> **Subject**: ML slides pack - clustering with and without kmeans by Claudio Sartori
>
> **Course:** Artificial Intelligence - LM
>
> **Department**: DISI (Department of Computer Science and Engineering) - University of Bologna, Italy
>
> **Author**: Roberto Zanolli
>
> **TIPS**
>
> - w.r.t means "with respect to"

Clustering

What is clustering

Given: A set of N objects x_i, each described by D values x_{id}
Task: Find a natural partitioning into K clusters and possibly identify noise objects

Result: A clustering scheme - a function mapping each data object to $\{1..K\}$ (or to noise)

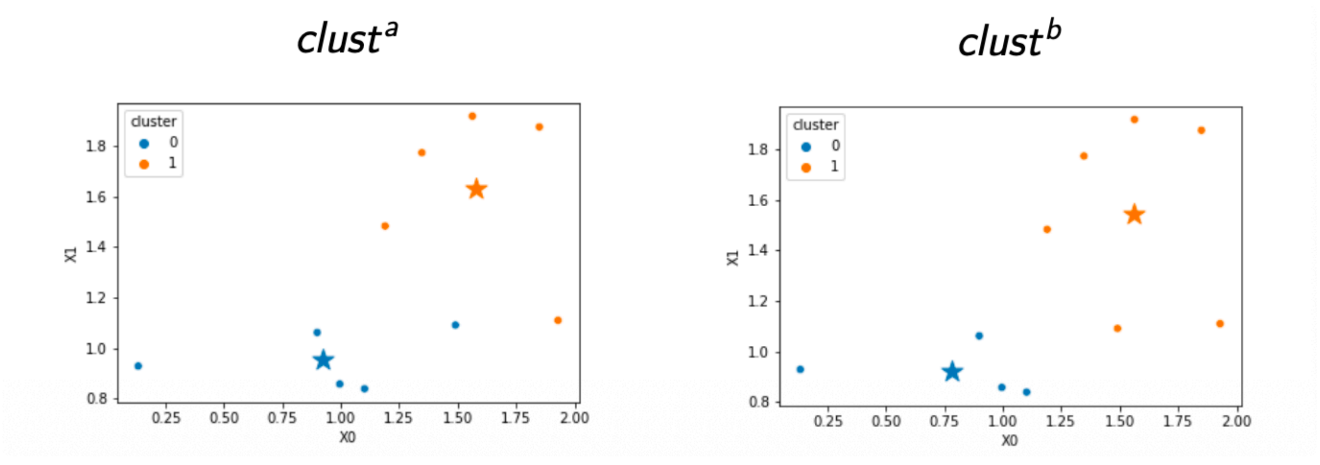
Desired cluster property:

- Objects in same cluster are similar
- Look for clustering scheme that **maximizes intra-cluster similarity**

Formal definition for clustering function

> Find a function $clust()$ from X to $\{1..K\}$ such that:
>
> 1. $\forall x_1, x_2 \in X$, $clust(x_1) = clust(x_2)$ when x_1 and x_2 are similar
> 2. $\forall x_1, x_2 \in X$, $clust(x_1) \neq clust(x_2)$ when x_1 and x_2 are not similar

How do we choose a measure to determine clusters



Centroid

Definition: A point whose coordinates are the average coordinates of all points in the cluster

Calculation: For each cluster k and dimension d, the d-th coordinate of the centroid is:

> $$centroid_d^k = \frac{1}{|x_i : clust(x_i) = k|} \sum_{x_i : clust(x_i) = k} x_{id}$$

Taxonomy of clustering method

The ones with means that have been seen during lab*

Partitioning:

- **K-means (MacQueen 67)** *
- Expectation maximization (Lauritzen 95)
- CLARANS (Ng and Han 94)

Hierarchic:

- **Agglomerative/divisive*** *
- BIRCH (Zhang et al 96)
- CURE (Guha et al 98)
- Based on linkage

Based on density:

- **DBSCAN (Ester et al 96)** *
- DENCLUE (Hinnenburg and Keim 98)

Statistics:

- IBM-IM demographic clustering
- COBWEB (Fisher 87)
- Autoclass (Cheeseman 96)

K-MEANS CLUSTERING

The challenge: Given data that appears to form distinct groups (like a five-component gaussian mixture), how do we automatically discover these groups in high-dimensional space?

Core idea: View clustering as a lossy compression problem. If we must transmit point coordinates using very few bits, we need to group similar points and represent them by a common "code" (centroid).

Loss function: We measure quality by the sum of squared errors (SSE) between actual points and their representative centroids.

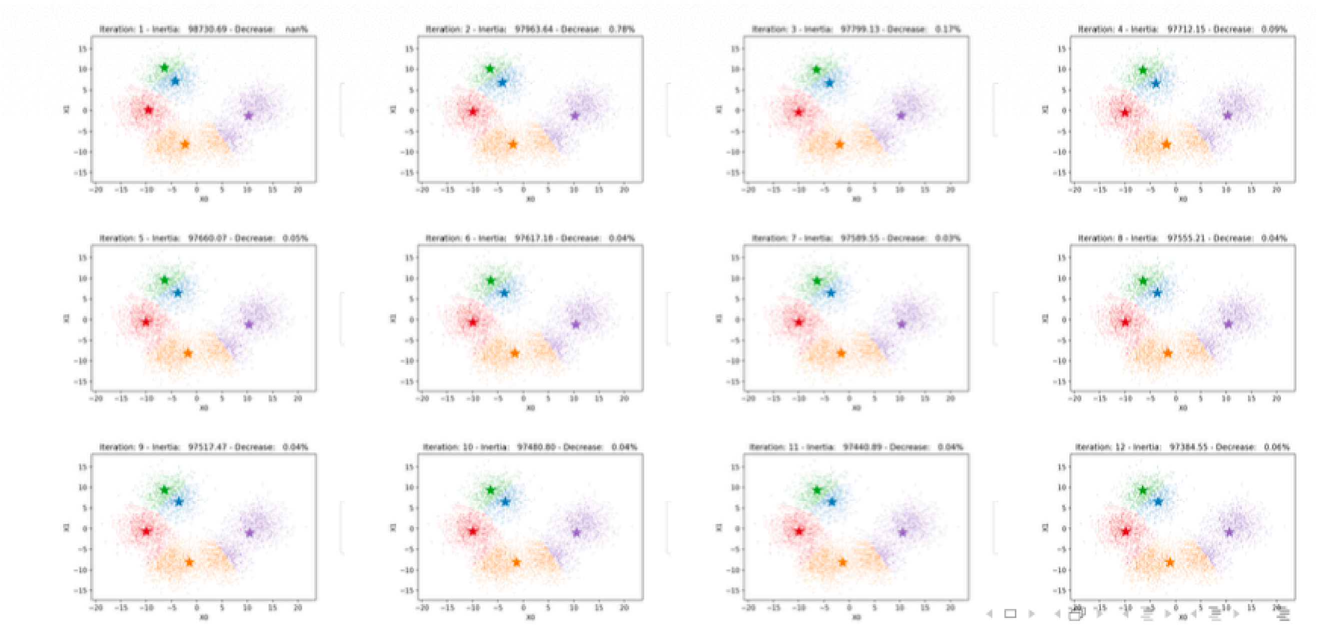
> **Algorithm intuition:** Instead of a fixed grid, let the data itself determine optimal group representatives:
> 1. Start with random guesses for group centers
> 2. Assign each point to its nearest center

> 3. Update centers to be the centroid (mean) of their assigned points
> 4. Repeat until assignments stabilize

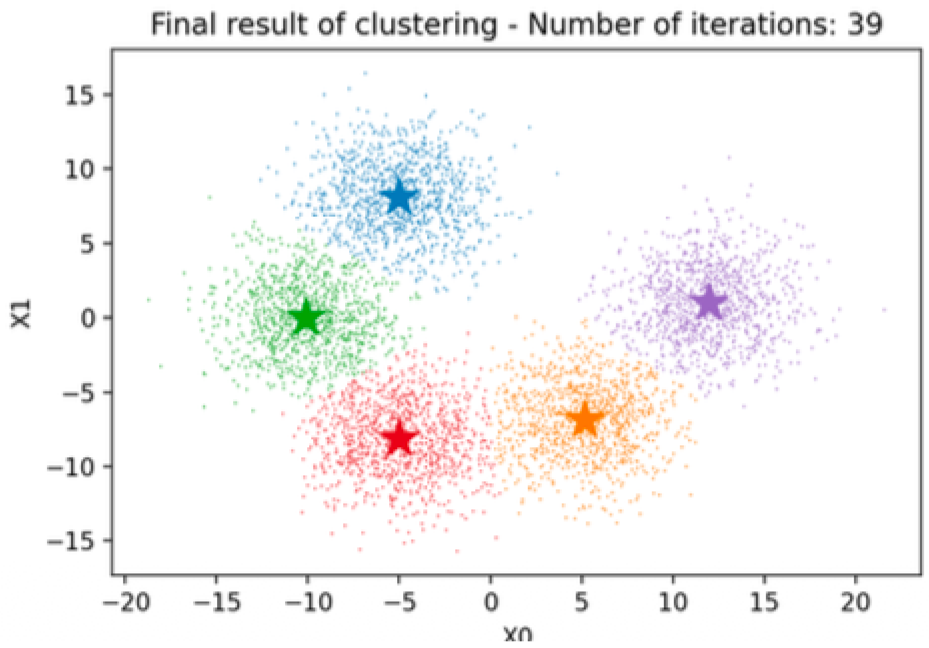
Key questions that remain:

1. What are we trying to optimize?
2. Is termination guaranteed?
3. Are we sure that the best clustering scheme is found? (And what defines "best"?)
4. How should we initialize the centers?
5. How can we determine the optimal number of clusters?

Applying the algorithm seen above we observe the example:



From numerous iterations we get here:



Question 1: What are we trying to optimize?

Answer: Distortion (Inertia)

Definition: Given:

- Dataset $\{x_i, i = 1 \ldots N\}$
- Encoding function $Encode : \mathbb{R}^D \rightarrow \{1..K\}$
- Decoding function $Decode : \{1..K\} \rightarrow \mathbb{R}^D$

$$Distortion = \sum_{i=1}^{N} (x_i - Decode(Encode(x_i)))^2$$

Let $Decode(k) = c_k$, then:

$$Distortion = \sum_{i=1}^{N} (x_i - c_{Encode(x_i)})^2$$

Minimal distortion

Question: What properties do $c_1, \ldots, c_K$ need for minimal distortion?

Property 1: x_i must be ***encoded with the nearest center***

- Otherwise distortion could be reduced by choosing the nearest center $c_{Encode(x_i)} = \arg\min_{c_j \in \{c_1,\ldots,c_K\}} (x_i - c_j)^2$

Property 2: Partial derivative of distortion w.r.t. each center must be zero

- Indicates maximum or minimum of function

Mathematical derivation

$$Distortion = \sum_{i=1}^{N} (x_i - c_{Encode(x_i)})^2 = \sum_{j=1}^{K} \sum_{i \in OwnedBy(c_j)} (x_i - c_j)^2$$

$$\frac{\partial Distortion}{\partial c_j} = -2 \sum_{i \in OwnedBy(c_j)} (x_i - c_j) = 0 \text{ at minimum}$$

Solution:

$$c_j = \frac{1}{|OwnedBy(c_j)|} \sum_{i \in OwnedBy(c_j)} x_i$$

Question 2: Is termination guaranteed?

Answer: YES
Proof: Finite number of states ensures termination

- Finite ways to partition N objects into K groups
- Finite configurations where centers are centroids of their points
- Each iteration reduces distortion → new state never visited before
- Eventually no new states reachable → algorithm stops

Question 3: Are we sure that the best clustering scheme is found? (And what defines "best"?)

Answer: model the problem like a local vs global minimum problem

Problem: Ending state not necessarily global optimum

- Multiple local minima possible
- Final clustering depends on initialization

Question 4: How should we initialize the centers?

Answer: we try several times and pick the best
Strategies:

1. Random first point, subsequent points far from previous ones
2. Run algorithm multiple times with different random starts
3. Choose best result based on distortion

Question 5: How can we determine the optimal number of clusters?

Answer: we try several times and pick the best
Challenge: No easy automatic method

Approach:

- Try various K values
- Use quantitative evaluation metrics
- Balance intra-cluster compactness vs inter-cluster separation

Summary of application

- Try with different starting points with same K and choose the best starting point to ensure you get to global minimum (at least we try)
- After you choose

Proximity function

Default: Euclidean distance (works well for vector spaces)
Alternatives: Various distance metrics for specific data types

Sum of Squared Errors (SSE)

Formula:

$$SSE = \sum_{j=1}^{K} \sum_{i \in OwnedBy(c_j)} (x_i - c_j)^2$$

Properties:

- $SSE_j = 0$ only if all points coincide with centroid
- SSE decreases with increasing K, reaches 0 when $K = N$
- Cannot minimize SSE to choose K (always prefers more clusters)

Outliers

Impact: High distance from centroid → high contribution to SSE

- Bad influence on clustering results
- Sometimes beneficial to remove (domain-dependent)

Common uses of K-means

1. **Data exploration:** Quick initial clustering
2. **1D discretization:** Create non-uniform bins
3. **Vector quantization:** Signal processing, compression
4. **Color quantization:** GIF compression, color palette selection

Time complexity

The time complexity is described as it follows.

> **Time:** $O(TKND)$
>
> *Where*:
>
> - T = number of iterations
> - K = number of clusters
> - N = number of data points
> - D = number of dimensions

Pros and cons

PROS:

- Fairly efficient (nearly linear in N)
- $T, K, D \ll N$ in practice

CONS:

- Requires space where centroid computable (works with Euclidean, some other distances)
- Cannot handle nominal data
- Requires K parameter (though can be searched)
- Sensitive to outliers
- No noise handling
- Poor with non-convex clusters

Evaluation of clustering schemes

Clustering evaluation focuses solely on the results, not the technique used. Since clustering is an unsupervised method with little a priori information (like class labels), evaluation becomes critical. We need score functions to measure various properties of clusters and the overall clustering scheme. The terms "score" and "index" are synonymous in this context.

When supervised data is available, it can aid evaluation. In 2D, clusters can be visually inspected, while in higher-dimensional spaces, 2D projections help but formal methods are preferred.

Key evaluation challenges

- Distinguishing genuine patterns from random apparent regularities
- Determining the optimal number of clusters
- Conducting non-supervised evaluation
- Performing supervised evaluation when possible
- Comparing clustering schemes relatively

Proximity measures

Similarity/Proximity: A two-variable function measuring how similar two objects are based on their property values.

Dissimilarity: A two-variable function measuring how different two objects are based on their property values (e.g., Euclidean distance).

Cohesion and separation metrics

Global separation (SSB): Sum of Squares Between clusters measures separation. Let c be the global centroid:

$$SSB = \sum_{i=1}^{K} N_i \cdot Dist(c_i, c)^2$$

Total Sum of Squares relationship: The total variance can be decomposed into within-cluster and between-cluster components:

$$TSS = SSE + SSB$$

(The total sum of squares is a global property of the dataset, independent from the clustering scheme).
This shows that minimizing within-cluster variance (SSE) is equivalent to maximizing between-cluster variance (SSB) for a fixed dataset.

Mathematical derivation

The total sum of squares expands as:

$$\begin{aligned} TSS &= \sum_{i=1}^{K} \sum_{x \in k_i} (x - c)^2 \\ &= \sum_{i=1}^{K} \sum_{x \in k_i} [(x - c_i) - (c - c_i)]^2 \\ &= \sum_{i=1}^{K} \sum_{x \in k_i} (x - c_i)^2 - 2 \sum_{i=1}^{K} \sum_{x \in k_i} (x - c_i)(c - c_i) + \sum_{i=1}^{K} \sum_{x \in k_i} (c - c_i)^2 \\ &= SSE + SSB \end{aligned}$$

The cross-term vanishes because $\sum_{x \in k_i} (x - c_i) = 0$ by definition of the cluster centroid c_i.

Cluster-specific evaluation

Individual clusters can be evaluated separately:

- Poorly performing clusters may benefit from splitting
- Weakly separated clusters might be merged
- Border objects can negatively impact cohesion or separation

Silhouette score

The silhouette score provides a standardized **measure of clustering quality** with values in $[-1, 1]$. It increases with better separation and decreases with poor cohesion (high sparsity).

The within-cluster sum of squares decomposes as:

$$SSE = \sum_{j=1}^{K} \sum_{i \in \text{OwnedBy}(c_j)} (x_i - c_j)^2 = \sum_{j=1}^{K} SSE_j$$

Individual object contribution

For each object x_i, we compute:

Cohesion measure (a_i): Average distance to other objects in the same cluster:

$$a_i = \text{average}_{j, y(x_j)=y(x_i)} \text{dist}(x_i, x_j)$$

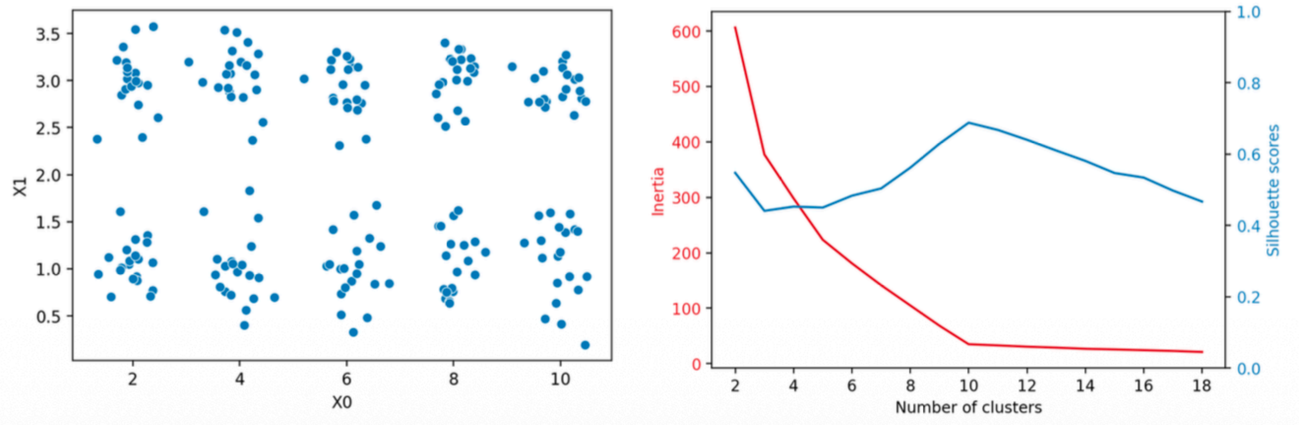
Separation measure (b_i): Minimum average distance to objects in other clusters:

$$b_i = \min_{k \in Y, k \neq y(x_i)} (\text{average}_{j, y(x_j)=k} \text{dist}(x_i, x_j))$$

Silhouette calculation

> The silhouette score for object x_i is:
>
> $$s_i = \frac{b_i - a_i}{\max(a_i, b_i)} \in [-1, 1]$$
>
> Global scores are obtained by averaging over clusters or the entire dataset.

Interpretation: A negative score indicates an object is closer to other clusters than to its own, suggesting poor cluster assignment.



Looking for the best number of clusters

Some clustering algorithms, such as K-means, require specifying the number of clusters K as a parameter beforehand. Since evaluation measures like SSE and Silhouette score are influenced by the choice of K, they can be used to optimize this parameter.

Behavior of evaluation metrics

SSE (within-cluster variance) exhibits predictable behavior:

- Decreases monotonically as K increases
- Equals TSS (total variance) when $K = 1$ (all points in one cluster)
- Approaches zero when $K = N$ (each point is its own cluster)

Silhouette score provides a more nuanced evaluation but is computationally expensive to compute.

Finding the optimal number of clusters

Elbow method

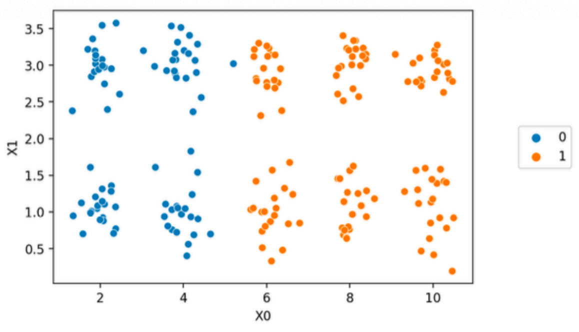
- Plot inertia (SSE) against different values of K
- Look for points where the slope decreases significantly
- These inflection points often represent plausible values for K
- The method identifies where adding more clusters provides diminishing returns

Silhouette score method

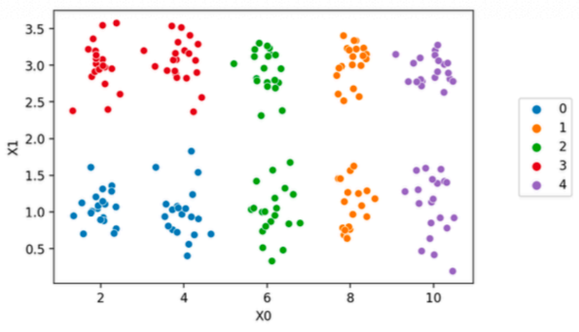
- Calculate silhouette score for different values of K
- The score typically reaches a maximum at one specific K value
- This maximum point indicates the optimal number of clusters
- Provides a clear, point-like optimum for cluster selection

Example of a dataset

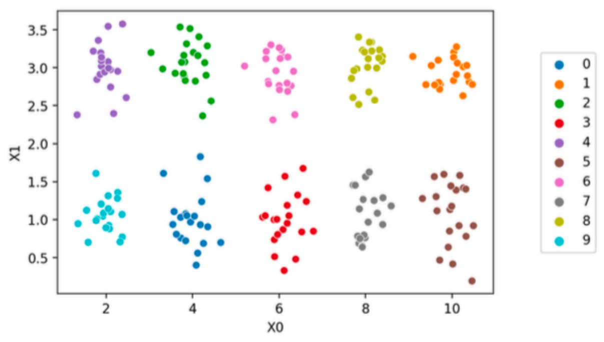
Using this example dataset we can observe the clustering on different K values.
We obtain the best separation on $K = 10$

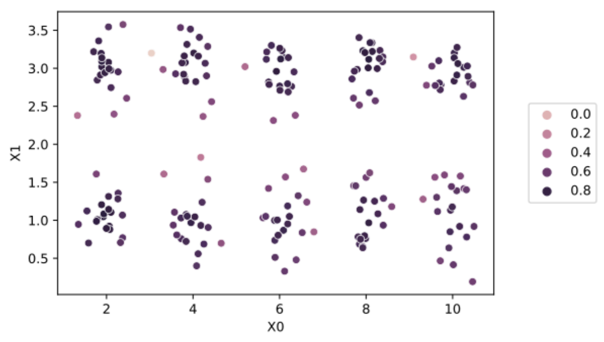


$K = 2$



$K = 5$





Supervised evaluation measures

Gold standard comparison

When a reference partition (gold standard) is available with labeling scheme $y_g(\cdot)$, similar to supervised data for classifier training, we can compare it with our clustering scheme $y_k(\cdot)$. Note that:

- The number of distinct labels V_g and V_k may differ
- Even with identical groupings, label permutations may be needed for comparison

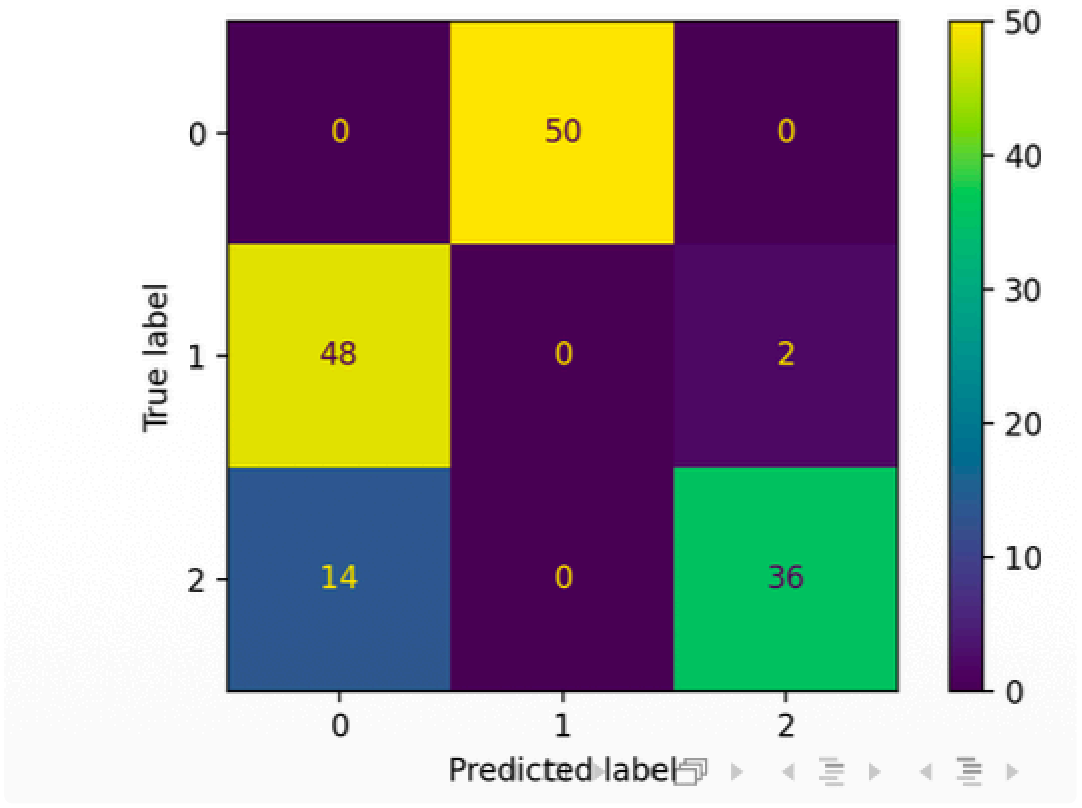
Purpose of gold standard comparison

- Validate clustering techniques for application to new, unlabeled data
- Similar to classifier testing, but focused on grouping rather than labeling
- Ensures the clustering method produces meaningful groupings

Classification-oriented measures

Using confusion matrix analysis:

```
X, y = load_iris(return_X_y=True)
estimator = KMeans(n_clusters=3, random_state=363)
y_km = estimator.fit_predict(X)
disp = ConfusionMatrixDisplay(confusion_matrix(y,y_km))
disp.plot()
```



Measures how gold standard classes distribute among clusters using:

- Confusion matrix
- Precision, recall, F-measure
- Best label permutation matching

Similarity-oriented measures - I

Compare pairs of objects based on their grouping in both schemes:

- **SGSK**: Same group in both y_g and y_k
- **SGDK**: Same group in y_g, different in y_k
- **DGSK**: Different groups in y_g, same in y_k
- **DGDK**: Different groups in both schemes

Similarity-oriented measures - II

Results given by `pair_confusion_matrix(y_g,y_k)`

	SK	DK
SG	13512	1488
DG	1200	6150

Rand Score:

$$\frac{SGSK + DGDK}{SGSK + DGDK + SGDK + DGSK} = 0.88$$

Adjusted Rand Score: Excludes matches expected by chance = 0.73

Jaccard Coefficient for label c:

$$\frac{SG_cSK_c}{SG_cSK_c + SG_cDK_c + DG_cSK_c} = (1, 0.75, 0.69)$$

Requires remapping of $y_g(\cdot)$ to obtain best match.

Hierarchical clustering

Hierarchical clustering generates a nested structure of clusters through two main approaches:

Agglomerative (bottom-up)

- Each data point starts as its own cluster
- At each step, the two least separated clusters are merged
- Requires a measure of separation between clusters

Divisive (top-down)

- The entire dataset starts as one cluster
- At each step, the cluster with lowest cohesion is split
- Requires measures of cohesion and a splitting procedure

Hierarchical clustering output

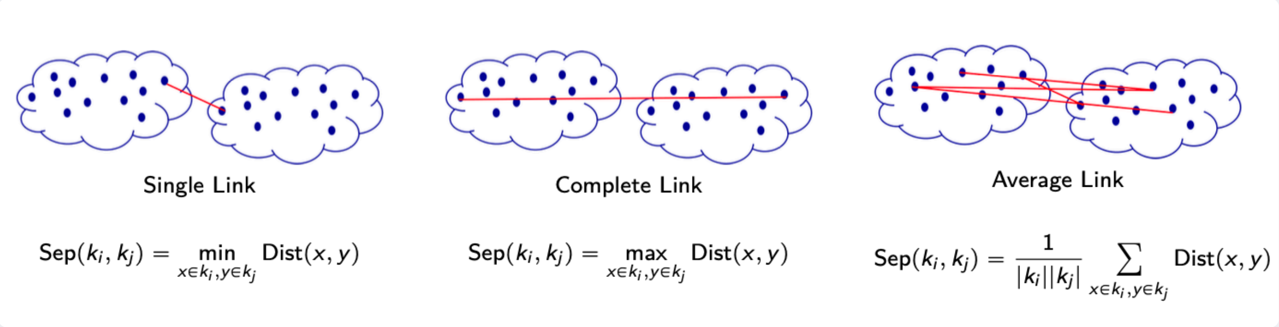
The results are typically visualized as:

- **Dendrogram** - shows merging/splitting hierarchy
- **Nested cluster diagram** - represents the same structure

Both representations work for agglomerative and divisive methods, though agglomerative approaches are more commonly used.

Separation measures between clusters

Separation between clusters k_i and k_j can be calculated using graph-based approaches:


Single Link
Complete Link
Average Link
$\text{Sep}(k_i, k_j) = \min_{x \in k_i, y \in k_j} \text{Dist}(x, y)$
$\text{Sep}(k_i, k_j) = \max_{x \in k_i, y \in k_j} \text{Dist}(x, y)$
$\text{Sep}(k_i, k_j) = \frac{1}{|k_i||k_j|} \sum_{x \in k_i, y \in k_j} \text{Dist}(x, y)$

Single Link (minimum distance):

$$Sep(k_i, k_j) = \min_{x \in k_i, y \in k_j} Dist(x, y)$$

Complete Link (maximum distance):

$$Sep(k_i, k_j) = \max_{x \in k_i, y \in k_j} Dist(x, y)$$

Average Link (mean distance):

$$Sep(k_i, k_j) = \frac{1}{|k_i||k_j|} \sum_{x \in k_i, y \in k_j} Dist(x, y)$$

These measures define cluster distances based on pairwise distances between objects in different clusters.

Ward's method for cluster separation

Ward's method measures separation between clusters S_1 and S_2 by calculating the increase in within-cluster variance if they were merged:

$$d(S_1, S_2) = SSE(S_1 \cup S_2) - (SSE(S_1) + SSE(S_2))$$

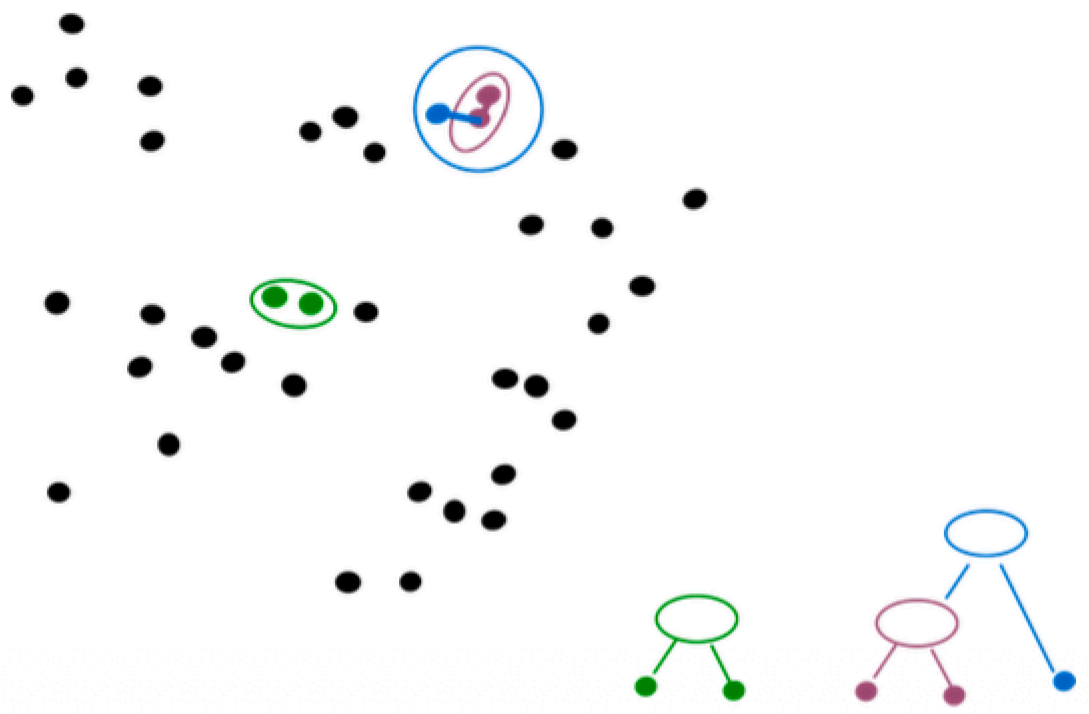
A smaller separation value indicates that merging the clusters would result in a smaller increase in total SSE, making them better candidates for combination.

Single linkage hierarchical clustering

Algorithm steps

1. **Initialization**: Each object starts as its own cluster
2. **Find least separated pair**: Identify the two clusters with minimum separation
3. **Merge clusters**: Combine the closest pair into a single cluster
4. **Repeat**: Continue finding and merging the closest clusters until only one remains

This process builds a **dendrogram** showing the hierarchical relationship between objects.



Single linkage algorithm details

- Initialize clusters: one per object

- Compute $N \times N$ distance matrix (symmetric, zero diagonal)
- While clusters > 1:
  - Find clusters k_r and k_s with minimum separation
  - Merge them into new cluster
  - Update distance matrix: remove rows/columns for k_r and k_s, add new row/column
  - Distance to merged cluster: $Dist(k_k, k_{(r \cup s)}) = \min(Dist(k_k, k_r), Dist(k_k, k_s))$ for all $k \in [1, K]$

Complexity analysis

- **Space**: $O(N^2)$ for storing distance matrix
- **Time**: $O(N^3)$ worst case
  - $N - 1$ iterations to reach single cluster
  - Each iteration: $O((N-i)^2)$ for search + $O(N-i)$ for recomputation
- **Optimization**: Can be reduced to $O(N^2 \log(N))$ with indexing structures

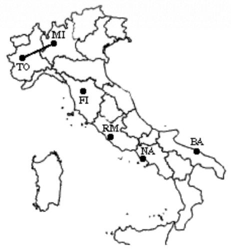
Example of single linkage algorithm being applied:

Finding recursively the shortest link we build the dendrogran has shown in the last pic.

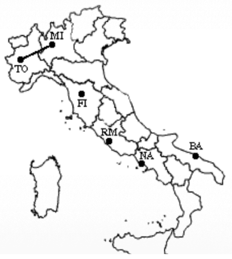
	BA	FI	MI	NA	RM	TO
BA	0	662	877	255	412	996
FI	662	0	295	468	268	400
MI	877	295	0	754	564	138
NA	255	468	754	0	219	869
RM	412	268	564	219	0	669
TO	996	400	138	869	669	0

MI
TO
FI
RM
NA
BA

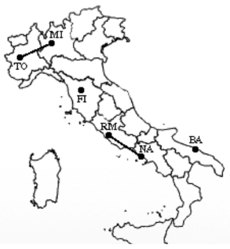
	BA	FI	MI	NA	RM	TO
BA	0	662	877	255	412	996
FI	662	0	295	468	268	400
MI	877	295	0	754	564	138
NA	255	468	754	0	219	869
RM	412	268	564	219	0	669
TO	996	400	138	869	669	0



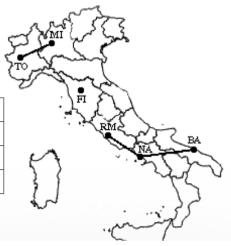
	BA	FI	MI/TO	NA	RM
BA	0	662	877	255	412
FI	662	0	295	468	268
MI/TO	877	295	0	754	564
NA	255	468	754	0	219
RM	412	268	564	219	0



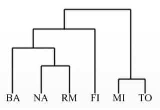
	BA	FI	MI/TO	NA/RM
BA	0	662	877	255
FI	662	0	295	268
MI/TO	877	295	0	564
NA/RM	255	268	564	0

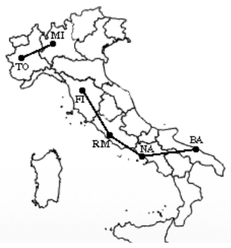


	BA/NA/RM	FI	MI/TO
BA/NA/RM	0	268	564
FI	268	0	295
MI/TO	564	295	0



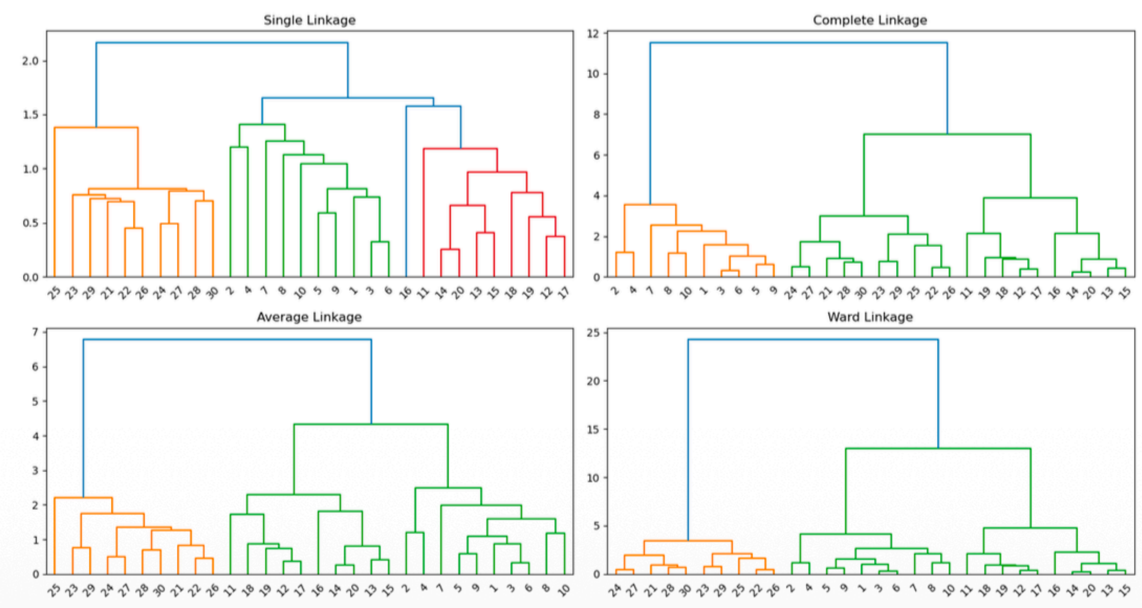
	BA/FI/NA/RM	MI/TO
BA/FI/NA/RM	0	295
MI/TO	295	0





Comparison between linkage methods

Here we can see the final dendrogram based on what linkage algorithm we chose.



Density-based clustering

Density-based clustering identifies clusters as contiguous high-density regions separated by contiguous low-density regions. Unlike centroid-based methods like K-means, it can discover clusters of arbitrary shapes and naturally handles noise and outliers.

Computing density

Two primary approaches exist:

Grid-based method:

- Partition the feature space into a regularly spaced grid of hypercubes
- Count the number of objects falling within each grid cell
- Density is estimated per cell

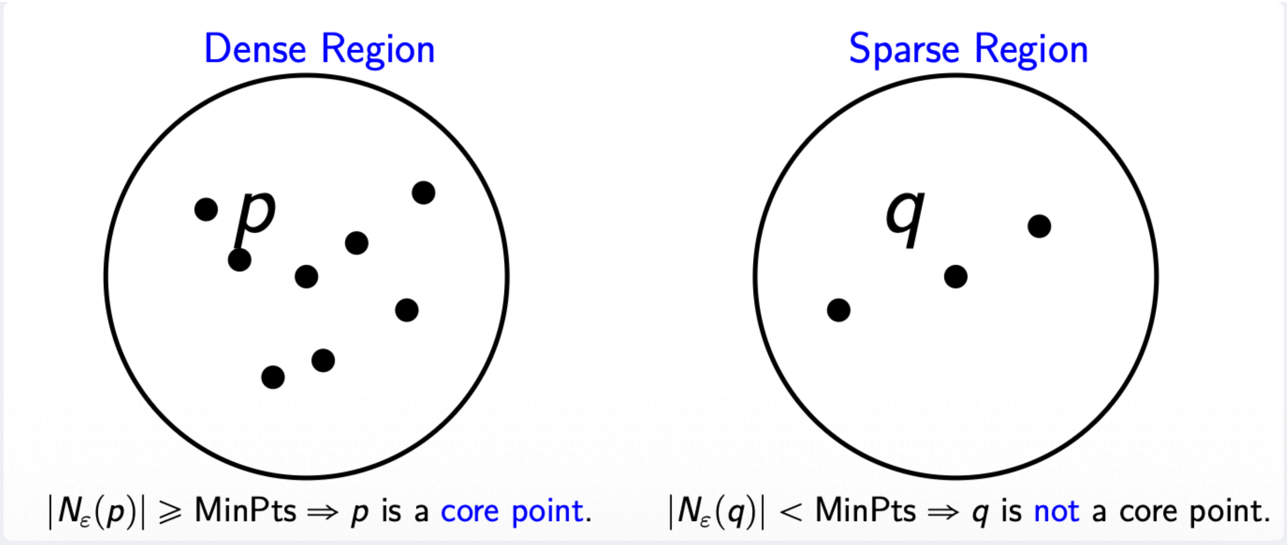
Object-centered method:

- For each data point p, define a hypersphere of fixed radius ϵ centered at p
- Count the number of other points q such that $dist(p, q) \leq \epsilon$
- This count represents the local density around p

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

Core concepts

> Define the ϵ-neighborhood of a point p as:
>
> $$N_\epsilon(p) = \{q \in D \mid dist(p, q) \leq \epsilon\}$$
>
> where D is the dataset and $dist$ is a distance metric (typically Euclidean).


Dense Region
Sparse Region
p
q
$|N_\varepsilon(p)| \geqslant \text{MinPts} \Rightarrow p$ is a core point.
$|N_\varepsilon(q)| < \text{MinPts} \Rightarrow q$ is not a core point.

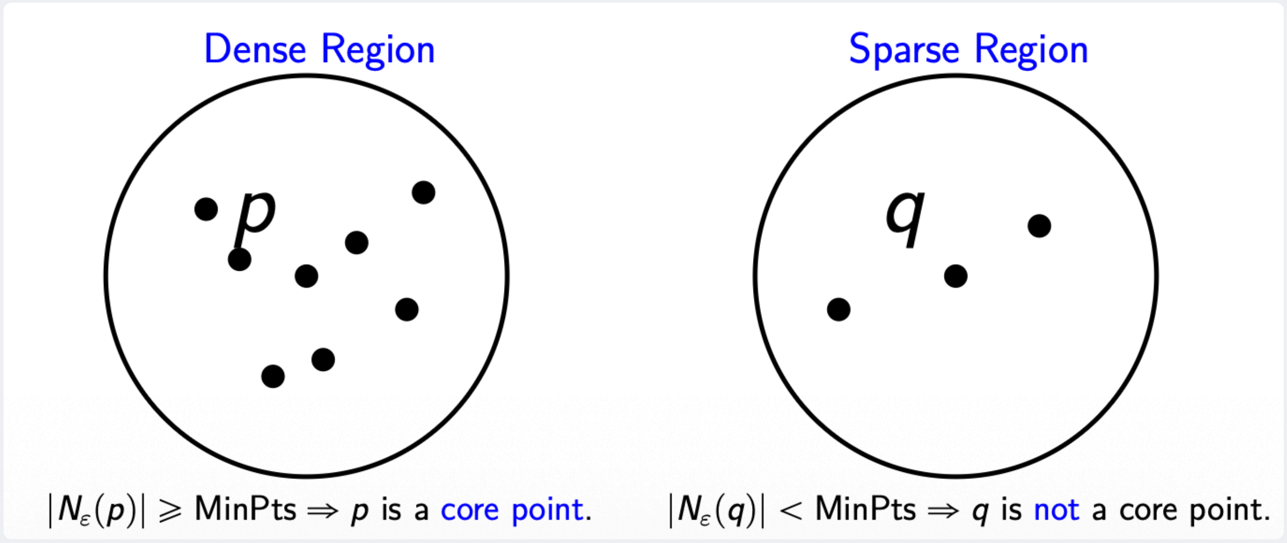
Given a parameter MinPts:

Core point: A point p is a core point if $|N_\epsilon(p)| \geq \text{MinPts}$

Border point: A point p is a border point if:

1. $|N_\epsilon(p)| < \text{MinPts}$ (not a core point)
2. $\exists q$ (a core point) such that $p \in N_\epsilon(q)$

Noise point: A point that is neither a core point nor a border point


Dense Region
Sparse Region
p
q
$|N_\varepsilon(p)| \geqslant \text{MinPts} \Rightarrow p$ is a core point.
$|N_\varepsilon(q)| < \text{MinPts} \Rightarrow q$ is not a core point.

Density relationships

Direct density reachability: Point p is directly density reachable from point q if:

1. q is a core point
2. $p \in N_\epsilon(q)$

Note: Direct density reachability is not symmetric. If p is a border point, q may not be directly density reachable from p.

Density reachability: Point p is density reachable from point q if there exists a chain of points $q_1, q_2, \ldots, q_n$ such that:

- $q_1 = q$
- $q_n = p$

- q_{i+1} is directly density reachable from q_i for all $i \in \{1, 2, \ldots, n-1\}$

Density connection: Points p and q are density connected if there exists a point s such that both p and q are density reachable from s.

Note: Density connection is symmetric.

Cluster formation

A **cluster** C is a non-empty subset of D satisfying:

1. **Maximality**: If $p \in C$ and q is density reachable from p, then $q \in C$
2. **Connectivity**: For any $p, q \in C$, p and q are density connected

Noise: All points not belonging to any cluster are labeled as noise.

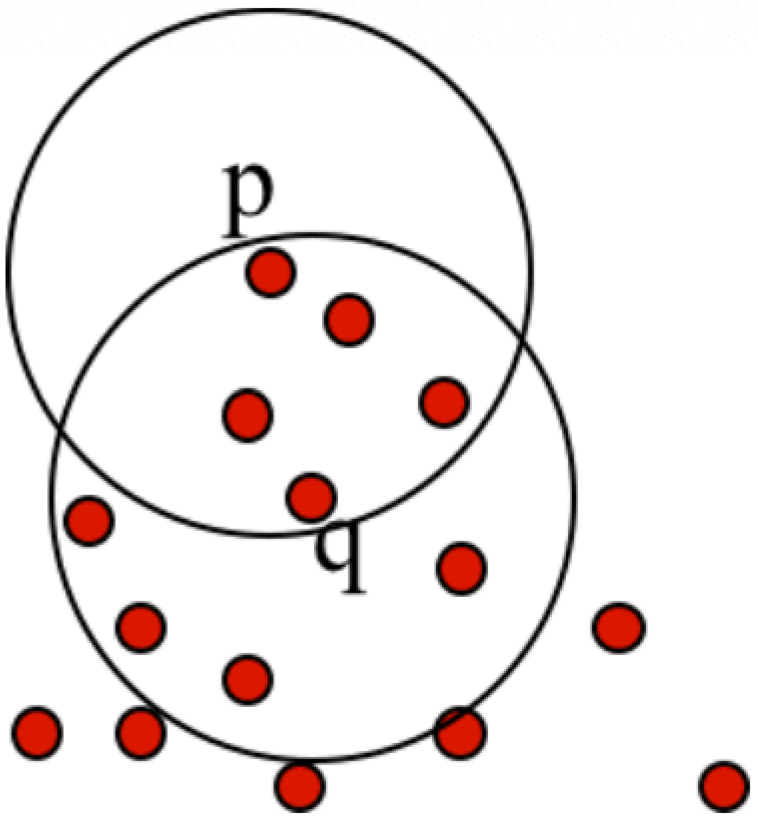
The algorithm requires two parameters: ϵ (neighborhood radius) and MinPts (minimum points to form a dense region).

Neighborhood definition

Given a radius parameter ϵ, we define the neighborhood of a point p as the ϵ-hypersphere centered at that point:

$$N_\epsilon(p) = \{q \in D \mid dist(p, q) \leq \epsilon\}$$

The neighborhood relationship is symmetric: if p is in the neighborhood of q, then q is also in the neighborhood of p.



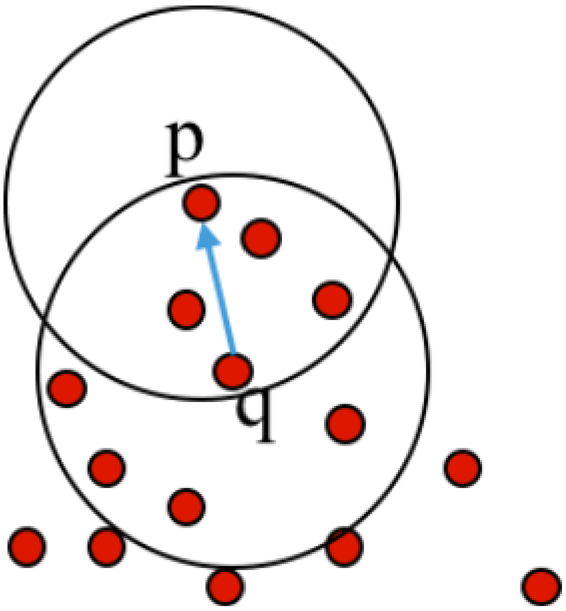
Direct density reachability

Given a threshold parameter minPts, we classify points as:

- **Core point**: $|N_\epsilon(p)| \geq \text{minPts}$
- **Border point**: $|N_\epsilon(p)| < \text{minPts}$ but $\exists$ core point q with $p \in N_\epsilon(q)$

A point p is **directly density reachable** from point q if:

1. q is a core point
2. $p \in N_\epsilon(q)$



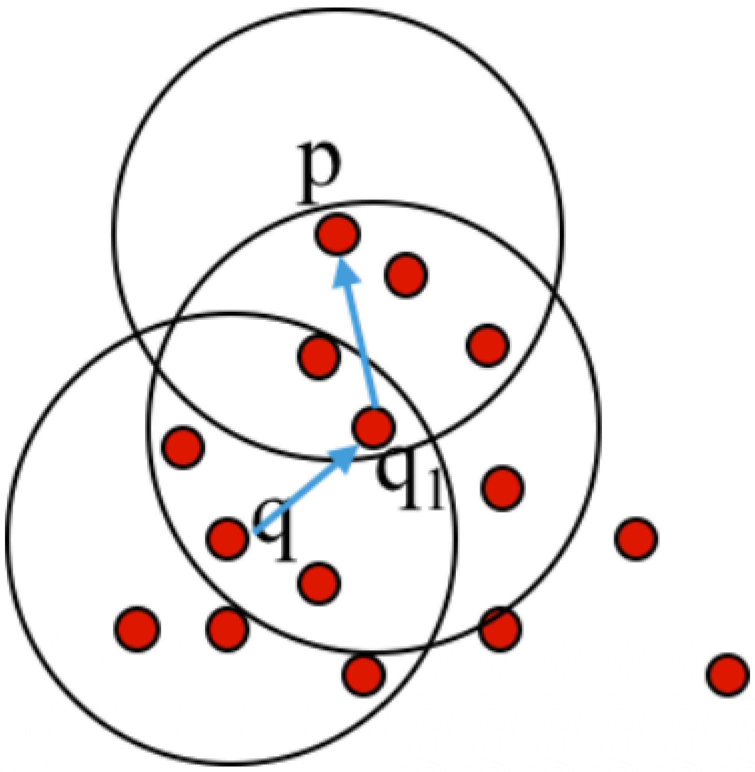
> **Important**: Direct density reachability is not symmetric. If p is a border point and q is a core point, p is directly density reachable from q, but q is not directly density reachable from p.

Density reachability

A point p is **density reachable** from point q if:

1. q is a core point
2. There exists a sequence of points $q_1, q_2, \ldots, q_n$ such that:
   - q_1 is directly density reachable from q
   - q_{i+1} is directly density reachable from q_i for $i \in \{1, 2, \ldots, n-1\}$

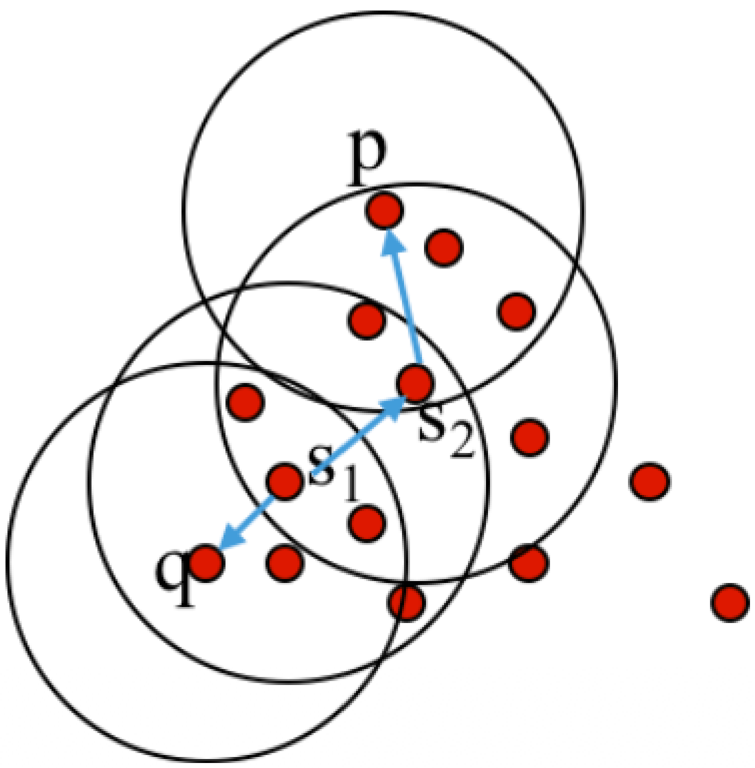
- p is directly density reachable from q_n



> **Important**: Density reachability is also not symmetric. If q is density reachable from p, it does not necessarily mean p is density reachable from q.

Density connection

A point p is **density connected** to point q if there exists a point s such that both p and q are density reachable from s.



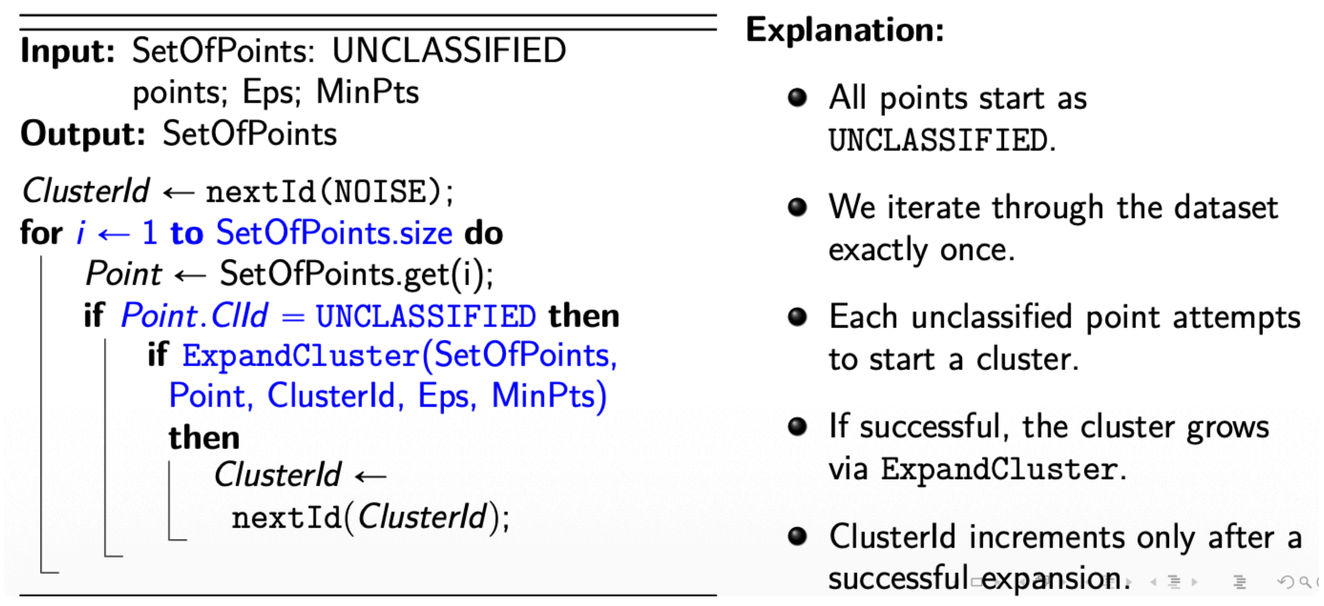
Important: Unlike the previous relationships, density connection is symmetric. If p is density connected to q, then q is density connected to p.

Cluster generation

A **cluster** is defined as a maximal set of points that are density connected to each other.

Noise points are border points that are not density connected to any core point. These points do not belong to any cluster and are considered outliers or noise in the dataset.

DB-SCAN main loop



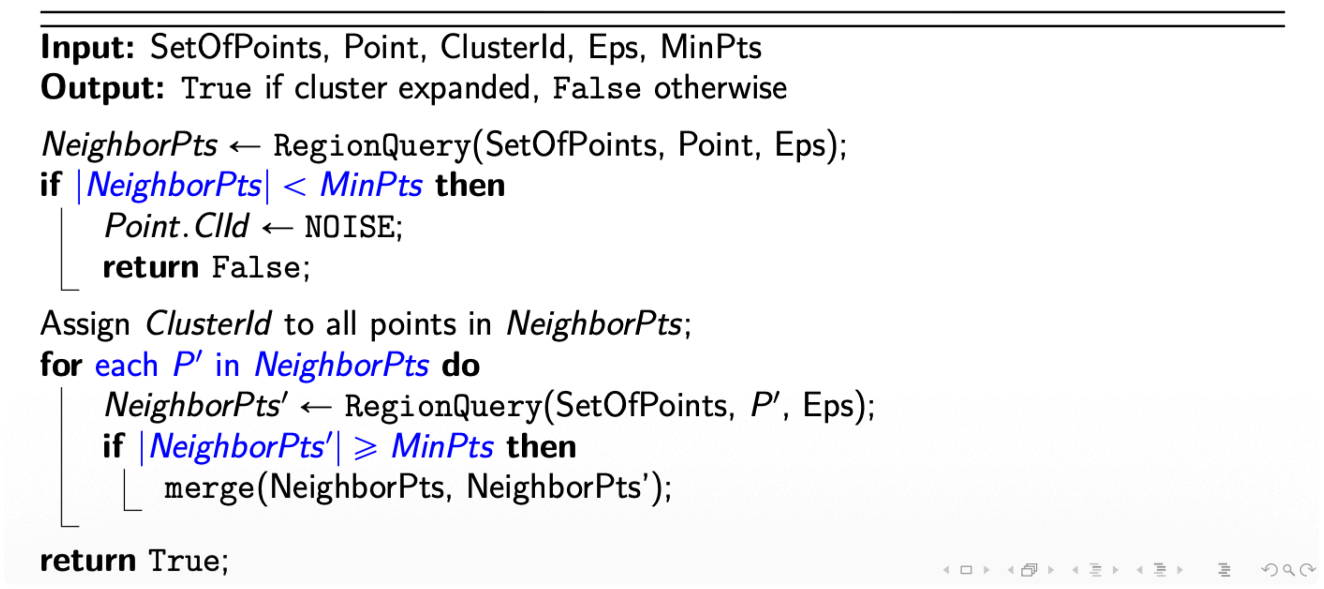
```
Input: SetOfPoints: UNCLASSIFIED
       points; Eps; MinPts
Output: SetOfPoints

ClusterId ← nextId(NOISE);
for i ← 1 to SetOfPoints.size do
    Point ← SetOfPoints.get(i);
    if Point.ClId = UNCLASSIFIED then
        if ExpandCluster(SetOfPoints,
          Point, ClusterId, Eps, MinPts)
          then
            ClusterId ←
              nextId(ClusterId);
```

Explanation:

- All points start as `UNCLASSIFIED`.
- We iterate through the dataset exactly once.
- Each unclassified point attempts to start a cluster.
- If successful, the cluster grows via `ExpandCluster`.
- ClusterId increments only after a successful expansion.

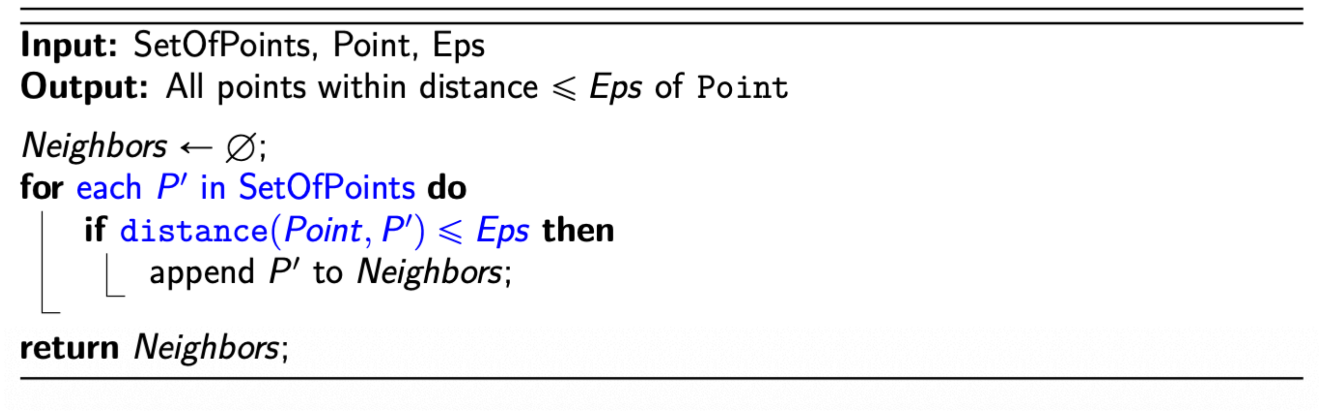
ExpandCluster subroutine



```
Input: SetOfPoints, Point, ClusterId, Eps, MinPts
Output: True if cluster expanded, False otherwise

$NeighborPts \leftarrow$ RegionQuery(SetOfPoints, Point, Eps);
if $|NeighborPts| < MinPts$ then
    $Point.ClId \leftarrow$ NOISE;
    return False;
Assign $ClusterId$ to all points in $NeighborPts$;
for each $P'$ in $NeighborPts$ do
    $NeighborPts' \leftarrow$ RegionQuery(SetOfPoints, $P'$, Eps);
    if $|NeighborPts'| \geqslant MinPts$ then
        merge(NeighborPts, NeighborPts');
return True;
```

RegionQuery subroutine

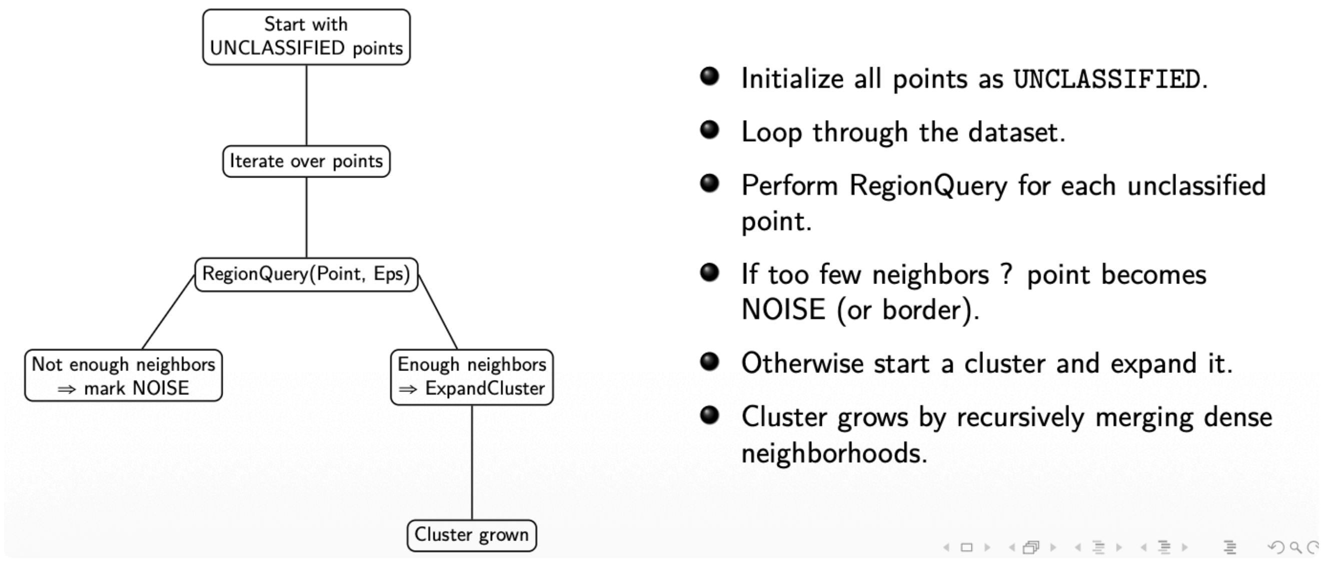


```
Input: SetOfPoints, Point, Eps
Output: All points within distance $\leqslant Eps$ of Point

$Neighbors \leftarrow \varnothing$;
for each $P'$ in SetOfPoints do
    if distance$(Point, P') \leqslant Eps$ then
        append $P'$ to $Neighbors$;
return $Neighbors$;
```

Full algorithm

After looking at the individual phases we can combine them and understand the whole algorithm.


Start with UNCLASSIFIED points
Iterate over points
RegionQuery(Point, Eps)
Not enough neighbors ⇒ mark NOISE
Enough neighbors ⇒ ExpandCluster
Cluster grown

- Initialize all points as UNCLASSIFIED.
- Loop through the dataset.
- Perform RegionQuery for each unclassified point.
- If too few neighbors ? point becomes NOISE (or border).
- Otherwise start a cluster and expand it.
- Cluster grows by recursively merging dense neighborhoods.

START

Start with all data points marked as **UNCLASSIFIED**.

MAIN LOOP

Iterate through each unclassified point in the dataset:

1. **Region Query**: For the current point, find all points within distance ϵ (its ϵ-neighborhood).
2. **Core Point Check**:
   - If the neighborhood contains fewer than **minPts** points, mark the current point as **NOISE**.
   - If the neighborhood contains **minPts** or more points, the current point is a **core point** - proceed to cluster expansion.

CLUSTER EXPANSION

When a core point is found:

- Create a new cluster and assign the core point to it.
- Add all points in the core point's ϵ-neighborhood to a "seed set".

While the seed set is not empty:

- Remove a point from the seed set.
- If that point is unclassified or noise:
  - Assign it to the current cluster.
  - Check if it's a core point by performing another region query.
  - If it's a core point, add all points from its ϵ-neighborhood to the seed set (excluding those already processed).

Key Details

- **Noise points** may be reclassified later if they're within ϵ of a core point from another cluster.
- The algorithm ensures that all density-connected points end up in the same cluster.
- Points are only visited once, making the algorithm efficient.
- The process continues until all points are classified (either in a cluster or as noise).

Termination

When all points have been processed:

- All core points and their density-connected border points are assigned to clusters.
- Remaining unclassified points are marked as noise.

This algorithm naturally discovers clusters of arbitrary shapes and handles outliers without requiring the number of clusters as input.

Question: How to set ϵ and minPoints? → Parameter selection for DBSCAN

General guidelines

As with many machine learning algorithms, a grid search over hyperparameter combinations is recommended. Some useful heuristics exist:

For minPts:

- Try $minPts = 2 \times D$, where D is the number of dimensions
- If the algorithm identifies too much noise, increase $minPts$

For ϵ:

- Requires more careful consideration
- Use the distance to the k-nearest neighbor, with $k = minPts$

The k-distance plot method

1. **Compute k-distances**: For each point, calculate the distance to its k-th nearest neighbor (with $k = minPts$).
2. **Sort distances**: Sort all points by their k-distance in descending order.
3. **Create plot**: Plot the sorted k-distances (y-axis) against point indices (x-axis).
4. **Identify elbow**: Datasets with clustering tendency typically show a change in slope (an "elbow") in this plot.
5. **Select ϵ**: Choose ϵ in the region of the elbow point. Points to the left of this ϵ value (with higher k-distance) will be considered border or noise points.

Practical application

- For a dataset with 1500 points and $minPts = 4$, the k-distance plot might suggest fine-tuning ϵ in the interval 0.2–0.3.
- Perform grid search around the elbow region to find the optimal ϵ.

PROS:

- Discovers clusters of arbitrary shapes
- Robust to noise and outliers
- Complexity: $O(N^2)$ for naive implementation
- Can be reduced to $O(N \log N)$ with spatial indexing (e.g., R*-tree)

CONS:

- Struggles with clusters of widely varying densities
- Very sensitive to ϵ and $minPts$ parameter choices
- Decreasing ϵ or increasing $minPts$ reduces cluster sizes and increases noise points
- Requires careful parameter tuning for each dataset

Model-based clustering

Model-based clustering estimates parameters of a statistical model to maximize the model's ability to explain the data. The main technique uses **mixture models**, where data is viewed as observations from a mixture of different probability distributions.

Key concepts:

- Typically uses multivariate normal distributions as base models (well-understood, mathematically tractable)
- Parameter estimation via **maximum likelihood**
- Given dataset X, the likelihood function expresses data probability as a function of model parameters
- Attributes are assumed to be random independent variables

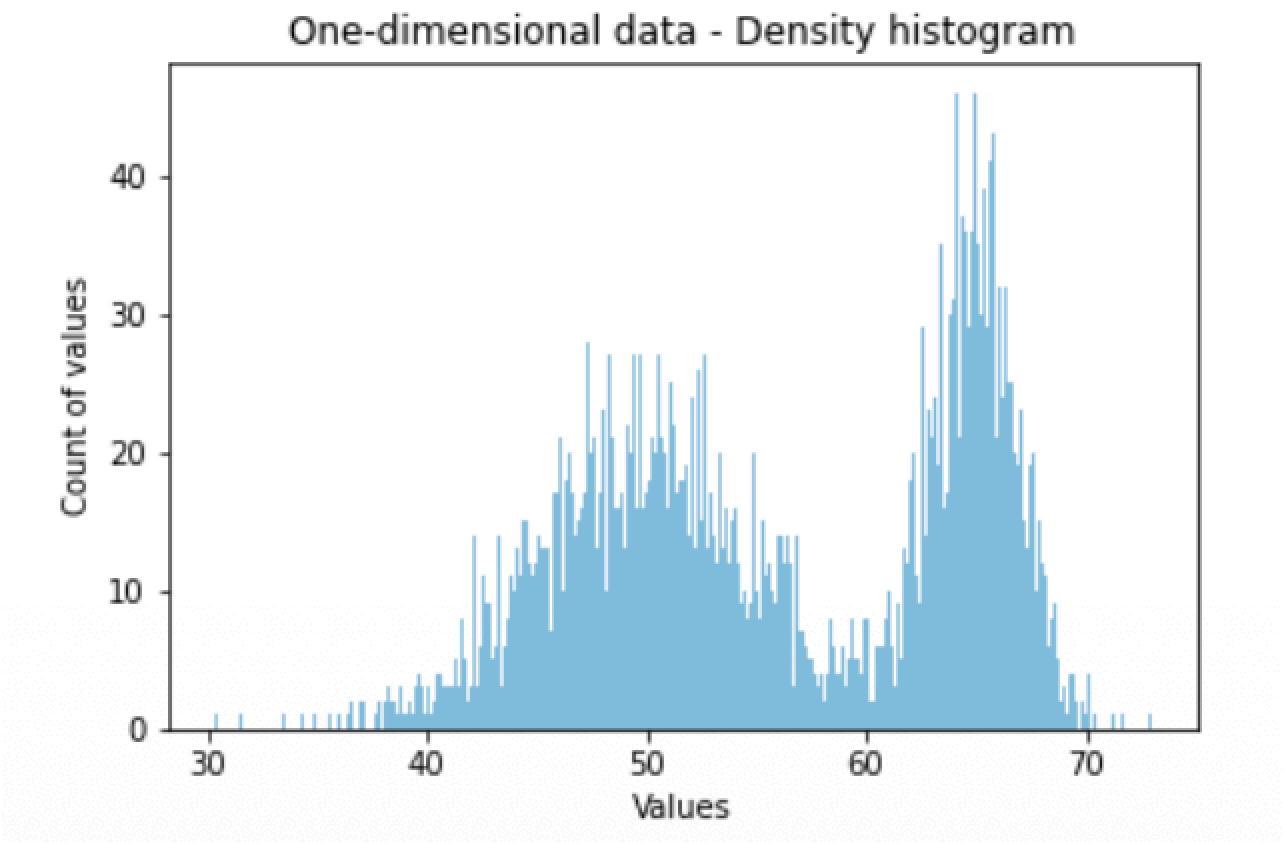
Gaussian mixture models (Expectation-Maximization algorithm)

When data can be approximated by a single distribution, parameter derivation is straightforward. For multiple mixed distributions, the **EM algorithm** is used.

EM algorithm steps:

1. **Select initial parameters** for the mixture model
2. **Repeat until convergence**:
   - **Expectation step**: For each object, calculate probability of belonging to each distribution
   - **Maximization step**: Given these probabilities, find new parameter estimates that maximize expected likelihood
3. **Terminate** when parameters stabilize

One-dimensional mixture example



Consider two clusters A and B with normal distributions:

- Cluster A: $\mu_A = 50$, $\sigma_A = 5$, sampling probability $p_A = 0.6$
- Cluster B: $\mu_B = 65$, $\sigma_B = 2$, sampling probability $p_B = 0.4$

The probability density function combines both distributions, creating a "mountain range" with peaks for each component.

Parameter estimation problem

Given only the data points (without class labels), estimate the five parameters: μ_A, σ_A, μ_B, σ_B, and p_A ($p_B = 1 - p_A$).

EM algorithm for one dimension, two clusters

Need to estimate 5 parameters: μ_A, σ_A, μ_B, σ_B, p_A

Probability calculation:
Using Bayes' theorem:

$$Pr(A|x) = \frac{Pr(x|A)Pr(A)}{Pr(x)} = \frac{f(x;\mu_A,\sigma_A)p_A}{Pr(x)}$$

Where the Gaussian probability density function is:

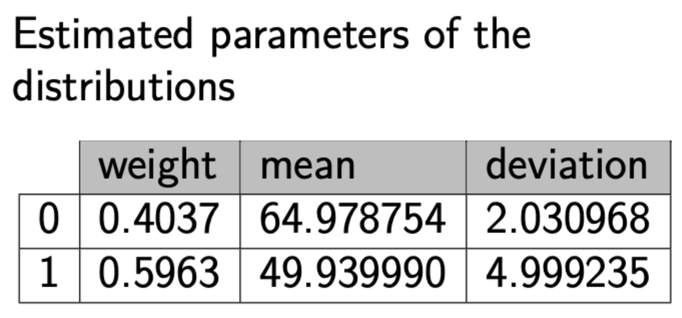
$$f(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Algorithm execution:

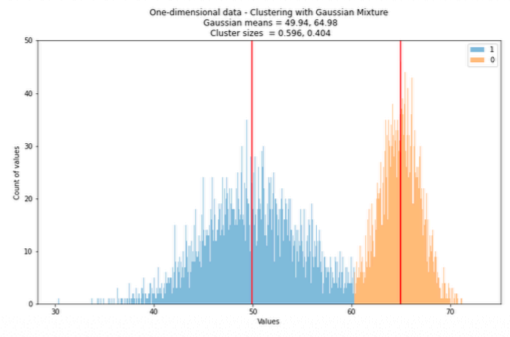
1. **Expectation step**: Compute $Pr(A|x)$ and $Pr(B|x)$ using current distribution parameters
   - Compute numerators for both probabilities
   - Normalize by dividing by their sum
2. **Maximization step**: Update distribution parameters by weighting probabilities according to current distributions
3. **Repeat** until convergence
4. **Final assignment**: Label each object with cluster A or B according to maximum probability using final distribution parameters

This approach eliminates "brittleness" of hard assignments by using soft probabilities for cluster membership.

Example parameters estimation with this method:


Estimated parameters of the distributions

	weight	mean	deviation
0	0.4037	64.978754	2.030968
1	0.5963	49.939990	4.999235



BIRCH clustering

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is a scalable clustering algorithm designed for large datasets. It incrementally builds a compact representation called a **CF-tree** (Clustering Feature Tree) that summarizes data points using three statistics:

- N: number of points
- LS: linear sum of points
- SS: squared sum of points

This structure enables fast, memory-efficient clustering with a single scan of the data.

The CF node

Each CF-tree node stores clustering features that summarize a subcluster. A CF triple $\langle N, LS, SS \rangle$ compactly captures centroid and radius information:

- Internal nodes group subclusters hierarchically
- Leaf nodes contain final subcluster summaries

This allows BIRCH to operate with constrained memory while preserving cluster quality.

Algorithm phases

1. **Build CF-tree**: Insert each point; absorb into closest subcluster if within threshold T, otherwise split
2. **Condense CF-tree**: Optionally remove outliers or merge small subclusters
3. **Global clustering**: Apply standard clustering (e.g., agglomerative) to leaf subclusters
4. **Refinement**: Optionally reassign original data to improved cluster centers

Phases 3 and 4 are optional but improve accuracy.

Advantages and limitations

Advantages:

- Designed for very large datasets (single scan)
- Low memory usage via compact CF representations
- Naturally supports incremental and dynamic updates
- Performs well for numerical, metric-space data

Limitation: Struggles with non-spherical or poorly separated clusters due to centroid-based thresholding

Spectral clustering

Spectral clustering uses the eigenstructure of a similarity graph to partition data into clusters. It transforms data into a low-dimensional space using the graph Laplacian, where clusters become more easily separable. Works especially well for non-convex or manifold-shaped clusters.

Similarity graph construction

Given data points, construct a weighted graph G:

- Vertices = data points
- Edge weights = similarities (e.g., Gaussian kernel)
- Matrix form: adjacency matrix W

The choice of similarity function and neighborhood size strongly influences results.

Graph Laplacian and embedding

Compute a Laplacian matrix of the graph, such as:

$$L = D - W \quad \text{or} \quad L_{sym} = I - D^{-1/2} W D^{-1/2}$$

where D is the degree matrix.

Extract the first k eigenvectors of L to form an embedding in $\mathbb{R}^k$ that separates clusters.

Clustering step

Apply standard clustering (typically K-means) to the rows of the eigenvector matrix. These rows represent the data in a spectral embedding. Cluster assignments in this space map back to clusters in the original data. This combination captures structure missed by purely distance-based methods.

Advantages and limitations

Advantages:

- Captures non-linear and arbitrary-shaped clusters
- Works well when clusters are connected components in a graph

Limitations:

- Requires constructing and storing a similarity matrix
- Not ideal for very large datasets due to eigenvector computation

method name	parameters	scalability	usecase	geometry (metric used)
k-means	number of clusters	very large n_samples, medium n_clusters with minibatch code	general-purpose, even cluster size, flat geometry, not too many clusters, inductive	distances between points
affinity propagation	damping, sample preference	not scalable with n_samples	many clusters, uneven cluster size, non-flat geometry, inductive	graph distance (e.g. nearest-neighbor graph)
mean-shift	bandwidth	not scalable with n_samples	many clusters, uneven cluster size, non-flat geometry, inductive	distances between points
spectral clustering	number of clusters	medium n_samples, small n_clusters	few clusters, even cluster size, non-flat geometry, transductive	graph distance (e.g. nearest-neighbor graph)
ward hierarchical clustering	number of clusters or distance threshold	large n_samples and n_clusters	many clusters, possibly connectivity constraints, transductive	distances between points
agglomerative clustering	number of clusters or distance threshold, linkage type, distance	large n_samples and n_clusters	many clusters, possibly connectivity constraints, non euclidean distances, transductive	any pairwise distance

method name	parameters	scalability	usecase	geometry (metric used)
dbscan	neighborhood size	very large n_samples, medium n_clusters	non-flat geometry, uneven cluster sizes, outlier removal, transductive	distances between nearest points
optics	minimum cluster membership	very large n_samples, large n_clusters	non-flat geometry, uneven cluster sizes, variable cluster density, outlier removal, transductive	distances between points
gaussian mixtures	many	not scalable	flat geometry, good for density estimation, inductive	mahalanobis distances to centers
birch	branching factor, threshold, optional global cluster	large n_clusters and n_samples	large dataset, outlier removal, data reduction, inductive	euclidean distance between points