8

Cluster Analysis: Additional Issues and Algorithms

A large number of clustering algorithms have been developed in a variety of domains for different types of applications. No algorithm is suitable for all types of data, clusters, and applications. In fact, it seems that there is always room for a new clustering algorithm that is more efficient or better suited to a particular type of data, cluster, or application. Instead, we can only claim that we have techniques that work well in some situations. The reason is that, in many cases, what constitutes a good set of clusters is open to subjective interpretation. Furthermore, when an objective measure is employed to give a precise definition of a cluster, the problem of finding the optimal clustering is often computationally infeasible.

This chapter focuses on important issues in cluster analysis and explores the concepts and approaches that have been developed to address them. We begin with a discussion of the key issues of cluster analysis, namely, the characteristics of data, clusters, and algorithms that strongly impact clustering. These issues are important for understanding, describing, and comparing clustering techniques, and provide the basis for deciding which technique to use in a specific situation. For example, many clustering algorithms have a time or space complexity of $O(m^2)$ (m being the number of objects) and, thus, are not suitable for large data sets. We then discuss additional clustering techniques. For each technique, we describe the algorithm, including the issues it addresses and the methods that it uses to address them. We conclude this chapter by providing some general guidelines for selecting a clustering algorithm for a given application.

8.1 Characteristics of Data, Clusters, and Clustering Algorithms

This section explores issues related to the characteristics of data, clusters, and algorithms that are important for a broad understanding of cluster analysis. Some of these issues represent challenges, such as handling noise and outliers. Other issues involve a desired feature of an algorithm, such as an ability to produce the same result regardless of the order in which the data objects are processed. The discussion in this section, along with the discussion of different types of clusterings in Section 5.1.2 and different types of clusters in Section 5.1.3, identifies a number of "dimensions" that can be used to describe and compare various clustering algorithms and the clustering results that they produce. To illustrate this, we begin this section with an example that compares two clustering algorithms that were described in Chapter 5, DBSCAN and K-means. This is followed by a more detailed description of the characteristics of data, clusters, and algorithms that impact cluster analysis.

8.1.1 Example: Comparing K-means and DBSCAN

To simplify the comparison, we assume that there are no ties in distances for either K-means or DBSCAN and that DBSCAN always assigns a border point that is associated with several core points to the closest core point.

- Both DBSCAN and K-means are partitional clustering algorithms that assign each object to a single cluster, but K-means typically clusters all the objects, while DBSCAN discards objects that it classifies as noise.

- K-means uses a prototype-based notion of a cluster; DBSCAN uses a density-based concept.

- DBSCAN can handle clusters of different sizes and shapes and is not strongly affected by noise or outliers. K-means has difficulty with non-globular clusters and clusters of different sizes. Both algorithms can perform poorly when clusters have widely differing densities.

- K-means can only be used for data that has a well-defined centroid, such as a mean or median. DBSCAN requires that its definition of density, which is based on the traditional Euclidean notion of density, be meaningful for the data.

- K-means can be applied to sparse, high-dimensional data, such as document data. DBSCAN typically performs poorly for such data because the traditional Euclidean definition of density does not work well for high-dimensional data.

- The original versions of K-means and DBSCAN were designed for Euclidean data, but both have been extended to handle other types of data.

- DBSCAN makes no assumption about the distribution of the data. The basic K-means algorithm is equivalent to a statistical clustering approach (mixture models) that assumes all clusters come from spherical Gaussian distributions with different means but the same covariance matrix. See Section 8.2.2.

- DBSCAN and K-means both look for clusters using all attributes, that is, they do not look for clusters that involve only a subset of the attributes.

- K-means can find clusters that are not well separated, even if they overlap (see Figure 5.2(b)), but DBSCAN merges clusters that overlap.

- The K-means algorithm has a time complexity of $O(m)$, while DBSCAN takes $O(m^2)$ time, except for special cases such as low-dimensional Euclidean data.

- DBSCAN produces the same set of clusters from one run to another, while K-means, which is typically used with random initialization of centroids, does not.

- DBSCAN automatically determines the number of clusters; for K-means, the number of clusters needs to be specified as a parameter. However, DBSCAN has two other parameters that must be specified, *Eps* and $MinPts$.

- K-means clustering can be viewed as an optimization problem; i.e., minimize the sum of the squared error of each point to its closest centroid, and as a specific case of a statistical clustering approach (mixture models). DBSCAN is not based on any formal model.

8.1.2 Data Characteristics

The following are some characteristics of data that can strongly affect cluster analysis.

High Dimensionality In high-dimensional data sets, the traditional Euclidean notion of density, which is the number of points per unit volume, becomes meaningless. To see this, consider that as the number of dimensions increases, the volume increases rapidly, and unless the number of points grows exponentially with the number of dimensions, the density tends to 0. (Volume is exponential in the number of dimensions. For instance, a hypersphere with radius, r, and dimension, d, has volume proportional to r^d.) Also, proximity tends to become more uniform in high-dimensional spaces. Another way to view this fact is that there are more dimensions (attributes) that contribute to the proximity between two points and this tends to make the proximity more uniform. Since most clustering techniques are based on proximity or density, they can often have difficulty with high-dimensional data. One way to address such problems is to employ dimensionality reduction techniques. Another approach, as discussed in Sections 8.4.6 and 8.4.8, is to redefine the notions of proximity and density.

Size Many clustering algorithms that work well for small or medium-size data sets are unable to handle larger data sets. This is addressed further in the discussion of the characteristics of clustering algorithms—scalability is one such characteristic—and in Section 8.5, which discusses scalable clustering algorithms.

Sparseness Sparse data often consists of asymmetric attributes, where zero values are not as important as non-zero values. Therefore, similarity measures appropriate for asymmetric attributes are commonly used. However, other, related issues also arise. For example, are the magnitudes of non-zero entries important, or do they distort the clustering? In other words, does the clustering work best when there are only two values, 0 and 1?

Noise and Outliers An atypical point (outlier) can often severely degrade the performance of clustering algorithms, especially algorithms such as K-means that are prototype-based. On the other hand, noise can cause techniques, such as single link, to join clusters that should not be joined. In some cases, algorithms for removing noise and outliers are applied before a clustering algorithm is used. Alternatively, some algorithms can detect points that represent noise and outliers during the clustering process and then delete them or otherwise eliminate their negative effects. In Chapter 5, for instance, we saw that DBSCAN automatically classifies low-density points as noise and removes them from the clustering process. Chameleon (Section 8.4.4), SNN density-based clustering (Section 8.4.9), and CURE (Section 8.5.3) are three of the algorithms in this chapter that explicitly deal with noise and outliers during the clustering process.

Type of Attributes and Data Set As discussed in Chapter 2, data sets can be of various types, such as structured, graph, or ordered, while attributes are usually categorical (nominal or ordinal) or quantitative (interval or ratio), and are binary, discrete, or continuous. Different proximity and density measures are appropriate for different types of data. In some situations, data needs to be discretized or binarized so that a desired proximity measure or clustering algorithm can be used. Another complication occurs when attributes are of widely differing types, e.g., continuous and nominal. In such cases, proximity and density are more difficult to define and often more ad hoc. Finally, special data structures and algorithms are often needed to handle certain types of data efficiently.

Scale Different attributes, e.g., height and weight, are often measured on different scales. These differences can strongly affect the distance or similarity between two objects and, consequently, the results of a cluster analysis. Consider clustering a group of people based on their heights, which are measured in meters, and their weights, which are measured in kilograms. If we use Euclidean distance as our proximity measure, then height will have little impact and people will be clustered mostly based on the weight attribute. If, however, we standardize each attribute by subtracting off its mean and dividing by its standard deviation, then we will have eliminated effects due to the difference in scale. More generally, normalization techniques, such as those discussed in Section 2.3.7, are typically used to handle these issues.

Mathematical Properties of the Data Space Some clustering techniques calculate the mean of a collection of points or use other mathematical operations that only make sense in Euclidean space or in other specific data spaces. Other algorithms require that the definition of density be meaningful for the data.

8.1.3 Cluster Characteristics

The different types of clusters, such as prototype-, graph-, and density-based, were described earlier in Section 5.1.3. Here, we describe other important characteristics of clusters.

Data Distribution Some clustering techniques assume a particular type of distribution for the data. More specifically, they often assume that data can be modeled as arising from a mixture of distributions, where each cluster corresponds to a distribution. Clustering based on mixture models is discussed in Section 8.2.2.

Shape Some clusters are regularly shaped, e.g., rectangular or globular, but in general, clusters can be of arbitrary shape. Techniques such as DBSCAN and single link can handle clusters of arbitrary shape, but prototype-based schemes and some hierarchical techniques, such as complete link and group average, cannot. Chameleon (Section 8.4.4) and CURE (Section 8.5.3) are examples of techniques that were specifically designed to address this problem.

Differing Sizes Many clustering methods, such as K-means, don't work well when clusters have different sizes. (See Section 5.2.4.) This topic is discussed further in Section 8.6.

Differing Densities Clusters that have widely varying density can cause problems for methods such as DBSCAN and K-means. The SNN density-based clustering technique presented in Section 8.4.9 addresses this issue.

Poorly Separated Clusters When clusters touch or overlap, some clustering techniques combine clusters that should be kept separate. Even techniques that find distinct clusters arbitrarily assign points to one cluster or another. Fuzzy clustering, which is described in Section 8.2.1, is one technique for dealing with data that does not form well-separated clusters.

Relationships among Clusters In most clustering techniques, there is no explicit consideration of the relationships between clusters, such as their relative position. Self-organizing maps (SOM), which are described in Section 8.2.3, are a clustering technique that directly considers the relationships between clusters during the clustering process. Specifically, the assignment of a point to one cluster affects the definitions of nearby clusters.

Subspace Clusters Clusters may only exist in a subset of dimensions (attributes), and the clusters determined using one set of dimensions are frequently quite different from the clusters determined by using another set. While this issue can arise with as few as two dimensions, it becomes more acute as dimensionality increases, because the number of possible subsets of dimensions is exponential in the total number of dimensions. For that reason, it is not feasible to simply look for clusters in all possible subsets of dimensions unless the number of dimensions is relatively low.

One approach is to apply feature selection, which was discussed in Section 2.3.4. However, this approach assumes that there is only one subset of dimensions in which the clusters exist. In reality, clusters can exist in many distinct subspaces (sets of dimensions), some of which overlap. Section 8.3.2 considers techniques that address the general problem of subspace clustering, i.e., of finding both clusters and the dimensions they span.

8.1.4 General Characteristics of Clustering Algorithms

Clustering algorithms are quite varied. We provide a general discussion of important characteristics of clustering algorithms here, and make more specific comments during our discussion of particular techniques.

Order Dependence For some algorithms, the quality and number of clusters produced can vary, perhaps dramatically, depending on the order in which the data is processed. While it would seem desirable to avoid such algorithms, sometimes the order dependence is relatively minor or the algorithm has other desirable characteristics. SOM (Section 8.2.3) is an example of an algorithm that is order dependent.

Nondeterminism Clustering algorithms, such as K-means, are not order-dependent, but they produce different results for each run because they rely on an initialization step that requires a random choice. Because the quality of the clusters can vary from one run to another, multiple runs can be necessary.

Scalability It is not unusual for a data set to contain millions of objects, and the clustering algorithms used for such data sets should have linear or near-linear time and space complexity. Even algorithms that have a complexity of $O(m^2)$ are not practical for large data sets. Furthermore, clustering techniques for data sets cannot always assume that all the data will fit in main memory or that data elements can be randomly accessed. Such algorithms are infeasible for large data sets. Section 8.5 is devoted to the issue of scalability.

Parameter Selection Most clustering algorithms have one or more parameters that need to be set by the user. It can be difficult to choose the proper values; thus, the attitude is usually, “the fewer parameters, the better.”

Choosing parameter values becomes even more challenging if a small change in the parameters drastically changes the clustering results. Finally, unless a procedure (which might involve user input) is provided for determining parameter values, a user of the algorithm is reduced to using trial and error to find suitable parameter values.

Perhaps the most well-known parameter selection problem is that of "choosing the right number of clusters" for partitional clustering algorithms, such as K-means. One possible approach to that issue is given in Section 5.5.5, while references to others are provided in the Bibliographic Notes.

Transforming the Clustering Problem to Another Domain One approach taken by some clustering techniques is to map the clustering problem to a problem in a different domain. Graph-based clustering, for instance, maps the task of finding clusters to the task of partitioning a proximity graph into connected components.

Treating Clustering as an Optimization Problem Clustering is often viewed as an optimization problem: divide the points into clusters in a way that maximizes the goodness of the resulting set of clusters as measured by a user-specified objective function. For example, the K-means clustering algorithm (Section 5.2) tries to find the set of clusters that minimizes the sum of the squared distance of each point from its closest cluster centroid. In theory, such problems can be solved by enumerating all possible sets of clusters and selecting the one with the best value of the objective function, but this exhaustive approach is computationally infeasible. For this reason, many clustering techniques are based on heuristic approaches that produce good, but not optimal clusterings. Another approach is to use objective functions on a greedy or local basis. In particular, the hierarchical clustering techniques discussed in Section 5.3 proceed by making locally optimal (greedy) decisions at each step of the clustering process.

Road Map

We arrange our discussion of clustering algorithms in a manner similar to that of Chapter 5, grouping techniques primarily according to whether they are prototype-based, density-based, or graph-based. There is, however, a separate discussion for scalable clustering techniques. We conclude this chapter with a discussion of how to choose a clustering algorithm.

8.2 Prototype-Based Clustering

In prototype-based clustering, a cluster is a set of objects in which any object is closer to the prototype that defines the cluster than to the prototype of any other cluster. Section 5.2 described K-means, a simple prototype-based clustering algorithm that uses the centroid of the objects in a cluster as the prototype of the cluster. This section discusses clustering approaches that expand on the concept of prototype-based clustering in one or more ways, as discussed next:

- Objects are allowed to belong to more than one cluster. More specifically, an object belongs to every cluster with some weight. Such an approach addresses the fact that some objects are equally close to several cluster prototypes.

- A cluster is modeled as a statistical distribution, i.e., objects are generated by a random process from a statistical distribution that is characterized by a number of statistical parameters, such as the mean and variance. This viewpoint generalizes the notion of a prototype and enables the use of well-established statistical techniques.

- Clusters are constrained to have fixed relationships. Most commonly, these relationships are constraints that specify neighborhood relationships; i.e., the degree to which two clusters are neighbors of each other. Constraining the relationships among clusters can simplify the interpretation and visualization of the data.

We consider three specific clustering algorithms to illustrate these extensions of prototype-based clustering. Fuzzy c-means uses concepts from the field of fuzzy logic and fuzzy set theory to propose a clustering scheme, which is much like K-means, but which does not require a hard assignment of a point to only one cluster. Mixture model clustering takes the approach that a set of clusters can be modeled as a mixture of distributions, one for each cluster. The clustering scheme based on Self-Organizing Maps (SOM) performs clustering within a framework that requires clusters to have a prespecified relationship to one another, e.g., a two-dimensional grid structure.

8.2.1 Fuzzy Clustering

If data objects are distributed in well-separated groups, then a crisp classification of the objects into disjoint clusters seems like an ideal approach. However, in most cases, the objects in a data set cannot be partitioned into well-separated clusters, and there will be a certain arbitrariness in assigning an object to a particular cluster. Consider an object that lies near the boundary of two clusters, but is slightly closer to one of them. In many such cases, it might be more appropriate to assign a weight to each object and each cluster that indicates the degree to which the object belongs to the cluster. Mathematically, w_{ij} is the weight with which object $\mathbf{x}_i$ belongs to cluster C_j.

As shown in the next section, probabilistic approaches can also provide such weights. While probabilistic approaches are useful in many situations, there are times when it is difficult to determine an appropriate statistical model. In such cases, non-probabilistic clustering techniques are needed to provide similar capabilities. Fuzzy clustering techniques are based on fuzzy set theory and provide a natural technique for producing a clustering in which membership weights (the w_{ij}) have a natural (but not probabilistic) interpretation. This section describes the general approach of fuzzy clustering and provides a specific example in terms of fuzzy c-means (fuzzy K-means).

Fuzzy Sets

Lotfi Zadeh introduced **fuzzy set theory** and **fuzzy logic** in 1965 as a way of dealing with imprecision and uncertainty. Briefly, fuzzy set theory allows an object to 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 that 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 systems, 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, e.g., 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."

Fuzzy Clusters

Assume that we have a set of data points $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, where each point, $\mathbf{x}_i$, is an n-dimensional point, i.e., $\mathbf{x}_i = (x_{i1}, \ldots, x_{in})$. A collection of fuzzy clusters, C_1, C_2, ..., C_k is a subset of all possible fuzzy subsets of $\mathcal{X}$. (This simply means that the membership weights (degrees), w_{ij}, have been assigned values between 0 and 1 for each point, $\mathbf{x}_i$, and each cluster, C_j.) However, we also want to impose the following reasonable conditions on the clusters in order to ensure that the clusters form what is called a **fuzzy pseudo-partition**.

1. All the weights for a given point, $\mathbf{x}_i$, add up to 1.
   $\sum\limits_{j=1}^{k} w_{ij} = 1$
2. Each cluster, C_j, contains, with non-zero weight, at least one point, but does not contain, with a weight of one, all of the points.
   $0 < \sum\limits_{i=1}^{m} w_{ij} < m$

Fuzzy c-means

While there are many types of fuzzy clustering—indeed, many data analysis algorithms can be "fuzzified"—we only consider the fuzzy version of K-means, which is called fuzzy c-means. In the clustering literature, the version of K-means that does not use incremental updates of cluster centroids is sometimes referred to as **c-means**, and this was the term adapted by the fuzzy community for the fuzzy version of K-means. The fuzzy c-means algorithm, also sometimes known as FCM, is given by Algorithm 8.1.

Algorithm 8.1 Basic fuzzy c-means algorithm.

```
Select an initial fuzzy pseudo-partition, i.e., assign values to all the $w_{ij}$.
repeat
    Compute the centroid of each cluster using the fuzzy pseudo-partition.
    Recompute the fuzzy pseudo-partition, i.e., the $w_{ij}$.
until The centroids don't change.
(Alternative stopping conditions are "if the change in the error is below a specified
threshold" or "if the absolute change in any $w_{ij}$ is below a given threshold.")
```

After initialization, FCM repeatedly computes the centroids of each cluster and the fuzzy pseudo-partition until the partition does not change. FCM is similar in structure to the K-means algorithm, which after initialization, alternates between a step that updates the centroids and a step that assigns each object to the closest centroid. Specifically, computing a fuzzy pseudo-partition is equivalent to the assignment step. As with K-means, FCM can be interpreted as attempting to minimize the sum of the squared error (SSE), although FCM is based on a fuzzy version of SSE. Indeed, K-means can be regarded as a special case of FCM and the behavior of the two algorithms is quite similar. The details of FCM are described below.

Computing SSE The definition of the sum of the squared error (SSE) is modified as follows:

$$\text{SSE}(C_1, C_2, \ldots, C_\text{k}) = \sum_{j=1}^{k} \sum_{i=1}^{m} w_{ij}^{p} dist(\mathbf{x}_i, \mathbf{c}_j)^2 \tag{8.1}$$

where $\mathbf{c}_j$ is the centroid of the j^{th} cluster and p, which is the exponent that determines the influence of the weights, has a value between 1 and ∞. Note that this SSE is just a weighted version of the traditional K-means SSE given in Equation 5.1.

Initialization Random initialization is often used. In particular, weights are chosen randomly, subject to the constraint that the weights associated with any object must sum to 1. As with K-means, random initialization is simple, but often results in a clustering that represents a local minimum in terms of the SSE. Section 5.2.1, which contains a discussion on choosing initial centroids for K-means, has considerable relevance for FCM as well.

Computing Centroids The definition of the centroid given in Equation 8.2 can be derived by finding the centroid that minimizes the fuzzy SSE as given by Equation 8.1. (See the approach in Section 5.2.6.) For a cluster, C$_j$, the corresponding centroid, $\mathbf{c}_j$, is defined by the following equation:

$$c_j = \sum_{i=1}^{m} w_{ij}^{p} \mathbf{x}_i / \sum_{i=1}^{m} w_{ij}^{p} \tag{8.2}$$

The fuzzy centroid definition is similar to 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, where all w_{ij} are either 0 or 1, this definition reduces to the traditional definition of a centroid.

There are a few considerations when choosing the value of p. Choosing $p = 2$ simplifies the weight update formula—see Equation 8.4. However, if p

is chosen to be near 1, then fuzzy c-means behaves like traditional K-means. Going in the other direction, as p gets larger, all the cluster centroids approach the global centroid of all the data points. In other words, the partition becomes fuzzier as p increases.

Updating the Fuzzy Pseudo-partition Because the fuzzy pseudo-partition is defined by the weight, this step involves updating the weights w_{ij} associated with the i^{th} point and j^{th} cluster. The weight update formula given in Equation 8.3 can be derived by minimizing the SSE of Equation 8.1 subject to the constraint that the weights sum to 1.

$$w_{ij} = \left(1/dist(\mathbf{x}_i, \mathbf{c}_j)^2\right)^{\frac{1}{p-1}} \Bigg/ \sum_{q=1}^{k} \left(1/dist(\mathbf{x}_i, \mathbf{c}_q)^2\right)^{\frac{1}{p-1}} \tag{8.3}$$

This formula might appear a bit mysterious. However, note that if $p = 2$, then we obtain Equation 8.4, which is somewhat simpler. We provide an intuitive explanation of Equation 8.4, which, with a slight modification, also applies to Equation 8.3.

$$w_{ij} = 1/dist(\mathbf{x}_i, \mathbf{c}_j)^2 \Bigg/ \sum_{q=1}^{k} 1/dist(\mathbf{x}_i, \mathbf{c}_q)^2 \tag{8.4}$$

Intuitively, the weight w_{ij}, which indicates the degree of membership of point $\mathbf{x}_i$ in cluster C_j, should be relatively high if $\mathbf{x}_i$ is close to centroid $\mathbf{c}_j$ (if $dist(\mathbf{x}_i, \mathbf{c}_j)$ is low) and relatively low if $\mathbf{x}_i$ is far from centroid $\mathbf{c}_j$ (if $dist(\mathbf{x}_i, \mathbf{c}_j)$ is high). If $w_{ij} = 1/dist(\mathbf{x}_i, \mathbf{c}_j)^2$, which is the numerator of Equation 8.4, then this will indeed be the case. However, the membership weights for a point will not sum to one unless they are normalized; i.e., divided by the sum of all the weights as in Equation 8.4. To summarize, the membership weight of a point in a cluster is just the reciprocal of the square of the distance between the point and the cluster centroid divided by the sum of all the membership weights of the point.

Now consider the impact of the exponent $1/(p-1)$ in Equation 8.3. If $p > 2$, then this exponent decreases the weight assigned to clusters that are close to the point. Indeed, as p goes to infinity, the exponent tends to 0 and weights tend to the value $1/k$. On the other hand, as p approaches 1, the exponent increases the membership weights of points to which the cluster is close. As p goes to 1, the membership weight goes to 1 for the closest cluster and to 0 for all the other clusters. This corresponds to K-means.

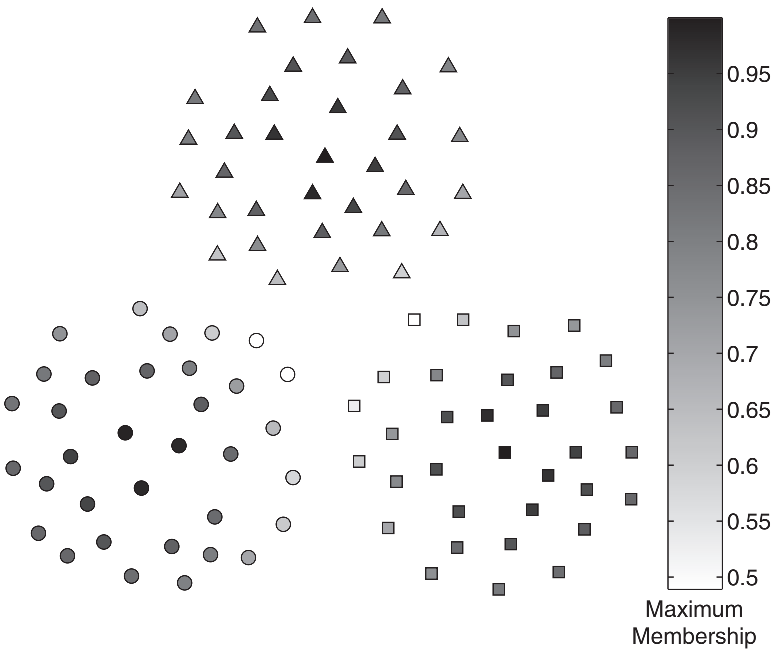


Figure 8.1. Fuzzy c-means clustering of a two-dimensional point set.

Example 8.1 (Fuzzy c-means on Three Circular Clusters)**.** Figure 8.1 shows the result of applying fuzzy c-means to find three clusters for a two-dimensional data set of 100 points. Each point was assigned to the cluster in which it had the largest membership weight. The points belonging to each cluster are shown by different marker shapes, while the degree of membership in the cluster is shown by the shading. The darker the points, the stronger their membership in the cluster to which they have been assigned. The membership in a cluster is strongest toward the center of the cluster and weakest for those points that are between clusters. ■

Strengths and Limitations

A positive feature of FCM is that it produces a clustering that provides an indication of the degree to which any point belongs to any cluster. Otherwise, it has much the same strengths and weaknesses as K-means, although it is somewhat more computationally intensive.

8.2.2 Clustering Using Mixture Models

This section considers clustering based on statistical models. It is often convenient and effective to assume that data has been generated as a result of a statistical process and to describe the data by finding the statistical model that best fits the data, where the statistical model is described in terms of a distribution and a set of parameters for that distribution. At a high level, this process involves deciding on a statistical model for the data and estimating the parameters of that model from the data. This section describes a particular kind of statistical model, **mixture models**, which model the data by using a number of statistical distributions. Each distribution corresponds to a cluster and the parameters of each distribution provide a description of the corresponding cluster, typically in terms of its center and spread.

The discussion in this section proceeds as follows. After providing a description of mixture models, we consider how parameters can be estimated for statistical data models. We first describe how a procedure known as **maximum likelihood estimation (MLE)** can be used to estimate parameters for simple statistical models and then discuss how we can extend this approach for estimating the parameters of mixture models. Specifically, we describe the well-known **Expectation-Maximization (EM) algorithm**, which makes an initial guess for the parameters, and then iteratively improves these estimates. We present examples of how the EM algorithm can be used to cluster data by estimating the parameters of a mixture model and discuss its strengths and limitations.

A firm understanding of statistics and probability, as covered in Appendix C, is essential for understanding this section. Also, for convenience in the following discussion, we use the term probability to refer to both probability and probability density.

Mixture Models

Mixture models view the data as a set of observations from a mixture of different probability distributions. The probability distributions can be anything, but are often taken to be multivariate normal, as this type of distribution is well understood, mathematically easy to work with, and has been shown to produce good results in many instances. These types of distributions can model ellipsoidal clusters.

Conceptually, mixture models correspond to the following process of generating data. Given several distributions, usually of the same type, but with different parameters, randomly select one of these distributions and generate an object from it. Repeat the process m times, where m is the number of objects.

More formally, assume that there are K distributions and m objects, $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$. Let the j^{th} distribution have parameters θ_j, and let Θ be the set of all parameters, i.e., $\Theta = \{\theta_1, \ldots, \theta_K\}$. Then, $prob(\mathbf{x}_i|\theta_j)$ is the probability of the i^{th} object if it comes from the j^{th} distribution. The probability that the j^{th} distribution is chosen to generate an object is given by the weight w_j, $1 \leq j \leq K$, where these weights (probabilities) are subject to the constraint that they sum to one, i.e., $\sum_{j=1}^{K} w_j = 1$. Then, the probability of an object $\mathbf{x}$ is given by Equation 8.5.

$$prob(\mathbf{x}|\Theta) = \sum_{j=1}^{K} w_j p_j(\mathbf{x}|\theta_j) \tag{8.5}$$

If the objects are generated in an independent manner, then the probability of the entire set of objects is just the product of the probabilities of each individual $\mathbf{x}_i$.

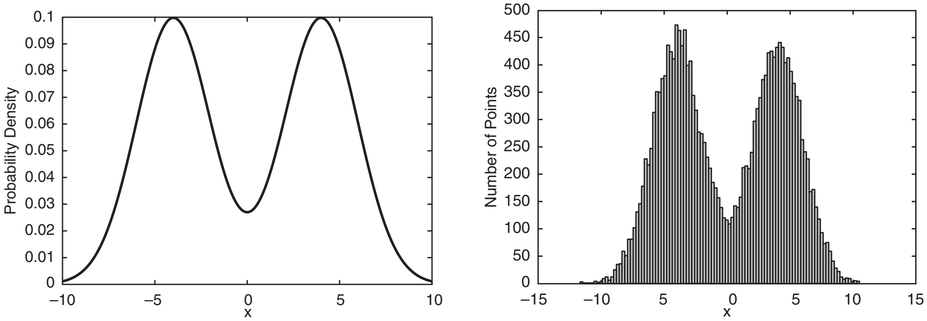
$$prob(\mathcal{X}|\Theta) = \prod_{i=1}^{m} prob(\mathbf{x}_i|\Theta) = \prod_{i=1}^{m} \sum_{j=1}^{K} w_j p_j(\mathbf{x}_i|\theta_j) \tag{8.6}$$

For mixture models, each distribution describes a different group, i.e., a different cluster. By using statistical methods, we can estimate the parameters of these distributions from the data and thus describe these distributions (clusters). We can also identify which objects belong to which clusters. However, mixture modeling does not produce a crisp assignment of objects to clusters, but rather gives the probability with which a specific object belongs to a particular cluster.

Example 8.2 (Univariate Gaussian Mixture)**.** We provide a concrete illustration of a mixture model in terms of Gaussian distributions. The probability density function for a one-dimensional Gaussian distribution at a point x is

$$prob(x|\Theta) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{8.7}$$

The parameters of the Gaussian distribution are given by $\theta = (\mu, \sigma)$, where μ is the mean of the distribution and σ is the standard deviation. Assume that there are two Gaussian distributions, with a common standard deviation



(a) Probability density function for the mixture model.

(b) 20,000 points generated from the mixture model.

Figure 8.2. Mixture model consisting of two normal distributions with means of -4 and 4, respectively. Both distributions have a standard deviation of 2.

of 2 and means of -4 and 4, respectively. Also assume that each of the two distributions is selected with equal probability, i.e., $w_1 = w_2 = 0.5$. Then Equation 8.5 becomes the following:

$$prob(x|\Theta) = \frac{1}{2\sqrt{2\pi}}\, e^{-\frac{(x+4)^2}{8}} + \frac{1}{2\sqrt{2\pi}}\, e^{-\frac{(x-4)^2}{8}}. \tag{8.8}$$

Figure 8.2(a) shows a plot of the probability density function of this mixture model, while Figure 8.2(b) shows the histogram for 20,000 points generated from this mixture model. ■

Estimating Model Parameters Using Maximum Likelihood

Given a statistical model for the data, it is necessary to estimate the parameters of that model. A standard approach used for this task is maximum likelihood estimation, which we now explain.

Consider a set of m points that are generated from a one-dimensional Gaussian distribution. Assuming that the points are generated independently, the probability of these points is just the product of their individual probabilities. (Again, we are dealing with probability densities, but to keep our terminology simple, we will refer to probabilities.) Using Equation 8.7, we can write this probability as shown in Equation 8.9. Because this probability would be a very small number, we typically will work with the log probability, as shown in Equation 8.10.

$$prob(\mathcal{X}|\Theta) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma}\, e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \tag{8.9}$$

$$log\ prob(\mathcal{X}|\Theta) = -\sum_{i=1}^{m} \frac{(x_i-\mu)^2}{2\sigma^2} - 0.5m\log 2\pi - m\log\sigma \tag{8.10}$$

We would like to find a procedure to estimate μ and σ if they are unknown. One approach is to choose the values of the parameters for which the data is most probable (most likely). In other words, choose the μ and σ that maximize Equation 8.9. This approach is known in statistics as the **maximum likelihood principle**, and the process of applying this principle to estimate the parameters of a statistical distribution from the data is known as **maximum likelihood estimation (MLE)**.

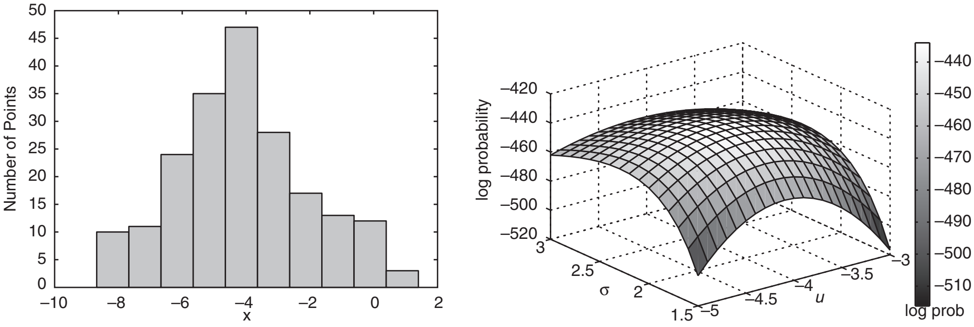
The principle is called the maximum likelihood principle because, given a set of data, the probability of the data, regarded as a function of the parameters, is called a **likelihood function**. To illustrate, we rewrite Equation 8.9 as Equation 8.11 to emphasize that we view the statistical parameters μ and σ as our variables and that the data is regarded as a constant. For practical reasons, the log likelihood is more commonly used. The log likelihood function derived from the log probability of Equation 8.10 is shown in Equation 8.12. Note that the parameter values that maximize the log likelihood also maximize the likelihood since log is a monotonically increasing function.

$$likelihood(\Theta|\mathcal{X}) = L(\Theta|\mathcal{X}) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi}\sigma}\, e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \tag{8.11}$$

$$log\ likelihood(\Theta|\mathcal{X}) = \ell(\Theta|\mathcal{X}) = -\sum_{i=1}^{m} \frac{(x_i-\mu)^2}{2\sigma^2} - 0.5m\log 2\pi - m\log\sigma \tag{8.12}$$

Example 8.3 (Maximum Likelihood Parameter Estimation)**.** We provide a concrete illustration of the use of MLE for finding parameter values. Suppose that we have the set of 200 points whose histogram is shown in Figure 8.3(a). Figure 8.3(b) shows the maximum log likelihood plot for the 200 points under consideration. The values of the parameters for which the log probability is a maximum are $\mu = -4.1$ and $\sigma = 2.1$, which are close to the parameter values of the underlying Gaussian distribution, $\mu = -4.0$ and $\sigma = 2.0$. ■

Graphing the likelihood of the data for different values of the parameters is not practical, at least if there are more than two parameters. Thus, standard



(a) Histogram of 200 points from a Gaussian distribution.

(b) Log likelihood plot of the 200 points for different values of the mean and standard deviation.

Figure 8.3. 200 points from a Gaussian distribution and their log probability for different parameter values.

statistical procedure is to derive the maximum likelihood estimates of a statistical parameter by taking the derivative of likelihood function with respect to that parameter, setting the result equal to 0, and solving. In particular, for a Gaussian distribution, it can be shown that the mean and standard deviation of the sample points are the maximum likelihood estimates of the corresponding parameters of the underlying distribution. (See Exercise 13 on 720.) Indeed, for the 200 points considered in our example, the parameter values that maximized the log likelihood were precisely the mean and standard deviation of the 200 points, i.e., $\mu = -4.1$ and $\sigma = 2.1$.

Estimating Mixture Model Parameters Using Maximum Likelihood: The EM Algorithm

We can also use the maximum likelihood approach to estimate the model parameters for a mixture model. In the simplest case, we know which data objects come from which distributions, and the situation reduces to one of estimating the parameters of a single distribution given data from that distribution. For most common distributions, the maximum likelihood estimates of the parameters are calculated from simple formulas involving the data.

In a more general (and more realistic) situation, we do not know which points were generated by which distribution. Thus, we cannot directly calculate the probability of each data point, and hence, it would seem that we cannot use the maximum likelihood principle to estimate parameters. The solution to this problem is the EM algorithm, which is shown in Algorithm 8.2. Briefly, given a guess for the parameter values, the EM algorithm calculates the probability that each point belongs to each distribution and then uses these probabilities to compute a new estimate for the parameters. (These parameters are the ones that maximize the likelihood.) This iteration continues until the estimates of the parameters either do not change or change very little. Thus, we still employ maximum likelihood estimation, but via an iterative search.

Algorithm 8.2 EM algorithm.

```
Select an initial set of model parameters.
(As with K-means, this can be done randomly or in a variety of ways.)
repeat
    Expectation Step For each object, calculate the probability that each object belongs to each distribution, i.e., calculate $prob(distribution\ j|\mathbf{x}_i, \Theta)$.
    Maximization Step Given the probabilities from the expectation step, find the new estimates of the parameters that maximize the expected likelihood.
until The parameters do not change.
(Alternatively, stop if the change in the parameters is below a specified threshold.)
```

The EM algorithm is similar to the K-means algorithm given in Section 5.2.1. Indeed, the K-means algorithm for Euclidean data is a special case of the EM algorithm for spherical Gaussian distributions with equal covariance matrices, but different means. The expectation step corresponds to the K-means step of assigning each object to a cluster. Instead, each object is assigned to every cluster (distribution) with some probability. The maximization step corresponds to computing the cluster centroids. Instead, all the parameters of the distributions, as well as the weight parameters, are selected to maximize the likelihood. This process is often straightforward, as the parameters are typically computed using formulas derived from maximum likelihood estimation. For instance, for a single Gaussian distribution, the MLE estimate of the mean is the mean of the objects in the distribution. In the context of mixture models and the EM algorithm, the computation of the mean is modified to account for the fact that every object belongs to a distribution with a certain probability. This is illustrated further in the following example.

Example 8.4 (Simple Example of EM Algorithm)**.** This example illustrates how EM operates when applied to the data in Figure 8.2. To keep the example as simple as possible, we assume that we know that the standard deviation of both distributions is 2.0 and that points were generated with equal probability from both distributions. We will refer to the left and right distributions as distributions 1 and 2, respectively.

We begin the EM algorithm by making initial guesses for μ_1 and μ_2, say, $\mu_1 = -2$ and $\mu_2 = 3$. Thus, the initial parameters, $\theta = (\mu, \sigma)$, for the two distributions are, respectively, $\theta_1 = (-2, 2)$ and $\theta_2 = (3, 2)$. The set of parameters for the entire mixture model is $\Theta = \{\theta_1, \theta_2\}$. For the expectation step of EM, we want to compute the probability that a point came from a particular distribution; i.e., we want to compute $prob(distribution\ 1|x_i, \Theta)$ and $prob(distribution\ 2|x_i, \Theta)$. These values can be expressed by Equation 8.13, which is a straightforward application of Bayes rule, which is described in Appendix C.

$$prob(distribution\ j|x_i, \theta) = \frac{0.5\ prob(x_i|\theta_j)}{0.5\ prob(x_i|\theta_1) + 0.5\ prob(x_i|\theta_2)}, \tag{8.13}$$

where 0.5 is the probability (weight) of each distribution and j is 1 or 2.

For instance, assume one of the points is 0. Using the Gaussian density function given in Equation 8.7, we compute that $prob(0|\theta_1) = 0.12$ and $prob(0|\theta_2) = 0.06$. (Again, we are really computing probability densities.) Using these values and Equation 8.13, we find that $prob(distribution\ 1|0, \Theta) = 0.12/(0.12+0.06) = 0.66$ and $prob(distribution\ 2|0, \Theta) = 0.06/(0.12+0.06) = 0.33$. This means that the point 0 is twice as likely to belong to distribution 1 as distribution 2 based on the current assumptions for the parameter values.

After computing the cluster membership probabilities for all 20,000 points, we compute new estimates for μ_1 and μ_2 (using Equations 8.14 and 8.15) in the maximization step of the EM algorithm. Notice that the new estimate for the mean of a distribution is just a weighted average of the points, where the weights are the probabilities that the points belong to the distribution, i.e., the $prob(distribution\ j|x_i)$ values.

$$\mu_1 = \sum_{i=1}^{20{,}000} x_i \frac{prob(distribution\ 1|x_i, \Theta)}{\sum_{i=1}^{20{,}000} prob(distribution\ 1|x_i, \Theta)} \tag{8.14}$$

Table 8.1. First few iterations of the EM algorithm for the simple example.

Iteration	μ_1	μ_2
0	−2.00	3.00
1	−3.74	4.10
2	−3.94	4.07
3	−3.97	4.04
4	−3.98	4.03
5	−3.98	4.03

$$\mu_2 = \sum_{i=1}^{20{,}000} x_i \frac{prob(distribution\ 2|x_i, \Theta)}{\sum_{i=1}^{20{,}000} prob(distribution\ 2|x_i, \Theta)} \tag{8.15}$$

We repeat these two steps until the estimates of μ_1 and μ_2 either don't change or change very little. Table 8.1 gives the first few iterations of the EM algorithm when it is applied to the set of 20,000 points. For this data, we know which distribution generated which point, so we can also compute the mean of the points from each distribution. The means are $\mu_1 = -3.98$ and $\mu_2 = 4.03$. ■

Example 8.5 (The EM Algorithm on Sample Data Sets)**.** We give three examples that illustrate the use of the EM algorithm to find clusters using mixture models. The first example is based on the data set used to illustrate the fuzzy c-means algorithm—see Figure 8.1. We modeled this data as a mixture of three two-dimensional Gaussian distributions with different means and identical covariance matrices. We then clustered the data using the EM algorithm. The results are shown in Figure 8.4. Each point was assigned to the cluster in which it had the largest membership weight. The points belonging to each cluster are shown by different marker shapes, while the degree of membership in the cluster is shown by the shading. Membership in a cluster is relatively weak for those points that are on the border of the two clusters, but strong elsewhere. It is interesting to compare the membership weights and probabilities of Figures 8.4 and 8.1. (See Exercise 16 on page 720.)

For our second example, we apply mixture model clustering to data that contains clusters with different densities. The data consists of two natural clusters, each with roughly 500 points. This data was created by combining two sets of Gaussian data, one with a center at (−4,1) and a standard deviation of 2, and one with a center at (0,0) and a standard deviation of 0.5. Figure 8.5 shows the clustering produced by the EM algorithm. Despite the differences in the density, the EM algorithm is quite successful at identifying the original clusters.

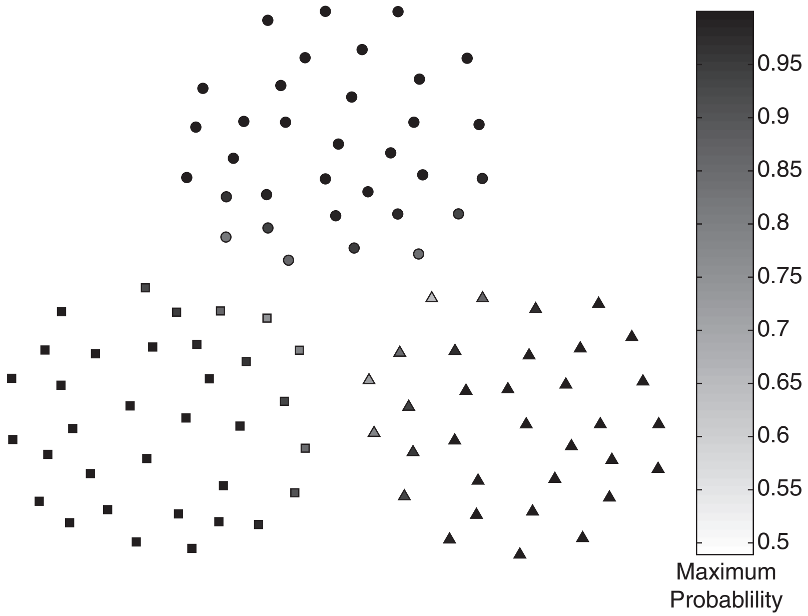


Figure 8.4. EM clustering of a two-dimensional point set with three clusters.

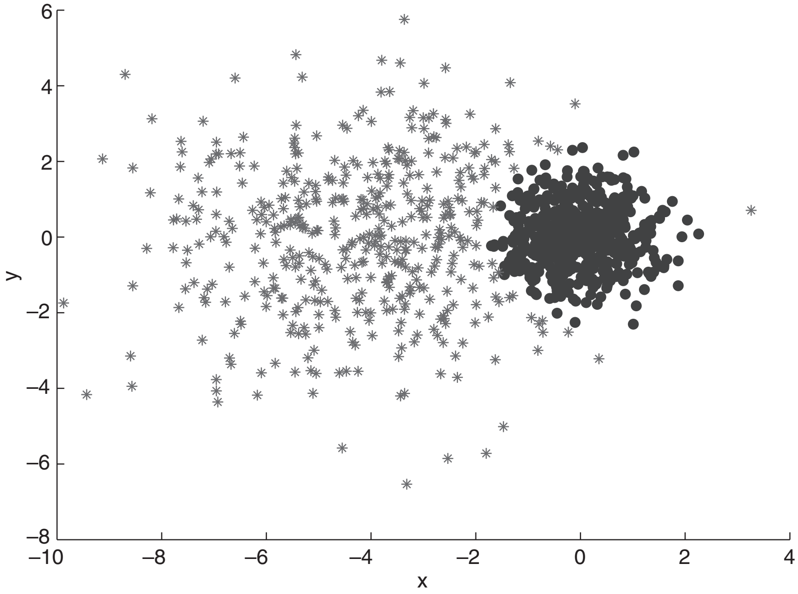
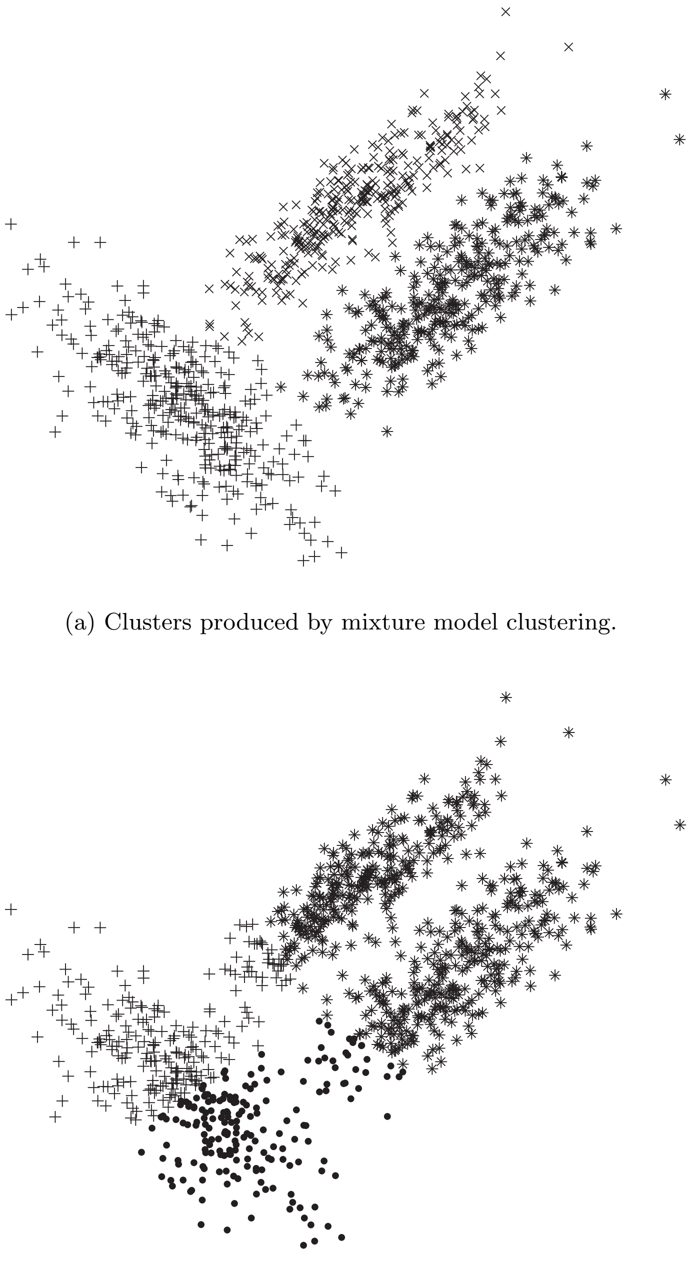


Figure 8.5. EM clustering of a two-dimensional point set with two clusters of differing density.



(a) Clusters produced by mixture model clustering.

(b) Clusters produced by K-means clustering.

Figure 8.6. Mixture model and K-means clustering of a set of two-dimensional points.

For our third example, we use mixture model clustering on a data set that K-means cannot properly handle. Figure 8.6(a) shows the clustering produced by a mixture model algorithm, while Figure 8.6(b) shows the K-means clustering of the same set of 1,000 points. For mixture model clustering, each point has been assigned to the cluster for which it has the highest probability. In both figures, different markers are used to distinguish different clusters. Do not confuse the '+' and 'x' markers in Figure 8.6(a). ■

Advantages and Limitations of Mixture Model Clustering Using the EM Algorithm

Finding clusters by modeling the data using mixture models and applying the EM algorithm to estimate the parameters of those models has a variety of advantages and disadvantages. On the negative side, the EM algorithm can be slow, it is not practical for models with large numbers of components, and it 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 typically has 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 can also have difficulty with noise and outliers, although work has been done to deal with this problem.

On the positive side, mixture models are more general than K-means or fuzzy c-means because they can use distributions of various types. As a result, mixture models (based on Gaussian distributions) can find clusters of different sizes and elliptical shapes. 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 is often necessary to simplify the data, and fitting the data to a model is a good way to do that if the model is a good match for the data. Furthermore, it is easy to characterize the clusters produced, because 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.

8.2.3 Self-Organizing Maps (SOM)

The Kohonen Self-Organizing Feature Map (SOFM or SOM) is a clustering and data visualization technique based on a neural network viewpoint. Despite the neural network origins of SOM, it is more easily presented—at least in the context of this chapter—as a variation of prototype-based clustering. As with

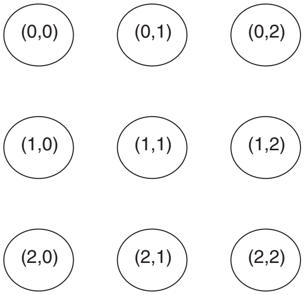


Figure 8.7. Two-dimensional 3-by-3 rectangular SOM neural network.

other types of centroid-based clustering, the goal of SOM is to find a set of centroids (**reference vectors** in SOM terminology) and to assign each object in the data set to the centroid that provides the best approximation of that object. In neural network terminology, there is one neuron associated with each centroid.

As with incremental K-means, data objects are processed one at a time and the closest centroid is updated. Unlike K-means, SOM imposes a topographic ordering on the centroids and nearby centroids are also updated. Furthermore, SOM does not keep track of the current cluster membership of an object, and, unlike K-means, if an object switches clusters, there is no explicit update of the old cluster centroid. However, if the old cluster is in the neighborhood of the new cluster, it will be updated. The processing of points continues until some predetermined limit is reached or the centroids are not changing very much. The final output of the SOM technique is a set of centroids that implicitly define clusters. Each cluster consists of the points closest to a particular centroid. The following section explores the details of this process.

The SOM Algorithm

A distinguishing feature of SOM is that it imposes a topographic (spatial) organization on the centroids (neurons). Figure 8.7 shows an example of a two-dimensional SOM in which the centroids are represented by nodes that are organized in a rectangular lattice. Each centroid is assigned a pair of coordinates (i, j). Sometimes, such a network is drawn with links between adjacent nodes, but that can be misleading because the influence of one centroid on another is via a neighborhood that is defined in terms of coordinates, not links. There are many types of SOM neural networks, but we restrict our discussion to two-dimensional SOMs with a rectangular or hexagonal organization of the centroids.

Even though SOM is similar to K-means or other prototype-based approaches, there is a fundamental difference. Centroids used in SOM have a predetermined topographic ordering relationship. During the training process, SOM uses each data point to update the closest centroid and centroids that are nearby in the topographic ordering. In this way, SOM produces an ordered set of centroids for any given data set. In other words, the centroids that are close to each other in the SOM grid are more closely related to each other than to the centroids that are farther away. Because of this constraint, the centroids of a two-dimensional SOM can be viewed as lying on a two-dimensional surface that tries to fit the n-dimensional data as well as possible. The SOM centroids can also be thought of as the result of a nonlinear regression with respect to the data points.

At a high level, clustering using the SOM technique consists of the steps described in Algorithm 8.3.

Algorithm 8.3 Basic SOM Algorithm.

```
1: Initialize the centroids.
2: repeat
3:    Select the next object.
4:    Determine the closest centroid to the object.
5:    Update this centroid and the centroids that are close, i.e., in a specified
      neighborhood.
6: until The centroids don't change much or a threshold is exceeded.
7: Assign each object to its closest centroid and return the centroids and clusters.
```

Initialization This step (line 1) can be performed in a number of ways. One approach is to choose each component of a centroid 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 rapid convergence. Another approach is to randomly choose the initial centroids from the available data points. This is very much like randomly selecting centroids for K-means.

Selection of an Object The first step in the loop (line 3) is the selection of the next object. This is fairly straightforward, but there are some difficulties. Because convergence can require many steps, each data object may be used multiple times, especially if the number of objects is small. However, if the number of objects is large, then not every object needs to be used. It is also possible to enhance the influence of certain groups of objects by increasing their frequency in the training set.

Assignment The determination of the closest centroid (line 4) is also relatively straightforward, although it requires the specification of a distance metric. The Euclidean distance metric is often used, as is the dot product metric. When using the dot product distance, the data vectors are typically normalized beforehand and the reference vectors are normalized at each step. In such cases, using the dot product metric is equivalent to using the cosine measure.

Update The update step (line 5) is the most complicated. Let $\mathbf{m}_1, \ldots, \mathbf{m}_k$ be the centroids. (For a rectangular grid, note that k is the product of the number of rows and the number of columns.) For time step t, let $\mathbf{p}(t)$ be the current object (point) and assume that the closest centroid to $\mathbf{p}(t)$ is $\mathbf{m}_j$. Then, for time $t+1$, the j^{th} centroid is updated by using the following equation. (We will see shortly that the update is really restricted to centroids whose neurons are in a small neighborhood of $\mathbf{m}_j$.)

Each object has coordinates

$$\mathbf{m}_j(t+1) = \mathbf{m}_j(t) + h_j(t)(\mathbf{p}(t) - \mathbf{m}_j(t)) \tag{8.16}$$

Thus, at time t, a centroid $\mathbf{m}_j(t)$ is updated by adding a term, $h_j(t)$ $(\mathbf{p}(t) - \mathbf{m}_j(t))$, which is proportional to the difference, $\mathbf{p}(t) - \mathbf{m}_j(t)$, between the current object, $\mathbf{p}(t)$, and centroid, $\mathbf{m}_j(t)$. $h_j(t)$, determines the effect that the difference, $\mathbf{p}(t) - \mathbf{m}_j(t)$, will have and is chosen so that (1) it diminishes with time and (2) it enforces a neighborhood effect, i.e., the effect of an object is strongest on the centroids closest to the centroid $\mathbf{m}_j$. Here we are referring to the distance in the grid, not the distance in the data space. Typically, $h_j(t)$ is chosen to be one of the following two functions:

$$\begin{aligned} h_j(t) &= \alpha(t) exp(-dist(\mathbf{r}_j, \mathbf{r}_k)^2/(2\sigma^2(t))) && \text{(Gaussian function)} \\ h_j(t) &= \alpha(t) \text{ if } dist(\mathbf{r}_j, \mathbf{r}_k) \leq threshold, \text{ 0 otherwise} && \text{(step function)} \end{aligned}$$

These functions require more explanation. $\alpha(t)$ is a learning rate parameter, $0 < \alpha(t) < 1$, which decreases monotonically with time and controls the rate of convergence. $\mathbf{r}_k = (x_k, y_k)$ is the two-dimensional point that gives the grid coordinates of the k^{th} centroid. $dist(\mathbf{r}_j, \mathbf{r}_k)$ is the Euclidean distance between the grid location of the two centroids, i.e., $\sqrt{(x_j - x_k)^2 + (y_j - y_k)^2}$.

Consequently, for centroids whose grid locations are far from the grid location of centroid $\mathbf{m}_j$, the influence of object $\mathbf{p}(t)$ 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.

Remember, it is the neighborhood updating technique that enforces a relationship (ordering) between centroids associated with neighboring neurons.

Termination Deciding when we are close enough to a stable set of centroids is an important issue. Ideally, iteration should continue until convergence occurs, that is, until the reference vectors either do not change or change very little. The rate of convergence will depend on a number of factors, such as the data and $\alpha(t)$. We will not discuss these issues further, except to mention that, in general, convergence can be slow and is not guaranteed.

Example 8.6 (Document Data)**.** We present two examples. In the first case, we apply SOM with a 4-by-4 hexagonal grid to document data. We clustered 3204 newspaper articles from the *Los Angeles Times*, which come from 6 different sections: Entertainment, Financial, Foreign, Metro, National, and Sports. Figure 8.8 shows the SOM grid. We have used a hexagonal grid, which allows each centroid to have six immediate neighbors instead of four. Each SOM grid cell (cluster) has been labeled with the majority class label of the associated points. The clusters of each particular category form contiguous groups, and their position relative to other categories of clusters gives us additional information, e.g., that the Metro section contains stories related to all other sections. ■

Example 8.7 (Two-Dimensional Points)**.** In the second case, we use a rectangular SOM and a set of two-dimensional data points. Figure 8.9(a) shows the points and the positions of the 36 reference vectors (shown as x's) produced by SOM. The points are arranged in a checkerboard pattern and are split into five classes: circles, triangles, squares, diamonds, and hexagons (stars). A 6-by-6 two-dimensional rectangular grid of centroids was used with random initialization. As Figure 8.9(a) shows, the centroids tend to distribute themselves to the dense areas. Figure 8.9(b) indicates the majority class of the points associated with that centroid. The clusters associated with triangle points are in one contiguous area, as are the centroids associated with the four other types of points. This is a result of the neighborhood constraints enforced by SOM. While there are the same number of points in each of the five groups, notice also that the centroids are not evenly distributed. This is partly due

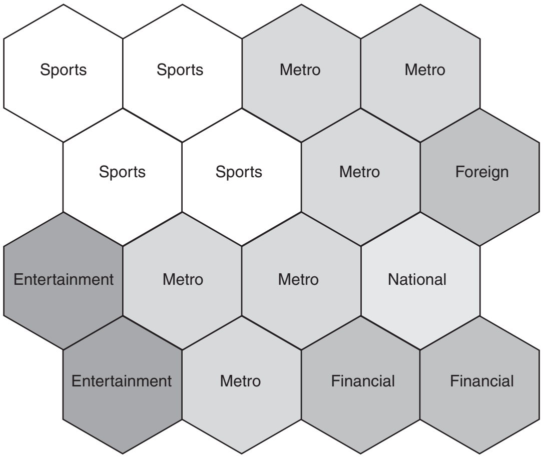
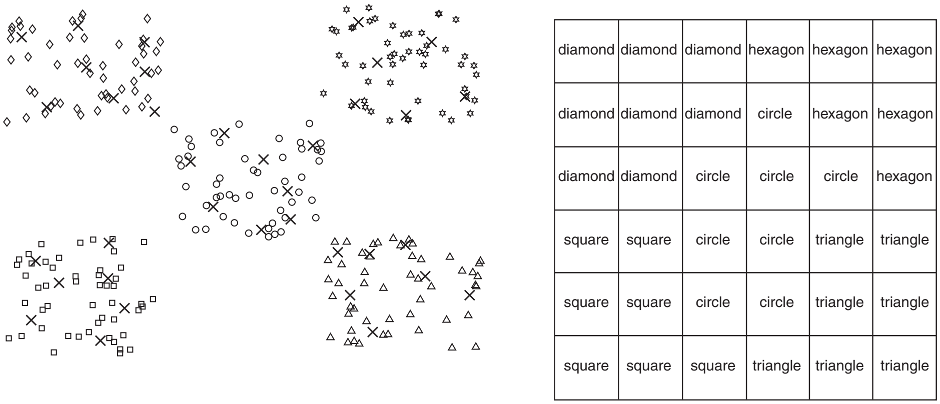


Figure 8.8. Visualization of the relationships between SOM cluster for *Los Angeles Times* document data set.



diamond	diamond	diamond	hexagon	hexagon	hexagon
diamond	diamond	diamond	circle	hexagon	hexagon
diamond	diamond	circle	circle	circle	hexagon
square	square	circle	circle	triangle	triangle
square	square	circle	circle	triangle	triangle
square	square	square	triangle	triangle	triangle

(a) Distribution of SOM reference vectors (**X**'s) for a two-dimensional point set.

(b) Classes of the SOM centroids.

Figure 8.9. SOM applied to two-dimensional data points.

to the overall distribution of points and partly an artifact of putting each centroid in a single cluster. ■

Applications

Once the SOM vectors are found, they can be used for many purposes other than clustering. For example, with a two-dimensional SOM, it is possible to associate various quantities with the grid points associated with each centroid (cluster) and to visualize the results via various types of plots. For example, plotting the number of points associated with each cluster yields a plot that reveals the distribution of points among clusters. A two-dimensional SOM is a nonlinear projection of the original probability distribution function into two dimensions. This projection attempts to preserve topological features; thus, using SOM to capture the structure of the data has been compared to the process of "pressing a flower."

Strengths and Limitations

SOM is a clustering technique that enforces neighborhood relationships on the resulting cluster centroids. Because of this, clusters that are neighbors are more related to one another than clusters that are not. Such relationships facilitate the interpretation and visualization of the clustering results. Indeed, this aspect of SOM has been exploited in many areas, such as visualizing web documents or gene array data.

SOM also has a number of limitations, which are listed next. Some of the listed limitations are only valid if we consider SOM to be a standard clustering technique that aims to find the true clusters in the data, rather than a technique that uses clustering to help discover the structure of the data. Also, some of these limitations have been addressed either by extensions of SOM or by clustering algorithms inspired by SOM. (See the Bibliographic Notes.)

- The user must choose the settings of parameters, the neighborhood function, the grid type, and the number of centroids.

- A SOM cluster often does not correspond to a single natural cluster. In some cases, a SOM cluster might encompass several natural clusters, while in other cases a single natural cluster is split into several SOM clusters. This problem is partly due to the use of a grid of centroids and partly due to the fact that SOM, like other prototype-based clustering techniques, tends to split or combine natural clusters when they are of varying sizes, shapes, and densities.

- SOM lacks a specific objective function. SOM attempts to find a set of centroids that best approximate the data, subject to the topographic constraints among the centroids, but the success of SOM in doing this cannot be expressed by a function. This can make it difficult to compare different SOM clustering results.

- SOM is not guaranteed to converge, although, in practice, it typically does.

8.3 Density-Based Clustering

In Section 5.4, we considered DBSCAN, a simple, but effective algorithm for finding density-based clusters, i.e., dense regions of objects that are surrounded by low-density regions. This section examines additional density-based clustering techniques that address issues of efficiency, finding clusters in subspaces, and more accurately modeling density. First, we consider grid-based clustering, which breaks the data space into grid cells and then forms clusters from cells that are sufficiently dense. Such an approach can be efficient and effective, at least for low-dimensional data. Next, we consider subspace clustering, which looks for clusters (dense regions) in subsets of all dimensions. For a data space with n dimensions, potentially $2^n - 1$ subspaces need to be searched, and thus an efficient technique is needed to do this. CLIQUE is a grid-based clustering algorithm that provides an efficient approach to subspace clustering based on the observation that dense areas in a high-dimensional space imply the existence of dense areas in lower-dimensional space. Finally, we describe DENCLUE, a clustering technique that uses kernel density functions to model density as the sum of the influences of individual data objects. While DENCLUE is not fundamentally a grid-based technique, it does employ a grid-based approach to improve efficiency.

8.3.1 Grid-Based Clustering

A grid is an efficient way to organize a set of data, at least in low dimensions. The idea is to split the possible values of each attribute into a number of contiguous intervals, creating a set of grid cells. (We are assuming, for this discussion and the remainder of the section, that our attributes are ordinal, interval, or continuous.) Each object falls into a grid cell whose corresponding attribute intervals contain the values of the object. Objects can be assigned to grid cells in one pass through the data, and information about each cell, such as the number of points in the cell, can also be gathered at the same time.

There are a number of ways to perform clustering using a grid, but most approaches are based on density, at least in part, and thus, in this section, we will use grid-based clustering to mean density-based clustering using a grid. Algorithm 8.4 describes a basic approach to grid-based clustering. Various aspects of this approach are explored next.

Algorithm 8.4 Basic grid-based clustering algorithm.

```
1: Define a set of grid cells.
2: Assign objects to the appropriate cells and compute the density of each cell.
3: Eliminate cells having a density below a specified threshold, $\tau$.
4: Form clusters from contiguous (adjacent) groups of dense cells.
```

Defining Grid Cells

This is a key step in the process, but also the least well defined, as there are many ways to split the possible values of each attribute into a number of contiguous intervals. For continuous attributes, one common approach is to split the values into equal width intervals. If this approach is applied to each attribute, then the resulting grid cells all have the same volume, and the density of a cell is conveniently defined as the number of points in the cell.

However, more sophisticated approaches can also be used. In particular, for continuous attributes any of the techniques that are commonly used to discretize attributes can be applied. (See Section 2.3.6.) In addition to the equal width approach already mentioned, this includes (1) breaking the values of an attribute into intervals so that each interval contains an equal number of points, i.e., equal frequency discretization, or (2) using clustering. Another approach, which is used by the subspace clustering algorithm MAFIA, initially breaks the set of values of an attribute into a large number of equal width intervals and then combines intervals of similar density.

Regardless of the approach taken, the definition of the grid has a strong impact on the clustering results. We will consider specific aspects of this later.

The Density of Grid Cells

A natural way to define the density of a grid cell (or a more generally shaped region) is as the number of points divided by the volume of the region. In other words, density is the number of points per amount of space, regardless of the dimensionality of that space. Specific, low-dimensional examples of density are the number of road signs per mile (one dimension), the number of eagles per square kilometer of habitat (two dimensions), and the number of molecules of a gas per cubic centimeter (three dimensions). As mentioned, however, a common approach is to use grid cells that have the same volume so that the number of points per cell is a direct measure of the cell's density.

Example 8.8 (Grid-Based Density)**.** Figure 8.10 shows two sets of two-dimensional points divided into 49 cells using a 7-by-7 grid. The first set contains 200 points generated from a uniform distribution over a circle centered at (2, 3) of radius 2, while the second set has 100 points generated from a uniform distribution over a circle centered at (6, 3) of radius 1. The counts for the grid cells are shown in Table 8.2. Since the cells have equal volume (area), we can consider these values to be the densities of the cells. ■

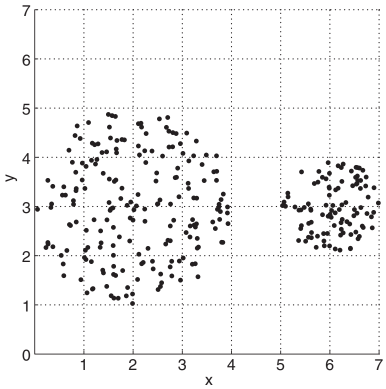


Figure 8.10. Grid-based density.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
4	**17**	**18**	**6**	0	0	0
14	**14**	**13**	**13**	0	**18**	**27**
11	**18**	**10**	**21**	0	**24**	**31**
3	**20**	**14**	**4**	0	0	0
0	0	0	0	0	0	0

Table 8.2. Point counts for grid cells.

Forming Clusters from Dense Grid Cells

Forming clusters from adjacent groups of dense cells is relatively straightforward. (In Figure 8.10, for example, it is clear that there would be two clusters.) There are, however, some issues. We need to define what we mean by adjacent cells. For example, does a two-dimensional grid cell have 4 adjacent cells or 8? Also, we need an efficient technique to find the adjacent cells, particularly when only occupied cells are stored.

The clustering approach defined by Algorithm 8.4 has some limitations that could be addressed by making the algorithm slightly more sophisticated. meaning what? For example, there are likely to be partially empty cells on the boundary of a cluster. Often, these cells are not dense. If so, they will be discarded and parts of a cluster will be lost. Figure 8.10 and Table 8.2 show that four parts of the larger cluster would be lost if the density threshold is 9. The clustering process could be modified to avoid discarding such cells, although this would require additional processing.

It is also possible to enhance basic grid-based clustering by using more than just density information. In many cases, the data has both spatial and non-spatial attributes. In other words, some of the attributes describe the location of objects in time or space, while other attributes describe other aspects of the objects. A common example is houses, which have both a location and a number of other characteristics, such as price or floor space in square feet. Because of spatial (or temporal) autocorrelation, objects in a particular cell often have similar values for their other attributes. In such cases, it is possible to filter the cells based on the statistical properties of one or more non-spatial attributes, e.g., average house price, and then form clusters based on the density of the remaining points.

Strengths and Limitations

On the positive side, grid-based clustering can be very efficient and effective. Given a partitioning of each attribute, a single pass through the data can determine the grid cell of every object and the count of every grid. Also, even though the number of potential grid cells can be high, grid cells need to be created only for non-empty cells. Thus, the time and space complexity of defining the grid, assigning each object to a cell, and computing the density of each cell is only $O(m)$, where m is the number of points. If adjacent, occupied cells can be efficiently accessed, for example, by using a search tree, then the entire clustering process will be highly efficient, e.g., with a time complexity of $O(m \log m)$. For this reason, the grid-based approach to density clustering forms the basis of a number of clustering algorithms, such as STING, GRIDCLUS, WaveCluster, Bang-Clustering, CLIQUE, and MAFIA.

On the negative side, grid-based clustering, like most density-based clustering schemes, is very dependent on the choice of the density threshold τ. If τ is too high, then clusters will be lost. If τ is too low, two clusters that should be separate may be joined. Furthermore, if there are clusters and noise of differing densities, then it might not be possible to find a single value of τ that works for all parts of the data space.

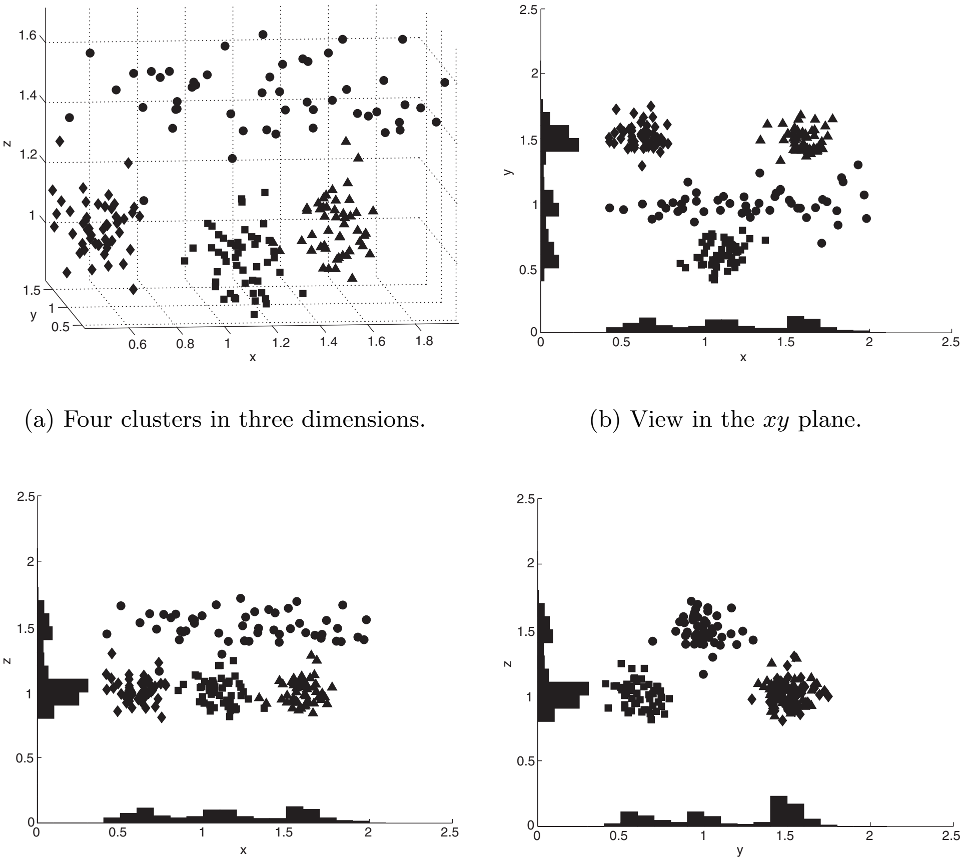
There are also a number of issues related to the grid-based approach. In Figure 8.10, for example, the rectangular grid cells do not accurately capture the density of the circular boundary areas. We could attempt to alleviate this problem by making the grid finer, but the number of points in the grid cells associated with a cluster would likely show more fluctuation because points in the cluster are not evenly distributed. Indeed, some grid cells, including those in the interior of the cluster, might even be empty. Another issue is that, depending on the placement or size of the cells, a group of points can appear in just one cell or be split between several different cells. The same group of points might be part of a cluster in the first case, but be discarded in the second. Finally, as dimensionality increases, the number of potential grid cells increases rapidly—exponentially in the number of dimensions. Even though it is not necessary to explicitly consider empty grid cells, it can easily happen that most grid cells contain a single object. In other words, grid-based clustering tends to work poorly for high-dimensional data.

8.3.2 Subspace Clustering

The clustering techniques considered until now found clusters by using all of the attributes. However, if only subsets of the features are considered, i.e., subspaces of the data, then the clusters that we find can be quite different from one subspace to another. There are two reasons that subspace clusters might be interesting. First, the data may be clustered with respect to a small set of attributes, but randomly distributed with respect to the remaining attributes. Second, there are cases in which different clusters exist in different sets of dimensions. Consider a data set that records the sales of various items at ????? various times. (The times are the dimensions and the items are the objects.) Some items might show similar behavior (cluster together) for particular sets of months, e.g., summer, but different clusters would likely be characterized by different months (dimensions).

Example 8.9 (Subspace Clusters)**.** Figure 8.11(a) shows a set of points in three-dimensional space. There are three clusters of points in the full space, which are represented by squares, diamonds, and triangles. In addition, there is one set of points, represented by circles, that is not a cluster in three-dimensional space. Each dimension (attribute) of the example data set is split into a fixed number (η) of equal width intervals. There are $\eta = 20$ intervals, each of size 0.1. This partitions the data space into rectangular cells of equal volume, and thus, the density of each unit is the fraction of points it contains. Clusters are contiguous groups of dense cells. To illustrate, if the threshold for a dense cell is $\xi = 0.06$, or 6% of the points, then three one-dimensional clusters can be identified in Figure 8.12, which shows a histogram of the data points of Figure 8.11(a) for the x attribute.

Figure 8.11(b) shows the points plotted in the xy plane. (The z attribute is ignored.) This figure also contains histograms along the x and y axes that



(a) Four clusters in three dimensions.

(b) View in the xy plane.

(c) View in the xz plane.

(d) View in the yz plane.

Figure 8.11. Example figures for subspace clustering.

show the distribution of the points with respect to their x and y coordinates, respectively. (A higher bar indicates that the corresponding interval contains relatively more points, and vice versa.) When we consider the y axis, we see three clusters. One is from the circle points that do not form a cluster in the full space, one consists of the square points, and one consists of the diamond and triangle points. There are also three clusters in the x dimension; they correspond to the three clusters—diamonds, triangles, and squares—in the full space. These points also form distinct clusters in the xy plane. Figure 8.11(c) shows the points plotted in the xz plane. There are two clusters, if we consider only the z attribute. One cluster corresponds to the points represented by

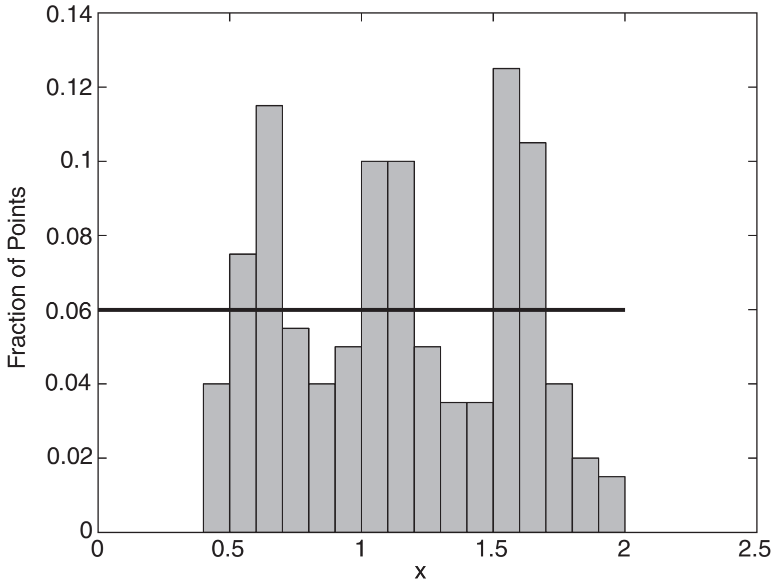


Figure 8.12. Histogram showing the distribution of points for the x attribute.

circles, while the other consists of the diamond, triangle, and square points. ????? These points also form distinct clusters in the xz plane. In Figure 8.11(d), there are three clusters when we consider both the y and z coordinates. One of these clusters consists of the circles; another consists of the points marked by squares. The diamonds and triangles form a single cluster in the yz plane. ■

These figures illustrate a couple of important facts. First, a set of points—the circles—may not form a cluster in the entire data space, but may form a cluster in a subspace. Second, clusters that exist in the full data space (or even a subspace) show up as clusters in lower-dimensional spaces. The first fact tells us that we need to look in subsets of dimensions to find clusters, while the second fact tells us that many of the clusters we find in subspaces are likely to be "shadows" (projections) of higher-dimensional clusters. The goal is to find the clusters and the dimensions in which they exist, but we are typically not interested in clusters that are projections of higher-dimensional clusters.

CLIQUE

CLIQUE (CLustering In QUEst) is a grid-based clustering algorithm that methodically finds subspace clusters. It is impractical to check each subspace for clusters because the number of such subspaces is exponential in the number of dimensions. Instead, CLIQUE relies on the following property:

Monotonicity property of density-based clusters If a set of points forms a density-based cluster in k dimensions (attributes), then the same set of points is also part of a density-based cluster in all possible subsets of those dimensions.

Consider a set of adjacent, k-dimensional cells that form a cluster; i.e., there is a collection of adjacent cells that have a density above the specified threshold ξ. A corresponding set of cells in $k-1$ dimensions can be found by omitting one of the k dimensions (attributes). The lower-dimensional cells are still adjacent, and each low-dimensional cell contains all points of the corresponding high-dimensional cell. It can contain additional points as well. Thus, a low-dimensional cell has a density greater than or equal to that of its corresponding high-dimensional cell. Consequently, the low-dimensional cells form a cluster; i.e., the points form a cluster with the reduced set of attributes.

Algorithm 8.5 gives a simplified version of the steps involved in CLIQUE. Conceptually, the CLIQUE algorithm is similar to the *Apriori* algorithm for finding frequent itemsets. See Chapter 4.

Algorithm 8.5 CLIQUE.

```
Find all the dense areas in the one-dimensional spaces corresponding to each
  attribute. This is the set of dense one-dimensional cells.
$k \leftarrow 2$
**repeat**
    Generate all candidate dense $k$-dimensional cells from dense $(k-1)$-dimensional
      cells.
    Eliminate cells that have fewer than $\xi$ points.
    $k \leftarrow k+1$
**until** There are no candidate dense $k$-dimensional cells.
Find clusters by taking the union of all adjacent, high-density cells.
Summarize each cluster using a small set of inequalities that describe the attribute
  ranges of the cells in the cluster.
```

Strengths and Limitations of CLIQUE

The most useful feature of CLIQUE is that it provides an efficient technique for searching subspaces for clusters. Since this approach is based on the well-known *Apriori* principle from association analysis, its properties are well understood. Another useful feature is CLIQUE's ability to summarize the list of cells that comprises a cluster with a small set of inequalities.

Many limitations of CLIQUE are identical to the previously discussed limitations of other grid-based density schemes. Other limitations are similar to those of the *Apriori* algorithm. Specifically, just as frequent itemsets can share items, the clusters found by CLIQUE can share objects. Allowing clusters to overlap can greatly increase the number of clusters and make interpretation difficult. Another issue is that *Apriori*—like CLIQUE—potentially has exponential time complexity. In particular, CLIQUE will have difficulty if too many dense cells are generated at lower values of k. Raising the density threshold ξ can alleviate this problem. Still another potential limitation of CLIQUE is explored in Exercise 25 on page 722.

8.3.3 DENCLUE: A Kernel-Based Scheme for Density-Based Clustering

DENCLUE (DENsity CLUstEring) is a density-based clustering approach that models 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 natural 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 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 the minimum density threshold, then the clusters associated with these local peaks are merged. Therefore, clusters of any shape can be discovered.

Example 8.10 (DENCLUE Density)**.** We illustrate these concepts with Figure 8.13, which shows a possible density function for a one-dimensional data set. Points A–E are the peaks of this density function and represent local density attractors. The dotted vertical lines delineate local regions of influence for the local density attractors. Points in these regions will become center-defined clusters. The dashed horizontal line shows a density threshold, ξ. All

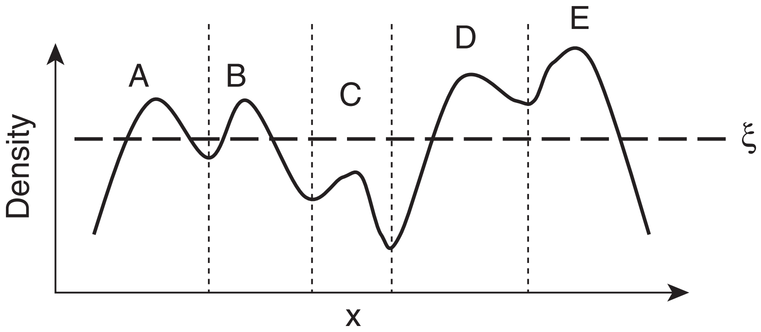


Figure 8.13. Illustration of DENCLUE density concepts in one dimension.

points associated with a local density attractor that has a density less than ξ, such as those associated with C, will be discarded. All other clusters are kept. Note that this can include points whose density is less than ξ, as long as they are associated with local density attractors whose density is greater than ξ. Finally, clusters that are connected by a path of points with a density above ξ are combined. Clusters A and B would remain separate, while clusters D and E would be combined. ■

The high-level details of the DENCLUE algorithm are summarized in Algorithm 8.6. Next, we explore various aspects of DENCLUE in more detail. First, we provide a brief overview of kernel density estimation and then present the grid-based approach that DENCLUE uses for approximating the density.

Algorithm 8.6 DENCLUE algorithm.

```
Derive a density function for the space occupied by the data points.
Identify the points that are local maxima.
  (These are the density attractors.)
Associate each point with a density attractor by moving in the direction of
  maximum increase in density.
Define clusters consisting of points associated with a particular density attractor.
Discard clusters whose density attractor has a density less than a user-specified
  threshold of $\xi$.
Combine clusters that are connected by a path of points that all have a density
  of $\xi$ or higher.
```

Kernel Density Estimation

DENCLUE is based on a well-developed area of statistics and pattern recognition that is known as **kernel density estimation**. The goal of this collection of techniques (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 or **kernel function**. The overall density function is simply the sum of the influence functions associated with each point.

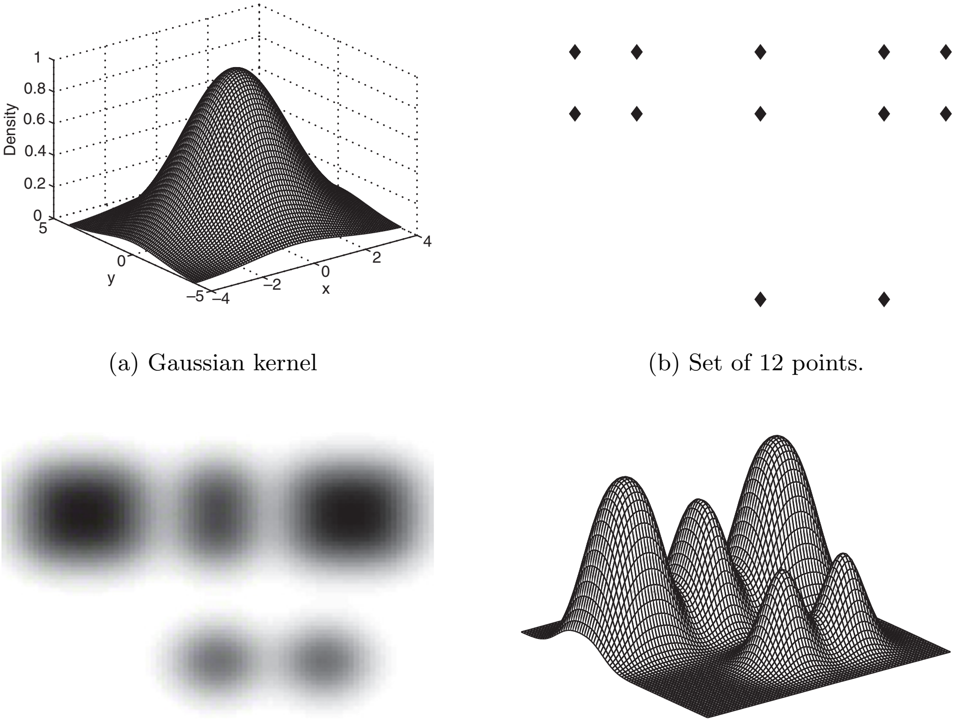
Typically, the influence or kernel function is symmetric (the same in all directions) and its value (contribution) decreases as the distance from the point increases. For example, for a particular point, **x**, the Gaussian function, $K(y) = e^{-distance(\mathbf{x},\mathbf{y})^2/2\sigma^2}$, is often used as a kernel function. σ is a parameter, analogous to standard deviation, which governs how quickly the influence of a point diminishes with distance. Figure 8.14(a) shows what a Gaussian density function would look like for a single point in two dimensions, while Figures 8.14(c) and 8.14(d) show the overall density function produced by applying the Gaussian influence function to the set of points shown in Figure 8.14(b).

Implementation Issues

Computation of kernel density can be quite expensive, and DENCLUE uses a number of approximations to implement its basic approach efficiently. First, it explicitly computes density only at data points. However, this still would result in an $O(m^2)$ time complexity because the density at each point is a function of the density contributed by every point. To reduce the time complexity, DENCLUE uses a grid-based implementation to efficiently define neighborhoods and thus limit the number of points that need to be considered to define the density at a point. First, a preprocessing step creates a set of grid cells. Only occupied cells are created, and these cells and their related information can be efficiently accessed via a search tree. Then, when computing the density of a point and finding its nearest density attractor, DENCLUE considers only the points in the neighborhood; i.e., points in the same cell and in cells that are connected to the point's cell. While this approach can sacrifice some accuracy with respect to density estimation, computational complexity is greatly reduced.

Strengths and Limitations of DENCLUE

DENCLUE has a solid theoretical foundation because it is based on the concept of kernel density estimation, which is a well-developed area of statistics.



(a) Gaussian kernel

(b) Set of 12 points.

(c) Overall density—gray scale plot.

(d) Overall density—surface plot.

Figure 8.14. Example of the Gaussian influence (kernel) function and an overall density function.

For this reason, DENCLUE provides a more flexible and potentially more accurate way to compute density than other grid-based clustering techniques and DBSCAN. (DBSCAN is a special case of DENCLUE.) An approach based on kernel density functions is inherently computationally expensive, but DENCLUE employs grid-based techniques to address such issues. Nonetheless, DENCLUE can be more computationally expensive than other density-based clustering techniques. Also, the use of a grid can adversely affect the accuracy of the density estimation, and it makes DENCLUE susceptible to problems common to grid-based approaches; e.g., the difficulty of choosing the proper grid size. More generally, DENCLUE shares many of the strengths and limitations of other density-based approaches. For instance, DENCLUE is good at handling noise and outliers and it can find clusters of different shapes and size, but it has trouble with high-dimensional data and data that contains clusters of widely different densities.

8.4 Graph-Based Clustering

Section 5.3 discussed a number of clustering techniques that took a graph-based view of data, in which data objects are represented by nodes and the proximity between two data objects is represented by the weight of the edge between the corresponding nodes. This section considers some additional graph-based clustering algorithms that use a number of key properties and characteristics of graphs. The following are some key approaches, different subsets of which are employed by these algorithms.

1. Sparsify the proximity graph to keep only the connections of an object with its nearest neighbors. This sparsification is useful for handling noise and outliers. It also allows the use of highly efficient graph partitioning algorithms that have been developed for sparse graphs.

2. Define a similarity measure between two objects based on the number of nearest neighbors that they share. This approach, which is based on the observation that an object and its nearest neighbors usually belong to the same class, is useful for overcoming problems with high dimensionality and clusters of varying density.

3. Define core objects and build clusters around them. To do this for graph-based clustering, it is necessary to introduce a notion of density-based on a proximity graph or a sparsified proximity graph. As with DBSCAN, building clusters around core objects leads to a clustering technique that can find clusters of differing shapes and sizes.

4. Use the information in the proximity graph to provide a more sophisticated evaluation of whether two clusters should be merged. Specifically, two clusters are merged only if the resulting cluster will have characteristics similar to the original two clusters.

We begin by discussing the sparsification of proximity graphs, providing three examples of techniques whose approach to clustering is based solely on this technique: MST, which is equivalent to the single link clustering algorithm, Opossum, and spectral clustering. We then discuss Chameleon, a hierarchical clustering algorithm that uses a notion of self-similarity to determine if clusters should be merged. We next define Shared Nearest Neighbor (SNN) similarity, a new similarity measure, and introduce the Jarvis-Patrick clustering algorithm, which uses this similarity. Finally, we discuss how to define density and core objects based on SNN similarity and introduce an SNN density-based clustering algorithm, which can be viewed as DBSCAN with a new similarity measure.

8.4.1 Sparsification

The m by m proximity matrix for m data points can be represented as a dense graph in which each node is connected to all others and the weight of the edge between any pair of nodes reflects their pairwise proximity. Although every object has some level of similarity to every other object, for most data sets, objects are highly similar to a small number of objects and weakly similar to most other objects. This property can be used to sparsify the proximity graph (matrix), by setting many of these low-similarity (high-dissimilarity) values to 0 before beginning the actual clustering process. The sparsification may be performed, for example, by breaking all links that have a similarity (dissimilarity) below (above) a specified threshold or by keeping only links to the k-nearest neighbors of point. This latter approach creates what is called a **k-nearest neighbor graph**.

Sparsification has several beneficial effects:

- **Data size is reduced.** The amount of data that needs to be processed to cluster the data is drastically reduced. Sparsification can often eliminate more than 99% of the entries in a proximity matrix. As a result, the size of problems that can be handled is increased.

- **Clustering often works better.** Sparsification techniques keep the connections to their nearest neighbors of an object while breaking the connections to more distant objects. This is in keeping with the **nearest neighbor principle** that the nearest neighbors of an object tend to belong to the same class (cluster) as the object itself. This reduces the impact of noise and outliers and sharpens the distinction between clusters.

- **Graph partitioning algorithms can be used.** There has been a considerable amount of work on heuristic algorithms for finding min-cut partitionings of sparse graphs, especially in the areas of parallel computing and the design of integrated circuits. Sparsification of the proximity graph makes it possible to use graph partitioning algorithms for the clustering process. For example, Opossum and Chameleon use graph partitioning.

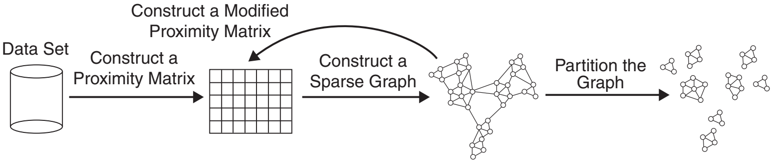

Data Set
Construct a Proximity Matrix
Construct a Modified Proximity Matrix
Construct a Sparse Graph
Partition the Graph

Figure 8.15. Ideal process of clustering using sparsification.

Sparsification of the proximity graph should be regarded as an initial step before the use of actual 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 easy for a single edge to link two clusters or for a single cluster to be split into several disconnected subclusters. As we shall see when we discuss Jarvis-Patrick and SNN density-based clustering, the sparse proximity graph is often modified to yield a new proximity graph. This new proximity graph can again be sparsified. Clustering algorithms work with the proximity graph that is the result of all these preprocessing steps. This process is summarized in Figure 8.15.

8.4.2 Minimum Spanning Tree (MST) Clustering

In Section 5.3, where we described agglomerative hierarchical clustering techniques, we mentioned that divisive hierarchical clustering algorithms also exist. We saw an example of one such technique, bisecting K-means, in Section 5.2.3. Another divisive hierarchical technique, **MST**, starts with the minimum spanning tree of the proximity graph and can be viewed as an application of sparsification for finding clusters. We briefly describe this algorithm. Interestingly, this algorithm also produces the same clustering as single link agglomerative clustering. See Exercise 18 on page 720.

A **minimum spanning tree** of a graph is a subgraph that (1) has no cycles, i.e., is a tree, (2) contains all the nodes of the graph, and (3) has the minimum total edge weight of all possible spanning trees. The terminology, minimum spanning tree, assumes that we are working only with dissimilarities or distances, and we will follow this convention. This is not a limitation, however, since we can convert similarities to dissimilarities or modify the notion of a minimum spanning tree to work with similarities. An example of a minimum spanning tree for some two-dimensional points is shown in Figure 8.16.

The MST divisive hierarchical algorithm is shown in Algorithm 8.7. The first step is to find the MST of the original dissimilarity graph. Note that a

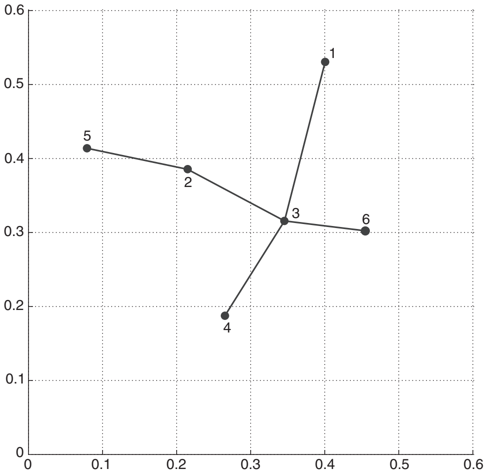


Figure 8.16. Minimum spanning tree for a set of six two-dimensional points.

minimum spanning tree can be viewed as a special type of sparsified graph. Step 3 can also be viewed as graph sparsification. Hence, MST can be viewed as a clustering algorithm based on the sparsification of the dissimilarity graph.

Algorithm 8.7 MST divisive hierarchical clustering algorithm.

```
1: Compute a minimum spanning tree for the dissimilarity graph.
2: **repeat**
3:    Create a new cluster by breaking the link corresponding to the largest
      dissimilarity.
4: **until** Only singleton clusters remain.
```

8.4.3 OPOSSUM: Optimal Partitioning of Sparse Similarities Using METIS

OPOSSUM is a clustering technique for clustering sparse, high-dimensional data, e.g., document or market basket data. Like MST, it performs clustering based on the sparsification of a proximity graph. However, OPOSSUM uses the METIS algorithm, which was specifically created for partitioning sparse graphs. The steps of OPOSSUM are given in Algorithm 8.8.

The similarity measures used are those appropriate for sparse, high-dimensional data, such as the extended Jaccard measure or the cosine measure. The METIS graph partitioning program partitions a sparse graph into k distinct

Algorithm 8.8 OPOSSUM clustering algorithm.

```
1: Compute a sparsified similarity graph.
2: Partition the similarity graph into $k$ distinct components (clusters) using METIS.
```

components, where k is a user-specified parameter, in order to (1) minimize the weight of the edges (the similarity) between components and (2) fulfill a balance constraint. OPOSSUM uses one of the following two balance constraints: (1) the number of objects in each cluster must be roughly the same, or (2) the sum of the attribute values must be roughly the same. The second constraint is useful when, for example, the attribute values represent the cost of an item.

Strengths and Weaknesses

OPOSSUM is simple and fast. It partitions the data into roughly equal-sized clusters, which, depending on the goal of the clustering, can be viewed as an advantage or a disadvantage. Because they are constrained to be of roughly equal size, clusters can be broken or combined. However, if OPOSSUM is used to generate a large number of clusters, then these clusters are typically relatively pure pieces of larger clusters. Indeed, OPOSSUM is similar to the initial step of the Chameleon clustering routine, which is discussed next.

8.4.4 Chameleon: Hierarchical Clustering with Dynamic Modeling

Agglomerative hierarchical clustering techniques operate by merging the two most similar clusters, where the definition of cluster similarity depends on the particular algorithm. Some agglomerative algorithms, such as group average, base their notion of similarity on the strength of the connections between the two clusters (e.g., the pairwise similarity of points in the two clusters), while other techniques, such as the single link method, use the closeness of the clusters (e.g., the minimum distance between points in different clusters) to measure cluster similarity. Although there are two basic approaches, using only one of these two approaches can lead to mistakes in merging clusters. Consider Figure 8.17, which shows four clusters. If we use the closeness of clusters (as measured by the closest two points in different clusters) as our merging criterion, then we would merge the two circular clusters, (c) and (d),which almost touch, instead of the rectangular clusters, (a) and (b), which are separated by a small gap. However, intuitively, we should have merged

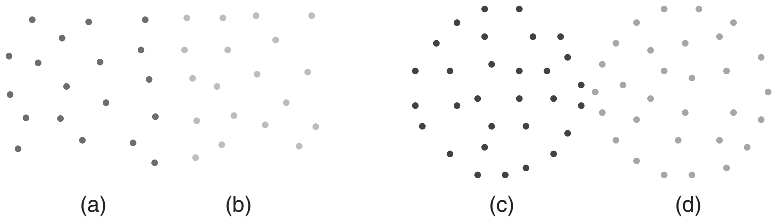


Figure 8.17. Situation in which closeness is not the appropriate merging criterion. ©1999, IEEE

rectangular clusters, (a) and (b). Exercise 20 on page 721 asks for an example of a situation in which the strength of connections likewise leads to an unintuitive result.

Another problem is that most clustering techniques have a global (static) model of clusters. For instance, K-means assumes that the clusters will be globular, while DBSCAN defines clusters based on a single density threshold. Clustering schemes that use such a global model cannot handle cases in which cluster characteristics, such as size, shape, and density, vary widely between clusters. As an example of the importance of the local (dynamic) modeling of clusters, consider Figure 8.18. If we use the closeness of clusters to determine which pair of clusters should be merged, as would be the case if we used, for example, the single link clustering algorithm, then we would merge clusters (a) and (b). However, we have not taken into account the characteristics of each individual cluster. Specifically, we have ignored the density of the individual clusters. For clusters (a) and (b), which are relatively dense, the distance between the two clusters is significantly larger than the distance between a point and its nearest neighbors within the same cluster. This is not the case for clusters (c) and (d), which are relatively sparse. Indeed, when clusters (c) and (d) are merged, they yield a cluster that seems more similar to the original clusters than the cluster that results from merging clusters (a) and (b).

Chameleon is an agglomerative clustering algorithm that addresses the issues of the previous two paragraphs. It combines an initial partitioning of the data, using an efficient graph partitioning algorithm, with a novel hierarchical clustering scheme that uses the notions of closeness and interconnectivity, together with the local modeling of clusters. The key idea is that two clusters should be merged only if the resulting cluster is similar to the two original clusters. Self-similarity is described first, and then the remaining details of the Chameleon algorithm are presented.

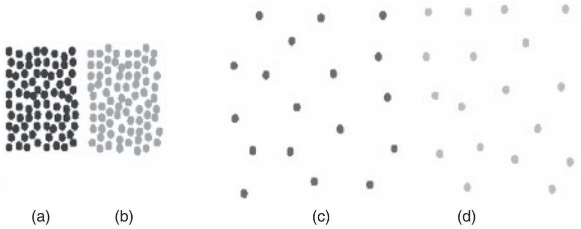


Figure 8.18. Illustration of the notion of relative closeness. ©1999, IEEE

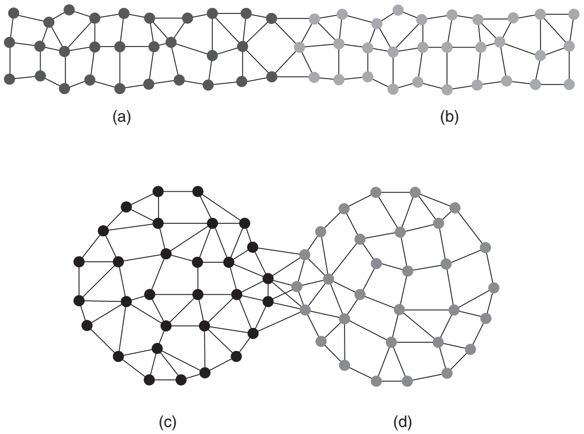


Figure 8.19. Illustration of the notion of relative interconnectedness. ©1999, IEEE

Deciding Which Clusters to Merge

The agglomerative hierarchical clustering techniques considered in Section 5.3 repeatedly combine the two closest clusters and are principally distinguished from one another by the way they define cluster proximity. In contrast, Chameleon aims to merge the pair of clusters that results in a cluster that is most similar to the original pair of clusters, as measured by closeness and interconnectivity. Because this approach depends only on the pair of clusters and not on a global model, Chameleon can handle data that contains clusters with widely different characteristics.

Following are more detailed explanations of the properties of closeness and interconnectivity. To understand these properties, it is necessary to take a proximity graph viewpoint and to consider the number of the links and the strength of those links among points within a cluster and across clusters.

- **Relative Closeness (RC)** is the absolute closeness of two clusters normalized by the internal closeness of the clusters. Two clusters are combined only if the points in the resulting cluster are almost as close to each other as in each of the original clusters. Mathematically,

$$RC(C_i, C_j) = \frac{\bar{S}_{EC}(C_i, C_j)}{\frac{m_i}{m_i+m_j}\bar{S}_{EC}(C_i) + \frac{m_j}{m_i+m_j}\bar{S}_{EC}(C_j)}, \tag{8.17}$$

  where m_i and m_j are the sizes of clusters C_i and C_j, respectively; $\bar{S}_{EC}(C_i, C_j)$ is the average weight of the edges (of the k-nearest neighbor graph) that connect clusters C_i and C_j; $\bar{S}_{EC}(C_i)$ is the average weight of edges if we bisect cluster C_i; and $\bar{S}_{EC}(C_j)$ is the average weight of edges if we bisect cluster C_j. (EC stands for edge cut.) Figure 8.18 illustrates the notion of relative closeness. As discussed previously, while clusters (a) and (b) are closer in absolute terms than clusters (c) and (d), this is not true if the nature of the clusters is taken into account.

- **Relative Interconnectivity (RI)** is the absolute interconnectivity of two clusters normalized by the internal connectivity of the clusters. Two clusters are combined if the points in the resulting cluster are almost as strongly connected as points in each of the original clusters. Mathematically,

$$RI(C_i, C_j) = \frac{EC(C_i, C_j)}{\frac{1}{2}(EC(C_i) + EC(C_j))}, \tag{8.18}$$

  where $EC(C_i, C_j)$ is the sum of the edges (of the 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. Figure 8.19 illustrates the notion of relative interconnectivity. The two circular clusters, (c) and (d), have more connections than the rectangular clusters, (a) and (b). However, merging (c) and (d) produces a cluster that has connectivity quite different from that of (c) and (d). In contrast, merging (a) and (b) produces a cluster with connectivity very similar to that of (a) and (b).

RI and RC can be combined in many different ways to yield an overall measure of self-similarity. One approach used in Chameleon is to merge the pair of clusters that maximizes $RI(C_i, C_j) * RC(C_i, C_j)^\alpha$, where α is a user-specified parameter that is typically greater than 1.

Chameleon Algorithm

Chameleon consists of three key steps: sparsification, graph partitioning, and hierarchical clustering. Algorithm 8.9 and Figure 8.20 describe these steps.

Algorithm 8.9 Chameleon algorithm.

```
1: Build a $k$-nearest neighbor graph.
2: Partition the graph using a multilevel graph partitioning algorithm.
3: repeat
4:     Merge the clusters that best preserve the cluster self-similarity with respect to
       relative interconnectivity and relative closeness.
5: until No more clusters can be merged.
```

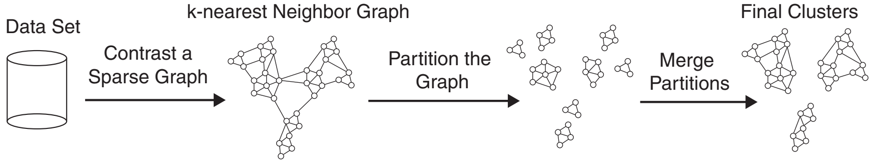


Figure 8.20. Overall process by which Chameleon performs clustering. ©1999, IEEE

Sparsification The first step in Chameleon is to generate a k-nearest neighbor graph. Conceptually, such a graph is derived from the proximity graph, and it contains links only between a point and its k-nearest neighbors, i.e., the points to which it is closest. As mentioned, working with a sparsified proximity graph instead of the full proximity graph can significantly reduce the effects of noise and outliers and improve computational efficiency.

Graph Partitioning

Once a sparsified graph has been obtained, an efficient multilevel graph partitioning algorithm, such as METIS (see Bibliographic Notes), can be used to partition the data set. Chameleon starts with an all-inclusive graph (cluster) and then bisects the largest current subgraph (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 roughly equally sized groups of well-connected vertices (highly similar data points). The goal is to ensure that each partition contains objects mostly from one true cluster.

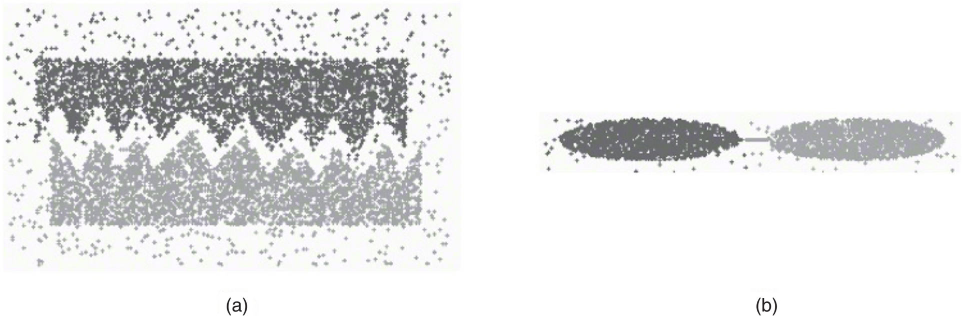
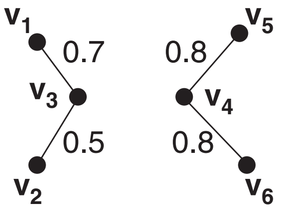


Figure 8.21. Chameleon applied to cluster a pair of two-dimensional sets of points. ©1999, IEEE

Agglomerative Hierarchical Clustering As discussed previously, Chameleon merges clusters based on the notion of self-similarity. Chameleon can be parameterized to merge more than one pair of clusters in a single step and to stop before all objects have been merged into a single cluster.

Complexity Assume that m is the number of data points and p is the number of partitions. Performing an agglomerative hierarchical clustering of the p partitions obtained from the graph partitioning requires time $O(p^2 \log p)$. (See Section 5.3.1.) The amount of time required for partitioning the graph is $O(mp + m \log m)$. The time complexity of graph sparsification depends on how much time it takes to build the k-nearest neighbor graph. For low-dimensional data, this takes $O(m \log m)$ time if a k-d tree or a similar type of data structure is used. Unfortunately, such data structures only work well for low-dimensional data sets, and thus, for high-dimensional data sets, the time complexity of the sparsification becomes $O(m^2)$. Because only the k-nearest neighbor list needs to be stored, the space complexity is $O(km)$ plus the space required to store the data.

Example 8.11. Chameleon was applied to two data sets that clustering algorithms such as K-means and DBSCAN have difficulty clustering. The results of this clustering are shown in Figure 8.21. The clusters are identified by the shading of the points. In Figure 8.21(a), the two clusters are irregularly shaped and quite close to each other. Also, noise is present. In Figure 8.21(b), the two clusters are connected by a bridge, and again, noise is present. Nonetheless, Chameleon identifies what most people would identify as the natural clusters. Chameleon has specifically been shown to be very effective for clustering spatial data. Finally, notice that Chameleon does not discard noise points, as do other clustering schemes, but instead assigns them to the clusters. ■



$$W = \begin{bmatrix} 0 & 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0.7 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.8 & 0.8 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0.7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.8 \end{bmatrix}$$

$$L = \begin{bmatrix} 0.7 & 0 & -0.7 & 0 & 0 & 0 \\ 0 & 0.5 & -0.5 & 0 & 0 & 0 \\ -0.7 & -0.5 & 1.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.6 & -0.8 & -0.8 \\ 0 & 0 & 0 & -0.8 & 0.8 & 0 \\ 0 & 0 & 0 & -0.8 & 0 & 0.8 \end{bmatrix} \quad \Lambda = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.58 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.4 \end{bmatrix} \quad V = \begin{bmatrix} 0.58 & 0 & 0.64 & 0 & -0.50 & 0 \\ 0.58 & 0 & -0.76 & 0 & -0.31 & 0 \\ 0.58 & 0 & 0.11 & 0 & 0.81 & 0 \\ 0 & -0.58 & 0 & 0 & 0 & -0.82 \\ 0 & -0.5 & 0 & -0.71 & 0 & 0.41 \\ 0 & -0.5 & 0 & 0.71 & 0 & 0.41 \end{bmatrix}$$

Figure 8.22. Example of a similarity graph with two connected components along with its weighted adjacency matrix (**W**), graph Laplacian matrix (**L**), and eigendecomposition.

Strengths and Limitations

Chameleon can effectively cluster spatial data, even though noise and outliers are present and the clusters are of different shapes, sizes, and density. Chameleon assumes that the groups of objects produced by the sparsification and graph partitioning process are subclusters; i.e., that most of the points in a partition belong to the same true cluster. If not, then agglomerative hierarchical clustering will only compound the errors because it can never separate objects that have been wrongly put together. (See the discussion in Section 5.3.4.) Thus, Chameleon has problems when the partitioning process does not produce subclusters, as is often the case for high-dimensional data.

8.4.5 Spectral Clustering

Spectral clustering is an elegant graph partitioning approach that exploits properties of the similarity graph to determine the cluster partitions. Specifically, it examines the graph's spectrum, i.e., eigenvalues and eigenvectors associated with the adjacency matrix of the graph, to identify the natural clusters of the data. To motivate the ideas behind this approach, consider the similarity graph shown in Figure 8.22 for a data set that contains 6 data points. The link weights in the graph are computed based on some similarity measure, with a threshold applied to remove links with low similarity values. The sparsification produces a graph with two connected components, which trivially represent the two clusters in the data, $\{v_1, v_2, v_3\}$ and $\{v_4, v_5, v_6\}$.

The top right-hand panel of the figure also shows the weighted adjacency matrix of the graph, denoted as $\mathbf{W}$, and a diagonal matrix, $\mathbf{D}$, whose diagonal elements correspond to the sum of the weights of the links incident to each node in the graph, i.e.,

$$D_{ij} = \begin{cases} \sum_k W_{ik}, & \text{if } i = j; \\ 0, & \text{otherwise.} \end{cases}$$

Note that the rows and columns of the weighted adjacency matrix have been ordered in such a way that nodes belonging to the same connected component are next to each other. With this ordering, the matrix $\mathbf{W}$ has a block structure of the form

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_2 \end{pmatrix},$$

in which the off-diagonal blocks are matrices of zero values since there are no links connecting a node from the first connected component to a node from the second connected component. Indeed, if the sparse graph contains k connected components, its weighted adjacency matrix can be re-ordered into the following block diagonal form:

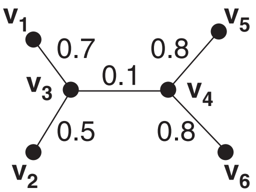
$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{W}_2 & \cdots & \mathbf{0} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{W}_k \end{pmatrix}, \tag{8.19}$$

This example suggests the possibility of identifying the inherent clusters of a data set by examining the block structure of its weighted adjacency matrix.

Unfortunately, unless the clusters are well-separated, the adjacency matrices associated with most similarity graphs are not in block diagonal form. For example, consider the graph shown in Figure 8.23, in which there is a link between nodes v_3 and v_4, with a low similarity value. If we are interested in generating two clusters, we could break the weakest link, located between (v_3, v_4), to split the graph into two partitions. Because there is only one connected component in the graph, the block structure in $\mathbf{W}$ is harder to discern.

Fortunately, there is a more objective way to create the cluster partitions by considering the graph spectrum. First, we need to compute the graph Laplacian matrix, which is formally defined as follows:

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \tag{8.20}$$



$$W = \begin{bmatrix} 0 & 0 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0.7 & 0.5 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0.8 & 0.8 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \\ 0 & 0 & 0 & 0.8 & 0 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0.7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.7 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.8 \end{bmatrix}$$

$$L = \begin{bmatrix} 0.7 & 0 & -0.7 & 0 & 0 & 0 \\ 0 & 0.5 & -0.5 & 0 & 0 & 0 \\ -0.7 & -0.5 & 1.3 & -0.1 & 0 & 0 \\ 0 & 0 & -0.1 & 1.7 & -0.8 & -0.8 \\ 0 & 0 & 0 & -0.8 & 0.8 & 0 \\ 0 & 0 & 0 & -0.8 & 0 & 0.8 \end{bmatrix} \quad \Lambda = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.06 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.58 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.80 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.88 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2.47 \end{bmatrix} \quad V = \begin{bmatrix} 0.41 & -0.41 & 0.65 & 0 & 0.48 & -0.04 \\ 0.41 & -0.43 & -0.75 & 0 & 0.29 & 0.03 \\ 0.41 & -0.38 & 0.11 & 0 & -0.81 & 0.10 \\ 0.41 & 0.38 & 0 & 0 & 0.08 & -0.82 \\ 0.41 & 0.42 & 0 & -0.71 & 0.06 & 0.39 \\ 0.41 & 0.42 & 0 & 0.71 & 0.06 & 0.39 \end{bmatrix}$$

Figure 8.23. Example of a similarity graph with a single connected component along with its weighted adjacency matrix ($\mathbf{W}$), graph Laplacian matrix ($\mathbf{L}$), and eigendecomposition.

The graph Laplacian matrices for the examples shown in Figures 8.22 and 8.23 are depicted in the bottom left panel of both diagrams. The matrix has several notable properties:

1. It is a symmetric matrix since both $\mathbf{W}$ and $\mathbf{D}$ are symmetric.

2. It is a positive semi-definite matrix, which means $\mathbf{v}^T\mathbf{L}\mathbf{v} \geq 0$ for any input vector $\mathbf{v}$.

3. All eigenvalues of $\mathbf{L}$ must be non-negative. The eigenvalues and eigenvectors for the graphs shown in Figure 8.22 and 8.23 are denoted in the diagrams as $\mathbf{\Lambda}$ and $\mathbf{V}$, respectively. Note that the eigenvalues of the graph Laplacian matrix are given by the diagonal elements of $\mathbf{\Lambda}$.

4. The smallest eigenvalue of $\mathbf{L}$ is zero, with the corresponding eigenvector $\mathbf{e}$, which is a vector of 1s. This is because

$$\mathbf{We} = \begin{pmatrix} W_{11} & W_{12} & \cdots & W_{1n} \\ W_{21} & W_{22} & \cdots & W_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ W_{n1} & W_{n2} & \cdots & W_{nn} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ \cdots \\ 1 \end{pmatrix} = \begin{pmatrix} \sum_j W_{1j} \\ \sum_j W_{2j} \\ \cdots \\ \sum_j W_{nj} \end{pmatrix}$$

$$\mathbf{De} = \begin{pmatrix} \sum_j W_{1j} & 0 & \cdots & 0 \\ 0 & \sum_j W_{2j} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & \sum_j W_{nj} \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ \cdots \\ 1 \end{pmatrix} = \begin{pmatrix} \sum_j W_{1j} \\ \sum_j W_{2j} \\ \cdots \\ \sum_j W_{nj} \end{pmatrix}$$

Thus, $\mathbf{We} = \mathbf{De}$, which is equivalent to $(\mathbf{D} - \mathbf{W})\mathbf{e} = \mathbf{0}$. This can be simplified into the eigenvalue equation $\mathbf{Le} = 0\mathbf{e}$ since $\mathbf{L} = \mathbf{D} - \mathbf{W}$.

5. A graph with k connected components has an adjacency matrix $\mathbf{W}$ in block diagonal form as shown in Equation 8.19. Its graph Laplacian matrix also has a block diagonal form

$$\mathbf{L} = \begin{pmatrix} \mathbf{L}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{L}_2 & \cdots & \mathbf{0} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{L}_k \end{pmatrix},$$

In addition, its graph Laplacian matrix has k eigenvalues of zeros, with the corresponding eigenvectors

$$\begin{pmatrix} \mathbf{e}_1 \\ \mathbf{0} \\ \cdots \\ \mathbf{0} \end{pmatrix}, \begin{pmatrix} \mathbf{0} \\ \mathbf{e}_2 \\ \cdots \\ \mathbf{0} \end{pmatrix}, \cdots, \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \cdots \\ \mathbf{e}_k \end{pmatrix},$$

where the $\mathbf{e}_i$'s are vectors of 1's and $\mathbf{0}$'s are vectors of 0's. For example, the graph shown in Figure 8.22 contains two connected components, which is why its graph Laplacian matrix has two eigenvalues of zeros. More importantly, its first two eigenvectors (normalized to unit length),

$$\begin{array}{c} v_1 \longrightarrow \\ v_2 \longrightarrow \\ v_3 \longrightarrow \\ v_4 \longrightarrow \\ v_5 \longrightarrow \\ v_6 \longrightarrow \end{array} \begin{pmatrix} 0.58 & 0 \\ 0.58 & 0 \\ 0.58 & 0 \\ 0 & -0.58 \\ 0 & -0.58 \\ 0 & -0.58 \end{pmatrix},$$

corresponding to the first two columns in $\mathbf{V}$, provide information about the cluster membership of each node. A node that belong to the first cluster has a positive value in the its first eigenvector and a zero value in its second eigenvector, whereas a node that belong to the second cluster has a zero value in the first eigenvector and a negative value in the second eigenvector.

The graph shown in Figure 8.23 has one eigenvalue of zero because it has only one connected component. Nevertheless, if we examine the first two eigenvectors of its graph Laplacian matrix

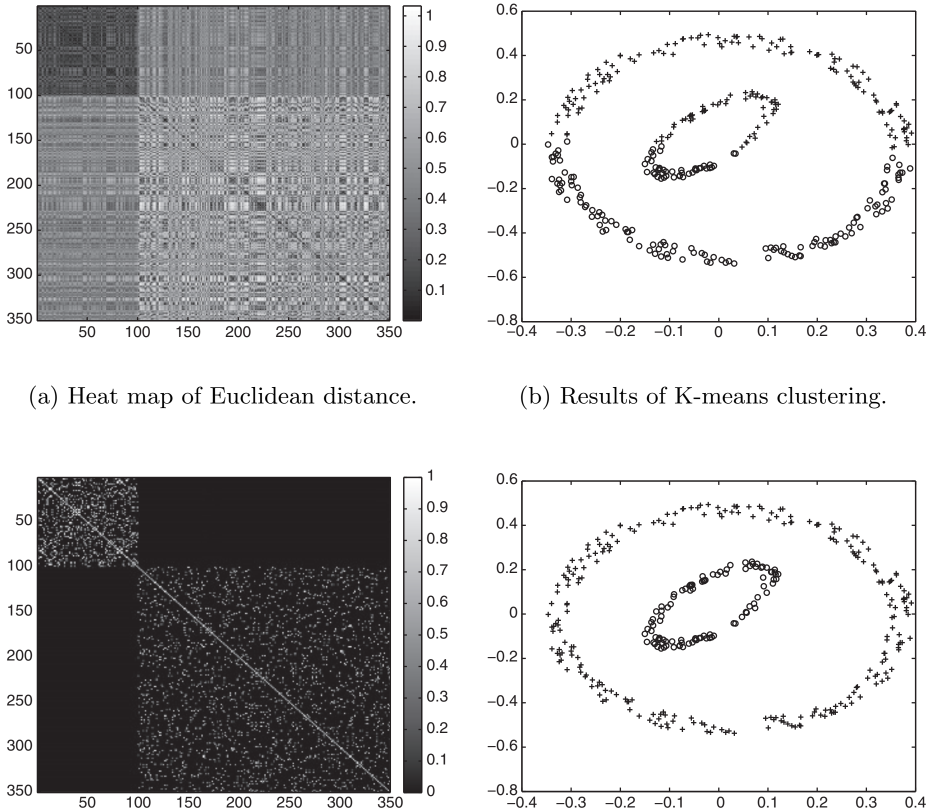
$$\begin{array}{c} v_1 \rightarrow \\ v_2 \rightarrow \\ v_3 \rightarrow \\ v_4 \rightarrow \\ v_5 \rightarrow \\ v_6 \rightarrow \end{array} \begin{pmatrix} 0.41 & -0.41 \\ 0.41 & -0.43 \\ 0.41 & -0.38 \\ 0.41 & 0.38 \\ 0.41 & 0.42 \\ 0.41 & 0.42 \end{pmatrix},$$

the graph can be easily split into two clusters since the set of nodes $\{v_1, v_2, v_3\}$ has a negative value in the second eigenvector whereas $\{v_4, v_5, v_6\}$ has a positive value in the second eigenvector. In short, the eigenvectors of the graph Laplacian matrix contain information that can be used to partition the graph into its underlying components. However, instead of manually checking the eigenvectors, it is common practice to apply a simple clustering algorithm such as K-means to help extract the clusters from the eigenvectors. A summary of the spectral clustering algorithm is given in Algorithm 8.10.

Algorithm 8.10 Spectral clustering algorithm.

```
Create a sparsified similarity graph $\mathcal{G}$.
Compute the graph Laplacian for $\mathcal{G}$, $\mathbf{L}$ (see Equation (8.20)).
Create a matrix $\mathbf{V}$ from the first $k$ eigenvectors of $\mathbf{L}$.
Apply K-means clustering on $\mathbf{V}$ to obtain the $k$ clusters.
```

Example 8.12. Consider the two-dimensional ring data shown in Figure 8.24(b), which contains 350 data points. The first 100 points belong to the inner ring while the remaining 250 points belong to the outer ring. A heat map showing the Euclidean distance between every pair of points is depicted in Figure 8.24(a). While the points in the inner ring are relatively close to each other, those located in the outer ring can be quite far from each other. As a result, standard clustering algorithms such as K-means perform poorly on the data. In contrast, applying spectral clustering on the sparsified similarity graph can produce the correct clustering results (see Figure 8.24(d)). Here, the similarity between points is calculated using the Gaussian radial basis function and the graph is sparsified by choosing the 10-nearest neighbors for each data point. The sparsification reduces the similarity between a data point located in the inner ring and a corresponding point in the outer ring, which enables spectral clustering to effectively partition the data set into two clusters. ■



(a) Heat map of Euclidean distance.

(b) Results of K-means clustering.

(c) Heat map of sparsified similarity.

(d) Results of spectral clustering.

Figure 8.24. Application of K-means and spectral clustering to a two-dimensional ring data.

Relationship between Spectral Clustering and Graph Partitioning

The objective of graph partitioning is to break the weak links in a graph until the desired number of cluster partitions is obtained. One way to assess the quality of the partitions is by summing up the weights of the links that were removed. The resulting measure is known as graph cut. Unfortunately, minimizing the graph cut of the partitions alone is insufficient as it tends to produce clusters with highly imbalanced sizes. For example, consider the graph shown in Figure 8.25. Suppose we are interested in partitioning the graph into two connected components. The graph cut measure prefers to break the link between v_4 and v_5 because it has the lowest weight.

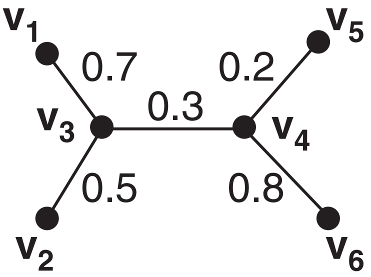


Figure 8.25. Example to illustrate the limitation of using graph cut as evaluation measure for graph partitioning.

Unfortunately, such a split would create one cluster with a single isolated node and another cluster containing all the remaining nodes. To overcome this limitation, alternative measures have been proposed including

$$\text{Ratio cut}(C_1, C_2, \cdots, C_k) = \frac{1}{2}\sum_{i=1}^{k} \frac{\sum_{p\in C_i, q\notin C_i} W_{pq}}{|C_i|},$$

where $C_1, C_2, \cdots, C_k$ denote the cluster partitions. The numerator represents the sum of the weights of the broken links, i.e., the graph cut, while the denominator represents the size of each cluster partition. Such a measure can be used to ensure that the resulting clusters are more balanced in terms of their sizes. More importantly, it can be shown that minimizing the ratio cut for a graph is equivalent to finding a cluster membership matrix $\mathbf{Y}$ that minimizes the expression $\text{Tr}[\mathbf{Y}^T\mathbf{L}\mathbf{Y}]$, where $\text{Tr}[\cdot]$ denotes the trace of a matrix and $\mathbf{L}$ is the graph Laplacian, subject to the constraint $\mathbf{Y}^T\mathbf{Y} = \mathbf{I}$. By relaxing the requirement that $\mathbf{Y}$ is a binary matrix, we can use the Lagrange multiplier method to solve the optimization problem.

$$\begin{aligned}
\text{Lagrangian, } \mathcal{L} &= \text{Tr}[\mathbf{Y}^T\mathbf{L}\mathbf{Y}] - \lambda(\text{Tr}[\mathbf{Y}^T\mathbf{Y} - \mathbf{I}]) \\
\frac{\partial \mathcal{L}}{\partial \mathbf{Y}} &= \mathbf{L}\mathbf{Y} - \lambda\mathbf{Y} = 0 \\
\Longrightarrow \mathbf{L}\mathbf{Y} &= \lambda\mathbf{Y}
\end{aligned}$$

In other words, an approximate solution to the ratio cut minimization problem can be obtained by finding the eigenvectors of the graph Laplacian matrix, which is exactly the approach used by spectral clustering.

Strengths and Limitations

As shown in Example 8.12, the strength of spectral clustering lies in its ability to detect clusters of varying sizes and shapes. However, the clustering performance depends on how the similarity graph is created and sparsified. In particular, tuning the parameters of the similarity function (e.g., Gaussian radial basis function) to produce an appropriate sparse graph for spectral clustering can be quite a challenge. The time complexity of the algorithm depends on how fast the eigenvectors of the graph Laplacian matrix can be computed. Efficient eigensolvers for sparse matrices are available, e.g., those based on Krylov subspace methods, especially when the number of clusters chosen is small. The storage complexity is $O(N^2)$, though it can be significantly reduced using a sparse representation for the graph Laplacian matrix. In many ways, spectral clustering behaves similarly to the K-means clustering algorithm. First, they both require the user to specify the number of clusters as input parameter. Both methods are also susceptible to the presence of outliers, which tend to form their own connected components (clusters). Thus, preprocessing or postprocessing methods will be needed to handle outliers in the data.

8.4.6 Shared Nearest Neighbor Similarity

In some cases, clustering techniques that rely on standard approaches to similarity and density do not produce the desired clustering results. This section examines the reasons for this and introduces an indirect approach to similarity that is based on the following principle:

> If two points are similar to many of the same points, then they are similar to one another, even if a direct measurement of similarity does not indicate this.

We motivate the discussion by first explaining two problems that an SNN version of similarity addresses: low similarity and differences in density.

Problems with Traditional Similarity in High-Dimensional Data

In high-dimensional spaces, it is not unusual for similarity to be low. Consider, for example, a set of documents such as a collection of newspaper articles that come from a variety of sections of the newspaper: Entertainment, Financial, Foreign, Metro, National, and Sports. As explained in Chapter 2, these documents can be viewed as vectors in a high-dimensional space, where each component of the vector (attribute) records the number of times that each word in a vocabulary occurs in a document. The cosine similarity measure is often used to assess the similarity between documents. For this example, which comes from a collection of articles from the *Los Angeles Times*, Table

Table 8.3. Similarity among documents in different sections of a newspaper.

Section	Average Cosine Similarity
Entertainment	0.032
Financial	0.030
Foreign	0.030
Metro	0.021
National	0.027
Sports	0.036
All Sections	0.014

8.3 gives the average cosine similarity in each section and among the entire set of documents.

The similarity of each document to its most similar document (the first nearest neighbor) is better, 0.39 on average. However, a consequence of low similarity among objects of the same class is that their nearest neighbor is often not of the same class. In the collection of documents from which Table 8.3 was generated, about 20% of the documents have a nearest neighbor of a different class. In general, if direct similarity is low, then it becomes an unreliable guide for clustering objects, especially for agglomerative hierarchical clustering, where the closest points are put together and cannot be separated afterward. Nonetheless, it is still usually the case that a large majority of the nearest neighbors of an object belong to the same class; this fact can be used to define a proximity measure that is more suitable for clustering.

Problems with Differences in Density

Another problem relates to differences in densities between clusters. Figure 8.26 shows a pair of two-dimensional clusters of points with differing density. The lower density of the rightmost cluster is reflected in a lower average distance among the points. Even though the points in the less dense cluster form an equally valid cluster, typical clustering techniques will have more difficulty finding such clusters. Also, normal measures of cohesion, such as SSE, will indicate that these clusters are less cohesive. To illustrate with a real example, the stars in a galaxy are no less real clusters of stellar objects than the planets in a solar system, even though the planets in a solar system are considerably closer to one another on average, than the stars in a galaxy.

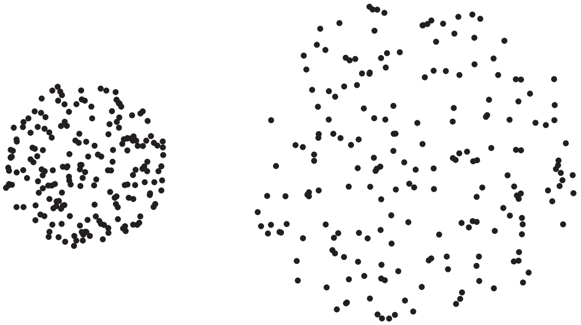


Figure 8.26. Two circular clusters of 200 uniformly distributed points.

SNN Similarity Computation

In both situations, the key idea is to take the context of points into account in defining the similarity measure. This idea can be made quantitative by using a **shared nearest neighbor** definition of similarity in the manner indicated by Algorithm 8.11. Essentially, the SNN similarity is the number of shared neighbors as long as the two objects are on each other's nearest neighbor lists. Note that the underlying proximity measure can be any meaningful similarity or dissimilarity measure.

Algorithm 8.11 Computing shared nearest neighbor similarity

```
Find the $k$-nearest neighbors of all points.
**if** two points, $\mathbf{x}$ and $\mathbf{y}$, are *not* among the $k$-nearest neighbors of each other **then**
    $similarity(\mathbf{x}, \mathbf{y}) \leftarrow 0$
**else**
    $similarity(\mathbf{x}, \mathbf{y}) \leftarrow$ number of shared neighbors
**end if**
```

The computation of SNN similarity is described by Algorithm 8.11 and graphically illustrated by Figure 8.27. Each of the two black points has eight nearest neighbors, including each other. Four of those nearest neighbors—the points in gray—are shared. Thus, the shared nearest neighbor similarity between the two points is 4.

The similarity graph of the SNN similarities among objects is called the **SNN similarity graph**. Because many pairs of objects will have an SNN similarity of 0, this is a very sparse graph.

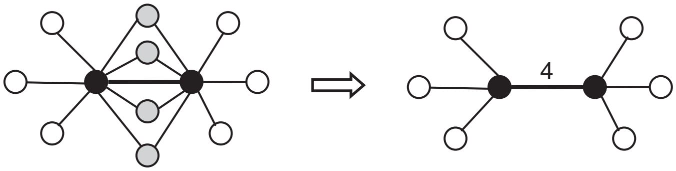


Figure 8.27. Computation of SNN similarity between two points.

SNN Similarity versus Direct Similarity

SNN similarity is useful because it addresses some of the problems that occur with direct similarity. First, since it takes into account the context of an object by using the number of shared nearest neighbors, SNN similarity handles the situation in which an object happens to be relatively close to another object, but belongs to a different class. In such cases, the objects typically do not share many near neighbors and their SNN similarity is low.

SNN similarity also addresses problems with clusters of varying density. In a low-density region, the objects are farther apart than objects in denser regions. However, the SNN similarity of a pair of points only depends on the number of nearest neighbors two objects share, not how far these neighbors are from each object. Thus, SNN similarity performs an automatic scaling with respect to the density of the points.

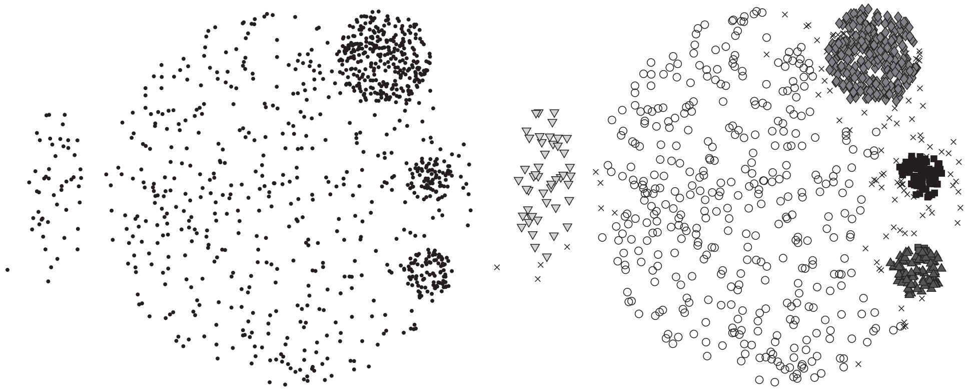
8.4.7 The Jarvis-Patrick Clustering Algorithm

Algorithm 8.12 expresses the Jarvis-Patrick clustering algorithm using the concepts of the last section. The JP clustering algorithm replaces the proximity between two points with the SNN similarity, which is calculated as described in Algorithm 8.11. A threshold is then used to sparsify this matrix of SNN similarities. In graph terms, an SNN similarity graph is created and sparsified. Clusters are simply the connected components of the SNN graph.

Algorithm 8.12 Jarvis-Patrick clustering algorithm.

```
1: Compute the SNN similarity graph.
2: Sparsify the SNN similarity graph by applying a similarity threshold.
3: Find the connected components (clusters) of the sparsified SNN similarity graph.
```

The storage requirements of the JP clustering algorithm are only $O(km)$, because it is not necessary to store the entire similarity matrix, even initially. The basic time complexity of JP clustering is $O(m^2)$, since the creation of



(a) Original data.

(b) Clusters found by Jarvis-Patrick.

Figure 8.28. Jarvis-Patrick clustering of a two-dimensional point set.

the k-nearest neighbor list can require the computation of $O(m^2)$ proximities. However, for certain types of data, such as low-dimensional Euclidean data, special techniques, e.g., a k-d tree, can be used to more efficiently find the k-nearest neighbors without computing the entire similarity matrix. This can reduce the time complexity from $O(m^2)$ to $O(m \log m)$.

Example 8.13 (JP Clustering of a Two-Dimensional Data Set)**.** We applied JP clustering to the "fish" data set shown in Figure 8.28(a) to find the clusters shown in Figure 8.28(b). The size of the nearest neighbor list was 20, and two points were placed in the same cluster if they shared at least 10 points. The different clusters are shown by the different markers and different shading. The points whose marker is an "x" were classified as noise by Jarvis-Patrick. They are mostly in the transition regions between clusters of different density.

■

Strengths and Limitations

Because JP clustering is based on the notion of SNN similarity, it is good at dealing with noise and outliers and can handle clusters of different sizes, shapes, and densities. The algorithm works well for high-dimensional data and is particularly good at finding tight clusters of strongly related objects.

However, JP clustering defines a cluster as a connected component in the SNN similarity graph. Thus, whether a set of objects is split into two clusters or left as one can depend on a single link. Hence, JP clustering is somewhat brittle; i.e., it can split true clusters or join clusters that should be kept separate.

Another potential limitation is that not all objects are clustered. However, these objects can be added to existing clusters, and in some cases, there is no requirement for a complete clustering. JP clustering has a basic time complexity of $O(m^2)$, which is the time required to compute the nearest neighbor list for a set of objects in the general case. In certain cases, e.g., low-dimensional data, special techniques can be used to reduce the time complexity for finding nearest neighbors to $O(m \log m)$. Finally, as with other clustering algorithms, choosing the best values for the parameters can be challenging.

8.4.8 SNN Density

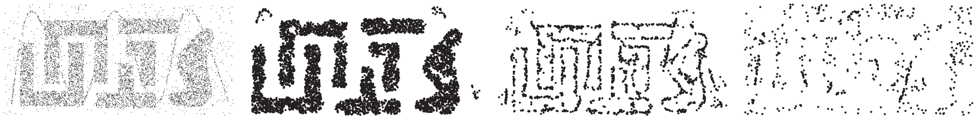
As discussed in the introduction to this chapter, traditional Euclidean density becomes meaningless in high dimensions. This is true whether we take a grid-based view, such as that used by CLIQUE, a center-based view, such as that used by DBSCAN, or a kernel-density estimation approach, such as that used by DENCLUE. It is possible to use the center-based definition of density with a similarity measure that works well for high dimensions, e.g., cosine or Jaccard, but as described in Section 8.4.6, such measures still have problems. However, because the SNN similarity measure reflects the local configuration of the points in the data space, it is relatively insensitive to variations in density and the dimensionality of the space, and is a promising candidate for a new measure of density.

This section explains how to define a concept of SNN density by using SNN similarity and following the DBSCAN approach described in Section 5.4. For clarity, the definitions of that section are repeated, with appropriate modification to account for the fact that we are using SNN similarity.

Core points. A point is a core point if the number of points within a given neighborhood around the point, as determined by SNN similarity and a supplied parameter *Eps* exceeds a certain threshold *MinPts*, which is also a supplied parameter.

Border points. A border point is a point that is not a core point, i.e., there are not enough points in its neighborhood for it to be a core point, but it falls within the neighborhood of a core point.

Noise points. A noise point is any point that is neither a core point nor a border point.



(a) All points. (b) High SNN density. (c) Medium SNN density. (d) Low SNN density.

Figure 8.29. SNN density of two-dimensional points.

SNN density measures the degree to which a point is surrounded by similar points (with respect to nearest neighbors). Thus, points in regions of high and low density will typically have relatively high SNN density, while points in regions where there is a transition from low to high density—points that are between clusters—will tend to have low SNN density. Such an approach is well-suited for data sets in which there are wide variations in density, but clusters of low density are still interesting.

Example 8.14 (Core, Border, and Noise Points)**.** To make the preceding discussion of SNN density more concrete, we provide an example of how SNN density can be used to find core points and remove noise and outliers. There are 10,000 points in the 2D point data set shown in Figure 8.29(a). Figures 8.29(b–d) distinguish between these points based on their SNN density. Figure 8.29(b) shows the points with the highest SNN density, while Figure 8.29(c) shows points of intermediate SNN density, and Figure 8.29(d) shows figures of the lowest SNN density. From these figures, we see that the points that have high density (i.e., high connectivity in the SNN graph) are candidates for being representative or core points since they tend to be located well inside the cluster, while the points that have low connectivity are candidates for being noise points and outliers, as they are mostly in the regions surrounding the clusters. ■

8.4.9 SNN Density-Based Clustering

The SNN density defined above can be combined with the DBSCAN algorithm to create a new clustering algorithm. This algorithm is similar to the JP clustering algorithm in that it starts with the SNN similarity graph. However, instead of using a threshold to sparsify the SNN similarity graph and then taking connected components as clusters, the SNN density-based clustering algorithm simply applies DBSCAN.

The SNN Density-based Clustering Algorithm

The steps of the SNN density-based clustering algorithm are shown in Algorithm 8.13.

Algorithm 8.13 SNN density-based clustering algorithm.

```
1: Compute the SNN similarity graph.
2: Apply DBSCAN with user-specified parameters for $Eps$ and $MinPts$.
```

The algorithm automatically determines the number of clusters in the data. Note that not all the points are clustered. The points that are discarded include noise and outliers, as well as points that are not strongly connected to a group of points. SNN density-based clustering finds clusters in which the points are strongly related to one another. Depending on the application, we might want to discard many of the points. For example, SNN density-based clustering is good for finding topics in groups of documents.

Example 8.15 (SNN Density-based Clustering of Time Series)**.** The SNN density-based clustering algorithm presented in this section is more flexible than Jarvis-Patrick clustering or DBSCAN. Unlike DBSCAN, it can be used for high-dimensional data and situations in which the clusters have different densities. Unlike Jarvis-Patrick, which performs a simple thresholding and then takes the connected components as clusters, SNN density-based clustering uses a less brittle approach that relies on the concepts of SNN density and core points.

To demonstrate the capabilities of SNN density-based clustering on high-dimensional data, we applied it to monthly time series data of atmospheric pressure at various points on the Earth. More specifically, the data consists of the average monthly sea-level pressure (SLP) for a period of 41 years at each point on a 2.5° longitude-latitude grid. The SNN density-based clustering algorithm found the clusters (gray regions) indicated in Figure 8.30. Note that these are clusters of time series of length 492 months, even though they are visualized as two-dimensional regions. The white areas are regions in which the pressure was not as uniform. The clusters near the poles are elongated because of the distortion of mapping a spherical surface to a rectangle.

Using SLP, Earth scientists have defined time series, called **climate indices**, which are useful for capturing the behavior of phenomena involving the Earth's climate. For example, anomalies in climate indices are related to abnormally low or high precipitation or temperature in various parts of the world. Some of the clusters found by SNN density-based clustering have a strong connection to some of the climate indices known to Earth scientists.

Figure 8.31 shows the SNN density structure of the data from which the clusters were extracted. The density has been normalized to be on a scale between 0 and 1. The density of a time series may seem like an unusual concept, but it measures the degree to which the time series and its nearest neighbors have the same nearest neighbors. Because each time series is associated with a location, it is possible to plot these densities on a two-dimensional plot. Because of temporal autocorrelation, these densities form meaningful patterns, e.g., it is possible to visually identify the clusters of Figure 8.31. ■

Strengths and Limitations

The strengths and limitations of SNN density-based clustering are similar to those of JP clustering. However, the use of core points and SNN density adds considerable power and flexibility to this approach.

8.5 Scalable Clustering Algorithms

Even the best clustering algorithm is of little value if it takes an unacceptably long time to execute or requires too much memory. This section examines clustering techniques that place significant emphasis on scalability to the very large data sets that are becoming increasingly common. We start by discussing some general strategies for scalability, including approaches for reducing the number of proximity calculations, sampling the data, partitioning the data, and clustering a summarized representation of the data. We then discuss two specific examples of scalable clustering algorithms: CURE and BIRCH.

8.5.1 Scalability: General Issues and Approaches

The amount of storage required for many clustering algorithms is more than linear; e.g., with hierarchical clustering, memory requirements are usually $O(m^2)$, where m is the number of objects. For 10,000,000 objects, for example, the amount of memory required is proportional to 10^{14}, a number still well beyond the capacities of current systems. Note that because of the requirement for random data access, many clustering algorithms cannot easily be modified to efficiently use secondary storage (disk), for which random data access is slow. Likewise, the amount of computation required for some clustering algorithms is more than linear. In the remainder of this section, we discuss a variety of techniques for reducing the amount of computation and storage

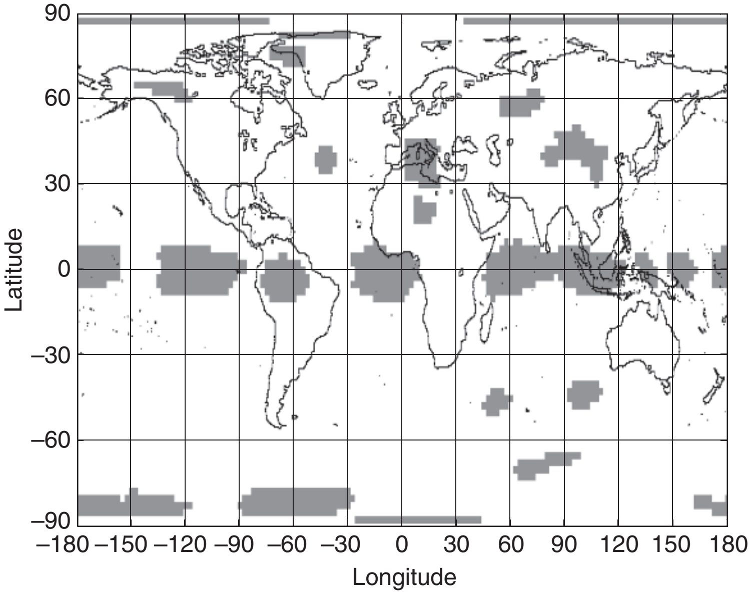


Figure 8.30. Clusters of pressure time series found using SNN density-based clustering.

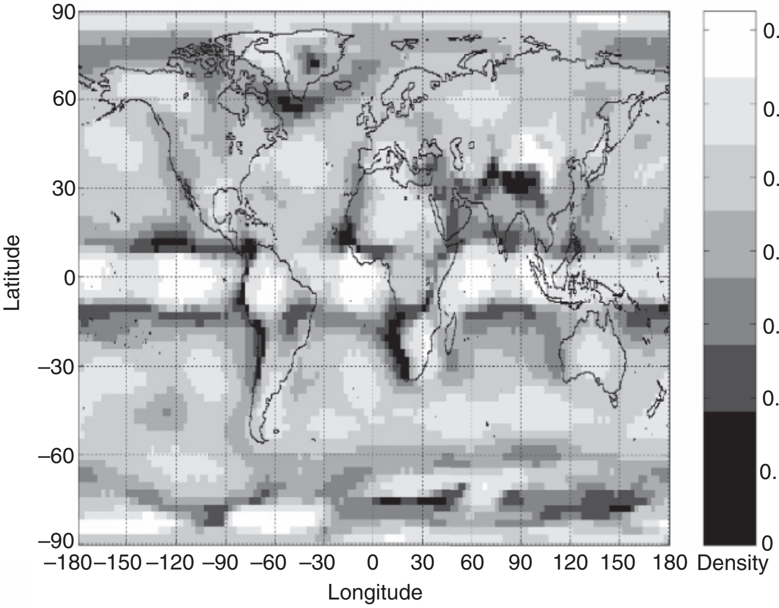


Figure 8.31. SNN density of pressure time series.

required by a clustering algorithm. CURE and BIRCH use some of these techniques.

Multidimensional or Spatial Access Methods Many techniques, such as K-means, Jarvis Patrick clustering, and DBSCAN, need to find the closest centroid, the nearest neighbors of a point, or all points within a specified distance. It is possible to use special techniques called multidimensional or spatial access methods to more efficiently perform these tasks, at least for low-dimensional data. These techniques, such as the k-d tree or R*-tree, typically produce a hierarchical partition of the data space that can be used to reduce the time required to find the nearest neighbors of a point. Note that grid-based clustering schemes also partition the data space.

Bounds on Proximities Another approach to avoiding proximity computations is to use bounds on proximities. For instance, when using Euclidean distance, it is possible to use the triangle inequality to avoid many distance calculations. To illustrate, at each stage of traditional K-means, it is necessary to evaluate whether a point should stay in its current cluster or be moved to a new cluster. If we know the distance between the centroids and the distance of a point to the (newly updated) centroid of the cluster to which it currently belongs, then we might be able to use the triangle inequality to avoid computing the distance of the point to any of the other centroids. See Exercise 27 on page 722.

Sampling Another approach to reducing the time complexity is to sample. In this approach, a sample of points is taken, these points are clustered, and then the remaining points are assigned to the existing clusters—typically to the closest cluster. If the number of points sampled is $\sqrt{m}$, then the time complexity of an $O(m^2)$ algorithm is reduced to $O(m)$. A key problem with sampling, though, is that small clusters can be lost. When we discuss CURE, we will provide a technique for investigating how frequently such problems occur.

Partitioning the Data Objects Another common approach to reducing time complexity is to use some efficient technique to partition the data into disjoint sets and then cluster these sets separately. The final set of clusters either is the union of these separate sets of clusters or is obtained by combining and/or refining the separate sets of clusters. We only discuss bisecting K-means (Section 5.2.3) in this section, although many other approaches based on partitioning are possible. One such approach will be described, when we describe CURE later on in this section.

If K-means is used to find K clusters, then the distance of each point to each cluster centroid is calculated at each iteration. When K is large, this can be very expensive. Bisecting K-means starts with the entire set of points and uses K-means to repeatedly bisect an existing cluster until we have obtained K clusters. At each step, the distance of points to two cluster centroids is computed. Except for the first step, in which the cluster being bisected consists of all the points, we only compute the distance of a subset of points to the two centroids being considered. Because of this fact, bisecting K-means can run significantly faster than regular K-means.

Summarization Another approach to clustering is to summarize the data, typically in a single pass, and then cluster the summarized data. In particular, the leader algorithm (see Exercise 12 in Chapter 5, page 387) either puts a data object in the closest cluster (if that cluster is sufficiently close) or starts a new cluster that contains the current object. This algorithm is linear in the number of objects and can be used to summarize the data so that other clustering techniques can be used. The BIRCH algorithm uses a similar concept.

Parallel and Distributed Computation If it is not possible to take advantage of the techniques described earlier, or if these approaches do not yield the desired accuracy or reduction in computation time, then other approaches are needed. A highly effective approach is to distribute the computation among multiple processors.

8.5.2 BIRCH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) 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 one pass and can improve that clustering with additional passes. BIRCH 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 that can be updated incrementally and that can be used to compute a number of important quantities, such as the centroid of a cluster and its variance (standard deviation). The variance is used as a measure of the diameter of a cluster.

These quantities can also be used to compute the distance between clusters. The simplest approach is to calculate an L_1 (city block) or L_2 (Euclidean) distance between centroids. We can also use the diameter (variance) of the merged cluster as a distance. A number of different distance measures for clusters are defined by BIRCH, but all can be computed using the summary statistics.

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

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, together with the page size, determines the number of entries in a leaf.

By adjusting the 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 has 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 a 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 reabsorbed. If not, they are deleted.

BIRCH consists of a number of phases beyond the initial creation of the CF tree. All the phases of BIRCH are described briefly in Algorithm 8.14.

Algorithm 8.14 BIRCH.

```
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 that
   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 a 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
   locally optimal solution.
```

8.5.3 CURE

CURE (Clustering Using REpresentatives) is a clustering algorithm that uses a variety of different techniques to create an approach that 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 farthest 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, 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 $\alpha = 0.7$), while a representative point at a distance of 1 unit would only move 0.3 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 it may consist 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.

Because the worst-case complexity of CURE is $O(m^2 \log m)$, it cannot be applied directly to large data sets. For this reason, CURE uses two techniques to speed 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 the closest representative point. We discuss CURE's sampling approach in more detail later.

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 and then clusters the points in each partition. This preclustering step is then followed by a clustering of the intermediate clusters and a final pass that assigns each point in the data set to one of the clusters. CURE's partitioning scheme is also discussed in more detail later.

Algorithm 8.15 summarizes CURE. Note that K is the desired number of clusters, m is the number of points, p is the number of partitions, and q is the desired reduction of points in a partition, i.e., the number of clusters in a partition is $\frac{m}{pq}$. Therefore, the total number of clusters is $\frac{m}{q}$. For example, if $m = 10{,}000$, $p = 10$, and $q = 100$, then each partition contains $10{,}000/10 = 1000$ points, and there would be $1000/100 = 10$ clusters in each partition and $10{,}000/100 = 100$ clusters overall.

Algorithm 8.15 CURE.

```
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 partition into $\frac{m}{pq}$ clusters using CURE's
   hierarchical clustering algorithm to obtain a total of $\frac{m}{q}$ clusters. Note
   that some outlier elimination occurs during this process.
4: Use CURE's hierarchical clustering algorithm to cluster the $\frac{m}{q}$ clusters
   found in the previous step until only $K$ clusters remain.
5: Eliminate outliers. This is the second phase of outlier elimination.
6: Assign all remaining data points to the nearest cluster to obtain a
   complete clustering.
```

Sampling in CURE

A key issue in using sampling is whether the sample is representative, that is, whether it captures the characteristics of interest. For clustering, the issue is whether we can find the same clusters in the sample as in the entire set of objects. Ideally, we would like the sample to contain some objects for each cluster and for there to be a separate cluster in the sample for those objects that belong to separate clusters in the entire data set.

A more concrete and attainable goal is to guarantee (with a high probability) that we have at least some points from each cluster. The number of points required for such a sample varies from one data set to another and depends on the number of objects and the sizes of the clusters. The creators of CURE derived a bound for the sample size that would be needed to ensure (with high probability) that we obtain at least a certain number of points from a cluster. Using the notation of this book, this bound is given by the following theorem. **Theorem 8.1.** *Let f be a fraction, $0 \leq f \leq 1$. For cluster C_i of size m_i, we will obtain at least $f * m_i$ objects from cluster C_i with a probability of $1 - \delta, 0 \leq \delta \leq 1$, if our sample size s is given by the following:*

$$s = fm + \frac{m}{m_i} * \log\frac{1}{\delta} + \frac{m}{m_i}\sqrt{\log\frac{1}{\delta}^2 + 2 * f * m_i * \log\frac{1}{\delta}}. \tag{8.21}$$

where m is the number of objects.

While this expression might look intimidating, it is reasonably easy to use. Suppose that there are 100,000 objects and that the goal is to have an 80% chance of obtaining 10% of the objects in cluster C_i, which has a size of 1000. In this case, $f = 0.1$, $\delta = 0.2$, $m =$100,000, $m_i = 1000$, and thus $s =$11,962. If the goal is a 5% sample of C_i, which is 50 objects, then a sample size of 6440 will suffice.

Again, CURE uses sampling in the following way. First a sample is drawn, and then CURE is used to cluster this sample. After clusters have been found, each unclustered point is assigned to the closest cluster.

Partitioning

When sampling is not enough, CURE also uses a partitioning approach. The idea is to divide the points into p groups of size m/p and to use CURE to cluster each partition in order to reduce the number of objects by a factor of $q > 1$, where q can be roughly thought of as the average size of a cluster in a partition. Overall, $\frac{m}{q}$ clusters are produced. (Note that since CURE represents each cluster by a number of representative points, the reduction in the number of objects is not q.) This preclustering step is then followed by a final clustering of the m/q intermediate clusters to produce the desired number of clusters (K). Both clustering passes use 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.

The key issue is how p and q should be chosen. Algorithms such as CURE have a time complexity of $O(m^2)$ or higher, and furthermore, require that all the data be in main memory. We therefore want to choose p small enough so that an entire partition can be processed in main memory and in a 'reasonable' amount of time. At the current time, a typical desktop computer can perform a hierarchical clustering of a few thousand objects in a few seconds.

Another factor for choosing p, and also q, concerns the quality of the clustering. Specifically, the objective is to choose the values of p and q such that objects from the same underlying cluster end up in the same clusters eventually. To illustrate, suppose there are 1000 objects and a cluster of size 100. If we randomly generate 100 partitions, then each partition will, on average, have only one point from our cluster. These points will likely be put in clusters with points from other clusters or will be discarded as outliers. If we generate only 10 partitions of 100 objects, but q is 50, then the 10 points from each cluster (on average) will likely still be combined with points from other clusters, because there are only (on average) 10 points per cluster and we need to produce, for each partition, two clusters. To avoid this last problem, which concerns the proper choice of q, a suggested strategy is not to combine clusters if they are too dissimilar.

8.6 Which Clustering Algorithm?

A variety of factors need to be considered when deciding which type of clustering technique to use. Many, if not all, of these factors have been discussed to some extent in the current and previous chapters. Our goal in this section is to succinctly summarize these factors in a way that sheds some light on which clustering algorithm might be appropriate for a particular clustering task.

Type of Clustering For a clustering algorithm to be appropriate for a task, the type of clustering produced by the algorithm needs to match the type of clustering needed by the application. For some applications, such as creating a biological taxonomy, a hierarchy is preferred. In the case of clustering for summarization, a partitional clustering is typical. In yet other applications, both can prove useful.

Most clustering applications require a clustering of all (or almost all) of the objects. For instance, if clustering is used to organize a set of documents for browsing, then we would like most documents to belong to a group. However, if we wanted to find the strongest themes in a set of documents, then we might prefer to have a clustering scheme that produces only very cohesive clusters, even if many documents were left unclustered.

Finally, most applications of clustering assume that each object is assigned to one cluster (or one cluster on a level for hierarchical schemes). As we have seen, however, probabilistic and fuzzy schemes provide weights that indicate the degree or probability of membership in various clusters. Other techniques, such as DBSCAN and SNN density-based clustering, have the notion of core points, which strongly belong to one cluster. Such concepts may be useful in certain applications.

Type of Cluster Another key aspect is whether the type of cluster matches the intended application. There are three commonly encountered types of clusters: prototype-, graph-, and density-based. Prototype-based clustering schemes, as well as some graph-based clustering schemes—complete link, centroid, and Ward's—tend to produce globular clusters in which each object is close to the cluster's prototype and/or to the other objects in the cluster. If, for example, we want to summarize the data to reduce its size and we want to do so with the minimum amount of error, then one of these types of techniques would be most appropriate. In contrast, density-based clustering techniques, as well as some graph-based clustering techniques, such as single link, tend to produce clusters that are not globular and thus contain many objects that are not very similar to one another. If clustering is used to segment a geographical area into contiguous regions based on the type of land cover, then one of these techniques is more suitable than a prototype-based scheme such as K-means.

Characteristics of Clusters Besides the general type of cluster, other cluster characteristics are important. If we want to find clusters in subspaces of the original data space, then we must choose an algorithm such as CLIQUE, which explicitly looks for such clusters. Similarly, if we are interested in enforcing spatial relationships between clusters, then SOM or some related approach would be appropriate. Also, clustering algorithms differ widely in their ability to handle clusters of varying shapes, sizes, and densities.

Characteristics of the Data Sets and Attributes As discussed in the introduction, the type of data set and attributes can dictate the type of algorithm to use. For instance, the K-means algorithm can only be used on data for which an appropriate proximity measure is available that allows meaningful computation of a cluster centroid. For other clustering techniques, such as many agglomerative hierarchical approaches, the underlying nature of the data sets and attributes is less important as long as a proximity matrix can be created.

Noise and Outliers Noise and outliers are particularly important aspects of the data. We have tried to indicate the effect of noise and outliers on the various clustering algorithms that we have discussed. In practice, however, it can be difficult to evaluate the amount of noise in the data set or the number of outliers. More than that, what is noise or an outlier to one person might be interesting to another person. For example, if we are using clustering to segment an area into regions of different population density, we do not want to use a density-based clustering technique, such as DBSCAN, that assumes that regions or points with density lower than a global threshold are noise or outliers. As another example, hierarchical clustering schemes, such as CURE, often discard clusters of points that are growing slowly as such groups tend to represent outliers. However, in some applications we are most interested in relatively small clusters; e.g., in market segmentation, such groups might represent the most profitable customers.

Number of Data Objects We have considered how clustering is affected by the number of data objects in considerable detail in previous sections. We reiterate, however, that this factor often plays an important role in determining the type of clustering algorithm to be used. Suppose that we want to create a hierarchical clustering of a set of data, we are not interested in a complete hierarchy that extends all the way to individual objects, but only to the point at which we have split the data into a few hundred clusters. If the data is very large, we cannot directly apply an agglomerative hierarchical clustering technique. We could, however, use a divisive clustering technique, such as the minimum spanning tree (MST) algorithm, which is the divisive analog to single link, but this would only work if the data set is not too large. Bisecting K-means would also work for many data sets, but if the data set is large enough that it cannot be contained completely in memory, then this scheme also runs into problems. In this situation, a technique such as BIRCH, which does not require that all data be in main memory, becomes more useful.

Number of Attributes We have also discussed the impact of dimensionality at some length. Again, the key point is to realize that an algorithm that works well in low or moderate dimensions may not work well in high dimensions. As in many other cases in which a clustering algorithm is inappropriately applied, the clustering algorithm will run and produce clusters, but the clusters will likely not represent the true structure of the data.

Cluster Description One aspect of clustering techniques that is often overlooked is how the resulting clusters are described. Prototype clusters are succinctly described by a small set of cluster prototypes. In the case of mixture models, the clusters are described in terms of small sets of parameters, such as the mean vector and the covariance matrix. This is also a very compact and understandable representation. For SOM, it is typically possible to visualize the relationships between clusters in a two-dimensional plot, such as that of Figure 8.8. For graph- and density-based clustering approaches, however, clusters are typically described as sets of cluster members. Nonetheless, in CURE, clusters can be described by a (relatively) small set of representative points. Also, for grid-based clustering schemes, such as CLIQUE, more compact descriptions can be generated in terms of conditions on the attribute values that describe the grid cells in the cluster.

Algorithmic Considerations There are also important aspects of algorithms that need to be considered. Is the algorithm non-deterministic or order-dependent? Does the algorithm automatically determine the number of clusters? Is there a technique for determining the values of various parameters? Many clustering algorithms try to solve the clustering problem by trying to optimize an objective function. Is the objective a good match for the application objective? If not, then even if the algorithm does a good job of finding a clustering that is optimal or close to optimal with respect to the objective function, the result is not meaningful. Also, most objective functions give preference to larger clusters at the expense of smaller clusters.

Summary The task of choosing the proper clustering algorithm involves considering all of these issues, and domain-specific issues as well. There is no formula for determining the proper technique. Nonetheless, a general knowledge of the types of clustering techniques that are available and consideration of the issues mentioned above, together with a focus on the intended application, should allow a data analyst to make an informed decision on which clustering approach (or approaches) to try.

8.7 Bibliographic Notes

An extensive discussion of fuzzy clustering, including a description of fuzzy c-means and formal derivations of the formulas presented in Section 8.2.1, can be found in the book on fuzzy cluster analysis by Höppner et al. [595]. While not discussed in this chapter, AutoClass by Cheeseman et al. [573] is one of the earliest and most prominent mixture-model clustering programs. An introduction to mixture models can be found in the tutorial of Bilmes [568], the book by Mitchell [606] (which also describes how the K-means algorithm can be derived from a mixture model approach), and the article by Fraley and Raftery [581]. Mixture model is an example of a probabilistic clustering method, in which the clusters are represented as hidden variables in the model. More sophisticated probabilistic clustering methods such as latent Dirichlet allocation (LDA) [570] have been developed in recent years for domains such as text clustering.

Besides data exploration, SOM and its supervised learning variant, Learning Vector Quantization (LVQ), have been used for many tasks: image segmentation, organization of document files, and speech processing. Our discussion of SOM was cast in the terminology of prototype-based clustering. The book on SOM by Kohonen et al. [601] contains an extensive introduction to SOM that emphasizes its neural network origins, as well as a discussion of some of its variations and applications. One important SOM-related clustering development is the Generative Topographic Map (GTM) algorithm by Bishop et al. [569], which uses the EM algorithm to find Gaussian models satisfying two-dimensional topographic constraints.

The description of Chameleon can be found in the paper by Karypis et al. [599]. Capabilities similar, although not identical to those of Chameleon have been implemented in the CLUTO clustering package by Karypis [575]. The METIS graph partitioning package by Karypis and Kumar [600] is used to perform graph partitioning in both programs, as well as in the OPOSSUM clustering algorithm by Strehl and Ghosh [616]. A detailed discussion on spectral clustering can be found in the tutorial by von Luxburg [618]. The spectral clustering method described in this chapter is based on an unnormalized graph Laplacian matrix and the ratio cut measure [590]. Alternative formulations of spectral clustering have been developed using normalized graph Laplacian matrices for other evaluation measures [613].

The notion of SNN similarity was introduced by Jarvis and Patrick [596]. A hierarchical clustering scheme based on a similar concept of mutual nearest neighbors was proposed by Gowda and Krishna [586]. Guha et al. [589] created ROCK, a hierarchical graph-based clustering algorithm for clustering transaction data, which among other interesting features, also uses a notion of similarity based on shared neighbors that closely resembles the SNN similarity developed by Jarvis and Patrick. A description of the SNN density-based clustering technique can be found in the publications of Ertöz et al. [578, 579]. SNN density-based clustering was used by Steinbach et al. [614] to find climate indices.

Examples of grid-based clustering algorithms are OptiGrid (Hinneburg and Keim [594]), the BANG clustering system (Schikuta and Erhart [611]), and WaveCluster (Sheikholeslami et al. [612]). The CLIQUE algorithm is described in the paper by Agrawal et al. [564]. MAFIA (Nagesh et al. [608]) is a modification of CLIQUE whose goal is improved efficiency. Kailing et al. [598] have developed SUBCLU (density-connected SUBspace CLUstering), a subspace clustering algorithm based on DBSCAN. The DENCLUE algorithm was proposed by Hinneburg and Keim [593].

Our discussion of scalability was strongly influenced by the article of Ghosh [584]. A wide-ranging discussion of specific techniques for clustering massive data sets can be found in the paper by Murtagh [607]. CURE is work by Guha et al. [588], while details of BIRCH are in the paper by Zhang et al. [620]. CLARANS (Ng and Han [609]) is an algorithm for scaling K-medoid clustering to larger databases. A discussion of scaling EM and K-means clustering to large data sets is provided by Bradley et al. [571, 572]. A parallel implementation of K-means on the MapReduce framework has also been developed [621]. In addition to K-means, other clustering algorithms that have been implemented on the MapReduce framework include DBScan [592], spectral clustering [574], and hierarchical clustering [617].

In addition to the approaches described in this chapter, there are many other clustering methods proposed in the literature. One class of methods that has become increasingly popular in recent years is based on non-negative matrix factorization (NMF) [602]. The idea is an extension of the singular value decomposition (SVD) approach described in Chapter 2, in which the data matrix is decomposed into a product of lower-rank matrices that represent the underlying components or clusters in the data. In NMF, additional constraints are imposed to ensure non-negativity in the elements of the component matrices. With different formulations and constraints, the NMF method can be shown to be equivalent to other clustering approaches, including K-means and spectral clustering [577, 603]. Another popular class of methods utilizes the constraints provided by users to guide the clustering algorithm. Such algorithms are commonly known as constrained clustering or semi-supervised clustering [566, 567, 576, 619].

There are many aspects of clustering that we have not covered. Additional pointers are given in the books and surveys mentioned in the Bibliographic Notes of Chapter 5. Here, we mention four areas—omitting, unfortunately, many more. Clustering of transaction data (Ganti et al. [582], Gibson et al. [585], Han et al. [591], and Peters and Zaki [610]) is an important area, as transaction data is common and of commercial importance. Streaming data is also becoming increasingly common and important as communications and sensor networks become pervasive. Two introductions to clustering for data streams are given in articles by Barbará [565] and Guha et al. [587]. Conceptual clustering (Fisher and Langley [580], Jonyer et al. [597], Mishra et al. [605], Michalski and Stepp [604], Stepp and Michalski [615]), which uses more complicated definitions of clusters that often correspond better to human notions of a cluster, is an area of clustering whose potential has perhaps not been fully realized. Finally, there has been a great deal of clustering work for data compression in the area of vector quantization. The book by Gersho and Gray [583] is a standard text in this area.

Bibliography

[564] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 94–105, Seattle, Washington, June 1998. ACM Press.

[565] D. Barbará. Requirements for clustering data streams. *SIGKDD Explorations Newsletter*, 3(2):23–27, 2002.

[566] S. Basu, A. Banerjee, and R. Mooney. Semi-supervised clustering by seeding. In *Proceedings of 19th International Conference on Machine Learning*, pages 19–26, 2002.

[567] S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms, Theory, and Applications.* CRC Press, 2008.

[568] J. Bilmes. A Gentle Tutorial on the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical Report ICSI-TR-97-021, University of California at Berkeley, 1997.

[569] C. M. Bishop, M. Svensen, and C. K. I. Williams. GTM: A principled alternative to the self-organizing map. In C. von der Malsburg, W. von Seelen, J. C. Vorbruggen, and B. Sendhoff, editors, *Artificial Neural Networks—ICANN96. Intl. Conf, Proc.*, pages 165–170. Springer-Verlag, Berlin, Germany, 1996.

[570] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet Allocation. *Journal of Machine Learning Research*, 3(4-5):993–1022, 2003.

[571] P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling Clustering Algorithms to Large Databases. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 9–15, New York City, August 1998. AAAI Press.

[572] P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling EM (Expectation Maximization) Clustering to Large Databases. Technical Report MSR-TR-98-35, Microsoft Research, October 1999.

[573] P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. AutoClass: a Bayesian classification system. In *Readings in knowledge acquisition and learning: automating the construction and improvement of expert systems*, pages 431–441. Morgan Kaufmann Publishers Inc., 1993.

[574] W. Y. Chen, Y. Song, H. Bai, C. J. Lin, and E. Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568586, 2011.

[575] CLUTO 2.1.2: Software for Clustering High-Dimensional Datasets. www.cs.umn.edu/∼karypis, October 2016.

[576] I. Davidson and S. Basu. A survey of clustering with instance level constraints. *ACM Transactions on Knowledge Discovery from Data*, 1:1–41, 2007.

[577] C. Ding, X. He, and H. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proc of the SIAM International Conference on Data Mining*, page 606610, 2005.

[578] L. Ertöz, M. Steinbach, and V. Kumar. A New Shared Nearest Neighbor Clustering Algorithm and its Applications. In *Workshop on Clustering High Dimensional Data and its Applications, Proc. of Text Mine'01, First SIAM Intl. Conf. on Data Mining, Chicago, IL, USA*, 2001.

[579] L. Ertöz, M. Steinbach, and V. Kumar. Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data. In *Proc. of the 2003 SIAM Intl. Conf. on Data Mining*, San Francisco, May 2003. SIAM.

[580] D. Fisher and P. Langley. Conceptual clustering and its relation to numerical taxonomy. *Artificial Intelligence and Statistics*, pages 77–116, 1986.

[581] C. Fraley and A. E. Raftery. How Many Clusters? Which Clustering Method? Answers Via Model-Based Cluster Analysis. *The Computer Journal*, 41(8):578–588, 1998.

[582] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS–Clustering Categorical Data Using Summaries. In *Proc. of the 5th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 73–83. ACM Press, 1999.

[583] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*, volume 159 of *Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, 1992.

[584] J. Ghosh. Scalable Clustering Methods for Data Mining. In N. Ye, editor, *Handbook of Data Mining*, pages 247–277. Lawrence Ealbaum Assoc, 2003.

[585] D. Gibson, J. M. Kleinberg, and P. Raghavan. Clustering Categorical Data: An Approach Based on Dynamical Systems. *VLDB Journal*, 8(3–4):222–236, 2000.

[586] K. C. Gowda and G. Krishna. Agglomerative Clustering Using the Concept of Mutual Nearest Neighborhood. *Pattern Recognition*, 10(2):105–112, 1978.

[587] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams: Theory and Practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, May/June 2003.

[588] S. Guha, R. Rastogi, and K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proc. of 1998 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 73–84. ACM Press, June 1998.

[589] S. Guha, R. Rastogi, and K. Shim. ROCK: A Robust Clustering Algorithm for Categorical Attributes. In *Proc. of the 15th Intl. Conf. on Data Engineering*, pages 512–521. IEEE Computer Society, March 1999.

[590] L. Hagen and A. Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. Computer-Aided Design*, 11(9):1074 1085, 1992.

[591] E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher. Hypergraph Based Clustering in High-Dimensional Data Sets: A Summary of Results. *IEEE Data Eng. Bulletin*, 21 (1):15–22, 1998.

[592] Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan. MR-DBSCAN: an efficient parallel density-based clustering algorithm using MapReduce. In *Proc of the IEEE International Conference on Parallel and Distributed Systems*, pages 473–480, 2011.

[593] A. Hinneburg and D. A. Keim. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. In *Proc. of the 4th Intl. Conf. on Knowledge Discovery and Data Mining*, pages 58–65, New York City, August 1998. AAAI Press.

[594] A. Hinneburg and D. A. Keim. Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering. In *Proc. of the 25th VLDB Conf.*, pages 506–517, Edinburgh, Scotland, UK, September 1999. Morgan Kaufmann.

[595] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler. *Fuzzy Cluster Analysis: Methods for Classification, Data Analysis and Image Recognition*. John Wiley & Sons, New York, July 2 1999.

[596] R. A. Jarvis and E. A. Patrick. Clustering Using a Similarity Measure Based on Shared Nearest Neighbors. *IEEE Transactions on Computers*, C-22(11):1025–1034, 1973.

[597] I. Jonyer, D. J. Cook, and L. B. Holder. Graph-based hierarchical conceptual clustering. *Journal of Machine Learning Research*, 2:19–43, 2002.

[598] K. Kailing, H.-P. Kriegel, and P. Kröger. Density-Connected Subspace Clustering for High-Dimensional Data. In *Proc. of the 2004 SIAM Intl. Conf. on Data Mining*, pages 428–439, Lake Buena Vista, Florida, April 2004. SIAM.

[599] G. Karypis, E.-H. Han, and V. Kumar. CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling. *IEEE Computer*, 32(8):68–75, August 1999.

[600] G. Karypis and V. Kumar. Multilevel k-way Partitioning Scheme for Irregular Graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[601] T. Kohonen, T. S. Huang, and M. R. Schroeder. *Self-Organizing Maps*. Springer-Verlag, December 2000.

[602] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788791, 1999.

[603] T. Li and C. H. Q. Ding. The Relationships Among Various Nonnegative Matrix Factorization Methods for Clustering. In *Proc of the IEEE International Conference on Data Mining*, pages 362–371, 2006.

[604] R. S. Michalski and R. E. Stepp. Automated Construction of Classifications: Conceptual Clustering Versus Numerical Taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(4):396–409, 1983.

[605] N. Mishra, D. Ron, and R. Swaminathan. A New Conceptual Clustering Framework. *Machine Learning Journal*, 56(1–3):115–151, July/August/September 2004.

[606] T. Mitchell. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.

[607] F. Murtagh. Clustering massive data sets. In J. Abello, P. M. Pardalos, and M. G. C. Reisende, editors, *Handbook of Massive Data Sets*. Kluwer, 2000.

[608] H. Nagesh, S. Goil, and A. Choudhary. Parallel Algorithms for Clustering High-Dimensional Large-Scale Datasets. In R. L. Grossman, C. Kamath, P. Kegelmeyer, V. Kumar, and R. Namburu, editors, *Data Mining for Scientific and Engineering Applications*, pages 335–356. Kluwer Academic Publishers, Dordrecht, Netherlands, October 2001.

[609] R. T. Ng and J. Han. CLARANS: A Method for Clustering Objects for Spatial Data Mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1003–1016, 2002.

[610] M. Peters and M. J. Zaki. CLICKS: Clustering Categorical Data using K-partite Maximal Cliques. In *Proc. of the 21st Intl. Conf. on Data Engineering*, Tokyo, Japan, April 2005.

[611] E. Schikuta and M. Erhart. The BANG-Clustering System: Grid-Based Data Analysis. In *Advances in Intelligent Data Analysis, Reasoning about Data, Second Intl. Symposium, IDA-97, London*, volume 1280 of *Lecture Notes in Computer Science*, pages 513–524. Springer, August 1997.

[612] G. Sheikholeslami, S. Chatterjee, and A. Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *Proc. of the 24th VLDB Conf.*, pages 428–439, New York City, August 1998. Morgan Kaufmann.

[613] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888 905, 2000.

[614] M. Steinbach, P.-N. Tan, V. Kumar, S. Klooster, and C. Potter. Discovery of climate indices using clustering. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 446–455, New York, NY, USA, 2003. ACM Press.

[615] R. E. Stepp and R. S. Michalski. Conceptual clustering of structured objects: A goal-oriented approach. *Artificial Intelligence*, 28(1):43–69, 1986.

[616] A. Strehl and J. Ghosh. A Scalable Approach to Balanced, High-dimensional Clustering of Market-Baskets. In *Proc. of the 7th Intl. Conf. on High Performance Computing (HiPC 2000)*, volume 1970 of *Lecture Notes in Computer Science*, pages 525–536, Bangalore, India, December 2000. Springer.

[617] T. Sun, C. Shu, F. Li, H. Yu, L. Ma, and Y. Fang. An efficient hierarchical clustering method for large datasets with map-reduce. In *Proc of the IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 494–499, 2009.

[618] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4): 395–416, 2007.

[619] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained K-means Clustering with Background Knowledge. In *Proceedings of 18th International Conference on Machine Learning*, pages 577–584, 2001.

[620] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. of 1996 ACM-SIGMOD Intl. Conf. on Management of Data*, pages 103–114, Montreal, Quebec, Canada, June 1996. ACM Press.

[621] W. Zhao, H. Ma, and Q. He. Parallel K-Means Clustering based on MapReduce. In *Proc of the IEEE International Conference on Cloud Computing*, page 674679, 2009.

8.8 Exercises

1. For sparse data, discuss why considering only the presence of non-zero values might give a more accurate view of the objects than considering the actual magnitudes of values. When would such an approach not be desirable?

2. What are the general strengths and weaknesses of density-based clustering algorithms?

3. Describe the change in the time complexity of fuzzy c-means as the number of clusters to be found increases.

4. Describe the change in the time complexity of K-means as the number of clusters to be found increases.

5. Consider a set of documents. Assume that all documents have been normalized to have unit length of 1. What is the "shape" of a cluster that consists of all documents whose cosine similarity to a centroid is greater than some specified constant? In other words, $\cos(d, c) \geq \delta$, where $0 < \delta \leq 1$.

6. Discuss the advantages and disadvantages of treating clustering as an optimization problem. Among other factors, consider efficiency, non-determinism, and whether an optimization-based approach captures all types of clusterings that are of interest.

7. Which of the following can handle (i) noise and outliers and (ii) clusters of different sizes and densities?