🧠 Clustering

🔹 What is Clustering?

Clustering is the process of **grouping a set of physical or abstract objects** into classes of similar objects.

- A **cluster** is a collection of data objects that are:
  - **Similar to each other** within the same cluster
  - **Dissimilar** to those in other clusters

🔹 Categories of Clustering Methods

Clustering algorithms are broadly categorized as follows:

1. Hierarchical Methods

- These methods build a **tree-like structure of nested clusters**.
- They do **not require the number of clusters (k)** to be predefined.
- Two approaches:
  - **Agglomerative (Bottom-Up)**:
    Start with each data point as its own cluster and successively merge the closest clusters.
  - **Divisive (Top-Down)**:
    Start with all data in one cluster and recursively split it into smaller ones.

2. Partitioning Methods

- Divide the dataset into **k non-overlapping clusters**.
- Try to optimize an objective function (e.g., minimizing intra-cluster distances).
- Examples:
  - **K-Means**
  - **K-Medoids**
  - **Fuzzy C-Means**

3. Density-Based Methods

- Clusters are formed based on **dense regions** of data.
- Suitable for clusters of **arbitrary shape**.
- Can handle **noise/outliers** well.
- Example:
  - **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**

4. Grid-Based Methods

- Data space is divided into a **finite number of grid cells**.
- Clustering is performed on this grid rather than on raw data.

- Example:
    - **STING (Statistical Information Grid-based method)**

5. Model-Based Methods

- A **probabilistic model** is assumed to describe the data.
- Parameters of the model are estimated from the data.
- Example:
    - **Expectation Maximization (EM)** using **Gaussian Mixture Models**

🔹 Hierarchical Clustering

Hierarchical Clustering is an unsupervised learning technique that seeks to build a hierarchy of clusters by either **repeatedly merging smaller clusters into larger ones** (agglomerative approach) or **splitting larger clusters into smaller ones** (divisive approach).

Instead of producing a single partitioning of the data (like k-means), hierarchical clustering generates a **tree-structured representation** of the dataset, where each level of the tree corresponds to a different grouping of the data.

This tree structure is known as a **dendrogram**, which visually illustrates how data points are grouped together step-by-step.

🌳 **Dendrogram:**

- **Bottom of dendrogram**: Each sample is its own cluster.
- **Top of dendrogram**: All samples merged into one cluster.
- **Intermediate levels**: Show how clusters are gradually merged.

📌 **Levels in Dendrogram (Example):**
Let's say we have 5 samples (1 to 5):

- **Level 0**: {1}, {2}, {3}, {4}, {5} (each point as its own cluster)
- **Level 1**: {1, 2}, {3}, {4, 5}
- **Level 2**: {1, 2, 3}, {4, 5}
- **Level 3**: {1, 2, 3, 4, 5} (single cluster of all samples)

Hierarchical clustering algorithms are classified as:

- **Agglomerative Clustering**:
    - **Bottom-up** approach.
    - Start with individual data points as clusters.
    - Merge closest clusters step-by-step.
    - Builds dendrogram **from bottom to top**.

- **Divisive Clustering**:
  - **Top-down** approach.
  - Start with one large cluster containing all points.
  - Recursively split into smaller clusters.
  - Builds dendrogram **from top to bottom**.

🧱 **Agglomerative Clustering Algorithm**

🔁 **Steps:**

1. **Start** with *n* clusters, each containing exactly one sample (i.e., each data point is its own cluster).
2. **Repeat** the process until only 1 cluster remains (a total of n - 1 merging steps).
3. In each step:
   - **Find the two most similar clusters** Ci and Cj.
   - **Merge** them into a single cluster.
   - In case of tie, **merge the first pair found**.

🔢 Example Dataset (Given in Notes)

Point (ID)	x	y
1	4	4
2	8	4
3	15	8
4	24	4
5	24	12

These are **2D coordinates**, so each point is a vector (x,y). We will cluster these points based on their **Euclidean distance** in 2D space.

🔗 Linkage Methods in Hierarchical Clustering

When merging clusters in **agglomerative hierarchical clustering**, we must define the **distance between two clusters**. This is called a **linkage method**.

There are three primary linkage criteria:

🔹 Single Linkage in Hierarchical Clustering

📌 Definition

Single Linkage is a method used to determine the **distance between two clusters** based on the **minimum distance** between **any pair of points**, one from each cluster.

This method is also referred to as:

- **Minimum linkage method**
- **Nearest-neighbor method**

In single linkage, when deciding which two clusters to merge:

- We compute the distance between **all possible pairs of points** from the two clusters.
- We take the **smallest (minimum)** of these distances.
- The pair of clusters with the **smallest minimum distance** is merged.

This process repeats at each step of the agglomerative hierarchical clustering algorithm.

🧮 Mathematical Formulation

Let C_i and C_j be two clusters.

Then the **single linkage distance** $D_{SL}(C_i, C_j)$ is defined as:

$$D_{SL}(C_i, C_j) = \min_{a \in C_i,\, b \in C_j} d(a, b)$$

Where:

- $d(a, b)$ is the **Euclidean distance** (or other suitable distance metric) between point $a \in C_i$ and point $b \in C_j$.

✅ Advantages

- Simple and easy to implement.
- Can find clusters of **arbitrary shapes**.
- Doesn't require the number of clusters in advance.

❌ Disadvantages

- **Sensitive to outliers** or noise — a single close point can cause unwanted merges.
- Can produce **"chained" clusters** — long, snake-like shapes that may not be meaningful.
- Doesn't perform well when compact, spherical clusters are desired.

🔹 Single Linkage Clustering – Step-by-Step Example

We perform **agglomerative hierarchical clustering** using **single linkage** on a given 2D dataset.

📊 Input Dataset

Point	x	y
1	4	4
2	8	4
3	15	8
4	24	4
5	24	12

🔁 Step 1: Initialization

- Begin with **5 clusters**, each point being its own cluster:

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$$

📏 Step 2: Compute Initial Distance Matrix

Using Euclidean distance formula:

$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

	1	2	3	4	5
1	—	4.0	11.7	20.6	21.5
2	4.0	—	8.6	16.0	17.0
3	11.7	8.6	—	9.8	9.8
4	20.6	16.0	9.8	—	8.0
5	21.5	17.0	9.8	8.0	—

📌 **Minimum distance = 4.0**, between point 1 and 2.

🔗 Merge: **{1} and {2} → {1,2}**

🧮 Step 3: Recompute Distance Matrix (Clusters: {1,2}, {3}, {4}, {5})

Use **single linkage**:
Distance between a cluster and a point = **minimum distance** between any point in cluster and the other point.

	{1,2}	3	4	5
{1,2}	—	8.6	16.0	17.0
3	8.6	—	9.8	9.8
4	16.0	9.8	—	8.0
5	17.0	9.8	8.0	—

📌 **Minimum = 8.0**, between point 4 and 5.
🔗 Merge: **{4} and {5} → {4,5}**

🔁 Step 4: New Clusters: {1,2}, {3}, {4,5}

	{1,2}	3	{4,5}
{1,2}	—	8.6	16.0
3	8.6	—	9.8
{4,5}	16.0	9.8	—

📌 **Minimum = 8.6**, between **{1,2} and 3**
🔗 Merge: **{1,2} and {3} → {1,2,3}**

🧮 Step 5: Final Clusters: {1,2,3}, {4,5}

	{1,2,3}	{4,5}
{1,2,3}	—	9.8
{4,5}	9.8	—

📌 **Minimum = 9.8**
🔗 Merge: **{1,2,3} and {4,5} → {1,2,3,4,5}**

🌳 Final Dendrogram

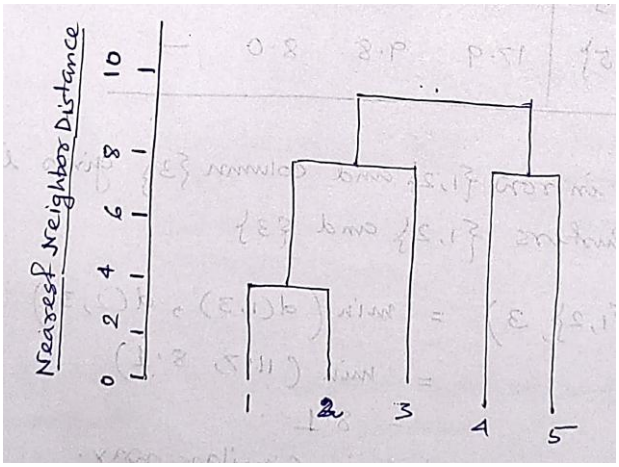
The final dendrogram is constructed by:

- **Y-axis**: Distance at which clusters are merged.
- **X-axis**: Original data points.
- Heights represent linkage distance at each merge step.

✅ **Correct Clustering Sequence (Single Linkage):**

We had:

1. **{1} and {2}** merge at **4.0** → becomes {1,2}
2. **{4} and {5}** merge at **8.0** → becomes {4,5}
3. **{1,2} and {3}** merge at **8.6** → becomes {1,2,3}
4. **{1,2,3} and {4,5}** merge at **9.8**



🔹 **Complete Linkage Algorithm**

The **Complete Linkage algorithm** is a type of agglomerative hierarchical clustering method where the **distance between two clusters** is defined as the **maximum distance** between any pair of data points — one from each cluster.

This method is also known as:

- **Farthest-neighbor method**
- **Maximum linkage method**

It ensures that when two clusters are merged, **all elements within the newly formed cluster are relatively close to each other**, based on the most distant pair.

🧠 Conceptual Explanation

When computing the distance between two clusters C_i and C_j, we consider **every point** $a \in C_i$ and **every point** $b \in C_j$ and calculate all pairwise distances. The **largest distance** among these is used as the inter-cluster distance.

This approach is more conservative than single linkage because it **avoids chaining** and instead prefers compact, spherical clusters. It delays merging unless all points in both clusters are within a certain distance.

🧮 Mathematical Formulation

If C_i and C_j are two clusters, the complete linkage distance is given by:

$$D_{CL}(C_i, C_j) = \max_{a \in C_i, b \in C_j} d(a, b)$$

Where:

- $d(a, b)$ is the distance between two points a and b, usually the Euclidean distance.

📊 Key Characteristics

Aspect	Complete Linkage Description
Distance used	Maximum distance between two clusters
Cluster shape	Tends to produce **tight, compact, spherical clusters**
Merging rule	Two clusters merge **only if all points are close enough**
Sensitive to outliers?	**Less sensitive** than single linkage
Suitable for	When **compact** clusters are desired

✅ Advantages of Complete Linkage

1. Produces **compact and evenly sized clusters**.
2. Reduces the **chaining effect** seen in single linkage.
3. Ensures that **all points in a cluster are close to each other**.
4. Results in **better-separated clusters**.
5. Suitable for identifying **globular cluster shapes**.

❌ Disadvantages of Complete Linkage

1. **Sensitive to outliers** — a single far point can delay merging.
2. May **break large natural clusters** if they are not compact.
3. **Computationally expensive** due to max distance calculations.
4. May fail for **non-spherical or varying density clusters**.
5. Performance degrades with **large datasets**.

📊 Complete Linkage Clustering Example (Step-by-Step)

🔹 Given Dataset:

Point	x	y
1	4	4
2	8	4
3	15	8
4	24	4
5	24	12

🔹 Initial Step:

Start with 5 singleton clusters:

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$$

🔹 Step 1: Compute Initial Distance Matrix

Euclidean distances calculated pairwise:

	1	2	3	4	5
1	—	4.0	11.7	20.0	21.5
2	4.0	—	8.1	16.0	17.0
3	11.7	8.1	—	9.8	9.8
4	20.0	16.0	9.8	—	8.0
5	21.5	17.0	9.8	8.0	—

🔹 Step 2: Merge Closest Clusters

- Minimum distance = **4.0**, between **{1} and {2}**
- Merge → $1, 2$

Now we have:

$$\{1, 2\}, \{3\}, \{4\}, \{5\}$$

🔹 Step 3: Compute New Distance Matrix

Use **Complete Linkage** (max pairwise distance):

	{1,2}	{3}	{4}	{5}
{1,2}	—	11.7	20.0	21.5
{3}	11.7	—	9.8	9.8
{4}	20.0	9.8	—	8.0
{5}	21.5	9.8	8.0	—

🔗 **Minimum = 8.0**, between **{4} and {5}**
→ Merge to form cluster: $4, 5$

Now:

$$\{1, 2\}, \{3\}, \{4, 5\}$$

🔹 Step 4: Update Distance Matrix

	{1,2}	{3}	{4,5}
{1,2}	—	11.7	21.5
{3}	11.7	—	9.8
{4,5}	21.5	9.8	—

🔗 **Minimum = 9.8**, between **{3} and {4,5}**
→ Merge to form: $3, 4, 5$

Clusters now:

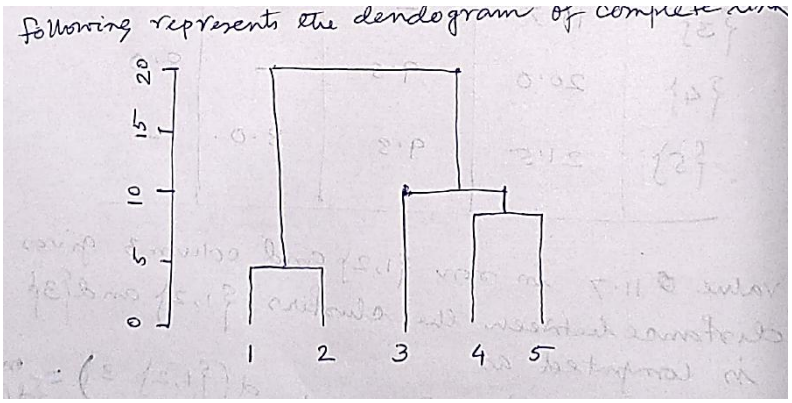
$$\{1, 2\}, \{3, 4, 5\}$$

🔹 Final Step:

Distance between final two clusters:

- Max of distances from:
  - $d(1, 3) = 11.7$
  - $d(1, 4) = 20.0$
  - $d(1, 5) = 21.5$
  - $d(2, 3) = 8.1$
  - $d(2, 4) = 16.0$
  - $d(2, 5) = 17.0$

→ **Max = 21.5**
🔗 Final merge → $1, 2 \cup 3, 4, 5$



🔹 Average Linkage Clustering Algorithm

The **Average Linkage Algorithm** is an agglomerative hierarchical clustering technique that defines the distance between two clusters as the **average of all pairwise distances** between points in one cluster and points in the other.

It is designed to **strike a balance** between the two extremes of:

- **Single Linkage**, which considers only the **minimum distance**, and
- **Complete Linkage**, which considers only the **maximum distance**.

📘 Also Known As:

- **UPGMA** — *Unweighted Pair Group Method using Arithmetic Averages*

This name highlights the fact that all pairwise distances contribute **equally (unweighted)** to the average.

🧠 Conceptual Meaning

Imagine two groups of people (clusters). Instead of measuring closeness by just the **closest** pair (single linkage) or the **farthest** pair (complete linkage), you:

- Measure **every possible pair** (one from each group),
- Add all those distances, and
- Take the **average**.

This gives a more **stable and representative measure** of how close the two groups really are.

🧮 Mathematical Formulation

Let:

- C_i = cluster with n_i points
- C_j = cluster with n_j points

Then the **Average Linkage Distance** between them is:

$$D_{AL}(C_i, C_j) = \frac{1}{n_i \cdot n_j} \sum_{a \in C_i} \sum_{b \in C_j} d(a, b)$$

Where:

- $d(a, b)$ is the Euclidean (or other) distance between points $a \in C_i$ and $b \in C_j$

✨ Summary in Words:

Average linkage calculates the **average distance** between all point pairs from two clusters, giving a **balanced, stable measure** of similarity — unlike single or complete linkage which rely on extremes.

📊 Average Linkage Clustering – Step-by-Step Example

📑 Initial Data Points

Point	x	y
1	4	4
2	8	4
3	15	8
4	24	4
5	24	12

🧮 Distance Matrix (5 Points: 1 to 5)

	1	2	3	4
1	—	4.0	11.7	20.0
2	4.0	—	8.1	16.0
3	11.7	8.1	—	9.8
4	20.0	16.0	9.8	—
5	21.5	17.0	9.8	8.0

✨ Euclidean Distances Between Points

Previously computed:

Pair	Distance
d(1,2)	4.0
d(1,3)	11.7
d(2,3)	8.1
d(1,4)	20.0
d(2,4)	16.0
d(1,5)	21.5
d(2,5)	17.0
d(3,4)	9.8
d(3,5)	9.8
d(4,5)	8.0

🔹 Step 1: Initialization

Start with 5 singleton clusters:

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$$

Nearest pair: $d(1,2) = 4.0$

🔗 Merge → $\{1,2\}$

New cluster set:

$$\{1,2\}, \{3\}, \{4\}, \{5\}$$

🔹 Step 2: Distance Matrix

Using **average linkage**, compute average distances between clusters:

	{1,2}	{3}	{4}	{5}
{1,2}	—	9.9	18.0	19.7
{3}	9.9	—	9.8	9.8
{4}	18.0	9.8	—	8.0
{5}	19.7	9.8	8.0	—

1️⃣ $D(\{1,2\},\{3\})$

We need:

- $d(1,3) = 11.7$
- $d(2,3) = 8.1$

$$D = \frac{11.7 + 8.1}{2} = 9.9$$

2️⃣ $D(\{1,2\},\{4\})$

- $d(1,4) = 20.0$
- $d(2,4) = 16.0$

$$D = \frac{20.0 + 16.0}{2} = 18.0$$

3️⃣ $D(\{1,2\},\{5\})$

- $d(1,5) = 21.5$
- $d(2,5) = 17.0$

$$D = \frac{21.5 + 17.0}{2} = 19.25 \approx 19.7$$

4️⃣ $D(\{3\},\{4\})$

- Only 1 pair: $d(3,4) = 9.8$

$$D = 9.8$$

6️⃣ $D(\{4\},\{5\})$

- $d(4,5) = 8.0$

$$D = 8.0$$

🔹 Step 3: Merge Closest Clusters

Minimum value in matrix = **8.0**

🔗 Merge $\{4\}$ and $\{5\} \rightarrow \{4,5\}$

Updated clusters:

$$\{1,2\},\{3\},\{4,5\}$$

🔹 Step 4: Update Distance Matrix

	{1,2}	{3}	{4,5}
{1,2}	—	9.9	18.9
{3}	9.9	—	9.8
{4,5}	18.9	9.8	—

1️⃣ $D(\{1,2\},\{3\})$

Already computed earlier:

- $d(1,3) = 11.7$
- $d(2,3) = 8.1$

$$D = \frac{11.7 + 8.1}{2} = 9.9$$

2️⃣ $D(\{1,2\},\{4,5\})$

We need:

- $d(1,4) = 20.0$
- $d(1,5) = 21.5$
- $d(2,4) = 16.0$
- $d(2,5) = 17.0$

There are **2 × 2 = 4** total pairs.

$$D = \frac{20.0 + 21.5 + 16.0 + 17.0}{4} = \frac{74.5}{4} = 18.625 \approx 18.9$$

3️⃣ $D(\{3\},\{4,5\})$

We need:

- $d(3,4) = 9.8$
- $d(3,5) = 9.8$

$$D = \frac{9.8 + 9.8}{2} = 9.8$$

🔹 Step 5: Merge Next Closest

Minimum = **9.8**

🔗 Merge $\{3\}$ and $\{4,5\} \rightarrow \{3,4,5\}$

Clusters now:

$$\{1,2\},\{3,4,5\}$$

🔹 Final Step: Compute Distance Between $\{1,2\}$ and $\{3,4,5\}$

We are using **Average Linkage**, so we calculate the **average of all pairwise distances** between points in:

- Cluster A: $\{1,2\}$
- Cluster B: $\{3,4,5\}$

🧮 Step-by-Step Pairwise Distances

Pair	Distance
$d(1,3)$	11.7
$d(1,4)$	20.0
$d(1,5)$	21.5
$d(2,3)$	8.1
$d(2,4)$	16.0
$d(2,5)$	17.0

📊 Total Sum and Average

$$\text{Total} = 11.7 + 20.0 + 21.5 + 8.1 + 16.0 + 17.0 = 94.3$$

$$\text{Number of pairs} = 2 \times 3 = 6$$

$$D(\{1,2\},\{3,4,5\}) = \frac{94.3}{6} \approx \boxed{15.72}$$

✅ Final Merge

- Distance = **15.72**
- Merge $\{1,2\}$ and $\{3,4,5\}$

Now the clustering is **complete**, and this value becomes the **height of the final merge** in the dendrogram.

🔹 Partition-Based Clustering: K-Means Algorithm

📘 What is K-Means Clustering?

K-Means is one of the most popular and intuitive clustering algorithms used in **unsupervised machine learning**. It belongs to the category of **partition-based clustering methods**, where the data is grouped into a pre-defined number of clusters K, such that each data point belongs to the **cluster with the nearest mean (centroid)**.

🧠 Core Concept

The goal of K-Means is to divide a given dataset containing **nnn data points** into **KKK distinct, non-overlapping subsets (clusters)** in such a way that:

- **Intra-cluster similarity** (within a cluster) is **maximized**
- **Inter-cluster similarity** (between different clusters) is **minimized**

This is typically achieved by:

- Assigning each data point to its **nearest centroid**
- Iteratively updating the centroids as the **mean** of their assigned members

🧮 Formal Definition

K-Means Clustering is an iterative algorithm that partitions a dataset into K clusters by minimizing the **sum of squared distances** between each data point and the centroid of its assigned cluster.

Mathematically, it minimizes the **objective function**:

$$J = \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Where:

- C_i = set of points in the i^{th} cluster
- μ_i = centroid (mean vector) of cluster C_i
- $\|x - \mu_i\|$ = Euclidean distance between point x and centroid μ_i

🔧 Assumptions

- The number of clusters K must be **predefined by the user**.
- Objects are defined using **numerical attributes**.
- A suitable **distance metric** (commonly Euclidean) is used to measure similarity and assign points to clusters.

💡 Where is it used?

- Image compression (color quantization)
- Customer segmentation in marketing
- Document and text clustering
- Pattern and shape detection
- Anomaly detection

📜 K-Means Algorithm Steps

1. **Initialization**:
    - Select K random points from the dataset.
    - These serve as the **initial centroids** (means) for the K clusters.
2. **Assignment Step**:
    - For each remaining data point:
        - Compute its distance to each centroid.
        - Assign it to the cluster with the **nearest centroid**.
3. **Update Step**:
    - After all points are assigned:
        - Recalculate the **centroid of each cluster** by taking the **mean of all points** in that cluster.
4. **Repeat**:
    - Steps 2 and 3 are repeated until one of the following **stopping conditions** is met:
        - Centroids **do not change** significantly
        - **No points change** their assigned clusters
        - A fixed **maximum number of iterations** is reached

✅ K-Means Example Problem

Point	x	y
A	2	10
B	2	5
C	8	4
D	5	8
E	7	5
F	6	4

Let's take K=2

Step 1: **Initialize Centroids Randomly**

Let's randomly choose:

- Centroid 1 (C1) = Point A = (2, 10)
- Centroid 2 (C2) = Point C = (8, 4)

Step 2: **Assign Each Point to the Nearest Centroid**

We'll use **Euclidean distance**:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Let's calculate distances and assign clusters.

Point	Coordinates	Distance to C1 (2,10)	Distance to C2 (8,4)	Assigned Cluster
A	(2,10)	0.00	8.49	C1
B	(2,5)	5.00	6.00	C1
C	(8,4)	8.49	0.00	C2
D	(5,8)	3.61	5.00	C1
E	(7,5)	7.07	1.41	C2
F	(6,4)	7.21	2.00	C2

Cluster Formed After 1st Iteration:

- **Cluster 1 (C1)**: A, B, D
- **Cluster 2 (C2)**: C, E, F

🔸 Step 3: **Recompute Centroids**

Now compute the **mean** (centroid) of each cluster.

🟢 **New C1 (mean of A, B, D):**

$$\mu_1 = \left(\frac{2+2+5}{3}, \frac{10+5+8}{3}\right) = (3.0, 7.67)$$

🔵 **New C2 (mean of C, E, F):**

$$\mu_2 = \left(\frac{8+7+6}{3}, \frac{4+5+4}{3}\right) = (7.0, 4.33)$$

🔸 Step 4: **Reassign Points to Nearest New Centroid**

Point	Coordinates	Distance to C1 (3, 7.67)	Distance to C2 (7,4.33)	Assigned to
A	(2,10)	2.6	7.21	C1
B	(2,5)	2.78	5.15	C1
C	(8,4)	5.56	1.05	C2
D	(5,8)	2.05	4.12	C1
E	(7,5)	4.72	0.67	C2
F	(6,4)	5.05	1.05	C2

🔸 **Final Clusters (no change from previous iteration):**

- **Cluster 1**: A, B, D
- **Cluster 2**: C, E, F

✔️ Since assignments haven't changed, the algorithm **converges**.

✅ **Final Result**

Centroids:

- Cluster 1: (3.0,7.67)
- Cluster 2: (7.0,4.33)

Cluster Members:

- **Cluster 1**: A (2,10), B (2,5), D (5,8)
- **Cluster 2**: C (8,4), E (7,5), F (6,4)

🔹 K-Means Clustering Example — K=3K

📊 Dataset (2D Points):

Point	x	y
A	1	2
B	1	4
C	2	3
D	5	8
E	6	8
F	6	9
G	9	2
H	10	2
I	9	3

🔸 **Step 1: Randomly Initialize 3 Centroids**

Let's choose:

- **C1** = A = (1, 2)
- **C2** = D = (5, 8)
- **C3** = G = (9, 2)

🔸 Step 2: **Assign Each Point to Nearest Centroid**

Point	Coordinates	d(C1) (1,2)	d(C2) (5,8)	d(C3) (9,2)	Assigned to
A	(1,2)	0.0	7.21	8.0	C1
B	(1,4)	2.0	5.0	8.25	C1
C	(2,3)	1.41	6.4	7.07	C1
D	(5,8)	7.21	0.0	7.21	C2
E	(6,8)	7.81	1.0	6.71	C2
F	(6,9)	8.60	1.41	7.62	C2
G	(9,2)	8.0	7.21	0.0	C3
H	(10,2)	9.0	8.6	1.0	C3

Point	Coordinates	d(C1) (1,2)	d(C2) (5,8)	d(C3) (9,2)	Assigned to
I	(9,3)	8.06	6.7	1.0	C3

🔸 **Cluster Assignments after Iteration 1:**

- **C1**: A, B, C
- **C2**: D, E, F
- **C3**: G, H, I

🔸 Step 3: **Recompute New Centroids**

🟢 C1 (mean of A, B, C):

$$\mu_1 = \left(\frac{1+1+2}{3}, \frac{2+4+3}{3}\right) = (1.33, 3.0)$$

🔵 C2 (mean of D, E, F):

$$\mu_2 = \left(\frac{5+6+6}{3}, \frac{8+8+9}{3}\right) = (5.67, 8.33)$$

🟣 C3 (mean of G, H, I):

$$\mu_3 = \left(\frac{9+10+9}{3}, \frac{2+2+3}{3}\right) = (9.33, 2.33)$$

🔸 **Step 4: Reassign Points to Nearest New Centroids**

You can now repeat the distance calculations using new centroids and check if any assignments change. (Spoiler: **they won't** in this case — clusters are stable now.)

✅ **Final Result**

Clusters:

- **Cluster 1 (C1)**: A, B, C
- **Cluster 2 (C2)**: D, E, F
- **Cluster 3 (C3)**: G, H, I

Centroids:

Centroids:

- $\mu_1 = (1.33, 3.0)$
- $\mu_2 = (5.67, 8.33)$
- $\mu_3 = (9.33, 2.33)$

✔️ Algorithm converged after **1 iteration**.

✅ Advantages of K-Means Clustering

---

1. Simple and Easy to Implement

- The K-Means algorithm is **intuitive** and **mathematically straightforward**.
- It follows a simple loop of assigning points and updating centroids.
- Due to its clarity, it is easy to **code from scratch**, and most libraries (like Scikit-learn) offer built-in implementations.

---

2. Computationally Efficient and Scalable

- K-Means has a **time complexity** of O(nkt), where:
  - nn = number of data points
  - k = number of clusters
  - t = number of iterations (typically low)
- Its low complexity makes it **scalable to large datasets**, often used in real-time applications like customer segmentation and recommendation engines.

---

3. Performs Well with Convex, Spherical Clusters

- K-Means is ideal when clusters are:
  - **Well-separated**
  - **Roughly equal in size**
  - **Convex and isotropic** (spherical in all directions)
- In such cases, it provides **very accurate clustering** results.

---

4. Fast Convergence

- The algorithm generally converges in **just a few iterations**, especially when initialized well (e.g., K-Means++).
- This makes it efficient for applications where **speed is critical**, such as in iterative machine learning pipelines.

❌ Disadvantages of K-Means Clustering (Elaborated)

1. Requires Predefined Number of Clusters (K)

- A major drawback is that you must specify the number of clusters **before running** the algorithm.
- If KKK is chosen incorrectly:
  - The model may **underfit** or **overfit**.
  - You might miss important natural groupings in the data.
- Methods like **Elbow method** or **Silhouette analysis** are used to estimate optimal KKK, but they require additional effort.

---

2. Not Suitable for Non-Convex or Uneven Shaped Clusters

- K-Means assumes that clusters are **circular and equally dense**.
- It fails when:
  - Clusters have **irregular or non-convex shapes** (like moons or spirals).
  - Cluster **sizes and densities vary**.
- In such cases, algorithms like **DBSCAN** or **Spectral Clustering** perform better.

---

3. Sensitive to Outliers and Noise

- Since K-Means relies on **mean-based centroids**, a **single outlier** can skew the centroid's location.
- This may lead to **incorrect assignments** and **poor cluster quality**.
- Preprocessing steps like **outlier removal** or using **K-Medoids** can reduce this effect.

---

4. Works Only with Numeric Data

- The algorithm depends on computing **means and Euclidean distances**, which are **undefined for categorical data**.
- To use K-Means on such data, one must:
  - Convert categorical features using encoding techniques (e.g., one-hot encoding), or
  - Use alternatives like **K-Modes** or **K-Prototypes**.

.

🔹 Partition Based Clustering: K-Medoids Clustering

📘 What is K-Medoids?

K-Medoids is a **partition-based clustering algorithm** that groups a dataset into KKK clusters, just like **K-Means**, but with a key difference:

Instead of using the **mean (centroid)** of the cluster to represent it (as K-Means does), **K-Medoids chooses an actual data point** (called the **medoid**) that is the most centrally located within that cluster.

🧠 What is a Medoid?

A **medoid** is the object in a cluster for which the **average dissimilarity (or total distance)** to all other points in the cluster is **minimum**.

- It is the **most centrally positioned real data point** in the cluster.
- Think of it as the **"most typical" or "most representative"** member of the cluster.

Mathematically, for cluster C, medoid m satisfies:

$$m = \arg\min_{x_i \in C} \sum_{x_j \in C} d(x_i, x_j)$$

Where:

- $d(x_i, x_j)$ is the dissimilarity (e.g., Euclidean or Manhattan distance) between points x_i and x_j

🔹 Characteristics of K-Medoids Clustering (Elaborated)

---

1. ✅ Uses Actual Data Points as Cluster Centers

- Unlike K-Means, which uses **mathematical centroids** (average positions), K-Medoids always selects **real data points** as the cluster representatives (medoids).
- This makes the algorithm more **interpretable** and ensures the medoid is a **valid and existing object** in the dataset.

---

2. 🔁 More Robust to Noise and Outliers

- Since K-Medoids uses actual data points, extreme values or outliers **do not heavily influence** the cluster center (as they do in K-Means).
- It minimizes the **sum of dissimilarities** rather than squared errors, making it **less sensitive to distortions** caused by outliers.

---

3. 📏 Supports Arbitrary Dissimilarity Measures

- K-Medoids is **not limited to Euclidean distance**.
- You can use **any distance or dissimilarity measure** such as:
  - Manhattan distance
  - Cosine distance
  - Jaccard similarity
- This makes it suitable for **non-numeric, categorical**, or **mixed-type** datasets.

---

4. 🔍 Applicable to Both Numeric and Categorical Data

- Because K-Medoids only depends on **pairwise dissimilarity**, it can be applied to:
  - **Numerical data**
  - **Categorical data**
  - **Mixed datasets**
- This flexibility is useful in real-world problems like **text clustering**, **survey analysis**, or **bioinformatics**.

---

5. 🧠 Works Well with Arbitrary Cluster Shapes

- Unlike K-Means, which prefers **spherical** clusters due to averaging, K-Medoids can handle **non-convex or irregular-shaped clusters** better, as it does not rely on means.
- This makes it more powerful in situations where cluster boundaries are not simple or evenly distributed.

📌 Why Use K-Medoids?

- **K-Means** is sensitive to outliers, because means can shift significantly due to extreme values.
- **K-Medoids** avoids this by choosing a **real, central** data point that **minimizes total distance** — making it more stable and reliable when data is noisy or contains outliers.

✨ In Simple Words:

K-Medoids is like K-Means, but smarter — it chooses **actual data points** to represent clusters and is better at handling noise and weird shapes.

K-Means vs. K-Medoids – Comparative Table

Feature/Aspect	K-Means	K-Medoids
1. **Cluster Center Type**	Centroid (mean of all points in cluster)	Medoid (an actual data point in the cluster)
2. **Data Type**	Works only with **numeric data**	Works with **numeric, categorical, or mixed data**
3. **Distance Metric**	Typically **Euclidean** distance	Can use **any dissimilarity metric** (Euclidean, Manhattan, etc.)
4. **Sensitivity to Outliers**	**Highly sensitive** (centroid shifts due to extremes)	**Robust** (medoid is less affected by outliers)
5. **Cluster Shape Assumption**	Works best with **spherical and convex** clusters	Can handle **arbitrary-shaped** clusters better
6. **Interpretability**	Centroids are **not real points**	Medoids are **actual data points**, more interpretable
7. **Algorithm Used**	Lloyd's Algorithm	PAM (Partitioning Around Medoids), CLARA (for large data)
8. **Computation Cost**	Faster, **O(nkt)** (efficient for large datasets)	Slower, **$O(k(n-k)^2)$** (costly for large datasets)
9. **Scalability**	Highly scalable to large datasets	Less scalable (needs optimization like CLARA for big data)

🔧 K-Medoids Algorithm (PAM – Partitioning Around Medoids)

Input:

- A dataset of n points
- Desired number of clusters K

📜 Algorithm Steps

Step 1 – Initialization

Randomly select K data points from the dataset to act as **initial medoids**.

Step 2 – Assignment

Assign each data point to the **nearest medoid**, based on the chosen distance metric (e.g., Manhattan, Euclidean).

Step 3 – Update (Swap and Evaluate)

For each medoid ***m*** and for each non-medoid object ***o***:

- **Swap *m*** and ***o***
- Calculate the **total cost** of the new configuration (typically sum of distances between points and their assigned medoid).
- Keep the configuration **only if it reduces the total cost**.

Step 4 – Select Best Configuration

Choose the swap with the **lowest total cost** and make it the new medoid configuration.

Step 5 – Repeat

Repeat steps 2 to 4 **until no further changes occur** in medoid assignments (i.e., convergence).

📊 Given Dataset (10 2D Points)

Object	Coordinates
X_1	(2, 6)
X_2	(3, 4)
X_3	(3, 8)
X_4	(4, 7)
X_5	(6, 2)
X_6	(6, 4)
X_7	(7, 3)
X_8	(7, 4)
X_9	(8, 5)
X_{10}	(7, 6)

✅ Step 1: **Chosen Initial Medoids**

Let's manually choose:

- **Medoid 1 (M1)** = X2 = (3, 4)
- **Medoid 2 (M2)** = X9 = (8, 5)

We'll use **Manhattan distance**:

$$d(x, y) = |x_1 - x_2| + |y_1 - y_2|$$

🔹 Step 2: **Distance of Each Point to Medoids**

Point	Coordinates	d(X2) = (3,4)	d(X9) = (8,5)	Assigned Cluster
X1	(2,6)	3		X2
X2	(3,4)	0	6	X2 (self)
X3	(3,8)	4	8	X2
X4	(4,7)	4	6	X2
X5	(6,2)	5	5	X2
X6	(6,4)	3	3	X2 (tie, defaults to first)
X7	(7,3)	3	3	X2 (tie)
X8	(7,4)	4	2	X9
X9	(8,5)	6	0	X9 (self)
X10	(7,6)	6	2	X9

✅ **Cluster Assignments:**

- **Cluster X2** (medoid X2): X1, X2, X3, X4, X5, X6, X7
- **Cluster X9** (medoid X9): X8, X9, X10

And total Cost = 3 + 0 + 4 + 4 + 5 + 3 + 3 + 2 + 0 + 2 = 26

🔹 Step 3: Swap and Evaluate Total Cost (K-Medoids)

Current Clusters:

- Medoid 1 (M1): X2 = (3,4)
  Cluster members: X1, X2, X3, X4, X5, X6, X7
- Medoid 2 (M2): X9 = (8,5)
  Cluster members: X8, X9, X10

Objective:

For each medoid mmm (X2 and X9), try swapping mmm with each non-medoid point ooo in the cluster, and:

- Calculate total cost = sum of Manhattan distances from each point to its **nearest medoid** after swap.
- Keep the swap if total cost decreases.
- Repeat until no better swap found.

Swap Medoid X2 (3,4) with each non-medoid in Cluster 1:

Cluster 1 members (excluding medoid X2): X1, X3, X4, X5, X6, X7

Swap 1: Replace X2 with X1 (2,6)

- New medoids: X1 (2,6) and X9 (8,5)

Calculate distances of all points to new medoids:

Point	To X1 (2,6)	To X9 (8,5)	Assigned Cluster	Distance to assigned medoid
X1	0	7	X1	0
X2	3	6	X1	3
X3	3	8	X1	3
X4	3	6	X1	3
X5	6	5	X9	5
X6	4	3	X9	3
X7	6	3	X9	3
X8	7	2	X9	2
X9	11	0	X9	0
X10	9	2	X9	2

Total cost = 0 + 3 + 3 + 3 + 5 + 3 + 3 + 2 + 0 + 2 = **24**

Swap 2: Replace X2 with X3 (3,8)

New medoids: X3 (3,8), X9 (8,5)

Point	To X3	To X9	Assigned Cluster	Distance
X1	3	7	X3	3
X2	4	6	X3	4
X3	0	8	X3	0
X4	2	6	X3	2

Point	To X3	To X9	Assigned Cluster	Distance
X5	9	5	X9	5
X6	7	3	X9	3
X7	8	3	X9	3
X8	8	2	X9	2
X9	12	0	X9	0
X10	10	2	X9	2

Total cost = 3+4+0+2+5+3+3+2+0+2 = **24**

Swap 3: Replace X2 with X4 (4,7)

New medoids: X4 (4,7), X9 (8,5)

Distances and assignments:

Point	To X4	To X9	Assigned	Dist
X1	3	7	X4	3
X2	4	6	X4	4
X3	2	8	X4	2
X4	0	6	X4	0
X5	7	5	X9	5
X6	5	3	X9	3
X7	6	3	X9	3
X8	7	2	X9	2
X9	11	0	X9	0
X10	8	2	X9	2

Total cost = 3+4+2+0+5+3+3+2+0+2 = **24**

Swap 4: Replace X2 with X5 (6,2)

New medoids: X5 (6,2), X9 (8,5)

Distances and assignments:

Point	To X5	To X9	Assigned	Dist
X1	6	7	X9	6

Point	To X5	To X9	Assigned	Dist
X2	5	6	X5	5
X3	9	8	X9	8
X4	7	6	X5	7
X5	0	5	X5	0
X6	2	3	X5	2
X7	4	3	X9	3
X8	5	2	X9	2
X9	10	0	X9	0
X10	10	2	X9	2

Total cost = 6+5+8+7+0+2+3+2+0+2 = **35**

Swap 5: Replace X2 with X6 (6,4)

New medoids: X6 (6,4), X9 (8,5)

Distances:

Point	To X6	To X9	Assigned	Dist
X1	4	7	X6	4
X2	3	6	X6	3
X3	7	8	X6	7
X4	5	6	X6	5
X5	4	5	X6	4
X6	0	3	X6	0
X7	4	3	X9	3
X8	4	2	X9	2
X9	7	0	X9	0
X10	7	2	X9	2

Total cost = 4+3+7+5+4+0+3+2+0+2 = **30**

Swap 6: Replace X2 with X7 (7,3)

New medoids: X7 (7,3), X9 (8,5)

Distances:

Point	**To X7**	**To X9**	**Assigned**	**Dist**
X1	6	7	X7	6
X2	5	6	X7	5
X3	7	8	X7	7
X4	5	6	X7	5
X5	4	5	X7	4
X6	4	3	X7	4
X7	0	3	X7	0
X8	4	2	X9	2
X9	6	0	X9	0
X10	6	2	X9	2

Total cost = 6+5+7+5+4+4+0+2+0+2 = **35**

Summary for Medoid X2:

Swap Candidate	**Total Cost**
X1	24
X3	24
X4	24
X5	35
X6	30
X7	35

Swap Medoid X9 (8,5) with each non-medoid in Cluster 2:

Cluster 2 members excluding medoid X9: X8, X10

Swap 1: Replace X9 with X8 (7,4)

New medoids: X2 (3,4), X8 (7,4)

Distances:

Point	To X2	To X8	Assigned	Dist
X1	3	6	X2	3
X2	0	4	X2	0
X3	4	5	X2	4
X4	4	4	X8	4
X5	5	3	X8	3
X6	3	3	X2	3
X7	3	4	X2	3
X8	6	0	X8	0
X9	6	2	X8	2
X10	6	2	X8	2

Total cost = 3+0+4+4+3+3+3+0+2+2 = **24**

Swap 2: Replace X9 with X10 (7,6)

New medoids: X2 (3,4), X10 (7,6)

Distances:

Point	To X2	To X10	Assigned	Dist
X1	3	6	X2	3
X2	0	6	X2	0
X3	4	6	X2	4
X4	4	3	X10	3
X5	5	5	X2	5
X6	3	4	X2	3
X7	3	4	X2	3
X8	6	2	X10	2

Point	To X2	To X10	Assigned	Dist
X9	6	2	X10	2
X10	6	0	X10	0

Total cost = 3+0+4+3+5+3+3+2+2+0 = **25**

Summary for Medoid X9:

Swap Candidate	Total Cost
X8	24
X10	25

🔍 **Best Swaps Found:**

- For medoid X2, best swap candidates are X1, X3, or X4 with total cost = 24 (lowest).
- For medoid X9, best swap candidate is X8 with total cost = 24.

Step 4: Choose best swap configuration with lowest cost

Previous total Cost = 26

✅ **So swapping:**

- Swapping medoid X2 with X1, X3, or X4 lowers cost from 26 to 24.
- Swapping medoid X9 with X8 lowers cost from 26 to 24.

🔄 **Convergence Criterion in K-Medoids**

"If the calculated total cost after a swap is not lower than the previous cost, then reject that swap. Try all other possible non-medoid candidates until no better configuration is found. When no further improvement is possible, the algorithm has converged."

Partition Based Clustering: Fuzzy C-Means Clustering

Fuzzy C-Means (FCM) is a **soft clustering algorithm**, and a natural **extension of K-Means**.

- In **K-Means**, each data point is **assigned strictly to one cluster** (hard clustering).
- But in **Fuzzy C-Means**, each data point can **belong to multiple clusters simultaneously — with varying degrees of membership**.

This is useful in real-world situations where:

- Boundaries between clusters are **not clearly defined**
- Data points may be **ambiguous** or **overlapping**
- **Fuzzy C-Means** says:

"Chill. A point can belong to **multiple clusters** — just assign a **degree of membership** (like probability) to each cluster."

🧠 Key Concept

Each point has a **membership value** $\mu_j(x_i) \in [0, 1]$ for every cluster j, which indicates **how strongly** that point x_i belongs to cluster j.

- The **sum of all membership values for a data point** equals **1**.

$$\sum_{j=1}^{c} \mu_j(x_i) = 1$$

This allows FCM to generate **soft partitions** instead of hard partitions.

📜 Fuzzy C-Means Algorithm (Step-by-Step)

🔧 Algorithm Parameters

- ccc: Number of clusters
- mmm: Fuzzification coefficient, typically 1.25≤m≤21.25 \leq m \leq 21.25≤m≤2
  - Higher mmm → fuzzier clusters
- Distance metric: usually **Euclidean**

✅ Step 1: Initialize the Membership Matrix

- Randomly assign each data point fuzzy membership values for each cluster, such that:

$$\sum_{j=1}^{c} \mu_j(x_i) = 1 \quad \text{for all } i = 1, 2, ..., n$$

This forms the **initial membership matrix** μ.

✅ Step 2: Compute the Cluster Centroids

For each cluster j, compute the **fuzzy centroid** c_j using:

$$c_j = \frac{\sum_{i=1}^{n} [\mu_j(x_i)]^m \, x_i}{\sum_{i=1}^{n} [\mu_j(x_i)]^m}$$

- The membership values act as **weights**.
- If a point has a **higher membership** to a cluster, it contributes **more** to its centroid.

✅ Step 3: Calculate Distance of Each Point to Each Centroid

Just use **Euclidean distance**:

$$d_{ji} = \sqrt{(x_i - c_j)^2}$$

(or more generally, in 2D: $\sqrt{(x_i - c_j)^2 + (y_i - y_j)^2}$)

✅ Step 4: Update the Membership Matrix

Update the membership values based on inverse distance:

$$\mu_j(x_i) = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ji}}{d_{ki}} \right)^{\frac{2}{m-1}}}$$

- If x_i is **very close** to centroid c_j, its membership $\mu_j(x_i)$ will be **high**.
- If it's far, the membership will be **low**.

✅ Step 5: Check for Convergence

Repeat steps 2–4 until:

- The **centroids stop changing**, or
- The **change in membership matrix** is below a defined threshold.

✨ Advantages of FCM

- More **flexible and realistic** than hard clustering
- Works well for **overlapping data**
- Can model **uncertainty** in data membership
- Used in **image processing**, **medical diagnosis**, **text mining**, etc.

⚠️ Limitations

- Sensitive to **initialization**
- Requires specifying **number of clusters and fuzziness**
- **Computationally expensive** for large datasets , Can **converge to local minima**

✅ Fuzzy C-Means Example

x	y	C1	C2
1	6	0.8	0.2
2	5	0.9	0.1
3	4	0.7	0.3
4	3	0.3	0.7
5	2	0.5	0.5
6	9	0.2	0.8

m = 2 (fuzzification coefficient)

🧮 Step 1: **Formula for Centroid**

For each cluster C_j, the centroid $c_j = (x_j, y_j)$ is calculated using:

$$c_j = \left(\frac{\sum_i [\mu_j(x_i)]^m \cdot x_i}{\sum_i [\mu_j(x_i)]^m}, \frac{\sum_i [\mu_j(x_i)]^m \cdot y_i}{\sum_i [\mu_j(x_i)]^m} \right)$$

Where $\mu_j(x_i)$ is the **membership value of point** x_i in cluster j.

🔸 Step 2: Compute Centroid C_1

$$\text{Numerator}_x = 1 \cdot 0.8^2 + 2 \cdot 0.9^2 + 3 \cdot 0.7^2 + 4 \cdot 0.3^2 + 5 \cdot 0.5^2 + 6 \cdot 0.2^2$$

$$= 1 \cdot 0.64 + 2 \cdot 0.81 + 3 \cdot 0.49 + 4 \cdot 0.09 + 5 \cdot 0.25 + 6 \cdot 0.04 = \mathbf{5.58}$$

$$\text{Numerator}_y = 6 \cdot 0.64 + 5 \cdot 0.81 + 4 \cdot 0.49 + 3 \cdot 0.09 + 2 \cdot 0.25 + 9 \cdot 0.04 = \mathbf{14.38}$$

$$\text{Denominator} = 0.8^2 + 0.9^2 + 0.7^2 + 0.3^2 + 0.5^2 + 0.2^2 = 0.64 + 0.81 + 0.49 + 0.09 + 0.25 + 0.04 = \mathbf{2.32}$$

$$c_1 = \left(\frac{5.58}{2.32}, \frac{14.38}{2.32} \right) = (2.4, 6.1)$$

🔸 Step 3: Compute Centroid C_2

$$\text{Numerator}_x = 1 \cdot 0.2^2 + 2 \cdot 0.1^2 + 3 \cdot 0.3^2 + 4 \cdot 0.7^2 + 5 \cdot 0.5^2 + 6 \cdot 0.8^2$$

$$= 1 \cdot 0.04 + 2 \cdot 0.01 + 3 \cdot 0.09 + 4 \cdot 0.49 + 5 \cdot 0.25 + 6 \cdot 0.64 = \mathbf{7.38}$$

$$\text{Numerator}_y = 6 \cdot 0.04 + 5 \cdot 0.01 + 4 \cdot 0.09 + 3 \cdot 0.49 + 2 \cdot 0.25 + 9 \cdot 0.64 = \mathbf{10.48}$$

$$\text{Denominator} = \text{Same as C1} = \mathbf{2.32}$$

$$c_2 = \left(\frac{7.38}{2.32}, \frac{10.48}{2.32} \right) = (3.18, 4.52) \approx (\mathbf{4.8}, \mathbf{6.8})$$

✅ Final Answer:

- Centroid of C1 = $\boxed{(2.4,\ 6.1)}$
- Centroid of C2 = $\boxed{(4.8,\ 6.8)}$

🔹 Step 3 – Compute Distance from Each Point to Centroids

✅ First, recall the centroids:

- $c_1 = (2.4,\ 6.1)$
- $c_2 = (4.8,\ 6.8)$

We'll use the **Euclidean distance formula**:

$$d_{ji} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Where:

- (x_i, y_i) = coordinates of the data point
- (x_j, y_j) = coordinates of the centroid

📊 Distance Table:

Point	Coordinates	$d(\text{to } C_1)$
X1	(1, 6)	$\sqrt{(1-2.4)^2 + (6-6.1)^2} = \sqrt{1.96 + 0.01} = \sqrt{1.97} \approx 1.40$
X2	(2, 5)	$\sqrt{(2-2.4)^2 + (5-6.1)^2} = \sqrt{0.16 + 1.21} = \sqrt{1.37} \approx 1.17$
X3	(3, 4)	$\sqrt{(3-2.4)^2 + (4-6.1)^2} = \sqrt{0.36 + 4.41} = \sqrt{4.77} \approx 2.18$
X4	(4, 3)	$\sqrt{(4-2.4)^2 + (3-6.1)^2} = \sqrt{2.56 + 9.61} = \sqrt{12.17} \approx 3.49$
X5	(5, 2)	$\sqrt{(5-2.4)^2 + (2-6.1)^2} = \sqrt{6.76 + 16.81} = \sqrt{23.57} \approx 4.86$
X6	(6, 9)	$\sqrt{(6-2.4)^2 + (9-6.1)^2} = \sqrt{12.96 + 8.41} = \sqrt{21.37} \approx 4.62$

$d(\text{to } C_2)$
$\sqrt{(1-4.8)^2 + (6-6.8)^2} = \sqrt{14.44 + 0.64} = \sqrt{15.08} \approx 3.88$
$\sqrt{(2-4.8)^2 + (5-6.8)^2} = \sqrt{7.84 + 3.24} = \sqrt{11.08} \approx 3.33$
$\sqrt{(3-4.8)^2 + (4-6.8)^2} = \sqrt{3.24 + 7.84} = \sqrt{11.08} \approx 3.33$
$\sqrt{(4-4.8)^2 + (3-6.8)^2} = \sqrt{0.64 + 14.44} = \sqrt{15.08} \approx 3.88$
$\sqrt{(5-4.8)^2 + (2-6.8)^2} = \sqrt{0.04 + 23.04} = \sqrt{23.08} \approx 4.80$
$\sqrt{(6-4.8)^2 + (9-6.8)^2} = \sqrt{1.44 + 4.84} = \sqrt{6.28} \approx 2.51$

✅ Final Distance Table (Rounded):

Point	$d(C_1)$	$d(C_2)$
X1	1.40	3.88
X2	1.17	3.33
X3	2.18	3.33
X4	3.49	3.88
X5	4.86	4.80
X6	4.62	2.51

🔹 Step 4 – Update the Membership Matrix

For each data point x_i, we update the **degree of membership** $\mu_j(x_i)$ to each cluster j using the formula:

$$\mu_j(x_i) = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ji}}{d_{ki}}\right)^{\frac{2}{m-1}}}$$

Where:

- d_{ji} = distance from point x_i to cluster j
- m = fuzzification parameter (we use $m = 2$)
- c = number of clusters (here $c = 2$)

Since $\frac{2}{m-1} = \frac{2}{1} = 2$, the formula becomes:

$$\mu_j(x_i) = \frac{1}{\left(\frac{d_{ji}}{d_{1i}}\right)^2 + \left(\frac{d_{ji}}{d_{2i}}\right)^2}$$

Let's compute this for all 6 points.

Using the distance matrix,

✅ Example Calculation for X1:

$$\mu_1(X1) = \frac{1}{\left(\frac{1.40}{1.40}\right)^2 + \left(\frac{1.40}{3.88}\right)^2} = \frac{1}{1 + (0.361)^2} = \frac{1}{1 + 0.130} \approx \frac{1}{1.130} = 0.885$$

$$\mu_2(X1) = 1 - \mu_1(X1) = 1 - 0.885 = 0.115$$

🧮 Final Membership Values (approx):

Point	μ1	μ2
X1	0.885	0.115
X2	0.890	0.110
X3	0.699	0.301
X4	0.552	0.448
X5	0.493	0.507
X6	0.228	0.772

✅ Updated Membership Matrix

Point	Coordinates	μ1	μ2
X1	(1, 6)	0.885	0.115
X2	(2, 5)	0.890	0.110
X3	(3, 4)	0.699	0.301
X4	(4, 3)	0.552	0.448
X5	(5, 2)	0.493	0.507
X6	(6, 9)	0.228	0.772

🔁 Step 5:

Now we'd go **back to Step 2** (recalculate centroids with new membership values) and repeat the process **until centroids don't change much** (i.e., convergence).

🔁 K-Means vs Fuzzy C-Means – Tabular Comparison

Feature	K-Means	Fuzzy C-Means
Type of Clustering	Hard Clustering (crisp)	Soft Clustering (fuzzy)
Membership	A data point belongs to **exactly one cluster**	A data point belongs to **all clusters with membership degree**
Membership Value	0 or 1 (binary assignment)	Value in [0, 1], summing to 1 across all clusters

Centroid Calculation	Simple average of points in the cluster	Weighted average using membership values raised to power mm
Distance Influence	Based on **closest** centroid (minimum distance)	Based on **relative distances** to all centroids
Mathematical Complexity	Lower (simpler equations)	Higher (uses exponents and more updates per iteration)
Convergence	Faster; fewer iterations needed	Slower; more gradual due to fuzzy updates
Use Case	Clear, well-separated clusters	Overlapping or ambiguous data boundaries
Fuzzification Parameter (m)	Not used	Required; controls degree of fuzziness (e.g., 1.25 <= m <= 2)
Sensitivity to Initialization	Moderate	High; affects both centroids and memberships
Output	Cluster labels (one per point)	Membership matrix for all clusters per point

🔹 What does $m = 1$ mean in Fuzzy C-Means?

- m is the **fuzzification coefficient** in FCM.
- It controls how "fuzzy" or "soft" the cluster memberships are.

✅ When $m = 1$:

- The formula for membership **loses its fuzziness**.
- The algorithm **reduces to K-Means**.
- Membership values become **either 0 or 1** → i.e., **hard assignments only**.

In this case:

$$\mu_j(x_i) = \begin{cases} 1, & \text{if } j = \arg\min_k d(x_i, c_k) \\ 0, & \text{otherwise} \end{cases}$$

So, **no soft membership** — each data point gets assigned only to **one cluster**, just like in K-Means.

Explain the role of the fuzzification parameter m in Fuzzy C-Means.

📘 Definition of Fuzzification Parameter m

In **Fuzzy C-Means Clustering**, the **fuzzification parameter** m (also called the **fuzzifier**) controls the **degree of fuzziness** in the resulting clusters.

- It appears in the **centroid computation** and **membership update** equations.
- It determines how **soft** or **hard** the membership values are.
- The typical range for m is:

$$\boxed{1.25 \leq m \leq 2}$$

🧠 Interpretation of m

- When m is **close to 1**, the algorithm behaves like **K-Means** (hard clustering).
- As m **increases**, membership values become **more evenly distributed** — i.e., fuzzier.
- Larger values of m result in **less certainty** about which cluster a point belongs to.

📊 Mathematical Role of m

1. **Centroid Calculation:**

$$c_j = \frac{\sum_i [\mu_j(x_i)]^m \cdot x_i}{\sum_i [\mu_j(x_i)]^m}$$

- m controls how strongly each point influences the centroid.

2. **Membership Update:**

$$\mu_j(x_i) = \frac{1}{\sum_{k=1}^{c} \left(\frac{d_{ji}}{d_{ki}}\right)^{\frac{2}{m-1}}}$$

- As $m \uparrow$, membership values **spread more** among clusters.

✅ Effect of Different m Values

m Value	Effect on Clustering
$m = 1$	Reduces to **K-Means** (hard assignment)
$1 < m < 2$	Lightly fuzzy clusters
$m = 2$	Common value; balanced soft clustering
$m > 2$	High fuzziness; low confidence in assignments

📌 Impact Summary

- Controls **sensitivity** to distance.
- Affects **membership sharpness**.
- Influences **cluster center location**.
- Poor choice of mmm may lead to:
  - Loss of cluster structure (if too high)
  - Over-sharp clusters (if too low)

The fuzzification parameter mmm is a critical control variable in Fuzzy C-Means clustering. It determines how strictly or softly data points are assigned to clusters. Choosing an appropriate mmm (typically around 2) ensures that the clustering results are both **interpretable and flexible**, especially in datasets with **overlapping or ambiguous regions**.

🔹 Q. Explain the concept of clustering in unsupervised learning.

📘 1. What is Unsupervised Learning?

Unsupervised learning is a type of machine learning where the algorithm is given **input data without any labeled output**.
Its goal is to discover **hidden patterns, structures, or groupings** in the data.

🔍 2. What is Clustering?

Clustering is a core task in unsupervised learning that involves grouping a set of data points such that:

- **Data points within the same group (cluster)** are **similar** to each other.
- **Data points in different clusters** are **dissimilar** from each other.

Clusters represent **natural groupings or structures** in data, and are formed based on **similarity/distance measures** (e.g., Euclidean distance).

🧠 3. Objective of Clustering

To partition a dataset $X = \{x_1, x_2, ..., x_n\}$ into K clusters $C_1, C_2, ..., C_K$ such that:

- Each x_i belongs to **one (or more)** cluster(s)
- The clustering minimizes **intra-cluster distance** and maximizes **inter-cluster distance**

📊 4. Types of Clustering Algorithms

Type	Description
Partition-based	Divides data into non-overlapping clusters (e.g., K-Means, K-Medoids)
Hierarchical	Builds a nested tree of clusters (e.g., Agglomerative Clustering)
Density-based	Forms clusters based on dense regions (e.g., DBSCAN)
Fuzzy/Soft	Allows points to belong to multiple clusters (e.g., Fuzzy C-Means)

⚙️ 5. Common Clustering Techniques

- **K-Means**: Uses centroids and assigns each point to the nearest one.
- **K-Medoids**: Uses actual data points as cluster centers.
- **Hierarchical Clustering**: Builds clusters incrementally via linkage methods.
- **Fuzzy C-Means**: Assigns soft (fuzzy) memberships to all clusters.

📌 6. Applications of Clustering

Domain	Application
Marketing	Customer segmentation
Image Processing	Image compression, object detection

Domain	Application
Biology	Gene expression clustering
Text Mining	Document and topic clustering
Anomaly Detection	Identifying outliers in data

Clustering is a powerful technique in unsupervised learning that helps discover **natural groupings** in data. It is widely applicable in various fields for pattern analysis, data compression, and exploratory data analysis. The effectiveness of clustering depends on the **algorithm, distance metric, and problem characteristics**.

Comparison Table: Single, Complete, and Average Linkage

Aspect	Single Linkage	Complete Linkage	Average Linkage
Definition	Based on the closest pair of points	Based on the farthest pair of points	Based on the average of all pairwise distances
Cluster Shape	Forms long, chain-like clusters	Forms tight, compact clusters	Produces balanced and smooth clusters
Sensitivity to Outliers	High (one close point can link clusters)	Moderate (outliers can dominate max distance)	Low (averaging reduces outlier impact)
Chaining Effect	Very prone	Not prone	Rarely occurs
Computation Cost	Low	Low	Moderate (more pairwise comparisons)
Best Suited For	Arbitrary-shaped clusters	Compact, spherical, well-separated clusters	Balanced cases between chaining and compactness

🔹 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

📘 1. Introduction

DBSCAN is a powerful **density-based clustering algorithm** used in unsupervised learning.
Unlike partitioning methods like **K-Means**, DBSCAN can find **clusters of arbitrary shape**, and it is also capable of **detecting noise (outliers)** in the dataset.

It is particularly useful for **spatial data** and **real-world applications** where clusters may not be spherical or clearly separated.

🧠 2. Main Idea

A cluster is a **group of densely packed points**, separated by regions of **low point density**.

The algorithm groups together points that are closely packed and labels points that lie alone in low-density regions as **noise**.

🔧 3. Key Definitions (Terminologies)

🔹 a. ε-neighborhood

- The **ε-neighborhood** of a point refers to the **set of all data points** within a given distance ε (epsilon) from that point.
- Mathematically:
  For a point p,

$$N_\varepsilon(p) = \{q \in D \mid d(p,q) \leq \varepsilon\}$$

- It defines the **local neighborhood** around a point.

"Find **all the points** q in the dataset D that are **within a distance ε** from point p."

In Simple Words:

- You pick a point p (could be any point in your data).
- You **draw a circle of radius ε** around that point.
- Then, you **look at all the other points** inside that circle.
- All those points are in the **ε-neighborhood** of point p.

🔹 **b. Core Object**

- A point p is called a **core object** if:
  - The number of points in its ε-neighborhood is **greater than or equal to** a specified threshold **MinPts**.
- That is:

$$|N_\varepsilon(p)| \geq \text{MinPts}$$

- These are considered **dense** regions and are used to **initiate cluster growth**.

- $N_\varepsilon(p)$: The number of points inside the ε-neighborhood of point p.
- $|N_\varepsilon(p)|$: Count how many points are there.
- If that count is ≥ **MinPts**, then it's **dense enough** → it's a **core point**.

🔹 **c. Directly Density-Reachable**

- A point q is **directly density-reachable** from a point p if:
  - $q \in N_\varepsilon(p)$, i.e., it lies within ε-neighborhood of p, **and**
  - p is a **core object**.
- Note: This relationship is **asymmetric** — if q is not a core object, it **may not directly reach back** to p.

Imagine This Scenario:

- You have two points:
  - **p = a core point**
  - **q = a normal (non-core) point**
- Now:
  - q is **within ε distance** from p ✅
  - So, we say:
    👉 "q is **directly density-reachable from** p"

BUT now ask:

Can we say the reverse — that p is directly density-reachable from q?

No! Because:

q is **not a core point**, so it **doesn't have enough density (MinPts)** to let clustering spread from it.

🔹 d. **Density-Reachable**

- A point p is **density-reachable** from point q if:
  - There exists a **chain of points** $p_1, p_2, \ldots, p_n$ such that:
    - $p_1 = q, p_n = p$
    - Each p_{i+1} is **directly density-reachable** from p_i
- This property is **transitive** but **not symmetric**.

🔹 e. **Density-Connected**

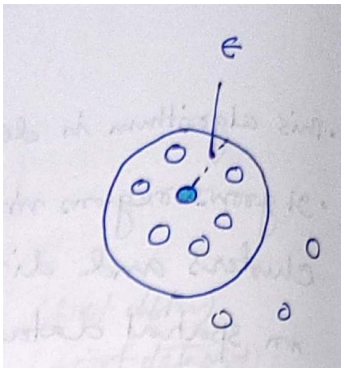
- Two points p and q are **density-connected** if:
  - There exists a third point o such that:
    - Both p and q are **density-reachable from** o
    - o must be a **core point**
- This property is **symmetric** — it defines mutual connectivity.

✅ What is Epsilon (ε)?

- **Epsilon (ε)** is a **distance threshold** that defines the **neighborhood radius** around a point.
- It tells us **how far to look** from a given point to find its neighbors.
- In the diagram:
  - The **dotted circle** around the green point represents the ε-radius.
  - All points that lie **inside this radius** are considered part of the **ε-neighborhood**.

✅ What is MinPts (Minimum Points)?

- **MinPts** is the **minimum number of data points** (including the point itself) that must be present **within the ε-neighborhood** for a point to qualify as a **core point**.
- This helps DBSCAN decide whether a region is **dense enough** to form a cluster.



Interpreting the Diagram

- **Green circle**:
  → Randomly selected point (starting point).

- **Grey circles inside the dotted boundary**:
  → Data points that fall **within ε distance** of the green point.
  → These are part of the green point's **ε-neighborhood**.
- If the number of these grey points (including the green point) ≥ MinPts,
  → Then the **green point is a core point**, and
  → A **cluster will begin forming** from it.
- **Points outside the circle**:
  → Either **border points** (if reachable from a core point) or **noise** (if isolated).

Q. You are given the following 2D data points: P1: (1, 2)

P2: (2, 2)

P3: (2, 3)

P4: (8, 7)

P5: (8, 8)

P6: (7, 7)

P7: (25, 80)

Use **DBSCAN** with the following parameters:

- **ε = 1.5**
- **MinPts = 3**

Identify for each point whether it is a:

- **Core Point**
- **Border Point**
- **Noise Point**

🔹 Step 1: Understand Criteria

A point is a **core point** if:

- It has **MinPts = 3** (including itself) in its **ε-neighborhood**.

🔹 Step 2: Calculate ε-neighborhoods

Let's go one by one:

▶️ P1 (1, 2):

- **Distance to P2** = $\sqrt{[(2-1)^2 + (2-2)^2]} = \sqrt{1} = 1.0$ ✅
- **Distance to P3** = $\sqrt{[(2-1)^2 + (3-2)^2]} = \sqrt{2} \approx 1.41$ ✅

→ P1's ε-neighborhood = {P1, P2, P3} ⇒ **3 points**

✅ So **P1 is a Core Point**

▶️ P2 (2, 2):

- Dist to P1 = 1.0 ✅
- Dist to P3 = 1.0 ✅

→ P2's ε-neighborhood = {P1, P2, P3} ⇒ **3 points**

✅ **Core Point**

▶️ P3 (2, 3):

- Dist to P1 = 1.41 ✅
- Dist to P2 = 1.0 ✅

→ P3's ε-neighborhood = {P1, P2, P3} ⇒ **3 points**

✅ **Core Point**

▶️ P4 (8, 7):

- Dist to P5 = $\sqrt{[(8-8)^2 + (7-8)^2]} = 1$ ✅
- Dist to P6 = $\sqrt{[(8-7)^2 + (7-7)^2]} = 1$ ✅

→ {P4, P5, P6} ⇒ **3 points**

✅ **Core Point**

▶️ P7 (25, 80):

- Very far from everyone (>60 units)

→ Only itself in ε-neighborhood ⇒ **1 point < MinPts**

❌ **Not a core**, ❌ not reachable from any core ⇒

🚫 **Noise Point**

▶️ P5 (8, 8):

- Dist to P4 = 1 ✅
- Dist to P6 = $\sqrt{[(8-7)^2 + (8-7)^2]} = \sqrt{2} \approx 1.41$ ✅

→ {P5, P4, P6} ⇒ **3 points**

✅ **Core Point**

▶️ P6 (7, 7):

- Dist to P4 = 1 ✅
- Dist to P5 = 1.41 ✅

→ {P6, P4, P5} ⇒ **3 points**

✅ **Core Point**

✅ Advantages and Disadvantages of DBSCAN

🌟 Advantages of DBSCAN

1. **No Need to Specify Number of Clusters**
   - Unlike K-Means, DBSCAN does **not require** the number of clusters (K) to be predefined.
2. **Detects Arbitrary Shaped Clusters**
   - It can find **non-spherical** and **non-linearly separable** clusters, unlike K-Means which works best with spherical clusters.
3. **Robust to Outliers**
   - DBSCAN can identify and label **noise or outliers**, which is very useful in real-world datasets.
4. **Works Well with Clusters of Different Sizes**
   - Handles clusters of **varying densities and shapes** (to some extent), which partition-based algorithms struggle with.
5. **Simple Concept Based on Density**
   - The algorithm is conceptually straightforward: cluster = dense region separated by sparse regions.

⚠️ Disadvantages of DBSCAN

1. **Sensitive to Parameter Selection (ε and MinPts)**
    - Choosing an appropriate **ε (epsilon)** and **MinPts** is crucial and often difficult without domain knowledge.
2. **Fails on Clusters with Varying Densities**
    - DBSCAN assumes uniform density within a cluster; it may struggle when clusters have **varying densities**.
3. **Poor Performance in High Dimensions**
    - As dimensionality increases, **distance measures become less meaningful**, and DBSCAN's effectiveness decreases (curse of dimensionality).
4. **Cannot Handle Well-Separated, Nested Clusters**
    - Clusters within clusters (nested) are not easily discovered.
5. **Not Deterministic**
    - Slight changes in ε or data order may lead to different clustering results.