Machine Learning

Clustering - Beyond KMeans

Claudio Sartori

DISI
Department of Computer Science and Engineering – University of Bologna, Italy
claudio.sartori@unibo.it

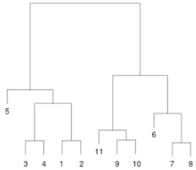
Hierarchical clustering

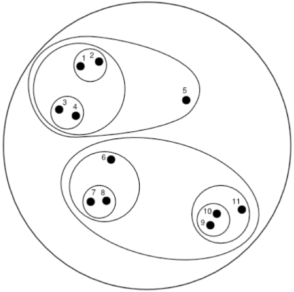
Generates a nested structure of clusters

- Agglomerative (bottom up)
  - as a starting state, each data point is a cluster
  - in each step the two less separated clusters are merged into one
  - a measure of separation between clusters is needed
- Divisive (top down)
  - as a starting state, the entire dataset is the only cluster
  - in each step, the cluster with the lowest cohesion is split
  - a measure of cluster cohesion and a split procedure are needed

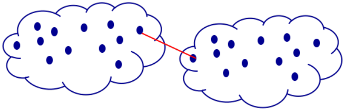
Hierarchical clustering output

- Dendrogram (left)
- Nested cluster diagram (right)
- They represent the same structure
- The representation is the same for agglomerative and divisive
- The agglomerative methods are the most used



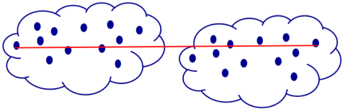


Separation between clusters – Graph based



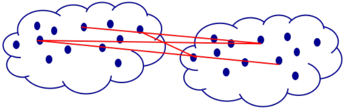
Single Link

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



Complete Link

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



Average Link

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

The distance between sets is based on the distances between objects belonging to the two sets, respectively

Separation between clusters – Ward's method

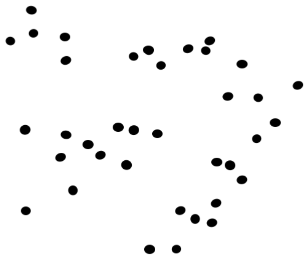
Given two sets S_1 and S_2 with the respective $SSE(S_1)$ and $SSE(S_2)$, the distance is computed as

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

Smaller separation implies a lower increase in the SSE after merging

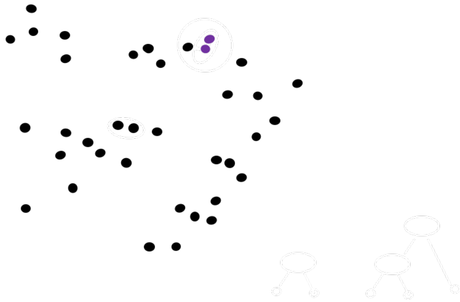
Single linkage hierarchical clustering I

1. Initialization: every object is a cluster



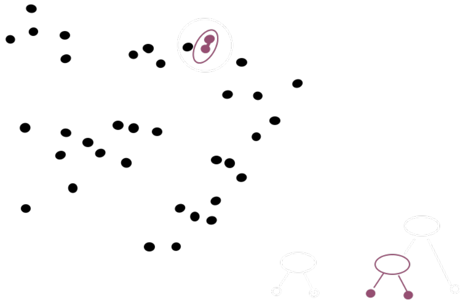
Single linkage hierarchical clustering II

1. Initialization: every object is a cluster
2. Find the less separated pair



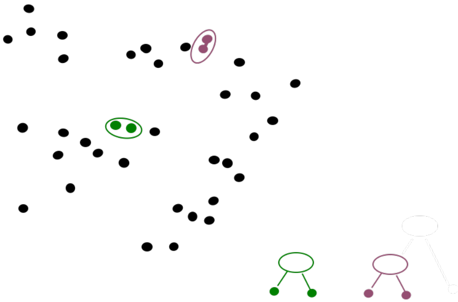
Single linkage hierarchical clustering III

1. Initialization: every object is a cluster
2. Find the less separated pair
3. Merge in a single cluster



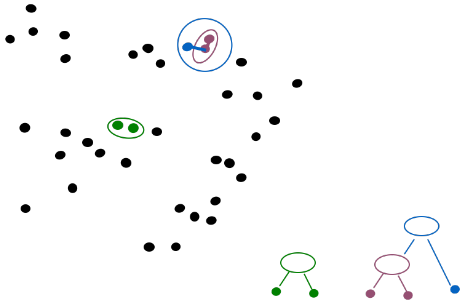
Single linkage hierarchical clustering IV

1. Initialization: every object is a cluster
2. Find the less separated pair
3. Merge in a single cluster
4. Repeat



Single linkage hierarchical clustering V

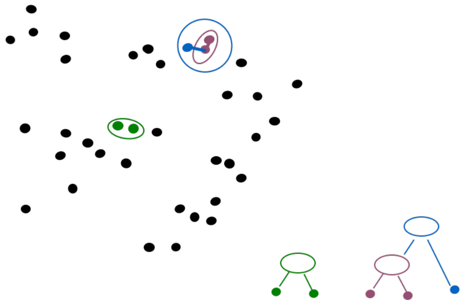
1. Initialization: every object is a cluster
2. Find the less separated pair
3. Merge in a single cluster
4. Repeat ...



Single linkage hierarchical clustering VI

1. Initialization: every object is a cluster
2. Find the less separated pair
3. Merge in a single cluster
4. Repeat ...

The result is a dendrogram (taxonomy, object hierarchy)



Single linkage algorithm

- Initialize the clusters, one for each objects
- Compute the distance matrix between the clusters, squared, symmetric, the size is the number of objects N, the main diagonal is null
- While the number of clusters is greater than 1
  - find the two clusters with lowest separation, say k_r and k_s
  - merge them in a cluster
  - delete from the distance matrix the rows and columns r and s and insert one new row and column with the distances of the new cluster from the others

$$\text{Dist}(k_k, k_{(r+s)}) = \min(\text{Dist}(k_k, k_r), \text{Dist}(k_k, k_s)) \forall k \in [1, K]$$

Time and space complexity

- Space and time: $\mathcal{O}(N^2)$ for the computation and the storage of the distance matrix
- Worst case $N-1$ iterations to reach the final single cluster
- For the i-th step of the main iteration:
  - search of the pair to merge $\mathcal{O}((N-i)^2)$
  - recomputation of the distance matrix $\mathcal{O}((N-i))$
- Time, in summary: $\mathcal{O}(N^3)$
  - can be reduced to $\mathcal{O}(N^2\log(N))$ with indexing structures

Italian cities example I

	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

Italian cities example II

	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

Italian cities example III

	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

Italian cities example IV

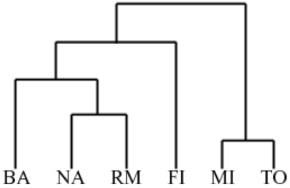
	BA	FI	MI/TO	NA/RM
BA	0	662	877	255
FI	662	0	295	268
MI/TO	877	295	0	564
NA/RM	255	268	564	0

Italian cities example V

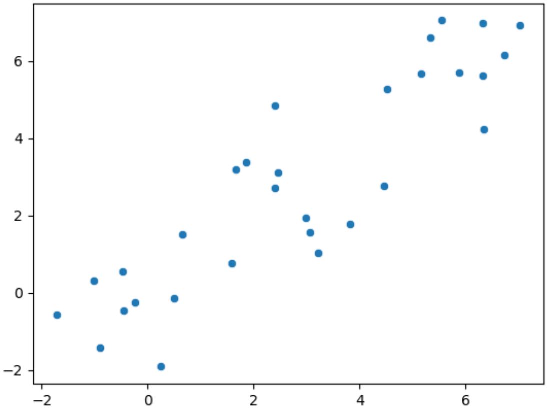
	BA/NA/RM	FI	MI/TO
BA/NA/RM	0	268	564
FI	268	0	295
MI/TO	564	295	0

Italian cities example VI

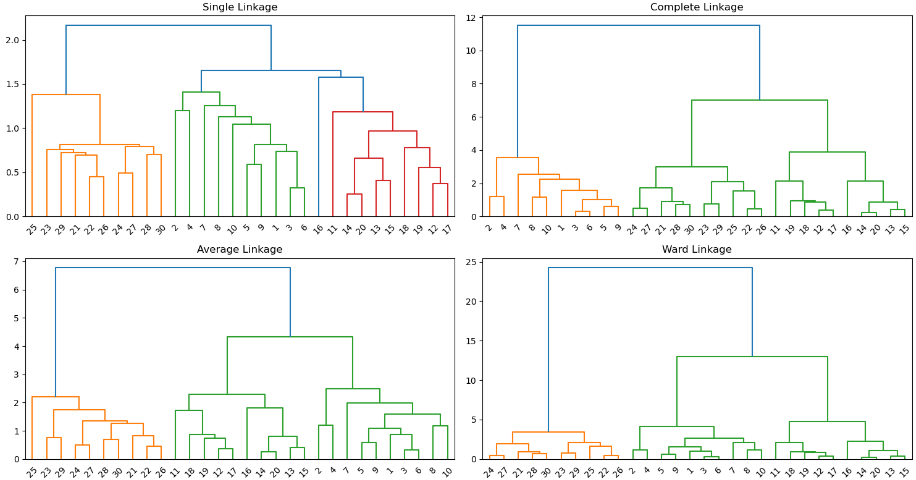
	BA/FI/NA/RM	MI/TO
BA/FI/NA/RM	0	295
MI/TO	295	0



Comparison of linking methods - Sample data

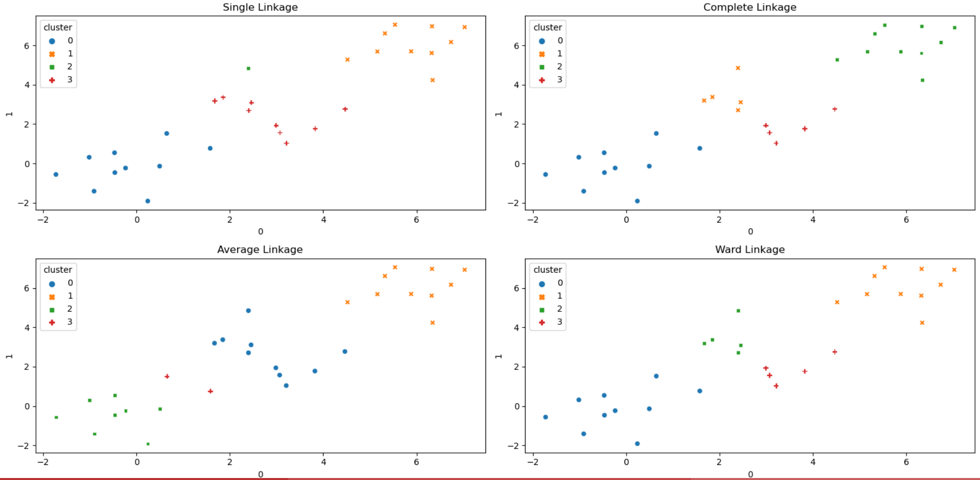


Comparison of linking methods - Dendrograms



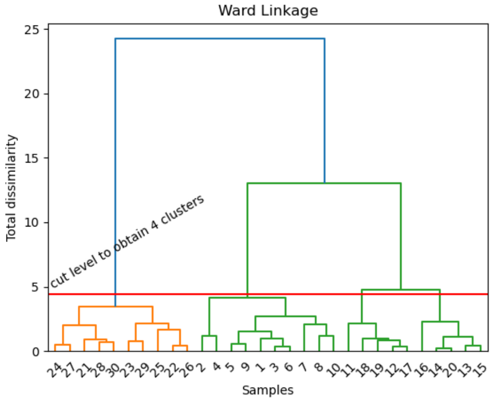
Comparison of linking methods

Labelled data with 4 clusters



Generating the clustering scheme

- The desired clustering scheme is obtained by choosing the number of clusters
- The choice of the level is application dependent, and can also be guided by indexes, as in the case of K-means
- The choice is equivalent to cutting the dendrogram at the appropriate level



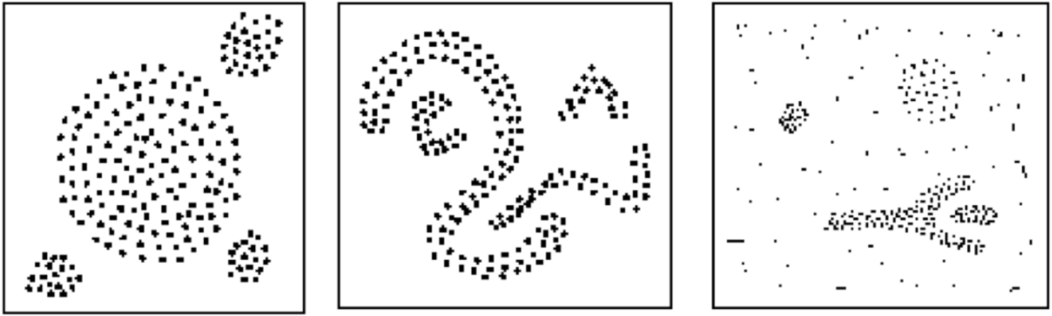
Discussion I

- The vertical axis in the dendrogram is the total dissimilarity inside the clusters, which obviously increases for decreasing number of clusters
- The diameter of a cluster is the distance among the most separated objects
  - Single linkage tends to generate clusters with larger diameters also at low levels
  - Complete linkage tends to generate more compact clusters
  - Ward linkage tends to generate more spheric clusters

Discussion II

- ☹ The scaling is poor, due to the high complexity
- ☹ There isn't a global objective function, the decision is always local and cannot be undone
- ☺ The dendrogram structure is of great help for the interpretation of the result
- ☺ Empirically, the result is frequently good

Density based clustering



Clusters are high–density regions separed by low–density regions

Computing density

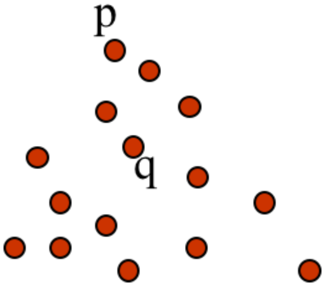
The two most obvious solutions

- Grid–based
  - split the (hyper)space into a regularly spaced grid
  - count the number of objects inside each grid element
- Object–centered
  - define the radius of a (hyper)sphere
  - attach to each object the number of objects which are inside that sphere

DBSCAN – Density Based Spatial Clustering of Applications with Noise[1]

Intuition

- intuitively, p is a border point, while q is a core point

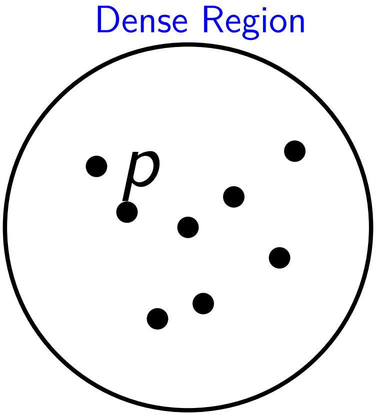


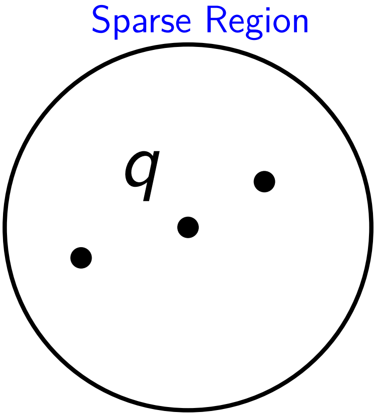
1 [Ester et al.(1996)Ester, Kriegel, Sander, and Xu]

Formalisation of density vs sparsity

- define the neighbourhood of a point p
- idea: define a hypersphere of radius ϵ centered in the p
- count how many neighbours are in the hypersphere $|N_\varepsilon(p)|$
- define a threshold

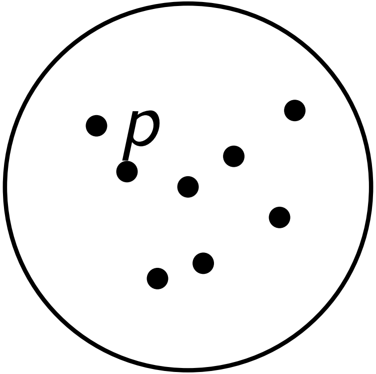
Eps-Neighborhoods: Dense vs Sparse Regions



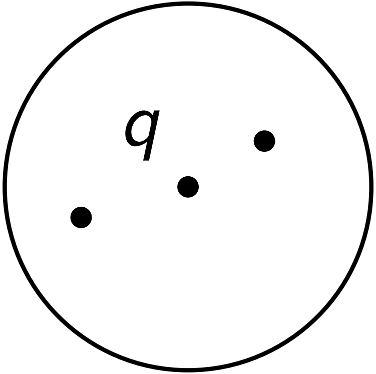


Eps-Neighborhoods: Dense vs Sparse Regions

Dense Region



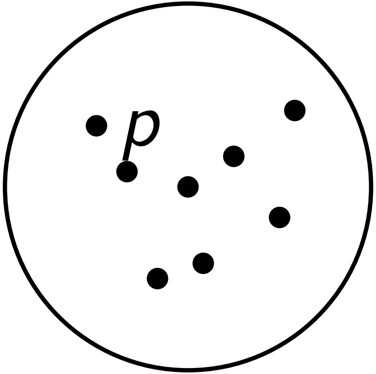
Sparse Region



$|N_\varepsilon(p)| \geqslant \text{MinPts} \Rightarrow p$ is a core point.

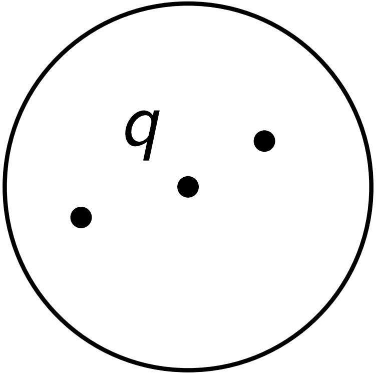
Eps-Neighborhoods: Dense vs Sparse Regions

Dense Region



$|N_\varepsilon(p)| \geqslant \text{MinPts} \Rightarrow p$ is a core point.

Sparse Region

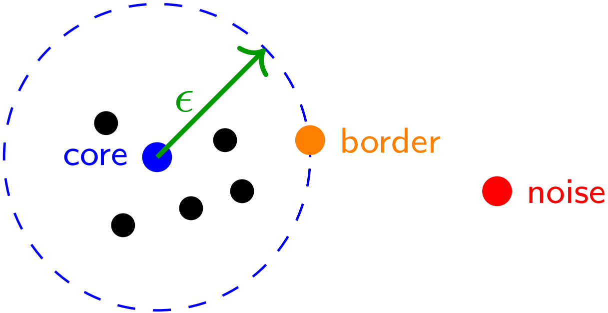


$|N_\varepsilon(q)| < \text{MinPts} \Rightarrow q$ is not a core point.

Core, Border, and Noise Points

Definitions

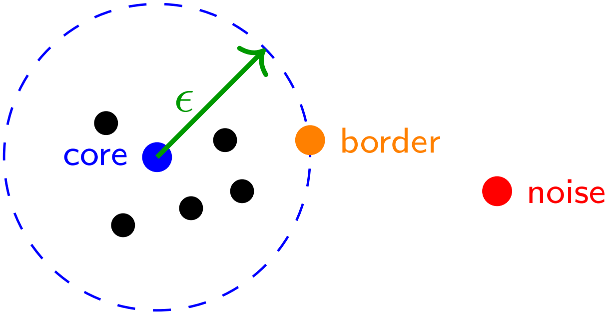
- Core point: at least `MinPts` points within ε.



Core, Border, and Noise Points

Definitions

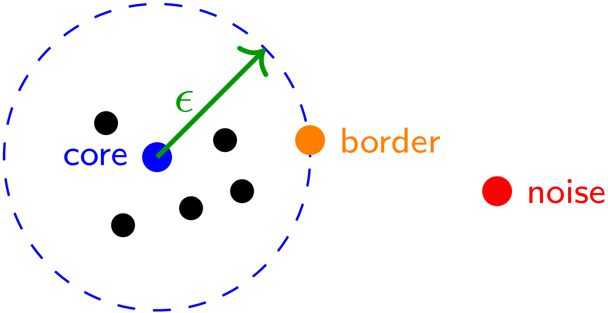
- Core point: at least `MinPts` points within ε.
- Border point: not a core point, but neighbor of a core.



Core, Border, and Noise Points

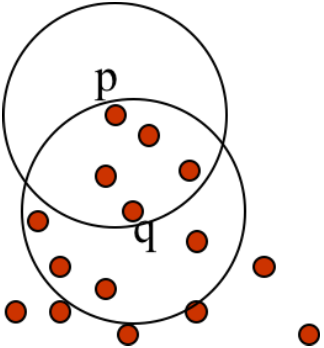
Definitions

- Core point: at least `MinPts` points within ε.
- Border point: not a core point, but neighbor of a core.
- Noise point: neither core nor border.



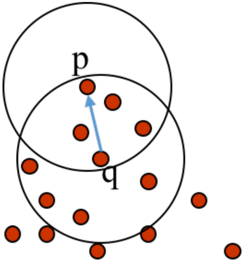
Neighborhood

- define a radius ϵ and define as neighborhood of a point the ϵ–hypersphere centered at that point
- points p and q are one in the neighborhood of the other
  - neighborhood is symmetric



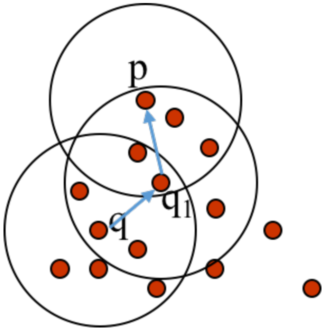
Direct Density Reachability

- define a threshold $\mathrm{minPoints}$ and define as core a point with at least $\mathrm{minPoints}$ points in its neighborhood, as border otherwise
  - with $\mathrm{minPoints} = 5$, q is core, p is border
- define that a point p is directly density reachable from point q iff
  - q is core
  - q is in the neighborhood of p
- direct density reachability is not symmetric
  - in the example q is not directly density reachable from p, since p is border



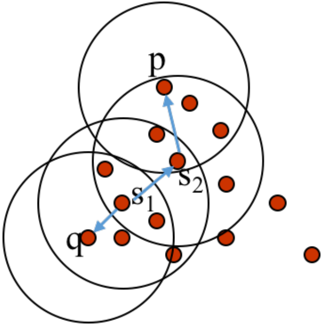
Density Reachability

- a point p is density reachable from point q iff
  - q is core
  - there is a sequence of points point q_i such that q_{i+1} is directly density reachable from q_i, $i \in [1, nq]$, q_1 is directly reachable from q and p is directly density reachable from q_{nq}
- reachability is not symmetric
  - in the example q is not density reachable from p, since p is border



Density Connection

- a point p is density connected to point q iff there is a point s such that p and q are density reachable from s
- density connection is symmetric



Generation of clusters

- A cluster is a maximal set of points connected by density
- Border points which are not connected by density to any core point are labelled as noise

DBSCAN 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`.

DBSCAN 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.

DBSCAN 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.

DBSCAN 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`.

DBSCAN 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 ← `RegionQuery`(SetOfPoints, Point, Eps);

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`;

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*;

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');

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`

$Neighbors \leftarrow \emptyset$;

RegionQuery Subroutine

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

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

RegionQuery Subroutine

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

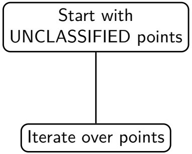
$Neighbors \leftarrow \emptyset$;
for each P' in SetOfPoints **do**
  if $\texttt{distance}(Point, P') \leqslant Eps$ **then**
    append P' to *Neighbors*;
return *Neighbors*;

DBSCAN: Full Algorithm Flow

Start with UNCLASSIFIED points

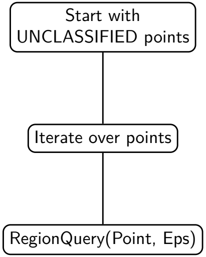
- Initialize all points as `UNCLASSIFIED`.

DBSCAN: Full Algorithm Flow



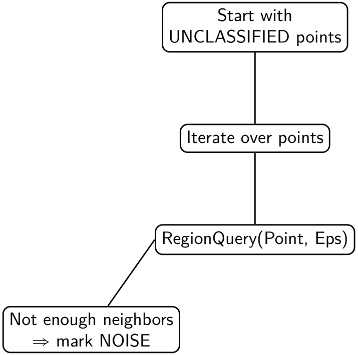
- Initialize all points as `UNCLASSIFIED`.
- Loop through the dataset.

DBSCAN: Full Algorithm Flow



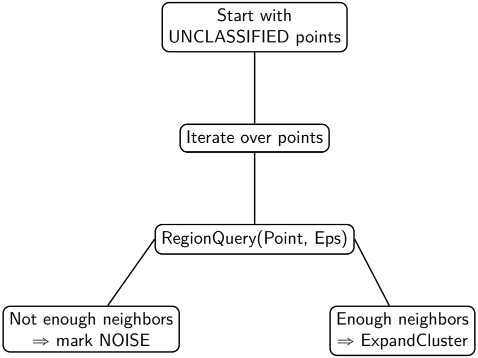
- Initialize all points as `UNCLASSIFIED`.
- Loop through the dataset.
- Perform RegionQuery for each unclassified point.

DBSCAN: Full Algorithm Flow



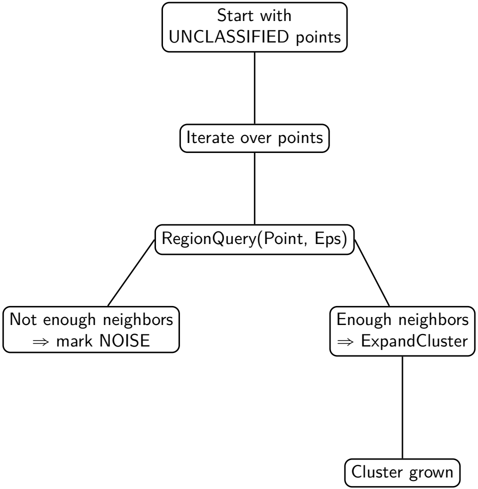
- Initialize all points as `UNCLASSIFIED`.
- Loop through the dataset.
- Perform RegionQuery for each unclassified point.
- If too few neighbors ? point becomes NOISE (or border).

DBSCAN: Full Algorithm Flow



- 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.

DBSCAN: Full Algorithm Flow



- 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.

How to set ϵ and $\mathrm{minPoints}$?

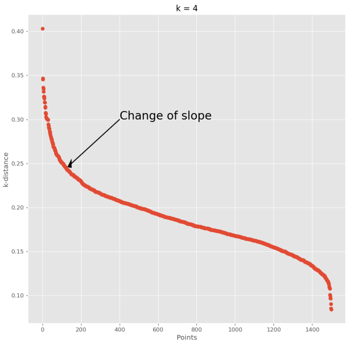
- As in many other machine learning algorithms, a grid search over several combination of hyperparameters can be useful
- As a rule of thumb, you can try $\mathrm{minPoints} = 2 * D$, the number of dimensions
- Noise suggest an increase in $\mathrm{minPoints}$
- A guess for ϵ requires more effort, considering the distance of the k–nearest neighbour, with $k = \mathrm{minPoints}$

Good guess for ϵ I

- Consider the vector of the k-distances
  - choose k
  - for each point we compute the distance of its k–nearest neighbour and we sort the points for decreasing k–distance
- Choosing a given k–distance as ϵ, it turns out that all the points with a k–distance bigger than ϵ will be considered as border
  - in the figure of next page they are the points to the left of the vertical of the chosen ϵ

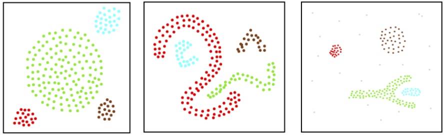
Good guess for ϵ II

- Usually, datasets which exhibit some tendency to clustering exhibit also a change of slope
- The best ϵ can be found with a grid search in the area of the change of slope
  - the figure refers to a dataset with 1500 points and with $\mathrm{minPoints}=4$
  - this figure suggests a fine tuning of ϵ in the interval 0.2–0.3



Comments

- ☺ Finds clusters of any shape
- ☺ Is robust w.r.t. noise
- ☹ Problems if clusters have widely varying densities
  - Being based on distances between points, the complexity is $\mathcal{O}(N^2)$
    - reduced to $\mathcal{O}(N\log(N))$ if spatial indexes, such as R*, are available
  - Very sensitive to the values of ϵ and $\mathrm{minPoints}$
  - Decreasing ϵ and increasing $\mathrm{minPoints}$ reduces the cluster size and increases the number of noise points



Model based (or statistic based) clustering[2]

- Estimate the parameters of a statistical model to maximize the ability of the model to explain the data
- The main technique is to use the mixture models
  - view the data as a set of observation from a mixture of different probability distributions
- Usually, the base model is a multivariate normal
  - well-known, easy to work with, good results
- The estimation is usually done using the maximum likelihood
  - given a set of data $\mathcal{X}$, the probability of the data, regarded as a function of the parameters, is called a likelihood function
- Attributes are assumed to be random independent variables

2 [Tan et al.(2006)Tan, Steinbach, and Kumar], Section 9.2.2

Gaussian Mixture a.k.a. Expectation Maximization – EM

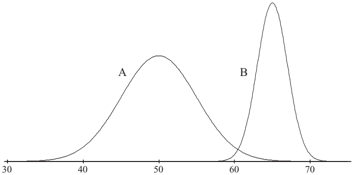
- If the data can be approximated by a single distribution, the derivation of the parameters is straightforward
- In the general case, with many mixed distributions, the EM algorithm is used

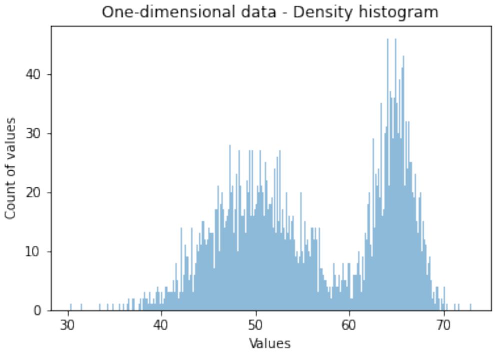
EM algorithm

1. Select an initial set of model parameters
2. repeat
   2.1 Expectation Step – For each object, calculate the probability that each object belongs to each distribution
   2.2 Maximization Step – Given the probabilities from the expectation step, find the new estimates of the parameters that maximize the expected likelihood
3. until – the parameters do not change (or the change is below a specified threshold)

One dimension mixture example

- Case with one dimension, two components
- Synthetic data randomly generated with two gaussians
  $\mu_A = 50, \sigma_A = 5, p_A = 0.6$
  $\mu_B = 65, \sigma_B = 2, p_B = 0.4$





EM – one dimension, two clusters example I

- Need to estimate 5 parameters
  - mean and standard deviation for cluster A
  - mean and standard deviation for cluster B
  - sampling probability p for cluster A

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

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

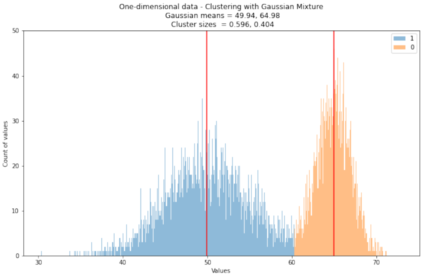
EM – one dimension, two clusters example II

- Repeat until convergence
  - Expectation: Compute p_A and p_B using the current distribution parameters
    - Compute the numerators for $\mathbf{Pr}(A \mid x)$ and $\mathbf{Pr}(B \mid x)$ and normalize dividing by their sum
  - Maximization of the distribution likelihood given the data
    - Compute the new distributions parameters, weighting the probabilities according to the current distribution parameters
- After convergence label each object with A or B according to the maximum probability, given the last distribution parameters

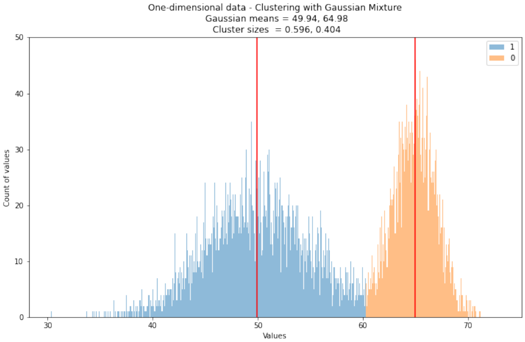
One dimension example - Gaussian Mixture result

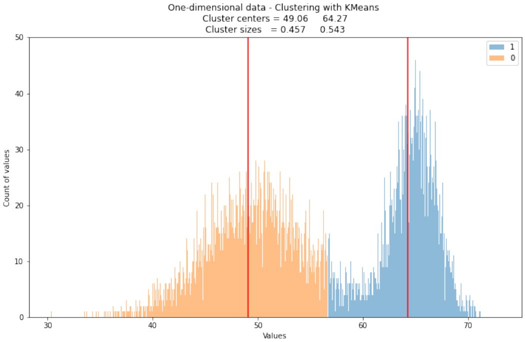
Estimated parameters of the distributions

	weight	mean	deviation
0	0.4037	64.978754	2.030968
1	0.5963	49.939990	4.999235



Gaussian Mixture and KMeans – Comparison of results





- These data have a bimodal, gaussian–like distribution
- The EM algorithm is founded on the hypothesis of modelling data with gaussians
- KMeans is non–parametric, and in this case the performance is worse

BIRCH Clustering: Overview

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

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

A CF-tree enables fast, memory-efficient clustering with a single scan of the data.

Key Idea: The CF Node

Each CF-tree node stores a set of Clustering Features that summarize a subcluster.

- A **CF** triple $\langle N, LS, SS \rangle$ compactly captures centroid and radius.
- Internal nodes group subclusters hierarchically.
- Leaf nodes contain the final subcluster summaries.

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

Algorithm Steps

The BIRCH algorithm proceeds in four conceptual phases:

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

Steps 3 and 4 are optional but improve accuracy.

Advantages of BIRCH

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

Limitation: BIRCH struggles with non-spherical or poorly separated clusters due to its reliance on centroid-based thresholding.

Spectral Clustering: Overview

Spectral clustering uses the eigenstructure of a similarity graph to partition data into clusters.
It transforms the 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

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:

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

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 a standard clustering algorithm (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

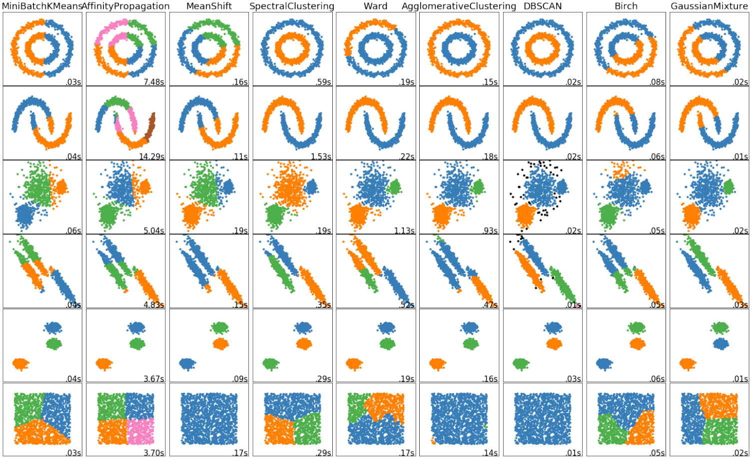
- Captures non-linear and arbitrary-shaped clusters.
- Works well when clusters are connected components in a graph.

Limitations

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

Comparison of results for selected algorithms

(from Scikit-Learn documentation)



A summary of selected clustering algorithms – I

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

A summary of selected clustering algorithms – II

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
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
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 clusterer.	Large n_clusters and n_samples	Large dataset, outlier removal, data reduction, inductive	Euclidean distance between points

Clustering types

- Partitioning
  - iteratively find partitions in the dataset, optimizing some quality criterion
- Hierarchic
  - recursively compute a structured hierarchy of subsets
- Density based
  - compute densities and aggregates clusters in high density areas
- Model based
  - assume a model for the distribution of the data and find the model parameters which guarantee the best fitting to the data

Clustering scalability

- Effectiveness decreases with
  - dimensionality D
  - noise level
- Computational cost increases with
  - dataset size N, at least linearly
  - dimensionality D

Research perspective

- From the past
  - well-known problem in statistics
  - recent research
    - machine learning
    - databases
    - visualization
- For the future
  - effective and efficient algorithms for big data clustering, with a large number of dimensions, noise requested scalability w.r.t.:
    - size (N)
    - dimensionality (D)
    - noise level
    - rate of new data arrivals

Uses of clustering – data comprehension

- Biology
  - Creation of taxonomies
  - Genetics
- Information Retrieval
  - Grouping documents
- Climatology
  - Repetition patterns
- Psychology and medicine
  - Identification of illness types in front of partial variation of evidence
- Business
  - Customer grouping

Uses of clustering – utilities

- Summarization
  - Reasoning with groups representatives instead of with the entire population
- Data compression
  - Reduce the amount of data
  - Find cluster prototypes and substitute data with the indexes of the prototypes
    - Vector quantization
- Find the nearest neighbours
  - Each object refers to his prototypes
  - Near neighbours refer to the same prototype

Bibliography I

- Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu.
  A density-based algorithm for discovering clusters in large spatial databases with noise.
  pages 226–231. AAAI Press, 1996.
- Pang-Nin Tan, Michael Steinbach, and Vipin Kumar.
  Data Mining.
  Addison Wesley, 2006.
  ISBN 0-321-32136-7.