## Overview

Cluster analysis divides data into groups (clusters) that are meaningful, useful, or both. Here we consider following two issues

(1) Different ways to group a set of objects into a set of clusters, and

(2) Types of clusters.

### Different Types of Clusterings

An entire collection of clusters is commonly referred to as a clustering.

**Hierarchical versus Partitional** The most commonly discussed distinction among different types of clusterings is whether the set of clusters is nested or un nested, or in more traditional terminology, hierarchical or partitional. A partitional clnstering is simply a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset. In the following Figures (b-d) is a partitional clustering.

If we permit clusters to have sub clusters, then we obtain a hierarchical clustering, which is a set of nested clusters that are organized as a tree. Each node (cluster) in the tree (except for the leaf nodes) is the union of its children (subclusters), and the root of the tree is the cluster containing all the objects. The clusters shown in following Figures (a-d), when taken in that order, also form a hierarchical (nested) clustering with, respectively, 1, 2, 4, and 6 clusters on each level.

**Exclusive versus Overlapping versus Fuzzy** The clusterings shown in following figure Figure are all exclusive, as they assign each object to a single cluster. There are many situations in which a point could reasonably be placed in more than one cluster, and these situations are better addressed by non-exclusive clustering. In the most general sense, an overlapping or non-exclusive clustering is used to reflect the fact that an object can *simultaneously* belong to more than one group (class). For instance, a person at a university can be both an enrolled student and an employee of the university. In a fuzzy clustering, every object belongs to every cluster with a membership weight that is between 0 (absolutely doesn't belong) and 1 (absolutely belongs). In other words, clusters are treated as fuzzy sets. (Mathematically, a fuzzy set is one in which an object belongs to any set with a weight that is between 0 and 1.

**Complete versus Partial** A complete clustering assigns every object to a cluster, whereas a partial clustering does not. The motivation for a partial clustering is that some objects in a data set may not belong to well-defined groups. Many times objects in the data set may represent noise, outliers, or "uninteresting background."

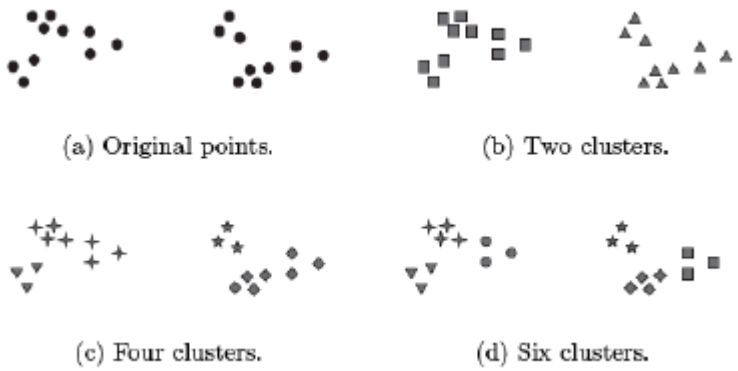



**Figure** Different ways of clustering the same set of points.

**Different Types of Clusters**

**Well-Separated** A cluster is a set of objects in which each object is closer (or more similar) to every other object in the cluster than to any object not in the cluster. Sometimes a threshold is used to specify that all the objects in a cluster must be sufficiently close (or similar) to one another. This idealistic definition of a cluster is satisfied only when the data contains natural clusters that are quite far from each other. The following Figure (a) gives an example of well separated clusters that consists of two groups of points in a two-dimensional space.

**Prototype-Based** A cluster is a set of objects in which each object is closer (more similar) to the prototype that defines the cluster than to the prototype of any other cluster. For data with continuous attributes, the prototype of a cluster is often a centroid, i.e., the average (mean) of all the points in the cluster. When a centroid is not meaningful, such as when the data has categorical attributes, the prototype is often a medoid, Figure (b) shows an example of center-based clusters.

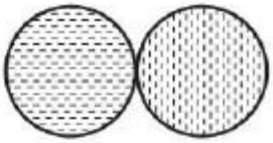
**Graph-Based** If the data is represented as a graph, where the nodes are objects and the links represent connections among objects then a cluster can be defined as a connected component; i.e., a group of objects that are connected to one another, but that have no connection to objects outside the group. An important example of graph-based clusters are contiguity-based clusters, where two objects are connected only if they are within a specified distance of each other. This implies that each object in a contiguity-based cluster is closer to some other object in the cluster than to any point in a different cluster. Figure (c) shows an example of such clusters for two-dimensional points.

**Density-Based** A cluster is a dense region of objects that is surrounded by a region of low density. Figure ( d) shows some density-based clusters for data created by adding noise to the data of Figure (c). The two circular clusters are not merged, as in Figure (c), because the bridge between them fades into the noise. Likewise, the curve that is present in Figure(c).

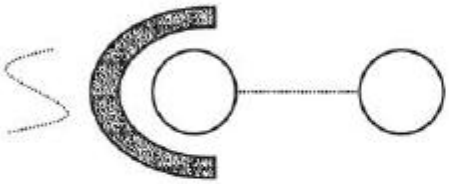
**Shared-Property** (Conceptual Clusters) More generally, we can define a cluster as a set of objects that share some property Consider the clusters shown in Figure (e). A

triangular area (cluster) is adjacent to a rectangular one, and there are two intertwined circles (clusters). In both cases, a clustering algorithm would need a very specific concept of a cluster to successfully detect these clusters. The process of finding such clusters is called conceptual clustering.

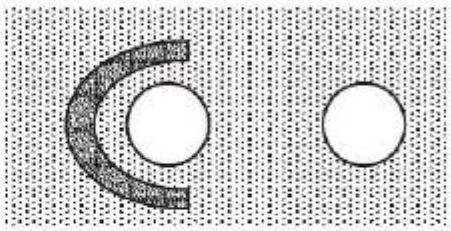
(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.



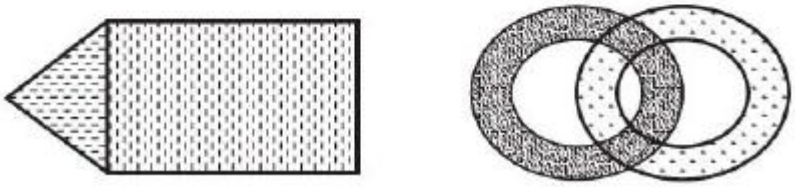
(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.



(c) Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster.



(d) Density-based clusters. Clusters are regions of high density separated by regions of low density.



(e) Conceptual clusters. Points in a cluster share some general property that derives from the entire set of points. (Points in the intersection of the circles belong to both.)

Figure. Different types of clusters as illustrated by sets of two-dimensional points.

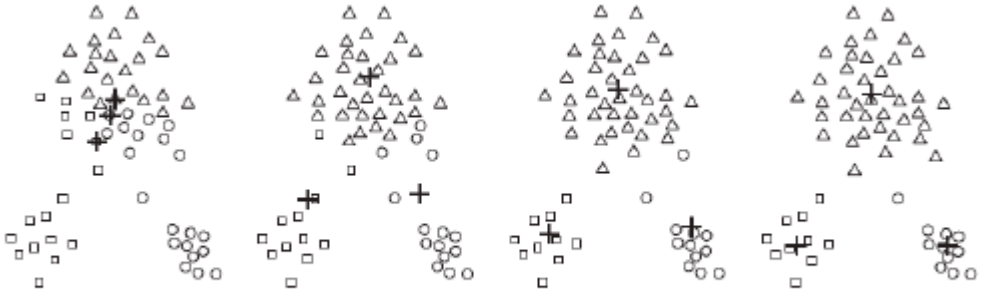
## The Basic K-means Algorithm

The K-means clustering technique is simple, and we begin with a description of the basic algorithm. We first choose *K* initial centroids, where *K* is a user specified parameter, namely, the number of clusters desired. Each point *is* then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster. The centroid of each cluster is then updated based on the points assigned to the cluster. We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroids remain the same.

K-means is formally described by Algorithm

1: Select $K$ points as initial centroids.
2: **repeat**
3: Form $K$ clusters by assigning each point to its closest centroid.
4: Recompute the centroid of each cluster.
5: **until** Centroids do not change.

The operation of K-means is illustrated in following Figure, which shows how, starting from three centroids, the final clusters are found in four assignment-update steps. In these and other figures displaying K-means clustering, each subfigure shows (1) the centroids at the start of the iteration and (2) the assignment of the points to those centroids. The centroids are indicated by the "+" symbol; all points belonging to the same cluster have the same marker shape.



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

**Figure**. Using the K·means algorithm to find three clusters in sample data.

In the first step, shown in Figure (a), points are assigned to the initial centroids, which are all in the larger group of points. For this example, we use the mean as the centroid. After points are assigned to a centroid the centroid is then updated. Again, the figure for each step shows the centroid at the beginning of the step and the assignment of points to those centroids. In the second step, points are assigned to the updated centroids, and the centroids are updated again. In steps 2, 3, and 4, which are shown in Figures (b), (c), and (d), respectively, two of the centroids move to the two small groups of points at the bottom of the figures. When the K-means algorithm terminates in Figure (d), because no more changes occur, the centroids have identified the natural groupings of points.

**Assigning Points to the Closest Centroid**

To assign a point to the closest centroid, we need a proximity measure that quantifies the notion of "closest" for the specific data under consideration. Euclidean (L2) distance is often used for data points in Euclidean space, while cosine similarity is more appropriate for documents. However, there may be several types of proximity measures that are appropriate for a given type of data. For example, Manhattan (L1) distance can be used for Euclidean data, while the Jaccard measure is often employed for documents. Usually, the similarity measures used for K-means are relatively simple since the algorithm repeatedly calculates the similarity of each point to each centroid.

Data in Euclidean Space consider data whose proximity measure is Euclidean distance. For our objective function, which measures the quality of a clustering, we use the sum of the squared error (SSE), which is also known *as* scatter. In other words, we calculate the error of each data point, i.e., its Euclidean distance to the closest centroid, and then compute the total sum of the squared errors. Given two different sets of clusters that are produced by two different runs of K-means, we prefer the one with the smallest squared error since this means that the prototypes (centroids) of this clustering are a better representation of the points in their cluster. Using the notation in
Table, the SSE is formally defined as follows:

| Symbol | Description |
|---|---|
| $\mathbf{x}$ | An object. |
| $C_i$ | The $i^{th}$ cluster. |
| $\mathbf{c}_i$ | The centroid of cluster $C_i$. |
| $\mathbf{c}$ | The centroid of all points. |
| $m_i$ | The number of objects in the $i^{th}$ cluster. |
| $m$ | The number of objects in the data set. |
| $K$ | The number of clusters. |

**Table**. Table of notation.

$$\text{SSE} = \sum_{i=1}^{K} \sum_{\mathbf{x} \in C_i} dist(\mathbf{c}_i, \mathbf{x})^2$$

Where *dist* is the standard Euclidean (L,,) distance between two objects in
Euclidean space using the notation in Table, the centroid (mean) of the *ith* cluster is defined by Equation

$$\mathbf{c}_i = \frac{1}{m_i} \sum_{\mathbf{x} \in C_i} \mathbf{x}$$

To illustrate, the centroid of a cluster containing the three two-dimensional points, (1,1), (2,3), and (6,2), is (1 + 2 + 6)/3,(1 + 3 + 2)/3) = (3,2). A common approach is to choose the initial centroids randomly, but the resulting clusters are often poor.
(Poor Initial Centroids). Randomly selected initial centroids may be poor.

**Limits of Random Initialization**

One technique that is commonly used to address the problem of choosing initial centroids is to perform multiple runs, each with a different set of randomly chosen initial centroids, and then select the set of clusters with the minimum SSE.

**K-means: Additional Issues**

**Handling Empty Clusters**

One of the problems with the basic K-means algorithm given earlier is that empty clusters can be obtained if no points are allocated to a cluster during the assignment step. If this happens, then a strategy is needed to choose a replacement centroid, since otherwise, the squared error will be larger than necessary. One approach is to choose the point that is farthest away from any current centroid.

Another approach is to choose the replacement centroid from the cluster that has the highest SSE. This will typically split the cluster and reduce the overall SSE of the clustering. If there are several empty clusters, then this process can be repeated several times.

**Outliers**

When the squared error criterion is used, outliers can unduly influence the clusters that are found. In particular, when outliers are present, the resulting cluster centroids (prototypes) may not be as representative. Because of this, it is often useful to discover outliers and eliminate them beforehand. An obvious issue is how to identify outliers.

If we use approaches that remove outliers before clustering, we avoid clustering points that will not cluster well. Alternatively, outliers can also be identified in a postprocessing step. For instance, we can keep track of the SSE contributed by each point, and eliminate those points with unusually high contributions, especially over multiple runs. Also, we may want to eliminate small clusters since they frequently represent groups of outliers.

**Reducing the SSE with Postprocessing**

An obvious way to reduce the SSE is to find more clusters, i.e., to use a larger *K*. However, in many cases, we would like to improve the SSE, but don't want to increase the number of clusters. Two strategies that decrease the total SSE by increasing the number of clusters are the following:

**Split a cluster:** The cluster with the largest SSE is usually chosen, but we could also split the cluster with the largest standard deviation for one particular attribute.

**Introduce a new cluster centroid:** Often the point that is farthest from any cluster center is chosen. We can easily determine this if we keep track of the SSE contributed by each point. Another approach is to choose randomly from all points or from the points with the highest SSE.

Two strategies that decrease the number of clusters, while trying to minimize the increase in total SSE, are the following:

**Disperse a cluster:** This is accomplished by removing the centroid that corresponds to the cluster and reassigning the points to other clusters. Ideally, the cluster that is dispersed should be the one that increases the total SSE the least.

**Merge two clusters:** The clusters with the closest centroids are typically chosen, although another, perhaps better, approach is to merge the two clusters that result in the smallest increase in total SSE. These two merging strategies are the same ones that are used in the hierarchical.

**Cluster Analysis:**

**Updating Centroids Incrementally**

Instead of updating cluster centroids after all points have been assigned to a cluster, the centroids can be updated incrementally, after each assigmnent of a point to a cluster. Notice that this requires either zero or two updates to cluster centroids at

each step, since a point either moves to a new cluster (two updates) or stays in its current cluster (zero updates). Using an incremental update strategy guarantees that empty clusters are not produced since all clusters start with a single point, and if a cluster ever has only one point, then that point will always be reassigned to the same cluster.

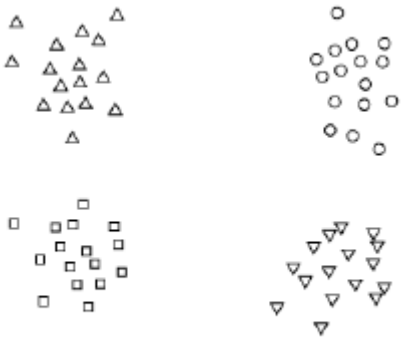
**Bisecting K-means**

The bisecting K-means algorithm is a straightforward extension of the basic K-means algorithm that is based on a simple idea: to obtain *K* clusters, split the set of all points into two clusters, select one of these clusters to split, and so on, until *k* clusters have been produced. There are a number of different ways to choose which cluster to split. We can choose the largest cluster at each step, choose the one with the largest SSE, or use a criterion based on both size and SSE. Different choices result in different clusters.The details of bisecting K-means are given by Algorithm

Algorithm Bisecting K-means algorithm.

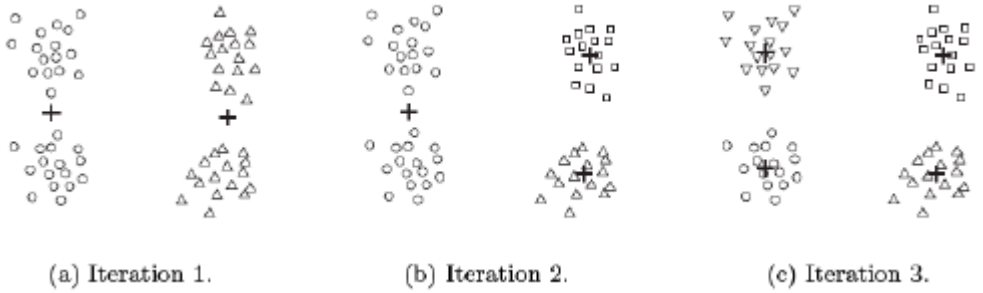
1: Initialize the list of clusters to contain the cluster consisting of all points.
2: repeat
3: Remove a cluster from the list of clusters.
4: {Perform several "trial" bisections of the chosen cluster.}
5: for i = 1 to *number of trials* do
6: Bisect the selected cluster using basic K-means.
7: end for
8: Select the two clusters from the bisection with the lowest total SSE.
9: Add these two clusters to the list of clusters.
10: until Until the list of clusters contains *K* clusters. To illustrate that

bisecting K-means is less susceptible to initialization problems, we show, in the following second Figure, how bisecting K-means finds four clusters in the data set originally shown in the following first Figure (a). In iteration 1, two pairs of clusters are found in iteration 2, the rightmost pair of clusters is split; and in iteration 3, the leftmost pair of clusters is split. Bisecting K-means has less trouble with initialization because it performs several trial bisections and takes the one with the lowest SSE, and because there are only two centroids at each step.



(a) Initial points.

Figure. Input



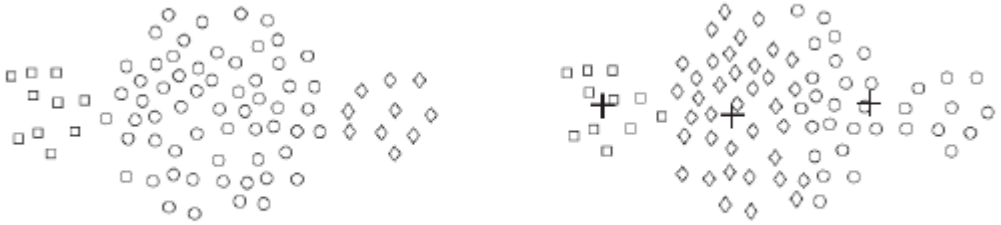
(a) Iteration 1. (b) Iteration 2. (c) Iteration 3.

**Figure**. Bisecting K-means on the four clusters example.

Finally, by recording the sequence of clusterings produced as K-means bisects clusters, we can also use bisecting K-means to produce a hierarchical clustering.

**K-means and Different Types of Clusters**

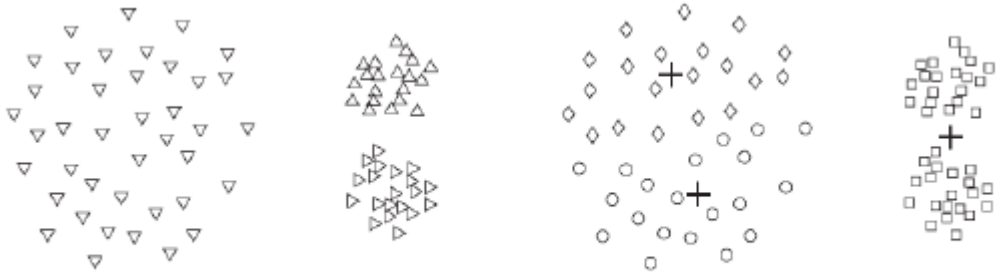
K-means and its variations have a number of limitations with respect to finding different types of clusters. In particular, K-means has difficulty detecting the "natural" clusters, when clusters have non-spherical shapes or widely different sizes or densities. This is illustrated by following Figures. In the following Figure, K-means cannot find the three natural clusters because one of the clusters is much larger than the other two, and hence, the larger cluster is broken, while one of the smaller clusters is combined with a portion of the larger cluster.



(a) Original points. (b) Three K-means clusters.

**Figure**. K·means with clusters of different size.

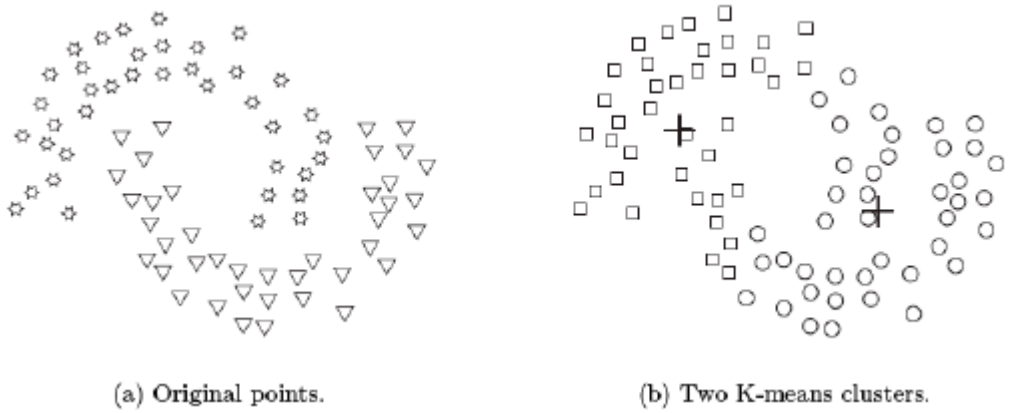
In the following Figure, K-means fails to find the three natural clusters because the two smaller clusters are much denser than the larger cluster.



(a) Original points. (b) Three K-means clusters.

**Figure**. K·rneans wrth clusters of different density.

Finally, in the following Figure, K-rneans finds two clusters that mix portions of the two natural clusters because the shape of the natural clusters is not globular.



(a) Original points. (b) Two K-means clusters.

**Figure**. K·means with non-globular clusters.

The difficulty in these three situations is that the K-means objective function is a mismatch for the kinds of clusters we are trying to find since it is minimized by globular clusters of equal size and density or by clusters that are well separated.

**Strengths and Weaknesses**

K-means is simple and can be used for a wide variety of data types. It is also quite efficient, even though multiple runs are often performed. Some variants, including bisecting K-means, are even more efficient, and are less susceptible to initialization problems.

K-means is not suitable for all types of data, It cannot handle non-globular clusters or clusters of different sizes and densities, although it can typically find pure subclusters if a large enough number of clusters is specified. K-means also has trouble clustering data that contains outliers. Outlier detection and removal can help significantly in such situations. Finally, K-means is restricted to data for which there is a notion of a center (centroid).

**K-means as an Optimization Problem**

As mentioned earlier, given an objective function such as "minimize SSE," clustering can be treated as an optimization problem. One way to solve this problem-to find a global optimum-is to enumerate all possible ways of dividing the points into clusters and then choose the set of clusters that best satisfies the objective function, e.g., that minimizes the total SSE.

**Derivation of K-means as an Algorithm to Minimize the SSE**

In this section, we show how the centroid for the K-means algorithm can be mathematically derived when the proximity function is Euclidean distance and the objective is to minimize the SSE. In mathematical terms, we seek to minimize Equation

$$\mathrm{SSE} = \sum_{i=1}^{K} \sum_{x \in C_i} (c_i - x)^2$$

Here, $C_i$, is the *ith* cluster, *x* is a point in C*i*, and $c_i$ is the mean of the *ith* cluster. We can solve for the *kth* centroid *Ck*, which minimizes above Equation, by differentiating the SSE, setting it equal to 0, and solving, as indicated below.

$$\begin{aligned}\frac{\partial}{\partial c_k}\text{SSE} &= \frac{\partial}{\partial c_k}\sum_{i=1}^{K}\sum_{x\in C_i}(c_i - x)^2 \\ &= \sum_{i=1}^{K}\sum_{x\in C_i}\frac{\partial}{\partial c_k}(c_i - x)^2 \\ &= \sum_{x\in C_k} 2 * (c_k - x_k) = 0\end{aligned}$$

$$\sum_{x\in C_k} 2 * (c_k - x_k) = 0 \Rightarrow m_k c_k = \sum_{x\in C_k} x_k \Rightarrow c_k = \frac{1}{m_k}\sum_{x\in C_k} x_k$$

Thus, as previously indicated, the best centroid for minimizing the SSE of a cluster is the mean of the points in the cluster.

**Derivation of K-means for SAE**

To demonstrate that the K-means algorithm can be applied to a variety of different objective functions, we consider how to partition the data into *k* clusters such that the sum of the Manhattan (L1) distances of points from the center of their clusters is minimized. We are seeking to minimize the sum of the Ll absolute errors (SAE) as given by the following equation, where *dist* is the Ll distance. Again, for notational simplicity, we use one-dimensional data,

$$\text{SAE} = \sum_{i=1}^{K}\sum_{x\in C_i} dist_{\text{L1}}(c_i, x)$$

We can solve for the *k'h* centroid *Ck*, which minimizes above Equation, by differentiating the SAE, setting it equal to 0, and solving.

$$\begin{aligned}\frac{\partial}{\partial c_k}\text{SAE} &= \frac{\partial}{\partial c_k}\sum_{i=1}^{K}\sum_{x\in C_i}|c_i - x| \\ &= \sum_{i=1}^{K}\sum_{x\in C_i}\frac{\partial}{\partial c_k}|c_i - x| \\ &= \sum_{x\in C_k}\frac{\partial}{\partial c_k}|c_k - x| = 0\end{aligned}$$

$$\sum_{x\in C_k}\frac{\partial}{\partial c_k}|c_k - x| = 0 \Rightarrow \sum_{x\in C_k} sign(x - c_k) = 0$$

**Agglomerative hierarchical clustering, Basic Agglomerative hierarchical clustering algorithm, Specific techniques, DBSCAN, Traditional density, center based approach, Strengths and weaknesses(Tan)**

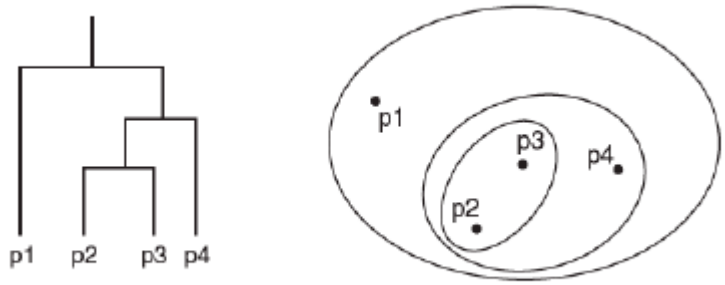
**Agglomerative hierarchical clustering**

Hierarchical clustering techniques are second important category of clustering methods. There are two basic approaches for generating a hierarchical clustering:

**Agglomerative:** Start with the points as individual clusters and, at each step, merge the closest pair of clusters. This requires defining a notion of cluster proximity.

**Divisive:** Start with one, all-inclusive cluster and, at each step, split a cluster until only singleton clusters of individual points remain, In this case, we need to decide which cluster to split at each step and how to do the splitting.

A hierarchical clustering is often displayed graphically using a tree-like diagram called a dendrogram, which displays both the cluster-sub cluster relationships and the order in which the clusters were merged (agglomerative view) or split (divisive view). The following figure shows an example of these two types of figures for a set of four two-dimensional points.




(a) Dendrogram. (b) Nested cluster diagram.

**Figure** A hierarchical clustering of four points shown as a dendrogram and as nested clusters

**Basic Agglomerative Hierarchical Clustering Algorithm**

Many agglomerative hierarchical clustering techniques are variations on a single approach: starting with individual points as clusters, successively merge the two closest clusters until only one cluster remains. This approach is expressed more formally in Algorithm as below.

Algorithm Basic agglomerative hierarchical clustering algorithm.

1: Compute the proximity matrix, if necessary.

2: repeat

3: Merge the closest two clusters.

4: Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.

5: until only one cluster remains.

For example, many agglomerative hierarchical clustering techniques, such as MIN, MAX, and Group Average, come from a graph-based view of clusters. MIN defines

cluster proximity as the proximity between the closest two points that are in different clusters, or using graph terms, the shortest edge between two nodes in different subsets of nodes.

Alternatively, MAX takes the proximity between the farthest two points in different clusters to be the cluster proximity, or using graph terms, the longest edge between **two nodes in different subsets of nodes.**

Another graph-based approach, the group average **technique, defines cluster proximity to be the average pairwise** proximities **(av**erage length of edges) of all pairs of points from different clusters. The following Figure illustrates these three approaches.

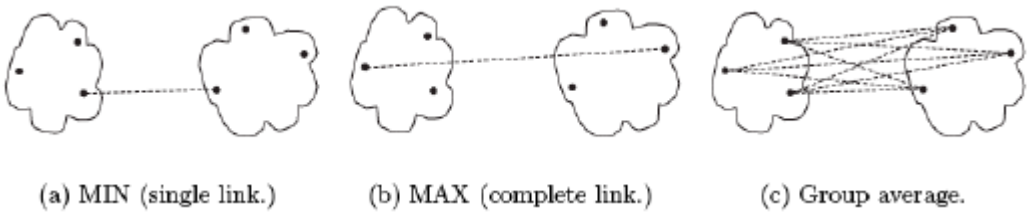



**Figure**. Graph-based definitions of cluster proximity

**Specific Techniques**

**Sample Data**

To illustrate the behavior of the various hierarchical clustering algorithms, we shall use sample data that consists of 6 two-dimensional points, which are shown in following Figure. The *x* and *y* coordinates of the points and the Euclidean distances between them are shown in following Tables.

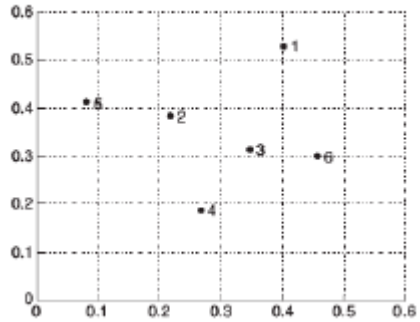



Figure. Set of 6 two-dimensional points

| Point | $x$ Coordinate | $y$ Coordinate |
|---|---|---|
| p1 | 0.40 | 0.53 |
| p2 | 0.22 | 0.38 |
| p3 | 0.35 | 0.32 |
| p4 | 0.26 | 0.19 |
| p5 | 0.08 | 0.41 |
| p6 | 0.45 | 0.30 |

Table. x,y coordinates of 6 points

| | p1 | p2 | p3 | p4 | p5 | p6 |
|---|---|---|---|---|---|---|
| p1 | 0.00 | 0.24 | 0.22 | 0.37 | 0.34 | 0.23 |
| p2 | 0.24 | 0.00 | 0.15 | 0.20 | 0.14 | 0.25 |
| p3 | 0.22 | 0.15 | 0.00 | 0.15 | 0.28 | 0.11 |
| p4 | 0.37 | 0.20 | 0.15 | 0.00 | 0.29 | 0.22 |
| p5 | 0.34 | 0.14 | 0.28 | 0.29 | 0.00 | 0.39 |
| p6 | 0.23 | 0.25 | 0.11 | 0.22 | 0.39 | 0.00 |

Table. Euclidean distance matrix for 6 points.

**Single Link or MIN**

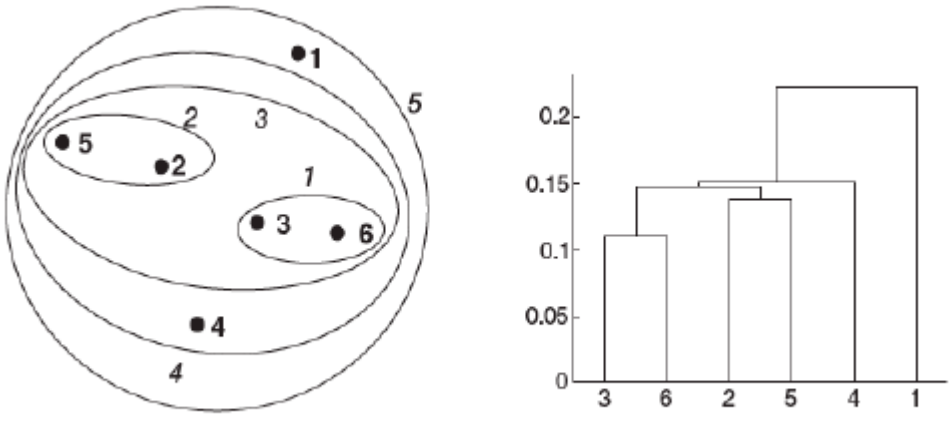
For the single link or MIN version of hierarchical clustering, the proximity of two clusters is defined as the minimum of the distance (maximum of the similarity) between any two points in the two different clusters. Using graph terminology, if you start with all points as singleton clusters and add links between points one at a time, shortest links first, then these single links combine the points into clusters. The single link technique is good at handling **non-elliptical shapes but is sensitive to noise and outliers.**

Example (Single Link). The following Figure shows the result of applying the single link technique to our example data set of six points. Figure (a) shows the nested clusters as a sequence of nested ellipses, where the numbers associated with the ellipses indicate the order of the clustering. Figure (b) shows the same information, but as a dendrogram. The height at which two clusters are merged in the dendrogram reflects the distance of the two clusters.For instance, from the above Table, we see that the distance between points 3 and 6 is 0.11, and that is the height at which they are joined into one cluster in the dendrogram. As another example, the distance between clusters {3,6} and {2,5} is given by

*dist(* {3, 6}, {2, 5}) *min(dist(3,* 2), *dist(6,* 2), *dist(3,* 5), *dist(6, 5))*

min(0.15, 0.25, 0.28, 0.39)

0.15.




(a) Single link clustering. (b) Single link dendrogram.

**Figure**. Single link clustering of the six points

Complete **Link** or MAX or **CLIQUE**

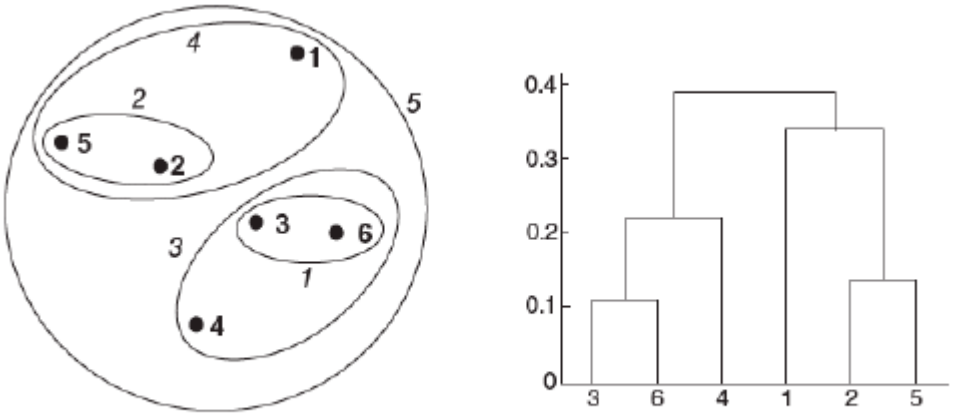
For the complete link or MAX version of hierarchical clustering, the proximity of two clusters is defined as the maximum of the distance (minimum of the similarity) between any two points in the two different clusters. Using graph terminology, if you start with all points as singleton clusters and add links between points one at a time, shortest links first, then a group of points is not a cluster until all the points in yit are completely linked, i.e., form a *clique*. Complete link is less susceptible to noise and outliers, but it can break large clusters and it favors globular shapes.

Example (Complete Link). The following Figure shows the results of applying MAX to the sample data set of six points. As with single link, points 3 and 6are merged first. However, {3,6} is merged with {4}, instead of {2, 5} or {1} **because**

*dist({3,6},{4)* =*max(dist(3,* 4), *dist(6, 4))*
max(0.15,0.22)
0.22.

*dist(* {3, 6}, {2, 5})=*max(dist(3,* 2), *dist(6,* 2), *dist(3,* 5), *dist(6, 5))*
max(0.15, 0.25, 0.28, 0.39)
0.39.

*dist({3,6},* {I}) =*max(dist(3,* 1), *dist(6, 1))*
max(0.22,0.23)
0.23.




(a) Complete link clustering. (b) Complete link dendrogram.

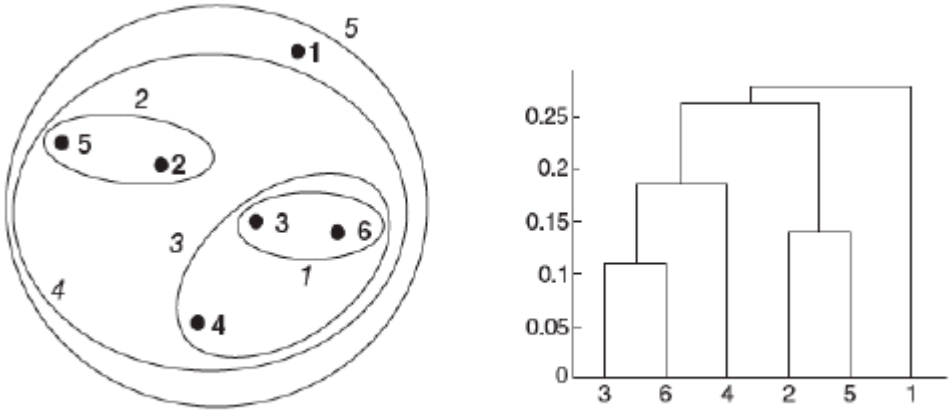
Figure. Complete link clustering of the six points

•Group Average

For the group average version of hierarchical clustering, the proximity of two clusters is defined as the average pairwise proximity among all pairs of points in the different clusters. This is an intermediate approach between the single and complete link approaches. Thus, for group average, the cluster proximty group average approach to the sample data set of six points. To illustrate**how group average works, we calculate the distance between some clusters.**

$$
\begin{aligned}
dist(\{3,6\},\{4\}) &= \max(dist(3,4), dist(6,4)) \\
&= \max(0.15, 0.22) \\
&= 0.22. \\
dist(\{3,6\},\{2,5\}) &= \max(dist(3,2), dist(6,2), dist(3,5), dist(6,5)) \\
&= \max(0.15, 0.25, 0.28, 0.39) \\
&= 0.39. \\
dist(\{3,6\},\{1\}) &= \max(dist(3,1), dist(6,1)) \\
&= \max(0.22, 0.23) \\
&= 0.23.
\end{aligned}
$$

Because *dist(* {3, 6,4}, {2, 5}) is smaller than *dist(* {3, 6,4}, {I}) and *dist(* {2, 5}, {1}),clusters {3, 6,4} and {2,5} are merged at the fourth stage.




(a) Group average clustering. (b) Group average dendrogram.

**Figure**. Group average clustering of the six points

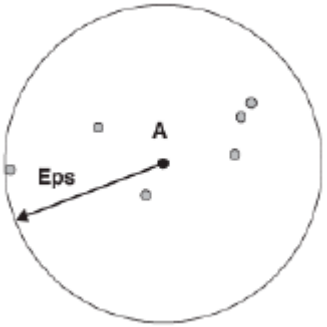
## DBSCAN

Density-based clustering locates regions of high density that are separated from one another by regions of low density. DB SCAN is a simple and effective density-based clustering algorithm. Any two core points that are close enough-within a distance *Eps* of one another-are put in the same cluster.

Algorithm 8.4 DBSCAN algorithm.

1: Label all points as core, border, or noise points.
2: Eliminate noise points.
3: Put an edge between all core points that are within *Eps* of each other.
4: Make each group of connected core points into a separate cluster.
5: Assign each border point to one of the clusters of its associated core points.

In the center-based approach, density is estimated for a particular point in the data set by counting the number of points within a specified radius, *Eps,* of that point. In the following figure, the number of points within a radius of *Eps* of point *A* is 7, including *A* itself.



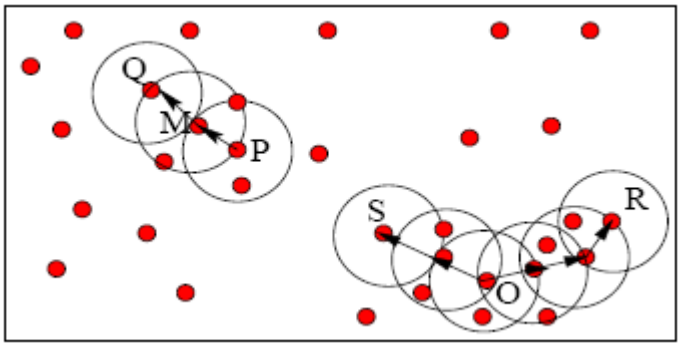

An object, within a given radius ($\epsilon$) containing no less than a minimum number of neighborhood objects ($MinPts$), is called a **core object** (with respect to a radius $\epsilon$ and a minimum number of points $MinPts$).

An object $p$ is **directly density-reachable** from object $q$ with respect to a radius $\epsilon$ and a minimum number of points $MinPts$ in a set of objects $D$ if $p$ is within the $\epsilon$-neighborhood of $q$ which contains at least a minimum number of points, $MinPts$.

An object $p$ is **density-reachable** from object $q$ with respect to $\epsilon$ and $MinPts$ in a set of objects $D$ if there is a chain of objects $p_1, \ldots, p_n, p_1 = q$ and $p_n = p$ such that for $1 \leq i \leq n$, $p_i \in D$ and $p_{i+1}$ is directly density-reachable from $p_i$ with respect to $\epsilon$ and $MinPts$.

An object $p$ is **density-connected** to object $q$ with respect to $\epsilon$ and $MinPts$ in a set of objects $D$ if there is an object $o \in D$ such that both $p$ and $q$ are density-reachable from $o$ with respect to $\epsilon$ and $MinPts$.

**Example 8.6** In Figure 8.8, based on a given $\epsilon$ represented by the radius of the circles, and let $MinPts = 3$, $M$ is directly density-reachable from $P$, and $Q$ is (indirectly) density-reachable from $P$. However, $P$ is not density-reachable from $Q$. Similarly, $R$ and $S$ are density-reachable from $O$; and $O$, $R$ and $S$ are all density-connected. ■




Density reachability and density connectivity in density-based clustering

## Strengths and Weaknesses

- Because DBSCAN uses a density-based definition of a cluster, it is relatively resistant to noise and can handle clusters of arbitrary shapes and sizes. Thus, DBSCAN can find many clusters that could not be found using K-means,

- However, DBSCAN has trouble when the clusters have widely varying densities. It also has trouble with high-dimensional data because density is more difficult to define for such data.
- DB SCAN can be expensive when the computation of nearest **neighbors requires computing all pairwise** proximities, **as is usually the case** for high-dimensional data.