MachineLearning _Clustering

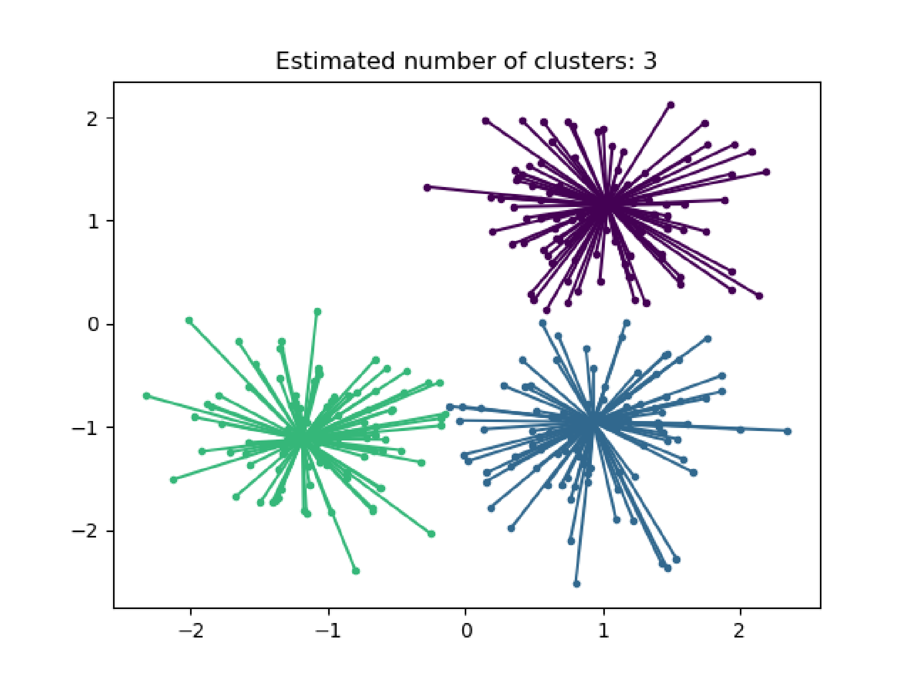
-Rathiga Ramesh

Full Stack Developer,AI Enthusiast

Affinity Propagation

- **How it works**:
  - Uses "message passing" between data points to identify **exemplars** (most representative points).
  - Does **not require pre-specifying** the number of clusters.
- **Strengths**:
  - Works well with **small to medium-sized datasets**.
  - Finds **natural clusters** without forcing a fixed structure.
- **Weaknesses**:
  - **High computational cost** ($O(N^2)$ memory/time complexity).
  - Sensitive to **damping factor** and **preference parameter**.
- **Best for**:
  - Small datasets where the number of clusters is unknown (e.g., gene expression, image segmentation).

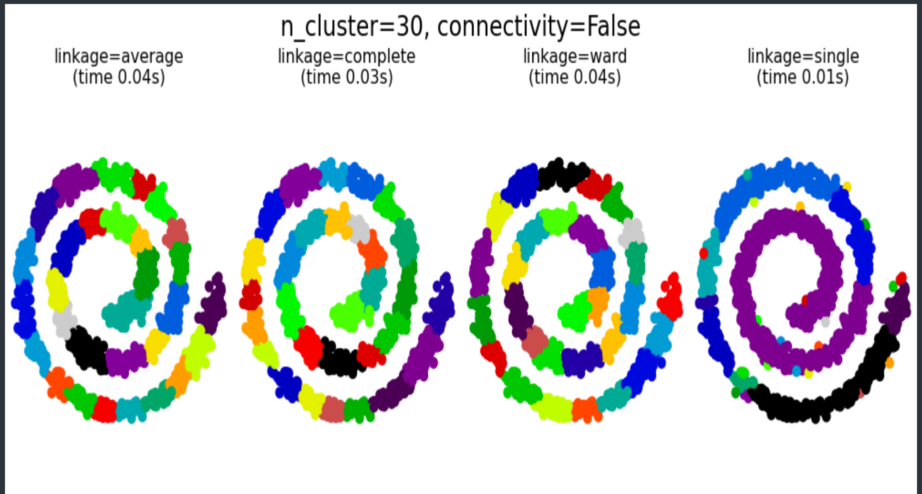
```python
from sklearn.cluster import AffinityPropagation
aff = AffinityPropagation(random_state=5)
y_pred = aff.fit_predict(X)
```



AgglomerativeClustering (Hierarchical Clustering)

- **How it works**:
  - **Bottom-up approach**: Starts with each point as a cluster, then **merges closest pairs** iteratively.
  - Uses **linkage criteria** (ward, complete, average, single).
- **Strengths**:
  - Produces a **dendrogram** for multi-level clustering analysis.
  - Flexible with **different distance metrics**.
- **Weaknesses**:
  - **Not scalable** for large datasets ($O(N^3)$ time complexity).
  - Once merged, clusters cannot be split.
- **Best for**:
  - Medium-sized datasets where hierarchy matters (e.g., taxonomy, document clustering).

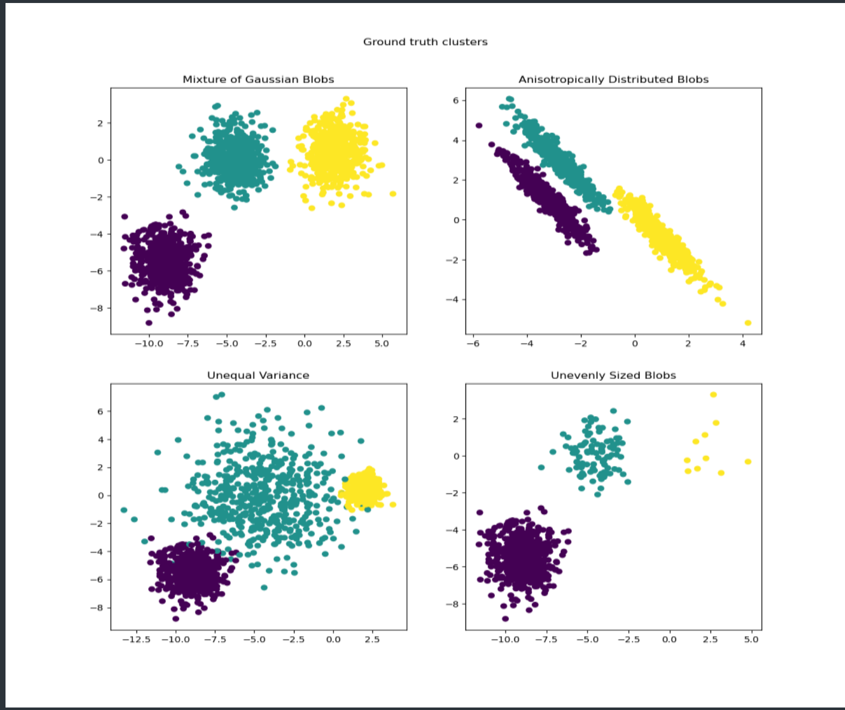
```python
from sklearn.cluster import AgglomerativeClustering
agg = AgglomerativeClustering(n_clusters=3)
y_pred = agg.fit_predict(X)
```



K-Means

- **How it works:**
  - **Partitions data into K clusters** by minimizing **inertia** (within-cluster variance).
  - Uses **iterative centroid updates**.
- **Strengths:**
  - **Fast and scalable** (O(N*K) per iteration).
  - Works well with **spherical clusters**.
- **Weaknesses:**
  - **Sensitive to initialization** (k-means++ helps).
  - Struggles with **non-convex clusters**.
- **Best for:**
  - Large datasets with clear separation (e.g., market segmentation, image compression).

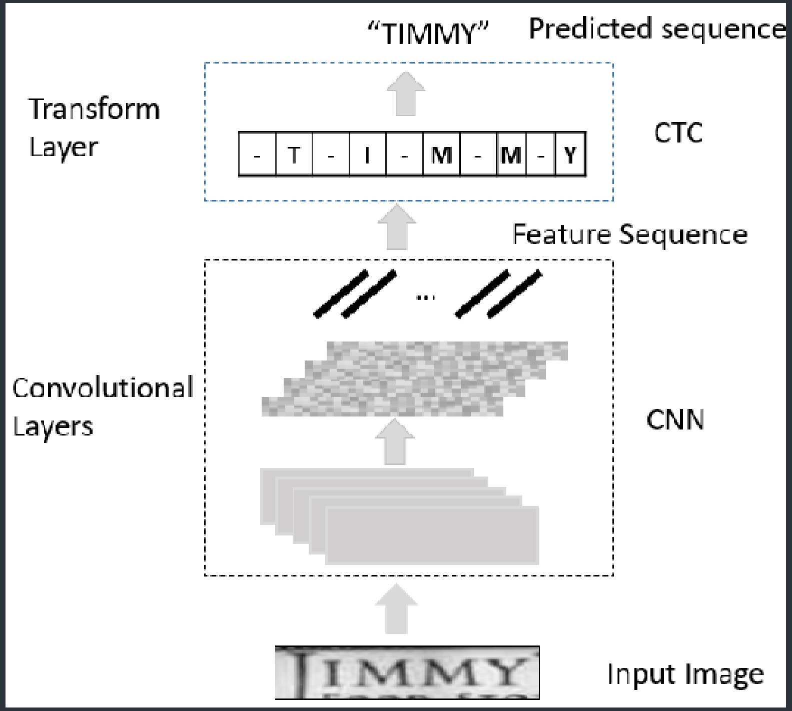
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, random_state=5)
y_pred = kmeans.fit_predict(X)
```



Mean Shift

- **How it works:**
  - **Kernel density estimation** to find cluster modes.
  - **Shifts points** towards high-density regions.
- **Strengths:**
  - **No predefined cluster count** needed.
  - Robust to **outliers**.
- **Weaknesses:**
  - **Computationally expensive** ($O(N^2)$).
  - **Bandwidth selection** is critical.
- **Best for:**
  - Image segmentation, object tracking.

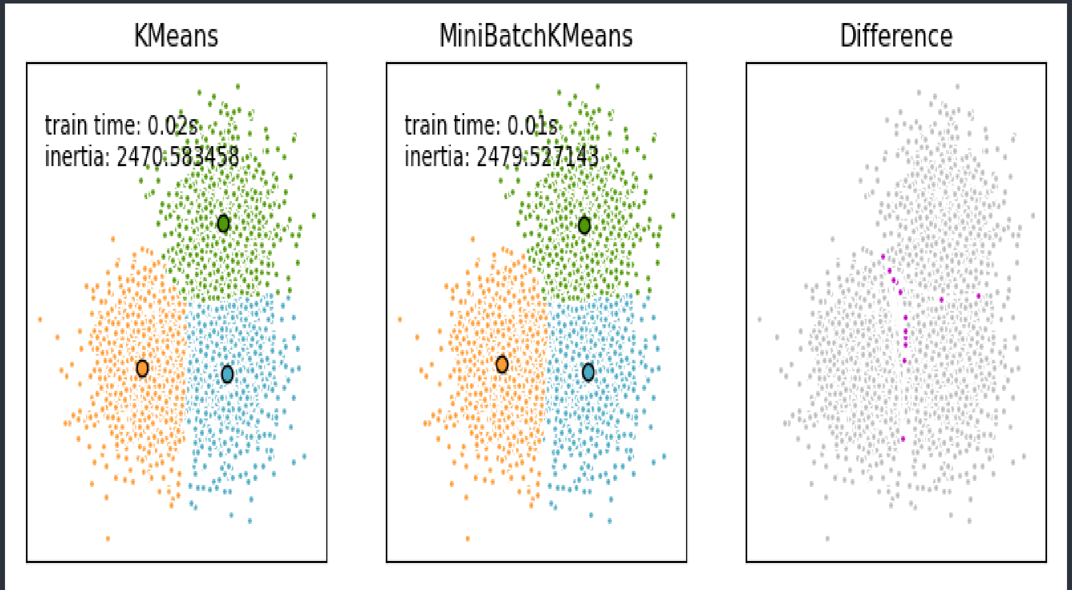
```python
from sklearn.cluster import MeanShift
ms = MeanShift(bandwidth=2)
y_pred = ms.fit_predict(X)
```



MiniBatch K-Means

- **How it works:**
  - **Approximate K-Means** using random mini-batches.
  - **Faster but less accurate** than full K-Means.
- **Strengths:**
  - **Scalable to very large datasets.**
  - Useful for **online learning**.
- **Weaknesses:**
  - **Lower accuracy** due to approximations.
- **Best for:**
  - Big data applications (e.g., real-time clustering, recommendation systems).

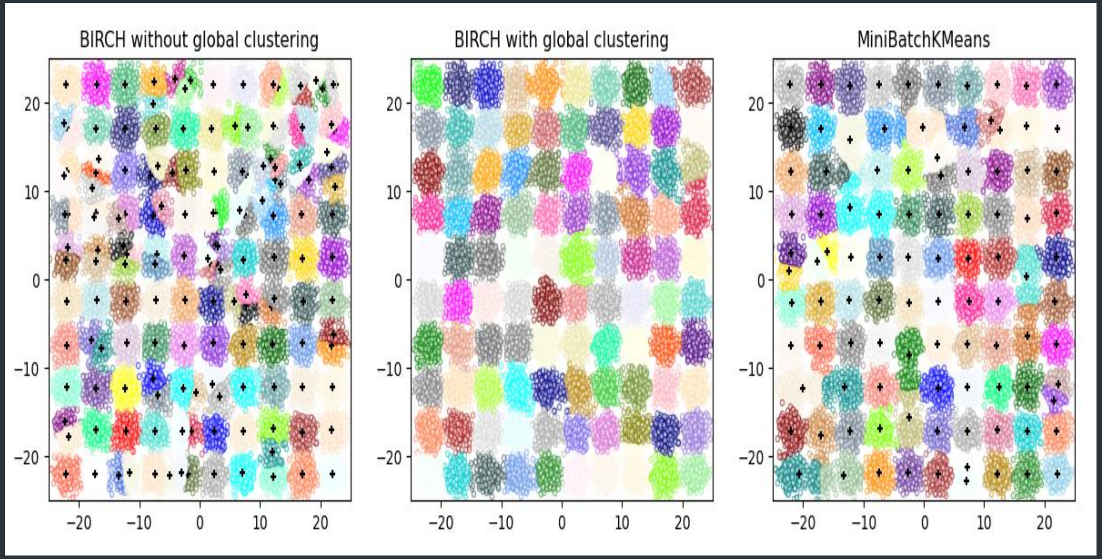
```python
from sklearn.cluster import MiniBatchKMeans
mbk = MiniBatchKMeans(n_clusters=3, random_state=5)
y_pred = mbk.fit_predict(X)
```



BIRCH (Balanced Iterative Reducing & Clustering using Hierarchies)

```
from sklearn.cluster import Birch
birch = Birch(n_clusters=3)
y_pred = birch.fit_predict(X)
```

- **How it works**:
  - Builds a **Clustering Feature (CF) Tree** for incremental clustering.
  - Compresses data into **subclusters** before final clustering.
- **Strengths**:
  - **Memory-efficient** for **large datasets**.
  - Handles **high-dimensional data** better than K-Means.
- **Weaknesses**:
  - **Sensitive to input order** of data.
  - Struggles with **non-spherical clusters**.
- **Best for**:
  - Large-scale datasets (e.g., customer segmentation, anomaly detection).



HDBSCAN (Hierarchical DBSCAN)

- **How it works:**

Extends DBSCAN with hierarchical clustering.

Automatically extracts clusters using stability analysis.

- **Strengths:**

No need for eps parameter (unlike DBSCAN).

Better for varying densities.

- **Weaknesses:**

Slower than DBSCAN.

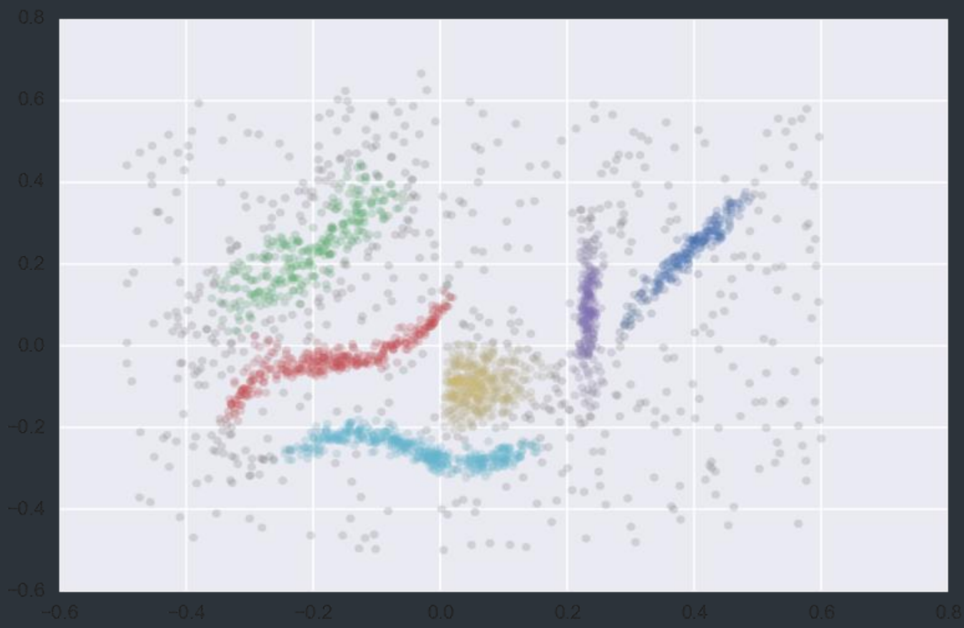
- **Best for:**

Complex datasets with varying densities (e.g., bioinformatics, social network analysis).

(Requires `hdbscan` library: `pip install hdbscan`)

python

```python
import hdbscan
hdb = hdbscan.HDBSCAN(min_cluster_size=5)
y_pred = hdb.fit_predict(X)
```



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

- **How it works:**

Groups points based on density (core, border, noise points).

- No predefined cluster count needed.
- **Strengths:**

Robust to noise and arbitrary cluster shapes.

Works well with spatial data.

- **Weaknesses:**

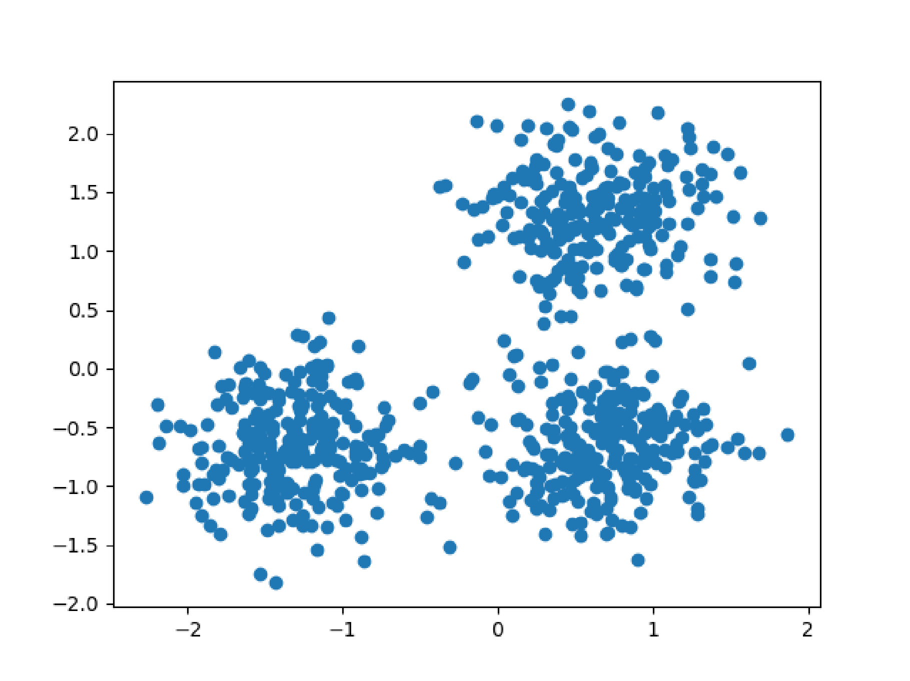
Struggles with varying densities.

Sensitive to eps and min_samples parameters.

- **Best for:**

Anomaly detection, geographic data, and irregularly shaped clusters.Rathiga_ML_Cluster

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
y_pred = dbscan.fit_predict(X)
```



OPTICS (Ordering Points To Identify Clustering Structure)

- **How it works:**

Generalized DBSCAN that creates a reachability plot.

Extracts clusters at multiple density levels.

- **Strengths:**

Handles varying densities better than DBSCAN.

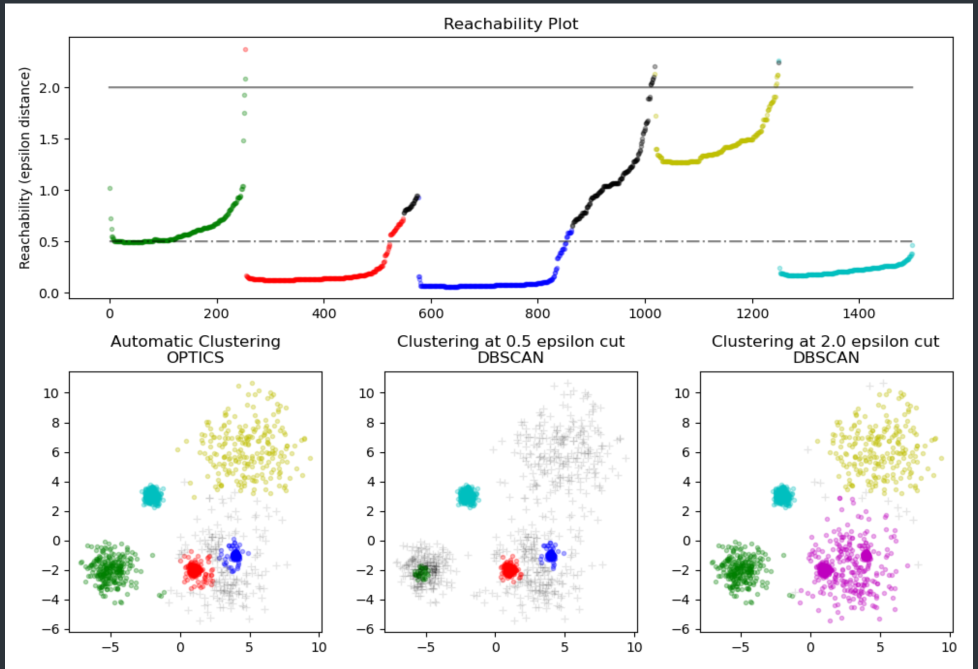
No single eps parameter needed.

- **Weaknesses:**

Slower than DBSCAN.

- **Best for:**
- Datasets with nested clusters (e.g., astronomy, geology).

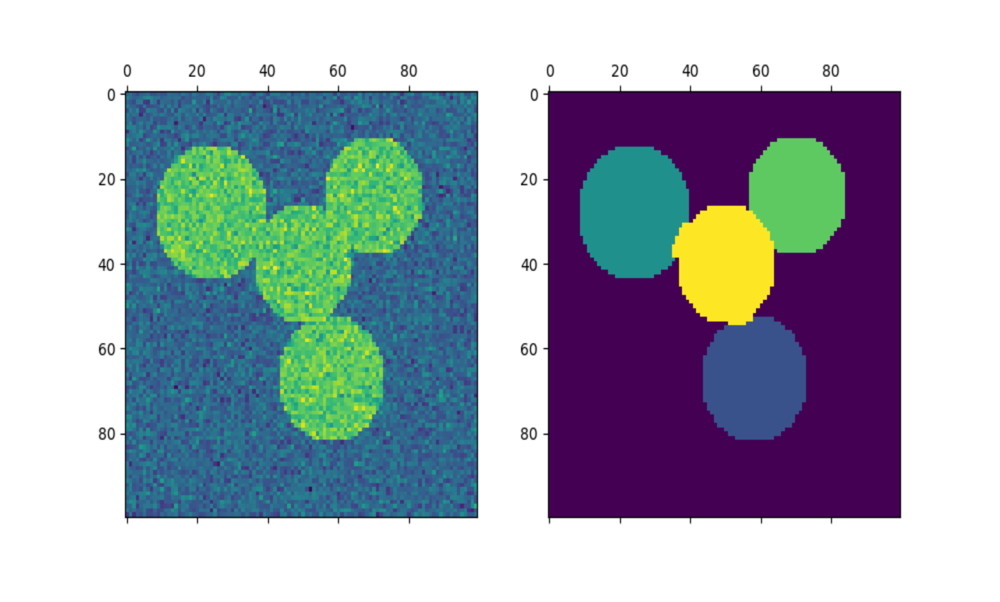
```
from sklearn.cluster import OPTICS
optics = OPTICS(min_samples=5)
y_pred = optics.fit_predict(X)
```



Spectral Clustering

- **How it works:**
  - Uses **graph Laplacian** to project data into lower dimensions.
  - Applies **K-Means on eigenvectors**.
- **Strengths:**
  - **Works with non-convex clusters.**
  - Effective for **graph-based data**.
- **Weaknesses:**
  - **Not scalable** (O(N^3) for eigen decomposition).
- **Best for:**
  - Image segmentation, community detection in networks.

```python
from sklearn.cluster import SpectralClustering
spec = SpectralClustering(n_clusters=3, random_state=5)
y_pred = spec.fit_predict(X)
```



Summary Table

Algorithm	Best For	Strengths	Weaknesses
AffinityPropagation	Small datasets, unknown K	No need for K, finds exemplars	High computational cost
Agglomerative	Hierarchical clustering	Dendrogram, flexible linkage	Slow for large data
BIRCH	Large datasets	Memory-efficient, fast	Sensitive to data order
DBSCAN	Noise, irregular shapes	No predefined K, robust	Struggles with varying density
K-Means	Large, spherical clusters	Fast, scalable	Needs K, sensitive to init
HDBSCAN	Varying densities	Automatic K, robust	Slower than DBSCAN
Mean Shift	Density peaks	No K needed	Computationally heavy
OPTICS	Nested clusters	Multi-density handling	Slower than DBSCAN
Spectral	Non-convex clusters	Works on graphs	Not scalable

Summary Table with Visual Behavior

Algorithm	Type	Diagram Behavior	Best Use Case
K-Means	Partition-based	Spherical clusters	Well-separated data
DBSCAN	Density-based	Arbitrary shapes, noise-resistant	Spatial data, anomalies
Agglomerative	Hierarchical	Dendrogram, nested clusters	Taxonomy, bioinformatics
Mean Shift	Density-based	Smooth blobs, no fixed K	Image segmentation
Affinity Prop.	Exemplar-based	Finds "representative" points	Small datasets
Spectral	Graph-based	Non-convex clusters	Image, network data
OPTICS	Density-hierarchy	Multi-level density clusters	Nested structures
BIRCH	Hierarchical	CF-Tree compression	Large-scale data

When to Use Which?

- **Default choice: K-Means** (if data is spherical and K is known).
- **Irregular shapes: DBSCAN / HDBSCAN.**
- **Hierarchical structure: Agglomerative / BIRCH.**
- **Unknown K: AffinityPropagation / Mean Shift / HDBSCAN.**
- **Very large data: MiniBatch K-Means / BIRCH.**