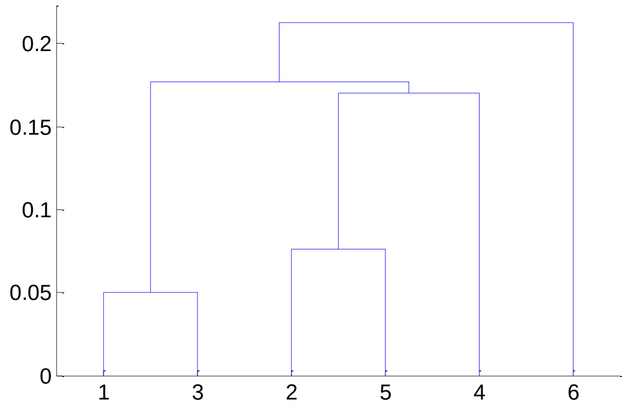
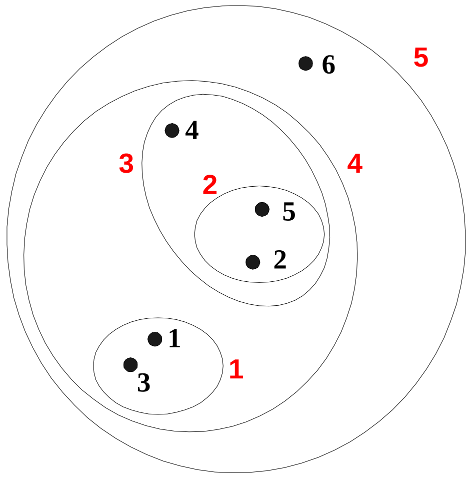
DATA MINING AND MACHINE LEARNING

Clustering Algorithms 3

Hierarchical Clustering

- Produces a set of nested clusters organized as a hierarchical tree
- Can be visualized as a dendrogram
  - *A tree like diagram that records the sequences of merges or splits*





Hierarchical Clustering

- Two main types of hierarchical clustering
  - *Agglomerative:*
    - Start with the points as individual clusters
    - At each step, merge the closest pair of clusters until only one cluster (or k clusters) left
  - *Divisive:*
    - Start with one, all-inclusive cluster
    - At each step, split a cluster until each cluster contains a point (or there are k clusters)
- Traditional hierarchical algorithms use a similarity or distance matrix
  - *Merge or split one cluster at a time*

Agglomerative Clustering Algorithm

- More popular hierarchical clustering technique

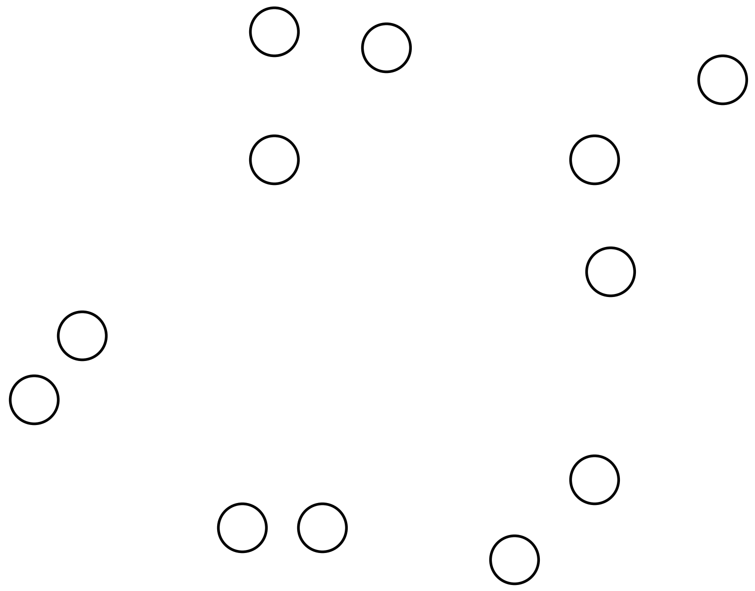
- Basic algorithm is straightforward

```
        Compute the proximity matrix
        Let each data point be a cluster
    Repeat
        Merge the two closest clusters
        Update the proximity matrix
    Until only a single cluster remains
```

- Key operation is the computation of the proximity of two clusters
  - *Different approaches to defining the distance between clusters distinguish the different algorithms*

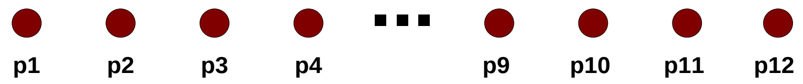
Starting Situation

- Start with clusters of individual points and a proximity matrix



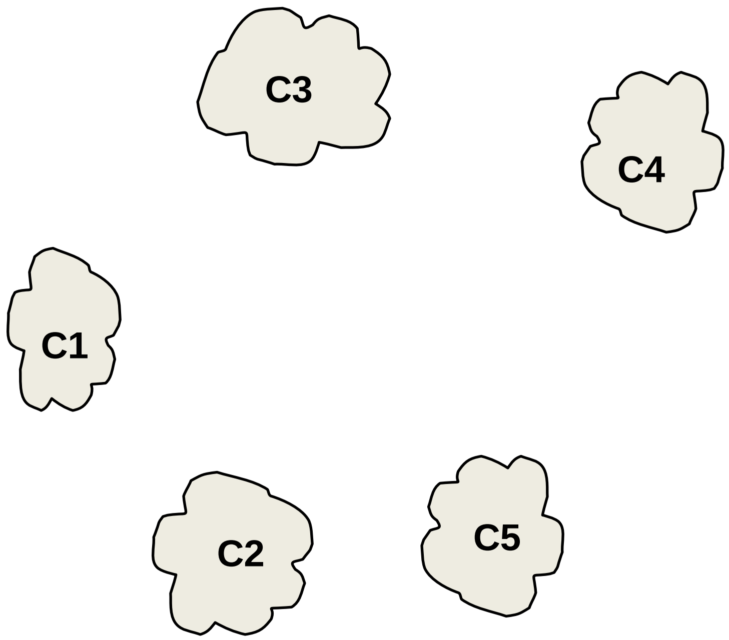
	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix



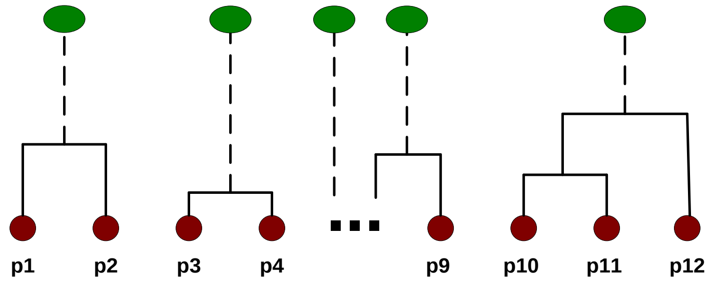
Intermediate Situation

- After some merging steps, we have some clusters



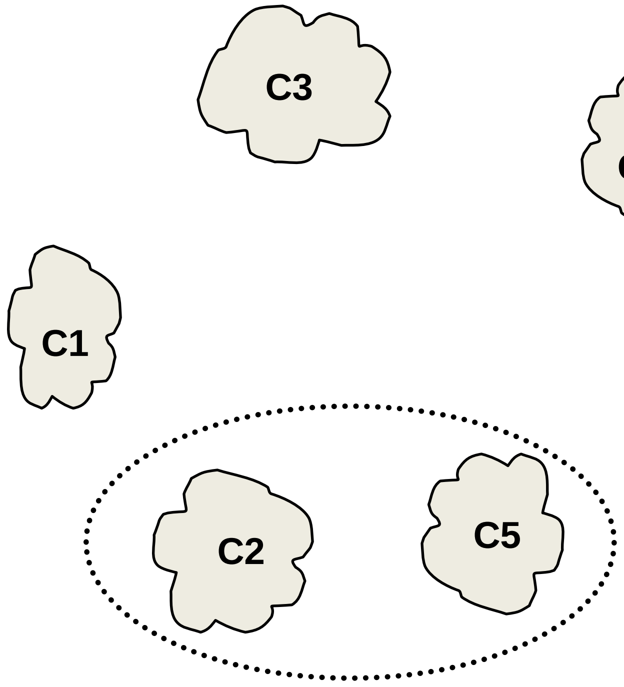
	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

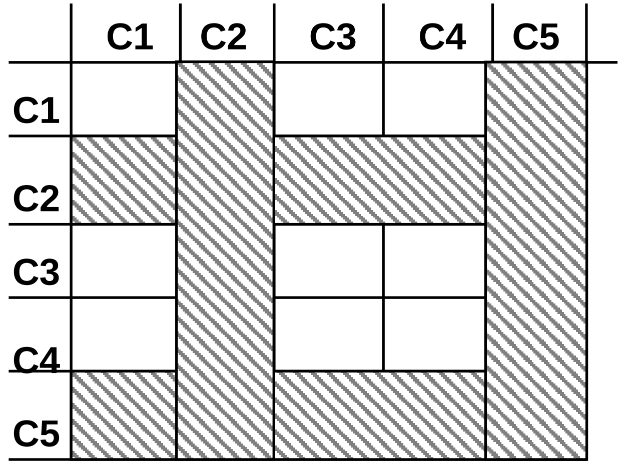
Proximity Matrix



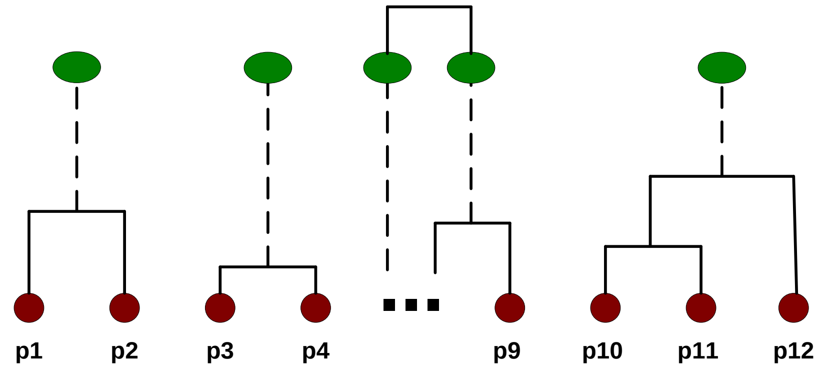
Intermediate Situation

- We want to merge the two closest clusters (C2 and C5) and update the proximity matrix.



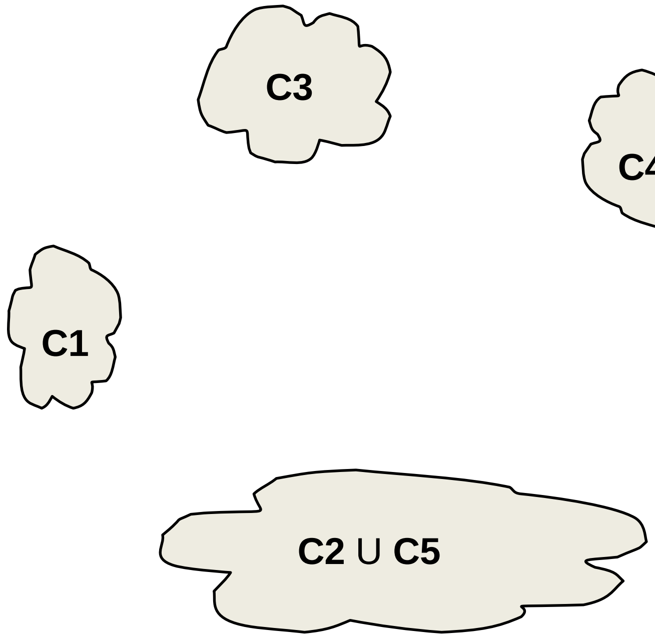


Proximity Matrix



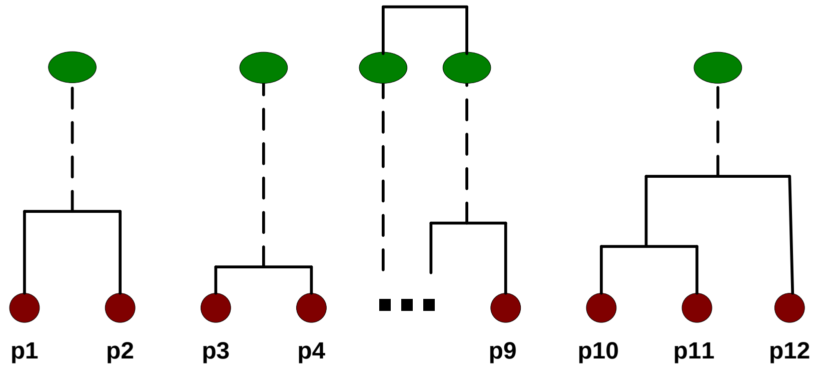
After Merging

- The question is “How do we update the proximity matrix?”

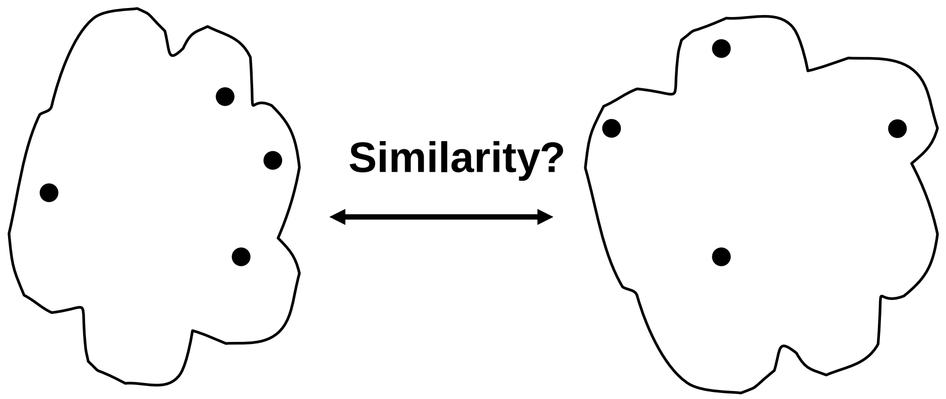


	C1	C2 U C5	C3	C4
C1		?		
C2 U C5	?		?	?
C3		?		
C4		?		

Proximity Matrix



How to Define Inter-Cluster Similarity

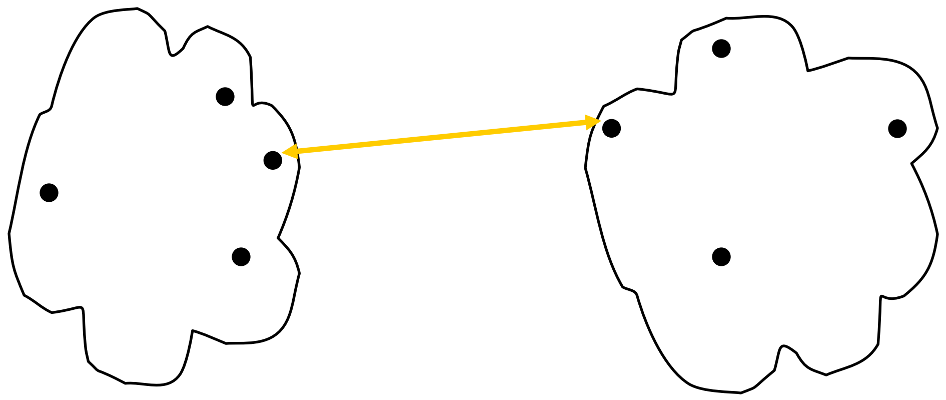


	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids

How to Define Inter-Cluster Similarity

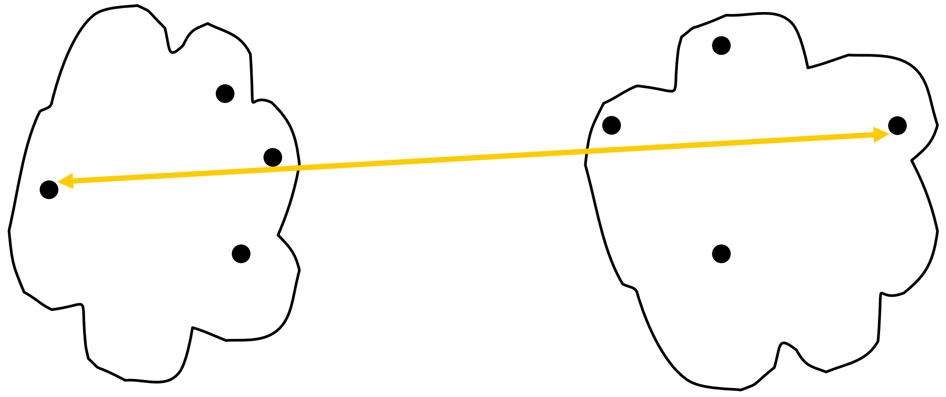


	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids

How to Define Inter-Cluster Similarity

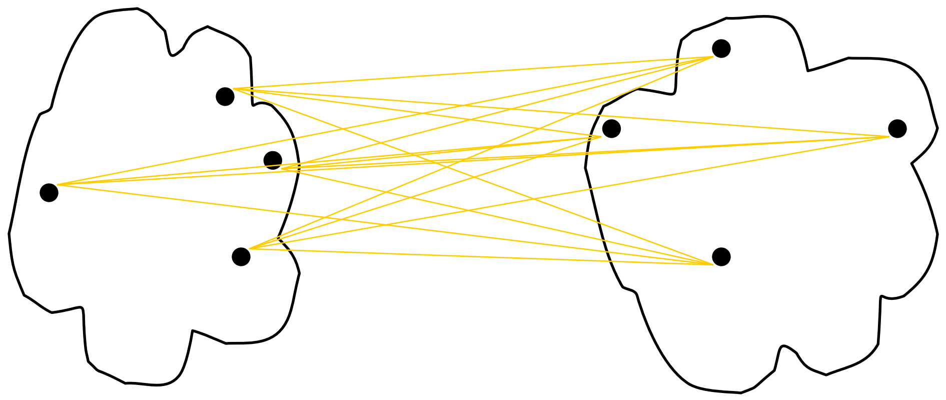


	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids

How to Define Inter-Cluster Similarity

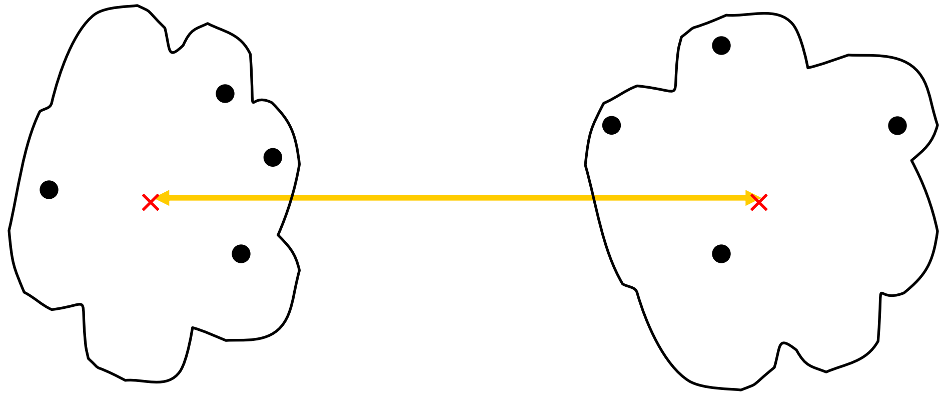


	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						
.						
.						

Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids

How to Define Inter-Cluster Similarity



	p1	p2	p3	p4	p5	. . .
p1						
p2						
p3						
p4						
p5						
.						

.

.

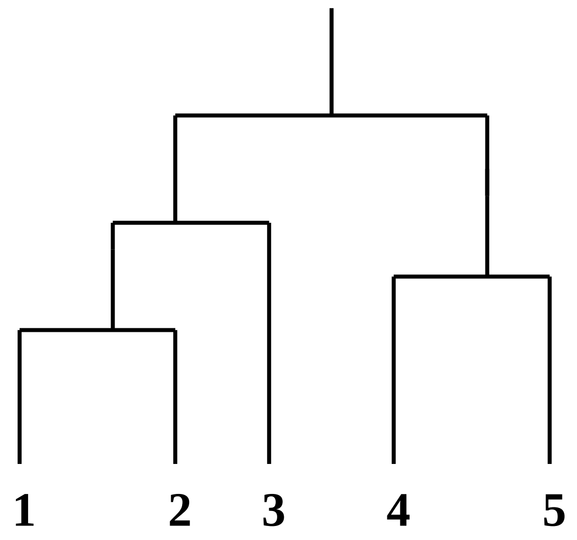
Proximity Matrix

- MIN
- MAX
- Group Average
- Distance Between Centroids

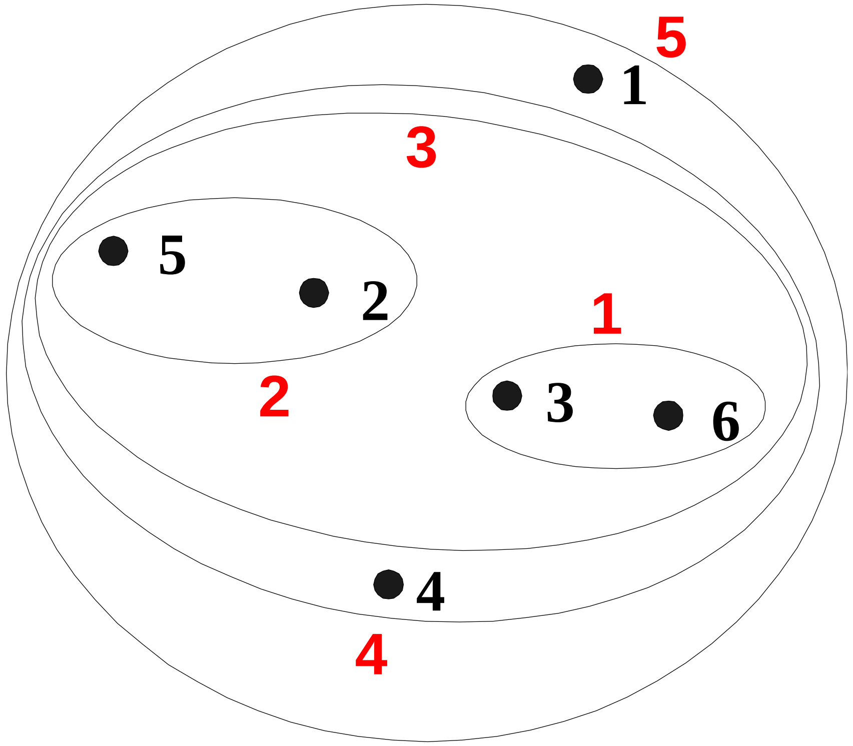
Cluster Similarity: MIN or Single Link

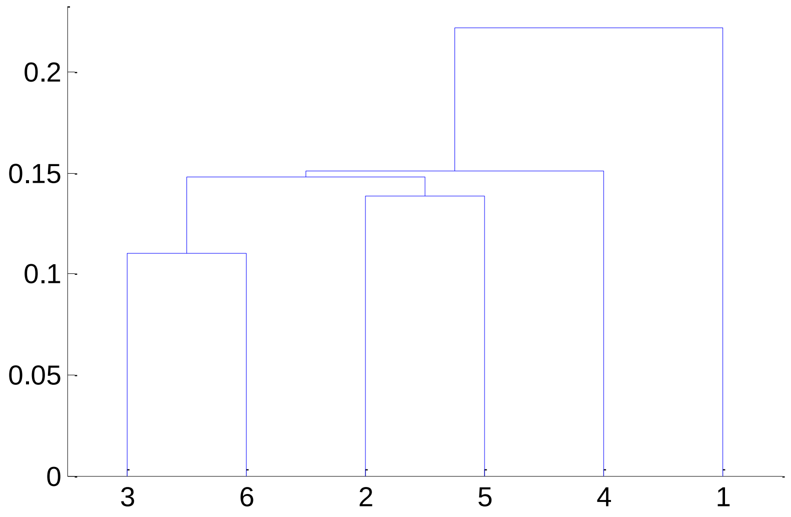
- Similarity of two clusters is based on the two most similar (closest) points in the different clusters
  - *Determined by one pair of points, i.e., by one link in the proximity graph.*

	I1	I2	I3	I4	I5
I1	1.00	0.90	0.10	0.65	0.20
I2	0.90	1.00	0.70	0.60	0.50
I3	0.10	0.70	1.00	0.40	0.30
I4	0.65	0.60	0.40	1.00	0.80
I5	0.20	0.50	0.30	0.80	1.00



Hierarchical Clustering: MIN



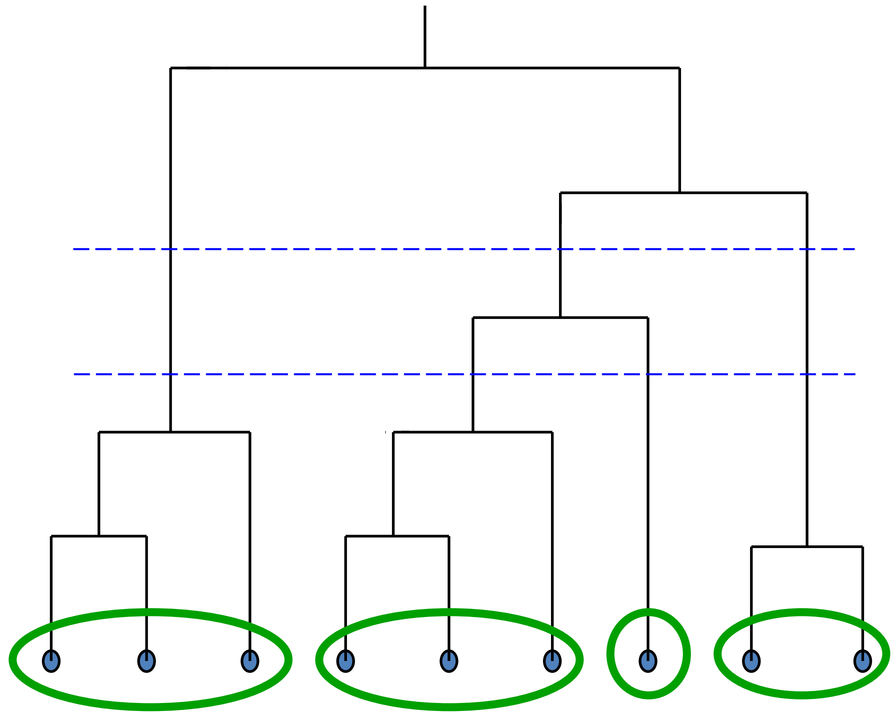


Nested Clusters

Dendrogram

Dendrogram: Hierarchical Clustering

- Clustering obtained by cutting the dendrogram at a desired level: each connected component forms a cluster.



Python support for Hierarchical Clustering

class sklearn.cluster.**AgglomerativeClustering**(*n_clusters=2, *, affinity='euclidean',memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None*)

Tuneable parameters are

- n_clusters – in hierarchical clustering need to specify a cut-of point
- linkage – how the distance between points is defined
- distance_threshold – neighboring clusters with greater value than this will not be merged; by controlling this you effectively set the height of the tree

Strengths of Hierarchical Clustering

- Do not have to assume any particular number of clusters
  - *Any desired number of clusters can be obtained by ‘cutting’ the dendogram at the proper level*

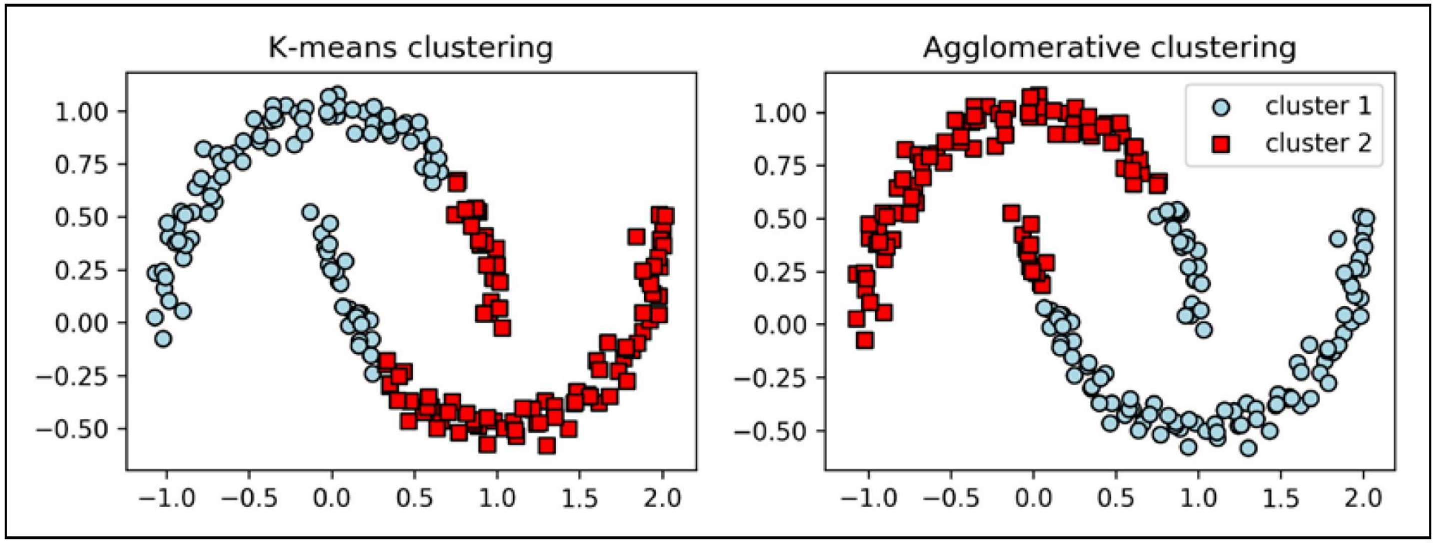
- They may correspond to meaningful taxonomies
  - *Example in biological sciences (e.g., animal kingdom, phylogeny reconstruction, …)*

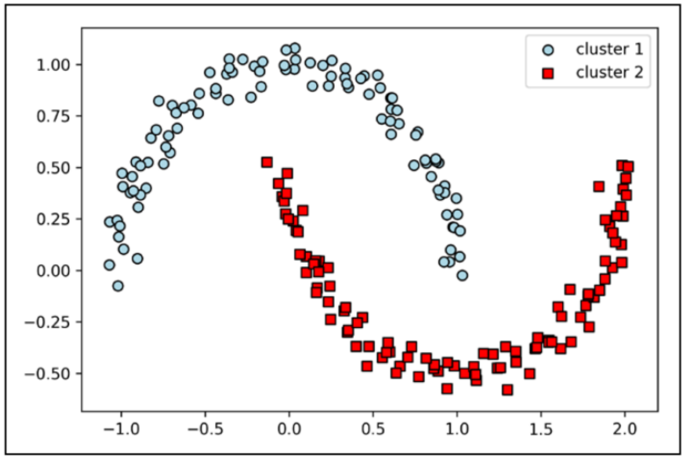
Density-Based Clustering Methods

- Clustering based on density (local cluster criterion), such as density-connected points or based on an explicitly constructed density function
- Major features:
  - *Discover clusters of arbitrary shape*
  - *Handle noise*
  - *One scan*
  - *Need density parameters*
- Several interesting studies:
  - *DBSCAN: Ester, et al. (KDD'96)*
  - *DENCLUE: Hinneburg & D. Keim (KDD'98/2006)*
  - *OPTICS: Ankerst, et al (SIGMOD'99).*
  - *CLIQUE: Agrawal, et al. (SIGMOD'98)*

Density Based Clustering

- DBSCAN (Density Based Spatial Clustering of Applications with Noise) is a density based scheme that works well with non spherical clusters.





DBSCAN Algorithm

(http://www2.cs.uh.edu/~ceick/7363/Papers/dbscan.pdf)

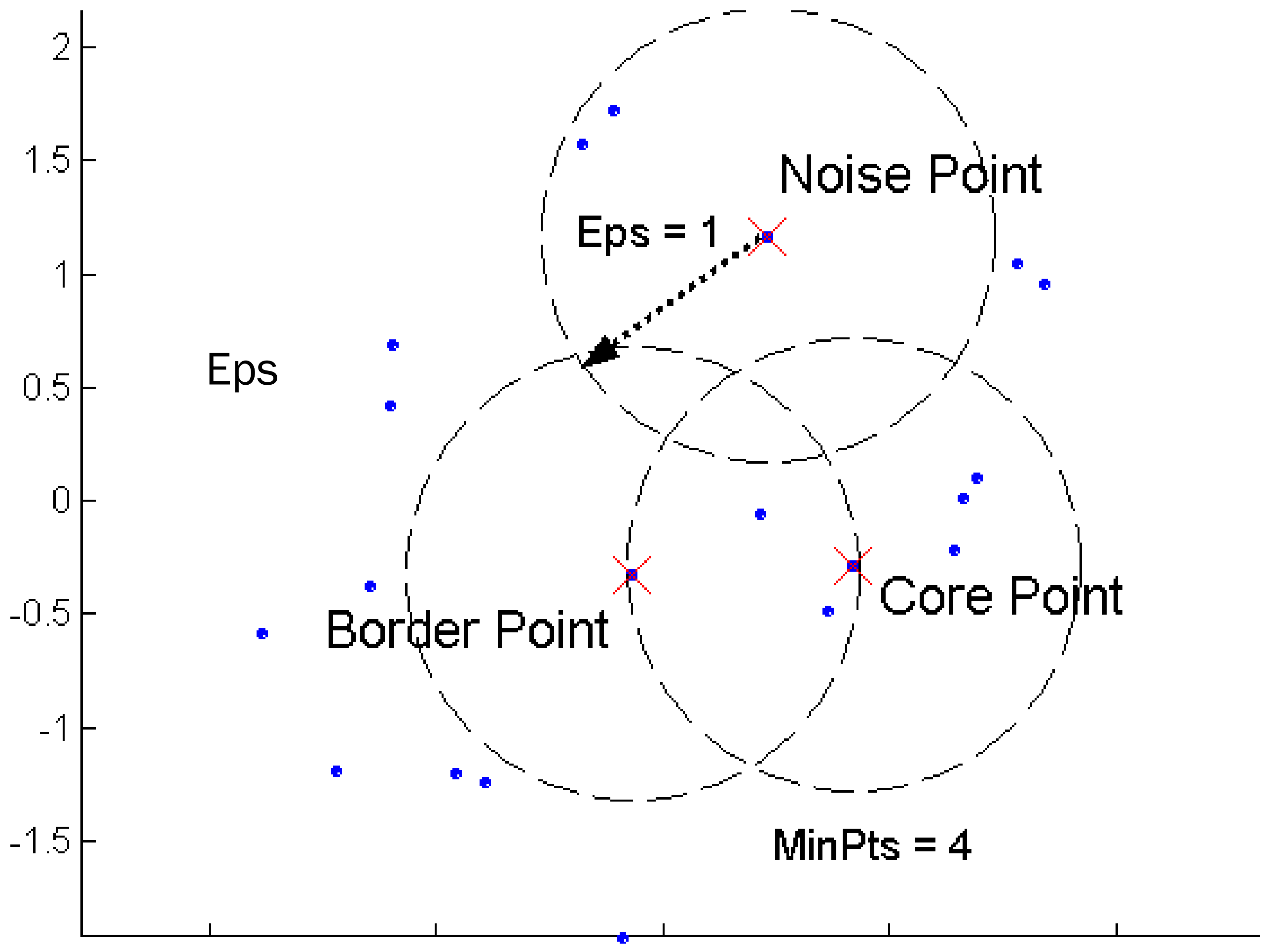
- Density = number of points within a specified radius r (Eps)

- A point is a core point if it has more than a specified number of points (MinPts) within Eps
  - *These are points that are at the interior of a cluster*

- A border point has fewer than MinPts within Eps, but is in the neighborhood of a core point
  - *Greater MinPts and smaller r => Higher Density*

- A noise point is any point that is not a core point or a border point.

DBSCAN: Core, Border, and Noise Points

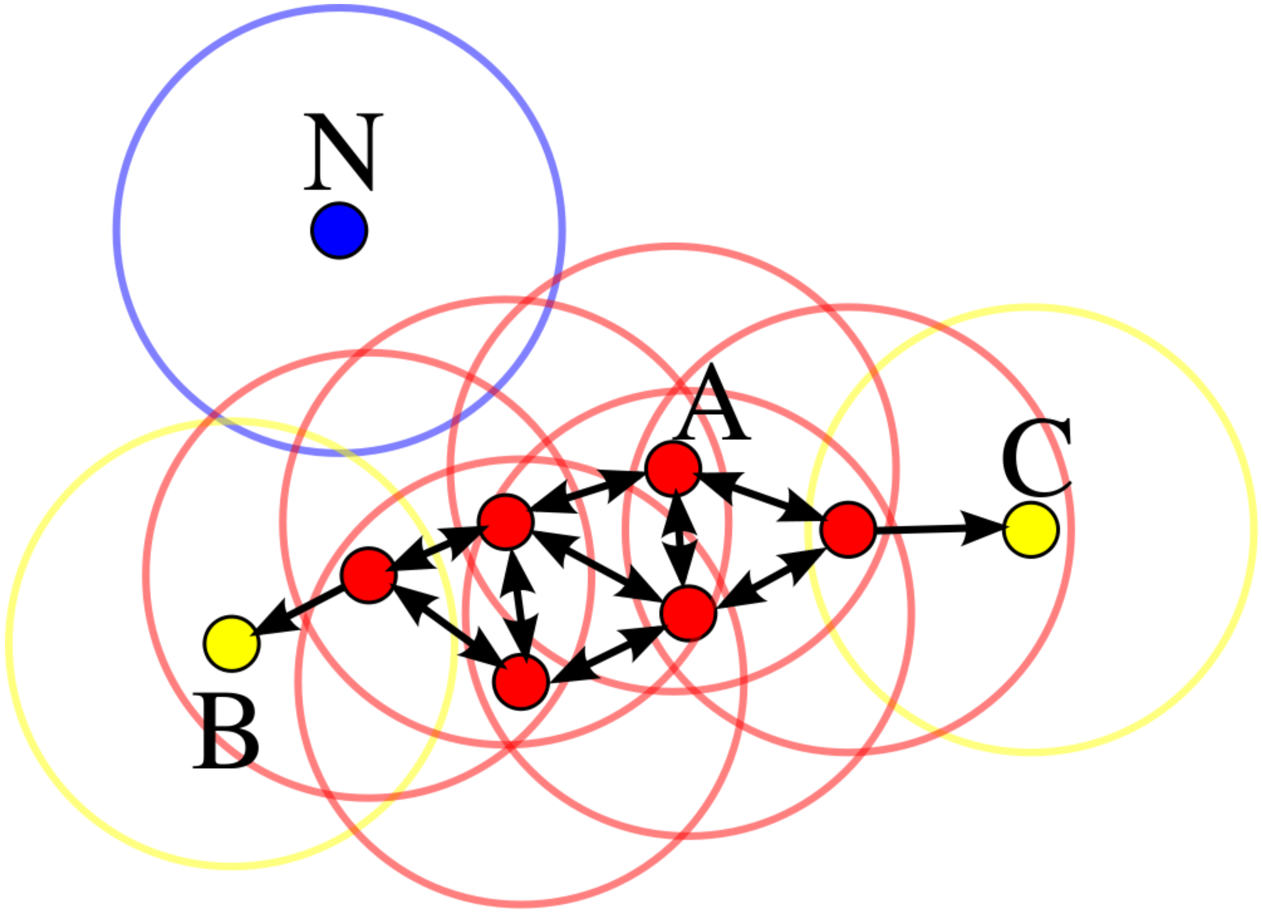


DBSCA

DBSCAN uses these two pamperers (Eps and MinPts) to divide the sample space into 3 different categories:

- Core points: a sample s is a core point IF at least MinPts points are within distance Eps of it (including s).
- Reachable (border) points: a sample t is reachable from s IF point t is within distance Eps from core point s.
- Noise points: All samples not reachable from any other sample is said to be noise points.

DBSCAN



This image extracted from:
Schubert, Erich; Sander, Jörg; Ester, Martin; Kriegel, Hans Peter; Xu, Xiaowei (July 2017). "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN". *ACM Trans. Database Syst*. **42** (3): 19:1–19:21.

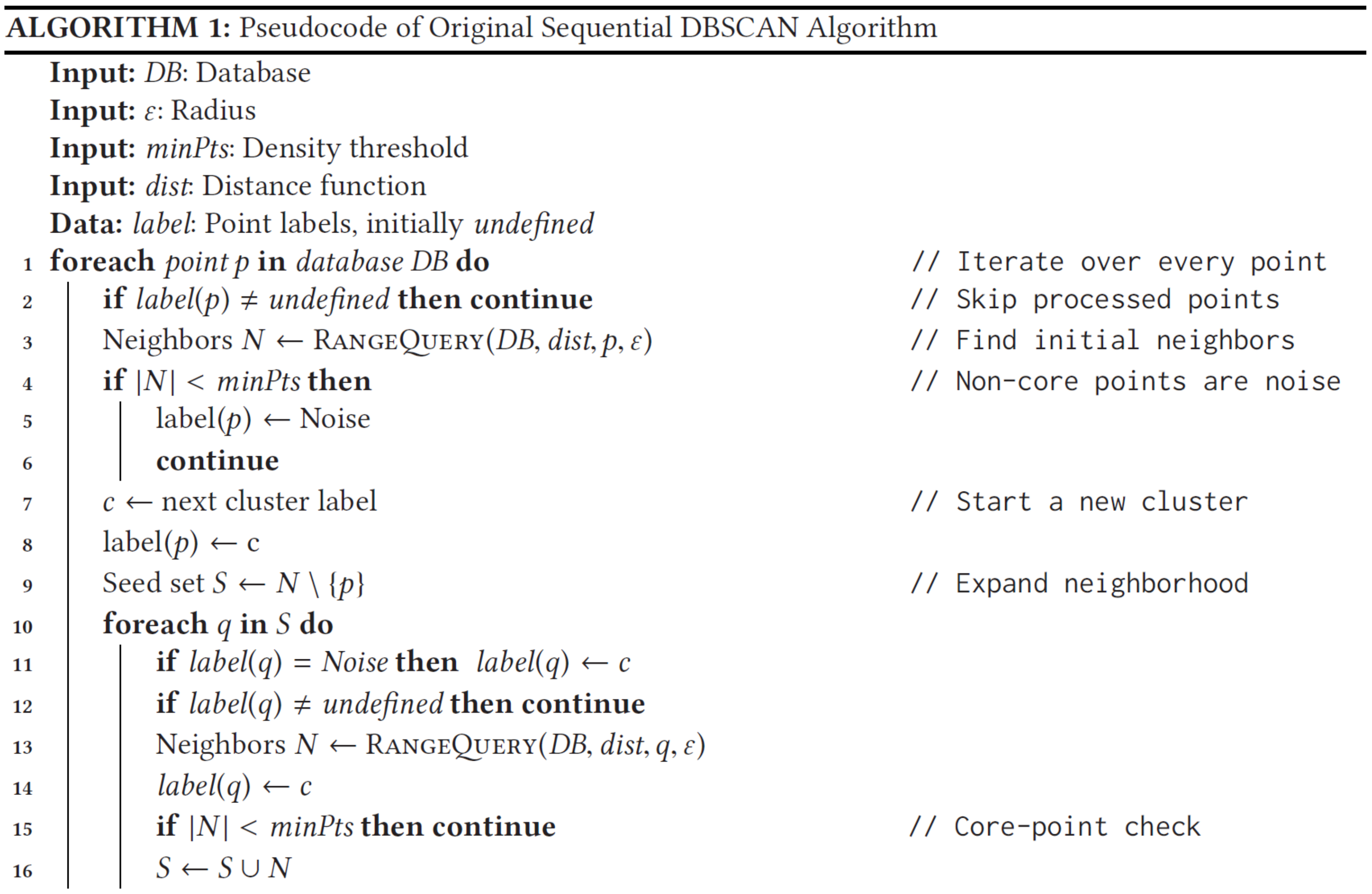
DBSCAN Demo

https://www.youtube.com/watch?v=h53WMIImUuc

Outline Algorithm

1. Create a cluster (C) for each set of core points (S)
2. Include in C any points reachable from any core point in set S.

DBSCAN: Detailed Algorithm



ALGORITHM 1: Pseudocode of Original Sequential DBSCAN Algorithm

```
Input: $DB$: Database
Input: $\varepsilon$: Radius
Input: $minPts$: Density threshold
Input: $dist$: Distance function
Data: $label$: Point labels, initially $undefined$
foreach point $p$ in database $DB$ do                          // Iterate over every point
    if $label(p) \neq undefined$ then continue                 // Skip processed points
    Neighbors $N \leftarrow$ RangeQuery($DB, dist, p, \varepsilon$)   // Find initial neighbors
    if $|N| < minPts$ then                                     // Non-core points are noise
        label($p$) $\leftarrow$ Noise
        continue
    $c \leftarrow$ next cluster label                          // Start a new cluster
    label($p$) $\leftarrow$ c
    Seed set $S \leftarrow N \setminus \{p\}$                  // Expand neighborhood
    foreach $q$ in $S$ do
        if $label(q) = Noise$ then $label(q) \leftarrow c$
        if $label(q) \neq undefined$ then continue
        Neighbors $N \leftarrow$ RangeQuery($DB, dist, q, \varepsilon$)
        $label(q) \leftarrow c$
        if $|N| < minPts$ then continue                        // Core-point check
        $S \leftarrow S \cup N$
```

- Above code taken from Schubert et al.

DBSCAN Algorithm (simplified)

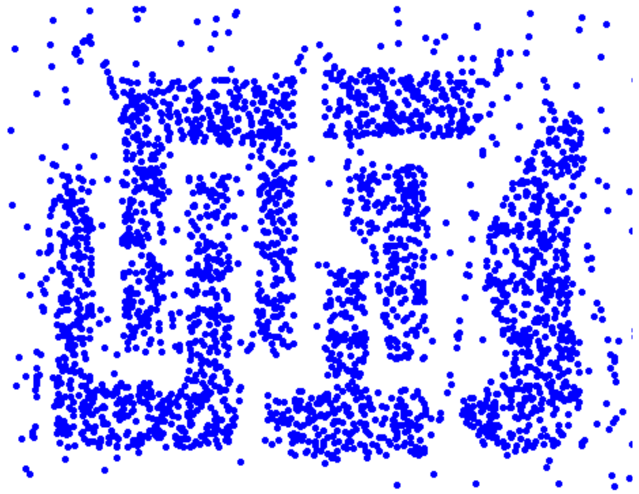
1. Create a graph whose nodes are the points to be clustered
2. For each core-point c create an edge from c to every point p in the ε-neighborhood of c
3. Set N to the nodes of the graph;
4. If N does not contain any core points terminate
5. Pick a core point c in N
6. Let X be the set of nodes that can be reached from c by going forward;
    1. *create a cluster containing* $X \cup \{c\}$
    2. $N = N / (X \cup \{c\})$
7. Continue with step 4

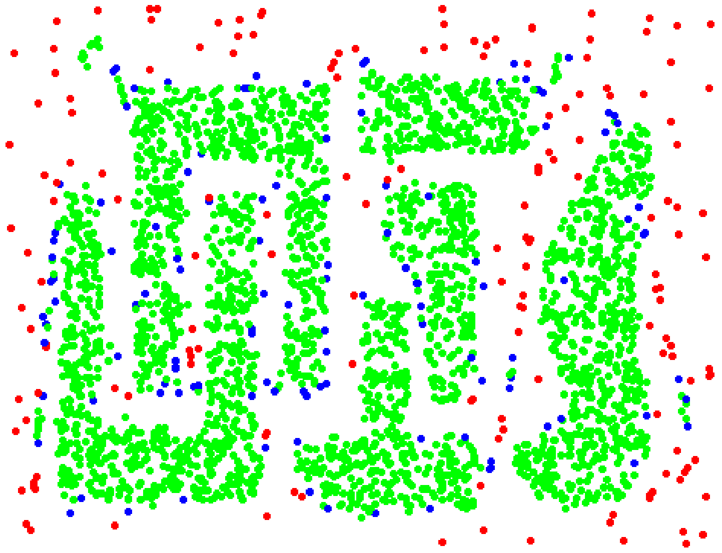
Note: points that are not assigned to any cluster are outliers;

Parameter estimation

- We first try to establish a value for *min_pts* and then set a value for ∈
- As a rule of thumb we set $\min\_pts \geq 2 \times D$ where D is the number of dimensions
- For datasets which are large or are noisy values of $min\_pts > 2 \times D$ are appropriate
- Now use the *min_pts* parameter and vary the values of ∈ until the silhouette measure is maximized.

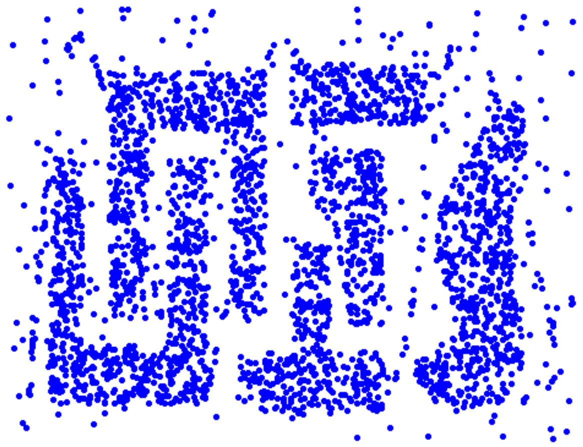
DBSCAN: Core, Border and Noise Points

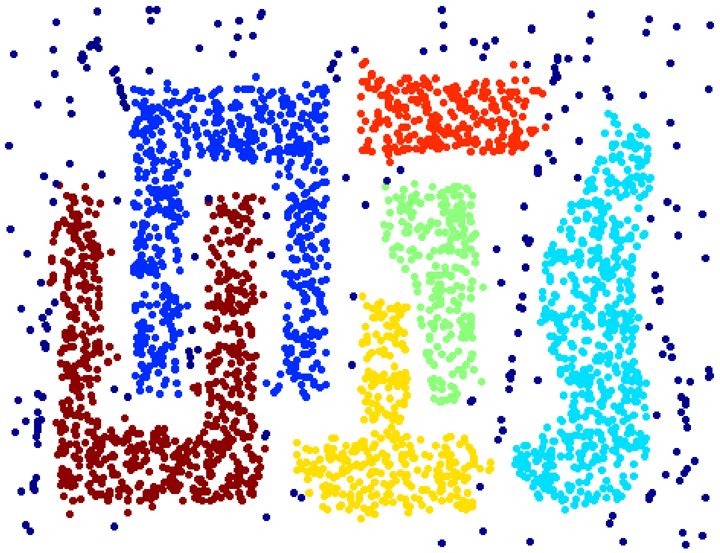

Original Points


Point types: core, border and noise

Eps = 10, MinPts = 4

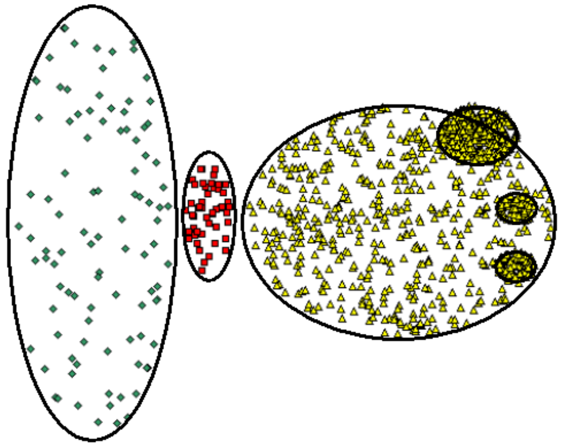
When DBSCAN Works Well


Original Points

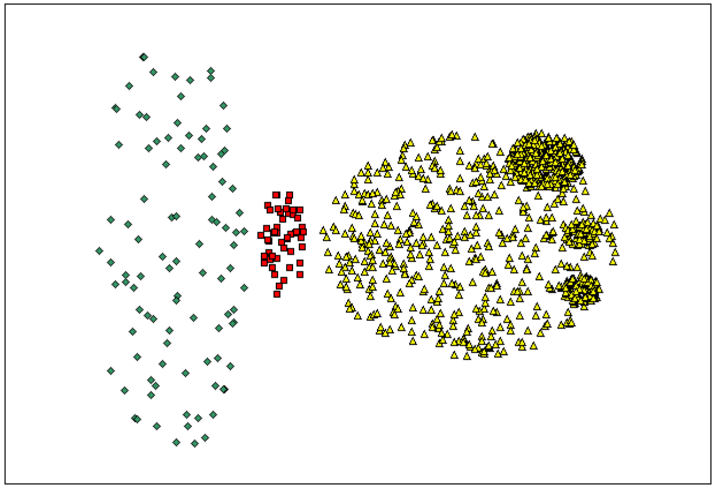

Clusters

- Resistant to Noise
- Can handle clusters of different shapes and sizes

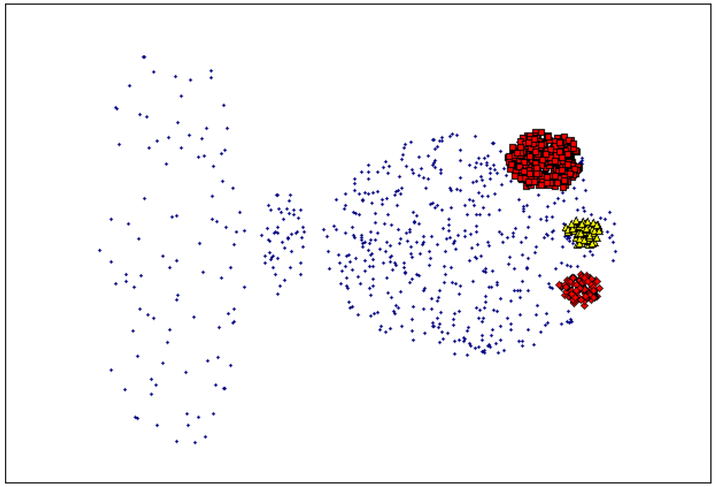
When DBSCAN Does NOT Work Well



Original Points



(MinPts=4, Eps=9.75).



(MinPts=4, Eps=9.92)

- Varying densities
- High-dimensional data

Advantages

- Automatically finds the number of clusters once the $\in$ and min_pts parameters are specified
- Can capture non spherical clusters
- Can capture noise points in dataset

Disadvantages

- Can be slow for large datasets as complexity is $O(n^2)$
- Can produce poor quality solutions when data contains clusters of widely different density
- High dimensional datasets can also result in poor clustering as distances of points to clusters get distorted in space (affects other clustering algorithms as well)

Python support for DBSCAN

- Supported by sklearn.dbscan

class sklearn.cluster.**DBSCAN**(*eps=0.5, *, min_samples=5, metric='euclidean', metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None*)

- Tuneable parameters are eps and min_samples (min_pts)
- Returns *labels*, a one dimensional array of cluster indexes that samples belongs to; noisy samples are flagged by -1

Other Clustering Algorithms

Many other widely used algorithms exist such as:

- SOM (Self Organizing Map)
- EM (Expectation Maximization)
- BIRCH(Balanced Iterative Reducing and Clustering)
- Cascade K means
- etc.