Clustering Overview

Last lecture

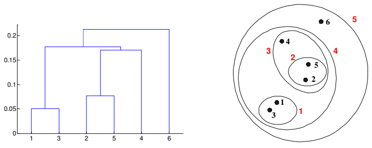
- What is clustering
- Partitional algorithms: K-means

Today's lecture

- Hierarchical algorithms
- Density-based algorithms: DBSCAN
- Techniques for clustering large databases
  - BIRCH
  - CURE

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



Strengths of Hierarchical Clustering

- We do not have to assume (or provide) 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
  - Examples in
    - biological sciences (e.g., animal kingdom, phylogeny reconstruction, …)
    - natural languages

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 there is 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

```
1.  Compute the proximity matrix
2.  Let each data point be a cluster
3.  Repeat
4.          Merge the two closest clusters
5.          Update the proximity matrix
6.  Until only a single cluster remains
```

Key operation is the computation of the proximity of two clusters

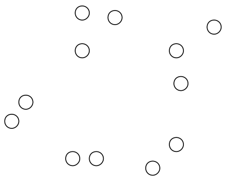
Different approaches to defining the distance between clusters distinguish the different algorithms

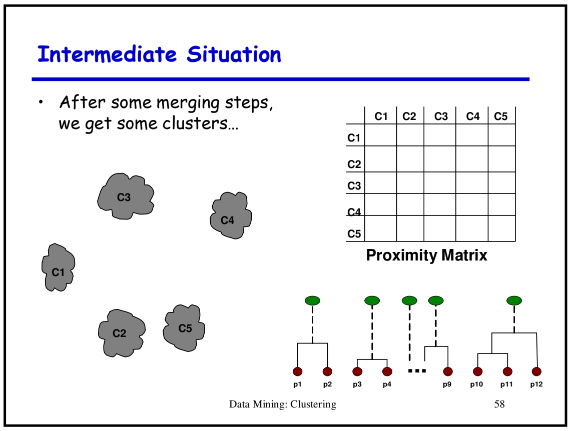
Starting Situation

- Start with clusters of individual points and a proximity matrix

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

Proximity Matrix



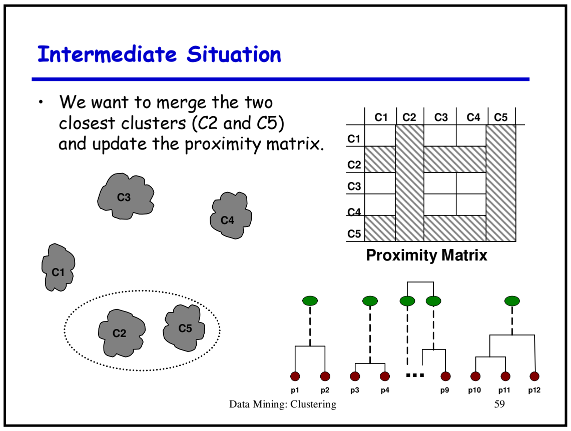


Intermediate Situation

- After some merging steps, we get some clusters…

	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

Proximity Matrix

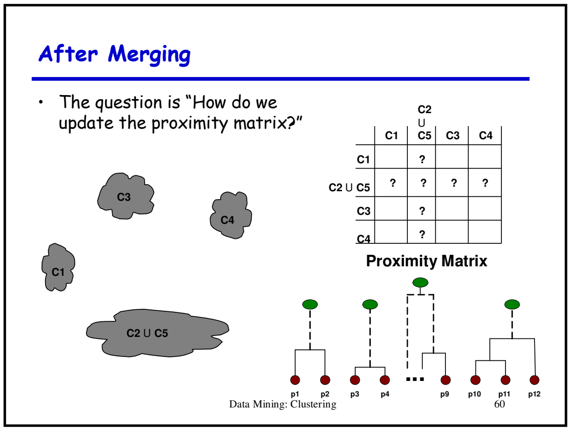


Intermediate Situation

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

	C1	C2	C3	C4	C5
C1		▨			▨
C2	▨	▨	▨	▨	▨
C3		▨			▨
C4		▨			▨
C5	▨	▨	▨	▨	▨

Proximity Matrix

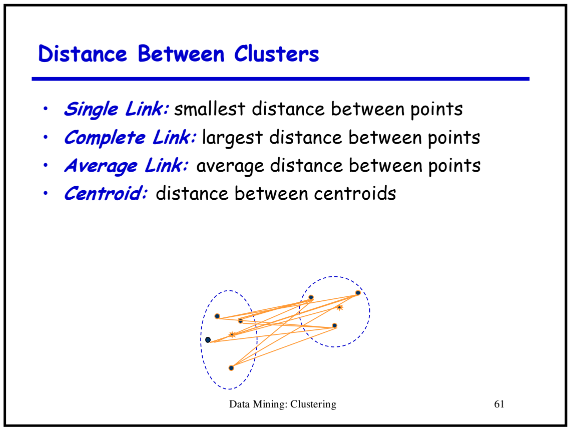


After Merging

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

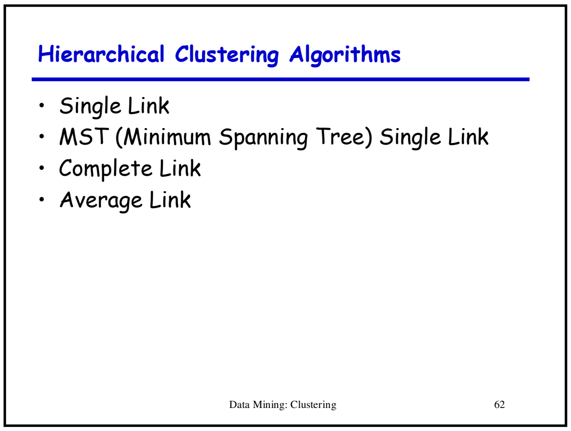
	C1	C2 U C5	C3	C4
C1		?		
C2 U C5	?	?	?	?
C3		?		
C4		?		

Proximity Matrix



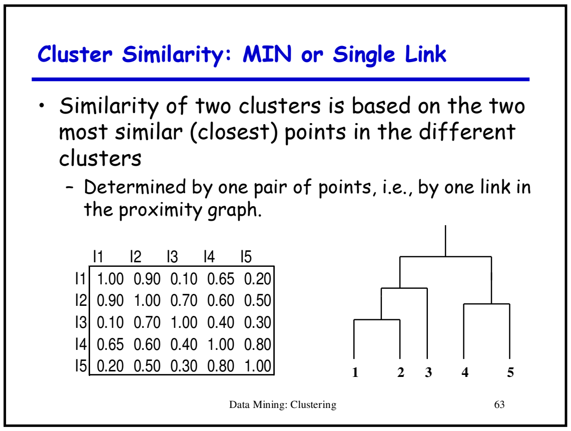
Distance Between Clusters

- *Single Link:* smallest distance between points
- *Complete Link:* largest distance between points
- *Average Link:* average distance between points
- *Centroid:* distance between centroids



Hierarchical Clustering Algorithms

- Single Link
- MST (Minimum Spanning Tree) Single Link
- Complete Link
- Average Link



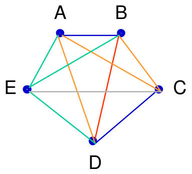
Cluster Similarity: MIN or Single Link

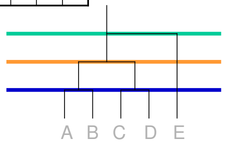
- Similarity of two clusters is based on the two most similar (closest) points in the different clusters
  - Determined by one pair of points, i.e., by one link in the proximity graph.

	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

Agglomerative MIN Clustering Example

	A	B	C	D	E
A	0	1	2	2	3
B	1	0	2	4	3
C	2	2	0	1	5
D	2	4	1	0	3
E	3	3	5	3	0

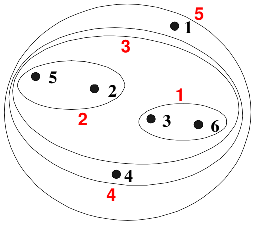


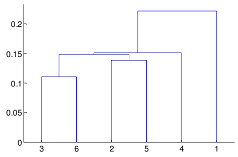


Threshold of

1 2 3 4 5

Hierarchical Clustering: MIN

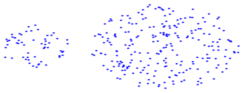


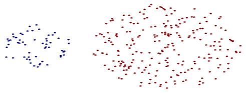


Nested Clusters

Dendrogram

Strength of MIN



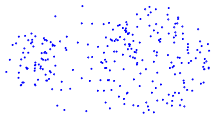


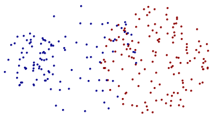
Original Points

Two Clusters

- Can handle non-elliptical shapes

Limitations of MIN





Original Points

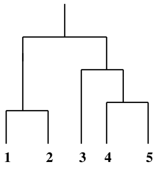
Two Clusters

- Sensitive to noise and outliers

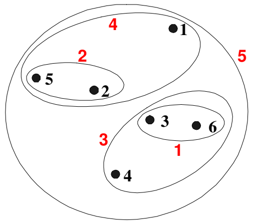
Cluster Similarity: MAX or Complete Linkage

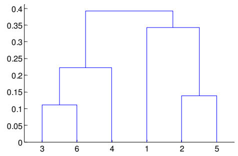
- Similarity of two clusters is based on the two least similar (most distant) points in the different clusters
  - Determined by all pairs of points in the two clusters

	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: MAX

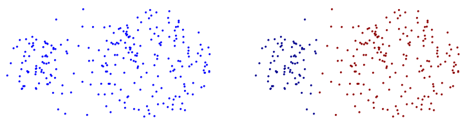




Nested Clusters

Dendrogram

Strength of MAX

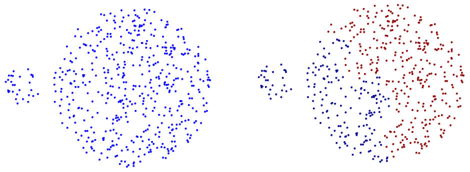


Original Points

Two Clusters

Less susceptible to noise and outliers

Limitations of MAX



Original Points

Two Clusters

Tends to break large clusters

Biased towards globular clusters

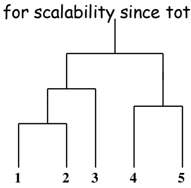
Cluster Similarity: Group Average

- Proximity of two clusters is the average of pairwise proximity between points in the two clusters.

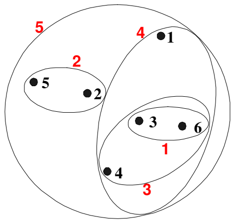
$$\boxed{\text{proximity}(\text{Cluster}_i, \text{Cluster}_j) = \frac{\sum\limits_{\substack{p_i \in \text{Cluster}_i \\ p_j \in \text{Cluster}_j}} \text{proximity}(p_i, p_j)}{|\text{Cluster}_i| * |\text{Cluster}_j|}}$$

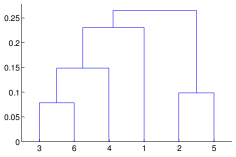
- Need to use average connectivity for scalability since total proximity favors large clusters

	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: Group Average





Nested Clusters

Dendrogram

Hierarchical Clustering: Group Average

- Compromise between Single and Complete Link

- Strengths
  - Less susceptible to noise and outliers

- Limitations
  - Biased towards globular clusters

Cluster Similarity: Ward's Method

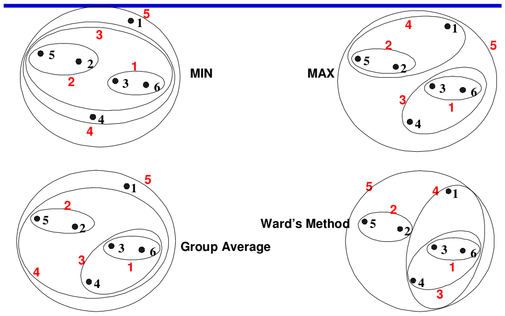
- Similarity of two clusters is based on the increase in squared error when two clusters are merged
  - Similar to group average if distance between points is distance squared

- Less susceptible to noise and outliers

- Biased towards globular clusters

- Hierarchical analogue of K-means
  - Can be used to initialize K-means

Hierarchical Clustering: Comparison



Hierarchical Clustering: Time & Space Requirements

- $O(N^2)$ space since it uses the proximity matrix.
  - N is the number of points.
- $O(N^3)$ time in many cases
  - There are N steps and at each step the size, N^2, proximity matrix must be updated and searched
  - Complexity can be reduced to $O(N^2 \log(N))$ time for some approaches

Hierarchical Clustering: Problems and Limitations

- Once a decision is made to combine two clusters, it cannot be undone
- No objective function is directly minimized
- Different schemes have problems with one or more of the following:
  - Sensitivity to noise and outliers
  - Difficulty handling different sized clusters and convex shapes
  - Breaking large clusters

Partitional Clustering (reminder)

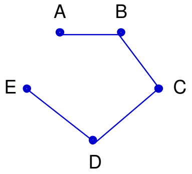
- Nonhierarchical
- Usually deals with static sets
- Creates clusters in one step as opposed to several steps
- Since only one set of clusters is output, the user normally has to input the desired number of clusters, k
- Need some metric/criterion that determines the goodness of each proposed solution

Some Partitional Clustering Algorithms

- MST: Minimum Spanning Tree
- Squared Error
- K-Means
- Nearest Neighbor
- PAM: Partitioning Around Medoids
  - CLARA: Clustering LARge Applications
  - CLARANS: Clustering Large Applications based on RANdomized Search

MST Example

	A	B	C	D	E
A	0	1	2	2	3
B	1	0	2	4	3
C	2	2	0	1	5
D	2	4	1	0	3
E	3	3	5	3	0



MST Algorithm

```
Input:
    $D = \{t_1, t_2, ..., t_n\}$    // Set of elements
    $A$      // Adjacency matrix showing distance between elements.
    $k$      // Number of desired clusters.
Output:
    $f$      // Mapping represented as a set of ordered pairs.
Partitional MST Algorithm:
    $M = MST(A)$;
    identify inconsistent edges in M;
    remove $k - 1$ inconsistent edges;
    create output representation;
```

Sum Squared Error (SSE)

$$se_{K_i} = \sum_{j=1}^{m} \|t_{ij} - C_k\|^2$$

$$se_K = \sum_{j=1}^{k} se_{K_j}$$

Squared Error Algorithm

```
Input:
    $D = \{t_1, t_2, ..., t_n\}$    // Set of elements
    $k$      // Number of desired clusters.
Output:
    $K$      // Set of clusters.
Squared Error Algorithm:
    assign each item $t_i$ to a cluster;
    calculate center for each cluster;
    repeat
        assign each item $t_i$ to the cluster which has the closest center ;
        calculate new center for each cluster;
        calculate squared error;
    until the difference between successive squared errors is below a threshold;
```

Nearest Neighbor Clustering

- Items are iteratively merged into the existing clusters that are closest
- Incremental and serial algorithm
- Threshold, t, used to determine if items are added to existing clusters or whether a new cluster should be created

Nearest Neighbor Clustering Algorithm

```
Input:
    $D = \{t_1, t_2, ..., t_n\}$    // Set of elements
    $A$      // Adjacency matrix showing distance between elements.
Output:
    $K$      // Set of clusters.
Nearest Neighbor Algorithm:
    $K_1 = \{t_1\}$;
    $K = \{K_1\}$;
    $k = 1$;
    for $i = 1\ to\ n$ do
        find the $t_m$ in some cluster $K_m$ in $K$ such that $dis(t_i, t_m)$ is the smallest;
        if $dis(t_i, t_m) \leq t$ then
            $K_m = K_m \cup t_i$
        else
            $k = k + 1$;
            $K_k = \{t_i\}$;
```

PAM: Partitioning Around Medoids (K-Medoids)

- Handles outliers well
- Ordering of input does not impact results
- Computationally complex - does not scale well
- Each cluster represented by one item, called the medoid
- Initial set of k medoids is randomly chosen and all items which are not medoids are examined to see if they should replace an existing medoid.

PAM Cost Calculation

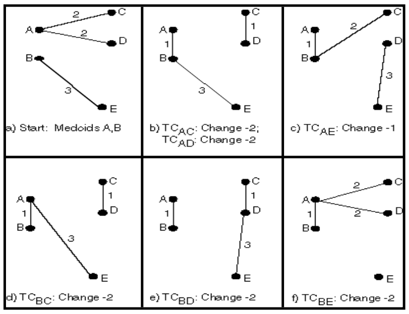
- At each step in algorithm, medoids are changed if the overall cost is improved.
- C_{jih} – cost change for an item t_j associated with swapping medoid t_i with non-medoid t_h.

1. $t_j \in K_i$, but $\exists$ another medoid t_m $where\ dis(t_j,t_m) \leq dis(t_j,t_h)$
2. $t_j \in K_i$, but $dis(t_j,t_h) \leq dis(t_j,t_m) \forall\ other\ medoids\ t_m$;
3. $t_j \in K_m$, $\notin K_i$, and $dis(t_j,t_m) \leq dis(t_j,t_h)$; and
4. $t_j \in K_m$, $\notin K_i$, but $dis(t_j,t_h) \leq dis(t_j,t_m)$.

PAM Algorithm

```
Input:
    $D = \{t_1, t_2, ..., t_n\}$    // Set of elements
    $A$    // Adjacency matrix showing distance between elements.
    $k$    // Number of desired clusters.
Output:
    $K$    // Set of clusters.
PAM Algorithm:
    arbitrarily select $k$ medoids from $D$;
    repeat
        for each $t_h$ not a medoid do
            for each medoid $t_i$ do
                calculate $TC_{ih}$;
        find $i, h$ where $TC_{ih}$ is the smallest;
        if $TC_{ih} < 0$ then
            replace medoid $t_i$ with $t_h$;
    until $TC_{ih} \geq 0$;
    for each $t_i \in D$ do
        assign $t_i$ to $K_j$ where $dis(t_i, t_j)$ is the smallest over all medoids;
```

PAM Example



CLARA and CLARANS

CLARA: Clustering LARge Applications

1. Determine set of medoids M by using a sample D' of the database D, where |D'| << |D|
2. Do clustering of D based on the set of medoids M

CLARANS: Clustering Large Applications based on RANdomized Search

- Improved CLARA clustering algorithm which uses several randomly picked samples D' instead of only one

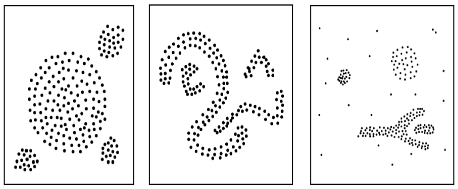
DBSCAN: Density Based Spatial Clustering of Applications with Noise

- Based on the notion of density
- Outliers will not effect creation of clusters

- Two parameters which are input:
  - ***MinPts*** – minimum number of points in cluster
  - ***Eps*** – for each point in cluster there must be another point in it less than this distance away

- The number of clusters, k, is not input but is determined by the algorithm itself.

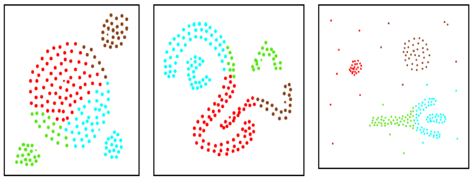
DBSCAN Example



database 1 database 2 database 3

figure 1: Sample databases

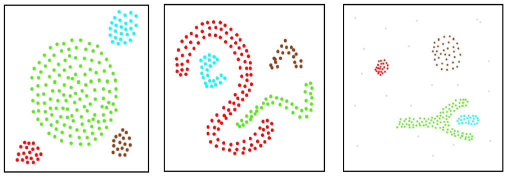
DBSCAN Example



database 1 database 2 database 3

figure 5: Clusterings discovered by CLARANS

DBSCAN Example



database 1 database 2 database 3

figure 6: Clusterings discovered by 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 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
  - A noise point is any point that is not a core point or a border point.

DBSCAN Density Concepts

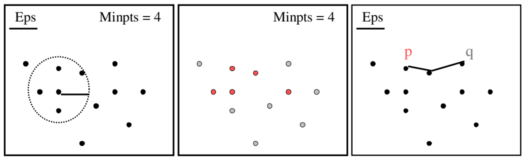
Eps-neighborhood: Points within Eps distance of a point.

Core point: Point whose Eps-neighborhood is dense enough (MinPts) and forms the main portion of some cluster

Directly density-reachable: A point p is directly density-reachable from a point q if the distance between them is small (Eps) and q is a core point.

Density-reachable: A point is density-reachable form another point if there is a path from one to the other consisting of only core points.

Density Concepts used in DBSCAN



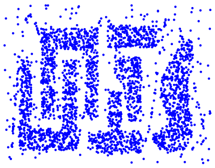
Eps-neighborhood Core points & Border points Density reachable

DBSCAN Algorithm

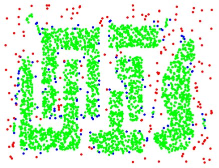
- Eliminate noise points
- Perform clustering on the remaining points

```
$current\_cluster\_label \leftarrow 1$
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



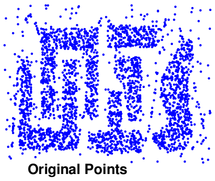
Original Points



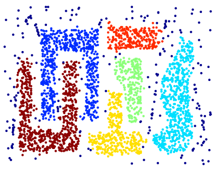
Point types: core, border and noise

Eps = 10, MinPts = 4

When DBSCAN Works Well



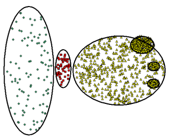
Original Points



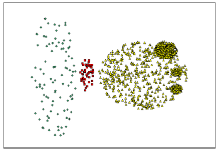
Clusters

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

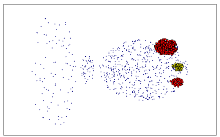
When DBSCAN Does NOT Work Well



Original Points



(MinPts=4, Eps=9.75).

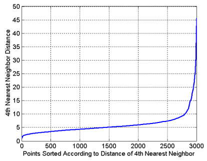


(MinPts=4, Eps=9.92)

- Varying densities
- High-dimensional data

DBSCAN: Determining EPS and MinPts

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



Clustering Large Databases

- Most clustering algorithms assume a large data structure which is memory resident
- Clustering may be performed first on a sample of the database, then applied to the entire database
- Algorithms
  - BIRCH
  - CURE

Clustering Large Databases: Desired Features

- One scan (or less) of DB
- Online
  - Able to report clustering status while running
- Suspendable, stopable, resumable
- Incremental
  - Able to handle dynamic updates
- Able to work with limited main memory
- Different techniques to scan the DB (e.g. use sampling)
- Process each tuple once

BIRCH: Balanced Iterative Reducing and Clustering using Hierarchies

- Incremental, hierarchical, one DB scan
- Saves clustering information in a balanced tree
- Each entry in the tree contains summary information about one cluster
- New nodes inserted in closest entry in tree
- Adapts to main memory size by changing the threshold value
  - Larger threshold ⇒ Smaller tree

Clustering Feature

- CF Triple: (N,$\overrightarrow{LS}$,SS)
  - N: Number of points in cluster
  - $\overrightarrow{LS}$: Sum of points in the cluster
  - SS: Sum of squares of points in the cluster
- CF Tree
  - Balanced search tree
  - Node has CF triple for each child
  - Leaf node represents cluster and has CF value for each subcluster in it
  - Subcluster has maximum diameter

BIRCH Algorithm

```
Input:
    $D = \{t_1, t_2, ..., t_n\}$    // Set of elements
    $T$    // Threshold for CF tree construction.
Output:
    $K$    // Set of clusters.
BIRCH Clustering Algorithm:
    for each $t_i \in D$ do
        determine correct leaf node for $t_i$ insertion;
        if threshold condition is not violated then
            add $t_i$ to cluster and update CF triples;
        else
            if room to insert $t_i$ then
                insert $t_i$ as single cluster and update CF triples;
            else
                split leaf node and redistribute CF features;
```

BIRCH: Improving Clusters

1. Create initial CF tree using Algorithm.
   If there is insufficient memory to construct the CF tree with a given threshold, the threshold value is increased and a new smaller CF tree is constructed.

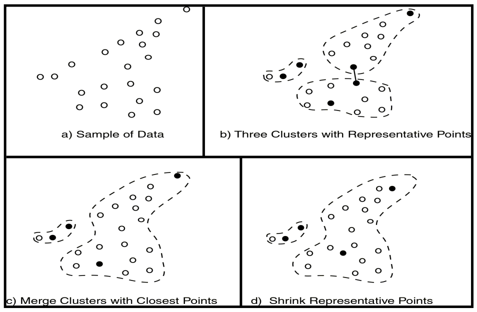
2. Apply another global clustering approach applied to the leaf nodes in the CF tree. Here each leaf node is treated as a single point for clustering.

3. The last phase (which is optional) re-clusters all points by placing them in the cluster which has the closest centroid.

CURE: Clustering Using REpresentatives

- Use many points to represent a cluster instead of only one
- Representative points need to be well scattered
- Handles outliers well – removes clusters which
  - Either grow very slowly
  - Or contain very few points

CURE Approach



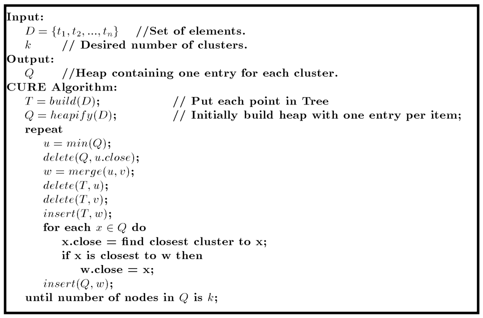
a) Sample of Data

b) Three Clusters with Representative Points

c) Merge Clusters with Closest Points

d) Shrink Representative Points

CURE Algorithm

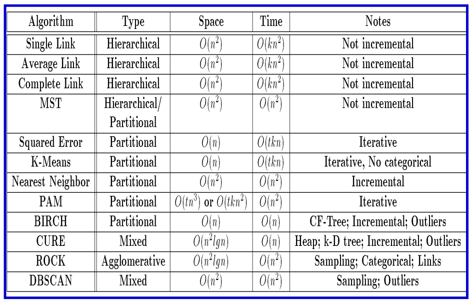


```
Input:
    $D = \{t_1, t_2, ..., t_n\}$    //Set of elements.
    $k$        // Desired number of clusters.
Output:
    $Q$        //Heap containing one entry for each cluster.
CURE Algorithm:
    $T = build(D)$;                  // Put each point in Tree
    $Q = heapify(D)$;                // Initially build heap with one entry per item;
    repeat
        $u = min(Q)$;
        $delete(Q, u.close)$;
        $w = merge(u, v)$;
        $delete(T, u)$;
        $delete(T, v)$;
        $insert(T, w)$;
        for each $x \in Q$ do
            x.close = find closest cluster to x;
            if x is closest to w then
                w.close = x;
        $insert(Q, w)$;
    until number of nodes in $Q$ is $k$;
```

CURE for Large Databases

1. Obtain a sample of the database
2. Partition the sample into *p* partitions
3. Partially cluster the points in each partition
4. Remove outliers based on size of cluster
5. Completely cluster all data in the samples (representatives)
6. Cluster entire database on disk using *c* points to represent each cluster. An item in the database is placed in the cluster which has the closest representative point to it.

Comparison of Clustering Techniques



Algorithm	Type	Space	Time	Notes
Single Link	Hierarchical	$O(n^2)$	$O(kn^2)$	Not incremental
Average Link	Hierarchical	$O(n^2)$	$O(kn^2)$	Not incremental
Complete Link	Hierarchical	$O(n^2)$	$O(kn^2)$	Not incremental
MST	Hierarchical/ Partitional	$O(n^2)$	$O(n^2)$	Not incremental
Squared Error	Partitional	$O(n)$	$O(tkn)$	Iterative
K-Means	Partitional	$O(n)$	$O(tkn)$	Iterative, No categorical
Nearest Neighbor	Partitional	$O(n^2)$	$O(n^2)$	Incremental
PAM	Partitional	$O(tn^3)$ or $O(tkn^2)$	$O(n^2)$	Iterative
BIRCH	Partitional	$O(n)$	$O(n)$	CF-Tree; Incremental; Outliers
CURE	Mixed	$O(n^2 lgn)$	$O(n)$	Heap; k-D tree; Incremental; Outliers
ROCK	Agglomerative	$O(n^2 lgn)$	$O(n^2)$	Sampling; Categorical; Links
DBSCAN	Mixed	$O(n^2)$	$O(n^2)$	Sampling; Outliers

Clustering Validity

- For supervised classification we have a variety of measures to evaluate how good our model is
  - Accuracy, precision, recall
- For cluster analysis, the analogous question is how to evaluate the "goodness" of the resulting clusters?
- But "clusters are in the eye of the beholder"!