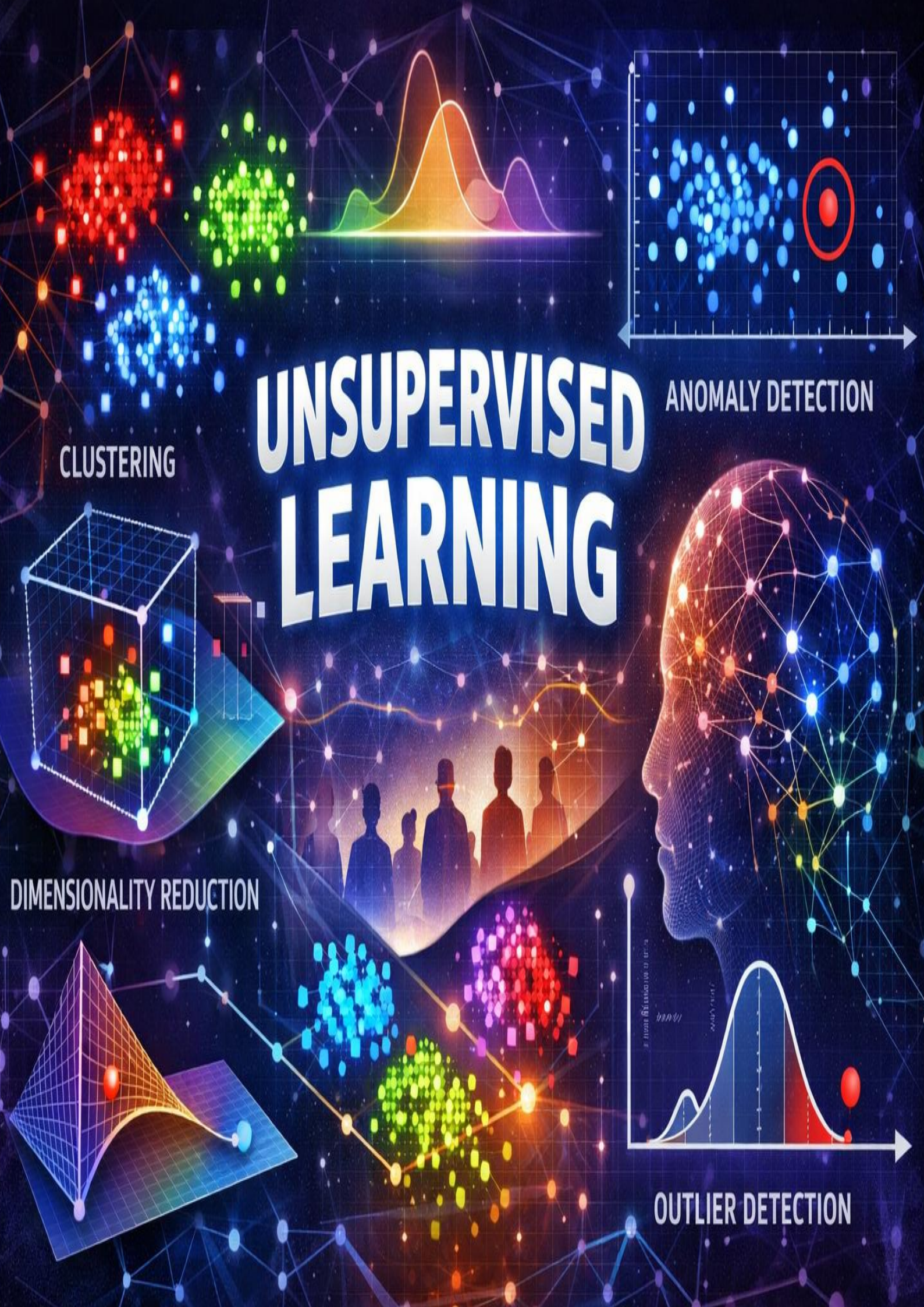

UNSUPERVISED LEARNING
CLUSTERING
ANOMALY DETECTION
DIMENSIONALITY REDUCTION
OUTLIER DETECTION

Table of Contents

Introduction

Purpose

Similar to the supervised learning module, this document aims to bridge theoretical knowledge with practical application, specifically for unsupervised learning. It is designed to help learners understand the theoretical foundations of clustering algorithms and selecting the appropriate technique for discovering patterns in unlabeled data.

Scope

The document focuses on the primary unsupervised learning technique: **Clustering**. It specifically covers **K-Means Clustering**, **Hierarchical Clustering** (Agglomerative), and **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise). Additionally, it includes a comprehensive section on **Clustering Evaluation Metrics**, such as the Silhouette Score, Davies-Bouldin Index, and Adjusted Rand Index.

Procedure

The content is presented through a methodical approach mirroring the supervised learning study:

1. **Theoretical Study:** Explaining the mechanics of algorithms, such as the K-Means objective function and DBSCAN density concepts.
2. **Implementation:** Providing Python (scikit-learn) code examples for each model.
3. **Visualization:** Using dendrograms and scatter plots to visualize clusters.
4. **Evaluation:** detailing internal and external metrics to assess cluster quality.
5. **Guidelines:** offering specific rules on "When to use each Model" based on data shape and density.

Brief Problem

The primary challenge addressed is the analysis of unlabeled data where no ground truth exists. Finding meaningful structures in complex datasets requires choosing between algorithms that make different assumptions—such as K-Means (which assumes spherical clusters) versus DBSCAN (which handles irregular shapes and noise).

Clustering

K-means Clustering Method

K-Means is an **unsupervised learning algorithm** used to group data into **K clusters** based on similarity (distance).

- Each cluster is represented by its **centroid** (mean of points in that cluster).
- Objective: minimize the **within-cluster variance** (distance between points and their cluster centroid).

---

1. How K-Means Works (Step by Step)

1. **Choose K** (the number of clusters).
2. **Initialize centroids** randomly.
3. **Assign each point** to the nearest centroid (based on Euclidean distance).
4. **Recalculate centroids** as the mean of all points assigned to each cluster.
5. **Repeat steps 3–4** until convergence (no change in centroids or max iterations reached).

---

2. Objective Function (Cost Function)

K-Means minimizes the **sum of squared distances** between points and their assigned cluster centroids:

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

Where:

- K = number of clusters
- Ci = set of points in cluster i
- μi = centroid of cluster i

Made by Youssef Amgad Elkhatib

3. Choosing K (Elbow Method)

- Try different values of K.
- Plot **K vs cost (inertia)**.
- The “elbow point” (where cost stops decreasing sharply) is usually the best K.

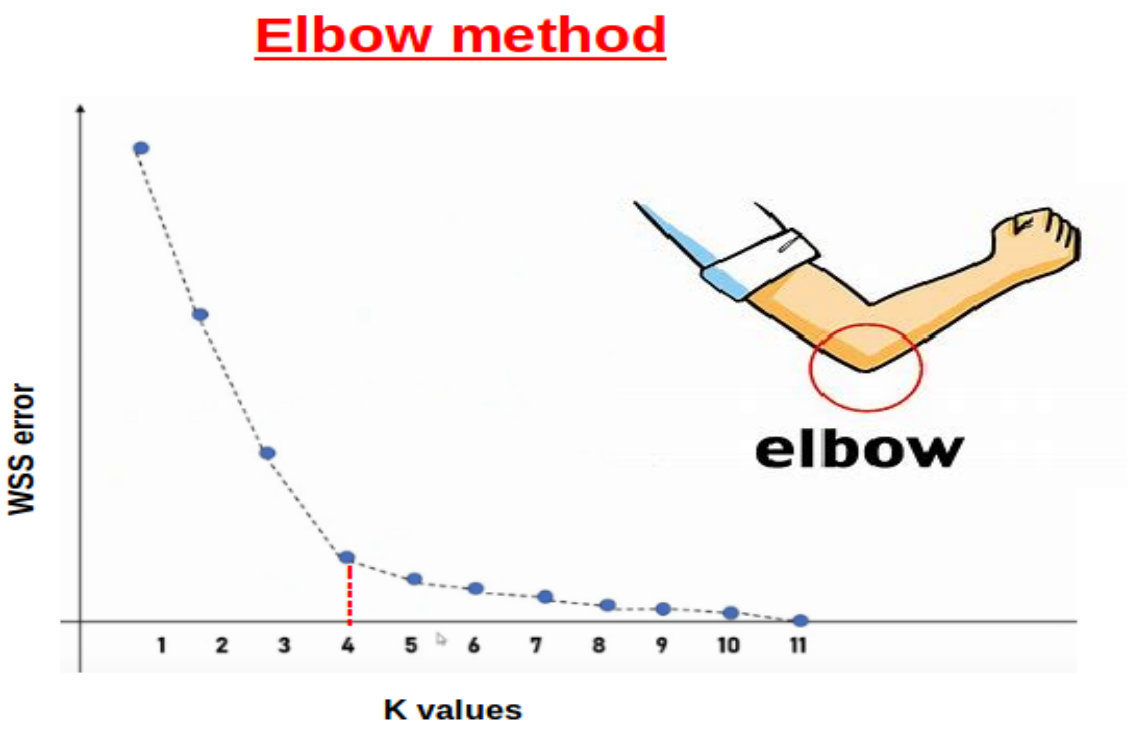
```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic dataset
X, _ = make_blobs(n_samples=500, centers=5, cluster_std=0.60, random_state=42)

inertia_values = []  # store cost (inertia) for each K
K_range = range(1, 11)  # test K = 1 to 10

for k in K_range:
    kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
    kmeans.fit(X)
    inertia_values.append(kmeans.inertia_)

# Plot
plt.plot(K_range, inertia_values, marker='o')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia (Sum of Squared Distances)")
plt.title("Elbow Method for Optimal K")
plt.show()
```



4. Code Example for K-means Clustering

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# Generate synthetic data
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.60, random_state=42)

# Fit KMeans
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
y_kmeans = kmeans.fit_predict(X)

# Plot results
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            s=200, c='red', marker='X')
plt.title("K-Means Clustering")
plt.show()
```

---

5. Parameters of KMeans (scikit-learn)

Parameter	Type	Default	Description
n_clusters	int	8	Number of clusters.
init	{‘k-means++’, ‘random’}	‘k-means++’	How to initialize centroids.
n_init	int	10	Number of times algorithm runs with different seeds.
max_iter	int	300	Maximum iterations per run.
tol	float	1e-4	Convergence tolerance.
random_state	int	None	Seed for reproducibility.
algorithm	{‘lloyd’, ‘elkan’}	‘lloyd’	K-Means algorithm variant.

Hierarchical Clustering

Unlike **K-Means**, where you predefine the number of clusters, **Hierarchical Clustering** builds a hierarchy (tree-like structure) of clusters.
It produces a **dendrogram** (tree diagram) showing how data points merge or split.

Two main approaches:

1. **Agglomerative (Bottom-Up, most common)**
    - Start with each point as its own cluster.
    - Iteratively merge the two closest clusters until only one remains.
2. **Divisive (Top-Down, less common)**
    - Start with one big cluster.
    - Recursively split clusters until each point is its own cluster.
    - Not in scikit-learn

---

1. Distance Metrics

You need to define how to measure the distance between **clusters**, not just points.

Common linkage methods:

- **Single Linkage** → distance between the two closest points.
- **Complete Linkage** → distance between the two farthest points.
- **Average Linkage** → average distance between all pairs of points.
- **Ward's Method** → minimizes variance within clusters (like variance in K-Means).

Made by Youssef Amgad Elkhatib

2. Code Example: Agglomerative Clustering

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_blobs
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

# Generate sample data
X, _ = make_blobs(n_samples=100, centers=3, cluster_std=0.8, random_state=42)

# --- Step 1: Plot dendrogram ---
Z = linkage(X, method='ward')  # 'ward', 'single', 'complete', 'average'
plt.figure(figsize=(10, 6))
dendrogram(Z)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Data Points")
plt.ylabel("Distance")
plt.show()

# --- Step 2: Apply Agglomerative Clustering ---
hc = AgglomerativeClustering(n_clusters=3, linkage="ward")
labels = hc.fit_predict(X)

# --- Step 3: Plot clustered data ---
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='rainbow', s=50)
plt.title("Agglomerative Clustering (Ward Linkage)")
plt.show()
```

---

3. Parameters of AgglomerativeClustering()

Parameter	Type	Default	Description
n_clusters	int	2	Number of clusters to find.
affinity	str	'euclidean'	Metric for distance ('euclidean', 'manhattan', 'cosine', etc.).
linkage	str	'ward'	How to merge clusters: 'ward', 'complete', 'average', 'single'.
distance_threshold	float	None	If set, defines the cut-off distance for merging (overrides n_clusters).
compute_full_tree	bool or str	'auto'	Whether to build the full tree.
compute_distances	bool	False	If True, stores distances between merges.

DBSCAN

DBSCAN is a **density-based clustering algorithm** that groups together points that are **close (densely packed)** and marks points that lie alone in low-density regions as **outliers (noise)**.

Unlike **K-Means**, DBSCAN does **not require specifying the number of clusters (k)** in advance — it automatically detects them based on density.

---

1. How DBSCAN Works (Step-by-Step)

DBSCAN relies on two key parameters:

- **ε (epsilon)** — the **radius of the neighborhood** around a point.
- **minPts** — the **minimum number of points** required to form a dense region.

Step 1: Classify Points

For each point p:

1. **ε-neighborhood:** Find all points within distance ε of p.

2. **Classify** p:
    - **Core Point:** if $|N(p)| \geq$ minPts
    - **Border Point:** if $|N(p)| <$ minPtsbut is in the neighborhood of a core point
    - **Noise Point:** if it's not a core or border point

Step 2: Form Clusters

- Pick an **unvisited core point** and create a **new cluster**.
- Add all points in its ε-neighborhood.
- For each point in that cluster that is also a **core point**, expand the cluster with its neighborhood.
- Continue until no more points can be added.
- Move on to the next unvisited core point.

Step 3: Mark Noise

- Any points not assigned to a cluster after all core points are processed are labeled as **noise (outliers)**.

2. Key Idea Behind DBSCAN

Dense areas of data points form clusters, and sparse areas (low density) represent noise.

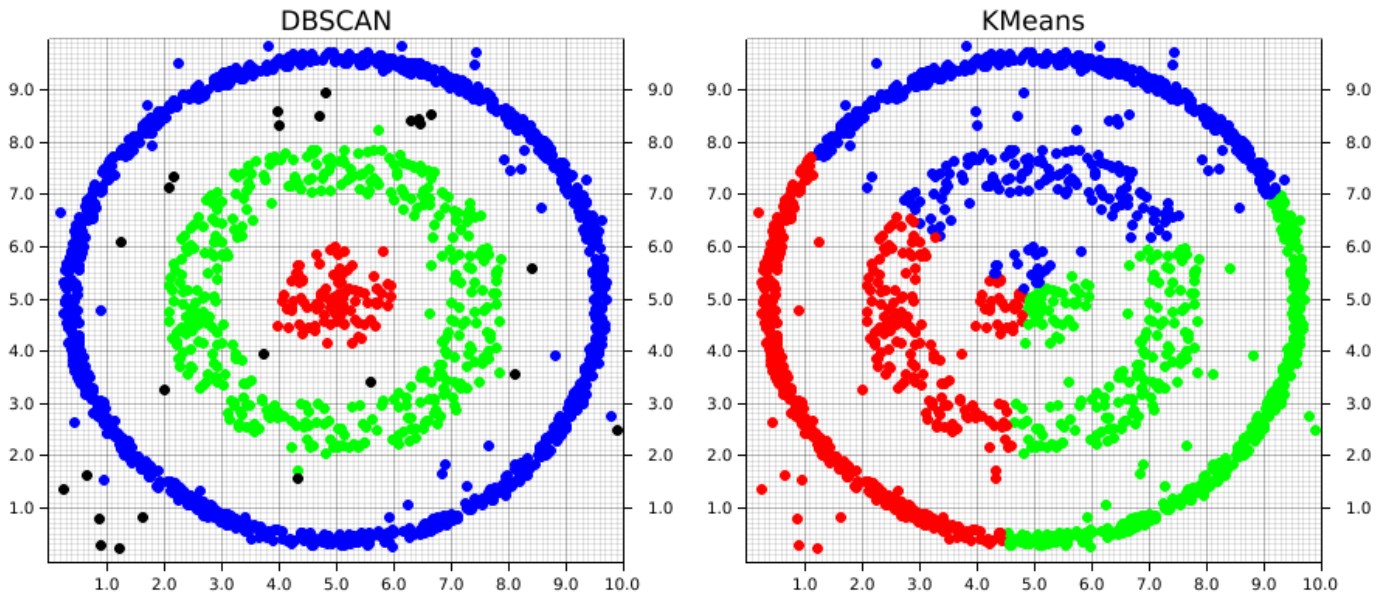
---

3. Cost Function

DBSCAN doesn't optimize a **traditional cost function** like K-Means (which minimizes sum of squared distances).

Instead, it's **rule-based**:

- It clusters based on density reachability.
- No explicit objective function is minimized.
- However, performance can be evaluated using **silhouette score**, **Davies-Bouldin index**, or **Adjusted Rand Index** (if ground truth available).



4. Code Example

```
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt

# Create synthetic dataset
X, y = make_moons(n_samples=500, noise=0.05, random_state=42)

# Train DBSCAN model
dbscan = DBSCAN(eps=0.2, min_samples=5, metric='euclidean')
labels = dbscan.fit_predict(X)

# Plot clusters
plt.figure(figsize=(8,5))
plt.scatter(X[:,0], X[:,1], c=labels, cmap='plasma', s=30)
plt.title("DBSCAN Clustering Example")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show()
```

---

5. DBSCAN Parameters

Parameter	Description	Effect on Model
eps (ε)	Maximum distance between two points to be considered neighbors	Too small → many points labeled noise; too large → clusters merge
min_samples	Minimum number of points to form a dense region	Higher → fewer, larger clusters; lower → more, smaller clusters
metric	Distance metric (default: 'euclidean')	Affects neighborhood computation
algorithm	Search algorithm ('auto', 'ball_tree', 'kd_tree', 'brute')	Affects performance (not results)
leaf_size	Leaf size for tree-based searches	Speed trade-off
n_jobs	Number of parallel jobs	Speed improvement

When to use each Model

1. K-Means

- **Use when:**
  - You want to group data into **k known clusters**.
  - Data has **spherical (round) clusters** with similar sizes.
- **Don't use when:** clusters are not convex (like moons, spirals).

---

2. Hierarchical Clustering

- **Use when:**
  - You want a **tree-like hierarchy (dendrogram)**.
  - Number of clusters is unknown.
- **Good for:** small datasets, exploratory analysis.
- **Don't use when:** dataset is very large ($O(n^2)$ complexity).

---

3. DBSCAN

- **Use when:**
  - Clusters are **irregular / non-spherical** (e.g., moons, spirals).
  - You **don't know the number of clusters** beforehand.
  - Data contains **noise or outliers** that should be excluded.
  - You want to find **dense regions separated by sparse areas**.
  - Dataset size is **small to medium** (thousands to tens of thousands of samples).
  - Useful for **spatial, pattern, or anomaly detection** tasks.
- **Don't use when:**
  - Clusters have **very different densities** (one eps won't fit all).
  - Data is **high-dimensional** (distance loses meaning).
  - Dataset is **very large** (computationally heavy).
  - You **need a fixed number of clusters** (DBSCAN finds them automatically).

Clustering Evaluation Metrics

1. Internal Clustering Metrics

These do **not** require ground-truth labels. They evaluate the "goodness" of clustering by measuring **compactness** (how close points in the same cluster are(intra-cluster)) and **separation** (how far apart different clusters are(inter-cluster)).

Silhouette Score

- Ranges: **-1 → 1** (higher is better).
- Formula:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

  where:

  - a(i) = average distance between point i and all other points in the same cluster (intra-cluster distance).
  - b(i) = minimum average distance from i to points in another cluster (nearest-cluster distance).

- Interpretation:
  - **1** → point is well clustered.
  - **0** → point is on the decision boundary.
  - **-1** → point is in the wrong cluster.

Davies–Bouldin Index (DBI)

- Ranges: **0 → ∞** (lower is better).
- Measures average similarity between each cluster and its most similar one.
- Small values mean clusters are compact and well-separated.

Dunn Index

- Higher is better.
- Ratio of minimum inter-cluster distance to maximum intra-cluster distance.
- Encourages clusters that are far apart and internally tight.

Calinski–Harabasz Index (Variance Ratio Criterion)

- Higher is better.
- Ratio of between-cluster dispersion to within-cluster dispersion.

2. External Clustering Metrics

These **require ground-truth labels**. They measure how well the clustering result matches the true partition.

Adjusted Rand Index (ARI)

- Ranges: **-1 → 1** (1 = perfect clustering, 0 = random, <0 = worse than random).
- Measures similarity between two partitions, correcting for chance.

Normalized Mutual Information (NMI)

- Ranges: **0 → 1**.
- Measures mutual dependence between true labels and predicted clusters.
- Higher = better alignment.

Homogeneity, Completeness, V-Measure

- **Homogeneity**: each cluster contains only members of a single class.
- **Completeness**: all members of a given class are assigned to the same cluster.
- **V-measure**: harmonic mean of homogeneity and completeness.

---

3. Relative Clustering Metrics

These are not direct quality scores, but **comparisons across different clustering results** (e.g., different K in K-means).

- **Elbow Method** → uses inertia (sum of squared distances within clusters) to pick optimal K.
- **Gap Statistic** → compares clustering performance with that of a random uniform distribution.

Conclusion

The document concludes that the strength of AI lies in understanding the theoretical suitability of various models rather than relying on a single approach. Techniques like K-means and hierarchical clustering enable the discovery of hidden patterns in unlabeled data, while methods like DBSCAN are essential for handling noise and irregular clusters. Success in unsupervised learning depends on combining theory, visualization, and application to select the right technique for the specific domain.

Feedback & Contribution

This is a living document. If you have feedback, suggestions, or additional insights that could improve it, I welcome your input. Sharing ideas helps keep this material accurate, relevant, and valuable for everyone.

Copyright & Usage

Acknowledgments

This document is authored and copyrighted by **[Youssef Amgad Elkhatib]**.
Special thanks to the contributors who provided valuable feedback and suggestions for improvements:

Name	Contribution