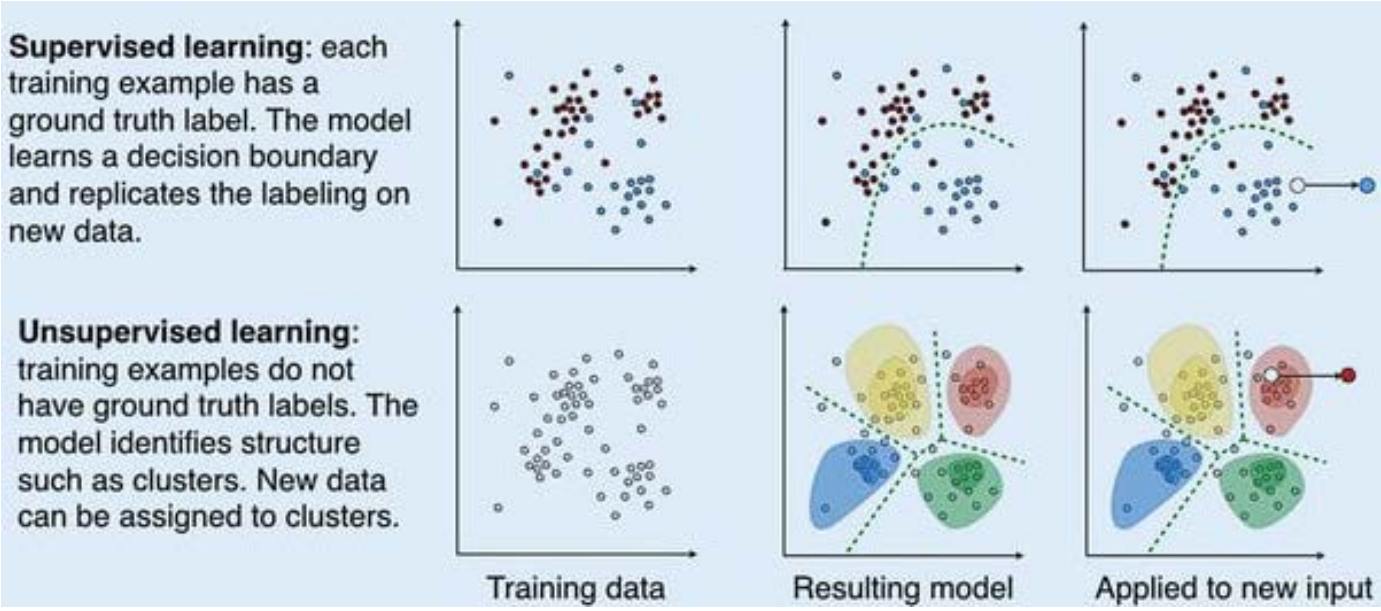
Chapter 05: Unsupervised Learning

🔷 Introduction to Unsupervised Learning

Unsupervised Learning is a type of **machine learning** where a model is trained on data **without labeled outputs.** Unlike supervised learning (where each input has a known target), here the algorithm must **discover patterns, structures, or relationships on its own.**

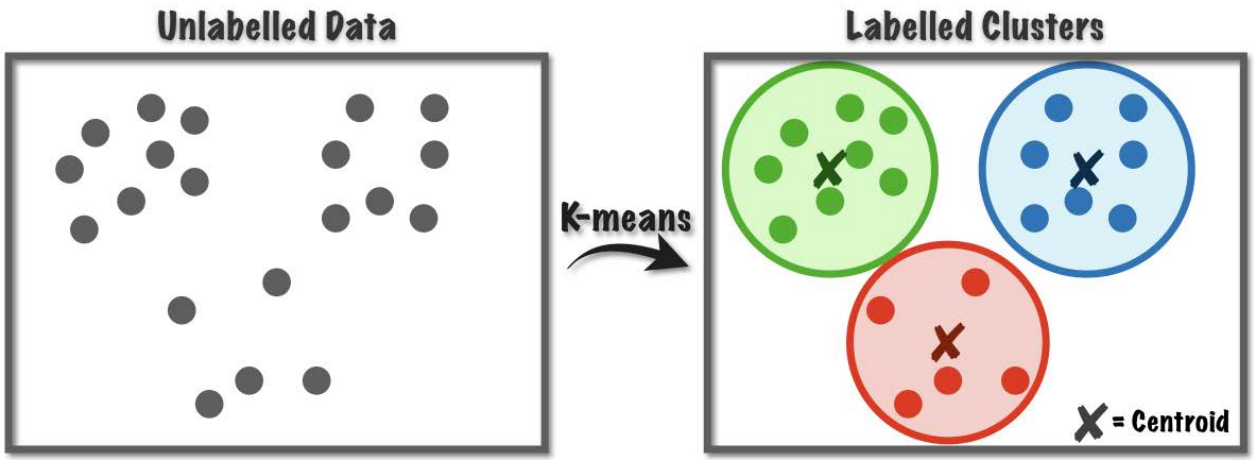


🔷 1. Definition of Clustering

Clustering is an **unsupervised learning technique** used to group data points into clusters such that:

- Points in the **same cluster are similar**
- Points in **different clusters are dissimilar**

👉 Unlike supervised learning, there are **no labels** provided.



🔷 2. Mathematical Idea of Clustering

Let a dataset be:

$$X = \{x_1, x_2, ..., x_n\}, \quad x_i \in \mathbb{R}^d$$

We want to divide it into **K clusters**:

$$C = \{C_1, C_2, ..., C_K\}$$

Objective:

Minimize **intra-cluster distance** and maximize **inter-cluster distance**

📌 Example: K-Means Objective Function

$$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

$J = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$

Where:

- μ_k: centroid of cluster C_k
- $\|x_i - \mu_k\|$: Euclidean distance

🔷 Distance Measure (Most Common)

$$d(x_i, x_j) = \sqrt{\sum_{m=1}^{d} (x_{im} - x_{jm})^2}$$

👉 This is the **Euclidean distance**, widely used in engineering problems.

🔷 3. Intuition

Clustering tries to answer:

> "Which data points naturally belong together?"

Example:

- Similar voltage-current patterns → same load type
- Similar signal shapes → same system behavior

🔷 4. Electrical Engineering Example (with Solution)

⚡ Problem: Load Type Identification

You measure electrical loads with:

- Voltage V
- Current I

Dataset:

Load	Voltage (V)	Current (A)
L1	220	2.0
L2	230	2.2
L3	110	5.0
L4	115	4.8

🔷 Step 1: Represent Data

$$x_1 = (220, 2.0), \quad x_2 = (230, 2.2)$$

$$x_3 = (110, 5.0), \quad x_4 = (115, 4.8)$$

🔷 Step 2: Apply Clustering Idea

We expect **2 clusters:**

- Cluster 1: High voltage, low current
- Cluster 2: Low voltage, high current

🔷 Step 3: Compute Distance Example

Distance between L1 and L2:

$$d = \sqrt{(220 - 230)^2 + (2.0 - 2.2)^2}$$

$$d = \sqrt{100 + 0.04} = \sqrt{100.04} \approx 10$$

👉 Small distance → same cluster

Distance between L1 and L3:

$$d = \sqrt{(220 - 110)^2 + (2.0 - 5.0)^2}$$

$$d = \sqrt{12100 + 9} = \sqrt{12109} \approx 110$$

👉 Large distance → different clusters

🔷 Step 4: Final Clusters

- **Cluster 1:** L1, L2 → (Resistive loads)
- **Cluster 2:** L3, L4 → (Inductive loads)

🔷 5. Interpretation (Engineering Insight)

Clustering helps to:

- Identify **types of electrical loads**
- Detect **abnormal behavior**
- Group **similar signals or faults**

```
import numpy as np
from sklearn.cluster import KMeans

# Data: Voltage, Current
X = np.array([
    [220, 2.0],
    [230, 2.2],
    [110, 5.0],
    [115, 4.8]
])

# Apply K-means
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)

# Results
print("Cluster labels:", kmeans.labels_)
print("Centroids:", kmeans.cluster_centers_)
```

```
Initializing environment
Installing packages
Running code
Cluster labels: [1 1 0 0]
Centroids: [[112.5   4.9]
 [225.    2.1]]
```

📘 DBSCAN Algorithm (Density-Based Clustering)

🔷 1. Definition

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a clustering algorithm that:

- Groups together **points that are close (dense regions)**
- Marks points in low-density regions as **noise (outliers)**

👉 Unlike K-means:

- No need to specify number of clusters K
- Can detect **arbitrary shapes**
- Handles **noise naturally**

🔷 2. Key Concepts

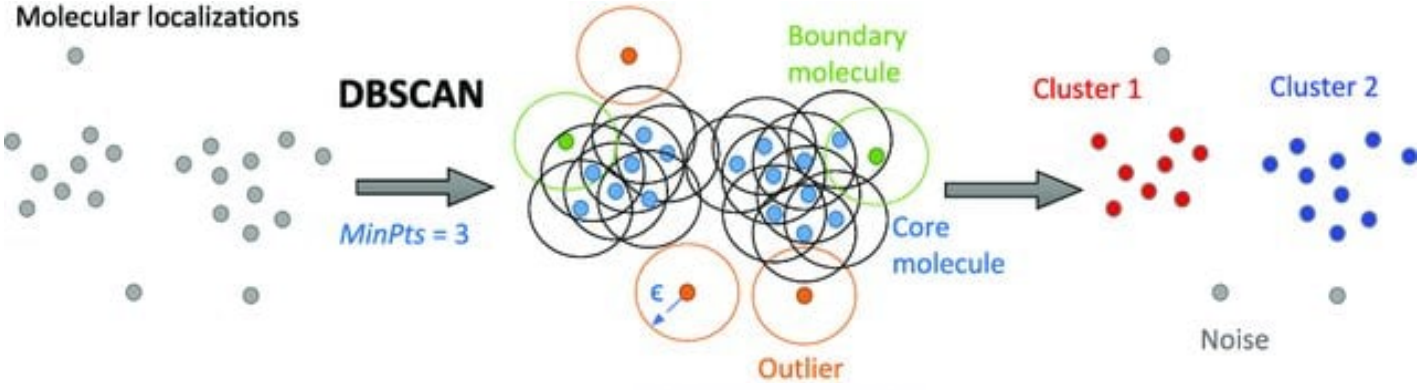
DBSCAN relies on **2 parameters:**

1. ε (Epsilon)

- Radius of neighborhood around a point

2. MinPts

- Minimum number of points required to form a dense region



📌 ε-Neighborhood

$$N_\varepsilon(x_i) = \{x_j \in X \mid d(x_i, x_j) \leq \varepsilon\}$$

$N_\varepsilon(x_i) = \{x_j \in X \mid d(x_i, x_j) \leq \varepsilon\}$

📌 Core Point

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

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

👉 It has enough neighbors → dense region

📌 Border Point

- Not a core point
- But inside neighborhood of a core point

📌 Noise Point

- Not reachable from any core point

🔷 4. DBSCAN Algorithm Steps

1. Choose a point x_i
2. Compute its ε-neighborhood
3. If it has ≥ MinPts → **start a cluster**
4. Expand cluster by including all density-reachable points
5. Repeat for all points

🔷 6. Electrical Engineering Example (with Solution)

⚡ Problem: Fault Detection in Electrical Signals

We measure:

- Voltage deviation (ΔV)
- Current deviation (ΔI)

Dataset:

Point	ΔV	ΔI
P1	0.1	0.2
P2	0.2	0.1
P3	0.15	0.2
P4	5.0	5.1
P5	5.2	5.0
P6	20	1

🔷 Step 1: Choose Parameters

- $\varepsilon = 0.5$
- MinPts = 2

🔷 Step 2: Compute Neighborhoods

Example:

For P1:

$$d(P1, P2) = \sqrt{(0.1 - 0.2)^2 + (0.2 - 0.1)^2} \approx 0.14$$

👉 Inside ε → neighbor

Similarly:

- P1, P2, P3 are close → **Cluster 1**

For P4:

$$d(P4, P5) \approx 0.22$$

👉 Form **Cluster 2**
For P6:

- Far from all points → **Noise**

🔷 Step 3: Final Result

- **Cluster 1:** P1, P2, P3 → Normal operation
- **Cluster 2:** P4, P5 → Fault type A
- **Noise:** P6 → Anomaly / critical fault

‹/› Python

```python
import numpy as np
from sklearn.cluster import DBSCAN

# Data
X = np.array([
    [0.1, 0.2],
    [0.2, 0.1],
    [0.15, 0.2],
    [5.0, 5.1],
    [5.2, 5.0],
    [20, 1]
])

# DBSCAN
db = DBSCAN(eps=0.5, min_samples=2)
labels = db.fit_predict(X)

print("Cluster labels:", labels)
```

👉 Output example:

```
[0 0 0 1 1 -1]
```

- `0,1` → clusters
- `-1` → noise

🔷 11. DBSCAN vs K-Means

Feature	K-Means	DBSCAN
Need K	Yes	No
Shape	Spherical	Any shape
Noise handling	Poor	Excellent
Speed	Fast	Medium

📊 5. Code Python pour visualisation 2D

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
# Data
X = np.array([
    [0.1, 0.2],
    [0.2, 0.1],
    [0.15, 0.2],
    [5.0, 5.1],
    [5.2, 5.0],
    [20, 1]
])
# DBSCAN
db = DBSCAN(eps=0.5, min_samples=2)
labels = db.fit_predict(X)
# Plot
plt.figure()

for label in set(labels):
    cluster = X[labels == label]
    if label == -1:
        plt.scatter(cluster[:,0], cluster[:,1], color='red', marker='x', s=100, label='Noise')
    else:
        plt.scatter(cluster[:,0], cluster[:,1], s=80, label=f'Cluster {label}')
plt.xlabel("ΔV (Voltage variation)")
plt.ylabel("ΔI (Current variation)")
plt.title("DBSCAN Clustering - Electrical Data")
plt.legend()
plt.grid()
plt.show()
```

```
Initializing environment

Installing packages

Running code
```