DATA MINING 1

Density-based Clustering

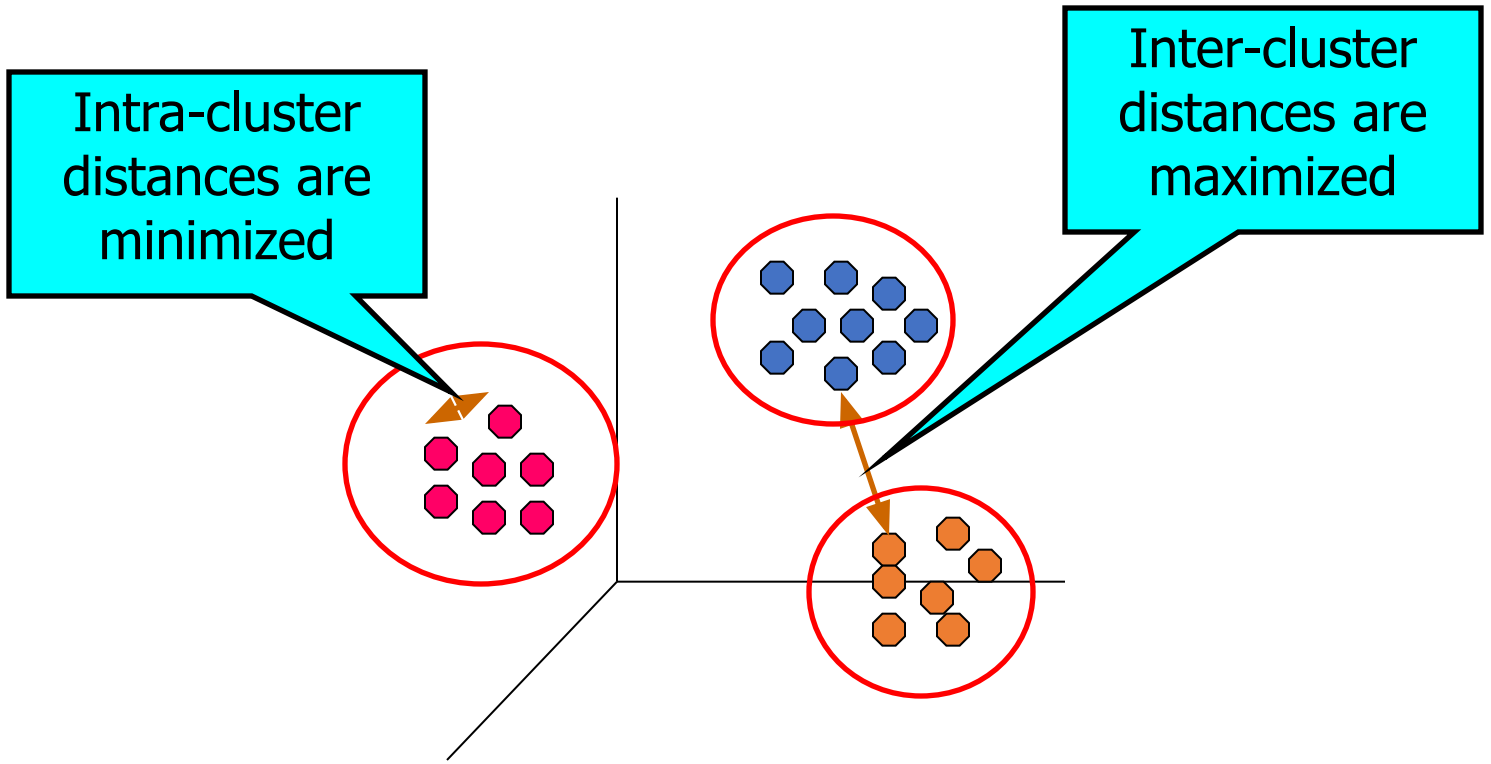
Riccardo Guidotti

Revisited slides from Lecture Notes for Chapter 7 “Introduction to Data Mining”, 2nd Edition by Tan, Steinbach, Karpatne, Kumar

UNIVERSITÀ DI PISA

What is Cluster Analysis?

- Finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups



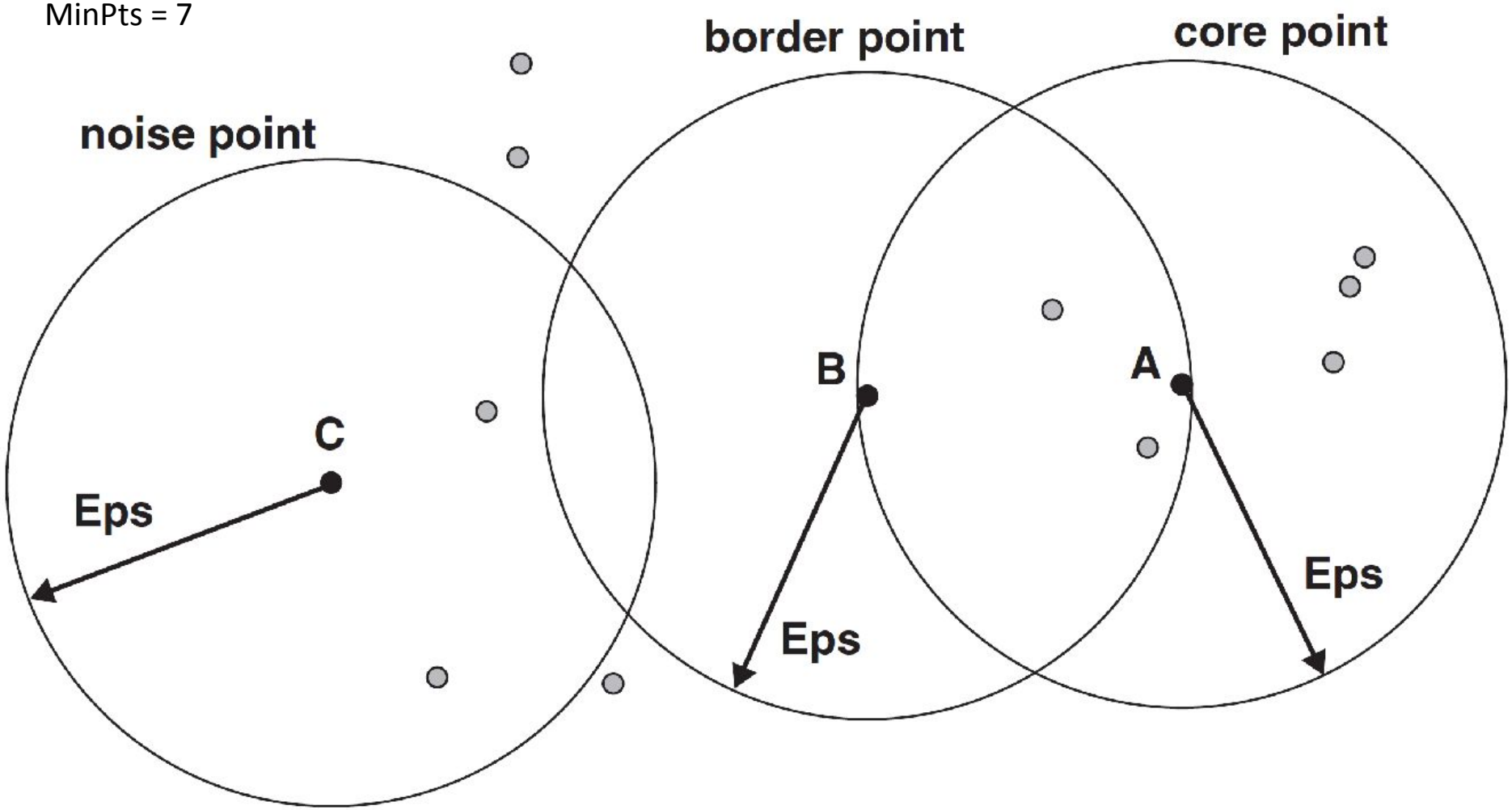
DBSCAN

DBSCAN

- DBSCAN is a density-based algorithm.
  - Density = number of points within a specified radius (Eps)
  - A point is a core point if it has at least a specified number of points (MinPts) within Eps
    - These are points that are at the interior of a cluster
    - Counts the point itself
  - A border point is not a core point, but is in the neighborhood of a core point
  - A noise point is any point that is not a core point or a border point

DBSCAN: Core, Border, and Noise Points

MinPts = 7

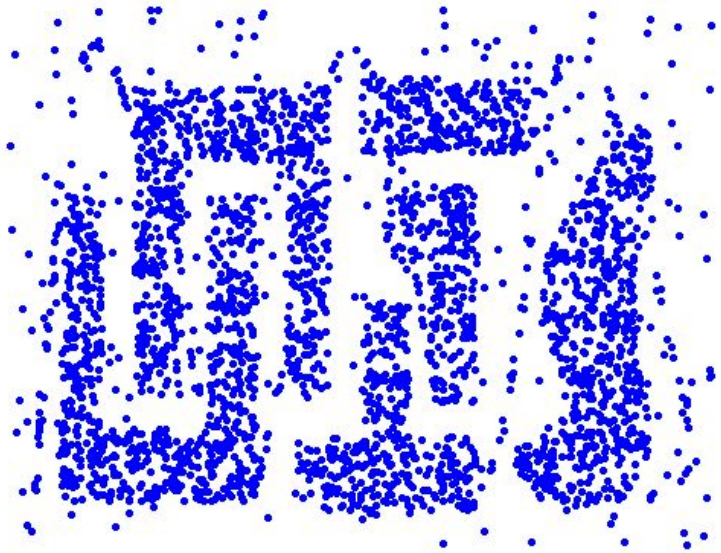


DBSCAN Algorithm

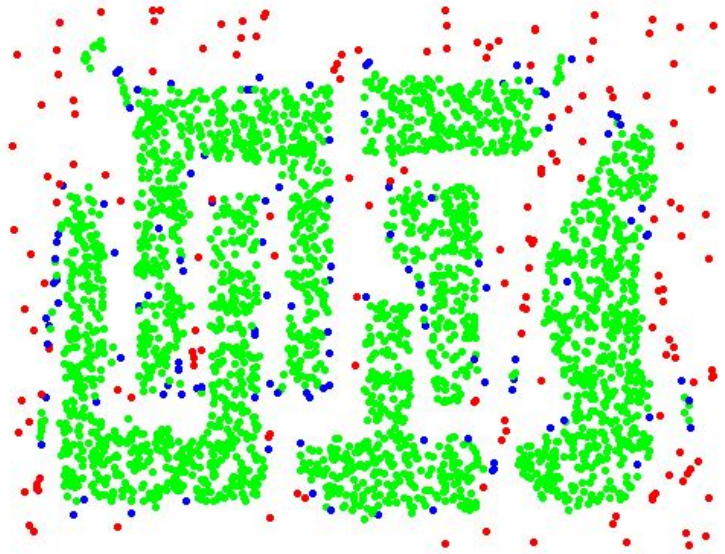
- Eliminate noise points
- Perform clustering on the remaining points

current_cluster_label $\leftarrow 1$
for all core points **do**
  if the core point has no cluster label **then**
    current_cluster_label $\leftarrow$ *current_cluster_label* $+ 1$
    Label the current core point with cluster label *current_cluster_label*
  end if
  for all points in the *Eps*-neighborhood, except i^{th} the point itself **do**
    if the point does not have a cluster label **then**
      Label the point with cluster label *current_cluster_label*
    end if
  end for
end for

DBSCAN: Core, Border and Noise Points



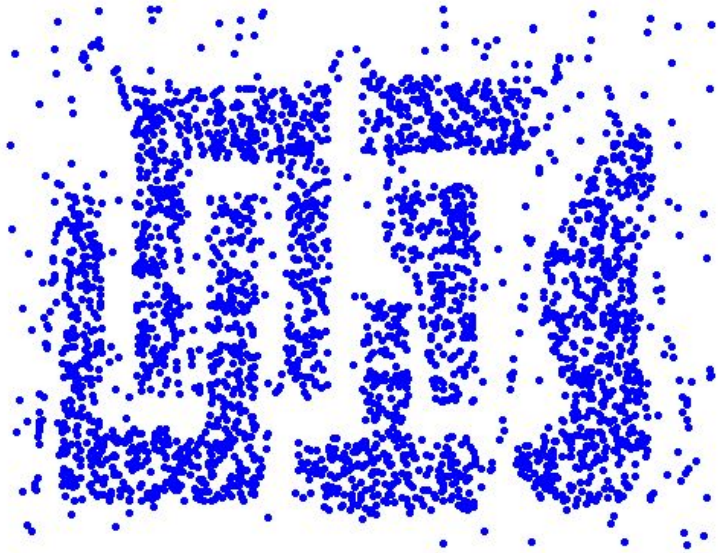
Original Points



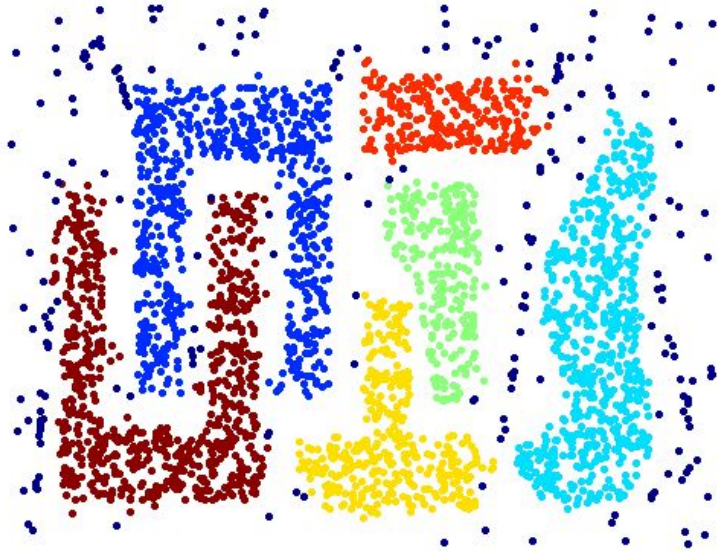
Point types: core, border and noise

Eps = 10, MinPts = 4

When DBSCAN Works Well



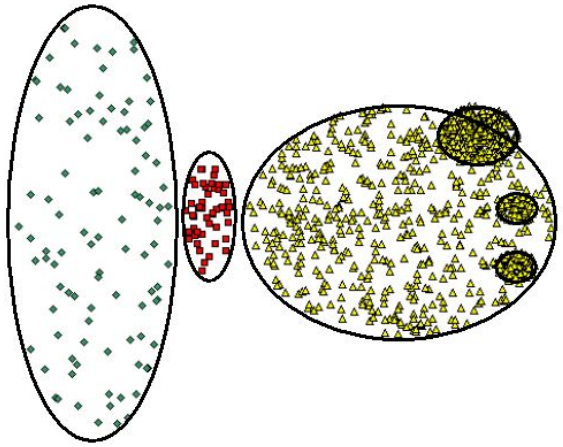
Original Points



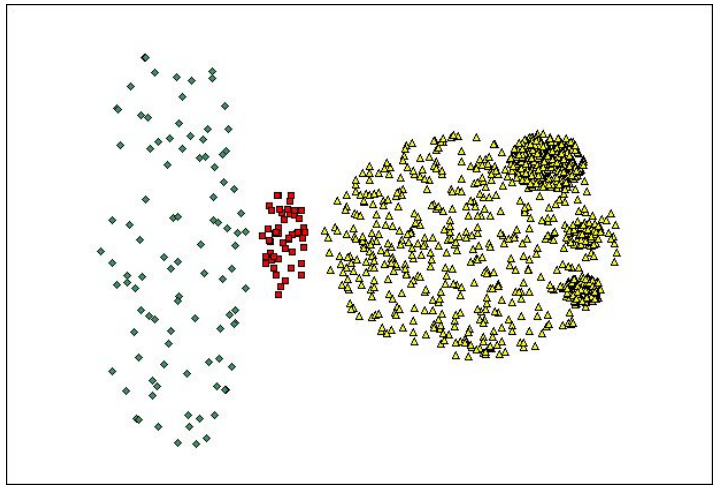
Clusters

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

When DBSCAN Does NOT Work Well

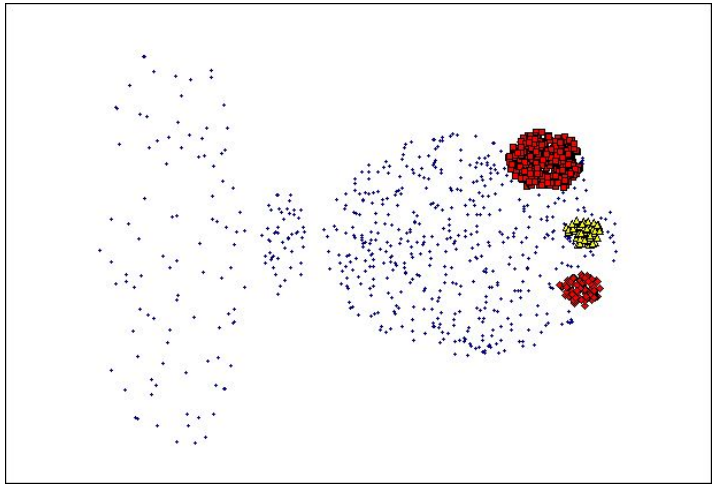


Original Points



(MinPts=4, Eps=9.75).

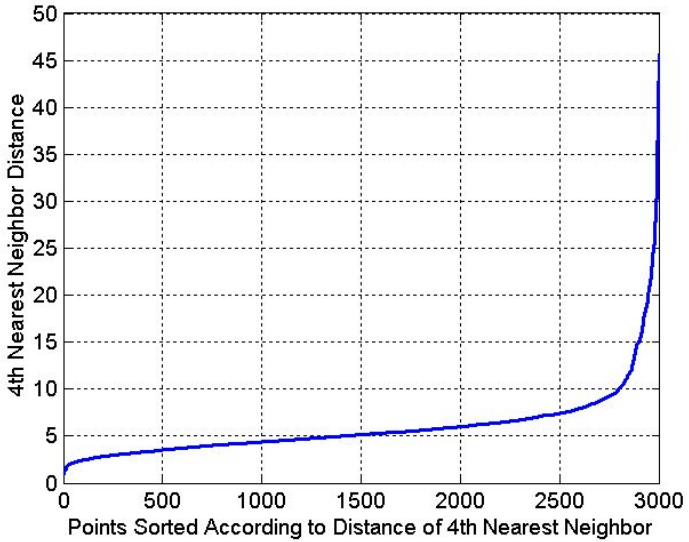
- **Varying densities**
- **High-dimensional data**



(MinPts=4, Eps=9.92)

DBSCAN: Determining EPS and MinPts

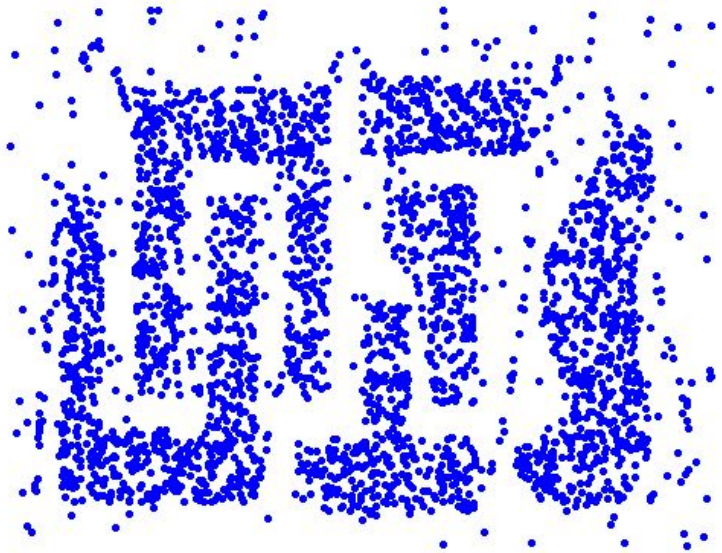
- Idea is that for points in a cluster, their k^{th} nearest neighbors are at roughly the same distance
- Noise points have the k^{th} nearest neighbor at farther distance
- So, plot sorted distance of every point to its k^{th} nearest neighbor



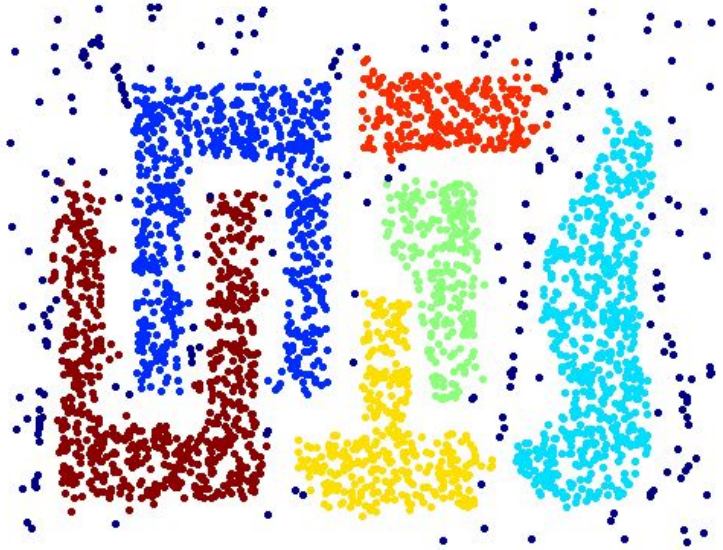
DBSCAN Evolution

OPTICS

When DBSCAN Works Well



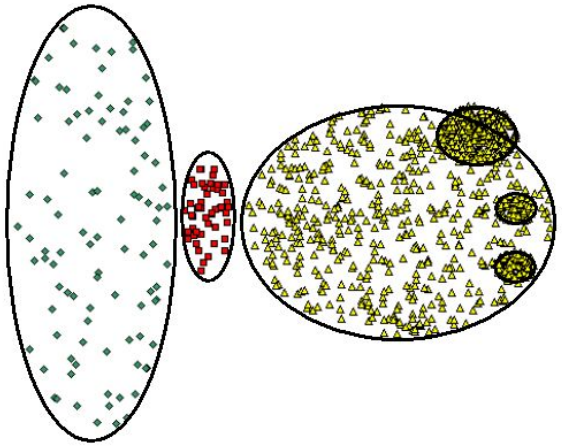
Original Points



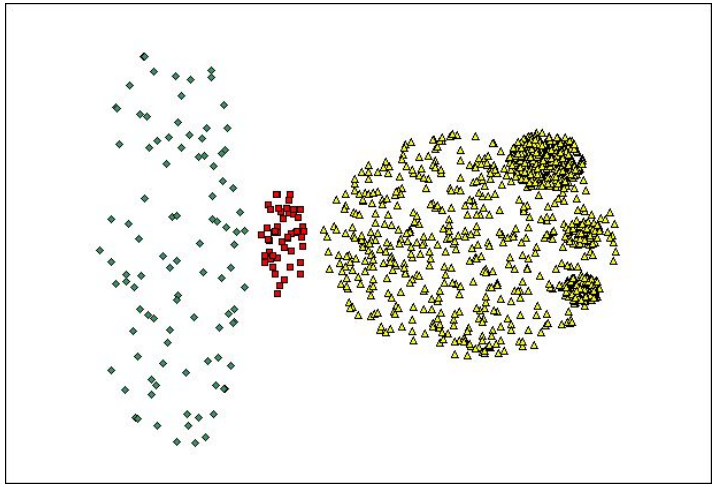
Clusters

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

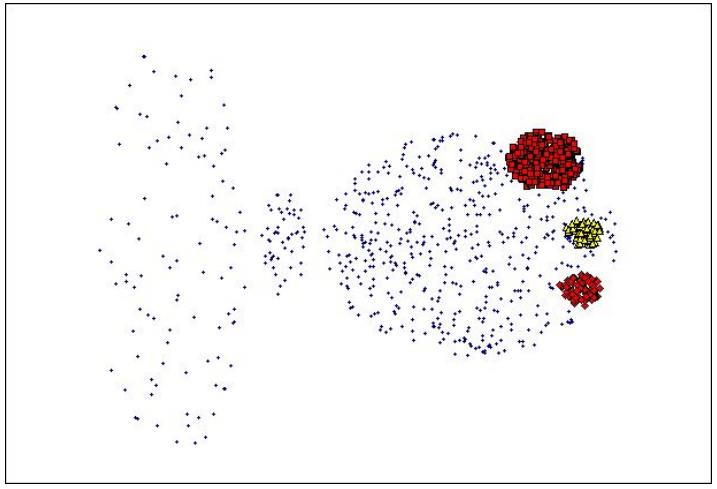
When DBSCAN Does NOT Work Well



Original Points



(MinPts=4, Eps=9.92).



(MinPts=4, Eps=9.75)

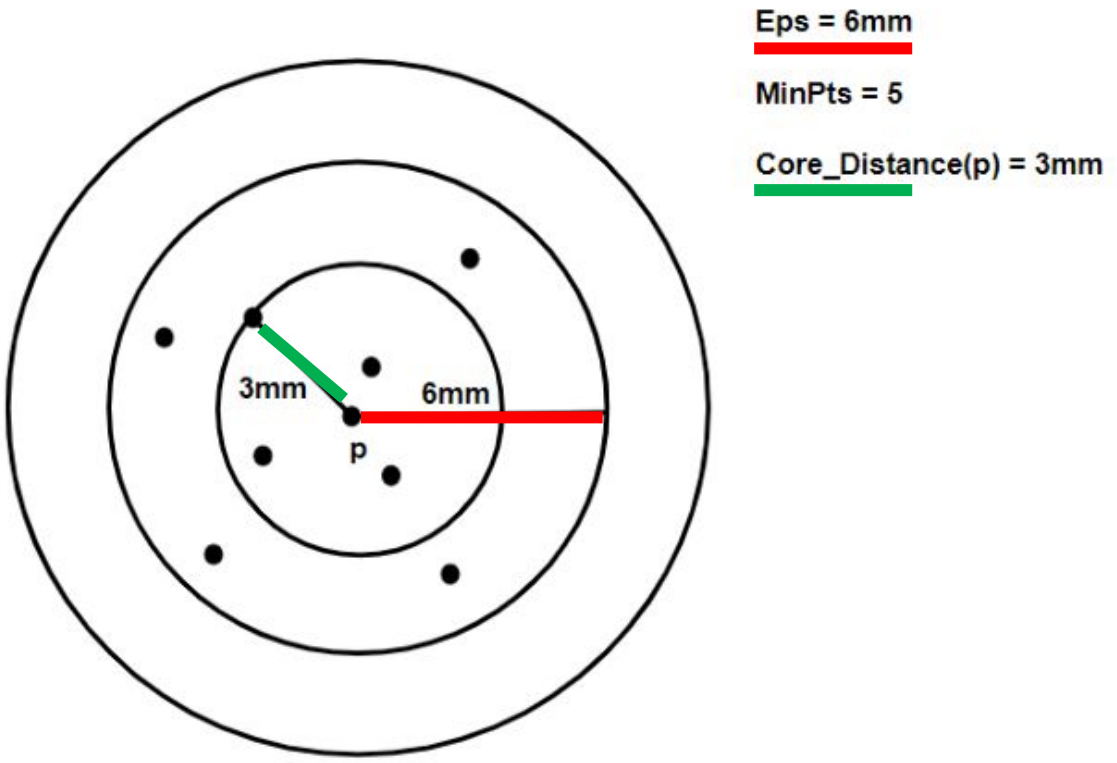
- **Varying densities**
- **High-dimensional data**

OPTICS

- OPTICS: Ordering Points To Identify the Clustering Structure
  - Produces a special order of the dataset wrt its density-based clustering structure.
  - This cluster-ordering contains info equivalent to the density-based clusterings corresponding to a broad range of parameter settings.
  - Good for both automatic and interactive cluster analysis, including finding intrinsic clustering structure.
  - Can be represented graphically or using visualization techniques.

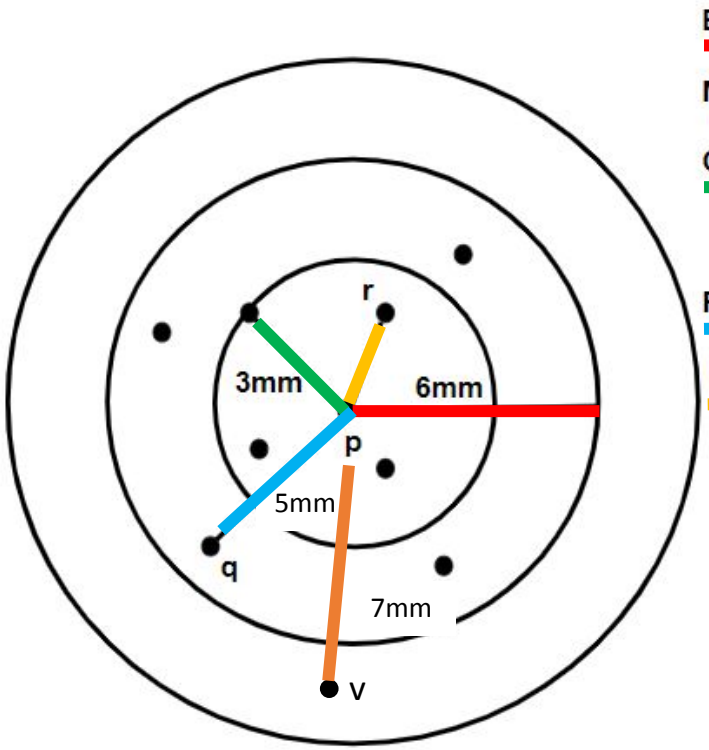
OPTICS: Extension from DBSCAN

- OPTICS requires two **parameters**:
  - ε, which describes the maximum distance (radius) to consider,
  - MinPts, describing the number of points required to form a cluster
- **Core point**. A point *p* is a core point if at least MinPts points are found within its ε-neighborhood.
- **Core Distance**. It is the **minimum** value of radius required to classify a given point as a core point. If the given point is not a Core point, then it's Core Distance is undefined.



OPTICS: Extension from DBSCAN

- **Reachability Distance**. The reachability distance between a point *p* and *q* is the **maximum** of the Core Distance of *p* and the Distance between p and q.
- The Reachability Distance is not defined if *q* is not a Core point. Below is the example of the Reachability Distance.
- In other words, if *q* is within the core distance of *p* then use the core distance, otherwise the real distance.



Eps = 6mm

MinPts = 5

Core_Distance(p) = 3mm

Reachability_Distance(q,p) = 5mm

Reachability_Distance(r,p) = 3mm

Reachability_Distance(v,p) = 7mm

OPTICS Pseudo-Code

- For each point *p* in the dataset
  - Initialize the reachability distance of *p* as undefined
- For each unprocessed point *p* in the dataset
  - Get the neighbors *N* of p
  - Mark *p* as processed and output to the *ordered list*
  - If *p* is a core point
    - Initialize a priority queue *Q* to get the closest point to *p* in terms of reachability
    - Call the function *update(N, p, Q)*
    - For each point *q* in *Q*
      - Get the neighbors *N'* of *q*
      - Mark q as processed and output to the *ordered list*
        - If *q* is a core point Call the function update(N', q, Q)

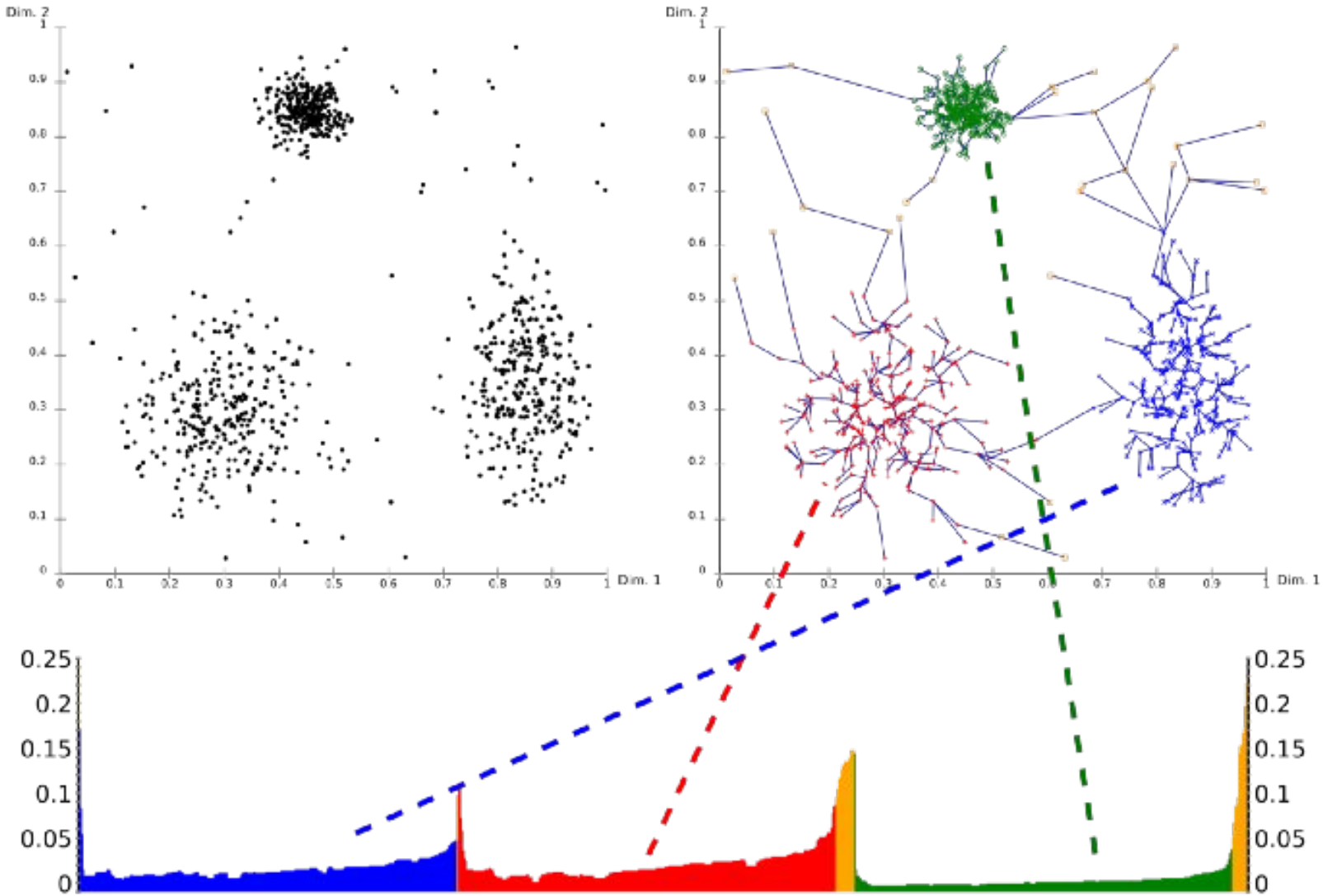
OPTICS Pseudo-Code

- Function *update(N, p, Q)*
  - Calculate the core distance for *p*
  - For each neighbor *q in N* (update the reachability)
    - If *q* is not processed
      - *new_rd* = reachability distance between *p* and *q*
    - If *q is not in Q*
      - *Q.insert(q, new_rd)*
    - Else
      - *If new_rd < q.rd*
        - *Q.move_up(q, new_rd)*

OPTICS Output

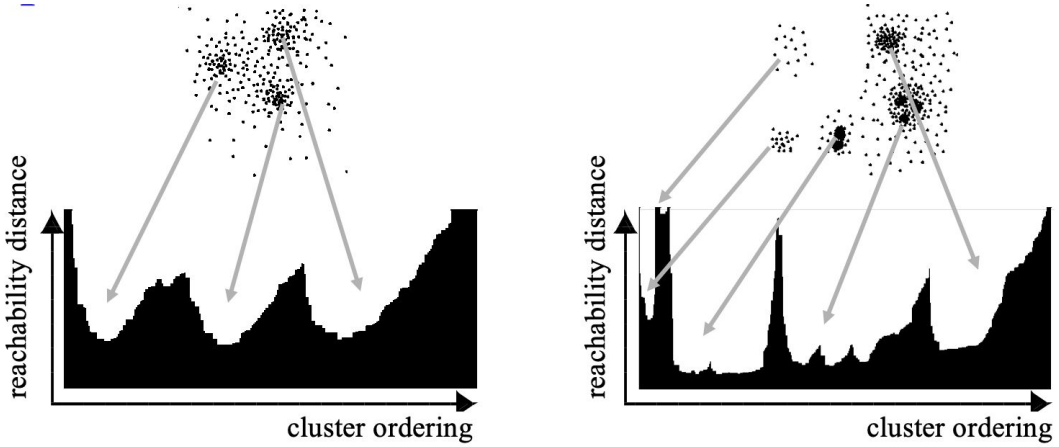
- OPTICS outputs the points in a particular ordering, annotated with their smallest reachability distance.
- A reachability-plot (a special kind of dendrogram), the hierarchical structure of the clusters can be obtained easily.
- x-axis: the ordering of the points as processed by OPTICS
- y-axis: the reachability distance
- Points belonging to a cluster have a low reachability distance to their nearest neighbor, the clusters show up as valleys in the reachability plot. The deeper the valley, the denser the cluster.

OPTICS Output

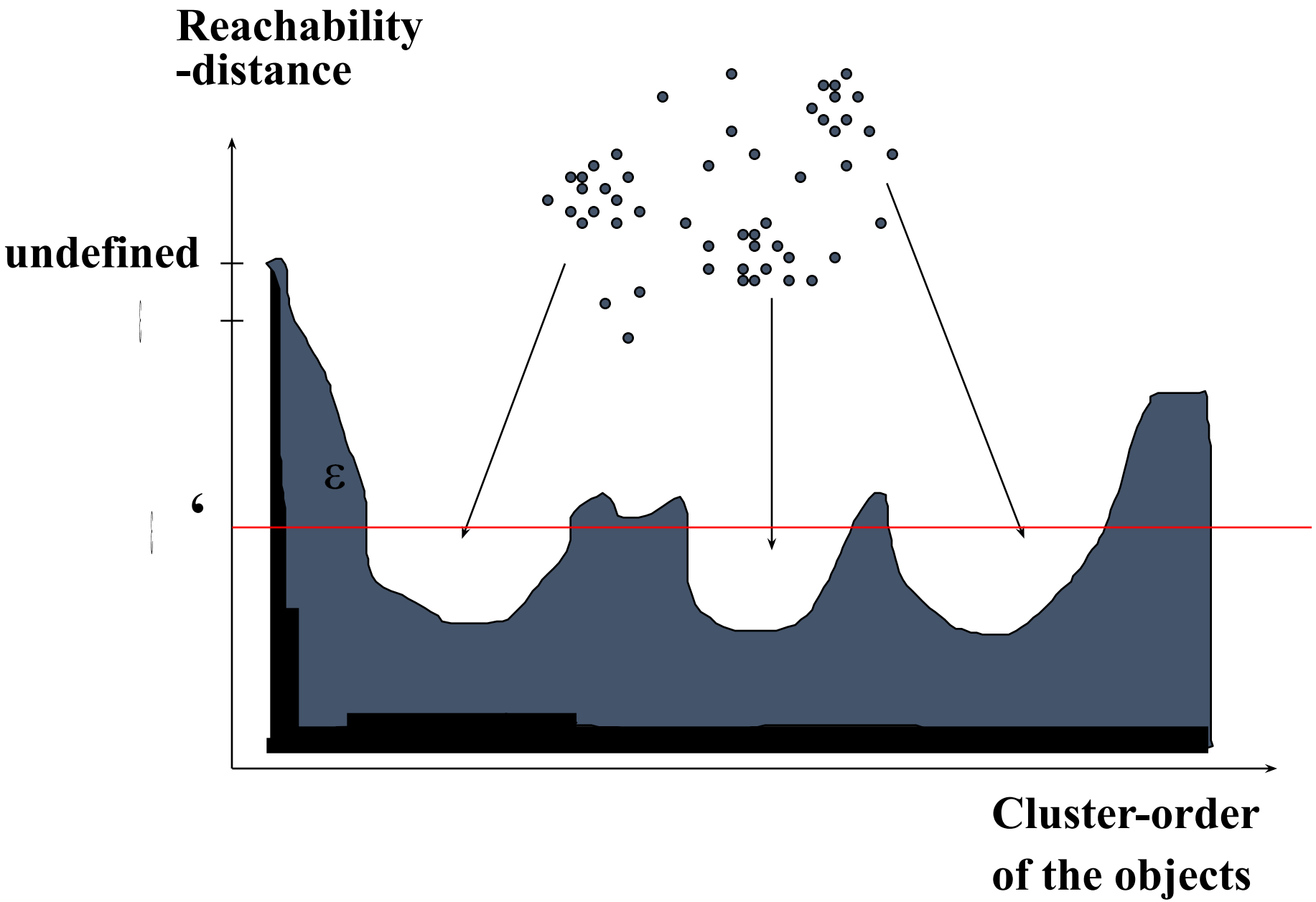


OPTICS Output

- Clusters are extracted
  1. by selecting a range on the x-axis after visual inspection,
  2. by selecting a threshold on the y-axis
  3. by different algorithms that try to detect the valleys by steepness, knee detection, or local maxima. Clustering obtained this way usually are hierarchical, and cannot be achieved by a single DBSCAN run.



https://scikit-learn.org/stable/auto_examples/cluster/plot_optics.html#sphx-glr-auto-examples-cluster-plot-optics-py


Reachability
-distance
undefined
ε
Cluster-order
of the objects

OPTICS: The Radius Parameter

- Both core-distance and reachability-distance are undefined if no sufficiently dense cluster (w.r.t. ε) is available.
- Given a sufficiently large ε, this never happens, but then every ε-neighborhood query returns the entire database.
- Hence, the ε parameter is required to cut off the density of clusters that are no longer interesting, and to speed up the algorithm.
- The parameter ε is, strictly speaking, not necessary.
- It can simply be set to the maximum possible value.
- When a spatial index is available, however, it does play a practical role with regards to complexity.
- OPTICS abstracts from DBSCAN by removing this parameter, at least to the extent of only having to give the maximum value.

DBSCAN Evolution

HDBSCAN

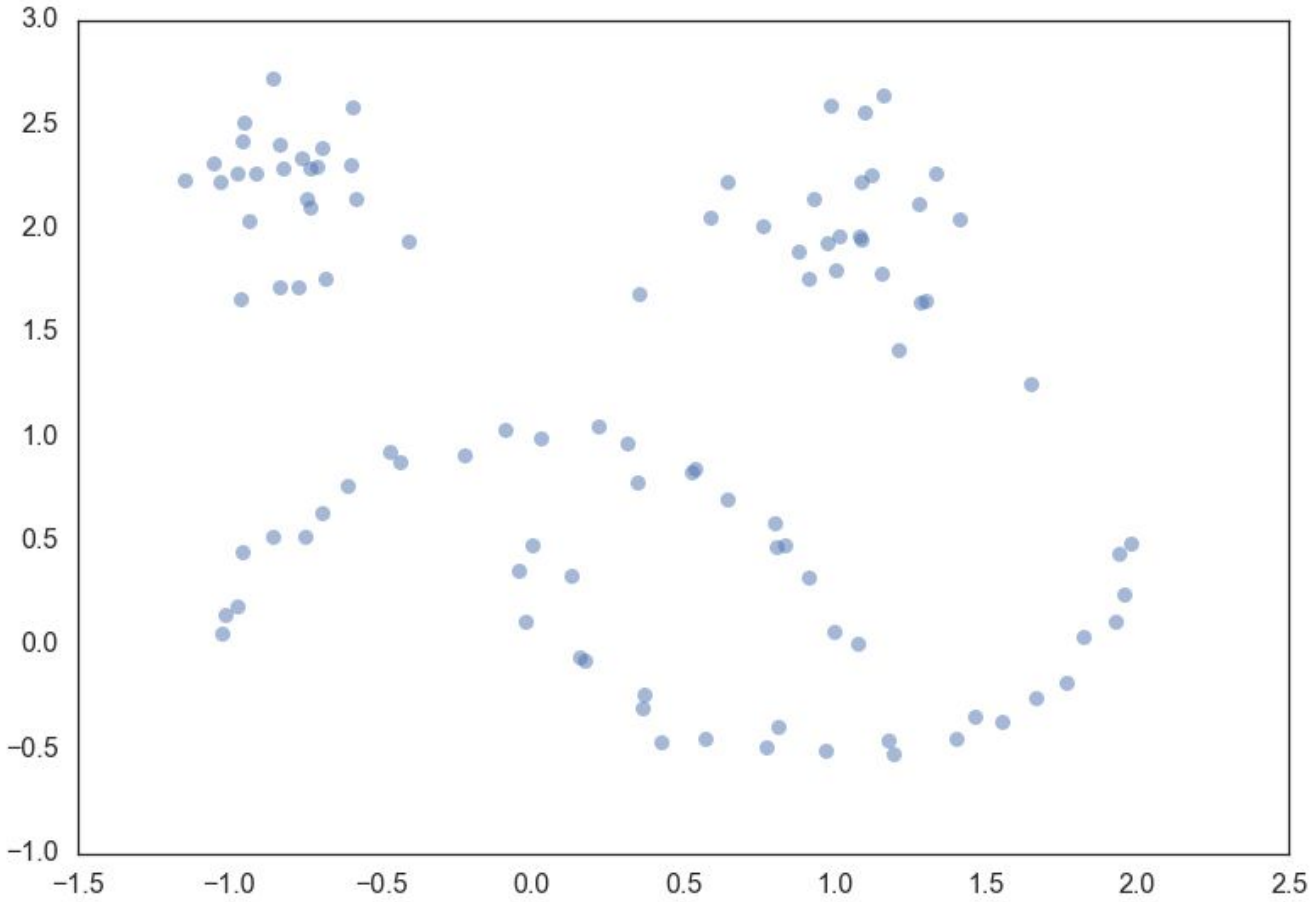
HDBSCAN

- HDBSCAN extends DBSCAN by converting it into a hierarchical clustering algorithm, but it also extracts a flat clustering.
- HDBSCAN bypass the choice of the Eps parameter
- HDBSCAN scans all possible solution with all values of Eps

HDBSCAN Main Steps

1. Transform the space according to the density/sparsity
2. Build the minimum spanning tree of the distance weighted graph
3. Construct a cluster hierarchy of connected components.
4. Condense the cluster hierarchy based on minimum cluster size.
5. Extract the stable clusters from the condensed tree.

How HDBSCAN Works



Step 1: Transform The Space

- **Goal**: Prepare the data for a single linkage clustering (real data is noisy and single linkage is not robust!)
- **Idea**: Push sparse points away from the rest of the data before clustering
- How do we evaluate density ?
  - Need an inexpensive density estimate ⇒ k-NN is the simplest
  - Call it the **core distance** for parameters k and point x_i, $core_k(x_i)$

Recall

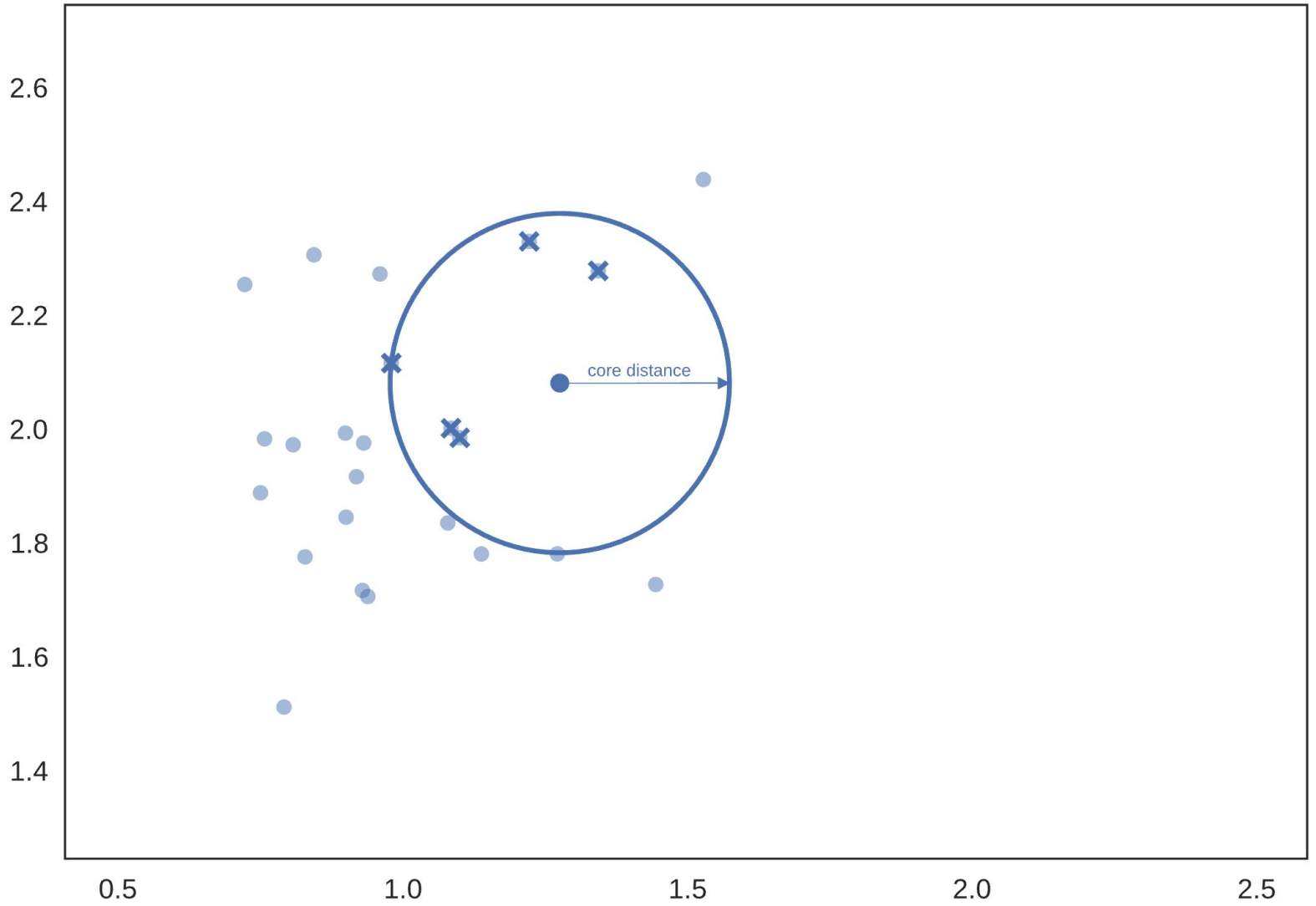
Core point. A point *p* is a core point if at least MinPts points are found within its ε-neighborhood.

Core Distance. It is the **minimum** value of radius required to classify a given point as a core point.
If the given point is not a Core point, then it's Core Distance is undefined.

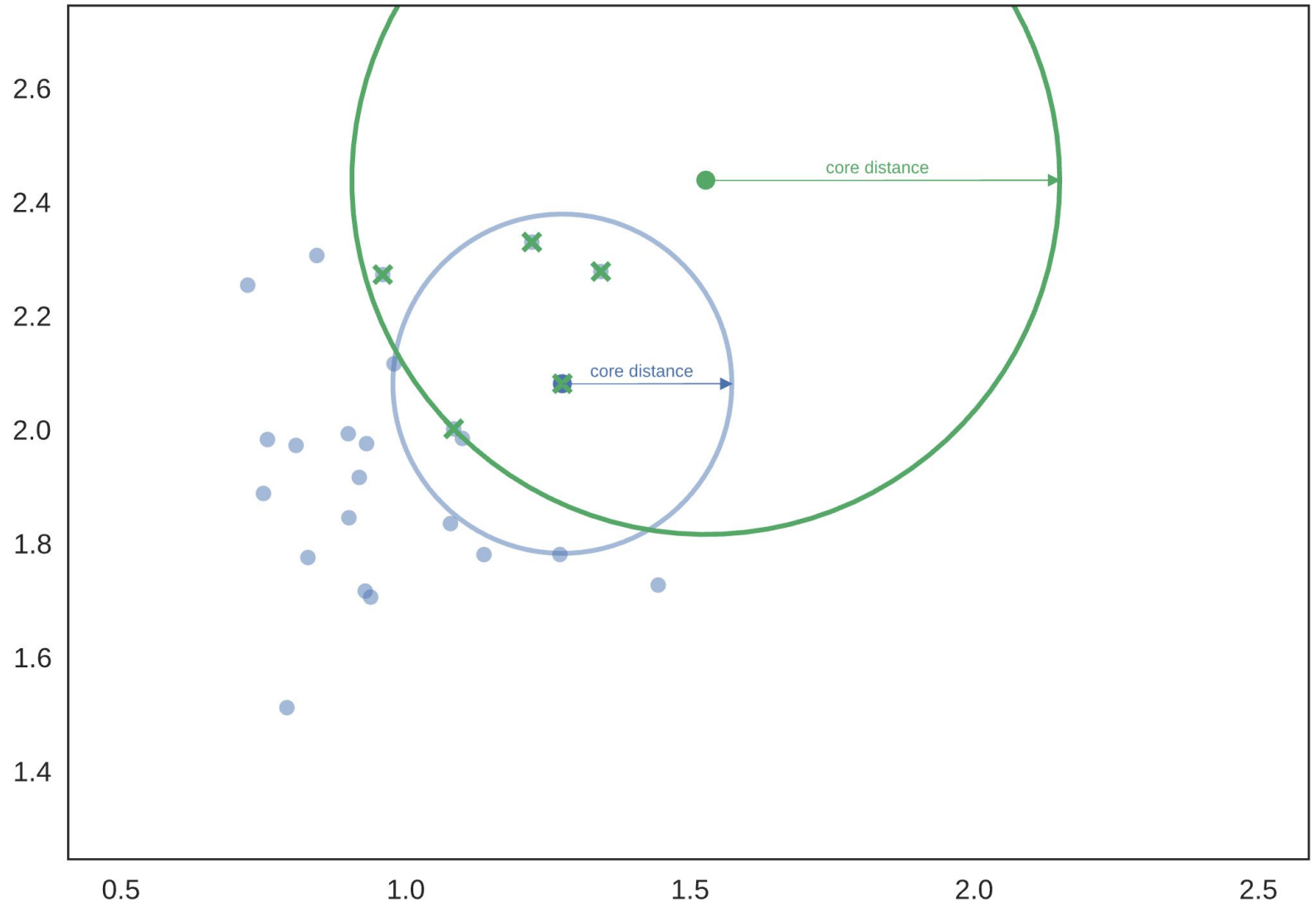
Step 1: Mutual Reachability Distance

- **Goal**: We need a way to spread apart points with low density (correspondingly high core distance).
- $d_{mreach\text{-}k}(x_i, x_j) = max\{core_k(x_i), core_k(x_j), d(x_i, x_j)\}$
- **Objectives:** connect points that are
  - close enough to each other : $d(x_i, x_j)$
  - in a dense enough region : $core_k(x_i)$
- With this metric dense points (with low core distance) remain the same distance from each other, but sparser points are pushed away to be at least their core distance away from any other point.

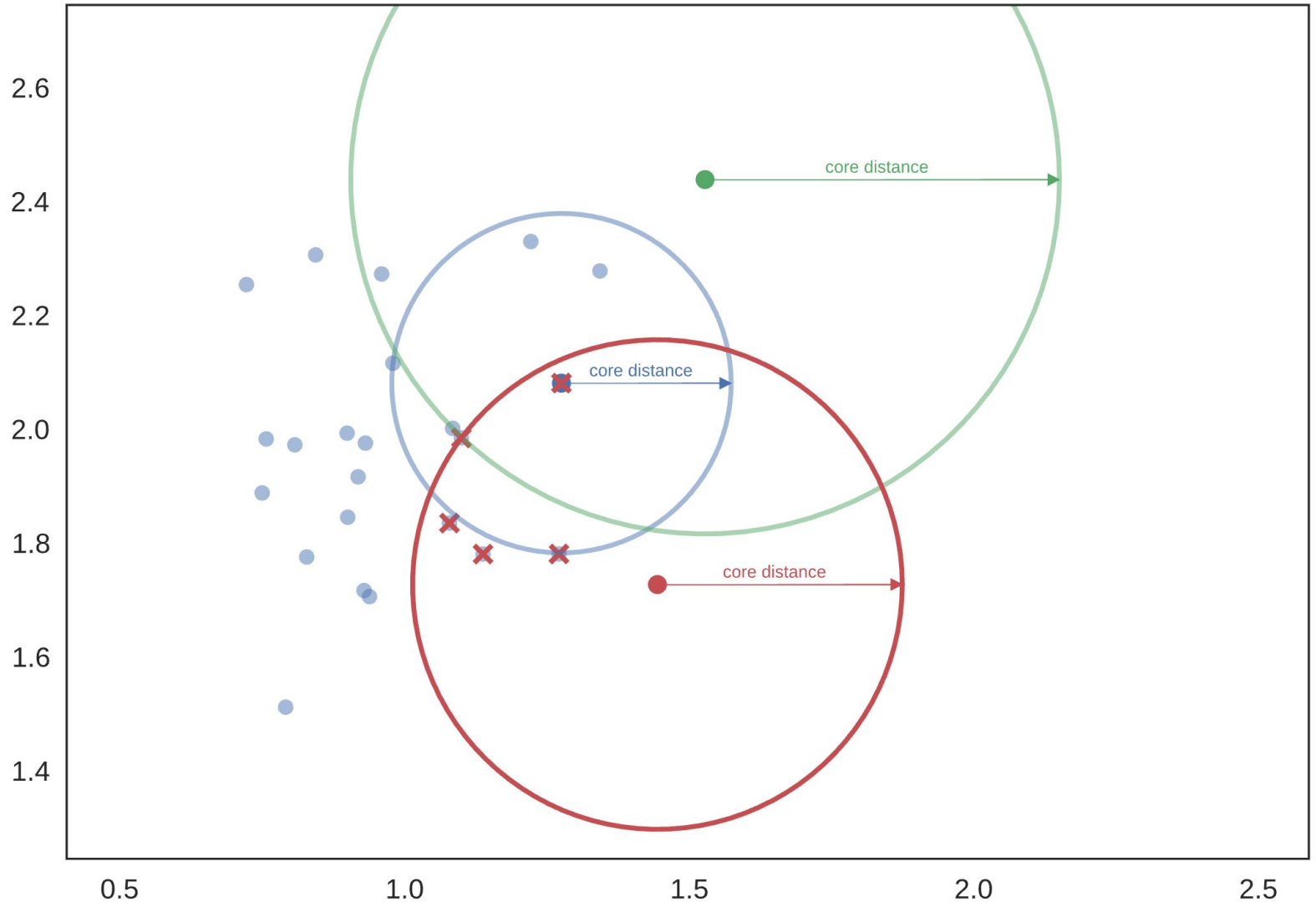
Step 1: Mutual Reachability Distance



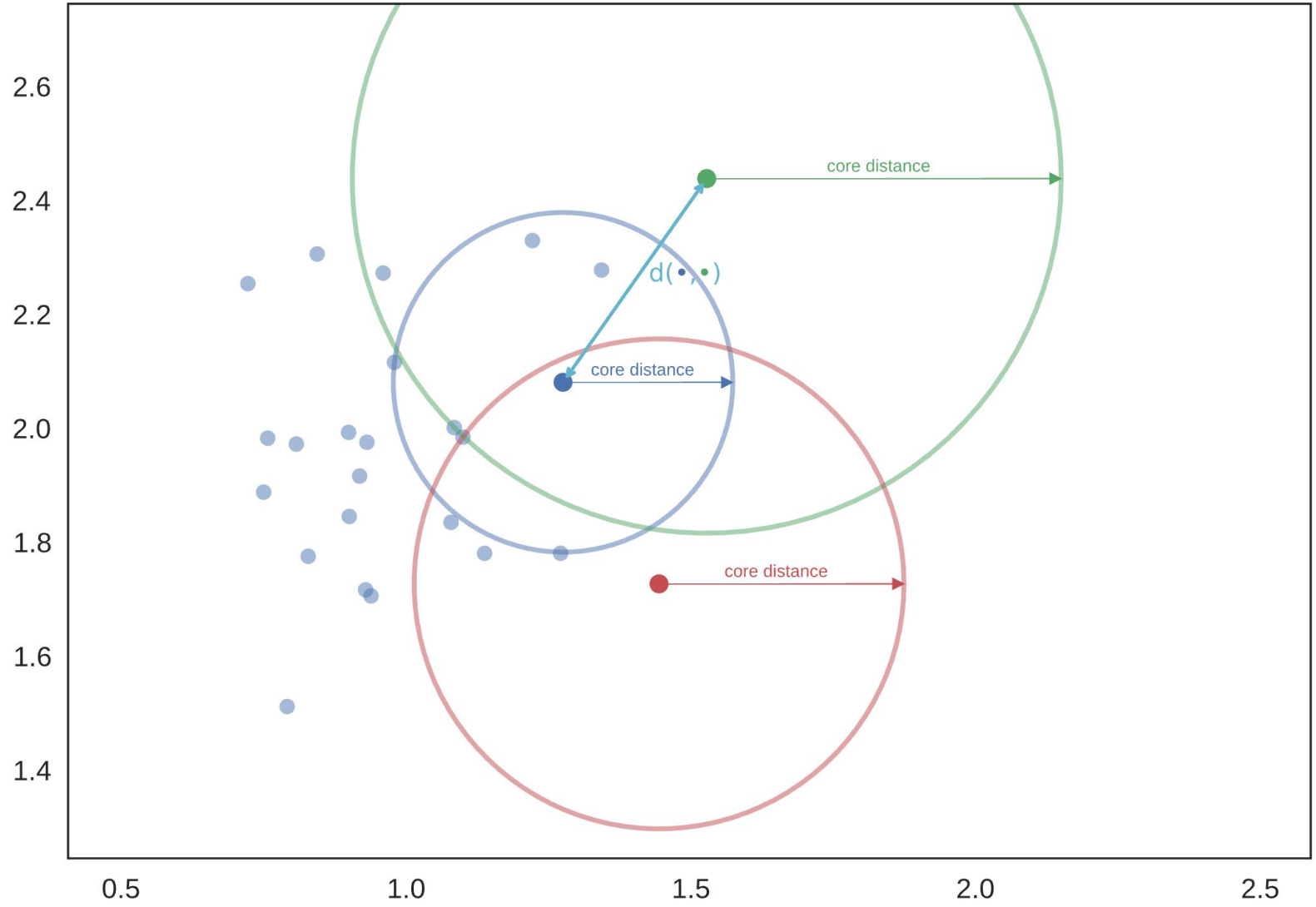
Step 1: Mutual Reachability Distance



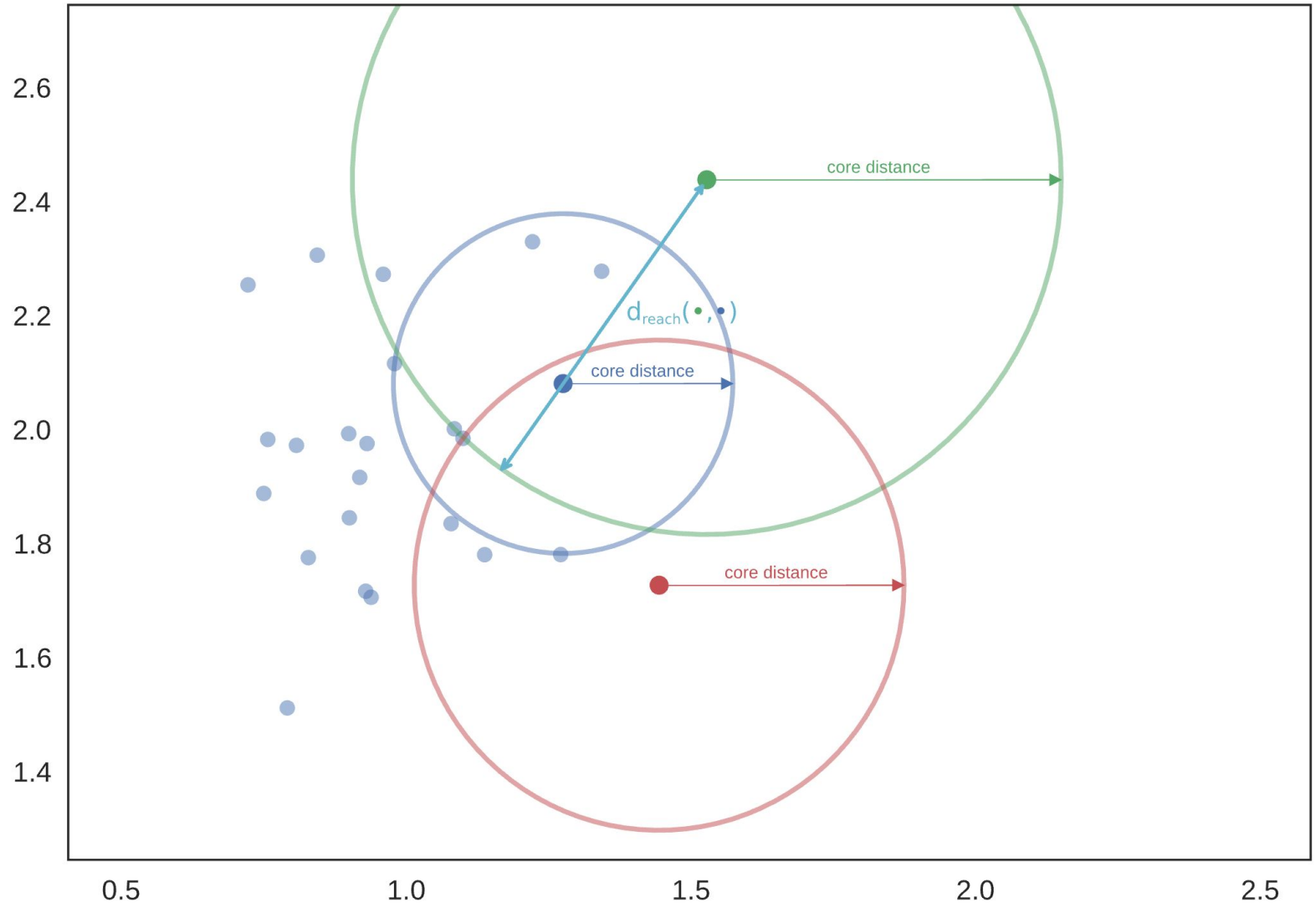
Step 1: Mutual Reachability Distance



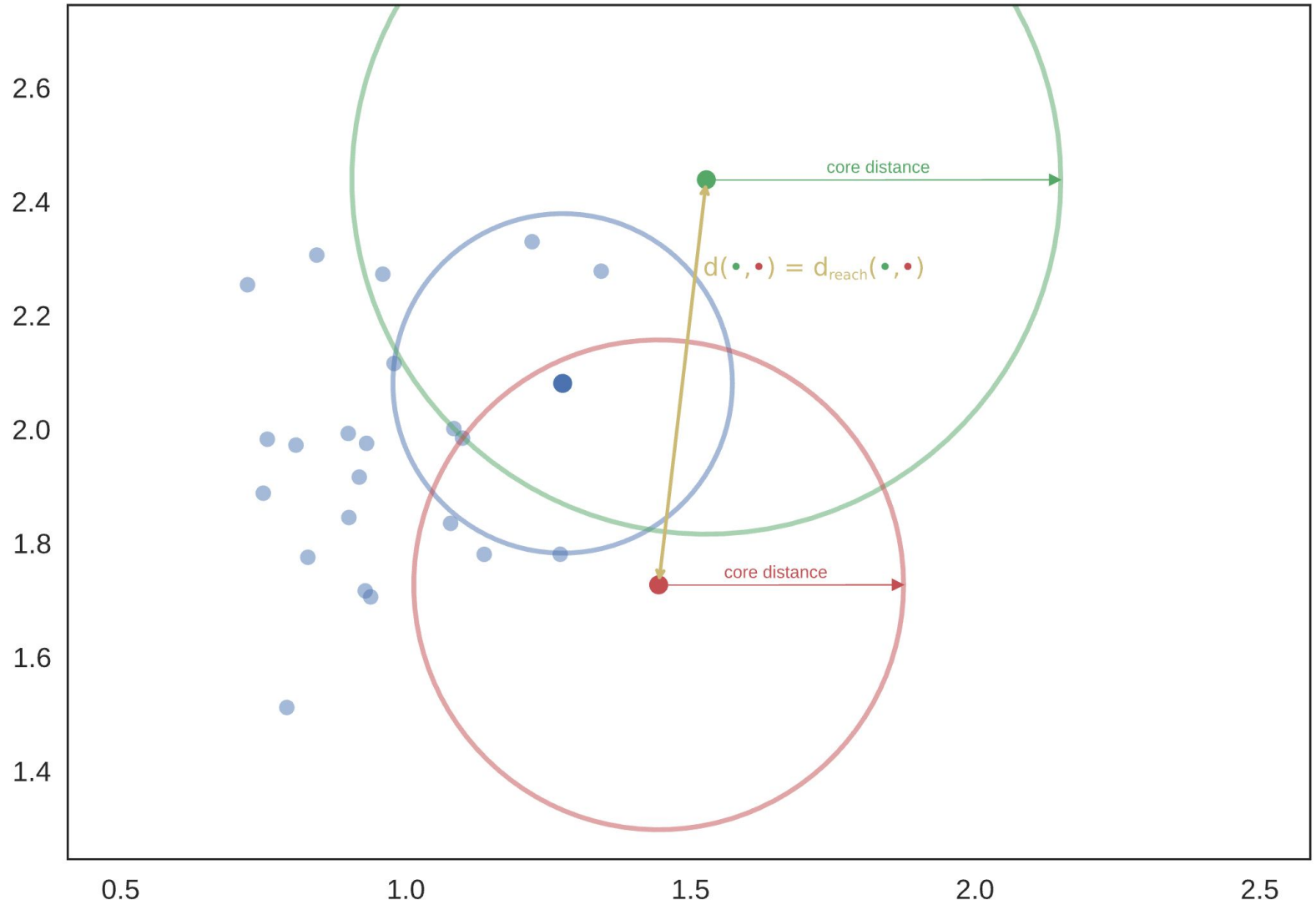
Step 1: Mutual Reachability Distance



Step 1: Mutual Reachability Distance



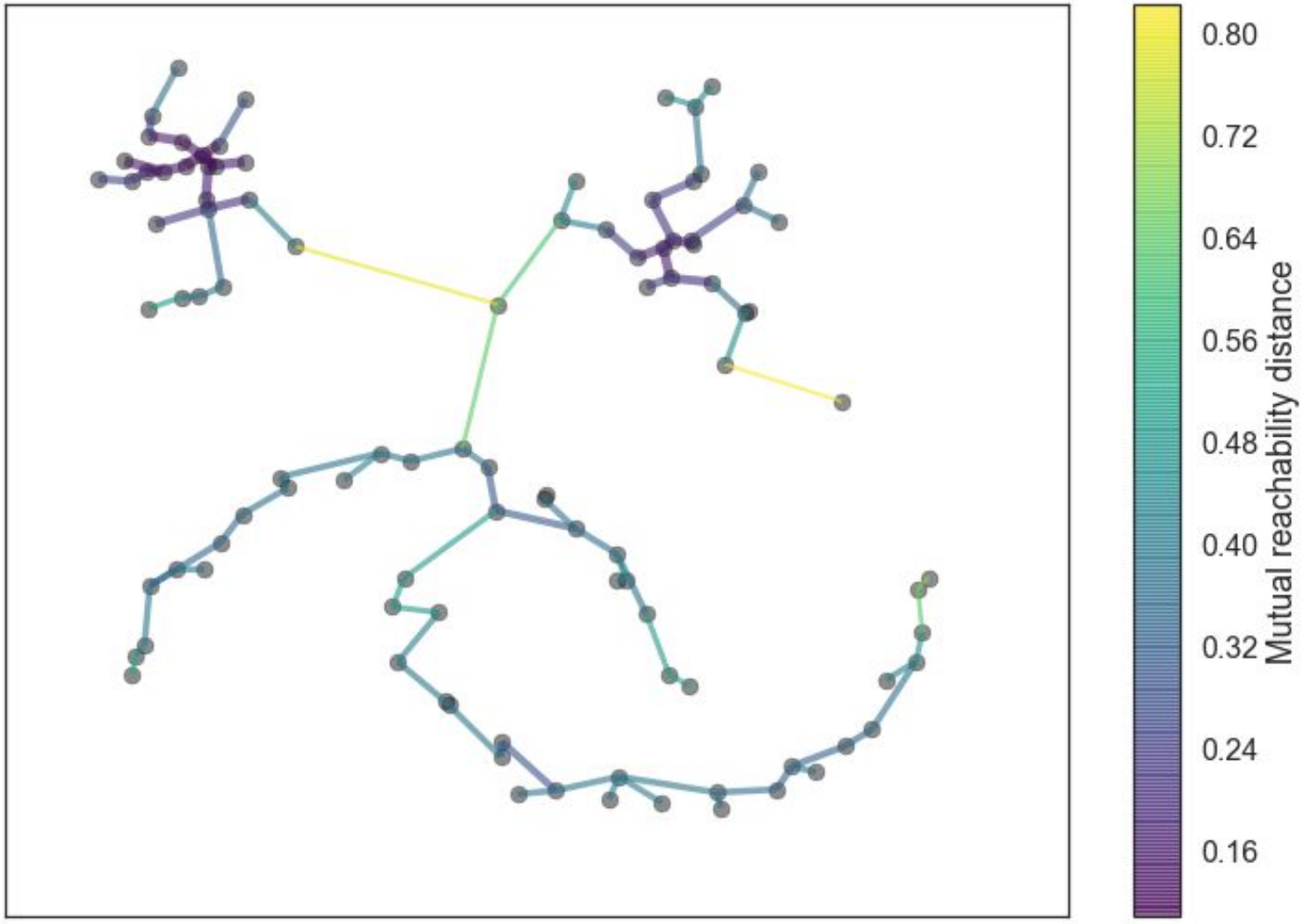
Step 1: Mutual Reachability Distance



Step 2 : The Minimum Spanning Tree

- **Goal**: Prepare the data for clustering using d_{mreach}
- Ideas:
  - Construct a graph that connects all points
  - Points are the vertices and the edges are weighted by d_{mreach}
  - Start disconnecting points by lowering a threshold
  - Among n^2 possible edges identify with the **minimum spanning tree** a threshold such that there is no lower weight edge that could connect the components
  - **Prim's algorithm:** build the tree one edge at a time, always adding the lowest weight edge that connects the current tree to a vertex not yet in the tree

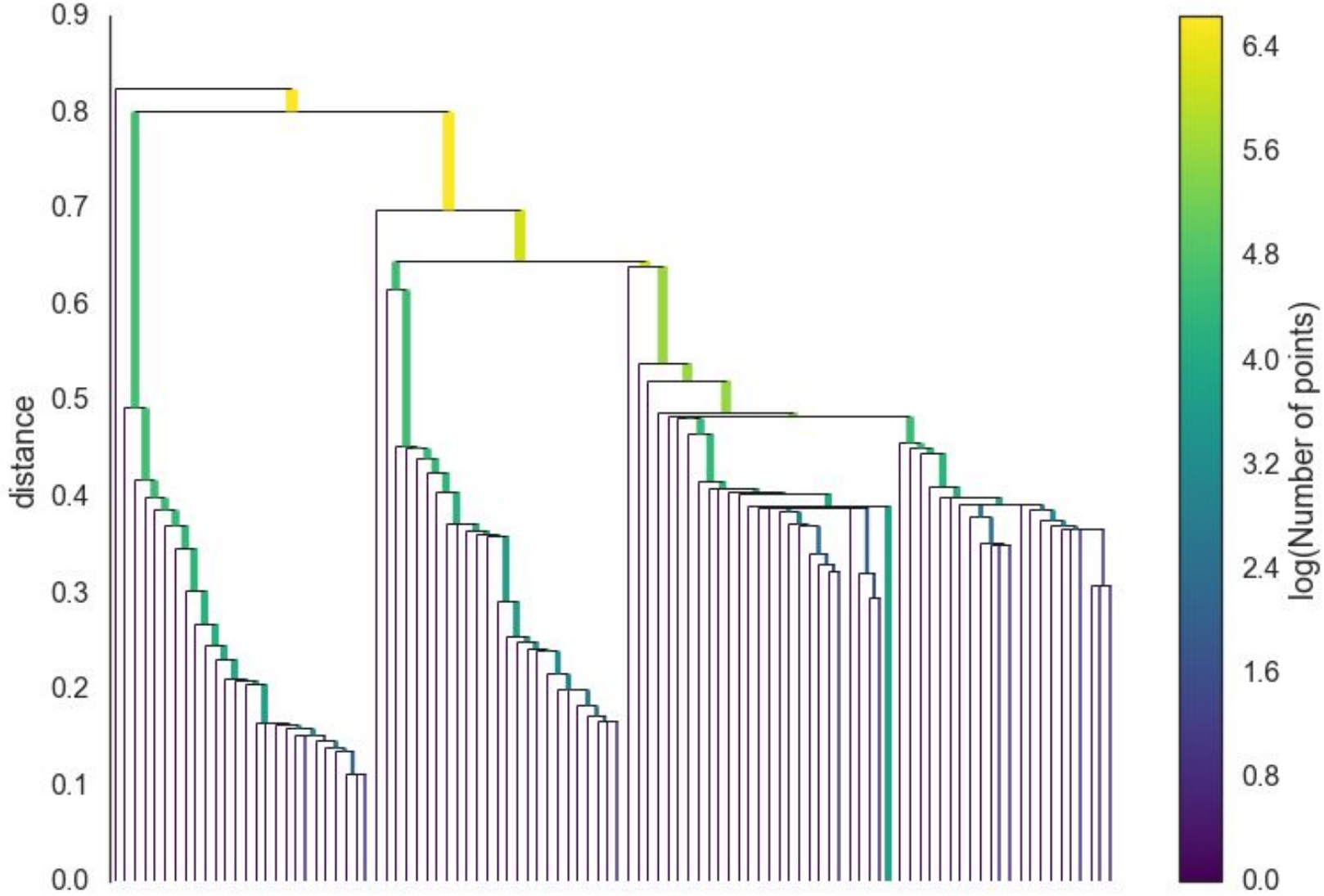
Step 2 : The Minimum Spanning Tree



Step 3: Build the Cluster Hierarchy

- **Goal**: Given the minimal spanning tree, the next step is to convert that into the hierarchy of connected components.
- **Idea**: Sort the edges of the tree by distance (in increasing order) and then iterate through, creating a new merged cluster for each edge, i.e., run single linkage hierarchical cluster.ing algorithm

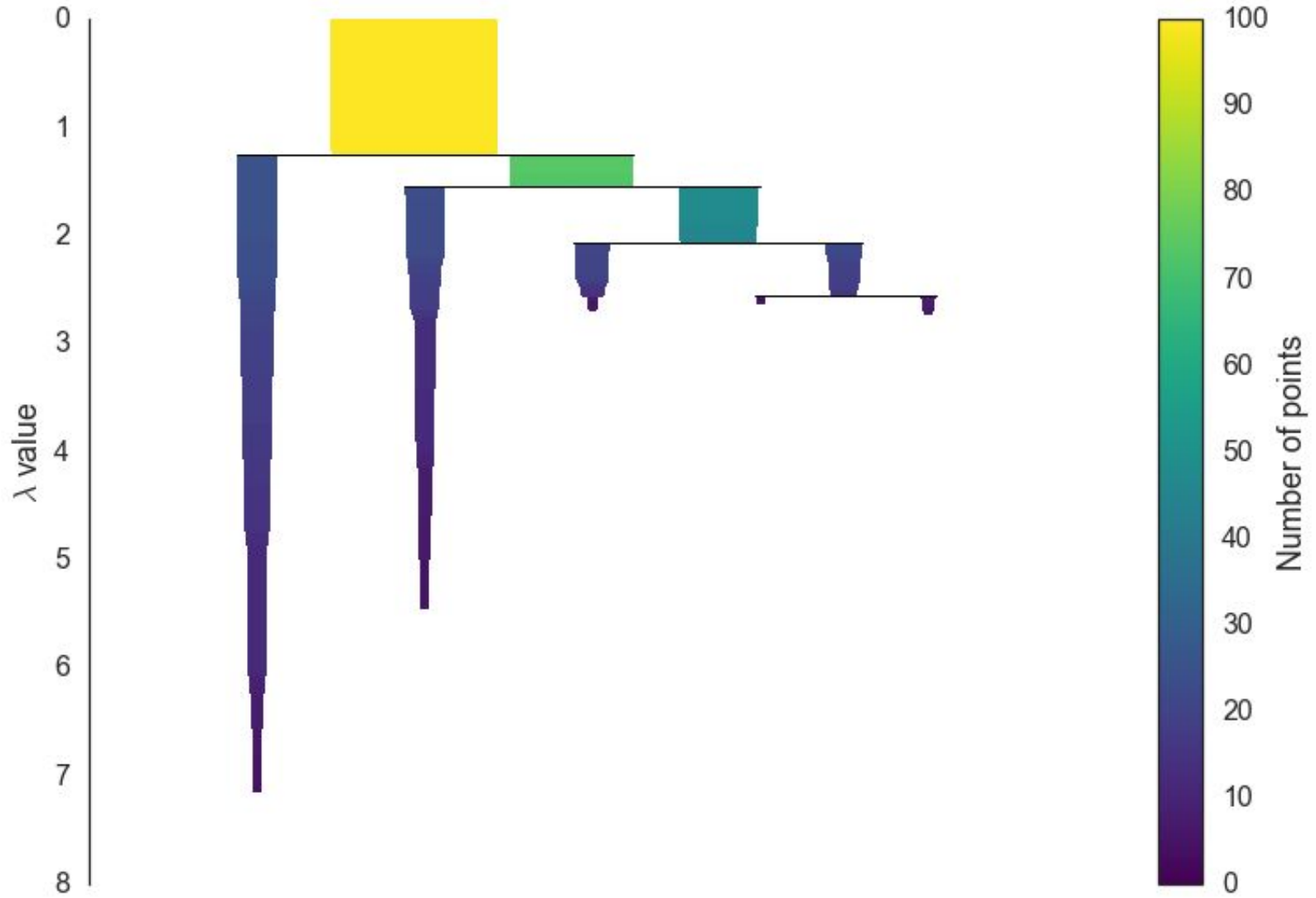
Step 3: Build the Cluster Hierarchy



Step 4 : Condense the Cluster Tree

- **Goal**: Condensing the large and complicated cluster hierarchy into a smaller tree with a little more data attached to each node.
- **Idea**: Use a notion of **minimum cluster size** which we take as a parameter to HDBSCAN.
  - Walk through the hierarchy top down and at each split ask if one of the new clusters created by the split has fewer points than the minimum cluster size.
  - If it is the case, declare it to be ‘points falling out of a cluster’ and have the larger cluster retain the cluster identity of the parent, marking down which points ‘fell out of the cluster’, i.e., noise points, and at what distance value that happened.
  - On the other hand, if the split is into two clusters each at least as large as the minimum cluster size then we consider that a true cluster split and let that split persist in the tree.

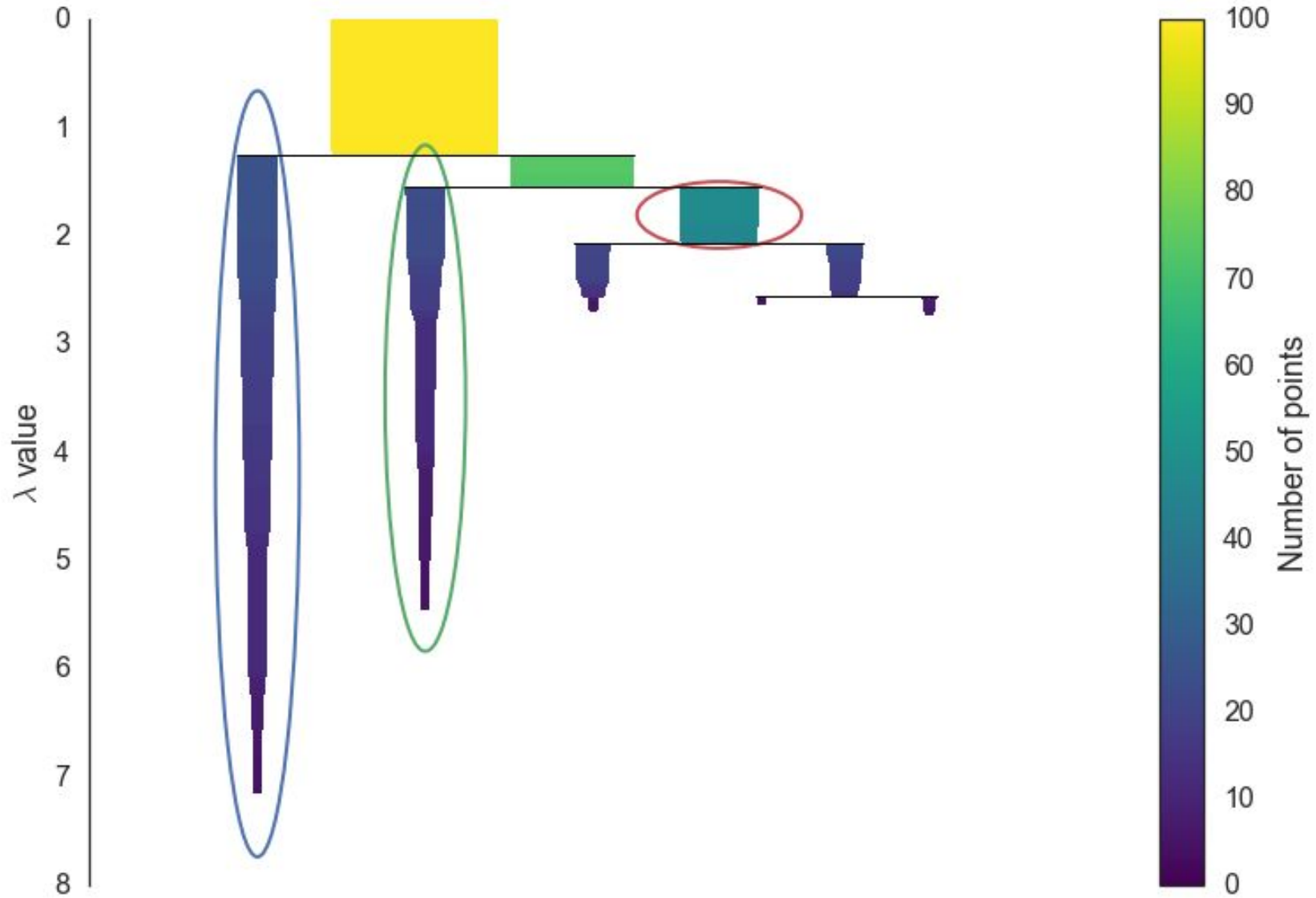
Step 4 : Condense the Cluster Tree



Step 5: Extract the Stable Clusters

- **Goal**: Choose clusters that persist and have a longer lifetime (short clusters are probably artifacts of single linkage)
- **Idea**: Looking at the previous plot we could say that we want to choose those clusters that have the greatest area of ink in the plot
- **Requirement**: if you select a cluster, then you cannot select any cluster that is a descendant of it

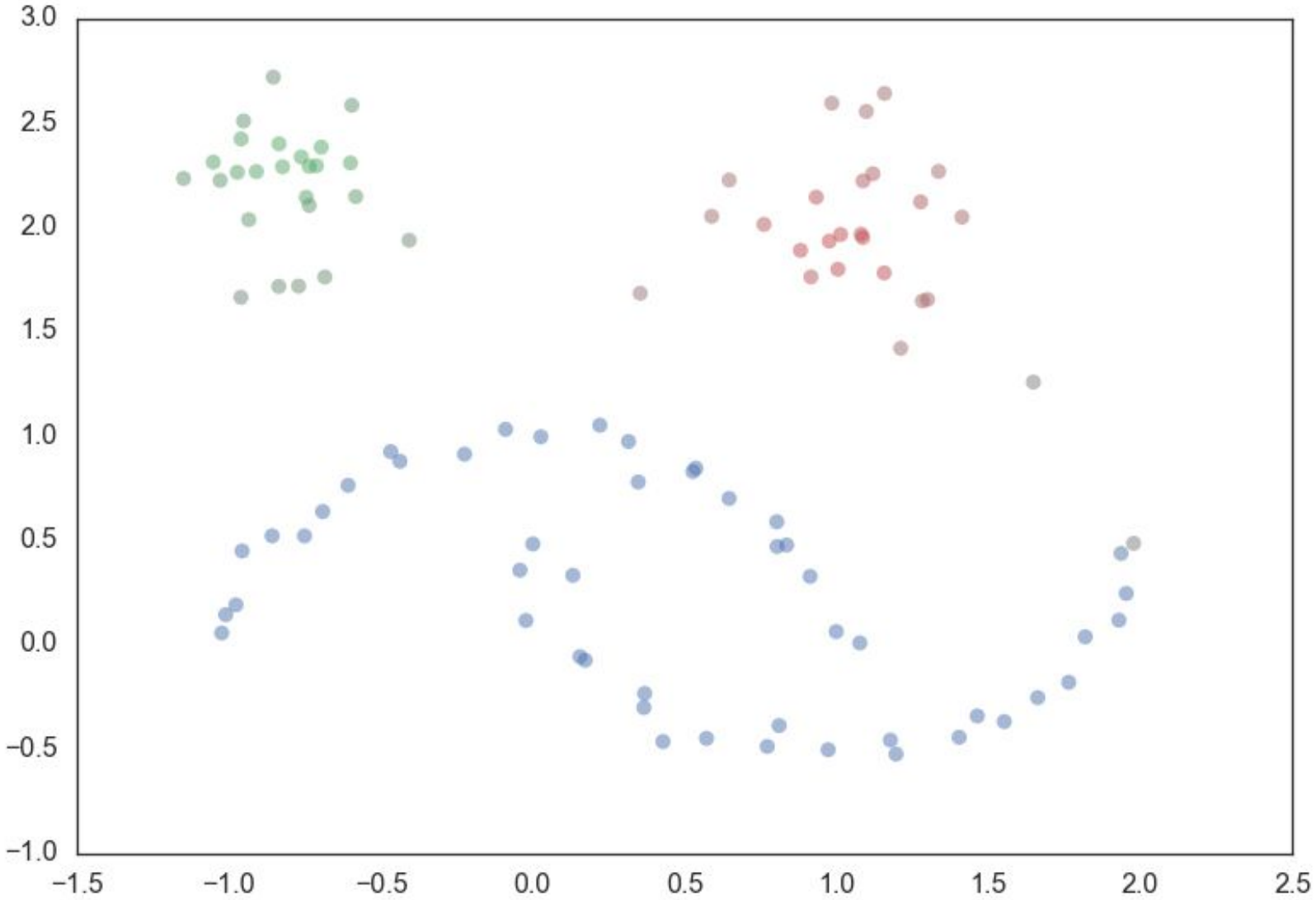
Step 5: Extract the Stable Clusters



Step 5: Extract the Stable Clusters (details)

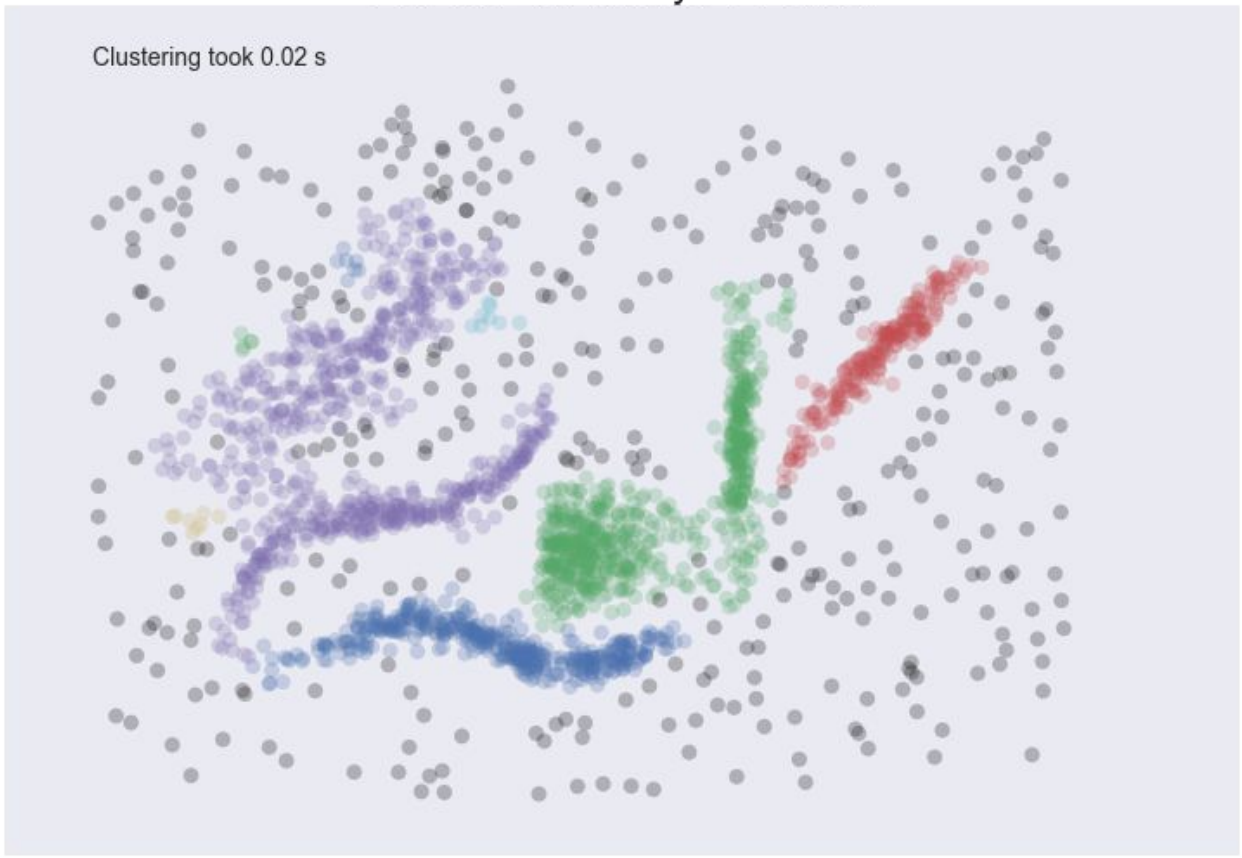
- Use $\lambda = 1/\text{distance}$
- For a given cluster we have λ_{start}, λ_{end} to identify the cluster boundaries
- For a given cluster, the value $\lambda_p \in [\lambda_{start}, \lambda_{end}]$ defines at which value a point "fell out of the cluster"
- Compute the **stability** of a cluster as $\sum_{p \in C} (\lambda_p - \lambda_{start})$
- Declare all leaf nodes to be clusters.
- Now work bottom up through the tree.
  - If the sum of the stabilities of the child clusters is greater than the stability of the cluster, then set the cluster stability to be the sum of the child stabilities.
  - If, on the other hand, the cluster's stability is greater than the sum of its children then we declare the cluster to be a selected cluster and unselect all its descendants.
  - Once at the root node all the set of selected clusters is the flat clustering returned.

HDBSCAN Result



DBSCAN vs HDBSCAN

Clusters found by DBSCAN



Clusters found by HDBSCAN

References

- Clustering. Chapter 7. Introduction to Data Mining.
- Mihael Ankerst; Markus M. Breunig; Hans-Peter Kriegel; Jörg Sander (1999). OPTICS: Ordering Points To Identify the Clustering Structure.