects to the centroid, and D is the average pairwise distance within a cluster.

BIRCH: Clustering Using Clustering Feature Trees

$$R = \sqrt{\frac{\sum_{i=1}^{n}(x_i - x_0)^2}{n}} = \sqrt{\frac{SS}{n} - \frac{LS^2}{n^2}}$$

$$\sum_{i=1}^{n}(x_i - x_0)^2 = \sum_{i=1}^{n}\left(x_i^2 - 2x_i x_0 + x_0^2\right) = \sum_{i=1}^{n} x_i^2 - 2x_0 \sum_{i=1}^{n} x_i + \sum_{i=1}^{n} x_0^2$$

$$\left.\begin{aligned} &\sum_{i=1}^{n} x_i^2 = SS \\ &2x_0 \sum_{i=1}^{n} x_i = 2\frac{LS}{n} \cdot LS = \frac{2LS^2}{n} \\ &\sum_{i=1}^{n} x_0^2 = n \cdot x_0^2 = n \cdot \left(\frac{LS}{n}\right)^2 = \frac{LS^2}{n} \end{aligned}\right\} \sum_{i=1}^{n}(x_i - x_0)^2 = SS - \frac{LS^2}{n}$$

$$\longrightarrow \quad \sqrt{\frac{\sum_{i=1}^{n}(x_i - x_0)^2}{n}} = \sqrt{\frac{SS}{n} - \frac{LS^2}{n^2}}$$

BIRCH: Clustering Using Clustering Feature Trees

$$D = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2}{n(n-1)}} = \sqrt{\frac{2nSS - 2LS^2}{n(n-1)}}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\left(x_i^2 - 2x_ix_j + x_j^2\right) = \sum_{i=1}^{n}\sum_{j=1}^{n}x_i^2 - 2\sum_{i=1}^{n}\sum_{j=1}^{n}x_ix_j + \sum_{i=1}^{n}\sum_{j=1}^{n}x_j^2$$

$$\left.\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=1}^{n}x_i^2 = n\sum_{i=1}^{n}x_i^2 = n\cdot SS \\
&\sum_{i=1}^{n}\sum_{j=1}^{n}x_ix_j = \left(\sum_{i=1}^{n}x_i\right)^2 = LS^2 \\
&\sum_{i=1}^{n}\sum_{j=1}^{n}x_j^2 = n\sum_{j=1}^{n}x_j^2 = n\cdot SS
\end{aligned}\right\} \sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2 = 2nSS - 2LS^2$$

$$\longrightarrow \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i - x_j)^2}{n(n-1)}} = \sqrt{\frac{2nSS - 2LS^2}{n(n-1)}}$$

BIRCH: Clustering Using Clustering Feature Trees

- Moreover, clustering features are additive.

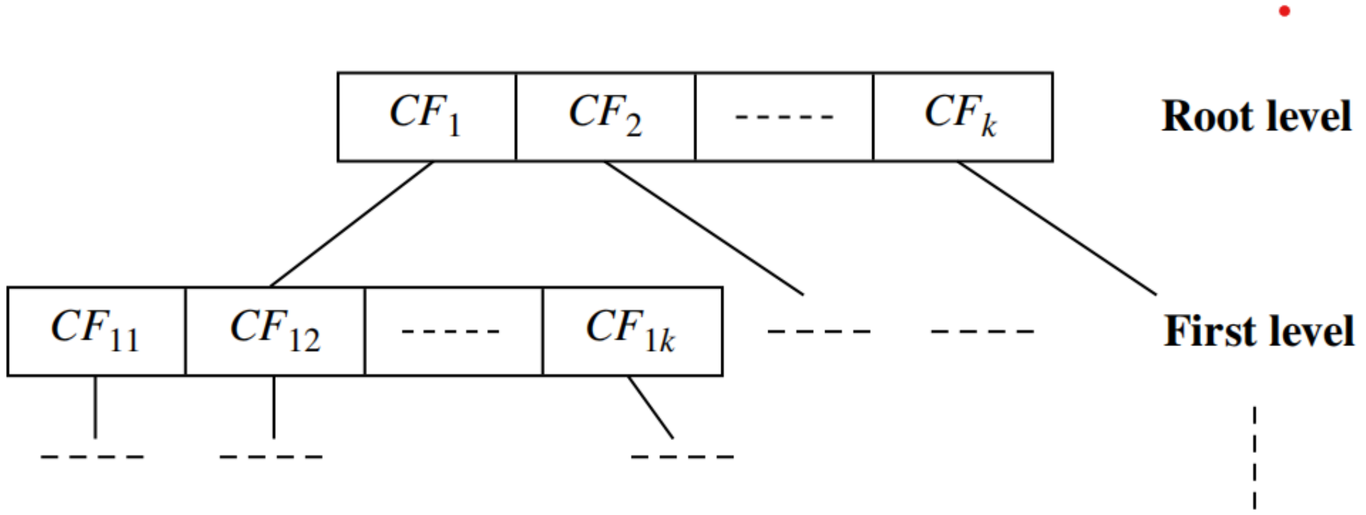
$$CF_1 = (n_1, LS_1, SS_1) \qquad CF_2 = (n_2, LS_2, SS2)$$

$$CF_{12} = CF_1 + CF_2 = (n_1 + n_2, LS_1 + LS_2, SS_1 + SS_2)$$

- The clustering feature for the cluster that formed by merging C1 and C2 is simply.

BIRCH: Clustering Using Clustering Feature Trees

- A CF-tree is a height-balanced tree that stores the clustering features for a hierarchical clustering.
- In a CF-tree, each non-leaf node has "children" or descendants, and it stores the sum of the CFs of these children, providing a summary of their clustering information.

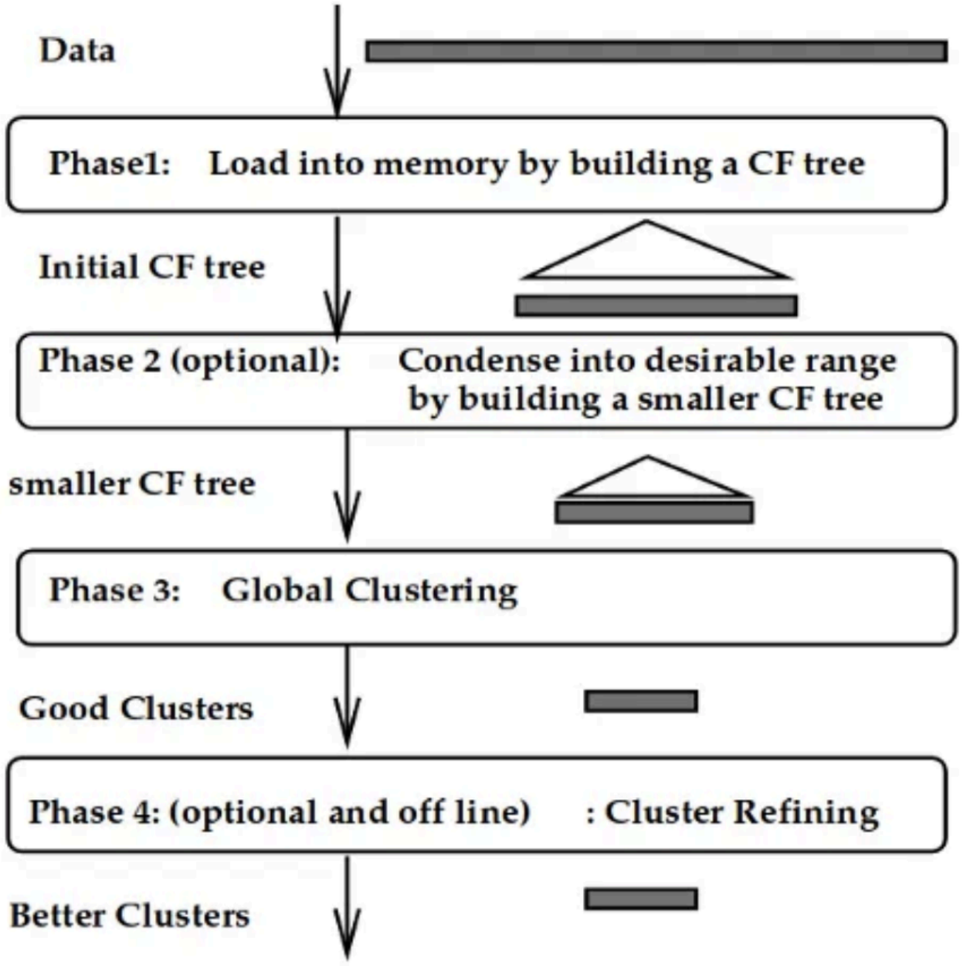


BIRCH: Clustering Using Clustering Feature Trees

- A CF-tree has two parameters: branching factor(B) and threshold(T).
  - The branching factor specifies the maximum number of children per nonleaf node.
  - The threshold parameter specifies the maximum diameter of subclusters stored at the leaf nodes of the tree.

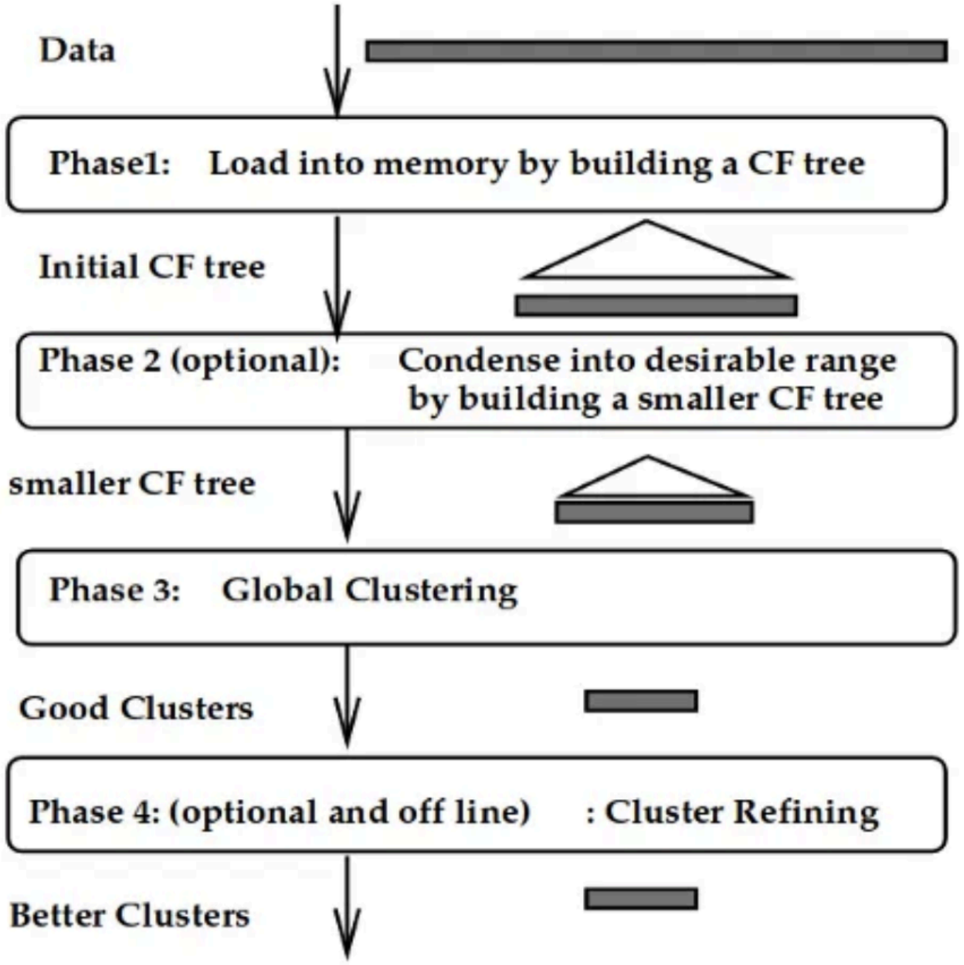
BIRCH: Clustering Using Clustering Feature Trees

- Phase 1:
  - The algorithm scans the data, inserts points into a CF-tree, and rebuilds it with a higher threshold if memory is insufficient. Choosing an optimal threshold reduces rebuilds.



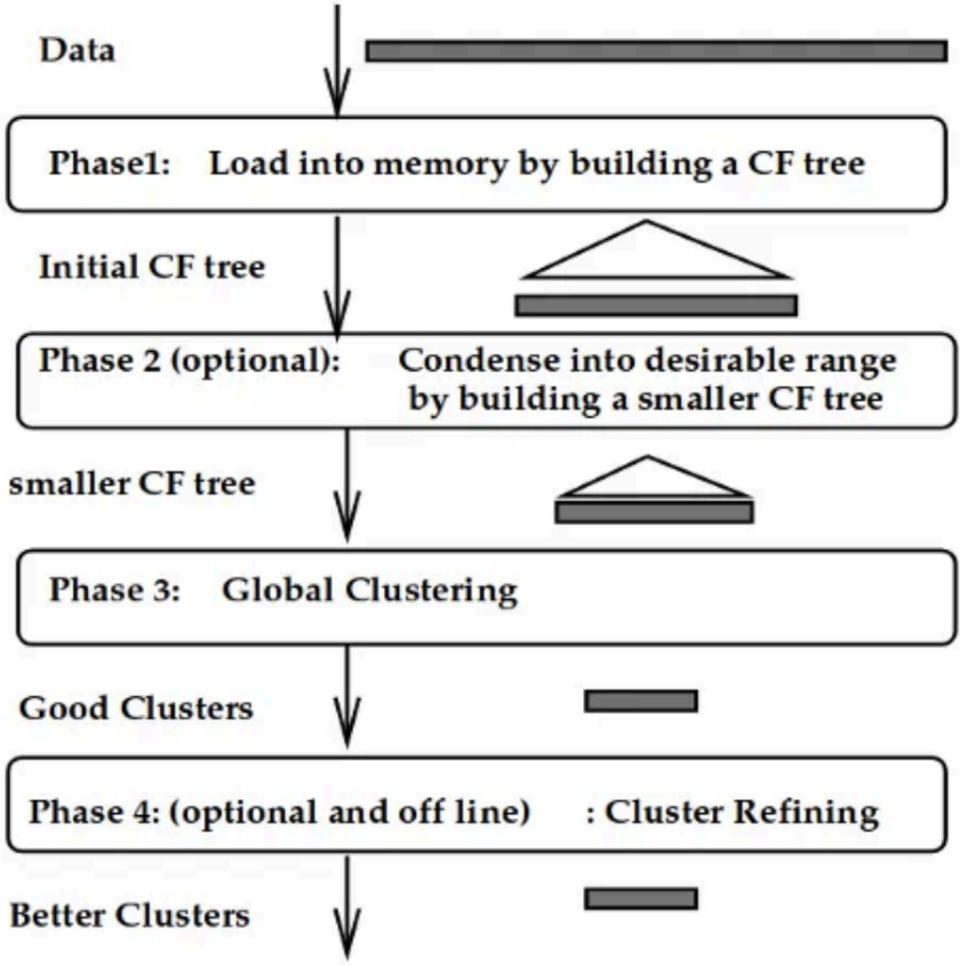
BIRCH: Clustering Using Clustering Feature Trees

- Phase 2:
  Crowded subclusters are grouped into larger clusters to create a more compact CF-tree.



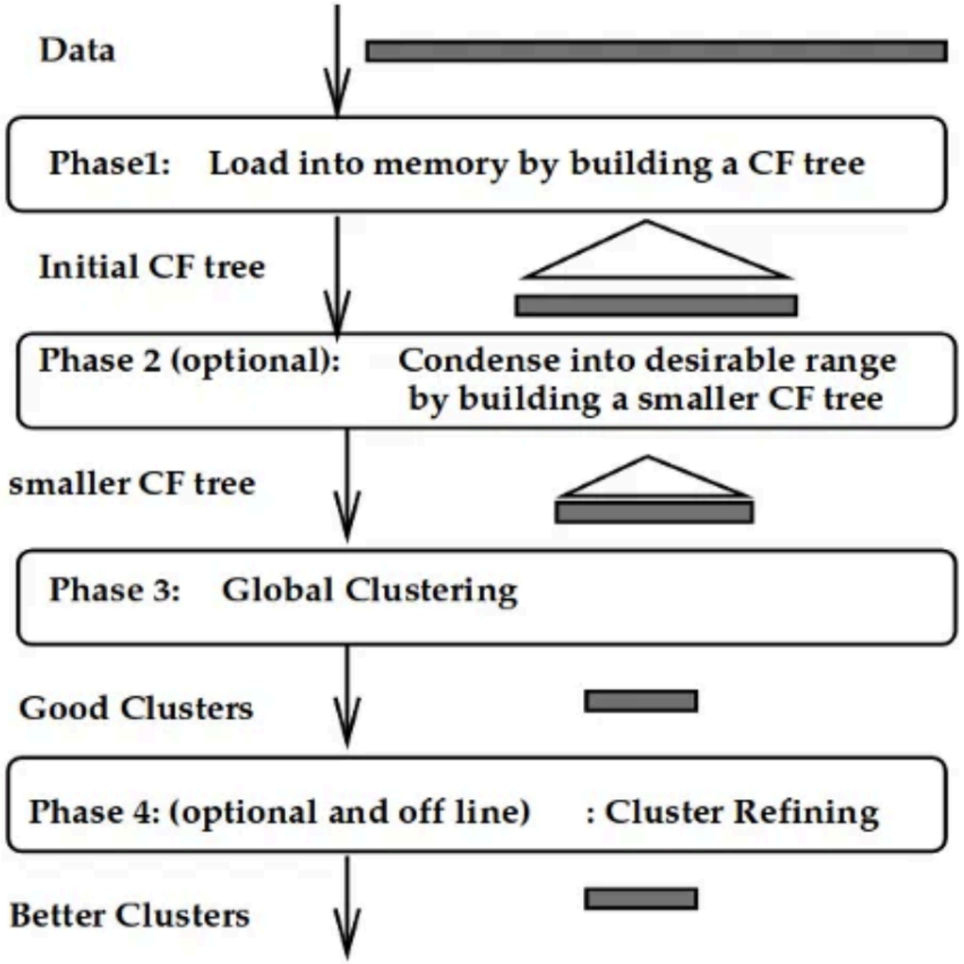
BIRCH: Clustering Using Clustering Feature Trees

- Phase 3:
  A clustering algorithm is applied to clustering features (CFs) rather than data points, minimizing I/O operations.



BIRCH: Clustering Using Clustering Feature Trees

- Phase 4:
  Additional passes refine clustering accuracy and offer the option to discard outliers by correcting course summaries from previous phases.



BIRCH: Clustering Using Clustering Feature Trees

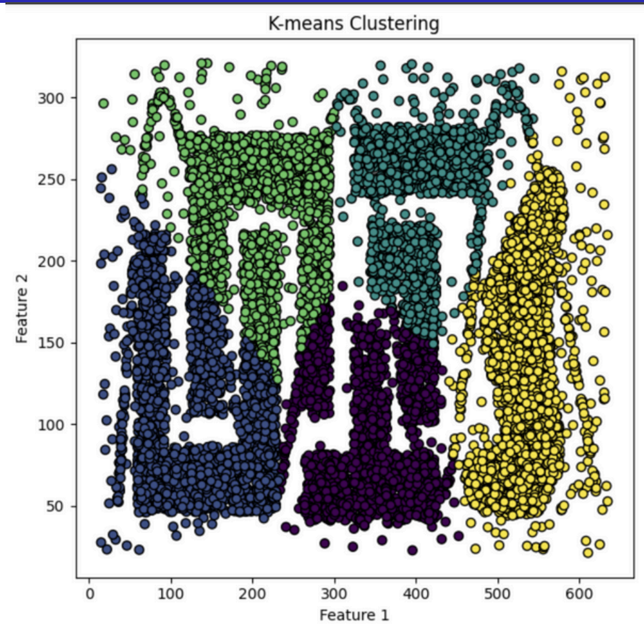
Advantages:

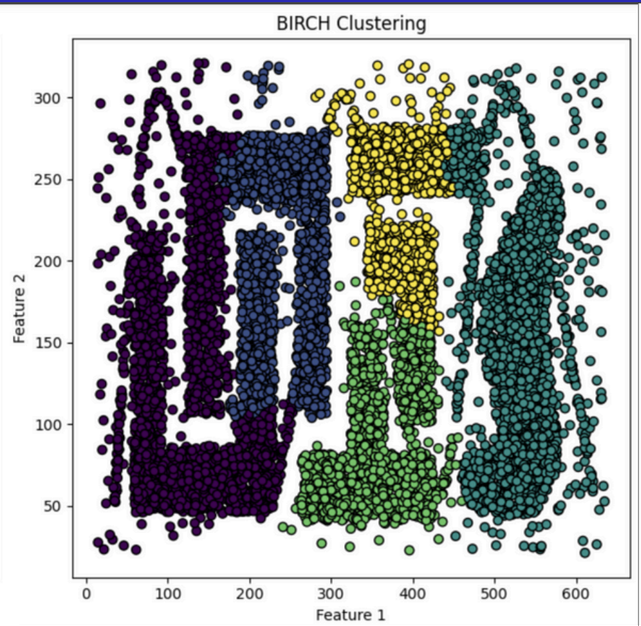
- Efficiently processes **large datasets** with a single scan and optional additional scans.
- Can handle incoming data and adjust clusters without rebuilding from scratch.
- The most important advantage of this algorithm is that its time complexity is only **O(n)**, where n is number of object.

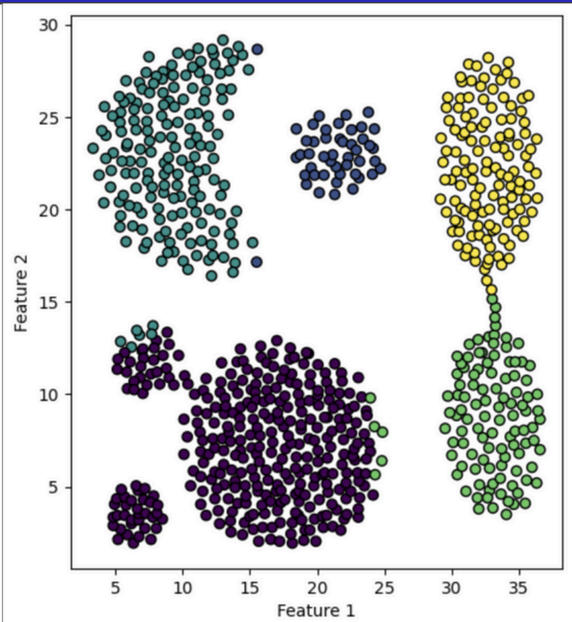
Disadvantages:

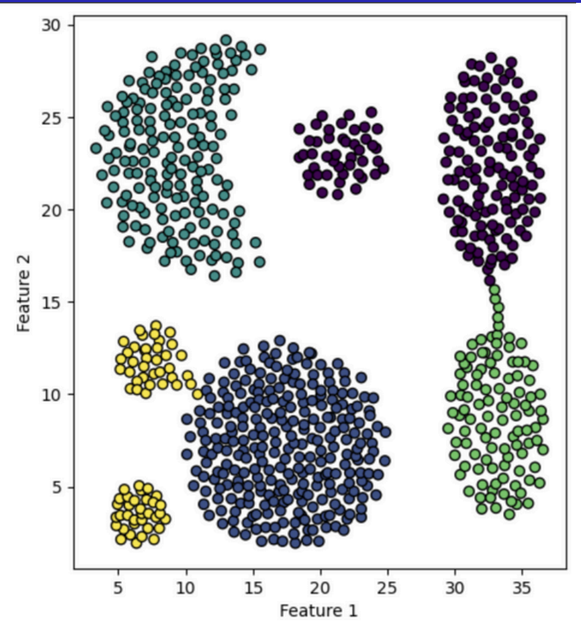
- Does not perform well for **non-spherical clusters** due to the use of **radius/diameter**.
- CF-tree nodes may not always correspond to natural clusters.

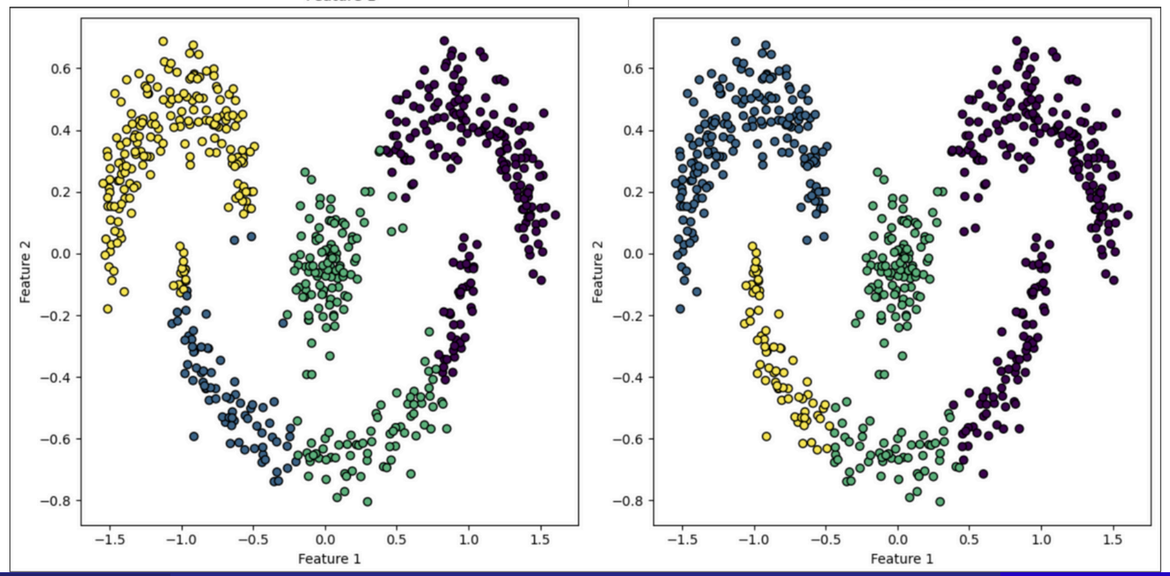
Experiment


K-means Clustering
Feature 1
Feature 2


BIRCH Clustering
Feature 1
Feature 2


Feature 1
Feature 2


Feature 1
Feature 2


Feature 1
Feature 2

Feature 1
Feature 2

Chameleon Clustering

- Traditional method often ignore the **interconnecity** of item of two cluster or **closesness** of two cluster.

  Closesness:
  - Within cluster: how close individual data points are to one another.
  - Between cluster: how close of two cluster.

  Interconnectity:
  - Within cluster: how related individual data points are to one another.
  - Between cluster: how related of two cluster.

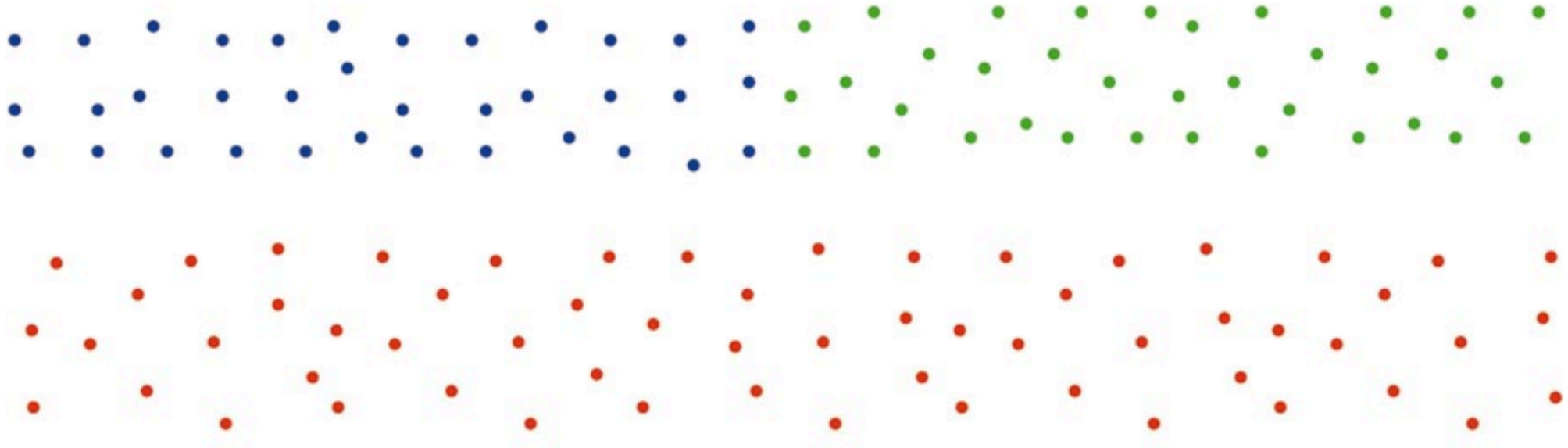
Chameleon Clustering

- Traditional method often ignore the **interconnecity** of item of two cluster or **closesness** of two cluster.

Algorithm incorrectly merge blue and green instead of merging red and cyan cluster

Chameleon Clustering

- Traditional method often ignore the **interconnecity** of item of two cluster or **closesness** of two cluster.

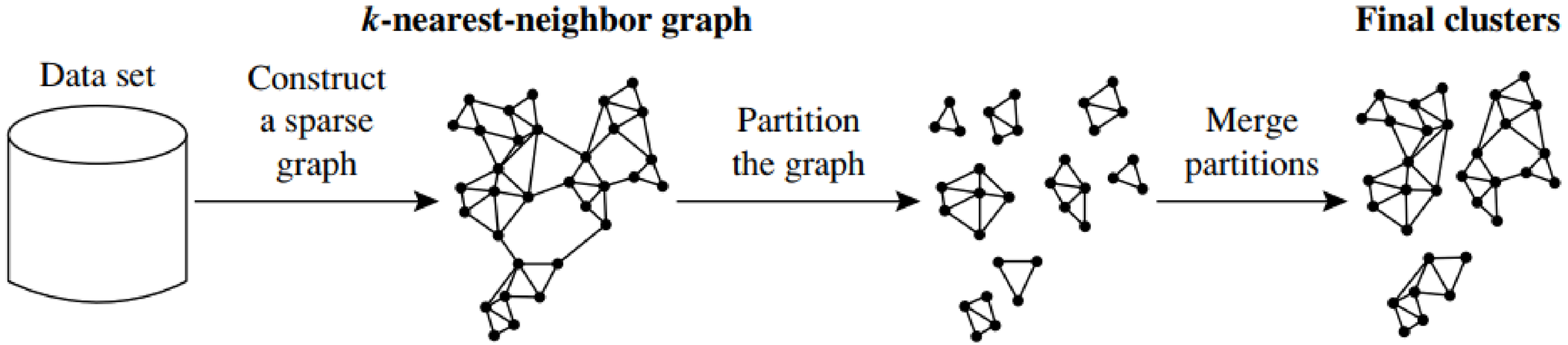


Algorithm incorrectly merge blue and red instead of merging blue and green cluster

Chameleon Clustering

- Traditional method often ignore the **interconnecity** of item of two cluster or **closesness** of two cluster.

- Chameleon is an algorithm that uses **graph-based techniques** and incorporates both **closeness and interconnectivity** to determine whether to merge two clusters.



Chameleon Clustering

- **Relative interconnecity (RI):**

$$RI = \frac{EC(C_i, C_j)}{\frac{1}{2}(EC(C_i) + EC(C_j))}$$

where is $EC(C_i, C_j)$ total weights of the edges that connect two clusters, $EC(C_i)$ is the total weights of the edges that connect two clusters that be bisec cutted of clusters C_i

Chameleon Clustering

- **Relative closeness (RC)**:

$$RC(C_i, C_j) = \frac{S_{EC(C_i,C_j)}}{\frac{m_i}{m_i+m_j} S_{EC(C_i)} + \frac{m_j}{m_i+m_j} S_{EC(C_i)}},$$

where is $S_{EC(C_i,C_j)}$ average weights of the edges that connect two clusters, $S_{EC(C_i)}$ is the total weights of the edges that connect two clusters that be bisec cutted of clusters C_i

Chameleon Clustering

Algorithm 3 Chameleon Clustering with Stopping Conditions

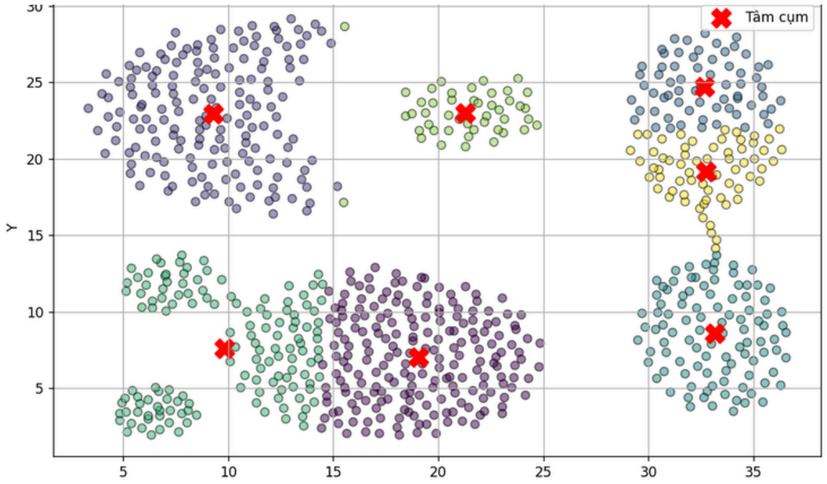
Input: Data D, number of nearest neighbors K
Output: Clusters $C_1, C_2, \ldots, C_k$

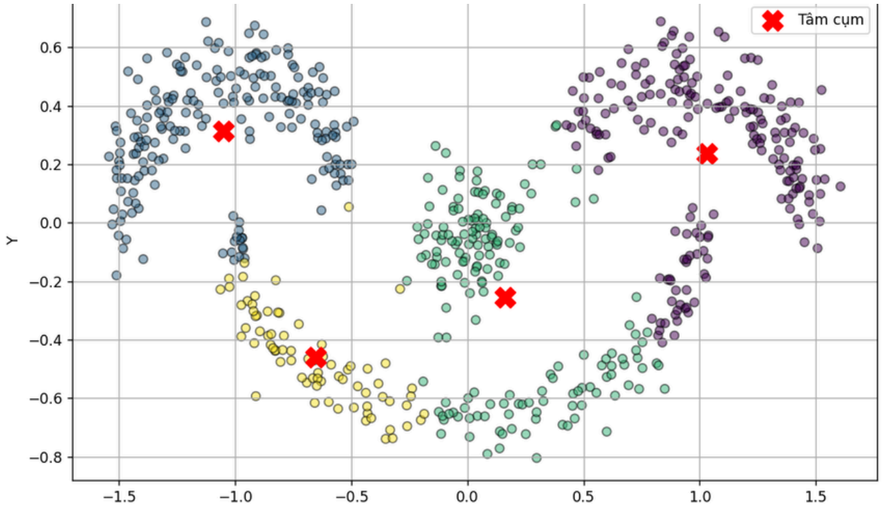
Create a sparse graph G from the data D:
for each point $x_i \in D$ **do**
  Find K nearest neighbors of x_i
  Add edges to the graph
end for

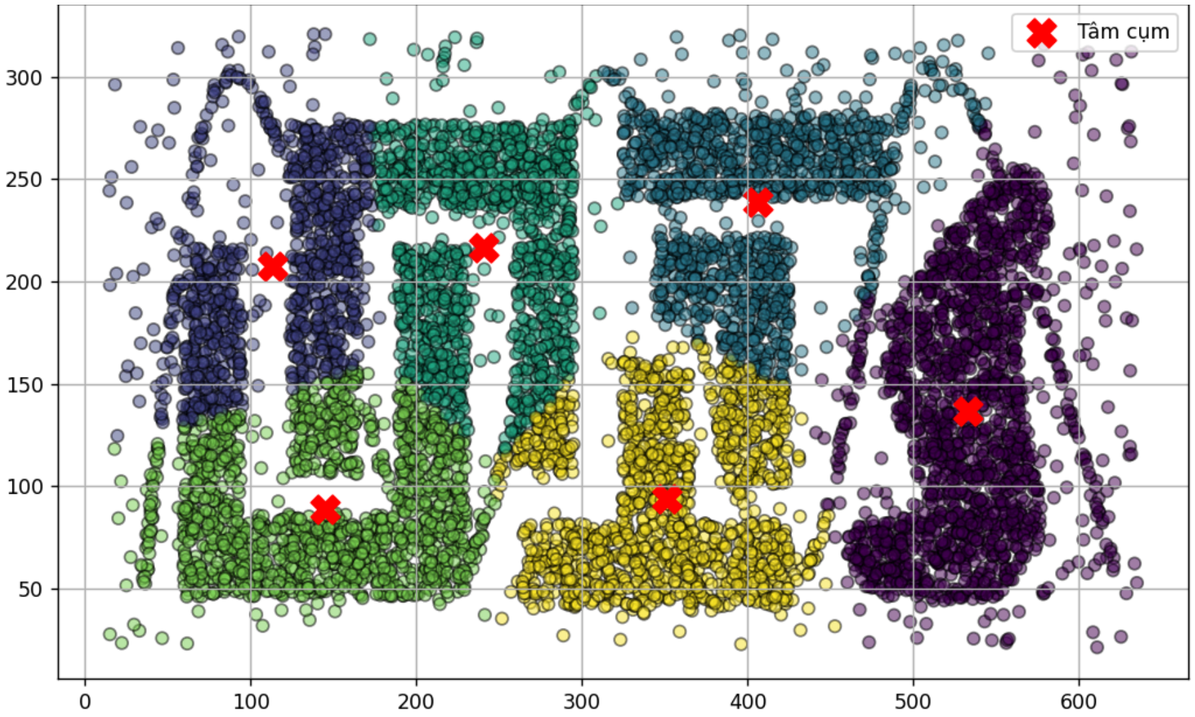
Partition the graph into connected components:
$Clusters \leftarrow$ MinimizeCut(G)

Agglomerative merge
while there are clusters to merge **do**
  Create the relative interconnectivity matrix RI for the clusters
  Create the relative closeness matrix SC for the clusters
  Select the pair of clusters (C_i, C_j) with $\max(RI(C_i, C_j), SC(C_i, C_j))$
  if $\max(RI(C_i, C_j), SC(C_i, C_j))$ is sufficiently high **then**
    Merge C_i and C_j
    Update the interconnectivity and closeness matrices
  else
    Break: No clusters meet the merging criteria
  end if
  if number of clusters reaches desired threshold **then**
    Break: Sufficient number of clusters
  end if
end while
return *Clusters*

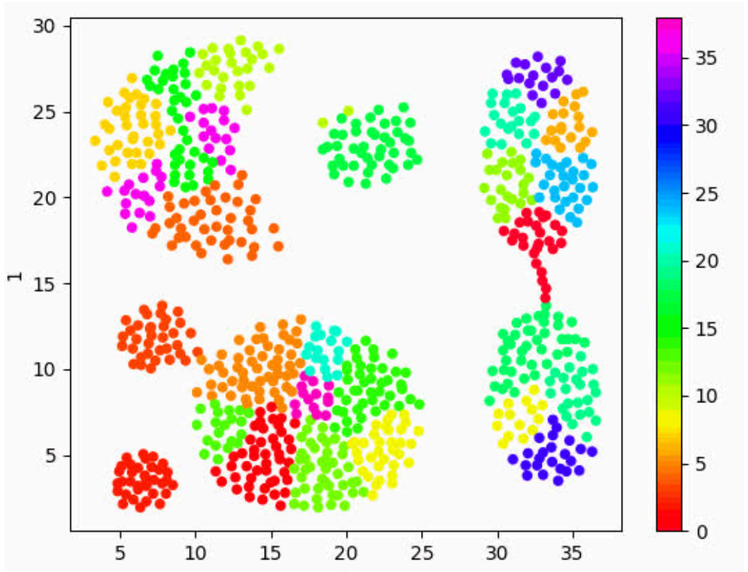
Experiments

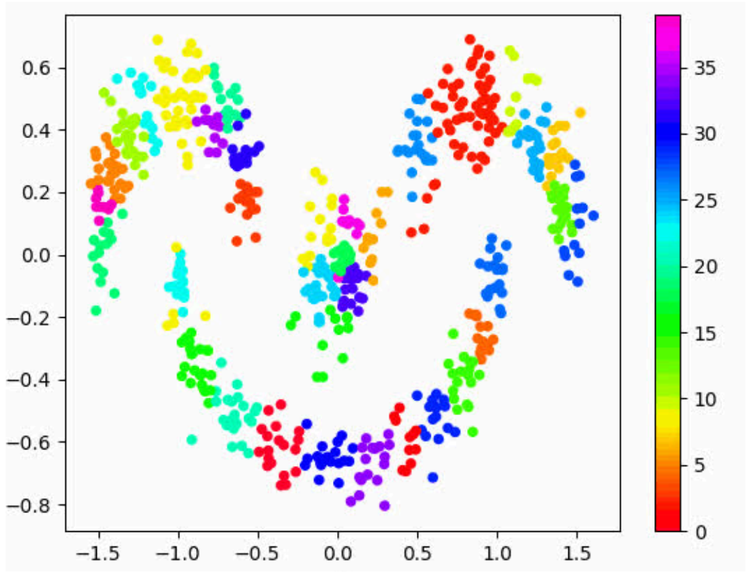


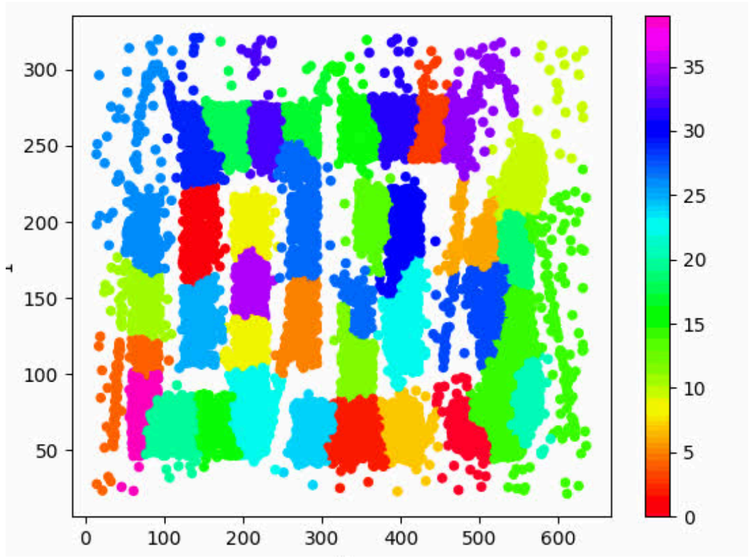




Experiments







Chameleon Clustering

Advantages:

- Don’t need to determine the number of clusters.
- Can takle with complex structural data.

Disadvantages:

- High Complexity

THANKS FOR WATCHING!

[Link Github]