An Introduction to Cluster Analysis for Data Mining

10/02/2000 11:42 AM

1. Introduction

1.1. Scope of This Paper

Cluster analysis divides data into meaningful or useful groups (clusters). If meaningful clusters are the goal, then the resulting clusters should capture the "natural" structure of the data. For example, cluster analysis has been used to group related documents for browsing, to find genes and proteins that have similar functionality, and to provide a grouping of spatial locations prone to earthquakes. However, in other cases, cluster analysis is only a useful starting point for other purposes, e.g., data compression or efficiently finding the nearest neighbors of points. Whether for understanding or utility, cluster analysis has long been used in a wide variety of fields: psychology and other social sciences, biology, statistics, pattern recognition, information retrieval, machine learning, and data mining.

The scope of this paper is modest: to provide an introduction to cluster analysis in the field of data mining, where we define data mining to be the discovery of useful, but non-obvious, information or patterns in large collections of data. Much of this paper is necessarily consumed with providing a general background for cluster analysis, but we also discuss a number of clustering techniques that have recently been developed specifically for data mining. While the paper strives to be self-contained from a conceptual point of view, many details have been omitted. Consequently, many references to relevant books and papers are provided.

1.2. What Cluster Analysis Is

Cluster analysis groups objects (observations, events) based on the information found in the data describing the objects or their relationships. The goal is that the objects in a group will be similar (or related) to one other and different from (or unrelated to) the objects in other groups. The greater the similarity (or homogeneity) within a group, and the greater the difference between groups, the "better" or more distinct the clustering.

The definition of what constitutes a cluster is not well defined, and, in many applications clusters are not well separated from one another. Nonetheless, most cluster analysis seeks as a result, a crisp classification of the data into non-overlapping groups. Fuzzy clustering, described in section 7.5, is an exception to this, and allows an object to partially belong to several groups.

To better understand the difficulty of deciding what constitutes a cluster, consider figures 1a through 1d, which show twenty points and three different ways that they can be divided into clusters. If we allow clusters to be nested, then the most reasonable interpretation of the structure of these points is that there are two clusters, each of which has three subclusters. However, the apparent division of the two larger clusters into three subclusters may simply be an artifact of the human visual system. Finally, it may not be unreasonable to say that the points form four clusters. Thus, we stress once again that the definition of what constitutes a cluster is imprecise, and the best definition depends on the type of data and the desired results.

Figure 1a: Initial points.

Figure 1b: Two clusters.

Figure 1c: Six clusters

Figure 1d: Four clusters.

1.3. What Cluster Analysis Is Not

Cluster analysis is a classification of objects from the data, where by classification we mean a labeling of objects with class (group) labels. As such, clustering does not use previously assigned class labels, except perhaps for verification of how well the clustering worked. Thus, cluster analysis is distinct from pattern recognition or the areas of statistics know as discriminant analysis and decision analysis, which seek to find rules for classifying objects given a set of pre-classified objects.

While cluster analysis can be useful in the previously mentioned areas, either directly or as a preliminary means of finding classes, there is much more to these areas than cluster analysis. For example, the decision of what features to use when representing objects is a key activity of fields such as pattern recognition. Cluster analysis typically takes the features as given and proceeds from there.

Thus, cluster analysis, while a useful tool in many areas (as described later), is normally only part of a solution to a larger problem which typically involves other steps and techniques.

2. Overview

2.1. Definitions

2.1.1. The Data Matrix

Objects (samples, measurements, patterns, events) are usually represented as points (vectors) in a multi-dimensional space, where each dimension represents a distinct attribute (variable, measurement) describing the object. For simplicity, it is normally assumed that values are present for all attributes. (Techniques for dealing with missing values are described in section 9.1.)

Thus, a set of objects is represented (at least conceptually) as an m by n matrix, where there are m rows, one for each object, and n columns, one for each attribute. This matrix has different names, e.g., pattern matrix or data matrix, depending on the particular field. Figure 2, below, provides a concrete example of some points and their corresponding data matrix.

The data is sometimes transformed before being used. One reason for this is that different attributes may be measured on different scales, e.g., centimeters and kilograms. In cases where the range of values differs widely from attribute to attribute, these differing attribute scales can dominate the results of the cluster analysis and it is common to standardize the data so that all attributes are on the same scale.

The following are some common approaches to data standardization:

(In what follows, x_i, is the i^{th} object, x_{ij} is the value of the j^{th} attribute of the i^{th} object, and x_{ij}' is the standardized attribute value.)

a) $x'_{ij} = \dfrac{x_{ij}}{\max_i |x_{ij}|}$. Divide each attribute value of an object by the maximum observed absolute value of that attribute. This restricts all attribute values to lie between –1 and 1. Often all values are positive, and thus, all transformed values lie between 0 and 1.

b) $x'_{ij} = \dfrac{x_{ij} - \mu_j}{\sigma_j}$. For each attribute value subtract off the mean, μ_j, of that attribute and then divide by the attribute's standard deviation, σ_j. If the data are normally distributed, then most attribute values will lie between –1 and 1 [KR90]. ($\mu_j = \dfrac{1}{m}\sum_{i=1}^{m} x_{ij}$ is the mean of the j^{th} feature, $\sigma_j = \dfrac{1}{m}\sqrt{\sum_{i=1}^{m}(x_{ij} - \mu_j)^2}$ is the standard deviation of the j^{th} feature.)

c) $x'_{ij} = \dfrac{x_{ij} - \mu_j}{\sigma_j^A}$. For each attribute value subtract off the mean of that attribute and divide by the attribute's absolute deviation, σ_j^A, [KR90]. Typically, most attribute values will lie between –1 and 1. ($\sigma_j^A = \dfrac{1}{m}\sum_{i=1}^{m}|x_{ij} - \mu_j|$ is the absolute standard deviation of the j^{th} feature.)

The first approach, (a), may not produce good results unless the attributes are uniformly distributed, while both the first and second approaches, (a) and (b), are sensitive to outliers. The third approach, (c), is the most robust in the presence of outliers, although an attempt to eliminate outliers is sometimes made before cluster analysis begins.

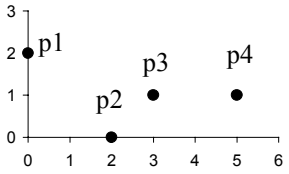
Another reason for initially transforming the data is to reduce the number of dimensions, particularly if the initial number of dimensions is large. (The problems caused by high dimensionality are discussed more fully in section 6.2.7.) This can be accomplished by discarding features that show little variation or that are highly correlated with other features. (Feature selection is a complicated subject in its own right.)

Another approach is to project points from a higher dimensional space to a lower dimensional space. Typically this is accomplished by applying techniques from linear algebra, which are based on the eigenvalue decomposition or the singular value decomposition of a data matrix or normalized data matrix. Examples of such techniques are Principal Component Analysis [DJ88] or Latent Semantic Indexing [FO95] (section 7.2.2).

2.1.2. The Proximity Matrix

While cluster analysis sometimes uses the original data matrix, many clustering algorithms use a similarity matrix, S, or a dissimilarity matrix, D. For convenience, both matrices are commonly referred to as a proximity matrix, P. A proximity matrix, P, is an m by m matrix containing all the pairwise dissimilarities or similarities between the objects being considered. If x_i and x_j are the i^{th} and j^{th} objects, respectively, then the entry

at the i^{th} row and j^{th} column of the proximity matrix is the similarity, s_{ij}, or the dissimilarity, d_{ij}, between x_i and x_j. For simplicity, we will use p_{ij} to represent either s_{ij} or d_{ij}. Figure 2 shows four points and the corresponding data and proximity (distance) matrices. More examples of dissimilarity and similarity will be provided shortly.



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

	p1	p2	p3	p4
p1	0.000	2.828	3.162	5.099
p2	2.828	0.000	1.414	3.162
p3	3.162	1.414	0.000	2.000
p4	5.099	3.162	2.000	0.000

Figure 2. Four points and their corresponding data and proximity (distance) matrices.

For completeness, we mention that objects are sometimes represented by more complicated data structures than vectors of attributes, e.g., character strings. Determining the similarity (or differences) of two objects in such a situation is more complicated, but if a reasonable similarity (dissimilarity) measure exists, then a clustering analysis can still be performed. In particular, clustering techniques that use a proximity matrix are unaffected by the lack of a data matrix.

2.1.3. The Proximity Graph

A proximity matrix defines a weighted graph, where the nodes are the points being clustered, and the weighted edges represent the proximities between points, i.e., the entries of the proximity matrix. While this proximity graph can be directed, which corresponds to an asymmetric proximity matrix, most clustering methods assume an undirected graph. Relaxing the symmetry requirement can be useful for clustering or pattern recognition, but we will assume undirected proximity graphs (symmetric proximity matrices) in our discussions.

From a graph point of view, clustering is equivalent to breaking the graph into connected components, one for each cluster. Likewise, many issues related to clustering can be cast in graph-theoretic terms, e.g., the issues of cluster cohesion and the degree of coupling with other clusters can be measured by the number and strength of links between and within clusters. Also, many clustering techniques, e.g., minimum spanning tree (section 4.2.1), single link (section 4.2.5), and complete link (section 4.2.6), are most naturally described using a graph representation.

Some clustering algorithms, e.g., Chameleon (section 4.2.12), first sparsify the proximity graph. Sparsification reduces the connectivity of nodes by breaking some of the links of the proximity graph. (This corresponds to setting a proximity matrix entry to 0 or ∞, depending on whether we are using similarities or dissimilarities, respectively.) See section 4.4.4 for details.

2.1.4. Some Working Definitions of a Cluster

As mentioned above, the term, cluster, does not have a precise definition. However, several working definitions of a cluster are commonly used.

1) **Well-Separated Cluster Definition**: A cluster is a set of points such that any point in a cluster is closer (or more similar) to every other point in the cluster than to any point not in the cluster. Sometimes a threshold is used to specify that all the points in a cluster must be sufficiently close (or similar) to one another.

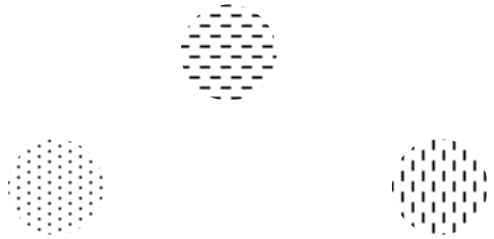


Figure 3: Three well-separated clusters of 2 dimensional points.

However, in many sets of data, a point on the edge of a cluster may be closer (or more similar) to some objects in another cluster than to objects in its own cluster. Consequently, many clustering algorithms use the following criterion.

2) **Center-based Cluster Definition**: A cluster is a set of objects such that an object in a cluster is closer (more similar) to the "center" of a cluster, than to the center of any other cluster. The center of a cluster is often a centroid, the average of all the points in the cluster, or a medoid, the most "representative" point of a cluster.

Figure 4: Four center-based clusters of 2 dimensional points.

3) **Contiguous Cluster Definition (Nearest neighbor or Transitive Clustering)**: A cluster is a set of points such that a point in a cluster is closer (or more similar) to one or more other points in the cluster than to any point not in the cluster.

Figure 5: Eight contiguous clusters of 2 dimensional points.

4) Density-based definition: A cluster is a dense region of points, which is separated by low-density regions, from other regions of high density. This definition is more often used when the clusters are irregular or intertwined, and when noise and outliers are present. Note that the contiguous definition would find only one cluster in figure 6. Also note that the three curves don't form clusters since they fade into the noise, as does the bridge between the two small circular clusters.

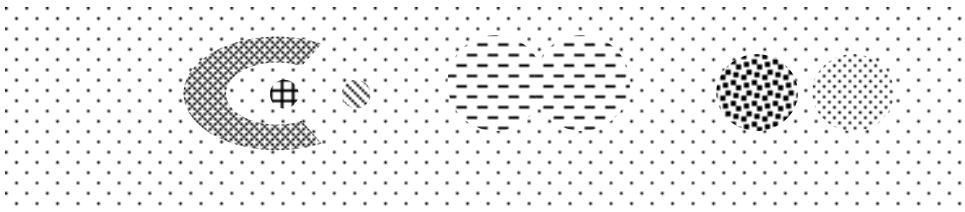


Figure 6: Six dense clusters of 2 dimensional points.

5) **Similarity-based Cluster definition**: A cluster is a set of objects that are "similar", and objects in other clusters are not "similar." A variation on this is to define a cluster as a set of points that together create a region with a uniform local property, e.g., density or shape.

2.2. Measures (Indices) of Similarity and Dissimilarity

2.2.1. Proximity Types and Scales

The attributes of the objects (or their pairwise similarities and dissimilarities) can be of different data types and can be measured on different data scales.

The different types of attributes are

1) **Binary** (two values)
2) **Discrete** (a finite number of values)
3) **Continuous** (an effectively infinite number of values)

The different data scales are

1) **Qualitative**
   a) **Nominal** – the values are just different names.
      Example 1: Zip Codes
      Example 2: Colors: white, black, green, red, yellow, etc.
      Example 3: Sex: male, female
      Example 4: 0 and 1 when they represent Yes and No.
   b) **Ordinal** – the values reflect an ordering, nothing more.
      Example 1: Good, Better, Best
      Example 2: Colors ordered by the spectrum.
      Example 3: 0 and 1 are often considered to be ordered when used as Boolean values (false/true), with $0 < 1$.

2) **Quantitative**
   a) **Interval** – the difference between values is meaningful, i.e., a unit of measurement exits.
      Example 1: On a scale of 1 to 10 rate the following potato chips.
   b) **Ratio** – the scale has an absolute zero so that ratios are meaningful.
      Example 1: The height, width, and length of an object.
      Example 2: Monetary quantities, e.g., salary or profit.
      Example 3: Many physical quantities like electrical current, pressure, etc.

Data scales and types are important since the type of clustering used often depends on the data scale and type.

2.2.2. Requirements on Similarity and Dissimilarity Measures

Proximities are normally subject to a number of requirements [DJ88]. We list these below. Recall that p_{ij} is the proximity between points, x_i and x_j.

1) **(a) For a dissimilarity: $p_{ii} = 0$ for all i. (Points aren't different from themselves.)**
   (b) For a similarity: $p_{ii} > \max p_{ij}$ (Points are most similar to themselves.)
   (Similarity measures are often required to be between 0 and 1, and in that case $p_{ii} = 1$, is the requirement.)

2) **$p_{ij} = p_{ji}$ (Symmetry)**
   This implies that the proximity matrix is symmetric. While this is typical, it is not always the case. An example is a confusion matrix, which implicitly defines a similarity between entities that is measured by how often various entities, e.g., characters, are misclassified as one another. Thus, while 0's and O's are confused with each other, it is not likely that 0's are mistaken for O's at the same rate that O's are mistaken for 0's.

3) **$p_{ij} \geq 0$ for all i and j (Positivity)**
   Additionally, if the proximity measure is real-valued and is a true metric in a mathematical sense, then the following two conditions also hold in addition to conditions 2 and 3.

4) **$p_{ij} = 0$ only if $i = j$.**

5) **$p_{ik} \leq p_{ij} + p_{jk}$ for all i, j, k. (Triangle inequality.)**

Dissimilarities that satisfy conditions 1-5 are called distances, while "dissimilarity" is a term commonly reserved for those dissimilarities that satisfy only conditions 1-3.

2.2.3. Common Proximity Measures

2.2.3.1. Distance Measures

The most commonly used proximity measure, at least for ratio scales (scales with an absolute 0) is the Minkowski metric, which is a generalization of the normal distance between points in Euclidean space. It is defined as

$$p_{ij} = \left(\sum_{k=1}^{d} \left| x_{ik} - x_{jk} \right|^{r} \right)^{1/r}$$

where, r is a parameter, d is the dimensionality of the data object, and x_{ik} and x_{jk} are, respectively, the k^{th} components of the i^{th} and j^{th} objects, x_i and x_j.

The following is a list of the common Minkowski distances for specific values of r.

1) $r = 1$. City block (Manhattan, taxicab, L_1 norm) distance.
A common example of this is the Hamming distance, which is just the number of bits that are different between two binary vectors.

2) $r = 2$. Euclidean distance. The most common measure of the distance between two points.

3) $r \to \infty$. "supremum" (L_{max} norm, L_∞ norm) distance.
This is the maximum difference between any component of the vectors.

Figure 7 gives the proximity matrices for L1, L2 and L_∞, respectively using the given data matrix which we copied from an earlier example.

point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

L2	p1	p2	p3	p4
p1	0.000	2.828	3.162	5.099
p2	2.828	0.000	1.414	3.162
p3	3.162	1.414	0.000	2.000
p4	5.099	3.162	2.000	0.000

L1	p1	p2	p3	p4
p1	0.000	4.000	4.000	6.000
p2	4.000	0.000	2.000	4.000
p3	4.000	2.000	0.000	2.000
p4	6.000	4.000	2.000	0.000

L_∞	p1	p2	p3	p4
p1	0.000	2.000	3.000	5.000
p2	2.000	0.000	1.000	3.000
p3	3.000	1.000	0.000	2.000
p4	5.000	3.000	2.000	0.000

Figure 7. Data matrix and the L1, L2, and L_∞ proximity matrices.

The r parameter should not be confused with the dimension, d. For example, Euclidean, Manhattan and supremum distances are defined for all values of d, 1, 2, 3, …, and specify different ways of combining the differences in each dimension (attribute) into an overall distance.

2.2.3.2. Ordinal Measures

Another common type of proximity measure is derived by ranking the distances between pairs of points from 1 to m * (m - 1) / 2. This type of measure can be used with most types of data and is often used with a number of the hierarchical clustering algorithms that are discussed in section 4.2. Figure 8 shows how the L2 proximity matrix of figure 7 would appear as an ordinal proximity matrix.

L2	p1	p2	p3	p4
p1	0.000	2.828	3.162	5.099
p2	2.828	0.000	1.414	3.162
p3	3.162	1.414	0.000	2.000
p4	5.099	3.162	2.000	0.000

ordinal	p1	p2	p3	p4
p1	0	3	4	5
p2	3	0	1	4
p3	4	1	0	2
p4	5	4	2	0

Figure 8. An L2 proximity matrix and the corresponding ordinal proximity matrix.

2.2.3.3. Similarity Measures Between Binary Vectors

There are many measures of similarity between binary vectors. These measures are referred to as similarity coefficients, and have values between 0 and 1. A value of 1 indicates that the two vectors are completely similar, while a value of 0 indicates that the vectors are not at all similar. There are many rationales for why one coefficient is better than another in specific instances [DJ88, KR90], but we will mention only a few.

The comparison of two binary vectors, *p* and *q*, leads to four quantities:
M_{01} = the number of positions where *p* was 0 and *q* was 1
M_{10} = the number of positions where *p* was 1 and *q* was 0
M_{00} = the number of positions where *p* was 0 and *q* was 0
M_{11} = the number of positions where *p* was 1 and *q* was 1

The simplest similarity coefficient is the simple matching coefficient

$$SMC = (M_{11} + M_{00}) / (M_{01} + M_{10} + M_{11} + M_{00})$$

Another commonly used measure is the Jaccard coefficient.

$$J = (M_{11}) / (M_{01} + M_{10} + M_{11})$$

Conceptually, SMC equates similarity with the total number of matches, while J considers only matches on 1's to be important. It is important to realize that there are situations in which both measures are more appropriate.

For example, consider the following pairs of vectors and their simple matching and Jaccard similarity coefficients

a = **1 0 0 0 0 0 0 0 0 0** **SMC = .8**
b = **0 0 0 0 0 0 1 0 0 1** **J = 0**

If the vectors represent student's answers to a True-False test, then both 0-0 and 1-1 matches are important and these two students are very similar, at least in terms of the grades they will get.

Suppose instead that the vectors indicate whether 10 particular items are in the "market baskets" of items purchased by two shoppers. Then the Jaccard measure is a more appropriate measure of similarity since it would be very odd to say that two market baskets that don't contain any similar items are the same. For calculating the similarity of market baskets, and in many other cases, the presence of an item is far more important than absence of an item.

2.2.3.4. Cosine Measure

For purposes of information retrieval, e.g., as in Web search engines, documents are often stored as vectors, where each attribute represents the frequency with which a particular term (word) occurs in the document. (It is much more complicated than this, of course, since certain common words are ignored and normalization is performed to account for different forms of the same word, document length, etc.) Such vectors can have thousands or tens of thousands of attributes.

If we wish to compare the similarity of two documents, then we face a situation much like that in the previous section, except that the vectors are not binary vectors. However, we still need a measure like the Jaccard measure, which ignores 0-0 matches. The motivation is that any two documents are likely to "not contain" many of the same words, and thus, if 0-0 matches are counted most documents will be highly similar to most other documents. The cosine measure, defined below, is the most common measure of document similarity. If d_1 and d_2 are two document vectors, then

$$\cos(d_1, d_2) = (d_1 \bullet d_2) / \|d_1\| \|d_2\| ,$$

where $\bullet$ indicates vector dot product and $\|d\|$ is the length of vector d.

For example,

$d_1 =$ **3 2 0 5 0 0 0 2 0 0**
$d_2 =$ **1 0 0 0 0 0 0 1 0 2**

$d_1 \bullet d_2 = 3*1 + 2*0 + 0*0 + 5*0 + 0*0 + 0*0 + 0*0 + 2*1 + 0*0 + 0*2 = 5$
$\|d_1\| = (3*3 + 2*2 + 0*0 + 5*5 + 0*0 + 0*0 + 0*0 + 2*2 + 0*0 + 0*0)^{0.5} = 6.480$
$\|d_2\| = (1*1 + 0*0 + 0*0 + 0*0 + 0*0 + 0*0 + 0*0 + 1*1 + 0*0 + 2*2)^{0.5} = 2.236$

$\cos(d_1, d_2) = .31$

3. Basic Clustering Techniques

3.1. Types of Clustering

Many different clustering techniques that have been proposed over the years. These techniques can be described using the following criteria [DJ88]:

1) **Hierarchical vs. partitional (nested and unnested)**. Hierarchical techniques produce a nested sequence of partitions, with a single, all inclusive cluster at the top

and singleton clusters of individual points at the bottom. Each intermediate level can be viewed as combining (splitting) two clusters from the next lower (next higher) level. (While most hierarchical algorithms involve joining two clusters or splitting a cluster into two sub-clusters, some hierarchical algorithms join more than two clusters in one step or split a cluster into more than two sub-clusters.)

The following figures indicate different ways of graphically viewing the hierarchical clustering process. Figures 9a and 9b illustrate the more "traditional" view of hierarchical clustering as a process of merging two clusters or splitting one cluster into two. Figure 9a gives a "nested set" representation of the process, while Figure 9b shows a "tree' representation, or dendogram. Figure 9c and 9d show a different, hierarchical clustering, one in which points p1 and p2 are grouped together in the same step that also groups points p3 and p4 together.

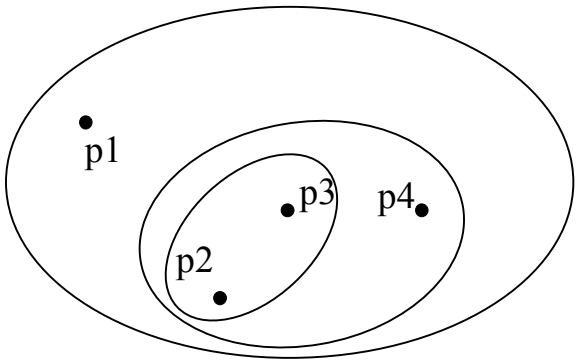


Figure 9a. Traditional nested set.

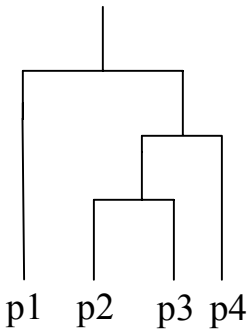


Figure 9b. Traditional dendogram

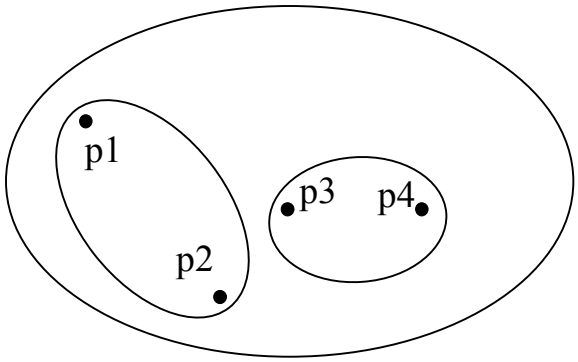


Figure 9c. Non-traditional nested set

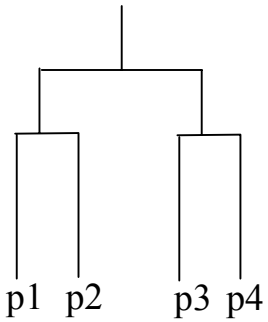


Figure 9d. Non-traditional dendogram.

Partitional techniques create a one-level (unnested) partitioning of the data points. If *K* is the desired number of clusters, then partitional approaches typically find all *K* clusters at once. Contrast this with traditional hierarchical schemes, which bisect a cluster to get two clusters or merge two clusters to get one. Of course, a hierarchical approach can be used to generate a flat partition of *K* clusters, and likewise, the repeated application of a partitional scheme can provide a hierarchical clustering.

2) **Divisive vs. agglomerative**. Hierarchical clustering techniques proceed either from the top to the bottom or from the bottom to the top, i.e., a technique starts with one large cluster and splits it, or starts with clusters each containing a point, and then merges them.

3) **Incremental or non-incremental**. Some clustering techniques work with an item at a time and decide how to cluster it given the current set of points that have already been processed. Other techniques use information about all the points at once. Non-incremental clustering algorithms are far more common.

3.2. Objective Functions: Clustering as an Optimization Problem

Many clustering techniques are based on trying to minimize or maximize a global objective function. The clustering problem then becomes an optimization problem, which, in theory, can be solved by enumerating all possible ways of dividing the points into clusters and evaluating the "goodness" of each potential set of clusters by using the given objective function. Of course, this "exhaustive" approach is computationally infeasible (NP complete) and as a result, a number of more practical techniques for optimizing a global objective function have been developed.

One approach to optimizing a global objective function is to rely on algorithms, which find solutions that are often good, but not optimal. An example of this approach is the K-means clustering algorithm (section 4.1.1) which tries to minimize the sum of the squared distances (error) between objects and their cluster centers.

Another approach is to fit the data to a model. An example of such techniques is mixture models (section 7.1.1), which assume that the data is a "mixture" of a number of underlying statistical distributions. These clustering algorithms seek to find a solution to a clustering problem by finding the maximum likelihood estimate for the statistical parameters that describe the clusters.

Still another approach is to forget about global objective functions. In particular, hierarchical clustering procedures proceed by making local decisions at each step of the clustering process. These 'local' or 'per-step' decisions are also based on an objective function, but the overall or global result is not easily interpreted in terms of a global objective function This will be discussed further in one of the sections on hierarchical clustering techniques (section 4.2.9).

4. Specific Clustering Techniques

4.1. Center-Based Partitional Clustering

As described earlier, partitional clustering techniques create a one-level partitioning of the data points. There are a number of such techniques, but we shall only describe two approaches in this section: K-means (section 4.1.1) and K-medoid (section 4.1.2).

Both these techniques are based on the idea that a center point can represent a cluster. For K-means we use the notion of a centroid, which is the mean or median point of a group of points. Note that a centroid almost never corresponds to an actual data point. For K-medoid we use the notion of a medoid, which is the most representative

(central) point of a group of points. By its definition a medoid is required to be an actual data point.

Section 4.1.3 introduces CLARANS, a more efficient version of the basic K-medoid that was applied to spatial data mining problems. BIRCH and Bubble are clustering methods for data mining that are based on center-based partitional approaches. (BIRCH is more like K-means and Bubble is more like K-medoid.) However, both of BIRCH and Bubble use a wide variety of techniques to help enhance their scalability, i.e., their ability to handle large data sets, and consequently, their discussion is postponed to sections 5.1.1 and section 5.1.2, respectively.

4.1.1. K-means Clustering

4.1.1.1. Basic Algorithm

The K-means clustering technique is very simple and we immediately begin with a description of the basic algorithm. We elaborate in the following sections.

Basic K-means Algorithm for finding *K* clusters.

1. Select *K* points as the initial centroids.
2. Assign all points to the closest centroid.
3. Recompute the centroid of each cluster.
4. Repeat steps 2 and 3 until the centroids don't change.

In the absence of numerical problems, this procedure always converges to a solution, although the solution is typically a local minimum. The following diagram gives an example of this. Figure 10a shows the case when the cluster centers coincide with the circle centers. This is a global minimum. Figure 10b shows a local minima.

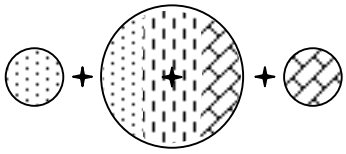


Figure 10a. A globally minimal clustering solution

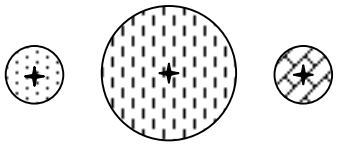


Figure 10b. A locally minimal clustering solution.

4.1.1.2. Time and Space Complexity

Since only the vectors are stored, the space requirements are basically *O(mn)*, where *m* is the number of points and *n* is the number of attributes. The time requirements are *O(I*K*m*n)*, where *I* is the number of iterations required for convergence. *I* is typically small (5-10) and can be easily bounded as most changes occur in the first few

iterations. Thus, K-means is linear in *m*, the number of points, and is efficient, as well as simple, as long as the number of clusters is significantly less than *m*.

4.1.1.3. Choosing initial centroids

Choosing the proper initial centroids is the key step of the basic K-means procedure. It is easy and efficient to choose initial centroids randomly, but the results are often poor. It is possible to perform multiple runs, each with a different set of randomly chosen initial centroids – one study advocates 30 - but this may still not work depending on the data set and the number of clusters sought.

We start with a very simple example of three clusters and 16 points. Figure 11a indicates the "natural" clustering that results when the initial centroids are "well" distributed. Figure 11b indicates a "less natural" clustering that happens when the initial centroids are poorly chosen.

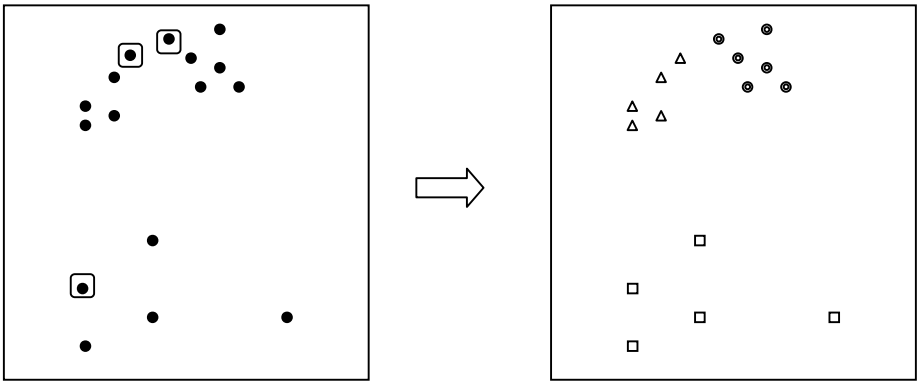


Figure 11a: Good starting centroids and a "natural" clustering.

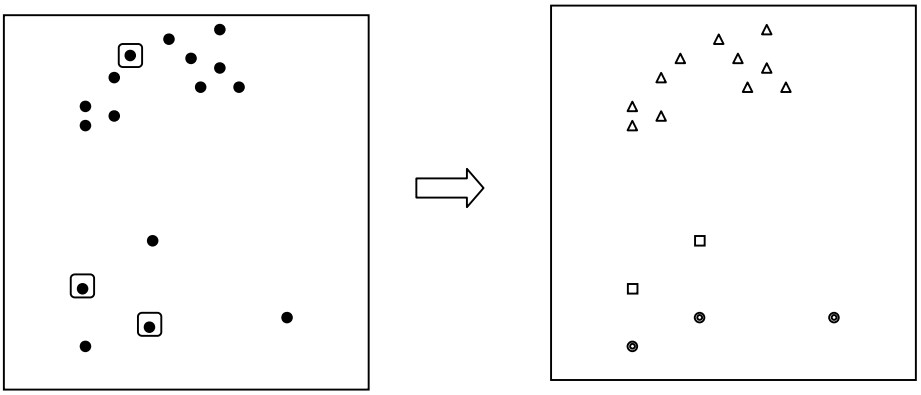


Figure 11b: Bad starting centroids and a "less natural" clustering.

We have also constructed the artificial data set, shown in figure 12a as another illustration of what can go wrong. The figure consists of 10 pairs of circular clusters, where each cluster of a pair of clusters is close to each other, but relatively far from the other clusters. The probability that an initial centroid will come from any given cluster is 0.10, but the probability that each cluster will have exactly one initial centroid is $10!/10^{10}$

= 0.00036. (Technical note: This assumes sampling with replacement, i.e., that two initial centroids could be the same point.)

There isn't any problem as long as two initial centroids fall anywhere in a pair of clusters, since the centroids will redistribute themselves, one to each cluster, and so achieve a globally minimal error,. However, it is very probable that one pair of clusters will have only one initial centroid. In that case, because the pairs of clusters are far apart, the K-means algorithm will not redistribute the centroids between pairs of clusters, and thus, only a local minima will be achieved. When starting with an uneven distribution of initial centroids as shown in figure 12b, we get a non-optimal clustering, as is shown in figure 12c, where different fill patterns indicate different clusters. One of the clusters is split into two clusters, while two clusters are joined in a single cluster.

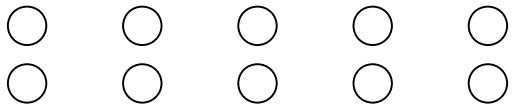


Figure 12a: Data distributed in 10 circular regions

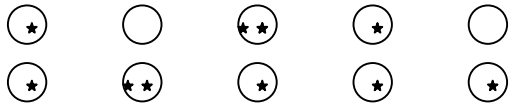


Figure 12b: Initial Centroids

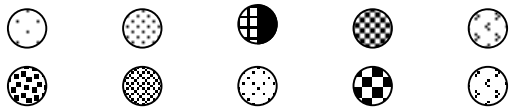


Figure 12c: K-means clustering result

Because random sampling may not cover all clusters, other techniques are often used for finding the initial centroids. For example, initial centroids are often chosen from dense regions, and so that they are well separated, i.e., so that no two centroids are chosen from the same cluster.

4.1.1.4. Updating Centroids Incrementally

Instead of updating the centroid of a cluster after all points have been assigned to clusters, the centroids can be updated as each point is assigned to a cluster. In addition, the relative weight of the point being added may be adjusted. The goal of these modifications is to achieve better accuracy and faster convergence. However, it may be difficult to make a good choice for the relative weight. These update issues are similar to those of updating weights for artificial neural nets.

Incremental update also has another advantage – empty clusters are not produced. (All clusters start with a single point and if a cluster ever gets down to one point, then that point will always be reassigned to that cluster.) Empty clusters are often observed when centroid updates are performed only after all points have been assigned to clusters.

This imposes the need for a technique to choose a new centroid for an empty cluster, for otherwise the squared error will certainly be larger than it would need to be. A common approach is to choose as the new centroid the point that is farthest away from any current center. If nothing else, this eliminates the point that currently contributes the most to the squared error.

Updating points incrementally may introduce an order dependency problem, which can be ameliorated by randomizing the order in which the points are processed. However, this is not really feasible unless the points are in main memory. Updating the centroids after all points are assigned to clusters results in order independence approach.

Finally, note that when centers are updated incrementally, each step of the process may require updating two centroids if a point switches clusters. However, K-means tends to converge rather quickly and so the number of points switching clusters will tend to be small after a few passes over all the points.

4.1.1.5. Different Definitions of Centroid

The K-means algorithm is derived by asking how we can obtain a partition of the data into *K* clusters such that the sum of the squared distance of a point from the "center" of the cluster is minimized. In mathematical terms we seek to minimize

$$Error = \sum_{\mathbf{i=1}}^{\mathbf{K}} \sum_{\vec{\mathbf{x}} \in \mathbf{C_i}} \left\| \vec{\mathbf{x}} - \vec{\mathbf{c}}_{\mathbf{i}} \right\|^{\mathbf{2}} = \sum_{i=1}^{K} \sum_{\vec{x} \in C_i} \sum_{j=1}^{d} (x_j - c_{ij})^2$$

where $\vec{\mathbf{x}}$ is a vector (point) and $\vec{\mathbf{c}}_{\mathbf{i}}$ is the "center" of the cluster, **d** is the dimension of $\vec{\mathbf{x}}$ and $\vec{\mathbf{c}}_{\mathbf{i}}$ and x_j and c_{ij} are the components of $\vec{\mathbf{x}}$ and $\vec{\mathbf{c}}_{\mathbf{i}}$.)

We can solve for the p^{th} cluster, $\vec{\mathbf{c}}_{\mathbf{p}}$, by solving for each component, c_{pk}, $1 \le k \le \mathbf{d}$ by differentiating the *Error*, setting it to 0, and solving as the following equations indicate.

$$\begin{aligned} \frac{\partial}{\partial c_{pk}} Error &= \frac{\partial}{\partial c_{pk}} \sum_{i=1}^{K} \sum_{\vec{x} \in C_i} \sum_{j=1}^{d} (x_j - c_{ij})^2 \\ &= \sum_{i=1}^{K} \sum_{\vec{x} \in C_i} \sum_{j=1}^{d} \frac{\partial}{\partial c_{pk}} (x_j - c_{ij})^2 \\ &= \sum_{\vec{x} \in C_p} 2 * (x_k - c_{pk}) = 0 \end{aligned}$$

$$\sum_{\vec{x} \in C_p} 2 * (x_k - c_{pk}) = 0 \Rightarrow n_p c_{pk} = \sum_{\vec{x} \in C_p} x_k \Rightarrow c_{pk} = \frac{1}{n_p} \sum_{\vec{x} \in C_p} x_k$$

Thus, we find that $\vec{\mathbf{c}}_{\mathbf{p}} = \frac{1}{n_p} \sum_{\vec{\mathbf{x}} \in \mathbf{C_p}} \vec{\mathbf{x}}$, the mean of the points in the cluster. This is pleasing since it agrees with our intuition of what the center of a cluster should be.

However, we can instead ask how we can obtain a partition of the data into K clusters such that the sum of the distance of a point from the "center" of the cluster is minimized. In mathematical terms, we now seek to minimize the equation

$$Error = \sum_{\mathbf{i=1}}^{\mathbf{K}} \sum_{\vec{\mathbf{x}} \in \mathbf{C_i}} \left\| \vec{\mathbf{x}} - \vec{\mathbf{c}}_{\mathbf{i}} \right\|$$

This problem has been extensively studied in operations research under the name of the multi-source Weber problem [Ta98] and has obvious applications to the location of warehouses and other facilities that need to be centrally located. We leave it as an exercise to the reader to verify that the centroid in this case is still the mean.

Finally, we can instead ask how we can obtain a partition of the data into K clusters such that the sum of the L1 distance of a point from the "center" of the cluster is minimized. In mathematical terms, we now seek to minimize the equation

$$Error = \sum_{\mathbf{i=1}}^{\mathbf{K}} \sum_{\vec{\mathbf{x}} \in \mathbf{C_i}} \sum_{\mathbf{j=1}}^{\mathbf{d}} \left| \mathbf{x_j} - \mathbf{c_{ij}} \right|$$

We can solve for the p^{th} cluster, $\vec{\mathbf{c}}_{\mathbf{p}}$, by solving for each component, c_{pk}, $1 \le k \le \mathbf{d}$ by differentiating the *Error*, setting it to 0, and solving as the following equations indicate.

$$\begin{aligned} \frac{\partial}{\partial c_{pk}} Error &= \frac{\partial}{\partial c_{pk}} \sum_{i=1}^{K} \sum_{\vec{x} \in C_i} \sum_{j=1}^{d} \left| x_j - c_{ij} \right| . \\ &= \sum_{i=1}^{K} \sum_{\vec{x} \in C_i} \sum_{j=1}^{d} \frac{\partial}{\partial c_{pk}} \left| x_j - c_{ij} \right| . \\ &= \sum_{\vec{x} \in C_p} \sum_{j=1}^{d} \frac{\partial}{\partial c_{pk}} \left| x_k - c_{pk} \right| = 0 . \end{aligned}$$

$$\sum_{\vec{x} \in C_p} \sum_{j=1}^{d} \frac{\partial}{\partial c_{pk}} \left| x_k - c_{pk} \right| = 0 \Rightarrow \sum_{\vec{x} \in C_p} sign(x_k - c_{pk}) = 0$$

Thus, if we solve for $\vec{\mathbf{c}}_{\mathbf{p}}$, we find that $\vec{\mathbf{c}}_{\mathbf{p}} = median\left(\vec{\mathbf{x}} \in \mathbf{C_p}\right)$, the point whose coordinates are the median values of the corresponding coordinate values of the points in the cluster. While the median of a group of points is straightforward to compute, this computation is not as efficient as the calculation of a mean.

In the first case we say we are attempting to minimize the within cluster squared error, while in the second case and third cases we just say that we are attempting to minimize the absolute within cluster error, where the error may either the L1 or L2 distance.

4.1.1.6. Pre and Post Processing

Sometimes the quality of the clusters that are found can be improved by pre-processing the data. For example, when we use the squared error criteria, outliers can

unduly influence the clusters that are found. Thus, it is common to try to find such values and eliminate them with a preprocessing step.

Another common approach uses post-processing steps to try to fix up the clusters that have been found. For example, small clusters are often eliminated since they frequently represent groups of outliers. Alternatively, two small clusters that are close together can be merged. Finally, large clusters can be split into smaller clusters.

While pre- and post-processing techniques can be effective, they are based on heuristics that may not always work. Furthermore, they often require that the user choose values for a number of parameters.

ISODATA [DJ88] is the most famous of such techniques and is still used today, especially in image processing. After choosing initial centroids in a manner that guarantees that they will be well separated, ISODATA proceeds in the following way:

1) Points are assigned to the closest centroid and cluster centroids are recomputed. This step is repeated until no points change clusters.
2) Clusters with "too few" points are discarded.
3) Clusters are merged (lumped) or split.
   a) If there are "too many" clusters, then clusters are split.
   b) If there are "too few" clusters compared to the number desired, then clusters are merged.
   c) Otherwise the cluster splitting and merging phases alternate.
   Clusters are merged if their centers are close, while clusters are split if the are "too loose."
4) Steps 1,2 and 3 are repeated until the results are "acceptable" or until some pre-determined number of iterations has been exceeded.

The splitting phase of ISODATA is an example of strategies that decrease the total error by increasing the number of clusters. Two general strategies are given below:

a) Split a cluster. Often the cluster with the largest error is chosen, but in the case of ISODATA the cluster with the largest standard deviation in some particular attribute is chosen.
b) Introduce a new cluster center. Often the point that is farthest from any cluster center is chosen.

Likewise, the merging phase of ISODATA is just one example of strategies that decrease the number of clusters. (This will increase the error.) Two general strategies are given below:

a) Disperse a cluster, i.e. remove the center that corresponds to the center and reassign the points to other centers. Ideally the cluster which is dispersed should be the one that increases the error the least. A good heuristic for determining this is based on keeping track of the second closest center of each point.
b) Merge two clusters. In ISODATA, the clusters with the closest centers are chosen, although another, perhaps better, approach is to merge the two clusters that result in the smallest increase in error. These two merging strategies are, respectively, the same strategies that are used in the hierarchical clustering techniques known as the Centroid method and Ward's method. Both methods are discussed in section 4.2.8.

Finally, it is important to realize that there are certain clustering applications, where outliers cannot be eliminated. In data compression every point must be clustered, and in financial analysis, apparent outliers are often the most interesting points, e.g., unusually profitable customers.

4.1.1.7. Limitations and problems

K-means attempts to minimize the squared or absolute error of points with respect to their cluster centroids. While this is sometimes a reasonable criterion and leads to a simple algorithm, K-means has a number of limitations and problems. In particular, Figures 13a and 13b show the problems that result when clusters have widely different sizes or have convex shapes.

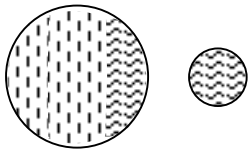


Figure 13a: Different sizes

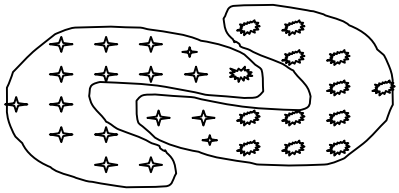


Figure 13b: Convex shapes

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

The K-means algorithm is also normally restricted to data in Euclidean spaces because in many cases the required means and medians do not make sense. (This is not true in all cases, e.g., documents.) A related technique, K-medoid clustering, does not have this restriction and is discussed in the next section.

4.1.2. K-medoid Clustering

The objective of K-medoid clustering [KR90] is to find a non-overlapping set of clusters such that each cluster has a most representative point, i.e., a point that is most centrally located with respect to some measure, e.g., distance. These representative points are called medoids. Once again, the algorithm is conceptually simple.

Basic K-medoid Algorithm for finding K clusters.

1) **Select *K* initial points.** These points are the candidate medoids and are intended to be the most central points of their clusters.
2) **Consider the effect of replacing one of the selected objects (medioids) with one of the non-selected objects.** Conceptually, this is done in the following way. The distance of each non-selected point from the closest candidate medoid is calculated, and this distance is summed over all points. This distance represents the "cost" of the current configuration. All possible swaps of a non-selected point for a selected one are considered, and the cost of each configuration is calculated.
3) **Select the configuration with the lowest cost.** If this is a new configuration, then repeat step 2.

4) **Otherwise, associate each non-selected point with its closest selected point (medoid) and stop.**

Step 3 requires more explanation. The i^{th} medoid is evaluated by using $\sum_{j=1}^{n_i} p_{ij}$, where p_{ij}, is the proximity between the i^{th} medoid and the j^{th} point in the cluster. For similarities (dissimilarities) we want this sum to be as large (small) as possible. It can be seen that such an approach is not restricted to Euclidean spaces and is likely to be more tolerant of outliers. However, finding a better medoid requires trying all points that are currently not medoids and is computationally expensive.

4.1.3. CLARANS

CLARANS [NH94] was one of the first clustering algorithms that was developed specifically for use in data mining – spatial data mining. CLARANS itself grew out of two clustering algorithms, PAM and CLARA [KR90], that were developed in the field of statistics.

PAM (Partitioning Around Medoids) is a "*K*-medoid" based clustering algorithm that attempts to cluster a set of *m* points into *K* clusters by performing the steps described in section 4.1.2.

CLARA (Clustering LARge Applications) is an adaptation of PAM for handling larger data sets. It works by repeatedly sampling a set of data points, calculating the medoids of the sample, and evaluating the cost of the configuration that consists of these "sample-derived" medoids and the entire data set. The set of medoids that minimizes the cost is selected. As an optimization, the set of medoids from one iteration is usually included in the sample for the next iteration. As presented, CLARA normally samples 40 + 5*K* objects and uses 5 iterations.

CLARANS uses a randomized search approach to improve on both CLARA and PAM. Conceptually, CLARANS does the following.

1) Randomly pick *K* candidate medoids.
2) Randomly consider a swap of one of the selected points for a non-selected point.
3) If the new configuration is better, i.e., has lower cost, then repeat step 2 with the new configuration.
4) Otherwise, repeat step 2 with the current configuration unless a parameterized limit has been exceeded. (This limit was set to max (250, *K* *(*m* - *K*)).
5) Compare the current solution with any previous solutions and keep track of the best.
6) Return to step 1 unless a parameterized limit has been exceeded. (This limit was set to 2.)

Thus, CLARANS differs from PAM in that, given the current configuration, it does not consider all possible swaps of medoid and non-medoid points, but rather, only a random selection of them and only until it finds a better configuration. Unlike CLARA, CLARANS works with all data points.

CLARANS was used in two spatial data mining tools, SD(CLARANS) and NSD(CLARANS). These tools added some capabilities related to cluster analysis.

In [EKX95], a number of techniques were proposed for making CLARANS more efficient. The basic idea was to use an R* trees to store the data being considered. R* trees are a type of nearest-neighbor tree (see section 8.1) that try to store data points in

the same disk page if the data points are "close" to one another. [EKX95] proposes to make use of this in three ways to improve CLARANS.

1) **Focus on representative objects**. For all the objects in a single page of the R* tree, find the representative object, i.e., the object which is closest to the calculated center of the objects in the page. Use CLARANS only on these representative objects. This amounts to an intelligent sampling of the database.
2) **Focus on relevant clusters**. When considering a possible swap of a non-medoid object for a medoid object, the change in the cost from the old configuration to the new configuration can be determined by only considering the clusters to which the medoid and non-medoid objects belong.
3) **Focus on the cluster**. It is possible to efficiently construct a bounding Voronoi polyhedron for a cluster by looking only at the medoids. This polyhedron is just the set of all points that are closer to the medoid of the cluster than to any other medoid. This polyhedron can be translated into a range query which can be efficiently implemented by the R* tree.

These improvements improve the speed of CLARANS by roughly two orders of magnitude, but reduce the clustering effectiveness by only a few percent.

4.2. Hierarchical Clustering

In hierarchical clustering the goal is to produce a hierarchical series of nested clusters, ranging from clusters of individual points at the bottom to an all-inclusive cluster at the top. A diagram called a dendogram graphically represents this hierarchy and is an inverted tree that describes the order in which points are merged (bottom-up view) or clusters are split (top-down view).

One of the attractions of hierarchical techniques is that they correspond to taxonomies that are very common in the biological sciences, e.g., kingdom, phylum, genus, species, … . (Some cluster analysis work occurs under the name of "mathematical taxonomy.") Another attractive feature is that hierarchical techniques do not assume any particular number of clusters. Instead any desired number of clusters can be obtained by "cutting" the dendogram at the proper level. Finally, hierarchical techniques are thought to produce better quality clusters [DJ88].

4.2.1. Agglomeration and Division

There are two basic approaches to generating a hierarchical clustering:

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

Divisive techniques are less common, and we will mention only one example before focusing on agglomerative techniques.

4.2.2. Simple Divisive Algorithm (Minimum Spanning Tree (MST))

1) Compute a minimum spanning tree for the proximity graph.

2) Create a new cluster by breaking the link corresponding to the largest distance (smallest similarity).
3) Repeat step 2 until only singleton clusters remain.

This approach is the divisive version of the "single link" agglomerative technique that we will see shortly.

4.2.3. Basic Agglomerative Hierarchical Clustering Algorithm

Many hierarchical agglomerative techniques can be expressed by the following algorithm, which is known as the Lance-Williams algorithm.

Basic Agglomerative Hierarchical Clustering Algorithm

1) Compute the proximity graph, if necessary.
(Sometimes the proximity graph is all that is available.)
2) Merge the closest (most similar) two clusters.
3) Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.
4) Repeat steps 3 and 4 until only a single cluster remains.

The key step of the previous algorithm is the calculation of the proximity between two clusters, and this is where the various agglomerative hierarchical techniques differ. Any of the cluster proximities that we discuss in this section can be viewed as a choice of different parameters (in the Lance-Williams formula) for the proximity between clusters Q and R, where R is formed by merging clusters A and B.

$$p(R, Q) = \alpha_A p(A, Q) + \alpha_B p(B, Q) + \beta p(A, Q) + \gamma | p(A, Q) - p(B, Q) |$$

In words, this formula says that after you merge clusters A and B to form cluster R, then the distance of the new cluster, R, to an existing cluster, Q, is a linear function of the distances of Q from the original clusters A and B.

Any hierarchical technique that can be phrased in this way does not need the original points, only the proximity matrix, which is updated as clustering occurs. However, while a general formula is nice, it is often easier to understand the different hierarchical methods by looking directly at the definition of cluster distance that each method uses, and that is the approach that we shall take here. [DJ88] and [KR90] both give a table that describes each method in terms of the Lance-Williams formula.

4.2.4. Time and Space Complexity

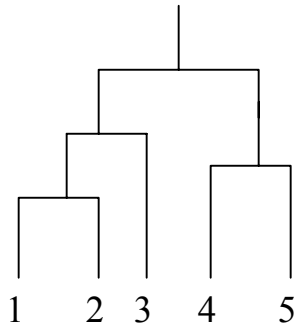
Hierarchical clustering techniques typically use a proximity matrix. This requires the computation and storage of m^2 proximities, a factor that limits the size of data sets that can be processed. (It is possible to compute the proximities on the fly and save space, but this increases computation time.) Once the proximity matrix is available, the time required for hierarchical clustering is $O(m^2)$.

4.2.5. MIN or Single Link

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

The following table gives a sample similarity matrix for five items (I1 – I5) and the dendogram shows the series of merges that result from using the single link technique.

	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

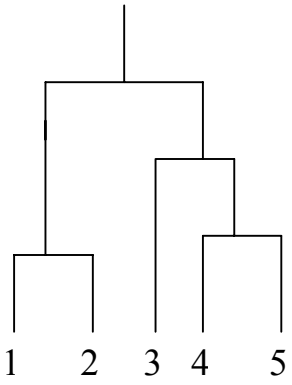


4.2.6. MAX or Complete Link or CLIQUE

For the complete link or MAX version of hierarchical clustering, the proximity of two clusters is defined to be maximum of the distance (minimum of the similarity) between any two points in the different clusters. (The technique is called complete link because, if you start with all points as singleton clusters, and add links between points, strongest links first, then a group of points is not a cluster until all the points in it are completely linked, i.e., form a clique.) Complete link is less susceptible to noise and outliers, but can break large clusters, and has trouble with convex shapes.

The following table gives a sample similarity matrix and the dendogram shows the series of merges that result from using the complete link technique.

	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



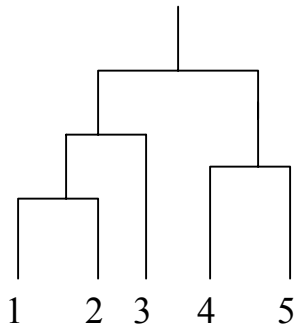
4.2.7. Group Average

For the group average version of hierarchical clustering, the proximity of two clusters is defined to be the average of the pairwise proximities between all pairs of points in the different clusters. Notice that this is an intermediate approach between MIN and MAX. This is expressed by the following equation:

$$proximity\ (cluster1,\ cluster2) = \frac{\sum_{\substack{p_1 \in cluster1 \\ p_2 \in cluster2}} proximity(p_1, p_2)}{size(cluster1) * size(cluster2)}$$

The following table gives a sample similarity matrix and the dendogram shows the series of merges that result from using the group average approach. The hierarchical clustering in this simple case is the same as produced by MIN.

	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



4.2.8. Ward's Method and Centroid methods

For Ward's method the proximity between two clusters is defined as the increase in the squared error that results when two clusters are merged. Thus, this method uses the same objective function as is used by the K-means clustering. While it may seem that this makes this technique somewhat distinct from other hierarchical techniques, some algebra will show that this technique is very similar to the group average method when the proximity between two points is taken to be the square of the distance between them.

Centroid methods calculate the proximity between two clusters by calculating the distance between the centroids of clusters. These techniques may seem similar to K-means, but as we have remarked, Ward's method is the correct hierarchical analogue.

Centroid methods also have a characteristic – often considered bad – that other hierarchical clustering techniques don't posses: the possibility of inversions. In other words, two clusters that are merged may be more similar than the pair of clusters that were merged in a previous step. For other methods the similarity of clusters merged decreases (the distance between merged clusters increases) as we proceed from singleton clusters to one all inclusive clusters.

4.2.9. Key Issues in Hierarchical Clustering

4.2.9.1. Lack of a Global Objective Function

In Section 3.2 we mentioned that hierarchical clustering cannot be viewed as globally optimizing an objective function. Instead, hierarchical clustering techniques use various criteria to decide "locally" at each step which clusters should be joined (or split for divisive approaches). For agglomerative hierarchical techniques, the criteria is typically to merge the "closest" pair of clusters, where "close" is defined by a specified measure of cluster proximity.

This approach yields clustering algorithms that avoid the difficulty of trying to solve a hard combinatorial optimization problem. (As previously, the general clustering problem for objective functions like squared error is NP complete.) Furthermore, such

approaches do not have problems with local minima or difficulties in choosing initial points. Of course, the time and space complexity is $O(m^2)$, which is prohibitive in many cases.

4.2.9.2. Cluster Proximity and Cluster Representation

Cluster proximity is typically defined by a conceptual view of the clusters. For example, if we view a cluster as being represented by all the points, then we can take the proximity between the closest (most similar) two points in different clusters as being the proximity between the two clusters. This defines the MIN technique (section 4.2.5). Likewise, we can also take the proximity between the farthest (least similar) two points in different clusters as our definition of cluster proximity. This defines the MAX technique (section 4.2.6). Finally, we can average the pairwise proximities of all pairs of two points from different clusters. This yields the Group Average technique (section 4.2.7).

If instead, we chose to represent each cluster by a centroid, then we find that different definitions of cluster proximity are more natural. In one case, the cluster distance is defined as the distance between cluster centroids. This defines the Centroid clustering technique (section 4.2.8). Ward's method (section 4.2.8), is conceptually related to Centroid clustering since it merges the two clusters that result in the smallest increase in the total squared error, i.e., it minimizes the sum of the squared distance of points from their centroids.

However, Ward's method can also be viewed as an example of cluster methods that take an "all points" view of cluster representation, but define proximity in terms of preserving desired cluster properties. For Ward's method, the desired cluster property is a low squared error, i.e., minimum squared distance of all points from the cluster center. Chameleon (section 4.2.12) uses the concepts of cluster closeness and connectivity to define cluster properties that must be preserved.

As we will see later, CURE (section 4.2.11) is a hybrid approach that mixes both the centroid and "all points" view. It does this by using a well-chosen subset of the points in a cluster to represent the cluster. Depending on parameter settings, this approach may approximate the MIN technique or the Centroid technique, or something in between.

4.2.9.3. The Impact of Cluster Size

Another aspect of hierarchical agglomerative clustering that should be considered is how to treat the relative sizes of the pairs of clusters that may be merged. There are basically two schemes: weighted and unweighted. Weighted schemes treat each cluster equally and thus, objects in smaller clusters effectively have larger weight. Unweighted schemes treat all objects in the clusters equally. Note that this discussion only applies to cluster proximity schemes that involve sums, i.e., Centroid and Group Average. Unweighted schemes seem to be more popular, and in our further discussions about the Group Average and Centroid schemes, we refer only to the unweighted versions.

4.2.9.4. Merging Decisions are Final

Hierarchical clustering tends to make good local decisions about combining two clusters since it has the entire proximity matrix available. However, once a decision is made to merge two clusters, the hierarchical scheme does not allow for that decision to be

changed. This prevents a local optimization criterion from becoming a global optimization criterion.

For example, in Ward's method, the "minimize squared error" criteria from K-means is used in deciding which clusters to merge. However, this does not result in a clustering that could be used to solve the K-means problem. Even though the local, per-step decisions try to minimize the squared error, the clusters produced on any level, except perhaps the very lowest levels, do not represent an optimal clustering from a "minimize global squared error" point of view. Furthermore, the clusters are not even "stable," in the sense that a point in a cluster may be closer to the centroid of some other cluster than to the centroid of its current cluster.

However, Ward's method can be used as a robust method of initializing a K-means clustering. Thus, a local "minimize squared error" objective function seems to have some connection with a global "minimize squared error" objective function.

Finally it is possible to attempt to fix up the hierarchical clustering produced by hierarchical clustering techniques. One idea is to move branches of the tree around so as to improve some global objective function [Fi96]. Another idea [KHK99b] is to refine the clusters produced by a hierarchical technique by using an approach similar to that used for the multi-level refinement of graph partitions.

4.2.10. Limitations and problems

We summarize the problems with agglomerative hierarchical clustering.

1) No global objective function is being optimized.
2) Merging decisions are final.
3) Good local merging decisions may not result in good global results.
4) Agglomerative hierarchical clustering techniques have trouble with one of more of the following: noise and outliers, non-convex shapes, and a tendency to break large clusters.

4.2.11. CURE

CURE (Clustering Using Representatives) [GRS98] is a clustering algorithm that uses a variety of different techniques to create an approach which can handle large data sets, outliers, and clusters with non-spherical shapes and non-uniform sizes.

CURE represents a cluster by using multiple "representative" points from the cluster. These points will, in theory, capture the geometry and shape of the cluster. The first representative point is chosen to be the point furthest from the center of the cluster, while the remaining points are chosen so that they are farthest from all the previously chosen points. In this way, the representative points are naturally relatively well distributed. The number of points chosen, is a parameter, c, but it was found that a value of 10 or more worked well.

Once the representative points are chosen, they are shrunk toward the center by a factor, α. This helps moderate the effect of outliers, which are usually farther away from the center and thus, are "shrunk" more. For example, a representative point that was a distance of 10 units from the center would move by 3 units (for $\alpha = .7$), while a representative point at a distance of 1 unit would only move .3 units.

CURE uses an agglomerative hierarchical scheme to perform the actual clustering. The distance between two clusters is the minimum distance between any two

representative points (after they are shrunk toward their respective centers). While this scheme is not exactly like any other hierarchical scheme that we have seen, it is equivalent to centroid based hierarchical clustering if $\alpha = 0$, and roughly the same as single link hierarchical clustering if $\alpha = 1$. Notice that while a hierarchical clustering scheme is used, the goal of CURE is to find a given number of clusters as specified by the user.

CURE takes advantage of certain characteristics of the hierarchical clustering process to eliminate outliers at two different points in the clustering process. First, if a cluster is growing slowly, then this may mean that it consists mostly of outliers, since by definition outliers are far from others and will not be merged with other points very often. In CURE this first phase of outlier elimination typically occurs when the number of clusters is 1/3 the original number of points. The second phase of outlier elimination occurs when the number of clusters is on the order of *K*, the number of desired clusters. At this point small clusters are again eliminated.

Since the worst case complexity of CURE is $O(m^2 \log m)$, it cannot be applied directly to large data sets directly. For this reason, CURE uses two techniques for speeding up the clustering process. The first technique takes a random sample and performs hierarchical clustering on the sampled data points. This is followed by a final pass that assigns each remaining point in the data set to one of the clusters by choosing the cluster with closest representative point.

In some cases, the sample required for clustering is still too large and a second additional technique is required. In this situation, CURE partitions the sample data into *p* partitions of size *m/p* and then clusters the points in each partition until there are *m/pq* clusters in each partition. This pre-clustering step is then followed by a final clustering of the *m/q* intermediate clusters. Both clustering passes uses CURE's hierarchical clustering algorithm and are followed by a final pass that assigns each point in the data set to one of the clusters.

We summarize our description of CURE by explicitly listing the different steps:

1) **Draw a random sample from the data set**. The CURE paper is notable for explicitly deriving a formula for what the size of this sample should be in order to guarantee, with high probability, that all clusters are represented by a minimum number of points.
2) **Partition the sample into *p* equal sized partitions.**
3) **Cluster the points in each cluster using the hierarchical clustering algorithm to obtain m/pq clusters in each partition and a total of m/q clusters.** Note that some outlier elimination occurs during this process.
4) **Eliminate outliers.** This is the second phase of outlier elimination.
5) **Assign all data to the nearest cluster to obtain a complete clustering.**

4.2.12. Chameleon

Chameleon [KHK99A] is a clustering algorithm that combines an initial partitioning of the data using an efficient graph partitioning algorithm with a novel hierarchical clustering scheme that dynamically models clusters. The key idea is that two clusters will be merged only if the resulting cluster is similar to the original clusters, i.e., self-similarity is preserved. While applicable to a wide range of data, Chameleon has

specifically been shown to be superior to DBSCAN (section 4.3.1) and CURE (section 4.2.11) for clustering spatial data.

The first step in Chameleon is to generate a k-nearest neighbor graph. Conceptually, such a graph is derived from the proximity graph, and contains links only between a point and its k-nearest neighbors, i.e., the points to which it is closest or most similar. Working with a sparsified proximity graph instead of the full proximity graph can significantly reduce the effects of noise and outliers. See section 4.4.4 for more discussion.

Once a sparsified graph has been obtained, an efficient multi-level graph-partitioning algorithm [KK98a, KK98b] can be used to partition the data set. Chameleon starts with an all inclusive cluster and then splits the largest current cluster until no cluster has more than MIN_SIZE points, where MIN_SIZE is a user specified parameter. This process results in a large number of almost equally sized groups of well-connected vertices (highly similar data points). Graph partitioning takes into account global structure and produces partitions that are sub-clusters, i.e., contain points mostly from one "true" cluster.

To recombine the sub-clusters a novel agglomerative hierarchical algorithm is used. Two clusters are combined if the resulting cluster shares certain properties with the constituent clusters, where two key properties are used to model cluster similarity:

- **Relative Interconnectivity**: The absolute interconnectivity of two clusters normalized by the internal connectivity of the clusters. In words, we will combine two clusters if the points in the resulting cluster are almost as "strongly" connected as points in each of the original clusters. Mathematically, $RI = \frac{EC(C_i, C_j)}{\frac{1}{2}\left(EC(C_i) + EC(C_j)\right)}$, where $EC(C_i, C_j)$is the sum of the edges of k-nearest neighbor graph that connect clusters C_i and C_j, $EC(C_i)$ is the minimum sum of the "cut" edges if we bisect cluster C_i, and $EC(C_j)$ is the minimum sum of the "cut" edges if we bisect cluster C_j. (EC stands for edge cut.)
- **Relative Closeness**: The absolute closeness of two clusters normalized by the internal closeness of the clusters. In words, we will combine two clusters only if the points in the resulting cluster are almost as "close" to each other as in the original cluster. Mathematically,

$RC = \frac{\overline{S}_{EC(C_i, C_j)}}{\frac{|C_i|}{|C_i|+|C_j|}\overline{S}_{EC(C_i)} + \frac{|C_j|}{|C_i|+|C_j|}\overline{S}_{EC(C_j)}}$, where instead of using the edge cut, i.e., the sum of the edges that are cut to bisect a graph, we use the average similarity of the edges involved in the edge cut.

Chameleon has two alternative merging schemes. For the first scheme Chameleon uses two thresholds, T_{RI} and T_{RC} to select candidate pairs of clusters to merge. If more than one pair of clusters has RI and RC values higher than these thresholds, then the pair to merge is chosen by taking the one with the largest value of RI, i.e., by choosing the pair of clusters that are most interconnected. For the second scheme, Chameleon chooses the pair of clusters that maximizes $RI(C_i, C_j) * RC(C_i, C_j)^{\alpha}$, where α is a user specified parameter, which is typically greater than 1.

Unlike traditional hierarchical clustering techniques, Chameleon will merge more than one pair of clusters during a single pass. Another difference (for the first merging scheme) is that Chameleon will reach a point at which it will stop merging clusters. (Recall that traditional agglomerative hierarchical algorithms continue merging until only one all-inclusive cluster remains.) If desired, the thresholds can be relaxed and the clustering resumed. The second merging scheme will automatically merge to a single cluster.

If m is the number of data points and p is the number of partitions the complexity of Chameleon is $O(mp + mlog\ m + p^2 log\ p)$. However, this depends on being able to build the k-nearest neighbor graph in $O(mlog\ m)$ time. This, in turn, requires the use of a k-d tree or a similar type of data structure. Such data structures only work well for low-dimensional data sets, and thus, for higher dimensional data sets the complexity of Chameleon becomes $O(m^2)$.

In summary, the steps of the Chameleon algorithms are:

1) **Build a k-nearest neighbor graph**.
2) **Partition the graph into partitions using a multilevel graph-partitioning algorithm.**
3) **Perform a hierarchical clustering starting with the partitions.** This hierarchical clustering will merge the clusters which best preserve the cluster self-similarity with respect to relative interconnectivity and relative closeness.

4.3. Density Based Clustering

CLIQUE and MAFIA are also density based clustering schemes, but because they are specifically designed for handling clusters in high-dimensional data, we discuss them in sections 5.2.1 and 5.2.2, respectively.

4.3.1. DBSCAN

DBSCAN [EKSX96] is a density based clustering algorithm that works with a number of different distance metrics. When DBSCAN has processed a set of data points, a point will either be in a cluster or will be classified as noise.

DBSCAN is based on the concepts of a point being "density reachable" and "density connected." These concepts are explained in detail in [EKSX96]. What follows is a more basic description.

Conceptually, data points fall into three classes:

a) **Core points**. These are points that are at the interior of a cluster. A point is an interior point if there are enough points in its neighborhood, i.e., if the number of points within a given neighborhood around the point, as determined by the distance function and a supplied distance parameter, exceeds a certain threshold - also a supplied parameter. If two core points belong to each other's neighborhoods, then the core points belong to the same cluster.
b) **Border points**. A border point is a point that is not a core point, i.e., there are not enough points in its neighborhood, but it falls within the neighborhood of a core point. Note that a border point may fall within the neighborhoods of core points from several different clusters. If so, the cluster that the border point is grouped with will depend on the details of the DBSCAN algorithm, and is order dependent.

c) Noise points. A noise point is any point that is not a core point or a border point.

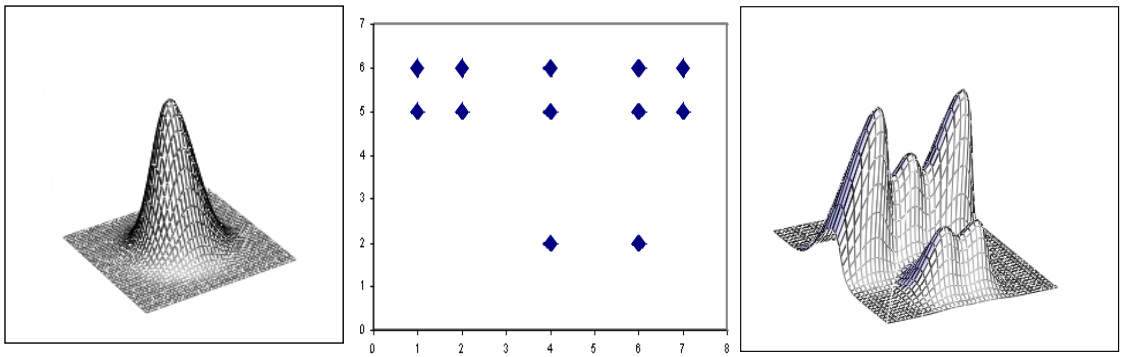
Thus, for DBSCAN, a cluster is the set of all core points whose neighborhoods transitively connect them together, along with some border points.

4.3.2. DENCLUE

DENCLUE (DENsity CLUstEring) [HK98] is a density clustering approach that takes a more formal approach to density based clustering by modeling the overall density of a set of points as the sum of "influence" functions associated with each point. The resulting overall density function will have local peaks, i.e., local density maxima, and these local peaks can be used to define clusters in a straightforward way. Specifically, for each data point, a hill climbing procedure finds the nearest peak associated with that point, and the set of all data points associated with a particular peak (called a local density attractor) becomes a (center-defined) cluster. However, if the density at a local peak is too low, then the points in the associated cluster are classified as noise and discarded. Also, if a local peak can be connected to a second local peak by a path of data points, and the density at each point on the path is above a minimum density threshold, ξ, then the clusters associated with these local peaks are merged. Thus, clusters of any shape can be discovered.

DENCLUE is based on a well-developed area of statistics and pattern recognition which is know as "kernel density estimation" [DH73]. The goal of kernel density estimation (and many other statistical techniques as well) is to describe the distribution of the data by a function. For kernel density estimation, the contribution of each point to the overall density function is expressed by an "influence" (kernel) function. The overall density is then merely the sum of the influence functions associated with each point.

Typically the influence or kernel function is symmetric (the same in all directions) and its value (contribution) decreases as the distance from the point increases. For example, for a particular point, x, the Gaussian function, $K(x) = e^{\frac{-\text{distance}(x,y)^2}{2\sigma^2}}$, is often used as a kernel function. (σ is a parameter which governs how quickly the influence of point drops off.) Figure 14a shows how a Gaussian function would look for a single two dimensional point, while figure 14c shows what the overall density function produced by the Gaussian influence functions of the set of points shown in 14b.



a) Gaussian Kernel b) Set of points c) Overall density function

Figure 14: Example of the Gaussian influence (kernel) function and an overall density function. ($\sigma = 0.75$)

The DENCLUE algorithm has two steps, a preprocessing step and a clustering step. In the pre-clustering step, a grid for the data is created by dividing the minimal bounding hyper-rectangle into *d*-dimensional hyper-rectangles with edge length 2σ. The rectangles that contain points are then determined. (Actually, only the occupied hyper-rectangles are constructed.) The hyper-rectangles are numbered with respect to a particular origin (at one edge of the bounding hyper-rectangle and these keys are stored in a search tree to provide efficient access in later processing. For each stored cell, the number of points, the sum of the points in the cell, and connections to neighboring population cubes are also stored.

For the clustering step DENCLUE, considers only the highly populated cubes and the cubes that are connected to them. For each point, *x*, the local density function is calculated only by considering those points that are from clusters which are a) in clusters that are connected to the one containing the point and b) have cluster centroids within a distance of $k\sigma$ of the point, where k = 4. As an efficiency measure, each point, x', on the path from *x* to its density attractor is assigned to the same cluster as *x* if $dist(x, x') \leq \sigma/2$. As mentioned above, DENCLUDE discards clusters associated with a density attractor whose density is less than ξ. Finally, DENCLUE merges density attractors that can be joined by a path of points, all of which have a density greater than ξ.

DENCLUE can be parameterized so that it behaves much like DBSCAN, but is much more efficient that DBSCAN. DENCLUE can also behave like K-means by choosing σ appropriately and by omitting the step that merges center-defined clusters into arbitrary shaped clusters. Finally, by doing repeated clusterings for different values of σ, a hierarchical clustering can be obtained.

4.3.3. WaveCluster

WaveCluster [SCZ98] is a clustering technique that interprets the original data as a two-dimensional signal and then applies signal processing techniques (the wavelet transform) to map the original data to a new space where cluster identification is more straightforward. More specifically, WaveCluster defines a uniform two-dimensional grid on the data and represents the points in each grid cell by the number of points. Thus, a collection of two-dimensional data points becomes an image, i.e., a set of "gray-scale" pixels, and the problem of finding clusters becomes one of image segmentation.

While there are a number of techniques for image segmentation, wavelets have a couple of features that make them an attractive choice. First, the wavelet approach naturally allows for a multiscale analysis, i.e., the wavelet transform allows features, and hence, clusters, to be detected at different scales, e.g., fine, medium, and coarse. Secondly, the wavelet transform naturally lends itself to noise elimination.

The basic algorithm of WaveCluster is as follows:

1) **Create a grid and assign each data object to a cell in the grid.** The grid is uniform, but the grid size will vary for different scales of analysis. Each grid cell keeps track of the statistical properties of the points in that cell, but for wave clustering only the number of points in the cell is used.
2) **Transform the data to a new space by applying the wavelet transform.** This results in 4 "subimages" at several different levels of resolutionan "average" image, an image that emphasizes the horizontal features, an image that emphasizes vertical features and an image that emphasizes corners.
3) **Find the connected components in the transformed space.** The average subimage is used to find connected clusters, which are just groups of connected "pixels," i.e., pixels which are connected to one another horizontally, vertically, or diagonally.
4) **Map the clusters labels of points in the transformed space back to points in the original space.** WaveCluster creates a lookup table that associates each point in the original with a point in the transformed space. Assignment of cluster labels to the original points is then straightforward.

In summary, the key features of WaveCluster are order independence, no need to specify a number of clusters (although it is helpful to know this in order to figure out the right scale to look at, speed (linear), the elimination of noise and outliers, and the ability to find arbitrarily shaped clusters. While the WaveCluster approach can theoretically be extended to more than two dimensions, it seems unlikely that WaveCluster will work well (efficiently and effectively) for medium or high dimensions.

4.4. Graph-Based Clustering

The hierarchical clustering algorithms that we discussed in section 4.2 can be viewed as operating on a proximity graph. However, they are most commonly viewed in terms of merging or splitting clusters, and often there is no mention of graph related concepts. There are some clustering techniques, however, that are explicitly cast in terms of a graph or a hypergraph. Many of these algorithms are based on the idea of looking at the nearest neighbors of a point. We begin first with a couple of old, but still quite relevant clustering techniques.

4.4.1. Historical techniques

Both approaches we consider use what could be called the nearest neighbor principle, which is also used in classification.

Nearest Neighbor Principle: A point probably belongs to the same class as the points it is closest to, i.e., with high probability a point belongs to the same class as it's nearest neighbors.

4.4.1.1. Shared Nearest Neighbor Clustering

Shared nearest neighbor clustering is described in [JP73].

1) **First the k-nearest neighbors of all points are found.** In graph terms this can be regarded as breaking all but the k strongest links from a point to other points in the proximity graph.
2) **All pairs of points are compared and if**

a) any two points share more than $k_t \leq k$ neighbors, and
b) The two points being compared are among the k-nearest neighbors of each,

Then the two points and any cluster they are part of are merged.

This approach has a number of nice properties. It can handle clusters of different densities since the nearest neighbor approach is self-scaling. This approach is transitive, i.e., if point, *p*, shares lots of near neighbors with point, *q*, which in turn shares lots of near neighbors with point, *r*, then points *p*, *q* and *r* all belong to the same cluster. This allows this technique to handle clusters of different sizes and shapes.

However, transitivity can also join clusters that shouldn't be joined, depending on the k and k_t parameters. Large values for both of these parameters tend to prevent these spurious connections, but also tend to favor the formation of globular clusters.

4.4.1.2. Mutual Nearest Neighbor Clustering

Mutual nearest neighbor clustering is described in [GK77]. It is based on the idea of the "mutual neighborhood value (*mnv*)" of two points, which is the sum of the ranks of the two points in each other's sorted nearest-neighbor lists. Two points are then said to be mutual nearest neighbors if they are the closest pair of points with that *mnv*.

Clusters are built up by starting with points as singleton clusters and then merging the closest pair of clusters, where close is defined in terms of the *mnv*. The *mnv* between two clusters is the maximum *mnv* between any pair of points in the combined cluster. If there are ties in *mnv* between pairs of clusters, they are resolved by looking at the original distances between points. Thus, the algorithm for mutual nearest neighbor clustering works in the following way.

a) First the k-nearest neighbors of all points are found. In graph terms this can be regarded as breaking all but the k strongest links from a point to other points in the proximity graph.
b) For each of the k points in a particular point's k-nearest neighbor list, calculate the *mnv* value for the two points. It can happen that a point is in one point's k-nearest neighbor list, but not vice-versa. In that case, set the *mnv* value to some value larger than 2k.
c) Merge the pair of clusters having the lowest *mnv* (and the lowest distance in case of ties).
d) Repeat step (c) until the desired number of clusters is reached or until the only clusters remaining cannot be merged. The latter case will occur when no points in different clusters are k-nearest neighbors of each other.

The mutual nearest neighbor technique has behavior similar to the shared nearest neighbor technique in that it can handle clusters of varying density, size, and shape. However, it is basically hierarchical in nature while the shared nearest neighbor approach is partitional in nature.

4.4.2. Hypergraph-Based Clustering

Hypergraph-based clustering [HKKM97] is an approach to clustering in high dimensional spaces that uses hypergraphs. Hypergraphs are an extension of regular

graphs, which relax the restriction that an edge can only join two vertices. Instead an edge can join many vertices.

Hypergraph-based clustering consists of the following steps:

1) Define the condition for connecting a number of objects (which will be the vertices of the hypergraph) by a hyperedge.
2) Define a measure of the strength or weight of a hyperedge.
3) Use a graph-partitioning algorithm [KK98b] to partition the hypergraph into two parts in such a way that the weight of the hyperedges cut is minimized.
4) Continue the partitioning until a fixed number of partitions are achieved, or until a "fitness" condition for the goodness of a cluster indicates that a current partition is a good cluster.

In [HKKM97], the data being classified is "market basket" data. With this kind of data there are a number of items and a number of "baskets", or transactions, each of which contains a subset of all possible items. (A prominent example of market basket data is the subset of store items purchased by customers in individual transactions – hence the name market basket data.) This data can be represented by a set of binary vectors – one for each transaction. Each item is associated with a dimension (variable), and a value of 1 indicates that the item was present in the transaction, while a value of 0 indicates that the item was not present.

The individual items are the vertices of the hypergraph. The hyperedges are determined by determining frequent itemsets [AS97]. These are subsets of items that frequently occur together. For example, baby formula and diapers are often purchased together.

The strength of the hyperedges is determined in the following manner. If the frequent itemset being considered is of size n, and the items of the frequent itemset are i_1, $i_2 \ldots i_n$, then the strength of a hyperedge is given by the following formula:

$$\frac{1}{n}\sum_{j=1}^{n} prob(i_j \mid i_1,...,i_{j-1},i_{j+1},i_n)$$

Less formally, we find the strength of each hyperedge by the following steps:

1) Consider each individual item, i_j, in the frequent itemset.
2) Determine what fraction of the market baskets that contain the other n - 1 items also contain i_j. This (estimate of the) conditional probability that i_j occurs when the other items do is a measure of the strength of the association between the items.
3) Average these conditional probabilities together.

In [HKKM97] this is explained in slightly different terms using the concept of association rules [AS97]. An association "rule" is an expression of the form $i_1,...,i_{j-1},i_{j+1},...,i_n \rightarrow i_j$, where the i's are individual items as above. (More generally, the consequent can consist of more than one item.) The "confidence" of a rule is the (estimated) conditional probability as described in point 2 of the preceding paragraph.

The quality of a cluster is measured by a fitness measure. This measure is the ratio of the weight of the edges that connect only vertices in the cluster to the weight of the edges that include at least one vertex in the cluster. Similarly, the degree to which a vertex is associated with a cluster is measured by a connectivity measure. This measure

is the ratio of number of edges that contain the vertex and are in the cluster to the number of edges that contain the vertex.

4.4.3. ROCK

ROCK (RObust Clustering using linKs) [GRS99] is a clustering algorithm for data with categorical and Boolean attributes. It redefines the distances between points to be the number of shared neighbors whose strength is greater than a given threshold and then uses a hierarchical clustering scheme to cluster the data.

As with hypergraph-based clustering discussed in section 4.4.2, ROCK deals primarily with market basket data. Traditional Euclidean distance measures are not appropriate for such data and instead, ROCK uses the Jaccard coefficient (section 2.2.3.3) to measure similarity. This rules out clustering approaches such as K-means or Centroid based hierarchical clustering. (However, K-means can actually be modified to work well for some non-Euclidean data, e.g., documents.)

While the Jaccard coefficient provides a reasonable measure of the distance between points, clusters are sometimes not well separated and so a new measure of similarity between points was introduced that reflects the neighborhood of a point. If $sim(p_i, p_j)$ is the similarity between points, p_i and p_j, and $0 \leq \theta \leq 1$ is a parameter, then

$$link(p_i, p_j) = |\{q : sim(p_i, q) \geq \theta\} \cap \{q : sim(p_j, q) \geq \theta\}|$$

In words, $link(p_i, p_j)$ is the number of shared neighbors of p_i and p_j. The idea is that two points will be "close" only if they share a relatively large number of neighbors. Such a strategy is intended to handle the problem of "border" points, which are close to each other, but belong to different clusters. (This approach is really a form of sparsification as is described in section 4.4.4).

ROCK also introduces a new objective function that is to be maximized:

$$E = \sum_{i=1}^{k} n_i * \sum_{p_q, p_r \in C_i} \frac{link(p_q, p_r)}{n_i^{1+2f(\theta)}}$$

Thus, we try to minimize the sum of the "link" similarity, i.e., the number of shared neighbors, between pairs of points in a cluster, subject to some scaling by the size of the cluster. (This scaling factor is just the expected number of links.) This criterion can be used to derive a criterion for merging clusters via a hierarchical agglomerative scheme by merging the two clusters that lead to the largest increase in E.

ROCK samples the data set in the same manner as CURE in order to avoid using a hierarchical clustering algorithm on a large number of points. This is followed by an assignment step where each remaining points is assigned to a cluster. A fraction of points is selected from each cluster and a calculation is performed to determine the number of those points that are neighbors of the point to be assigned. This quantity is scaled by the expected number of neighbors (based on the size of the cluster) and the point is assigned to cluster with the maximum number of neighbors (after scaling).

The basic steps of ROCK are as follows:

1) Obtain a sample of points from the data set.

2) Compute the link value for each set of points, i.e., transform the original similarities (computed by the Jaccard coefficient) into similarities that reflect the number of shared neighbors between points.
3) Perform an agglomerative hierarchical clustering on the data using the "number of shared neighbors" similarities and the "maximize the shared neighbors" objective function defined above.
4) Assign the remaining points to the clusters that have been found.

4.4.4. Sparsification

The sparsification of the proximity graph as a part of clustering has been discussed briefly before. This section provides a more detailed look at this technique.

4.4.4.1. Why Sparsify?

Many clustering techniques, e.g., hierarchical techniques, use a similarity matrix as their starting point. There are compelling advantages for sparsifying the similarity matrix before beginning the actual clustering process. In particular:

The amount of data that needs to be processed is drastically reduced. Sparsification can eliminate more than 99% of the entries in a similarity matrix. This has two beneficial effects:

- The amount of time required to cluster the data is drastically reduced.
- The size of the problems that can be handled is increased.

Clustering may work better. Sparsification techniques keep the connections to the most similar (nearest) neighbors of a point while breaking the connections to less similar points. (This is in keeping with the nearest neighbor principle that the nearest neighbors of a point tend to belong to the same class as the point itself.) This reduces the impact of noise and outliers and sharpens the distinction between clusters.

Sparsification facilitates the use of graph partitioning algorithms (or algorithms inspired by graph partitioning algorithms). Such algorithms often work best on sparse graphs. The METIS family of graph partitioning algorithms [KK98a, KK98b] are well known and widely used.

There are some techniques that use sparsification with good results. We provide examples of some specific techniques later, but for now we present a simple example. When we applied one form of sparsification to spatial data and then plotted the results, we were able to clearly differentiate core points, boundary points, and noise points. In the following figures (4 and 5), we show, from left to right, a) the initial spatial data (all points), b) points with low connectivity (noise points), c) points with medium connectivity (boundary points), and d) points with high connectivity (core points).

The DBSCAN clustering algorithm described in [EKSX96] can also make a classification of points into core, boundary and noise (outlier) points. In this case, the classification was intentional, and is the basis of the clustering algorithm.

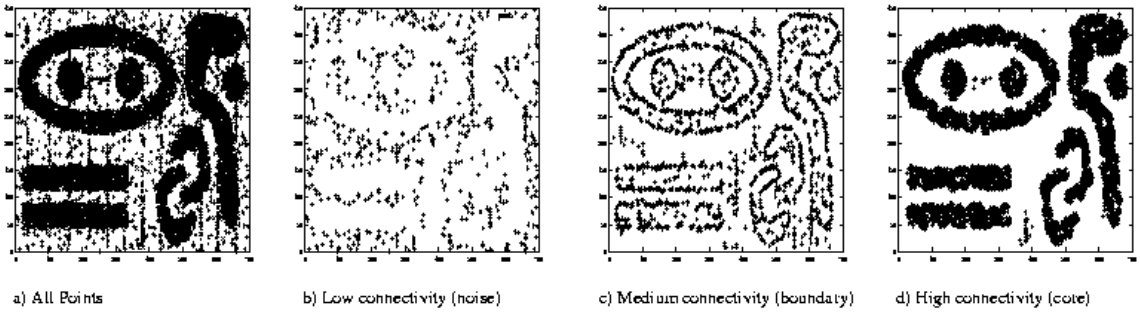


Figure 15: Separation of Spatial Data Points into Noise, Boundary, and Core Points.

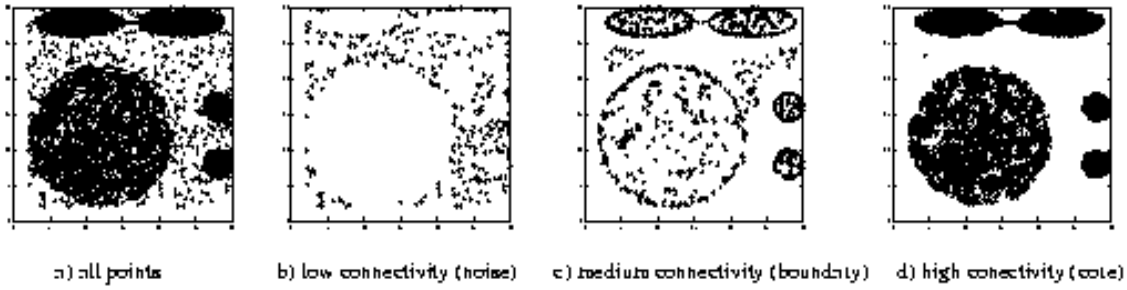


Figure 16: Separation of Spatial Data Points into Noise, Boundary, and Core Points.

4.4.4.2. Sparsification as a Pre-processing Step

Sparsification of the proximity graph must be regarded as an initial step before the use of real clustering algorithms. In theory, a perfect sparsification could leave the proximity matrix split into connected components corresponding to the desired clusters, but, in practice, this rarely happens. It is very easy for a single edge to link two clusters or for a single cluster to be split into several disconnected subclusters. Also, as mentioned, users often desire a hierarchical clustering.

Therefore, the real goal of sparsification should be the efficient and effective identification of the core (strongly connected) points belonging to clusters and subclusters. Sparsification discards outliers and/or boundary points, and may result in a cluster being broken into a number of "good," but separate subclusters. The actual clustering process that follows sparsification should be designed to take the core points and the original points as input and produce clusters or hierarchies of clusters as output. The following diagram gives a view of a complete clustering process that uses sparsification.

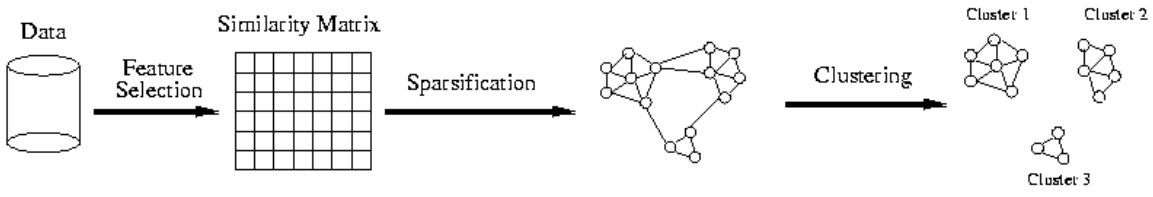


Figure 17. The Complete Clustering Process

4.4.4.3. Sparsification Techniques

In this section we discuss some basic issues of sparsification and review some sparsification techniques.

4.4.4.4. Keeping the Proximity Matrix Symmetric?

As mentioned earlier, most clustering techniques work with symmetric proximity matrices. Conceptually, it is convenient to think of the sparsification process as consisting of two parts: a first step that sparsifies the matrix, and a second step that makes the matrix symmetric again. (In practice, both steps are often carried out simultaneously.)

Two symmetrically corresponding entries, p_{ij} and $p_{ji,}$ may both still be equal after sparsification, i.e., either both values are sparsified or both retain their original and symmetric value. (We are considering only proximity matrices that were originally symmetric.) If not, then there are two options for keeping the proximity matrix symmetric. We can sparsify both entries, or we can set both entries to their original value. The first option is called *sparsification with inclusion*, while the second option is *sparsification with exclusion*. Sparsification with exclusion is sometimes referred to as a "reciprocal" or "mutual" approach. The same sparsification procedure can have quite different properties, depending on whether inclusion or exclusion is used.

4.4.4.5. Description of Sparsification Techniques

a) **Global Thresholds [Kow97].** All entries of the similarity (dissimilarity) matrix that are less than (greater than) the given threshold are set to 0 (∞). While this approach can be effective, members of different classes often have different levels of similarity or dissimilarity. For example, members of class A might have similarities ranging from 0.3 to 0.5, while members of class B may have similarities ranging from 0.6 to 0.8. Thus, a global threshold normally doesn't work well. (However, this procedure is frequently used.)

b) **Nearest neighbor techniques.** These techniques, while not explicitly cast as matrix sparsification techniques [JP73, GK78], use conditions involving the nearest neighbors of a point to sparsify the similarity matrix. These techniques have many variations, but we consider only three of the most straightforward approaches.

   1. **Adaptive Thresholds [KHK99].** The average similarity of a point's two nearest neighbors is calculated and is multiplied by a fixed constant, α, $0 \leq \alpha \leq 1$. This adaptively calculated threshold is used to sparsify the entries in the row of the proximity matrix that corresponds to this point. This technique can be used with inclusion or with exclusion.
   2. **K Nearest Neighbors [KHK99b].** For the row corresponding to a particular point, all entries except those of the K nearest neighbors are sparsified. This technique can be used with inclusion or with exclusion.
   3. **Shared Nearest Neighbors [JP73].** First we apply K nearest neighbors sparsification (with exclusion). The remaining entries are sparsified further by requiring that the entries of a row and the point corresponding to the row must share at least $K_1 \leq K$ neighbors. These two requirements automatically produce a symmetric matrix. In the degenerate case, when $K_1 = 0$, this approach is just K nearest neighbors (with exclusion).

Two additional comments about nearest neighbor approaches:

- Early work that used nearest neighbor techniques to sparsify proximity matrices [JP73, GK78] used the connected components of the sparsified proximity matrix to define the clusters. However, a bad link can ruin the clustering and this is one reason that additional steps are needed following the sparsification process.
- The ROCK algorithm [GRS99] is a new hierarchical clustering algorithm for categorical data, whose initial step redefines the proximity of two points to be the number of nearest neighbors that they share after sparsification with a global threshold. While using a shared nearest neighbors technique for defining proximity is a worthwhile approach - it has also been used in [JP73] and [GK78] - global thresholds do not work as well as some of the other techniques.

4.4.4.6. Evaluating Sparsification Results

If the "true" classes that points belong to are available, then the results of a sparsification can be evaluated by looking at the percentage of points that fall into different five categories.

a) **Percentage of isolated points.** This is the percentage of points that do not have connections to any points after the sparsification. Normally, isolating outliers is good, but isolating other points is bad.
b) **Percentage of really bad points, i.e., the percentage of points isolated from all points of the same class.** This is the percentage of points that are not isolated, but do not have any connections to points within their own classes. Ideally, this percentage should be low.
c) **Percentage of really good points, i.e., the percentage of points isolated from all points of different classes.** This is the percentage of points that are not isolated and do not have any connections to points in other classes. Ideally, this percentage should be high, since points with neighbors of only the same class should, intuitively, be easier to cluster correctly.
d) **Percentage of mostly bad points, i.e., the percentage of points that have a stronger connection to points of other classes.** This is the percentage of points that are not isolated, but whose connections to points in other classes have more weight than do its connections to points of the same class. Ideally, this percentage should be low.
e) **Percentage of mostly good points, i.e., the percentage of points that have a stronger connection to points of the same class.** This is the percentage of points that are not isolated, but whose connections to points of the same class have more weight than do its connections to points in other classes. Ideally, this percentage should be high.

The five percentages just described will sum to 100%, and can be plotted simultaneously to provide an easily understood measure of how specific sparsification techniques distribute points into the different categories described above.

By summing the percentage of mostly good points and the percentage of really good points, we can obtain the **percentage of good points**. This measure provides an overall indication of how well the sparsification is working. If all classes have large percentages of points whose connections are predominantly to the same class, then clustering "should" be easier. Conversely, if some or many classes have large numbers of

points which are isolated or which have neighbors that are predominantly connected to points of other classes, then clustering will be hard.

4.4.4.7. Tentative Conclusions

Sparsification has not been intensively investigated and the results that we present here are based on unpublished research.

1) **Sparsification is best used as a pre-processing step.** Sparsification often breaks "natural" clusters and should be used with an algorithm that can recombine these sub-clusters in an intelligent way. Chameleon is the best example of such an algorithm.
2) **Global thresholds don't work very well.** Not surprisingly, there are two problems:
   a) The effect of the global threshold varies widely for each class, and
   b) Each class has a different threshold that is best.
3) **Adaptive thresholds work better than global thresholds.** Again this is not surprising since such approaches model the data more accurately.
4) **Nearest neighbor techniques are the best of all.** K-nearest neighbor or shared nearest neighbor techniques appear to model the data more accurately than adaptive threshold techniques.
5) **There is a certain amount of flexibility in the choice of the nearest neighbor sparsification scheme and the parameters.** In the research done for Chameleon it was discovered that a number of different nearest neighbor approaches and parameters worked about as well. However, Chameleon a) uses a graph-partitioning algorithms to efficiently divide the points into strongly connected groups, and b) merges sub-clusters back together in a way that undoes the cluster breakage caused by sparsification.

5. Advanced Techniques for Specific Issues

5.1. Scalability

For data mining, the scalability of an algorithm to large data sets is key. The following two related techniques combine a variety of techniques to achieve that goal.

5.1.1. BIRCH

BIRCH (Balanced and Iterative Reducing and Clustering using Hierarchies) [ZRL96] is a highly efficient clustering technique for data in Euclidean vector spaces, i.e., data for which averages make sense. BIRCH can efficiently cluster such data with a single pass and can improve that clustering in additional passes. BIRCH also can also deal effectively with outliers.

BIRCH is based on the notion of a clustering feature (CF) and a CF tree. The idea is that a cluster of data points (vectors) can be represented by a triple of numbers (N, LS, SS), where N is the number of points in the cluster, LS is the linear sum of the points, and SS is the sum of squares of the points. These are common statistical quantities and a number of different inter-cluster distance measures can be derived from them.

A CF tree is a height-balanced tree. Each interior node has entries of the form $[CF_i, child_i]$, where $child_i$ is a pointer to the i^{th} child node. The space that each entry takes and the page size, P, determine the number of entries, B, in an interior node. The

space of each entry is, in turn, determined by the size of the points, i.e., by the dimensionality of the points.

Leaf nodes consist of a sequence of clustering features, CF_i, where each clustering feature represents a number of points that have been previously scanned. Leaf nodes are subject to the restriction that each leaf node must have a diameter that is less than a parameterized threshold, T. The space that each entry takes and the page size, P, determine the number of entries, L, of a leaf.

By adjusting a threshold parameter, T, the height of the tree can be controlled. T controls the fineness of the clustering, i.e., the extent to which the data in the original set of data is reduced. The goal is to keep the CF tree in main memory by adjusting the T parameter as necessary.

A CF tree is built as the data is scanned. As each data point is encountered, the CF tree is traversed, starting from the root and choosing the closest node at each level. When the closest "leaf" cluster for the current data point is finally identified, a test is performed to see if adding the data item to the candidate cluster will result in a new cluster with a diameter greater than the given threshold, T. If not, then the data point is "added" to the candidate cluster by updating the CF information. The cluster information for all nodes from the leaf to the root is also updated.

If the new cluster would have a diameter greater than T, then a new entry is created if the leaf node is not full. Otherwise the leaf node must be split. The two entries (clusters) that are farthest apart are selected as "seeds" and the remaining entries are distributed to one of the two new leaf nodes, based on which leaf node contains the closest "seed" cluster. Once the leaf node has been split, the parent node is updated, and split if necessary, i.e., if the parent node is full. This process may continue all the way to the root node.

BIRCH follows each split with a merge step. At the interior node where the split stops, the two closest entries are found. If these entries do not correspond to the two entries that just resulted from the split, then an attempt is made to merge these entries and their corresponding child nodes. This step is intended to increase space utilization and avoid problems with skewed data input order.

BIRCH also has a procedure for removing outliers. When the tree needs to be rebuilt, because it has run out of memory, then outliers can optionally be written to disk. (An outlier is defined to be node that has "far fewer" data points than average.) At certain points in the process, outliers are scanned to see if they can be absorbed back into the tree without causing the tree to grow in size. If so, they are re-absorbed. If not, they are deleted.

BIRCH consists of a number of phases beyond the initial creation of the CF tree. The phases of BIRCH are as follows:

1) **Load the data into memory by creating a CF tree that "summarizes" the data**.
2) **Build a smaller CF tree if it is necessary for phase 3**. T is increased, and then the leaf node entries (clusters) are reinserted. Since T has increased, some clusters will be merged.
3) **Perform global clustering**. Different forms of global clustering (clustering which uses the pairwise distances between all the clusters) can be used. However, an agglomerative, hierarchical technique was selected. Because the clustering features store summary information that is important to certain kinds of clustering, the global

clustering algorithm can be applied as if it were being applied to all the points in cluster represented by the CF.

4) **Redistribute the data points using the centroids of clusters discovered in step 3 and thus, discover a new set of clusters**. This overcomes certain problems that can occur in the first phase of BIRCH. Because of page size constraints and the T parameter, points that should be in one cluster are sometimes split, and points that should be in different clusters are sometimes combined. Also, if the data set contains duplicate points, these points can sometimes be clustered differently, depending on the order in which they are encountered. By repeating this phase, multiple times, the process converges to a (possibly local) minimum.

5.1.2. Bubble and Bubble-FM

Bubble and Bubble-FM [GRGPF99] are scalable clustering algorithms that are extensions of BIRCH to general metric spaces, i.e., collections of data points where only distances are meaningful. (In Euclidean space, points (vectors) can be added and concepts such as a mean or median of a group of points make sense. This is not the case for general metric spaces, although there are exceptions, e.g., documents.) Bubble and Bubble-FM can efficiently cluster general metric data with a single pass.

(A good example of non-Euclidean data is strings. An "edit" distance can be defined between two strings which is the minimum number of character insertions, deletions, and transpositions that would be needed to transform one string into another. While these distances form a metric space, they are not an Euclidean vector space.)

Bubble and Bubble-FM are based on BIRCH*, which generalizes the notion of a cluster feature that was introduced in BIRCH.

Conceptually, this new CF is a quintuple:

a) The number of objects in the cluster.

b) The objects the cluster.

c) The row sum of each object in the clusters. The row sum is the sum of the squared distance of an object to each other object in the cluster.

d) The clustroid of the cluster. The clustroid is the object in the cluster that has the smallest row sum. The clustroid represents a generalization of the idea of a centroid to general metric spaces. Note that the clustroid is always an actual object, and thus, is like a medoid.

e) The radius of the cluster, which is defined as $\sqrt{\frac{\sum_{i=1}^{n} d^2(object_i, clustroid)}{n}}$, where d is the distance function and n is the number of objects in the cluster.

However, from an efficiency point of view not all objects and row sums can be maintained in memory. Once a cluster becomes large enough, only a representative set of objects (and their corresponding row sums) is kept and updates to the CF are calculated approximately

As with BIRCH, these cluster features are organized into a CF tree, which is a height-balanced tree similar to an R* tree. Each interior node has entries of the form $[CF_i, child_i]$, where $child_i$ is a pointer to the i^{th} child node. Cluster features for non-leaf nodes are similar to those of leaf nodes, where the representative objects are chosen from

the representative objects of the node's children. However, a condition is imposed that at least one representative object be chosen from each child.

The basic steps of BIRCH* are the same as BIRCH, although the details are somewhat different. One important parameter governs the way in which the distance of point being added to a CF node is calculated. For Bubble, the point is considered to be a cluster and the distance to non-leaf nodes is calculated as the average distance of the point to each representative point in the cluster. For leaf nodes, the distance is calculated as the distance of the point to the clustroid.

When a new data point is processed, the data point is treated like a singleton cluster. The CF tree is traversed, starting from the root and choosing the closest node at each level. When the closest "leaf" cluster for the current data point is finally identified, a test is performed to see if adding the data item to the candidate cluster will result in a new cluster with a radius greater than a given threshold, T. If not, then the data point is "added" to the candidate cluster by updating the CF information. The cluster information for all nodes from the leaf to the root is also updated. If the new cluster would have a radius greater than T, then a new entry is created.

Bubble-FM uses an algorithm called Fast Map [FL95] to map the representative objects of a non-leaf node into Euclidean space, called image space, in a way that preserves the distances between the points. The clustroid is then replaced by the centroid of the mapped representative points. When a point is added to the CF tree and needs to be compared to a particular non-leaf node, it is mapped into the image space of that cluster and its distance to the centroid is calculated and compared to the threshold. Distance computations to the leaf nodes are still performed as with Bubble.

The mapping adds a lot of complexity and does not give as accurate an estimate of the distance of a point from a cluster, but it requires fewer distance computations. This can be important in many cases, e.g., string comparisons, where the distance calculations are time consuming. Distances to leaf nodes are not calculated using the mapping, because it is important not to make a mistake in adding a point to the wrong cluster. It is better to start a new cluster, and the frequent updates at leaf nodes would require too many re-mappings.

As with BIRCH, Bubble and Bubble-FM both follow the initial clustering by a hierarchical clustering to refine the clusters and another pass through the data that redistributes the points to the clusters.

5.2. Clusters in Subspaces

In high dimensional spaces, clusters often lie in a subspace. The following two related techniques have been devised to handle that situation.

5.2.1. CLIQUE

CLIQUE [AGGR98] is a clustering algorithm that finds high-density regions by partitioning the data space into cells (hyper-rectangles) and finding the dense cells. Clusters are found by taking the union of all adjacent, high-density cells. For simplicity and ease of use, clusters are described by expressing the cluster as a DNF (disjunctive normal form) expression and then simplifying the expression.

CLIQUE is based on the following simple property of clusters: Since a cluster represents a dense region in some subspace of the feature space, there will be dense areas

corresponding to the cluster in all lower dimensional subspaces. Thus, CLIQUE starts by finding all the dense areas in the one-dimensional spaces corresponding to each attribute. CLIQUE then generates the set of two-dimensional cells that might possibly be dense, by looking at dense one-dimensional cells, as each two-dimensional cell must be associated with a pair of dense one-dimensional cells. More generally, CLIQUE generates the possible set of k-dimensional cells that might possibly be dense by looking at dense (k - 1) dimensional cells, since each k-dimensional cell must be associated with a set of k dense (k - 1)-dimensional cells.

The CLIQUE algorithm is similar to the APRIORI algorithm [AS97] for finding frequent itemsets.

5.2.2. MAFIA (pMAFIA)

MAFIA (Merging of Adaptive Finite Intervals (And more than a CLIQUE)) [GHC99] is a modification of CLIQUE that runs faster (40+) and finds better quality clusters. pMAFIA is the parallel version.

The main modification is to use an adaptive grid. Initially, each dimension (attribute) is partitioned into a fixed number of cells. A histogram is generated that shows the number of data points in each cell. Groups of five adjacent cells are grouped into windows, and each window is assigned the maximum value of the number of points in its five cells. Adjacent windows are grouped together if the values of the two windows are close – within 20%. As a special case, if all windows are combined into one window, the dimension is partitioned into a fixed number of cells and the threshold for being considered a dense unit is increased for this dimension. A relatively uniform distribution of data in a particular dimension normally indicates that no clustering exists in this dimension.

6. Evaluations

In this section we evaluate some of the clustering techniques for data mining that have been discussed earlier. *(Actual reviews of the specific techniques are omitted in this preliminary version of the paper.)*

6.1. Requirements for Clustering Analysis for Data Mining

6.2. Typical Problems and Desired Characteristics

The desired characteristics of a clustering algorithm depend on the particular problem under consideration. The following is a list. (Note however, that there are other general requirements for data mining that have also been described [CHY96].)

6.2.1. Scalability

Clustering techniques for large sets of data must be scalable, both in terms of speed and space. It is not unusual for a database to contain millions of records, and thus, any clustering algorithm used should have linear or near linear time complexity to handle such large data sets. (Even algorithms that have complexity of $O(m^2)$ are not practical for large data sets.) Some clustering techniques use statistical sampling. Nonetheless, there are cases, e.g., situations where relatively rare points have a dramatic effect on the final clustering, where a sampling is insufficient.

Furthermore, clustering techniques for databases cannot assume that all the data will fit in main memory or that data elements can be randomly accessed. These algorithms are, likewise, infeasible for large data sets. Accessing data points sequentially and not being dependent on having all the data in main memory at once are important characteristics for scalability.

6.2.2. Independence of the order of input

Some clustering algorithms are dependent on the order of the input, i.e., if the order in which the data points are processed changes, then the resulting clusters may change. This is unappealing since it calls into question the validity of the clusters that have been discovered. They may just represent local minimums or artifacts of the algorithm.

6.2.3. Effective means of detecting and dealing with noise or outlying points

A point which is noise or is simply an atypical point (outlier) can often distort a clustering algorithm. By applying tests that determine if a particular point really belongs to a given cluster, some algorithms can detect noise and outliers and delete them or otherwise eliminate their negative effects. This processing can occur either while the clustering process is taking place or as a post-processing step.

However, in some instances, points cannot be discarded and must be clustered as well as possible. In such cases, it is important to make sure that these points do not distort the clustering process for the majority of the points.

6.2.4. Effective means of evaluating the validity of clusters that are produced.

It is common for clustering algorithms to produce clusters that are not "good" clusters when evaluated later.

6.2.5. Easy interpretability of results

Many clustering methods produce cluster descriptions that are just lists of the points belonging to each cluster. Such results are often hard to interpret. A description of a cluster as a region may be much more understandable than a list of points. This may take the form of a hyper-rectangle or a center point with a radius.

Also, data clustering is sometimes preceded by a transformation of the original data space – often into a space with a reduced number of dimensions. While this can be helpful for finding clusters, it can make the results very hard to interpret.

6.2.6. The ability to find clusters in subspaces of the original space.

Clusters often occupy a subspace of the full data space. (Hence, the popularity of dimensionality reduction techniques.) Many algorithms have difficulty finding, for example, a 5 dimensional cluster in a 10 dimensional space.

6.2.7. The ability to handle distances in high dimensional spaces properly.

High-dimensional spaces are quite different from low dimensional spaces. In [BGRS99], it is shown that the distances between the closest and farthest neighbors of a

point may be very similar in high dimensional spaces. Perhaps an intuitive way to see this is to realize that the volume of a hyper-sphere with radius, r, and dimension, d, is proportional to r^d, and thus, for high dimensions a small change in radius, means a large change in volume. Distance based clustering approaches may not work well in such cases.

If the distance between points in a high dimensional space are plotted, then the graph will often show two peaks: a "small" distance representing the distance between points in clusters, and a "larger" distance representing the average distance between points. (See [Bri95].) If only one peak is present or if the two peaks are close, then clustering via distance based approaches will likely be difficult.

Yet another set of problems has to do with how to weight the different dimensions. If different aspects of the data are being measured in different scales, then a number of difficult issues arise. Most distance functions will weight dimensions with greater ranges of data more highly. Also, clusters that are determined by using only certain dimensions may be quite different from the clusters determined by using different dimensions. Some techniques are based on using the dimensions that result in the greatest differentiation between data points. Many of these issues are related to the topic of feature selection, which is an important part of pattern recognition.

6.2.8. Robustness in the presence of different underlying data and cluster characteristics

A robust clustering technique must be take into account the following characteristics of the data and the clusters:

1) **Dimensionality** (Discussed separately in sections 6.2.6 and 6.2.7)
2) **Noise and Outliers** (Discussed separately in section 6.2.3)
3) **Statistical Distribution.** Some data is has a Gaussian (normal) or uniform distribution, but this is frequently not the case.
4) **Cluster Shape.** In some cases clusters are regularly shaped regions, e.g., rectangular or globular, but irregularly shaped, non-convex clusters are common.
5) **Cluster Size.** Some clustering methods, e.g., K-means don't work well in the presence of different size clusters.
6) **Cluster Density.** Some clustering methods, e.g., K-means don't work well in the presence of clusters that have different density.
7) **Cluster Separation.** In some cases the clusters are well-separated, but in many other cases the clusters may touch or overlap.
8) **Type of data space, e.g., Euclidean or non-Euclidean.** Some clustering techniques calculate means or use other vector operations that often only make sense in Euclidean space.
9) **Many and Mixed Attribute Types**, e.g., (temporal, continuous, categorical). A mix of attribute types is usually handled by having a proximity function that can combine all the different attributes in an "intelligent" way.

Many clustering techniques have assumption about the data and cluster characteristics and do not work well if those assumptions are violated. In such cases the clustering technique may miss clusters, split clusters, merge clusters, or just produce poor clusters.

6.2.9. An ability to estimate any parameters

(For example, the number of clusters, the size of clusters, or the density of clusters.) Many clustering algorithms take the number of clusters as a parameter. This can be a useful feature in certain instances, e.g., when using a clustering algorithm to create a balanced tree for nearest neighbor lookup, or when using clustering for compression. However, this is not generally good, since the number of clusters parameterized may not match the "real" number of clusters.

An algorithm that requires the number of clusters up front can always be run multiple times. Assuming that there is some way to compare the quality of the clusters produced, it is then possible to empirically determine the best number of clusters. Of course, this increases the amount of computation required.

Likewise, it is often difficult to estimate the proper values for other parameters of clustering algorithms. In general, the parameters of a clustering algorithm may identify areas of weakness. In the best case, the results produced by a clustering algorithm will be relatively insensitive to modest changes in the parameter values.

6.2.10. Ability to function in an incremental manner

In certain cases, e.g., data warehouses, the underlying data used for the original clustering can change over time. If the clustering algorithm can incrementally handle the addition of new data or the deletion of old data, then this is usually much more efficient than re-running the algorithm on the new data set.

7. Data Mining From Other Perspectives

7.1. Statistics-Based Data Mining

Cluster Analysis has a long history in statistics and statisticians developed some of the techniques that have been previously discussed. We will not discuss most of these techniques any further here. However, there is a class of techniques, mixture models, which deserve additional attention.

7.1.1. Mixture Models

The basic idea of mixture models is to view the data as a mixture of observations from a number of different probability distributions. The probability distributions are usually taken to be multivariate normal, since this type of distribution is well-understood, relatively easy to work with, and has been shown to work well in many instances. These types of distributions yield hyper-ellipsoidal clusters.

There are a number of different assumptions that can be made when taking a mixture model approach, i.e., whether all clusters have the same shape, volume, and orientation. These assumptions are reflected in the number of statistical parameters (mean vectors and covariance matrices), that need to be estimated. Of course, the more flexible the model, the more difficult and time consuming it is to find a solution.

Some form of the EM (expectation maximization) algorithm is typically used to perform the parameter estimation. This algorithm can be slow, is not practical for models with large numbers of components, and does not work well when clusters contain only a few data points or if the data points are nearly co-linear. There is also a problem in

estimating the number of clusters or, more generally, in choosing the exact form of the model to use. This problem has typically been dealt with by applying a Bayesian approach, which roughly speaking, gives the odds of one model versus another, based on an estimate derived from the data. Mixture models may also have difficulty with noise and outliers, although work has been done to deal with this problem.

In spite of all the apparent difficulties with mixture models, they do have the advantage of having a sound mathematical foundation. Also, a model-based approach provides a disciplined way of eliminating some of the complexity associated with data. (To see the patterns in data it often necessary to simplify the data, and fitting the data to a model is a good way to do that, at least if the model is a good match for the data.) Furthermore, it is easy to characterize the clusters produced, since they can be described by a small number of parameters. Finally, many sets of data are indeed the result of random processes and thus, should satisfy the statistical assumptions of these models.

AutoClass [CS96] is one of the most prominent of the mixture-model based programs and has been used for data mining to some extent. A readable and current technical report on clustering using mixture models is given by [FR98]. This source also provides a pointer to state-of-the-art mixture modeling software. An approach to using mixture models for co-occurrence data (frequent itemsets of size 2) is given in [HP98].

7.2. Text Mining and Information Retrieval

Information retrieval is the study the problems and techniques involved in automatically indexing documents and retrieve collections of documents. A common task in information retrieval is retrieving relevant documents based on keywords specified by a user, e.g., web search engines.

Text mining is the discovery of useful and non-obvious information in collections of text documents. Some people regard text mining as separate from data mining, since they view data mining as restricted to databases, but many of the concepts and techniques used for data mining and text mining are quite similar. It is true that some text mining techniques make use of natural language processing or the structure of documents, but many techniques do not.

For both text mining and information retrieval, documents are typically represented by vectors that indicate the frequency of keywords in the documents. In such cases, many of the clustering techniques that we have previously described, i.e., K-means and agglomerative hierarchical clustering, can be directly applied. (This is so even though document vectors are a very high dimensional space - thousands or tens of thousands of dimensions.) In what follows, we will restrict ourselves to the assumption that we are dealing with document vectors.

7.2.1. Uses of Clustering for Documents

Clustering has been used in three principal ways with documents:

1) **Document Retrieval**. Early researchers hoped that clustering documents would lead to quicker and better document retrieval. These hopes have not been realized as other (non-cluster based) techniques for document retrieval continue to be superior.
2) **Document browsing**. Document clustering can provide a way of summarizing a large number of documents into a small number of groups that are suitable for browsing. The collection of documents being browsed might be the entire collection

of documents or it might be just the documents returned by a user query. The documents can be reclustered to match a user's interest.

3) **Automatic Creation of Document Hierarchies**. Current hierarchical classifications of documents are created mostly by hand. There has been some work on trying to automate or partially automate this process.

Many of the techniques used for clustering documents are ones that we have seen in previous sections. Hierarchical techniques were often used in the past, but they do not scale well to the large number of documents often found in collections of documents today, e.g., consider the number of Web pages. Consequently more recent studies have often focused on using variations of K-means. [SKK00]

7.2.2. Latent Semantic Indexing

LSI is a dimension reduction technique that can be used as a preliminary to a clustering algorithm in the same way that principal component analysis is sometimes used. (Recall that principal component analysis (PCA) is a technique for dimensionality reduction commonly used in statistics.)

However, Latent Semantic Indexing is a technique that is mostly used for information retrieval. Given a number of keywords, n, and a number of documents, m, an m by n matrix is constructed, such that the ij^{th} entry of the matrix contains the number of times that the j^{th} keyword occurs in the i^{th} document. Using standard linear algebra techniques, an alternative form of this matrix, the singular value decomposition (SVD), is found. This form can be simplified (see [BD95] and [FO95]). Conceptually, the only information that is kept is the information that reflects the most significant connections between keywords and documents. The results of a query consisting of a keyword vector can be computed using linear algebra operations.

7.3. Vector Quantization

There has been a considerable amount of cluster analysis work done in the area of vector quantization. The idea is the following: given a set of vectors representing, for example an image, find a set of reference vectors that will be used instead of the full set of vectors. The idea is that in video and voice, many of the data points that describe a particular image or sound are similar, and, hence, not much information will be lost if each vector is replaced by its closest reference vector. In addition, the information can be represented in a much smaller space, i.e. a high degree of compression is possible. Vector quantization may also aid pattern recognition by significantly reducing the number of patterns that need to be considered.

The challenge is to minimize the error that results from approximating each vector by a reference vector and to maximize the possible compression by minimizing the number of reference vectors needed.

Various clustering techniques have been used to perform this function. In particular, the common K-means algorithm [DJ88] has been used. However, this algorithm requires specifying the number of clusters, and more recently, other information (entropy) based techniques [BK93] have been used that can determine both the reference vectors (the clusters) and the best value for K, the number of clusters.

This type of approach has been extended to more general clustering problems [BH97]. However, these approaches are very mathematically and computationally intense, using concepts similar to those used in statistical mechanics and information theory, and relying on simulated and deterministic annealing algorithms.

An interesting thing to note about vector quantization is that there is not any notion of an outlier or noise. All points must be mapped to some reference vector.

7.4. Self-Organizing Maps

7.4.1. SOM Overview

The Kohonen Self-Organizing Feature Map (SOFM or SOM) [Koh97] is a clustering and data visualization technique based on neural networks. The "self-organizing" part of the name derives from the ability of neural nets to learn their configuration from training data. The "map" part of the name comes from the way that SOM neural nets work, i.e., when an input (a point) is fed into a SOM neural network, the output is the label of a particular neuron in the network, and thus, a SOM neural network can be viewed as mapping a point to neuron. Finally, the "feature" portion of the name comes from observation that, in a trained SOM network, a small set of neurons are often associated with a particular data feature.

A detailed explanation of how SOM works is provided below, but conceptually, SOM can be viewed as a vector quantization technique where the reference vectors are learned by training a neural network. Thus, like vector quantization, the goal of SOM is to find a set of reference vectors and to assign each point in the data set to the "closest" reference vector. With the SOM approach, each reference vector is associated with a particular neuron, and the components of that reference vector become the "weights" of that neuron. As with other neural network systems, a SOM neural net is trained by considering the data points one at a time and adjusting the weights (reference vectors) so as to best fit the data.

The final output of the SOM technique is a set of reference vectors which implicitly defines clusters. (Each cluster consists of the points closest to a particular reference vector.) However, while SOM might seem very similar to K-means or other vector quantization approaches, there is a fundamental conceptual difference. During the training process, SOM uses each data point to update the closest reference vector and *the reference vectors of nearby neurons*. In this way, SOM produces an "ordered" set of reference vectors. In other words, the reference vectors of neurons which are close to each other in the SOM neural net will be more closely related to each other than to the reference vectors of neurons that are farther away in the neural net. Because of this constraint, the reference vectors in SOM can be viewed as lying on a smooth, elastic two-dimensional surface in m dimensional space, a surface which tries to fit the m dimensional data as well as possible. (Regression is a statistical techniques that tries to find the hyper-plane that best fits a set of data, and thus, you could think of the SOM reference vectors as the result of a nonlinear regression with respect to the data points.)

7.4.2. Details of a Two-dimensional SOM

There are many types of SOM neural networks, but we restrict our discussion to a simple variant where the neurons are organized as a two-dimensional lattice as shown in Figure 17.

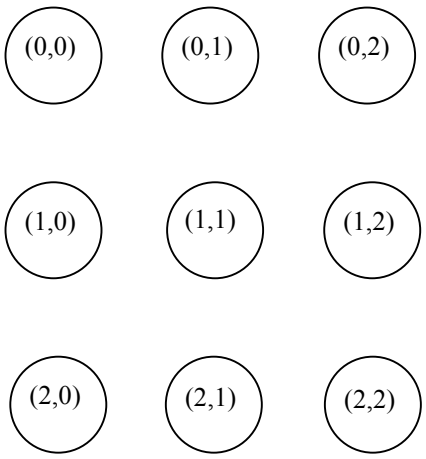


Figure 17. Two-dimensional 3 by 3 rectangular SOM neural network.

Note that each neuron (node) is assigned a pair of coordinates (i, j). Often such a network is drawn with links between adjacent neurons, but that is misleading since the influence of one neuron on another is via a neighborhood which is defined in terms of coordinates, not links. Each neuron is associated with an *m* dimensional reference vector, where *m* is the dimensionality of the data points.

At a high level, clustering using the SOM techniques consists of the following steps:

1) Initialize the reference vectors.
2) While the training is not complete do the following:
a. Select the next training sample (point).
b. Determine the neuron whose reference vector is closest to the current point.
c. Update this reference vector and the reference vectors of all neurons which are close, i.e., in a specified neighborhood.
3) When the training is complete, assign all points to the closest reference vectors and return the reference vectors and clusters.

Initialization can be performed in a number of ways. One approach is to choose each component of a reference vector randomly from the range of values observed in the data for that component. While this approach works, it is not necessarily the best approach, especially for producing a rapid convergence to an equilibrium state. Another approach is to randomly choose the initial reference vectors form the available data points. (This is very much like one of the techniques for initializing K-means centroids.) There are is also a more sophisticated technique which uses the autocorrelation matrix of the data and which is described in [Koh97].

The second step is the training or the convergence step. Selection of the next training sample is fairly straightforward, although there are a few issues. Since training may require a 100,000 steps, each data point may be used multiple times, especially if the number of points is small. However, if the number of points is large, then not every point must be used. Also, it is possible to enhance the influence of certain groups of points, e.g., a less frequent class of points, by increasing their frequency in the training set. (However, this becomes more like supervised classification.)

The determination of closest reference vector is also straightforward, although it does require the specification of a distance metric. The Euclidean distance metric is often used, although a dot product metric is often used as well. When using the dot product distance, the data vectors are often normalized beforehand and the reference vectors are

normalized at each step. In such cases dot product metric is equivalent to using the cosine measure.

The update step is the most complicated. Let $m_1, \ldots, m_k$ be the reference vectors associated with each neuron. (For a rectangular grid, note that k = number of rows * number of columns.) For time step t, let s be the current training point and assume that the closest reference vector to s is m_c, the reference vector associated with neuron c. Then, for time $t + 1$, the i^{th} reference vector is updated by using the following equation. (We will see shortly that the update is really restricted to reference vectors whose neurons are in a small neighborhood of neuron c.)

$$m_i(t+1) = m_i(t) + h_{ci}(t)\,(s(t) - m_i(t))$$

Thus, at time t, a reference vector, $m_i(t)$, is updated by adding a term, $h_{ci}(t)\,(s(t) - m_i(t))$, which is proportional to the difference, $s(t) - m_i(t)$, between the reference vector, $m_i(t)$, and the training point, x. $h_{ci}(t)$ determines the effect that the difference, $s(t) - m_i(t)$, will have and is chosen so that a) it diminishes with time and b) it enforces a neighborhood effect, i.e., the effect of a point is strongest on the neurons closest to the neuron c. Typically, $h_{ci}(t)$ is chosen to be one of the following two functions:

$$h_{ci}(t) = \alpha(t)\ \exp(-\|r_c - r_i\|^2 / (2\sigma^2(t)) \qquad \text{(Gaussian function)}$$

$$h_{ci}(t) = \alpha(t)\ \text{ if } \|r_c - r_i\| \leq \mathit{threshold}, \text{ 0 otherwise} \qquad \text{(step function)}$$

These function require quite a bit of explanation. $\alpha(t)$ is a learning rate parameter, $0 < \alpha(t) < 1$, which decreases monotonically and which controls the rate of convergence. $r_i = (x_i, y_i)$ is the two-dimensional point which gives the grid coordinates of the i^{th} neuron. $\|r_c - r_i\|$ is the Euclidean distance of the two neurons, i.e., $\sqrt{(x_i - x_c)^2 + (y_i - y_c)^2}$. Thus, we see that for neurons that are "far" from neuron c, the influence of training point s will be either greatly diminished or non-existent. Finally, note that σ is the typical Gaussian "variance" parameter and controls the width of the neighborhood, i.e., a small σ will yield a small neighborhood, while a large σ will yield a wide neighborhood. The *threshold* used for the step function also controls the neighborhood size.

(It is important to emphasize that it is the neighborhood updating technique that enforces a relationship (ordering) between reference vectors associated with neighboring neurons.)

Deciding when training is complete is an important issue. Ideally, the training procedure should continue until convergence occurs, i.e., until the reference vectors are not changing much. The rate of convergence will depend on a number of factors, e.g., the data and $\alpha(t)$. We will not discuss these issues further (see [Koh97]) except to mention that the most important issue, as with neural network in general, is the fact that convergence is often slow.

7.4.3. Applications

Once the SOM vectors are found, they can be used for many purposes besides clustering. For example, with a two-dimensional SOM it is possible to associate various

quantities at the grid points associated with each neuron and visualize the results via various types of plots. For example, plotting the number of points associated with each reference vector (neuron) yields a surface plot that reveals the distribution of points among neurons (clusters). (A two-dimensional SOM is a non-linear projection of the original probability distribution function into two dimensions. This projection attempts to preserve topological features and thus, SOM has been compared to the process of "pressing a flower.")

Besides data exploration, SOM and its supervised learning variant, LVQ (Learning Vector Quantization), have been used for many useful tasks, e.g., image segmentation, organization of document files, speech processing, etc. There are thousands of papers that deal with SOM neural networks and the basic techniques have been extended in many directions. For a good introduction and more than 2000 references see [Koh97].

7.5. Fuzzy Clustering

If data points are distributed in well-separated groups, then a crisp classification of the points into disjoint clusters seems like an ideal approach. However, in most cases, the points in a data set cannot be partitioned into well-separated clusters and thus, there will be a certain arbitrariness in assigning a point to a particular cluster, e.g., consider a point that lies near the boundary of two clusters, but is just slightly closer to one of the two clusters. Thus, for each point, x_i, and each cluster, C_j, it would be beneficial to determine a weight, w_{ij}, that indicates the "degree" to which x_i belongs to C_j.

For probabilistic approaches, e.g., mixture models, $w_{ij} = p_{ij}$, the probability that x_i belongs to c_j. While this approach is useful in many situations, there are significant limitations to probabilistic models, i.e., times when it is difficult to determine an appropriate statistical model, and thus, it would be useful to have non-probabilistic clustering techniques that provide the same "fuzzy" capabilities. Fuzzy clustering techniques [HKKR99] are based on fuzzy set theory [KY95] and provide a natural technique for producing a clustering where membership weights (the w_{ij}) have a natural (but not probabilistic) interpretation. In this section we will briefly describe the general approach of fuzzy clustering and provide a specific example in terms of fuzzy c-means (fuzzy K-means).

Lofti Zadeh introduced fuzzy set theory and fuzzy logic in 1965 [Zah65] as a way to deal with imprecision and uncertainty. Briefly, fuzzy set theory allows an object to partially belong to a set with a degree of membership between 0 and 1, while fuzzy logic allows a statement to be true with a degree of certainty between 0 and 1. (Traditional set theory and logic are special cases of their fuzzy counterparts which restrict the degree of set membership or the degree of certainty to be either 0 or 1.) Fuzzy concepts have been applied to many different areas, including control, pattern recognition, and data analysis (classification and clustering).

Consider the following example of fuzzy logic. The degree of truth of the statement "It is cloudy" can be defined to be the percentage of cloud cover in the sky, i.e., if the sky is 50% covered by clouds, then we would assign "It is cloudy" a degree of truth of 0.5. If we have two sets, "cloudy days" and "non-cloudy days," then we can similarly assign each day a degree of membership in the two sets. Thus, if a day were 25% cloudy,

it would have a 25% degree of membership in "cloudy days" and a 75% degree of membership in "non-cloudy" days.

We now define what fuzzy clusters are. Consider a set of data points X = {x_1, …, x_m}, where each point, x_i, is an m dimensional point, i.e., $x_i = (x_{i1}, \ldots, x_{im})$. We define a collection of fuzzy clusters, $C_1, C_2, \ldots, C_k$ to be a subset of all possible fuzzy subsets of X. (This simply means that the membership weights (degrees), w_{ij}, have been assigned values between 0 and 1 for each point, x_i, and each cluster, C_j.) However, we also want to impose the following "reasonable conditions on the clusters:

1. All the weights for a given point add up to 1.
   $$\sum_{j=1}^{k} w_{ij} = 1$$
2. Each cluster (fuzzy subset) "contains" at least one point, but not all of the points.
   $$0 < \sum_{i=1}^{m} w_{ij} < m$$

These two conditions ensure that the clusters form what is called a *fuzzy psuedo-partition*.

While there are many types of fuzzy clustering – indeed, many data analysis algorithms can be "fuzzified" – we will only consider the fuzzy version of K-means, which is called fuzzy c-means. (In the clustering literature, the version of K-means which does not use incremental updates of cluster centroids is sometimes referred to as c-means and this was the term adapted by the fuzzy people for the fuzzy version of K-means.) We will discuss some preliminaries and then present the fuzzy c-means algorithm.

First notice that we can also extend the definition of a centroid of a cluster to fuzzy sets in a straightforward way. For a cluster, C_j, the corresponding centroid, c_j, is defined by the following equation: $c_j = \dfrac{\sum_{i=1}^{m} w_{ij}^m x_i}{\sum_{i=1}^{m} w_{ij}^m}$, where m is a parameter, $1 < m < \infty$, that determines the influence of the weights (the membership degrees). Thus, the fuzzy centroid definition is much like the traditional definition except that all points are considered (any point can belong to any cluster, at least somewhat) and the contribution of each point to the centroid is weighted by its membership degree. In the case of traditional crisp sets, i.e., all w_{ij} are either 0 or 1, this definition reduces to the traditional definition of a centroid.

Furthermore, we can also modify the definition of overall cluster error.

Error($C_1, C_2, \ldots, C_k$) = $\sum_{j=1}^{k} \sum_{i=1}^{m} w_{ij}^m \left\| x_i - c_j \right\|$, which is just the weighted version of the typical K-means error.

Lastly notice that during the c-means clustering process we will need to update the weights w_{ij} associated with each point and cluster. This can be accomplished by using the following equation.

$$w'_{ij} = \left(\sum_{p=1}^{k} \left(\frac{\left\| x_i - c_j \right\|^2}{\left\| x_i - c_p \right\|^2} \right)^{\frac{1}{m-1}} \right)^{-1}$$

However, if $\left\| x_i - c_j \right\|^2 = 0$ for one or more values of j, $1 \le j \le k$, then we need to use a different method of updating the weights associated with x_i. Since this corresponds to a situation where a point is the same as one or more centroids (hopefully centroids are unique, but they don't have to be), we want to set $w'_{ij} = 0$ for any cluster j where $x_i \neq c_j$ and split the weight of the point (which is 1) between the w'_{ij} where $x_i = c_j$. In the normal case, a vector will only be the same as one centroid and thus, we can just set all weights to 0, except for the w'_{ij} where $x_i = c_j$. In that case, we set $w'_{ij} = 1$.

After preceding discussion the description of the fuzzy c-means algorithm is straightforward.

Fuzzy c-means Clustering Algorithm

1. Select an initial fuzzy fuzzy psuedo-partition, i.e., assign values to all the w_{ij}. (Like usual this can be done randomly or in a variety of ways.)
2. Update the cluster centroids.
3. Update the w_{ij}.
4. If the change in the error is below a specified threshold or the absolute change in any w_{ij} is below a given threshold, then stop. Otherwise, go to step 2.

This algorithm is similar in structure to the K-means algorithm and also behaves in a similar way. In particular, fuzzy c-means works best for globular clusters, which are roughly the same size and density.

There are a few considerations when choosing the value of m. Choosing $m = 2$ simplifies the weight update formula. However, if m is chosen to be near 1, then fuzzy c-means behaves like traditional K-means. Going in the other direction, as m gets larger all the cluster centroids approach the centroid of the data. In other words, the partition becomes fuzzier as m increases.

We close this discussion of fuzzy clustering with a cautionary note. Fuzzy clustering, as presented here, deals well with situations where points are on the boundary of one or more clusters. It does not deal well with the situation where a point is an outlier, i.e., is not close to any cluster. The reason for this is that we assumed that all points have a weight of 1, a weight which is split between the different clusters. Thus, if we have a situation, as illustrated in figure 18, a point can have a relatively high weight with respect to one or more clusters, but not really be a "true" member of any cluster.

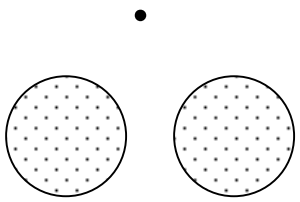


Figure 18: An outlier that is not really associated with either cluster.

In this case the point should have low membership weights for both clusters, but then the weights do not add up to 1. However, with such an approach it would be possible to eliminate noise and outliers effectively, i.e., just eliminate points that do not have high membership weights in any cluster. Indeed, there is a variation of fuzzy c-means that takes this approach [HKKR99], but it has other problems, e.g., a tendency for all the centroids to become very close together.

8. Related Topics

The following sections deal with topics that are usefully related to data mining.

8.1. Nearest-neighbor search (Multi-Dimensional Access Methods)

Given a set of complicated data, e.g., images, text strings, DNA sequences, polygons, etc., two problems arise:

1) **How can items be located efficiently?** While a "representative" vector is commonly used to "index" these objects, these data points will normally be very sparse in the space, and thus, it is not feasible to use an array to store the data. Also, many sets of data do not have any features that constitute a "key" that would allow the data to be accessed using standard and efficient database techniques.
2) **How can similarity queries be conducted?** Many applications require the nearest neighbor (or the k nearest neighbors) of a point. For example, in a database of photographs, a user may want to find all the pictures similar to a particular photograph. To do this photographs are represented as vectors and a "similarity" search is conducted. As another example, some of the clustering techniques discussed earlier (CURE, Chameleon) depended on data structures and algorithms that allowed the nearest neighbors of a point to be located efficiently.

Techniques for nearest-neighbor search are often discussed in papers about multi-dimensional access methods or spatial access methods. Strictly speaking the topic of multi-dimensional access methods is broader than nearest-neighbor search since it addresses all of the many different types of queries and operations that a user might want to perform on multi-dimensional data.

A large amount of work has been done in the area of nearest neighbor search (and multi-dimensional access methods). Recent developments include the SS-tree [JW96], the SR-tree [KS97], the X-tree [BKK96], the GNAT tree [Bri95], the M-tree [CPZ97], and the "pyramid technique" [BBK98]. (The X-tree is a descendant of the older, but frequently mentioned R* tree, which is in turn a descendant of the R tree.) A good survey of nearest-neighbor search, albeit from the slightly more general perspective of multi-dimensional access methods is given by [GG98].

The basic approach is to create tree-based structures, such that the "closeness" of the data increases as the tree is traversed from top to bottom. Thus, the nodes at the bottom of the tree represent "clusters" of data that are relatively cohesive. Indeed, one of the simplest techniques for generating a nearest neighbor tree is to cluster the data into *K* clusters and then, recursively break each cluster into subclusters until the subclusters consist of individual points. The nearest-neighbor tree consists of the clusters and subclusters generated along the way.

A k-nearest-neighbor query is conducted as follows:

1) Add the children of the root node to a search queue.
2) While the search queue is not empty, take a node off the search queue.
   a) Evaluate whether that node and any of its descendants are "close enough" to be considered for the search.
   b) If not, then discard the subtree represented by this node.
   c) Otherwise, if the node is not a leaf node, i.e., a data point, add the children of this node to a search queue.
   d) If the node is a leaf node, then add the node to a list of possible solutions.
3) Sort the list of possible solutions by distance from the query point and return the k-nearest neighbors.

This sounds fairly simple and it seems as though nearest neighbor trees would be an easy way to cluster data and/or to make practical use of a clustering technique. However, there are significant problems.

1) One of the goals of many nearest-neighbor tree techniques is to serve as efficient secondary storage based access methods for non-traditional databases, e.g., spatial databases, multimedia databases, document databases, etc. Because of requirements related to page size and efficient page utilization, "natural" clusters may be split across pages or nodes. However, data is normally highly clustered and this can be used for "real" clustering as shown by the improvements made to CLARANS [EKX95] by the use of an R* tree.

2) Because of the nature of nearest-neighbor trees, the tree search is a branch-and-bound technique and needs to search large parts of the tree, i.e., at any particular level, many children and their descendants may need to be examined.

   To see this, consider a point and all points that are within a given distance of it. This hyper-sphere (or hyper-rectangle in the case of multi-dimensional range queries) may very well cut across a number of nodes (clusters) - particularly if the point is on the edge of a cluster and/or the query distance being considered is greater than the distance between clusters.

   More specifically, it is "hard" for the algorithms that construct nearest neighbor trees to avoid a significant amount of overlap in the volumes represented by different nodes in the tree. In [BKK96], it has been shown that the degree of overlap in a frequently used nearest neighbor tree, the R* tree, approaches 100% as the dimensionality of the vectors exceeds 5. Even in two dimensions the overlap was about 40%. Thus, the X-tree (proposed as a successor to the R* tree in [BKK96])

basically becomes linear search for high dimensional data. Other nearest neighbor search techniques suffer from the same problem.

3) The nature of high dimensional spaces. It has been shown that there are significant problems that arise with nearest neighbor searches in high-dimensional space.

In [BGRS99] it was shown that the concept of "nearest neighbor" is not meaningful in many situations. Minimum and maximum distances in high dimensional space tend to be very similar. Thus, unless there is significant clustering in the data and/or the query ranges stay within individual clusters, then the points returned by nearest neighbor queries are not much closer to the query point than to the points not returned. In this latter case, then (to use the terminology of [BGRS99]) the nearest neighbor query is "unstable."

Recently, in [BBK98], there has been some work on developing techniques that avoid this problem. However, this "pyramid" technique was restricted to hyper-cube shaped queries, and many similarity queries are spherical.

In some cases, a linear scan can work better than more sophisticated techniques.

4) It is not possible to discard outliers. All data points must be stored, and if some of the data points do not fit particularly well in clusters, then this often has deleterious effects on lookups of other data points.

8.2. Pattern Recognition

Clustering has been used in a number of different ways in pattern recognition. [DJ88] is a clustering book that is strongly oriented toward the area of pattern recognition giving a number of different and extensive examples of the use of clustering in pattern recognition. This paper only mentions some examples briefly.

In certain cases, e.g., the segmentation of an image into different regions, it is hard to "train" a recognition system to behave properly because there is such a wide variation in the possible input, and clear cut classes are not readily available. In such instances, the use of the appropriate clustering technique can prove very effective.

As was noted before, if clear-cut classes are available, then other techniques, e.g., discriminant analysis, are more appropriate. However, in some cases clustering analysis may be used to find a preliminary assignment of classes that are then refined by a domain expert.

The use of clustering to reduce the number of patterns that need to be considered in pattern recognition is discussed below.

8.3. Exploratory Data Analysis

Cluster analysis is a very important part of exploratory data analysis, although cluster analysis is just one of the tools used in exploratory data analysis to understand patterns that may occur in data.

When cluster analysis is used for exploratory data analysis, some of the more user-oriented factors assume additional importance. In particular, it is important for the user to be able to easily interpret the results of the cluster analysis. As mentioned above,

various clustering techniques differ in the kinds of results that are produced. Some produce lists of objects, some produce hyper-ellipsoids, and some produce hyper-rectangles.

In some cases, techniques (like multi-dimensional scaling [DJ88]) that map medium and high dimensional data to 1, 2, or 3 dimensions can be very helpful. Unfortunately, the distortions introduced by such mappings sometimes render the process useless. These techniques are more likely to work better for medium dimensionalities than for high dimensionalities.

Finally, exploratory data analysis is often an integral part of a clustering analysis. A user looks at the results of an initial clustering and, as a result, changes the parameters of a clustering algorithm or chooses a new clustering algorithm. Numerous iterations may be required before a suitable clustering is produced.

9. Special Topics

9.1. Missing Values

Omitted from preliminary version.

9.2. Combining Attributes

9.2.1. Monothetic or Polythetic Clustering

The features of an object can be used for clustering in an incremental or non-incremental way. While some techniques use one variable at a time (monothetic), most approaches use all features at once (polythetic) to calculate the distance or similarity. .([KR90] provides an example of Monothetic clustering algorithm.)

9.2.2. Assigning Weights to features

In some cases, the attributes are weighted so as to give more weight to certain attributes, but this typically only affects how the proximity of points is calculated and not how the clustering actually works. In some sense, the use of attribute weights can be regards as a more general version of feature selection.

10. Conclusions

The following conclusions seem most important:

- **Cluster analysis is still an active field of development**. In the areas of statistics (mixture models), computer science (Data Mining, machine learning, nearest neighbor search), pattern recognition, and vector quantitization, there is still a lot of work being done.

- **Many cluster analysis techniques do not have a strong formal basis**. While some techniques make use of formal mathematical methods, they often do not work better than more informal methods.

- **Cluster analysis is a rather ad-hoc field**. Almost all techniques have a number of arbitrary parameters that can be "adjusted" to improve results. It remains to be seen, whether this represents a temporary situation, or is an unavoidable use of problem and

domain specific heuristics.

- **There are a wide variety of clustering techniques**. Some would argue that the wide range of subject matter, size and type of data, and differing user goals makes this inevitable, and that cluster analysis is really a collection of different problems that require a variety of techniques for their solution. The relationships between the different types of problems and solutions are often not clear.

- **Comparisons among different clustering techniques are difficult**. While every article that presents a new clustering technique shows its superiority to other techniques, it is hard to judge how well the technique will really do.

- **There don't seem to be any standard benchmarks**. Thus, the authors of new techniques are reduced to devising their own comparisons to previous techniques. Usually authors obtain a few of the data sets used by previous authors and run comparisons based on those. Even the criteria to be measured are not standard.

- **All techniques seem to impose a certain structure on the data and yet few authors describe the type of limitations being imposed**. In [DJ88], the authors mention that the performance of a clustering algorithm on uniformly distributed data can sometimes be very informative. In general, it might be useful to understand what kind of artifacts particular clustering techniques introduce.

- **In spite of all these problems, clustering analysis is a useful (and interesting) field**. As mentioned in the introduction, many people use cluster analysis for a wide variety of useful tasks.

List of Articles and Books for Clustering for Data Mining

Note that web addresses are given for many of the articles. This usually takes you to a home page of one of the authors or to a list of publications. In the first case, it is usually straightforward to find the article by looking at the person's publications, although in a few cases a reference is all that is provided. Additionally, the proceedings of some conferences can be found online: SIGMOD and KDD (99 onward) at ACM Digital Library, www.acm.org, VLDB at the Very Large Database home page, www.vldb.org, ICDE (99 onward) at http://www.computer.org/proceedings/. A good place to locate the home pages of authors is at http://ftp.ust.hk/dblp/db/index.html. Finally, another good general site is the Research Index site at NEC, http://citeseer.nj.nec.com/cs.

The following three books provide an introduction to and overview of Cluster Analysis

[DJ88] Richard C. Dubes and Anil K. Jain, (1988), *Algorithms for Clustering Data*, Prentice Hall.

A survey and introduction to cluster analysis that is frequently referenced in the literature. It devotes considerable attention to cluster validity. It is somewhat oriented toward pattern recognition and has a whole chapter devoted to the use of clustering in pattern recognition. It also has appendices that summarize various areas related to cluster analysis.

[KR90] L. Kaufman and P. J. Rousseeuw, (1990), *Finding Groups in Data: an Introduction to Cluster Analysis*, John Wiley and Sons.

Probably the most commonly referenced text on clustering that I encountered, perhaps because it is still in print. Among other things, this text explains clustering based on medoids, which are objects that are the most representative of the other objects in a cluster. The clustering techniques described in this book are implemented in the S-Plus statistical system.

[OD74] Patrick L. Odell and Benjamin S. Duran, (1974), *Cluster Analysis: A Survey*, Springer-Verlag.

This book is somewhat dated, but is fairly short and basic. Also, since a lot of basic clustering techniques were in place by the time this book was written, most of the material it covers is still applicable.

The following papers describe specific techniques, algorithms, or systems used for clustering in Data Mining. The techniques described in these papers are described in section 4 of this document. Only very brief comments will be made here.

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopoulos, and Prabhakar Raghavan, (1998), *Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications*, Technical Report, IBM Almaden Research Center.
http://www.almaden.ibm.com/cs/people/ragrawal/

Describes CLIQUE, a density-based algorithm that has a number of good properties, including the ability to find lower dimensional clusters in high dimensional subspaces. This algorithm is similar in flavor to the APRIORI algorithm used for finding frequent itemsets, and finds higher dimensional regions by only considering those regions that are associated with lower dimensional regions of high density.

[CHY96] Ming-Syan Chen, Jiawei Han, and Philip, S. Yu, (1996*), Data Mining: An Overview from Database Perspective*, IEEE Transactions on Knowledge and Data Engineering, 8(6): 866-883.
http://db.cs.sfu.ca/sections/publication/kdd/kdd.html

Provides a general overview of data mining techniques. Included is an overview of CLARANS, enhanced CLARANS, and BIRCH. This paper has an extensive discussion of the characteristics that data mining techniques should have.

[EKSWX98] Martin Ester, Hans-Peter Kriegel, Jorg Sander, Michael Wimmer, Xiaowei Xu, (1998), *Incremental Clustering for Mining in a Data Warehousing Environment*, Proceedings of 1998 International Conference on Very Large Data Bases (VLDB'98), New York, USA.
http://www.dbs.informatik.uni-muenchen.de/index_e.html

Describes how the DBSCAN clustering algorithm can be made incremental for use in a data warehouse environment.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu, (1996), *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, pp. 226-231.
http://www.dbs.informatik.uni-muenchen.de/index_e.html

Describes, DBSCAN, a density-based clustering algorithm that handles noise well. While this algorithm is presented and evaluated in the context of spatial databases, there is no reason that it could not be used in a more general setting.

[EKX95] Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu, (1995*) Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification*, Proceedings of the 4th International Symposium on Large Spatial Databases (SSD'95), Portland, ME, 1995, in: Lecture Notes in Computer Science, Vol. 591, Springer, 1995, pp. 67-82. http://www.dbs.informatik.uni-muenchen.de/index_e.html

Describes a number of techniques for making CLARANS more efficient. The basic idea is to use an R* trees to store the data being considered. . (R* trees are a type of nearest-neighbor tree that try to store data points in the same page if the data points are "close" to one another.)

[GHC99] Sanjay Goil, Harasha Nagesh, and Alok Choudhary, (1999), *MAFIA: Efficient and Scalable Subspace Clustering for Very Large Data Sets*, Technical Report Number CPDC-TR-9906-019, Center for Parallel and Distributed Computing, Northwestern University, June 1999. http://www.ece.nwu.edu/~sgoil/

Describes MAFIA (Merging of Adaptive Finite Intervals (and more than a CLIQUE), which is an modification of CLIQUE that runs faster and finds better quality clusters. pMAFIA is the parallel version.

[GRGPF99] Venkatesh Ganti, Raghu Ramakrishnan, Johannes Gehrke, Allison L. Powell, James C. French, (1999*), Clustering Large Datasets in Arbitrary Metric Spaces*, ICDE 1999, pp. 502-511. http://www.cs.wisc.edu/~raghu/

Describes Bubble and Bubble-FM, which are clustering algorithms based on BIRCH*, an extension to the BIRCH framework to handle non-metric data. One of the key ideas is to define a new cluster feature based on the idea of the representative points of a cluster. Bubble-FM also makes used of a technique, multi-dimensional scaling, to map non-metric data into a metric space. This permits fewer and cheaper distance calculation in those cases where distance calculations are very expensive.

[GRS98] Sudipto Guha, Rajeev Rastogi, Kyuseok Shim, (1998*), CURE: An Efficient Clustering Algorithm for Large Databases*, ACM SIGMOD Conference, 1998, pp. 73-84. http://www.bell-labs.com/user/rastogi/

Describes a clustering algorithms that describes a cluster by widely scattered representative points that are shrunk by a constant factor toward the center of a cluster to reduce the effect of outliers. The basic clustering technique used is similar to single link and sampling and partitioning are used to make the method scalable to large data sets. CURE was shown to be effective at finding 2-D spatial clusters that were of non-circular shapes and which had widely varying sizes.

[GRS99] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim, (1998*), ROCK: A Robust Clustering Algorithm for Categorical Attributes*, In Proceedings of the 15th International Conference on Data Engineering, 1999.
http://www.bell-labs.com/user/rastogi/

Describes an approach for clustering algorithm for data with categorical and Boolean attributes. It redefines the distances between points to be the number of shared neighbors whose strength is greater than a given threshold and then uses a hierarchical clustering scheme to cluster the data.

[HK98] Alexander Hinneburg Daniel. A. Keim and: *An Efficient Approach to Clustering in Large Multimedia Databases with Noise*, Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, AAAI Press.
http://www.informatik.uni-halle.de/~keim/

Describes a density-based approach to clustering data which models the influence of each data point via influence (kernel) functions and then finds clusters using the resulting global density function. Initially, each data point is associated with a local maxima, forming a center based clusters. Clusters whose local maxima is "not dense enough" are classified as noise and discarded, while remaining clusters are merged if there is a path connecting their local maxima which is of "high enough" density.

[KHK99a] George Karypis, Eui-Hong Han, and Vipin Kumar, (1999) *CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling*, IEEE Computer, Vol. 32, No. 8, August, 1999. pp. 68-75.
http://www-users.cs.umn.edu/~karypis/

Describes a clustering algorithm that combines an initial partitioning of the data using an efficient graph-partitioning algorithm with a novel hierarchical clustering scheme that dynamically models clusters. The key idea is that two clusters will be merged only if the resulting cluster is similar to the original clusters in, i.e., self-similarity is preserved. While applicable to a wide range of data, Chameleon has specifically been shown to be superior to DBSCAN and CURE for clustering spatial data.

[KHK99b] George Karypis, Eui-Hong Han, and Vipin Kumar, (1999*) Multi-level Refinement for Hierarchical Clustering*. (Unpublished)
http://www-users.cs.umn.edu/~karypis/

Describes an approach to improving hierarchical clustering that is related to the technique of multi-level refinement that is commonly used to improve the solution to graph partitioning problems.

[HKKM97] Eui-Hong Han, George Karypis, Vipin Kumar, and Bomshad Mobasher, (1997), *Clustering In a High-Dimensional Space Using Hypergraph Models*, Technical Report TR-97-063, Department of Computer Science, University of Minnesota, Minneapolis, Minnesota.
http://www-users.cs.umn.edu/~karypis/

Describes the technique of hypergraph based clustering in the context of market basket data. This technique uses frequent itemsets and their related association rules to construct weighted hyperedges between the items being considered for clustering. A hypergraph based clustering algorithm (HMETIS) is then used to create the clusters. This algorithm naturally finds lower dimensional clusters in a higher dimensional space and has straightforward measures for detecting outliers and valid clusters.

[NH94] Raymond Ng and Jiawei Han, (1994), *Efficient and Effective Clustering Method for Spatial Data Mining*, Proceedings of 1994 International Conference on Very Large Data Bases (VLDB'94), Santiago, Chile, September 1994, pp. 144-155.
http://db.cs.sfu.ca/sections/publication/kdd/kdd.html

Describes CLARANS, a medoid based clustering algorithm for spatial databases. This was one of the first clustering algorithms specifically oriented towards data mining.

[SCZ98] Gholamhosein Sheikholeslami, Surojit Chatterjee and Aidong Zhang (1998), *Wavecluster: A multi-resolution clustering approach for very large spatial databases*. In Proceedings of the 24th VLDB Conference.
http://www.cs.buffalo.edu/~surojit/

WaveCluster is a grid-based clustering algorithm for low-dimensional data. WaveCluster transforms the data so that it looks like image data, i.e., a rectangular grid of pixels each of which has an integer values (the count of data points in the grid cell) and then using image processing techniques (wavelets) to segment the "image" and produce a clustering of the points.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny, (1996), *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, in Proceedings of ACM SIGMOD International Conference on Data Management, June 1996, Canada.
http://www.cs.wisc.edu/~raghu/

Describes BIRCH, a very efficient clustering technique based on a CF (Clustering Feature), tree. This tree is a summary of the data in the form of a hierarchical clustering. This rough clustering can be refined in later steps. This basic CF tree structure can be used with a number of underlying clustering algorithms.

The following papers discuss nearest neighbor clustering techniques.

[GK78] K. C. Gowda and G. Krishna, (1978), *Agglomerative Clustering Using the Concept of Mutual Nearest Neighborhood, Pattern Recognition*, Vol. 10, pp. 105-112.

[JP73] R. A. Jarvis and E. A. Patrick, (1973), *Clustering Using a Similarity Measure Based on Shared Nearest Neighbors*, IEEE Transactions on Computers, Vol. C-22, No. 11, November, 1973.

The following papers discuss association rules.

[AS97] Rakesh Agrawal, and Ramakrishnan Srikant, (1997), *Fast Algorithms for Mining Association Rules*, In Proceedings of the 20th VLDB Conference, pages 487-499, Santiago, Chile.
http://www.almaden.ibm.com/cs/people/ragrawal/

Describes the APRIORI algorithm (and a couple variants), a very efficient algorithm for finding frequent itemsets and association rules.

[AS97b] Rakesh Agrawal and Ramakrishnan Srikant, (1997), *Mining Generalized Association Rules*, Technical Report, IBM Almaden Research Center.
http://www.almaden.ibm.com/cs/people/ragrawal/

Describes the mining of generalized association rules when items are grouped into a taxonomy. Also contains an interest measure for rules that can be used to prune uninteresting rules.

[BMS97] Sergey Brin, Rajeev Motwani, and Craig Silverstein, (1997*), Beyond Market Baskets: Generalizing Association Rules to Correlations*, In Proceedings of 1997 ACM-SIGMOD International Conference on Management of Data, Tucson Arizona.
http://theory.stanford.edu/people/motwani/motwani.html

Discusses the limitations of association rules and proposes using correlations instead.

The following papers are more statistically oriented.

[FR98] C. Fraley and A. E. Raferty, (1998), *How Many Clusters? Which Clustering Method? Answers Via Model-Based Cluster Analysis*, Technical Report No. 329, Department of Statistics, University of Washington, Seattle, Washington.
http://www.stat.washington.edu/fraley/

Contains a readable description of a mixture model based clustering technique that can determine both the clusters and the number of clusters. The technique described is available as software that works with the S-Plus statistical package.

[CHS90] Peter Cheeseman, Robin Hanson, and John Stutz, (1990), *Bayesian Classification Theory*, Technical Report FIA-90-12-7-01, NASA Ames Research Center, Moffet Field, California. http://ic-www.arc.nasa.gov/ic/projects/bayes-group/people/cheeseman/home.html

Contains a description of the theory on which the clustering program, AutoClass is based.

[CS96] Peter Cheeseman and John Stutz, (1996), Bayesian Classification (AutoClass): Theory and Results, In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smith, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 153-180, AAAI/MIT Press. http://ic-www.arc.nasa.gov/ic/projects/bayes-group/people/cheeseman/home.html

Describes the AutoClass clustering program and its use in analyzing sensor and DNA data. AutoClass uses a mixture model based approach and was one of the first programs used for clustering in databases.

[HP98] Thomas Hofmann and Jan Puzicha, (1998), *Statistical Models for Co-occurrence Data*, AI Memo No. 1625, Artificial Intelligence Laboratory, Massachusetts Institute of Technology. http://www.ai.mit.edu/people/hofmann/

Describes a mixture model approach to modeling co-occurrence data, i.e., data of the form (x,y), where x and y are any members of two sets, X and Y. This approach has a number of possible applications, including data mining and information retrieval.

The following papers and books are related to fuzzy clustering.

[HKKR99] Frank Hoppner, Frank Klawonn, Rudolf Kruse, and Thomas Runkler, (1999), *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis, and Image Recognition*, John Wiley & Sons. http://www.fuzzy-clustering.de/

A fairly advanced book that provides a discussion of fuzzy clustering techniques including some interesting approaches for finding clusters that are shells, polygons or lines. Fuzzy clustering software is available from the website listed above.

[KY95] George J. Klir and Bo Yuan, (1995), *Fuzzy Sets and Fuzzy Logic*, Prentice Hall PTR.

A comprehensive introduction to fuzzy set theory and fuzzy logic which covers a broad range of topics. The books includes a brief, but clear description of fuzzy c-means and also discussion an approach to fuzzy hierarchical clustering.

[Zah65] Lofti Zadeh, (1965), *Fuzzy Sets*, Information and Control, Vol. 8, 338:353.

This is the original paper on fuzzy sets.

The following papers are related to vector quantization and maximum entropy related clustering approaches.

[BK93] Joachim Buhmann and Hans Kuhnel, (1993), *Vector Quantization with Complexity Costs*, IEEE Transactions on Information Theory 39:1133-1145. http://www-dbv.informatik.uni-bonn.de/papers.html

Describes a maximum entropy approach to vector quantization that can simultaneously determine the clusters and the number of clusters. Also contains a comparison to the K-means algorithm in the field of image compression.

[BH97] Joachim M. Buhmann and Thomas Hofmann, (1997), *Pairwise Data Clustering by Deterministic Annealing*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 19, No. 1, January 1997. http://www-dbv.informatik.uni-bonn.de/papers.html

Describes a maximum entropy approach to clustering which is similar to the previous article. This approach uses the deterministic annealing optimization technique.

The following two papers are related to graph partitioning.

[KK98a] George Karypis and Vipin Kumar, (1998), *hMETIS 1.5: A hypergraph partitioning package*, Technical report, Department of Computer Science, University of Minnesota, 1998. http://www-users.cs.umn.edu/~karypis/metis/

[KK98b] George Karypis and Vipin. Kumar, (1998), *METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system*, Technical report, Department of Computer Science, University of Minnesota, 1998. http://www-users.cs.umn.edu/~karypis/metis/

The following papers are related to clustering in information retrieval.

[BD95] Michael W. Berry and Susan T. Dumais, (1995), *Using Linear Algebra for Intelligent Information Retrieval*, SIAM Review, 37:573-595. http://research.microsoft.com/~sdumais/

Describes the technique of Latent Semantic Indexing. This is a dimensionality reduction technique that uses the data matrix, i.e., the matrix of the original data vectors. This technique is used extensively in information retrieval.

[FO95] Christos Faloutsos and Douglas W. Oard, (1995), *A Survey of Information Retrieval and Filtering Methods*, Technical Report CS-TR-3514, University of Maryland, College Park, Maryland.
http://www.cs.cmu.edu/~christos/

Describes a variety of techniques used in information retrieval. Among the topics covered are clustering and Latent Semantic Indexing.

[SKK00b] Michael Steinbach, George Karypis, and Vipin Kumar, *A Comparison of Document Clustering Techniques*, University of Minnesota, Technical Report #00-034, 2000.

This paper presents the results of an experimental study of some common document clustering techniques. In particular, it compares the two main approaches to document clustering, agglomerative hierarchical clustering and K-means. (For K-means we used a "standard" K-means algorithm and a variant of K-means, "bisecting" K-means.) Hierarchical clustering is often portrayed as the better quality clustering approach, but is limited because of its quadratic time complexity. In contrast, K-means and its variants have a time complexity which is linear in the number of documents, but are thought to produce inferior clusters. Sometimes K-means and agglomerative hierarchical approaches are combined so as to "get the best of both worlds." However, the paper's results indicate that the bisecting K-means technique is better than the standard K-means approach and as good or better than the hierarchical approaches that were tested.

The following papers are related to nearest neighbor search and nearest neighbor trees. In the literature such techniques are found in the related area of multi-dimensional access methods.

[BBK98] Stefan Berchtold, Christian Bohm, Hans-Peter Kriegel, (1998), *The Pyramid Technique: Towards Breaking the Curse of Dimensionality*, Proceedings of 1998 International Conference on the Management of Data (SIGMOD '98), Seattle, Washington.
http://www.dbs.informatik.uni-muenchen.de/index_e.html

Describes a new indexing method for high dimensional spaces that attempts to overcome some of the problems of previous methods, i.e., for high dimensional spaces linear search is often better. The technique bounds the data by a hyper-cube, splits the hyper-cube into pyramids whose tips meet at the center of the cube, and then slices the pyramids into sections. By doing this, the technique is able to map each point in a high dimensional space into a linear space. B+ trees can then be used to manage the data. So far this technique only works well for hyper-cube shaped queries.

[BGRS99] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft, (1998), *When is "Nearest Neighbor" Meaningful?*, International Conference on Database Theory, 1999, pp. 217-235.
http://www.cs.wisc.edu/~raghu/

An article on the problems encountered by nearest neighbor search in high dimensions. This paper proposes the ε-Radius Nearest Neighbor problem, which is to find all points within a radius of DMIN * (1+ε) of a query point, where DMIN is the minimum distance from the query point to its closest neighbor. Such a query is only meaningful if the number of points within the radius is a small fraction of all the points. The paper then shows that for many situations, particularly if the number of dimensions is greater than 10 or 15, such queries are meaningless. A major contribution of this paper is to show that any new nearest neighbor search techniques should always be compared to a linear scan.

[BKK96] Stefan Berchtold, Daniel A. Keim, Hans-Peter Kriegel, (1996), *The X-tree: An Index Structure for High-Dimensional Data*, Proceedings of 1996 International Conference on Very Large Databases (VLDB'96), Bombay, India.
http://www.dbs.informatik.uni-muenchen.de/index_e.html

Describes the X-tree, a high dimensional indexing structure that is based on the R* tree. This paper contains an analysis of the number of nodes of the R* tree that have to be searched for a particular query. For dimensions more than 5, typically 90% of the tree must be searched. The X-tree tries to be more efficient, by allowing larger nodes, i.e., by going more to a linear search, as the dimensionality increases.

[Bri95] Sergey Brin, (1995), *Near Neighbor Search in Large Metric Spaces*, Proceedings of the 21st International Conference on Very Large DataBases, pp. 574-584, Zurich Switzerland.
http://www-db.stanford.edu/~sergey/

Describes the GNAT (Geometric Nearest Neighbor) tree. This tree provides a data structure for high dimensional data that attempts to exploit (model) the geometry of the data. This technique is oriented towards cases where distance calculations are relatively expensive and initial build time is not as important. The technique is not intended for cases where data is added or deleted. Finally, this technique is intended for data whose distance function is a metric.

[CPZ97] Paolo Ciaccia, Marco Patella, Pavel Zezula, (1997), *M-tree: An Efficient Access Method for Similarity Search in Metric Spaces*, Proceedings of 1997 International Conference on Very Large Databases (VLDB'97), pp. 426-435, Athens, Greece.
http://www.cs.unibo.it/~ciaccia/

Describes the M-tree, a high dimensional access technique for data whose distance function is a metric. This technique generates balanced trees and can be used as a secondary storage based access method (like the R*-tree, the X-tree, and the pyramid technique). This technique relies on a careful use of the triangle inequality to minimize potentially expensive distance calculations.

[GG98] Volker Gaede and Oliver Gunther, (1998), *Multidimensional Access Methods*, ACM Computing Surveys, Vol. 30, No. 2, http://www.wiwi.hu-berlin.de/~gaede/

A very understandable and comprehensive survey of multi-dimensional access methods.

[JW96] Ramesh Jain and David A. White, (1996), *Similarity Indexing with the SS-tree*, Proceedings of the 12th International Conference on Data Engineering, pp. 516-523, New Orleans, USA. http://vision.ucsd.edu/~jain/

Describes the SS-tree and demonstrates that it is somewhat better than the R* tree. Basically, the SS tree uses hyper-spherical clusters of points, while the R*-tree uses hyper-rectangular clusters. Like the R* tree, the SS tree can be used as a secondary storage access method.

[KS97] Norio Katayama and Shin'ichi Satoh, (1997), *The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries*, Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Tucson, Arizona. http://www.rd.nacsis.ac.jp/~katayama/homepage/research/srtree/English.html

Describes an improvement of the SS tree that uses the intersection of a hyper-sphere and a hyper-rectangle.

The following are papers on miscellaneous topics.

[Fi96] Doug Fisher, (1996), *Iterative Optimization and Simplification of Hierarchical Clusterings*, Journal of Artificial Intelligence 4 (1996) 147-179. http://www.vuse.vanderbilt.edu/~dfisher/

Describes techniques for improving hierarchical clusterings. A number of iterative techniques are introduced and evaluated. The viewpoints taken in this article is a that of machine learning.

[Ko97] T. Kohonen, (1997), *Self-Organizing Maps, Second Extended Edition*, Springer Series in Information Sciences, Vol. 30, Springer, Berlin, Heidelberg, New York, 1995, 1997. http://www.cis.hut.fi/teuvo/

[FL95] Christos Floutsos and King-Ip Lin, (1995), *Fastmap: A fast algorithm for indexing of traditional and multimedia databases*. Proceedings of SIGMOD 1995. http://www.cs.cmu.edu/~christos/

Describes the Fastmap algorithm. This algorithm takes maps a set of *m* points into Euclidean space in time linear in *m* and (approximately) preserves the distances

between the points. Such a mapping is an example of a technique called multi-dimensional scaling. These techniques are useful for giving a Euclidean vector representation to points that do not originally belong to a Euclidean space. A variation on Fastmap, IncrFastmap, is unique among multi-dimensional scaling techniques in that after the original mapping is defined, IncrFastmap can incrementally map new points without changing the original mapping.

[Ta96] Eric D. Taillard, (1998), *Heuristic Methods for Large Centroid Clustering Problems*, Technical Report IDSIA96-96, revision April 1998. IDSIA, Lugano, Switzerland.
http://www.idisia.ch/~eric/

Describes some new heuristic methods for solving centroid clustering problems that are important in operations research.