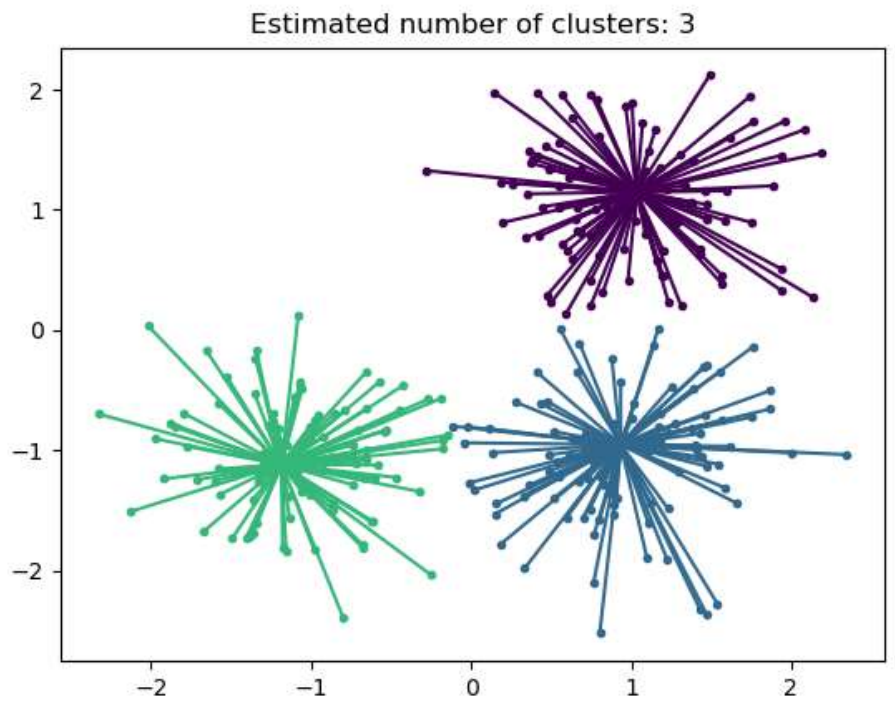
# Affinity Propagation

Affinity Propagation is a clustering algorithm that identifies **exemplars** (representative data points) and groups other points around these exemplars. Unlike methods like k-means, it does not require predefining the number of clusters. Instead, it uses **message passing** between points to determine clusters automatically.



```
from sklearn.cluster import AffinityPropagation
model = AffinityPropagation(damping=0.9)
model.fit(X)
```

## How Affinity Propagation Works

**1.Similarity Matrix**:
The algorithm computes pairwise similarities (e.g., negative Euclidean distance) between points.
Example: Similarity between (-2, -2) and (2, 2) would be low, while similarity between (-2, -2) and (-1, -1) would be high.

The algorithm iteratively updates two types of messages:

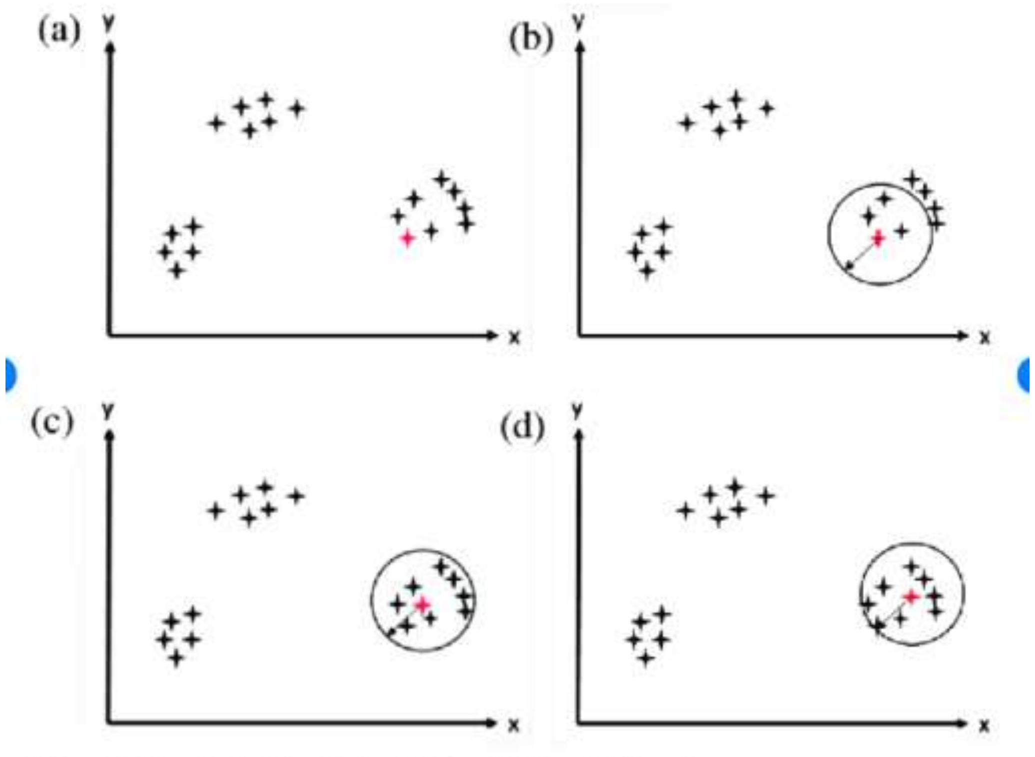
**1.Responsibility r(i,k):**
How well point k serves as an exemplar for point i, compared to other candidates.

**2.Availability a(i,k):**
How "available" point k is to be an exemplar for point i.

**3.Exemplar Selection**:
Points with high "responsibility + availability" scores become exemplars.
Here 3 exemplars emerged from the data.
In your case, 3 exemplars emerged from the data.

# Mean Shift

Mean Shift is a **density-based clustering algorithm** that identifies clusters by iteratively shifting data points toward regions of higher data density. Unlike *k-means*, it does not require predefining the number of clusters. Instead, it relies on a **bandwidth** parameter and automatically determines clusters based on data distribution.



| Mean-shift clustering algorithm. (a) Step 1. (b) Step 2. (c) Step 3. (d) Step 4.

## How Mean Shift Works

**1.Initialization**: Treat each data point as a potential cluster centroid.

**2.Iterative Shifting**:

a) For each point, compute the **mean** of all data points within its bandwidth.

b) Shift the centroid toward this mean.

**1.Convergence**: Repeat until centroids stabilize (minimal movement between iterations).

**2.Cluster Merging**: Merge centroids that are closer than the bandwidth to form final clusters.

3.In your case, 3 exemplars emerged from the data.

```
from sklearn.cluster import MeanShift
model = MeanShift(bandwidth=2)
model.fit(X)
```

## Spectral Clustering

Spectral Clustering is a technique that groups data points by analyzing their **connections** (similarities) rather than their distance. It’s like finding friend groups at a party by seeing who talks to whom, not just where they stand in the room.

### How Spectral Clustering Works

**1. Build an Affinity Matrix (Similarity Graph)**

**What**: Create a matrix where each entry represents how similar two data points are.

**Example**: If pixels in an image have similar colors, their similarity is high.

**2. Compute the Graph Laplacian**

**Degree Matrix (D)**: Tells you how "popular" each point is (sum of its similarities to others).

**Laplacian Matrix (L)**: L=D−W$L=D-W$. Highlights the *structure* of connections.

**3. Eigenvalue Decomposition**

**What**: Find the **eigenvectors** and **eigenvalues** of L.

**Why**: Eigenvectors map the data to a lower-dimensional space where clusters are easier to separate.

**Key Insight**: The first few eigenvectors (corresponding to the smallest eigenvalues) reveal the natural clusters.

**4. Low-Dimensional Embedding**

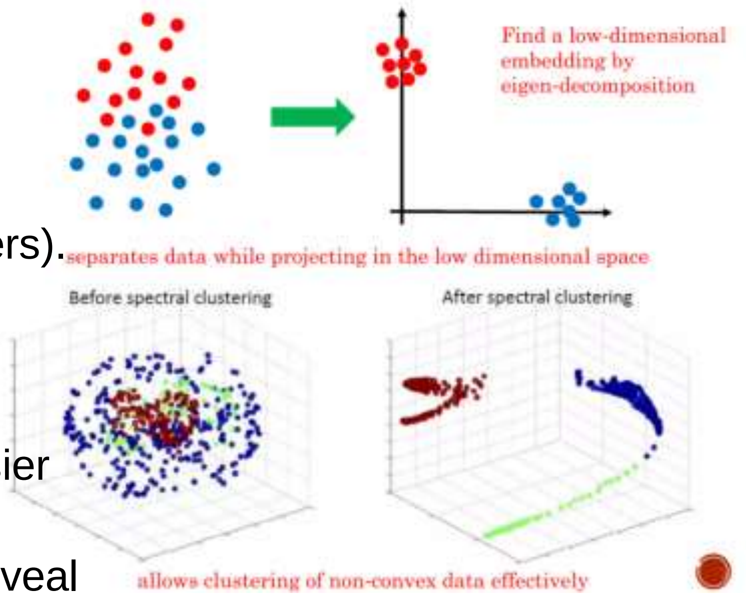
**Embed**: Project data points onto the eigenvectors (like compressing a 3D shape into 2D).

**Example**: Concentric circles in 2D → Straight lines in eigenvector space.

**5. Cluster with k-means**

**What**: Apply k-means on the transformed low-dimensional data.

**Why**: Clusters are now linearly separable in this new space.

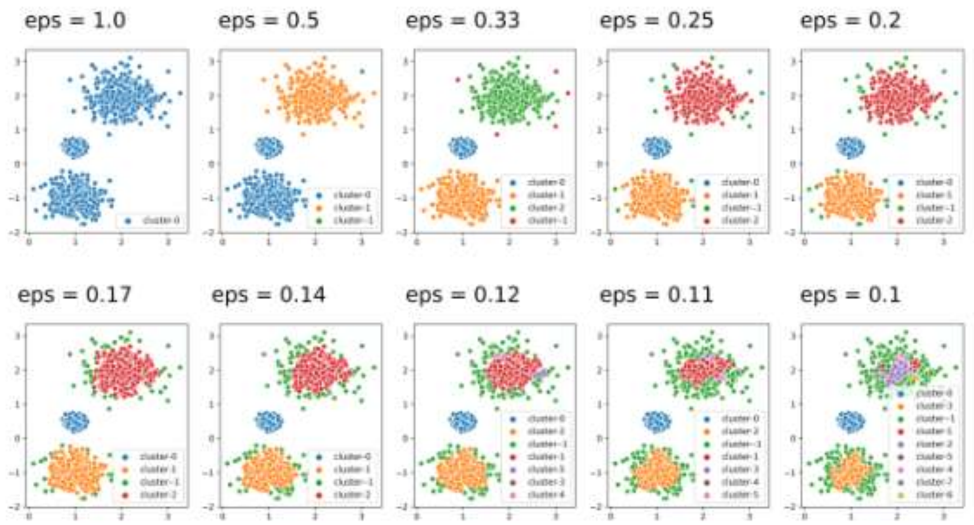

Find a low-dimensional embedding by eigen-decomposition
separates data while projecting in the low dimensional space
Before spectral clustering
After spectral clustering
allows clustering of non-convex data effectively

```
from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters=2,
affinity='nearest_neighbors')
model.fit(X)
```

# DBSCAN

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

DBSCAN is a clustering algorithm that groups data points based on density. Unlike k-means, it can find clusters of **any shape** and detect **outliers.**



## How DBSCAN Works

**Step 1: Find Core Samples**
For each point, check if it has ≥min_samples neighbors within eps distance. Mark these as core points.
For each point, check if it has ≥min_samples neighbors within eps distance, but lacking enough neighbors to be a core. Mark these as border points.
**Noise :** Points that are neither core nor border.

**Step 2: Expand Clusters**
Start with a core point.
Recursively add all reachable core points (within eps distance).
Include border points that are within eps of any core point in the cluster.

**Step 3: Repeat**
Move to the next unvisited core point and repeat Step 2.
Points not assigned to any cluster are labeled as noise.

```
from sklearn.cluster import DBSCAN
model = DBSCAN(eps=0.5, min_samples=5)
model.fit(X)
```

**Parameters:**

**eps (ε):**
Radius of the neighborhood around a point.
Smaller ε → Fewer neighbors per point → More noise.
Larger ε → Larger clusters → Risk of merging distinct clusters.

**min_samples:**
Minimum number of neighbors to classify a point as a core sample.
Higher values → More robust to noise but may miss small clusters.

# OPTICS

(Ordering Points To Identify the Clustering Structure)

**OPTICS** is a density-based clustering algorithm that generalizes DBSCAN. While DBSCAN requires a fixed neighborhood radius (eps) and struggles with clusters of varying densities, OPTICS relaxes this by analyzing clusters across a range of eps values. This makes it more flexible for datasets where clusters have different densities.

## How Optics Works

**Step 1: Build the Reachability Graph**
For each point, compute:
**Core Distance:** The smallest **eps** required to include **min_samples** neighbors.
**Reachability Distance:** The smallest distance to a core point that makes the point density-reachable.
Points are ordered to prioritize dense regions (e.g., tight clusters first).

**Step 2: Generate Reachability Plot**
Visualizes the reachability distances of points in the cluster ordering.
Valleys in the plot indicate clusters (lower reachability = denser regions).

**Step 3: Extract Clusters**
Use the **cluster_optics_dbscan** method to extract clusters for any **eps ≤ max_eps** without re-computing the graph.

## Advantages Over DBSCAN

| Feature | DBSCAN | OPTICS |
|---|---|---|
| **Fixed eps** | ✕ Requires manual tuning | ✓️ Analyzes a range of eps |
| **Varying Densities** | ✕ Struggles with mixed densities | ✓️ Handles via reachability plot |
| **Efficiency** | ✕ Re-runs needed for new eps | ✓️ Extract clusters in linear time post-processing |
| **Noise Handling** | ✓️ Labels outliers | ✓️ Same as DBSCAN |

```
from sklearn.cluster import OPTICS
model = OPTICS(max_eps=3, min_samples=10)
model.fit(X)
```

HDBSCAN eliminates the need for a fixed eps by analyzing clusters at all density scales and selecting the most stable ones.

# HDBSCAN

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

HDBSCAN is an advanced clustering algorithm that addresses a key limitation of DBSCAN: **the inability to handle clusters with varying densities**. HDBSCAN eliminates the need for a fixed eps by analyzing clusters at all density scales and selecting the most stable ones.

## How HDBSCAN Works

**Step 1: Build a Hierarchy of Clusters**

**a. Mutual Reachability Distance:**

HDBSCAN computes a distance metric that accounts for both:

- The actual distance between points.
- The distance to each point’s nearest neighbors.

This ensures sparse regions are "stretched" and dense regions are "compressed" in the hierarchy.

**b. Hierarchical Clustering:**

Creates a tree (dendrogram) where clusters form and merge as the density threshold decreases (equivalent to increasing eps in DBSCAN).

**Step 2: Condense the Hierarchy**

**Stability-Based Pruning:**

The hierarchy is simplified by removing short-lived clusters (i.e., clusters that only exist over a narrow range of densities).

Persistent Clusters: Clusters that survive across a wide range of densities are retained.

**Step 3: Extract Final Clusters**

**a. Optimal Clusters:**

Selects clusters that maximize stability (persistence) across density levels.

**b. Noise Detection:**

Points not belonging to any stable cluster are labeled as noise.

HDBSCAN eliminates the need for a fixed eps by analyzing clusters at all density scales and selecting the most stable ones.

# HDBSCAN

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

**Advantages Over DBSCAN and OPTICS**

| Feature | DBSCAN | OPTICS | HDBSCAN |
|---|---|---|---|
| **Varying Densities** | ✕ Fails | ✓ (via reachability plot) | ✓ Automatically handles |
| **Parameter Tuning** | Requires manual eps | Requires manual interpretation | Fully automated |
| **Noise Handling** | ✓ | ✓ | ✓ |
| **Cluster Shape** | Arbitrary, but density-limited | Arbitrary | Arbitrary + multi-density |

```
import hdbscan
model = hdbscan.HDBSCAN(min_cluster_size=5)
model.fit(X)
```

HDBSCAN eliminates the need for a fixed eps by analyzing clusters at all density scales and selecting the most stable ones.

# BIRCH

## (Balanced Iterative Reducing and Clustering using Hierarchies)

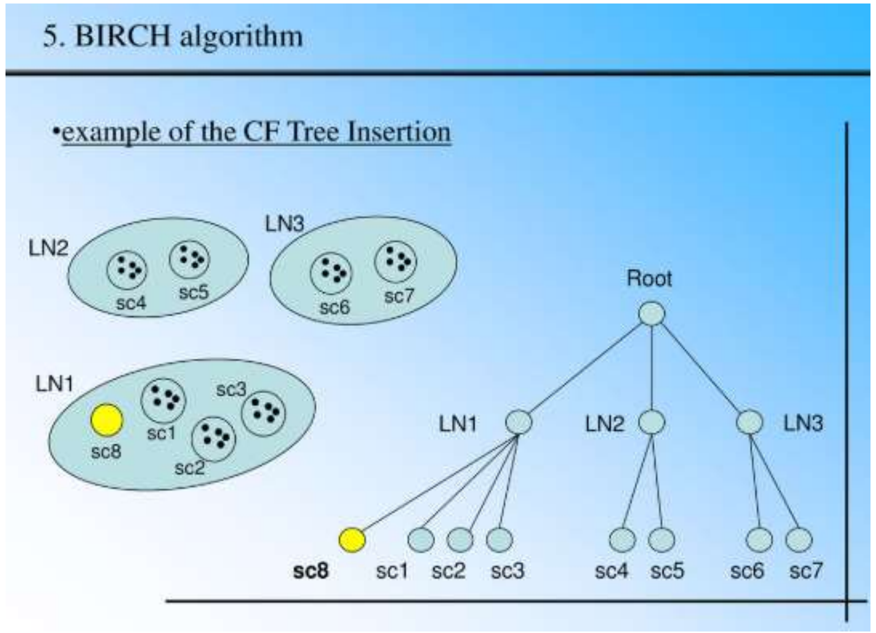
The BIRCH algorithm is designed for efficiently clustering large datasets by summarizing data into a compact hierarchical structure.

```
from sklearn.cluster import Birch
model = Birch(threshold=0.5, n_clusters=3)
model.fit(X)
```

## How BIRCH Works

**Step 1: Build the CF (Clustering Features) Tree**

- **Insert Data Incrementally:**
  For each new data point, traverse the CF Tree from root to leaves.
  At each node, find the closest CF Subcluster (using centroid distance).
  If the distance is within threshold, update the CF Subcluster's statistics (count, linear sum, etc.).
  If no suitable subcluster exists, create a new CF Subcluster (if branching factor allows).
- **Tree Balancing:**
  If a node exceeds the branching factor, split it into smaller nodes to maintain efficiency.



**Step 2: Global Clustering (Optional)**

- **If n_clusters is specified:**
  Extract CF Subclusters from the leaf nodes.
  Apply a global clustering algorithm (e.g., Agglomerative Clustering) to group subclusters into final clusters.

- **If n_clusters=None,** the CF Subclusters themselves are the final clusters.

HDBSCAN eliminates the need for a fixed eps by **analyzing clusters at all density scales** and selecting the most stable ones.

## Convex Clusters

Clusters where a straight line between any two points lies entirely inside the cluster.

**Examples**:

1. **Customer Segmentation by Income and Spending**
   - **Shape**: Spherical or elliptical clusters.
   - **Why Convex**: Customers with similar incomes/spending habits group tightly around a central point.
   - **Algorithm**: K-means.
2. **Image Segmentation for Compact Objects**
   - **Shape**: Circular regions (e.g., apples in a bowl).
   - **Why Convex**: Objects like fruits or coins have uniform, compact shapes.
   - **Algorithm**: Mean Shift.
3. **Sensor Networks in Circular Zones**
   - **Shape**: Sensors placed in circular regions (e.g., weather stations in a farm).
   - **Why Convex**: Sensors are grouped around a central hub.
   - **Algorithm**: Gaussian Mixture Models.
4. **Species Distribution Around a Resource**
   - **Shape**: Animals clustered around a water source (circular/spherical).
   - **Why Convex**: Resources like lakes create centralized groupings.
   - **Algorithm**: Hierarchical Clustering.
5. **Product Categories in E-commerce**
   - **Shape**: Books, electronics, or clothing grouped in distinct spheres.
   - **Why Convex**: Categories are mutually exclusive and centralized.
   - **Algorithm**: K-means.

HDBSCAN eliminates the need for a fixed eps by **analyzing clusters at all density scales** and selecting the most stable ones.

# Non Convex Clusters

Clusters with irregular shapes where a straight line between two points may exit the cluster.

**Examples**:

1. **Disease Spread Along a River**
   - **Shape**: Winding or branching clusters following the river's path.
   - **Why Non-Convex**: Infections spread linearly along water sources.
   - **Algorithm**: DBSCAN.

2. **Social Network Communities**
   - **Shape**: Interconnected but non-spherical groups (e.g., overlapping friend circles).
   - **Why Non-Convex**: Relationships form complex, intertwined networks.
   - **Algorithm**: Spectral Clustering.

3. **Urban vs. Rural Population Density**
   - **Shape**: Irregular boundaries between cities and countryside.
   - **Why Non-Convex**: Population density drops unevenly at city edges.

4. **Fraud Detection Patterns**
   - **Shape**: Sparse but connected fraudulent transactions.
   - **Why Non-Convex**: Fraudsters use multiple disjoint accounts.
   - **Algorithm**: OPTICS.

5. **Astronomical Structures (Spiral Galaxies)**
   - **Shape**: Spiral arms or filamentary structures.
   - **Why Non-Convex**: Gravitational forces create elongated patterns.
   - **Algorithm**: Spectral Clustering.