9

Unsupervised Learning: Clustering

In this Chapter, we introduce the concept of clustering, present the basic terminology, offer a commonly encountered taxonomy of clustering algorithms, and discuss in detail some representative algorithms. We cover essential issues of scalable clustering that offers an insight into the issues of handling large data sets.

1. From Data to Information Granules or Clusters

Making sense of data has been an ongoing quest within various types of research communities in almost every practical endeavor that deals with collected experimental evidence. The age of information technology, whose eminent manifestation is a vast amount of data, has amplified this quest and made it even more challenging. The collections of large quantities of data *anytime* and *anywhere* have become the predominant reality of our lives.

Within this context, clustering arises as a remarkably rich conceptual and algorithmic framework for data analysis and interpretation. In a nutshell, clustering is about **abstraction**–discovering structure in collections of data. The task of clustering is challenging both conceptually and computationally. As explained in Chapter 4, the term clustering is often used as a synonym for **unsupervised learning** (but we need to remember that another key unsupervised learning technique is association rules). As the name clustering implies, it is anticipated that a suitable, unsupervised algorithm is capable of discovering structure on its own by exploring **similarities** or differences (such as distances) between individual data points in a data set under consideration. This highly intuitive and appealing guideline sounds deceptively simple: cluster two data points if they are "close" to each other and keep doing the same by exploring the distances between newly formed clusters and the remaining data points. The number of different strategies for cluster formation is enormous, and a great many approaches try to determine what "similarity" between elements in the data means. Different clustering algorithms address various facets and properties of clusters. Their computational aspects are of paramount importance, and we need to become cognizant of them at the very beginning, in particular with reference to scalability issues. Let us stress that dividing N data (patterns) into c clusters (groups) gives rise to a huge number of possible **partitions**, which is expressed in the form of the Stirling number:

$$\frac{1}{c!}\sum_{i=1}^{c}(-1)^{c-i}\binom{c}{i}i^{N} \tag{1}$$

To illustrate the magnitude of the existing possibilities, let us consider $N = 100$ and $c = 5$, which sounds like a fairly limited problem. Even in this case, we end up with over 10^{67} partitions. Obviously, we need to resort to some optimization techniques the ones known as *clustering methods*.

2. Categories of Clustering Algorithms

Clustering techniques can be divided into three main categories:

1. Partition – based clustering, sometimes referred to as objective function-based clustering
2. Hierarchical clustering
3. Model-based (a mixture of probabilities) clustering.

The clustering principles for each of these categories are very different which implies very different style of processing and resulting formats of the results. In **partition based clustering**, we rely on a certain objective function whose minimization is supposed to lead us to the "discovery" of the structure existing in the data set. While the algorithmic setup is quite appealing and convincing (the optimization problem could be well formalized), one is never sure what type of structure to expect and hence what should be the most suitable form of the objective function. Typically, in this category of the methods, we predefine the number of clusters and proceed with the optimization of the objective function. There are some variants in which we also allow for successive splits of the clusters, a process that leads us to a dynamically adjusted number of clusters. The essence of **hierarchical clustering** lies in the successive development of clusters; we begin either with successive splits (starting with a single cluster that is an entire data set) or with individual points treated as initial clusters, which we and keep merging (this process leads us to the concept of agglomerative clustering). The essential feature of hierarchical clustering concerns a suitable choice of a distance function and a means to express the distance between data and patterns. These features, in essence, give rise to a spectrum of various clustering methods (single linkage, complete linkage, etc.). In **model-based clustering**, as the name itself stipulates, we assume a certain probabilistic model of the data and then estimate its parameters. In this case, we refer to a so-called mixture density model where we assume that the data are a result of a mixture of c sources of data. Each of these sources is treated as a potential cluster.

3. Similarity Measures

Although we have discussed the concept of **similarity** (or distance) in Chapter 4, it is instructive to cast these ideas in the setting of clustering. In the case of continuous features (variables) one can use many distance functions as similarity measures (see Table 9.1). Each of these distances comes with its own geometry (such as hyperspheres and hyperboxes). As will become clear later on, the choice of distance function implies some specific geometry of the clusters formed.

In the case of binary variables, we usually do not use distance as a similarity measure. Consider two binary vectors $\mathbf{x}$ and $\mathbf{y}$, that are two strings of binary data:

$$\mathbf{x} = [x_1 x_2 \ldots x_n]^T$$
$$\mathbf{y} = [y_1 y_2 \ldots y_n]^T$$

compare them coordinate-wise and then count the number of occurrences of specific combinations of 0's and 1's:

a) when x_k and y_k are both equal to 1
b) when $x_k = 0$ and $y_k = 1$
c) when $x_k = 1$ and $y_k = 0$
d) when x_k and y_k are both equal to 0

These four combinations of numbers can be organized into a 2×2 co-occurrence matrix to show how the two strings are "close" to each other. Note that since the strings are composed of

Table 9.1. Selected distance functions between patterns **x** and **y**.

Distance function	Formula and comments
Euclidean distance	$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$
Hamming (city block) distance	$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \lvert x_i - y_i \rvert$
Tchebyschev distance	$d(\mathbf{x}, \mathbf{y}) = \max_{i=1,2,\ldots,n} \lvert x_i - y_i \rvert$
Minkowski distance	$d(\mathbf{x}, \mathbf{y}) = \sqrt[p]{\sum_{i=1}^{n} (x_i - y_i)^p}$, $p > 0$
Canberra distance	$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} \frac{\lvert x_i - y_i \rvert}{x_i + y_i}$, x_i and y_i are positive
Angular separation	$d(\mathbf{x}, \mathbf{y}) = \frac{\sum_{i=1}^{n} x_i y_i}{\left[\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2\right]^{1/2}}$ Note: similarity measure expresses the angle between the unit vectors in the direction of **x** and **y**

0's and 1's, we encounter four combinations; these are represented in the tabular format shown below. For instance, the first row and the first column corresponds to the number of times 1s' occur in both strings (equal to a):

	1	0
1	a	b
0	c	d

Evidently, the zero nondiagonal entries of this matrix indicate ideal matching (the highest similarity). Based on these four entries, there are several commonly encountered measures of similarity of binary vectors. The simplest is the matching coefficient defined as

$$\frac{a+d}{a+b+c+d} \tag{2}$$

Another measure of similarity, Russell and Rao's, takes the following form:

$$\frac{a}{a+b+c+d} \tag{3}$$

The Jacard index is more focused since it involves cases in which both inputs assume values equal to 1:

$$\frac{a}{a+b+c} \tag{4}$$

The Czekanowski index is practically the same, but by adding the weight factor of 2 it emphasizes the coincidence of situations when both entries in **x** and **y** assume values equal to 1:

$$\frac{2a}{2a+b+c} \tag{5}$$

For binary data, the Hamming distance could be another viable alternative. It is essential to be aware of the existence of numerous approaches to defining similarity measures used in various application-oriented settings.

When the features assume p discrete values, we can express the level of similarity/matching by counting the number of situations in which the values of the corresponding entries of $\mathbf{x}$ and $\mathbf{y}$ coincide. If this occurs r times, the pertinent measure could be in the following form, where n denotes the dimension of the corresponding vectors:

$$\mathrm{d}(\mathbf{x}, \mathbf{y}) = \frac{n - r}{n} = 1 - \frac{r}{n} \tag{6}$$

4. Hierarchical Clustering

Hierarchical clustering algorithms produce a graphical representation of data. The construction of graphs (these methods reveal structure by considering each individual pattern) is done in two modes: the bottom-up and top-down. In the **bottom-up mode**, also known as the **agglomerative** approach, we treat each pattern as a single-element cluster and then successively merge the closest clusters. At each pass of the algorithm, we merge the two clusters that are the closest. The process repeats until we get to a single data set (cluster) or reach a predefined threshold value. The **top-down approach**, also known as the **divisive** approach, works in the opposite direction. We start with the entire set, treat it as a single cluster, and keep splitting it into smaller clusters. Considering the nature of the top-down and bottom-up processes, these methods are quite often computationally inefficient, except possibly in the case of binary patterns.

The results of hierarchical clustering are represented in the form of a **dendrogram**. A dendrogram is defined as a binary tree with a distinguished root that has all the data items at its leaves. An example of a dendrogram is shown in Figure 9.1, along with distance values guiding the process of successive merging of the clusters. Depending upon the distance value, we produce a sequence of nested clusters.

Dendrograms are visually appealing graphical constructs that help us understand how difficult it is to merge two clusters. The nodes (represented in the form of small dots) located at the bottom of the graph correspond to the patterns/data points $(a, b, c, \ldots)$. While moving up in the graph, we merge the points that are the closest in terms of some assumed similarity function. For instance, the distance between g and h is the smallest, and thus these two are merged. The distance scale shown at the right-hand side of the graph helps us visualize distance between the clusters. Moving upwards, the clusters get larger.

Thus, at any level of the graph (see the dotted line in Figure 9.1), we can explicitly enumerate the content of the clusters. For instance, following the dotted line, we end up with three clusters, that is $\{a\}$, $\{b, c, d, e\}$, and $\{f, g, h\}$. This process implies a simple stopping criterion: given a certain threshold value of the distance, we stop merging the clusters once the distance between

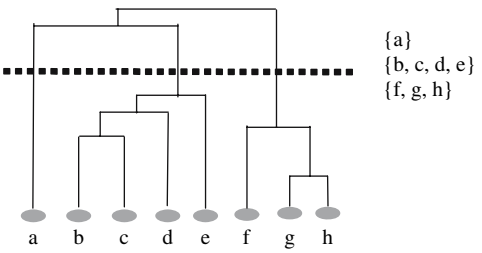

Figure 9.1. A dendrogram as a visualization of structure in the data.

them exceeds the threshold. In other words, merging two quite distinct structures (where their distinctiveness is expressed via the distance value) does not seem to be a good idea.

An important issue arises as to the way in which one can measure distance between two clusters. Note that we have discussed how to express distance between two patterns. Here, since each cluster may contain many patterns, distance computations are not that obvious and certainly not unique. Consider clusters A and B as illustrated in Figure 9.2. Let us describe the distance (between A and B) by $\mathrm{d}(A, B)$ and denote the number of patterns in A and B by n_1 and n_2, respectively.

We have several ways of computing the distance between two such clusters, as described below.

Single link method (see Figure 9.2 (a)) the distance $d(A, B)$ is based on the minimal distance between the patterns belonging to A and B. It is computed as

$$d(A, B) = \min_{\mathbf{x}\in A, \mathbf{y}\in B} d(\mathbf{x}, \mathbf{y}) \tag{7}$$

In essence, this distance is a radically "optimistic" mode, where we involve the closest patterns located in different clusters. The clustering method based on this distance is one of the most commonly used.

Complete link method (see Figure 9.2 (b)) This approach is on the opposite end of the spectrum, since it is based on the farthest distance between two patterns in two clusters:

$$d(A, B) = \max_{\mathbf{x}\in A, \mathbf{y}\in B} d(\mathbf{x}, \mathbf{y}) \tag{8}$$

Group average link (see Figure 9.2 (c)) in contrast to the two previous approaches, in which the distance is determined on the basis of extreme values of the distance function, in this method we calculate the average between the distances as computed between each pair of patterns, with one pattern from each cluster:

$$d(A, B) = \frac{1}{\mathrm{card}(A)\mathrm{card}(B)} \sum_{\mathbf{x}\in A, \mathbf{y}\in B} d(\mathbf{x}, \mathbf{y}) \tag{9}$$

Obviously, the computations are more intensive, but they reflect a general tendency between the distances computed for individual pairs of patterns.

One can develop other ways of expressing the distance between clusters A and B. For instance, we can calculate the Hausdorff distance between two sets of patterns:

$$\mathrm{d}(A, B) = \max\{\max_{\mathbf{x}\in \mathrm{A}} \min_{\mathbf{y}\in \mathrm{B}} \mathrm{d}(\mathbf{x}, \mathbf{y}), \max_{\mathbf{y}\in \mathrm{B}} \min_{\mathbf{x}\in \mathrm{A}} \mathrm{d}(\mathbf{x}, \mathbf{y})\} \tag{10}$$

To illustrate the essence of distance computations between a data point and a cluster we will use the following example.

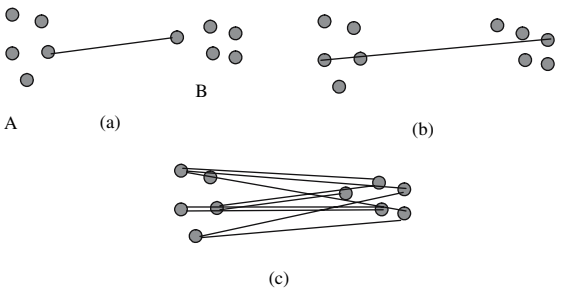

Figure 9.2. Two clusters A and B and several ways of computing the distance between them: (a) single link; (b) complete link; (c) group average link. Data are denoted by small circles.

Example: Let cluster A consist of three points (1,3), (2,3), and (1.5, 0.5). The distance between **x** and A, d(**x**,A) is computed using formulas (7)–(9). In the computations we use the Euclidean, Hamming and Tchebyschev distance, respectively. The results are displayed in a series of graphs shown in Figure 9.3.

An interesting general formula for expressing various agglomerative clustering approaches is known as the Lance-Williams recurrence formula. It expresses the distance between cluster A and B and the cluster formed by merging these two (which gives rise to cluster C):

$$d_{A\cup B,C} = \alpha_A d_{A,C} + \alpha_B d_{B,C} + \beta d_{A,B} + \gamma |d_{A,C} - d_{B,C}| \tag{11}$$

with adjustable values of the parameters $\alpha_A(\alpha_B), \beta$, and γ. The choice of values for these parameters implies a certain clustering method, as shown in Table 9.2.

With reference to the algorithmic considerations, hierarchical clustering can be realized in many different ways. One interesting alternative comes in the form of the Jarvis-Patrick (JP) algorithm. For each data point, we form a list of k nearest neighbors (where the neighborhood is expressed in terms of some predefined distance function).

We allocate two points to the same cluster if either of the following conditions is satisfied:

a) the points are within each other's list of nearest neighbors
b) the points have at least $k_{\min}$ nearest neighbors in common.

Here k and $k_{\min}$ are two integer parameters whose values are specified in advance before building the clusters. The JP algorithm has two evident advantages. First, it allows for nonconvex

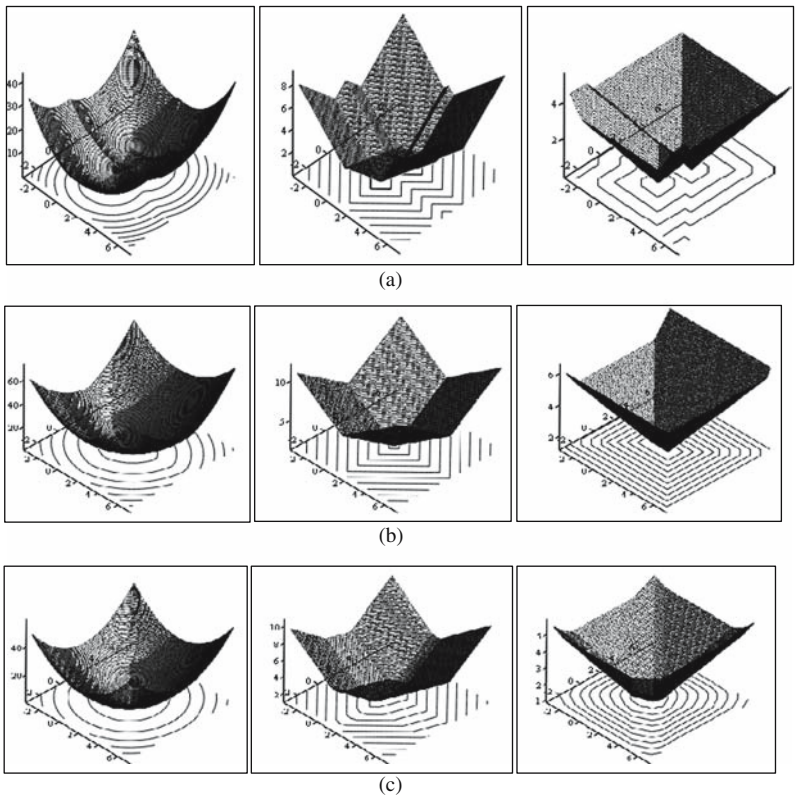

Figure 9.3. Plots of distance d(x,A) for (a) the single link (b) complete link, (c) group average link, and the Euclidean, Hamming, and Tchebyshev distances, from left to right, respectively.

Table 9.2. Values of the parameters in the Lance-Williams recurrence formula and the resulting agglomerative clustering; n_A, n_B and n_C denote the number of patterns in the corresponding clusters.

Clustering method	$\alpha_A(\alpha_B)$	β	γ
Single link	1/2	0	$-1/2$
Complete link	1/2	0	1/2
Centroid	$\frac{n_A}{n_A+n_B}$	$-\frac{n_A n_B}{(n_A+n_B)^2}$	0
Median	1/2	$-1/4$	0

clusters. This point becomes obvious: the data point a may be clustered with b and b could be grouped with c, and subsequently, the points a and c which do not seem to be related to each other end up in the same cluster. Second, the algorithm is nonparametric. Since it is based upon the ordering of the distances (ranking) it is less sensitive to potential outliers.

5. Objective Function-Based Clustering

The key design challenge in objective function-based clustering is the formulation of an objective function capable of reflecting the nature of the problem so that its minimization reveals meaningful structure (clusters) in the data.

Objective function-based clustering looks for a data structure through minimization of some **performance index** (called also **objective function**). Our anticipation is that a proper choice of the objective function and its minimization would help reveal a "genuine" structure in the data.

There are many possibilities for formulating the objective function and various ways of organizing the optimization activities. In what follows, we discuss several representative algorithms by emphasizing the very nature of the underlying methods. The representation of the structure (clusters) is provided in two ways, namely, as a collection of representatives (prototypes) and as a partition matrix. Let us denote the prototypes by $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c$. The partition matrix $\mathrm{U} = [u_{ik}]$ consists of c rows and N columns whose entries describe allocation of the corresponding data to the consecutive clusters.

5.1. K-Means Algorithm

The minimum variance criterion is one of the most common options that help organize the data. It comes with a clear and intuitive motivation: the **prototypes** of a large number of data should be such that they minimize a dispersion of data around them. Having N patterns in $\mathbf{R}^n$ and assuming that we are interested in forming c clusters, we compute the sum of dispersions between the patterns and a set of prototypes $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c$:

$$\mathrm{Q} = \sum_{\mathrm{i}=1}^{\mathrm{c}} \sum_{\mathrm{k}=1}^{\mathrm{N}} \mathrm{u}_{\mathrm{ik}} ||\mathbf{x}_{\mathrm{k}} - \mathbf{v}_{\mathrm{i}}||^2 \tag{12}$$

where $||\ ||^2$ being Euclidean distance between $\mathbf{x}_{\mathrm{k}}$ and $\mathbf{v}_{\mathrm{i}}$. The important component in the above sum is the partition matrix $U = [u_{ik}], i = 1, 2, \ldots, c$, $k = 1, 2, \ldots, N$ whose role is to allocate the patterns to the clusters. The entries of U are binary. Pattern k belongs to cluster i when $u_{ik} = 1$. The same pattern is excluded from the cluster when u_{ik} is equal to 0.

Partition matrices satisfy the following conditions:

- each cluster is nontrivial, i.e., it does not include all patterns and is nonempty:

$$0 < \sum_{k=1}^{N} u_{ik} < N, i = 1, 2, \ldots, c \tag{13}$$

- each pattern belongs to a single cluster

$$\sum_{i=1}^{C} u_{ik} = 1, k = 1, 2, \ldots, N \tag{14}$$

The family of partition matrices (binary matrices satisfying these two conditions) will be denoted by $\mathbf{U}$. As a result of minimization of Q, we construct the partition matrix and a set of prototypes. Formally, we express this construct in the following way, which is an optimization problem with constraints:

$$\text{Min } Q \text{ with respect to } \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_c \text{ and } U \in \mathbf{U} \tag{15}$$

There are a number of approaches for this optimization. The most common is K-Means, a well established way to cluster data.

The flow of the main optimization activities in ***K*-Means clustering** can be outlined in the following manner:

Start with some initial configuration of the prototypes $\mathbf{v}_i, i = 1, 2, \ldots, c$ (e.g., choose them randomly)

- iterate
- construct a partition matrix by assigning numeric values to U according to the following rule

$$u_{ik} = \begin{cases} 1, \text{ if } d(\mathbf{x}_k, \mathbf{v}_i) = \min_{j \neq i} d(\mathbf{x}_k, \mathbf{v}_j) \\ 0, \qquad\qquad \text{otherwise} \end{cases} \tag{16}$$

- update the prototypes by computing the weighted average, which involves the entries of the partition matrix

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} u_{ik} \mathbf{x}_k}{\sum_{k=1}^{N} u_{ik}} \tag{17}$$

until the performance index Q stabilizes and does not change, or until the changes are negligible.

Partition matrices form a vehicle to illustrate the structure of the patterns. For instance, the matrix formed for $N = 8$ patterns split into $c = 3$ clusters is shown as follows:

$$U = \begin{bmatrix} 1\,0\,0\,1\,0\,1\,0\,1 \\ 0\,1\,1\,0\,0\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,1\,0 \end{bmatrix}$$

Each row describes a single cluster. Thus we have the following arrangement: the first cluster consists of patterns $\{1, 4, 6, 8\}$, the second involves a set of patterns $\{2, 3\}$, and the third one covers the remaining patterns, that is $\{5, 7\}$

Graphical visualization of the partition matrix (data structure) can be shown in the form of a star or radar diagram as shown in Figure 9.4.

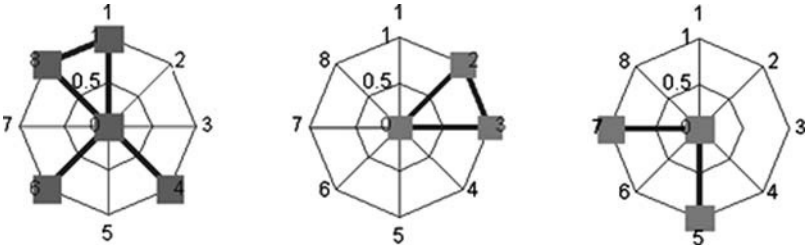

Figure 9.4. Star diagram as a graphical representation of the partition matrix for three clusters.

5.2. Growing a Hierarchy of Clusters

Objective function-based clustering can be organized in some **hierarchical topology** of clusters that helps us reveal a structure in a successive manner while reducing the required computing effort. The crux of this approach is as follows: we start with a fairly small number of clusters, say, 3–5. Given this low number, the optimization may not be very demanding. For each cluster formed here, we determine the value of the associated objective function, say the one given by expression (4) and computed for each cluster separately. Denote it by Q_i. The cluster with the highest value of Q_i is a candidate for further splitting (structural refinement). We cluster the data belonging to this cluster into c groups and next compute the values of the objective function for each of these groups. Again, we find the cluster with the highest objective function and proceed with its refinement (further splits). This process is repeated until we reach the point where the values of the objective functions for each cluster have fallen below a predefined threshold or we have exceeded the maximal number of clusters allowed in this process. The growth of the cluster tree (Figure 9.5) depends on the nature of the data. In some cases, we can envision a fairly balanced growth. In others, the growth could concentrate upon portions of the data where some newly split clusters are subject to further consecutive splits.

The advantage of this stepwise development process is that instead of proceeding with a large number of clusters in advance (which carries a significant computing cost associated with this grouping), we successively handle the clusters that require attention due to their heterogeneity (dispersion).

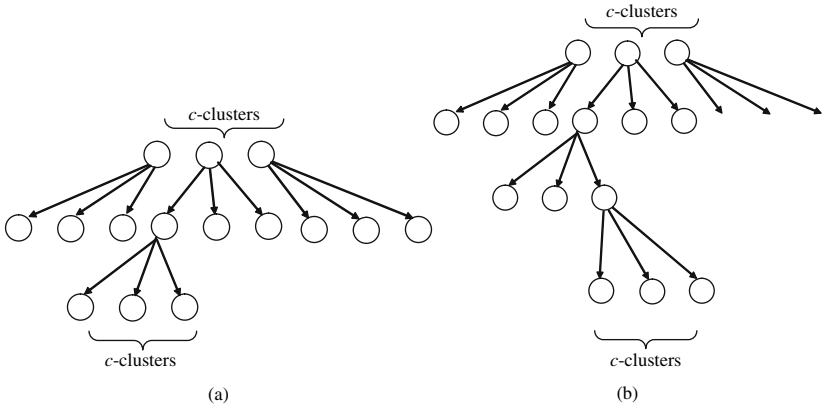

Figure 9.5. Growing a tree of clusters (a) balanced growth where most clusters at the higher level are split; (b) imbalanced growth, in which some clusters are subject to consecutive splits.

5.3. Kernel–based Clustering

The structure present in original data could be quite complicated – a situation that poses a genuine challenge to a variety of clustering techniques. One interesting alternative to address this problem is to elevate the original data $\mathbf{x}, \mathbf{y}, \mathbf{z}\ldots$ to a higher dimensional space M (where typically $M >> N$) in anticipation that the structure arising there could be made quite simple. This possibility could create an advantage when clustering the data in this new space. Consider a certain mapping $\phi(\mathbf{x})$. By applying it to all data to be clustered, e.g., $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ we end up with $\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \ldots, \phi(\mathbf{x}_N)$, which are now located in the extended (augmented) space and are subject to clustering. As an illustration of the concept, let us focus on the following objective function involving distances $||.||$ formulated in some augmented space:

$$Q = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^{m} ||\varphi(\mathbf{x}_k) - \varphi(\mathbf{v}_i)||^2$$

The optimization of Q is realized with respect to the partition matrix U and the prototypes $\phi(\mathbf{v}_1), \phi(\mathbf{v}_2), \ldots, \phi(\mathbf{v}_c)$ located in this new space. This task could appear to be more complicated given the fact that now the patterns are located in a highly dimensional space. Hopefully, there are ways around it that alleviate this potential difficulty. The use of the Mercer theorem allows us to express a scalar product of the transformed data as a **kernel function** $K(\mathbf{x}, \mathbf{y})$ so that

$$K(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y})$$

Obviously, there is a direct correspondence between the kernel function and the mapping ϕ. Here also originates the name of the kernel-based clustering itself.

If we use Gaussian kernels of the form

$$K(\mathbf{x}, \mathbf{y}) = exp(-||\mathbf{x} - \mathbf{y}||^2/\sigma^2)$$

then the distance present in the objective function transforms into the far more manageable form

$$||\phi(\mathbf{x}_k) - \phi(\mathbf{v}_i)||^2 = 2 - K(\mathbf{x}_k, \mathbf{v}_i)$$

Once this transformation has been noted, the derivations of the complete algorithm are straightforward. These will become obvious when we present the details of Fuzzy C-Means (FCM) clustering.

5.4. K-medoids Algorithms

Before we move on to the essence of the ***K*-medoids clustering** algorithm, we note that the prototypes of the clusters are formed on the basis of all elements, and if any data point is affected by noise, this affects the prototypes. Furthermore, the prototype is not one of the elements of the data, which again could be treated as an undesired phenomenon. To alleviate these difficulties, we resort to representatives of the data that are more robust. These come under the term *medoids*. To explain the concept, we start with the concept of a median.

The **median**, med $\{x_1, x_2, \ldots, x_N\}$, of real numbers is an **ordered statistic** that expresses a "central" element in the set. Assume that the above set of data is ordered, say, $x \leq, x_2, \ldots \leq x_N$. The median is defined as follows: (a) median $= x_{(N+I)/2}$ if N is odd and (b) median $= (x_{N/2} + x_{N/2+1})/2$ if N is even. It is worth stressing that the median is a robust estimator: the result does not depend on noisy data. To illustrates, let us imagine that x_1 assumes a value that is far lower than the rest of the data. x_1 is therefore an *outlier* and, as such, should not impact the result. The median

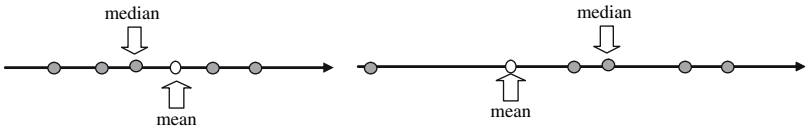

Figure 9.6. Median of dataset $\{x_1, x_2, \ldots, x_N\}$. Note its robustness property which manifests as no change even in the presence of one or more outlier(s) – see the situation represented on the right-hand side.

reflects this situation – note that its value has not been changed (see Figure 9.6). Evidently, in this case because of the outlier, the mean of this data set is pushed far down in comparison with the previous case.

One could easily demonstrate that the median is a solution to the following optimization problem:

$$\min_{ii} \sum_{k=1}^{N} |x_k - x_{ii}| = \sum_{k=1}^{N} |x_k - \text{med}| \tag{18}$$

where *med* denotes the median. Interestingly, in the above objective function we encounter a Hamming distance, which stands in sharp contrast with the Euclidean distance we found in the *K*-means algorithm. In other words, some objective functions promote the **robustness** of clustering.

In addition to the evident robustness property (highly desirable), we note that the prototype is one of the elements of the data. This is not the case in *K*-means clustering where the prototype is calculated from averaging and hence does not have any interpretability.

In general, for *n*-dimensional data, the objective function governing the clustering into *c* clusters is written in the form

$$Q = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik} || \mathbf{x}_k - \mathbf{v}_i ||^2 = \sum_{i=1}^{c} \sum_{k=1}^{N} \sum_{j=1}^{n} u_{ik} |x_{k_j} - v_{ij}| \tag{19}$$

Finding the minimum of this function leads to the centers of the clusters. Nevertheless, this optimization process can be quite tedious. To avoid this type of the optimization process, other arrangements are considered. They come under the name of specialized clustering techniques including Partitioning Around Medoids (PAM) and Clustering LARge Applications (CLARA).

The underlying idea of the PAM algorithm is to represent the structure in the data by a collection of **medoids** – a family of the most centrally positioned data points. For a given collection of medoids, each data point is grouped around the medoid to which its distance is the shortest. The quality of the produced clustering formed from the collection of medoids is quantified by taking the sum of distances between the medoids and the data belonging to the corresponding cluster represented by the specific medoid. PAM starts with an arbitrary collection of elements treated as medoids. At each step of the optimization, we make an exchange between a certain data point and one of the medoids, assuming that the swap results in improvement in the quality of the clustering. This method has some limitations with reference to the size of the dataset. Experimental results demonstrate that PAM works well for small datasets with a small number of clusters, for example, 100 data points and 5 clusters. To deal with larger datasets, the method has been modified to sample the dataset rather than operating on all data. The resulting method, called CLARA (Clustering LARge Applications), draws the sample, applies PAM to this sample and finds the medoids. CLARA draws multiple samples and produces the best clustering as the output of the clustering.

In summary, note that the K-medoids algorithm and its variants offer two evident and highly desirable features:

a) **robustness** and
b) **interpretability** of the prototype (as one of the elements of the dataset).

However it also comes with a high computational overhead, which needs to be phased into the overall knowledge discovery process.

5.5. Fuzzy C-Means Algorithm

In the above described "standard" clustering methods, we assumed that clusters are well-delineated structures, namely, that a data point belongs to only one of the clusters. While this assumption sounds mathematically appealing, it is not fully reflective of reality and comes with some conceptual deficiencies. Consider the two-dimensional data set shown in Figure 9.7.

How many clusters can we distinguish? "Three" seems like a sound answer. If so, we assume $c = 3$ and run a clustering algorithm, the two data points situated between the two quite dense and compact clusters have to be assigned to one of the clusters. K-means will force them to be assigned somewhere. While this assignment process is technically viable, the conceptual aspect is far from being fully accepted. Such points may be difficult to assign using Boolean logic, therefore we would do better to flag them out by showing their partial membership (belongingness) to both clusters. A split of membership of $1/2 - 1/2$ or $0.6 - 0.4$ or the like could be more reflective of the situation presented by these data. To adopt this line of thought, we have to abandon the concept of two-valued logic (0–1 membership). Doing so brings us to the world of fuzzy sets, described in Chapter 4. The allocation of the data point becomes a matter of degree – the higher the membership value, the stronger its bond to the cluster. At the limit, full membership (a degree of membership of 1) indicates that the data point is fully allocated to the cluster. Lower values indicate weaker membership in the cluster. The most "unclear" situation occurs when the membership in each cluster is equal to 1/c that is the element is shared among all clusters to the same extent. Membership degrees are indicative of "borderline" elements: their membership in the clusters is not obvious, and this situation is easily flagged for the user/data analyst.

The concept of **partial membership** in clusters is the cornerstone of fuzzy clustering. In line with objective function-based clustering, we introduce the concept of a fuzzy partition matrix. In contrast to the binary belongingness of elements to individual clusters, we now relax the condition of membership by allowing the values of u_{ik} to be positioned anywhere in [0,1]. The two other

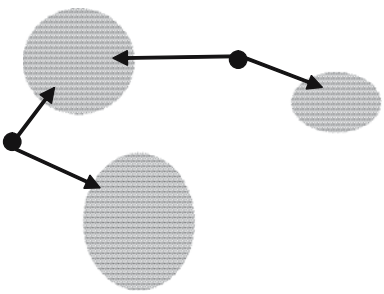

Figure 9.7. Three compact clusters and two isolated data points inbetween them: a two-valued logic challenge and the emergence of fuzzy sets.

fundamental conditions remain the same as before, as given by equations (13) and (14). The **objective function** incorporates the partition matrix and comes in the form of the double sum:

$$Q = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^{m} \, ||\mathbf{x}_k - \mathbf{v}_i||^2 \tag{20}$$

The elements of the partition matrix come with a fuzzification coefficient (m) whose values are greater than 1. The optimization of equation (20) is completed with respect to the prototypes and the partition matrix. Without going into computational details, we offer a general iterative scheme of **Fuzzy *C*-Means clustering**, in which we successively update the partition matrix and the prototypes.

Initialize: Select the number of clusters (c), stopping value (ε), and fuzzification coefficient (m). The distance function is Euclidean or weighted Euclidean. The initial partition matrix consists of random entries satisfying equations (13)–(14).

Repeat

update prototypes

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} u_{ik}^{m} \mathbf{x}_k}{\sum_{k=1}^{N} u_{ik}^{m}} \tag{21}$$

update partition matrix

$$u_{ik} = \frac{1}{\sum_{l=1}^{c} \left(\frac{||\mathbf{x}_k - \mathbf{v}_i||}{||\mathbf{x}_k - \mathbf{v}_j||} \right)^{2/(m-1)}} \tag{22}$$

Until a certain stopping criterion has been satisfied

The **stopping criterion** is usually taken to be the distance between two consecutive partition matrices, $U(\text{iter})$ and $U(\text{iter}+1)$. The algorithm is terminated once the following condition is satisfied:

$$\max_{ik} \, | u_{ik}(\text{iter}+1) - u_{ik}(\text{iter}) | \quad \leq \varepsilon \tag{23}$$

We may use for instance $\varepsilon = 10^{-6}$. Note that the above expression is nothing but a Tchebyshev distance. The role of the parameters of this clustering is the same as that already discussed in the case of K-means. The new parameter here is the fuzzification coefficient, which did not exist in the previous algorithms for the obvious reason that the entries of the partition matrix were only taken to be 0 or 1. Here the fuzzification coefficient plays a visible role by affecting the shape of the membership functions of the clusters. Some snapshots of membership functions are shown in Figure 9.8.

Notably, the values close to 1 yield almost Boolean (binary) membership functions.

5.6. Model-based Algorithms

In this approach, we assume a certain probabilistic model of the data and then estimate its parameters. This structure, which is highly intuitive comes under the name of *mixture density*. We assume that the data are a result of a **mixture** of c sources of data that might be thought of as clusters. Each component of this mixture is described by some conditional probability density function (pdf), $p(\mathbf{x}|\theta_i)$, characterized by a vector of parameters $\boldsymbol{\theta}_i$. The prior probabilities

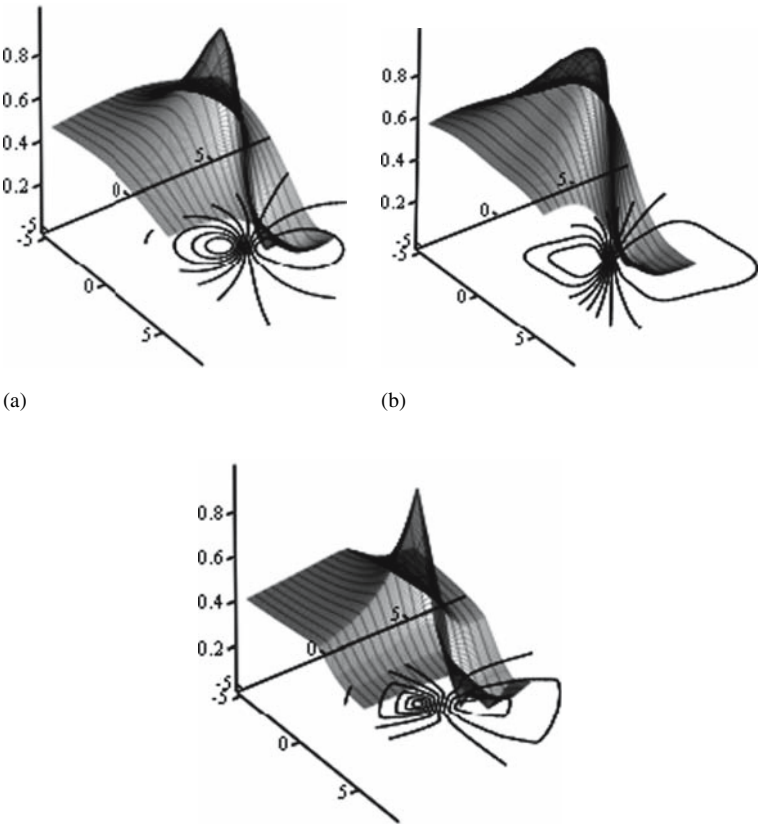

Figure 9.8. Plots of membership functions for several values of the fuzzification coefficient (m) used in the objective function (a) $m = 1.2$; (b) $m = 2.0$; (c) $m = 3.0$.

$p_1, p_2, \ldots, p_c$ of clusters are given. Under these assumptions, the model is additive and comes in the form of mixture densities:

$$p(\mathbf{x}|\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \ldots, \boldsymbol{\theta}_c) = \sum_{\mathrm{i}=1}^{c} p(\mathbf{x}|\boldsymbol{\theta}_i) p_i \tag{24}$$

Given the nature of the model, we also refer to p_1, p_2, .. and p_c as mixing parameters. To build the model, one has to estimate the parameters of the contributing pdfs. To do so we have to assume that $p(\mathbf{x}, \boldsymbol{\theta})$ is identifiable which means that if $\boldsymbol{\theta} \neq \boldsymbol{\theta}'$ then there exists an $\mathbf{x}$ such that $p(\mathbf{x}|\boldsymbol{\theta}) \neq p(\mathbf{x}|\boldsymbol{\theta})$. The standard approach used to discover the clusters is to carry out **maximum likelihood estimation**. In essence, this estimate maximizes the expression

$$P(\mathbf{X}|\boldsymbol{\theta}) = \prod_{k=1}^{N} p(\mathbf{x}_k|\boldsymbol{\theta}) \tag{25}$$

One should know that the above optimization problem is not straightforward, especially with high-dimensional data.

5.7. Scalable Clustering Algorithms

5.7.1. Density-Based Clustering (DBSCAN)

As the name indicates, **density-based clustering** methods rely on the formation of clustering on the basis of the density of data points. This approach follows a very intuitive observation: if in some region of the feature space the data are located close to each other, then their density is high, and hence they form a cluster. On the other hand, if there are some points in some region and their density is low, they are most likely potential outliers not associated with the majority of the data.

This appealing observation constitutes the rationale behind the algorithms belonging to the category of density-based clustering. The DBSCAN method is one of the common representatives of this category, with OPTICS, DENCLUE, and CLIQUE following the same line of thought.

To convert these intuitive and compelling ideas into the algorithmic environment, we introduce the notion of the ε-neighborhood. Given some data $\mathbf{x}_k$, the ε-neighborhood, denoted by $N_\varepsilon(\mathbf{x}_k)$, is defined as:

$$N_\varepsilon(\mathbf{x}_k) = \{\mathbf{x} | d(\mathbf{x}, \mathbf{x}_k) \quad \leq \varepsilon\} \tag{26}$$

Note that the form of the distance function implies the geometry of the ε-neighborhood. Obviously, higher values of ε produce larger neighborhoods. For the neighborhood of $\mathbf{x}_k$, we can count the number of data points falling within it. We introduce another parameter, N_Pts, that tells us how many data points fall within a neighborhood. If the neighborhood of $\mathbf{x}_k$ is highly populated by other data, that is

$$\text{card } (N_\varepsilon(\mathbf{x}_k)) \geq \text{N\_Pts}$$

we say that $\mathbf{x}_k$ satisfies a core point condition. Otherwise, we label $\mathbf{x}_k$ as a border point.

We say that $\mathbf{x}_i$ is $\mathbf{x}_k$ density-reachable with parameters ε and N_Pts if:

(a) $\mathbf{x}_i$ belongs to $N_\varepsilon(\mathbf{x}_k)$, and
(b) card $(N_\varepsilon(\mathbf{x}_k)) \geq$ N_Pts

Figure 9.8 illustrates this type of **reachability**. Note that the $\mathbf{x}_i$ could also be density-reachable from $\mathbf{x}_k$ by a chain of other data points:

$$\mathbf{x}_{k+1}, \mathbf{x}_{k+2}, \ldots, \mathbf{x}_{i-1} \tag{27}$$

such that $\mathbf{x}_{k+1}$ is density reachable from $\mathbf{x}_k$, $\mathbf{x}_{k+2}$ is density reachable from $\mathbf{x}_{k+1}$, etc. This type of transitivity is again illustrated in Figure 9.9.

The property of density reachability becomes the crux of the underlying clustering algorithm. We form the clusters on the basis of density reachability, and all data belonging to the same cluster are those that are density reachable.

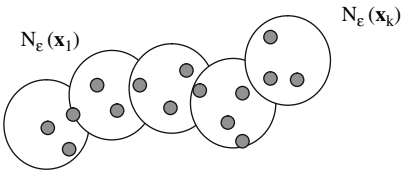

Figure 9.9. The concept of density reachability and its transitive character when a sequence of data is involved.

Given this background rationale, the generic DBSCAN algorithm consists of the following sequence of steps.

Set up the parameters of the neighborhood, e and N_Pts:

(a) arbitrarily select a data point, say, $\mathbf{x}_k$
(b) find (retrieve) all data that are density reachable from $\mathbf{x}_k$
(c) if $\mathbf{x}_k$ is a core point, then the cluster has been formed (all points are density reachable from $\mathbf{x}_k$)
(d) otherwise, consider $\mathbf{x}_k$ to be a border point and move on to the next data point

The sequence (a) – (d) is repeated until all data points have been processed.

If we take a closer look at the clustering mechanisms, we can see that the concept of density-based reachability focuses on the formation of groups in a local way. In essence, each data point (regarded as a potential core) "looks" at its surroundings, which are formed by its neighborhood of some predefined size (ε). This situation offers a broad variety of potential geometric shapes of clusters that could be formed in this manner. The elements that are not density reachable are treated as outliers. Obviously, as in any other mechanism of unsupervised learning, one should be aware of the number of parameters whose values directly impact the performance of clustering and the form of the results. Three of these play an essential role. The distance function, as already mentioned, defines the geometry of the formed neighborhood. The size of the neighborhood and the number of points, N_Pts, affect the granularity of the search. These latter two are related; a higher value of ε requires higher values of N_Pts. If we consider very small values of ε, the DBSCAN starts building a significant number of clusters. With increasing values of ε, far fewer clusters are formed, more data points are regarded as outliers, and a more general structure is revealed. In essence, these parameters offer a significant level of flexibility, yet the choice of suitable values of the parameters becomes a data-dependent task. The computational complexity of the method is O (NlogN).

5.7.2. Cure

The essence of CURE (Clustering Using Representatives) is to exploit the concept of scattered points in clusters. Let us contrast this method with the two extreme clustering techniques. In centroid-based clustering (such as, e.g., *K*-means), we use single elements (prototypes) to represent the clusters. In hierarchical clustering, on the other hand, at the beginning of the process all points are representative for the clustering process. In the CURE algorithm, we choose a collection of scattered data in each cluster. The intent of these scattered points is to reflect the shape of the clusters. During the process of clustering, the scattered points shrink toward the center (mean). The speed of shrinking is controlled by the damping coefficient (called the shrinking factor), which assumes values between 0 and 1. If the value of the shrinking factor is 1, then we end up with *K*-means clustering. At the other extreme, with the shrinking factor equal to 0 (where no shrinking occurs), we end up with hierarchical clustering. Since each cluster is represented by a collection of points, the method is less sensitive to outliers as shrinking helps eliminate their potential impact. For the realization of CURE in the presence of large datasets, the method implements the following process. First, it draws a random sample of data. Assuming that the sample is large enough, one could anticipate that this sample could reflect the overall structure. Next CURE partitions the random sample and clusters the data in each partition.

6. Grid - Based Clustering

The ideas and motivation behind **grid-based clustering** are appealing and quite convincing. Clustering reveals a structure at some level of generality, and we are interested in describing such structures in the language of generic geometric constructs like **hyperboxes** and their combinations.

These result in a highly descriptive shape of the structure. Hence it is appropriate to start by defining a grid in the data space and then processing the resulting hyperboxes. Obviously, each hyperbox is described in terms of some statistics of data falling within its boundaries, yet in further processing we are not concerned with dealing with individual data points. By avoiding this detailed processing, we save computing time. Let us start with a simple illustrative example. Consider a collection of quite irregular clusters as shown in Figure 9.10.

It is very likely that partition-based clustering could have problems with this dataset due to the diversity of geometric shapes of the clusters. Likewise, to carry out clustering, we have to identify the number of clusters in advance, which poses a major challenge.

Grid-based clustering alleviates these problems by successively merging the elements of the grid. The boxes combined together give rise to a fairly faithful description of the clusters, and this otcome becomes possible irrespective of the visible diversity in the geometry of the shapes.

Let us move into a more formal description of grid–based clustering by introducing the main concepts. The key notion is a family of hyperboxes (referred to as *blocks*) that are formed in the data space. Let us denote these by $B_1, B_2, \ldots, B_p$. They satisfy the following requirements: (a) B_i is nonempty in the sense that it includes some data points, (b) the hyperboxes are disjoint, that is $B_i \cap B_j = \emptyset$ if $i \neq j$, and (c) a union of all hyperboxes covers all data that is $\bigcup_{i=1}^{p} B_i = \mathbf{X}$ where $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$.We also require that such hyperboxes "cover" some maximal number (say, $b_{\max}$) of data points. Hyperboxes differ between themselves with respect to the number of data they cover. To measure the property of how well a certain hyperbox reflects the data, we compute a density index that is computed as the ratio of the number of data falling within the hyperbox and its volume.

Next, the clustering algorithm clusters the blocks B_i, (and, in essence, the data) into a nested sequence of nonempty and disjoint collections of hyperboxes. The hyperboxes with the highest density become the centers of clusters of hyperboxes. The remaining hyperboxes are afterwards clustered iteratively based on their density index, thereby building new cluster centers or merging with existing clusters. We can merge only those hyperboxes that are adjacent to a certain cluster ("neighbor"). A neighbor search is conducted, starting at the cluster center and inspecting adjacent blocks. If a neighbor block is found, the search proceeds recursively with this hyperbox.

Algorithmically, grid-based clustering comprises the following fundamental phases:

- Formation of the grid structure
- Insertion of data into the grid structure
- Computation of the density index of each hyperbox of the grid structure
- Sorting the hyperboxes with respect to the values of their density index
- Identification of cluster centres (viz. the hyperboxes of the highest density)
- Traversal of neighboring hyperboxes and the merging process

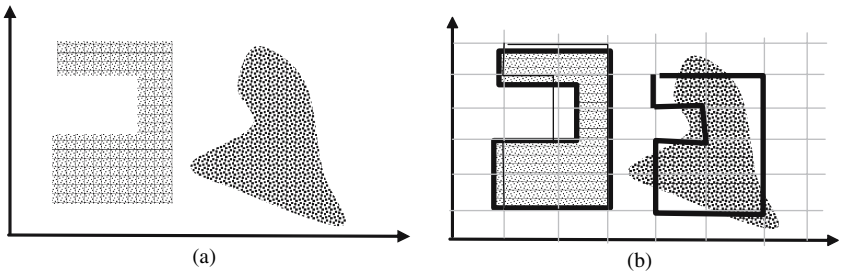

Figure 9.10. (a) A collection of geometrically different clusters; (b) and a grid structure formed in the data space along with clusters being built by merging the adjacent boxes of the grid structure.

One should be aware that since clustering moves only towards merging hyperboxes, the choice of grid (which becomes a prerequisite for the overall algorithm) does deserve careful attention. A grid which is rough may not help capture the details of the structure in the data. A grid that is too detailed produces a significant computational overhead.

In the literature, most studies focus on cases that involve a significant number of data (N) yet they shy away from discussing data of high dimensionality. This tendency is perhaps understandable since grid-based clustering becomes particularly beneficial in these cases (the processing of the hypeboxes abstracts all computing from the large number of the individual data points).

To summarize, let us highlight the outstanding features of grid-based clustering:

- The grid-based clustering algorithm scans the data set only once and in this way can potentially handle large data sets.
- By considering basic building blocks the method could handle a broad range of possible geometric shapes of clusters.
- Since grid-based clustering is predominantly concerned with the notion of density, it is helpful in handling clusters of arbitrary shapes. Similarly, although we rely on the density of points, we can detect potential outliers.

7. Self-Organizing Feature Maps

Objective function-based clustering forms one of the main optimization paradigms of data discovery. To put it in a broader perspective, we switch to an alternative arising in neural networks (see Chapter 13), namely **self-organizing feature maps**. This alternative will help us to contrast the underlying optimization mechanisms and to look at formats of results generated by different clustering methods.

The concept of a Self-Organizing feature Map (SOM) was originally developed by Kohonen. As emphasized in the literature, SOMs are regarded as neural networks composed of a grid of artificial neurons that attempt to show highly dimensional data in a low-dimensional structure, usually in the form of a two- or three-dimensional map. To make such visualization meaningful, one ultimate requirement is that the low-dimensional representation of the originally high-dimensional data has to preserve the **topological properties** of the data set.

In a nutshell, this requirement means that two data points (patterns) that are close to each other in the original feature higher-dimensional space should retain this similarity (or closeness or proximity) in their representation (mapping) in the reduced, lower-dimensional space. Similarly, two distant patterns in the original feature space should retain their distant locations in the lower-dimensional space. By being more descriptive, SOM acts as a **computer eye** that helps us gain insight into the structure of the data and observe relations occurring between patterns that were originally located in a high dimensional space by showing those relations in a low-dimensional, typically two- or three-dimensional space. In this way, we can confine ourselves to a two-dimensional map that apparently preserves all the essential relations between the data as well as the dependencies between the individual variables. In spite of many variations, the generic SOM architecture (as well as the learning algorithm) remains basically the same. Below we summarize the essence of the underlying self-organization algorithm that realizes a certain form of unsupervised learning.

Before proceeding with detailed computations, we introduce necessary notation. We assume, as usual, that the data are vectors composed of "n" real numbers, viz. they are elements of $\mathbf{R}^n$. The SOM is a collection of linear neurons organized in the form of a two-dimensional grid (array), as shown in Figure 9.11.

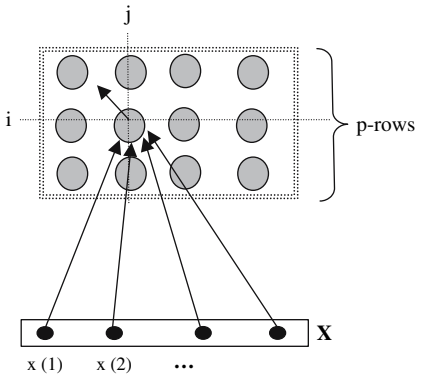

Figure 9.11. A basic topology of the SOM constructed as a grid of identical neurons.

In general, the grid may consist of p rows and r columns; quite commonly, we use the square array of $p \times p$ neurons. Each neuron is equipped with modifiable connections $\mathbf{w}(i, j)$ where these form an n-dimensional vector of connections:

$$\mathbf{w}(i, j) = [w_1(i, j) w_2(i, j) \ldots w_n(i, j)]$$

Note that the pair of indexes ($i0$, $j0$) refers to the location of this neuron on the map (array of neurons). It completes computing of the distance function $||.||$ between its connections and the input vector $\mathbf{x}$:

$$y(i, j) = ||\mathbf{w}(i, j) - \mathbf{x}|| \tag{28}$$

The distance function can be any of those discussed earlier. In particular, one could consider Euclidean distance or its weighted version. The same input $\mathbf{x}$ is fed to all neurons. The neuron with the shortest distance between the input vector and its own weight vector becomes activated and is called the winning neuron. Let us denote the coordinates of the neuron by ($i0$, $j0$). More precisely, we have

$$(i0, j0) = \arg\min_{(i,j)} ||\mathbf{w}(i, j) - \mathbf{x}|| \tag{29}$$

The winning neuron best matches (responds to, is similar to) the input vector $\mathbf{x}$. As a winner of the competition, it is rewarded by being allowed to modify its weight so that it becomes positioned even closer to the input. The **learning rule** is read as follows (see Chapter 13, Sec. 3.2.1 for more details):

$$\mathbf{w\_new}(i0, j0) = \mathbf{w}(i0, j0) + \eta(\mathbf{x} - \mathbf{w}(i0, j0)) \tag{30}$$

where η denotes a learning rate, $\eta > 0$. The higher the learning rate is, the more intensive the updates of the weight are. In addition to the changes of weight of the winning neuron, we often allow its neighbors (the neurons located at the consecutive coordinates of the map) to update their weights as well. This influence is quantified via a **neighbor function** Φ (i, j, i0, j0). In general, this function satisfies two intuitively appealing conditions:

a) it attains a maximum equal to 1 for the winning node, i = i0, j = j0, Φ(i0, j0, i0, j0) = 1 and
b) when the node is apart from the winning node, the value of the function gets lower (the updates are smaller). Evidently, there are also nodes where the neighbor function goes to zero and the nodes are not affected.

Considering the above, we rewrite Equation (30) in the following form:

$$\mathbf{w\_new}(i, j) = \mathbf{w}(i0, j0) + \eta\Phi(i, j, i0, j0)(\mathbf{x} - \mathbf{w}(i, j)) \tag{31}$$

The typical neighbor function comes in the form

$$\Phi(i, j, i0, j0) = \exp(-\beta((i - i0)^2 + (j - j0)^2)) \tag{32}$$

with parameter β usually assuming small positive values.

Expression (31) applies to all the nodes (i, j) of the map. As we iterate (update) the weights, the neighborhood function shrinks: at the beginning of the updates, we start with a large region, and when the learning settles down, we start reducing the size of the neighborhood. For instance, one may think of a linear decrease of neighborhood size. To emphasize this relationship, we can use the notation $\Phi(iter, i, j, i0, j0)$ where *iter* denotes consecutive iterations of the learning scheme.

Either the number of iterations is specified in advance or the learning terminates once there are no significant changes in the weights of the neurons.

Some conditions for successful learning of the SOM neural network algorithm are as follows:

- The training data set must be sufficiently large since self-organization relies on statistical properties of data.
- Proper selection of the neighbor function will assure that only the weights of the winning neuron and its neighborhood neurons are locally adjusted
- The radius, and thus the size, of the winning neighborhood must monotonically decrease with learning time (in successive iterations)
- The amount of weight adjustment for neurons in a winning neighborhood depends on how close they are to the input.

Another approach to designing a SOM network is to use heuristics and simulation-based findings:

- Since the weights of the SOM network have to approximate the probability density of input vectors $p(\mathbf{x})$, it is advisable to have visual inspection of this distribution first. This can be done by using Sammon **nonlinear projection** as an initial step before designing the SOM network (we elaborate on this idea later on).
- The size of the 2D array of neurons should be large enough to accommodate all the clusters that can be present in the data. An array that is too small may allow discovery of only the coarse clustering structure (smaller clusters may be put together). For smaller problems, the number of neurons in a 2D array should be approximately equal to the number of input vectors.
- Since good accuracy of statistical modeling requires a large number of samples (say 100,000), in practice, for a limited size of input data sets, we may use the data set several times repetitively, either randomly or by cycling the data in the same sequence through the network.
- To enhance the impact coming some patterns that are known to be important, we can simply present them to the network a large number of times, or increase their learning rate nd/or neighbor function $\Phi(i, j, i0, j0)$.
- Patterns with missing values can be still used for training, with the missing value being replaced by the average of that feature
- when it is desired in some applications such as, monitoring of real-time measurements, to map some nominal data in a specific location on the map (for instance, in the middle) we can copy these patterns as initial values for the neurons in the desired map location, and then keep their learning rate low during successive adjustments.

Example: This example shows the use of the SOM algorithm for finding the hierarchical structure of pattern vectors after Kohonen. It is an interesting example of SOM self-organization and global ordering as shown for five-dimensional patterns. The labeled training set, shown in Table 9.2 contains 32 input vectors with categories labeled by letters A, B, …,Z and digits 1,2,…,6. The SOM neural network has been composed with its 5 inputs fully connected by weights with 70 neurons.

The neurons have been arranged in a rectangular 7×10 array with hexagonal neighborhood geometry:

The SOM network has been trained using only unlabeled patterns selected randomly. After 10,000 iterations, the weights of the network converged to the stable values. Then the trained network was calibrated by presenting to it known labeled patterns (x_i, class_j), the classes being shown as feature x_6 in Table 9.2.

A B C D E
F
G
H K L M N O P Q R
I S W
J T X 1 2 3 4 5 6
U Y
V Z

For each labeled pattern, the array response was computed and the neuron with the strongest response was labeled by the pattern class. For example, when the pattern (2,0,0,0,0 ; B) was presented to the network, the left upper neuron in the array had the dominating strongest response was labeled as class B. It was shown that only 32 neurons were activated during calibration, whereas the remaining 38 neurons remained unlabeled, as is shown below.

This structure represents a global order, achieved through local interactions and weights adjustments, and reflects a *minimum spanning tree* relationship among the input patterns as shown below.

This example shows the power of self-organization. The minimum spanning tree is a mathematical technique, originating in graph theory. Assume that the data patterns from a given set will be presented as a planar graph, where each vertex represents one data pattern (vector). In addition, assume that one assigns to each edge, connecting two vertices a weight equal to the distance between the vertices. The minimum spanning tree is the spanning tree of the pattern vertices for which the sum of its connected edges is minimal. In another words, the minimum spanning tree for the data vectors is a corresponding planar tree that connects all data patterns to their closest neighbors such that total length of the tree edges is minimal. Analyzing the data in

Table 9.2. Kohonen's five-dimensional example.

x_1	x_2	x_3	x_4	x_5	Class
1	0	0	0	0	A
2	0	0	0	0	B
3	0	0	0	0	C
4	0	0	0	0	D
5	0	0	0	0	E
3	1	0	0	0	F
3	2	0	0	0	G
3	3	0	0	0	H
3	4	0	0	0	I
3	5	0	0	0	J
3	3	1	0	0	K
3	3	2	0	0	L
3	3	3	0	0	M
3	3	4	0	0	N
3	3	5	0	0	O
3	3	6	0	0	P
3	3	7	0	0	P
3	3	8	0	0	R
3	3	3	1	0	S
3	3	3	2	0	T
3	3	3	3	0	U
3	3	3	4	0	V
3	3	6	1	0	W
3	3	6	2	0	X
3	3	6	3	0	Y
3	3	6	4	0	Z
3	3	6	2	1	1
3	3	6	2	2	2
3	3	6	2	3	3
3	3	6	2	4	3
3	3	6	2	5	5
3	3	6	2	6	6

Table 9.2, we find that the Euclidean distances between subsequent neighboring patterns differ by a value of one. For instance, for patterns $\mathbf{x}_A$ and $\mathbf{x}_B$

$$||\mathbf{x}_A - \mathbf{x}_B|| = 1 < ||\mathbf{x}_A - \mathbf{x}_i||, \quad i = C, D, \ldots, Z, 1, \ldots, 6$$

SOM and FCM are complementary, and so are their advantages and shortcomings. FCM requires the number of groups (clusters) to be defined in advance. It is guided by a certain performance index (objective function), and the solution comes in the clear form of a certain partition matrix. In contrast, SOM is more user oriented. There is no number of clusters (group) that needs to be specified in advance.

As emphasized very clearly so far, SOM provides an important tool for visualization of high-dimensional data in a two- or three-dimensional space. The preservation of distances is crucial to this visualization process.

The same idea of distance preservation is a cornerstone of the Sammon's **projection method**; however, the realization of this concept is accomplished in quite a different manner. This nonlinear projection attempts to preserve topological relations between patterns in the original and the reduced spaces by preserving the interpattern distances. Sammon's projection algorithm minimizes an error defined as the difference between patterns in the original and reduced feature spaces. More

formally, let $\{\mathbf{x}_k\}$ be the set of L n-dimensional vectors $\mathbf{x}_k$ in the original feature space $\mathbf{R}^n$, and let $\{\mathbf{y}_k\}$ be the set of L corresponding m-dimensional vectors $\mathbf{y}$ in the reduced low-dimensional space R^{m}, with $m << n$. Most often we take $m = 2$. Let us denote by $d(\mathbf{x}_i, \mathbf{x}_j)$ the distance (usually Euclidean) between two vectors in the original feature space. Denote by $d(\mathbf{y}_i, \mathbf{y}_j)$ the distance between two vectors in the (reduced) projected feature space. Sammon's algorithm determines the projection such that it minimizes the following **distortion measure** J_d, for all L patterns, defined as follows

$$\mathrm{J_d} = \frac{1}{\sum_{\mathrm{i=1,i\neq j}}^{\mathrm{L}} \sum_{\mathrm{j=1,j\neq i}}^{\mathrm{L}} \mathrm{d}(\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{j})} \sum_{\mathrm{i=1,i\neq j}}^{\mathrm{L}} \sum_{\mathrm{j=1,j\neq i}}^{\mathrm{L}} \frac{(\mathrm{d}(\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{j}) - \mathrm{d}(\mathbf{y}_\mathrm{i}, \mathbf{y}_\mathrm{j}))^2}{\mathrm{d}(\mathbf{x}_\mathrm{i}, \mathbf{x}_\mathrm{j})}$$

This criterion, called Sammon's stress, expresses how well all interpattern distances are preserved in the projection into a lower-dimensional feature space. Minimization procedures, such as gradient descent, can be used to find optimal projections that minimize this distortion criterion. To avoid getting stuck in a local minimum, one may start from different, random, initial configurations, or add noise. Since for every iteration step, $\mathrm{L}(\mathrm{L}-1)/2$ inter-pattern distances need to be computed, this algorithm becomes impractical for large number of patterns. Other techniques for minimization, such as evolutionary programming, can also be used for finding a solution and possibly even finding the global minimum. Unfortunately, Sammon's algorithm does not provide an explicit function describing the relationship between pattern vectors in both the original and projected spaces. Consequently after finding the optimal projection for a given set of L patterns, the algorithm does not exhibit a generalization capability. Thus, for new data, the minimization procedure must be rerun from scratch to accommodate both old and new data.

8. Clustering and Vector Quantization

Briefly speaking, **vector quantization** concerns various means of data compression, which is essential when dealing with storage and transmission of images, audio files, multimedia information and so forth (see Chapter 7). There are two important criteria, namely, quality of reconstruction (here we are interested in minimal quantization error) and high compression rate (so that we can store and transmit only a small portion of the original data to faithfully reconstruct the original source). An overall scheme for such processing is shown in Figure 9.12.

At the encoding end, we represent data through prototypes. These are usually referred to as a **codebook**. Any input datum is then captured in terms of the elements of the codebook and the index of the representative that matches it best is transmitted or stored. Formally, we can explain the flow of processing in the following manner:

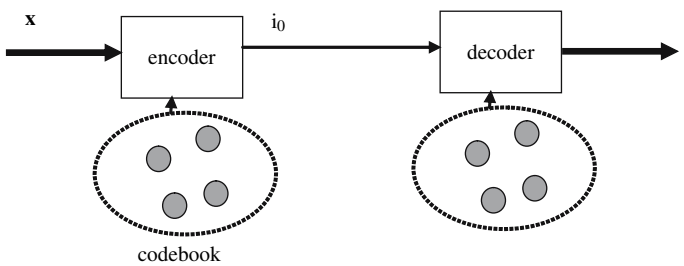

Figure 9.12. The principle of vector quantization. The encoder and decoder both use the same codebook; in transmission, rather than sending a multidimensional vector **x**, one transmits an index of the best prototypes (i_0) which are used at the decoding end (hence the result of decoding is one of the elements of the codebook).

Encoding: determine the best representative (prototype) of the codebook and store (transmit) its index i_0, $i_0 = \arg\min_i ||\mathbf{x} - \mathbf{v}_i||$ where $\mathbf{v}_i$ denotes the i^{th} prototype.

Decoding: recall the best prototype given the transmitted index (i_0).

Clustering as we have discussed here, is geared towards the discovery of structure in the data. There are some similarities and differences between the processes of vector quantization and clustering.

Clustering is an integral part of vector quantization. As a matter of fact, in the formation of the codebook we directly use various tools of clustering, such as K-Means.

The apparent differences concern key objectives. In vector quantization, we are interested in the quality of reconstruction (recall) while in clustering, cast in the framework of data mining, interpretability becomes very important. For instance, to maintain a low reconstruction error we might need to consider two very close elements of the codebook (their use helps us keep the error low enough). The same two prototypes, considered in the context of data mining may not be retained since their closeness makes them almost the same (conceptually redundant) from the point of view of data interpretation.

9. Cluster Validity

Since clustering is one of the two key unsupervised learning techniques, we should proceed very carefully with the assessment of its results. Are the generated clusters (along with their representation in the form of prototypes, partition matrices, dendrograms, etc.) reflective of the true nature of the data? This is a fundamental issue that permeates all clustering pursuits and profoundly impacts the practical usefulness of the technique. We should be fully cognizant of the fact that, while in essence being unsupervised, clustering is subconsciously endowed with some implicit components of supervision. The selection of these components impacts the character and quality of the results delivered by any clustering algorithm. We focus on the two main components present in almost any algorithm no matter what its nature and algorithmic details (SOM does not require a priori specification of the number of clusters, but its interpretation could be fairly complicated)

The choice of a **similarity measure** plays a primordial role in the search in the data space. We have discussed many types of such measures. We indicated that each distance measure used implies a certain geometry in the data space. As a consequence, the clustering technique endowed with specific distance is searching for structure in data that conforms to this specific geometry (such as hyperspheres in the case of using Euclidean distance). In the latter example, regardless of the "real" structure (shape) of clusters (and we never know it), an algorithm would searches for spherical clusters only. In short, we predispose the clustering right up front to search for clusters of some specific shape. This shortcoming is very much problem dependent. First, one could envision that due to the normality of data distribution, it is very likely that the structure could quite well conform to this geometric model of the clusters. In the case when a more complicated geometry of the clusters is encountered, we can envision that clustering with more clusters may take care of this problem (see Figure 9.13). In essence, one could adopt Euclidean distance as a fairly general and reasonable model of real shape in the data. The price is a larger number of clusters when there is a higher geometric diversity that could be resolved by asking for more clusters to be generated. The advantage is in the numeric treatment, since the Euclidean distance facilitates the optimization aspects of the clustering techniques.

The weighted Euclidean distance is an example of modified distance, which takes into consideration substantial differences in the ranges of the variables. Some other distances such as the Mahalanobis, come with an increased geometric flexibility but are associated with a high price of computing inverse of the corresponding covariance matrices (and thus are often quite prohibitive).

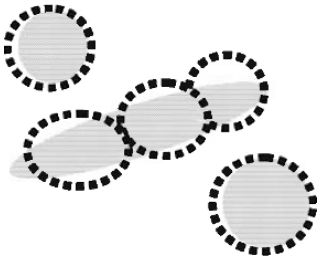

Figure 9.13. Diverse structure/shape of data in which clustering using the Euclidean distance can result in a larger number of spherical clusters (three clusters are required to represent the elongated cluster).

The number of clusters (c) that must be specified by users a priori (except when using SOM) is an extremely important parameter affecting the clustering outcome, since it explicitly determines a level of detail of the overall search for structure. On the one hand, we focus on the ensuing analysis and thus may anticipate that a certain range of the number of clusters (say between 3 and 8) does exist in the data. For example, in doing market analysis, we can easily guess the approximate number of categories of customers or at least contemplate a range of possible numbers of valid clusters.

In general, however, users must guess a priori the number of clusters that a given algorithm will be asked to generate. Normally, the user would guess the range of the number of clusters and then use **cluster validity measures** to assess which particular number of clusters best reveals the true structure in the data.

Two categories of tasks fall under the umbrella of what is referred to as **cluster validity** – a suite of methodologies and algorithms that offer us some mechanisms to validate clustering results.

There are many measures, called **cluster validity indices**, whose values relate to the number of clusters generated, and thus are used to judge the clusters detected in the data and to assess the quality of the structure revealed in this manner. In what follows, we present some of these measures and explain their motivation and computational details. In any case, we must always remember that the success of the clustering validity index depends upon the characteristics of the data and the clustering algorithm being used.

In spite of the diversity of clustering algorithms, we can spell out two fundamental and intuitively appealing requirements to which clusters should adhere.

Compactness. This property expresses how close the elements in a cluster are. For instance, consider a variance of the elements: the lower the value of the variance, the higher the compactness of the cluster. Since we are interested in compact clusters, low values of compactness are desirable. Likewise, we can calculate distances between the elements belonging to a cluster (intra cluster distances).

Separability. For this property, we evaluate how distinct the clusters are. An intuitive way of expressing separability is to compute inter cluster distances. Since we strive for high compactness and high separability, a structure should be characterized by small values of intra cluster distances and large values of inter cluster distances.

The realization of this observation comes in the form of the **Davies-Bouldin index**. To determine its value, we compute the within scatter distance for the i^{th} cluster:

$$s_i = \frac{1}{\text{card}(\Omega_i)} \sum_{\mathbf{x} \in \Omega_i} ||\mathbf{x} - \mathbf{v}_i||^2 \tag{33}$$

with ||.|| being some distance function. Here Ω_i denotes the i^{th} cluster. We introduce the following distance between the prototypes of the clusters:

$$d_{ij} = ||\mathbf{v}_i - \mathbf{v}_j||^2 \tag{34}$$

which can serve as the inter cluster distance between the two clusters. Now, we define the ratio

$$r_i = \max_{j, j \neq i} \frac{s_i + s_j}{d_{ij}} \tag{35}$$

and sum its values over the clusters arriving at the following sum:

$$r = \frac{1}{c} \sum_{i=1}^{c} r_i \tag{36}$$

The "optimal" ("correct") number of clusters (c) is the number for which the value of r attains its minimum. Note that the minimum of r favors minimal values of the nominator of r_i and maximal values of the denominator in this expression. This is really what makes the clusters compact and well separated.

In the same vein of the minimization of scattering with a cluster or the maximization of inter cluster distances comes the Dunn separation index. In this construct, we first define a **diameter of the cluster**:

$$\Delta(\Omega_\mathrm{i}) = \max_{\mathbf{x},\mathbf{y} \in \Omega_\mathrm{i}} ||\mathbf{x} - \mathbf{y}|| \tag{37}$$

Then the inter cluster distance is expressed as

$$\delta(\Omega_\mathrm{i}, \Omega_\mathrm{j}) = \min_{\mathbf{x} \in \Omega_\mathrm{i}, \mathbf{y} \in \Omega_\mathrm{j}} ||\mathbf{x} - \mathbf{y}|| \tag{38}$$

The **Dunn separation index** is formed as follows:

$$\mathrm{r} = \min_{\mathrm{i}} \min_{\mathrm{j}, \mathrm{j} \equiv \mathrm{c}} \frac{\delta(\Omega_\mathrm{i}, \Omega_\mathrm{j})}{\max_\mathrm{k} \Delta(\Omega_\mathrm{k})} \tag{39}$$

The values of r are maximized with respect to the number of the clusters.
The **Xie-Benie index** relates to fuzzy clustering and realizes the same concept as described above. Here we arrive at the following expression:

$$\mathrm{r} = \frac{\sum_{\mathrm{k}=1}^{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{c}} \mathrm{u}_{\mathrm{ik}}^{\mathrm{m}} ||\mathbf{x}_\mathrm{k} - \mathbf{v}_\mathrm{i}||^2}{\mathrm{N}\{\min_{\mathrm{i} \neq \mathrm{j}} ||\mathbf{v}_\mathrm{i} - \mathbf{v}_\mathrm{j}||^2\}} \tag{40}$$

Note that the "optimal" number of clusters will result in the lowest values of r yet the index may show some monotonicity when the number of clusters is quite close to the number of the data.

The concepts of scattering and compactness of clusters could be realized in different ways. One quite commonly encountered approach relies on forming a sound compromise between these two concepts. Let us introduce the average scattering

$$\mathrm{av\_scatter} = \frac{1}{c} \sum_{i=1}^{c} \frac{s_i}{s} \tag{41}$$

where s_i denotes a measure of scattering for the i^{th} cluster while s stands for the scattering reported for the entire data set. The expression for the separation between clusters reads as follows

$$\text{separation} = \frac{D_{\min}}{D_{\max}} \sum_{i=1}^{c} \left(\sum_{j=1}^{c} ||\mathbf{v}_i - \mathbf{v}_j|| \right)^{-1} \tag{42}$$

where $D_{\min} = \min_{i,j=1,2,\ldots,c} ||\mathbf{v}_i - \mathbf{v}_j||$ and $D_{\max} = \max_{i,j=1,2,\ldots,c} ||\mathbf{v}_i - \mathbf{v}_j||$. The validity index (SD) is taken as a weighted combination of the average scattering and separation, that is

$$\text{SD} = \alpha \text{ av\_scatter} + \text{separation} \tag{43}$$

where α is used to strike a sound balance between the two components of the index. By monitoring the values of SD treated as a function of c and determining its lowest value, we arrive at the plausible (or "optimal") number of clusters.

With reference to fuzzy clustering, there are several interesting cluster validity indicators. The first of these are based exclusively on the values of the partition matrix, while the third one takes into consideration the data as well as the prototypes. The **partition coefficient** P_1 is built upon the entries of the partition matrix:

$$P_1 = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^2 \tag{44}$$

The values assumed by this index are in $[1/c, 1]$. At the extreme, if the data belong to a single cluster, viz. u_{ik} equals 1 for some i, then P_1 is equal to 1. On the other hand, if we encounter an equal distribution of membership grades (so it is likely that the data set does not exhibit any evident strongly manifested structure), $u_{ik} = 1/c$ then P_1 is equal to 0. When searching for the number of clusters, we look at the plot of P_1 versus c and choose the number of clusters for which this plot exhibits some "knee" – a place where a substantial change in the values of the index occurs.

The **partition entropy** is defined as:

$$P_2 = -\frac{1}{N} \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik} \log_a u_{ik} \tag{45}$$

where $a > 0$. The values of P_2 are confined to the interval $[0, \log_a c]$. Again, the most suitable number of clusters is determined by inspecting the character of the relationship P_2 treated as a function of c.

The third validity index not only takes into account the partition matrix but also involves the data as well as the prototypes,

$$P_3 = \sum_{i=1}^{c} \sum_{k=1}^{N} u_{ik}^m (||\mathbf{x}_k - \mathbf{v}_i||^2 - ||\mathbf{v}_i - \mathbf{v}||^2) \tag{46}$$

The first component of P_3 expresses a level of compactness of the clusters (dispersion around the prototypes) while the second one is concerned with the distances between the clusters (more specifically, their prototypes) and the mean of all data ($\mathbf{v}$).

In general, while these validity indexes are useful, in many cases they may produce inconclusive results. Given this effect, one should treat them with a big grain of salt by understanding that they offer only some guidelines and do not decisively point at the unique "correct" number of clusters. Also remember that when the number of clusters approaches the number of data points the usefulness of these indices is limited. Hopefully, in practice the number of clusters will always be a small fraction of the size of the dataset. In many cases, the user (decision-maker) could also provide some general hints in this regard.

10. Random Sampling and Clustering as a Mechanism of Dealing with Large Datasets

Large data sets require very careful treatment. No matter what clustering technique one might envision, it may be impractical due to the size of the data set. The communication overhead could be enormous, and the computing itself could be quite prohibitive. However, a suitable solution to the problem can be identified trough the use of **sampling**. There are two fundamental conceptual and design issues in this respect. First, we should know what a random sample means and how large it should be to become representative of the dataset and the structure we are interested in revealing. Second, once sampling has been completed, one has to know how to perform further clustering. Let us note that some of these mechanisms have been already mentioned in the context of clustering algorithms, such as CURE.

10.1. Random Sampling of Datasets

In a nutshell, we are concerned with a random draw of data from a huge dataset so that the clustering completed for this subset (samples) leads to meaningful results (that could be "similar" to those obtained when we run the clustering algorithm on the entire dataset). Given the clustering, we could rephrase the question about size of the random sample so that the probability of deforming the structure in the sample (for example, by missing some clusters) is low. Let us note that the probability of missing some cluster Ω is low if the sample includes a fraction of the data belonging to this cluster, say f*card(Ω) with $f \in [0, 1]$. Recall that card(Ω) stands for the number of data in Ω. The value of f is dependent on the geometry of the clusters and their separability. In essence, we can envision that if the structure of data is well defined, the clusters will be condensed and well-separated and then the required fraction f could be made substantially lower. Let us determine the size s of the sample such that that the probability that the sample contains fewer than $f$card(Ω) is less than δ. Let Z_j be a binary variable of value 1 if the j^{th} data belongs to the cluster Ω and 0 otherwise. Assume that Z_j are 0-1 random variables treated as independent Bernoulli trials such that $P(Z_j = 1) = \text{card}(\Omega)/N$, $j = 1, 2, \ldots s$. The number of data in the sample that belong to cluster Ω is then expressed as the sum $Z = \sum_{i=1}^{s} Z_i$. Its expected value is $\mu = \text{E}(Z) = s\text{card}(\Omega)/N$. If we use the Chernoff bounds for the independent Poisson trials $Z_1, Z_2, \ldots, Z_s$ we find the following probability bounds:

$$P[Z < (1 - \varepsilon)\mu] < \exp(-\mu\varepsilon^2/2) \tag{47}$$

In other words, the probability that the number of data falls below the expected count m by more than $\varepsilon\mu$ is lower than the right–hand expression (47). We require that the probability that this number falls below $f$card(Ω) should be no more than d. In other words, we require that the following holds:

$$P(Z < f\text{card}(\Omega)) < \delta \tag{48}$$

Let us rewrite the expression in the following equivalent format:

$$P[Z < (1 - (1 - f\,\mathrm{card}(\Omega)/\mu))] < \delta \tag{49}$$

With the use of the Chernoff bound, the above inequality for the probability holds if:

$$\exp\left(-\frac{\mu\left(1-\frac{f\,\mathrm{card}(\Omega)}{\mu}\right)^2}{2}\right) \leq \delta \tag{50}$$

Given that $\mu = s\,\mathrm{card}(\Omega)/N$, we solve the above equation with respect to s we obtain the following inequality:

$$s \geq fN + \frac{N}{\mathrm{card}(\Omega)}\ln\left(\frac{1}{\delta}\right) + \frac{N}{\mathrm{card}(\Omega)}\sqrt{\left(\ln\left(\frac{1}{\delta}\right)\right)^2 + 2f\,\mathrm{card}(\Omega)\ln\left(\frac{1}{\delta}\right)} \tag{51}$$

In other words (51) holds if the sample size is not lower than the number provided above. If we neglect some very small clusters (which may not have played any significant role), it has been shown that the sample size is independent from the original number of data points.

Clustering based on the random samples of the data

The sampling of data is a viable alternative to the dimensionality problem. A random sample is drawn and, on that basis, we discover a structure of the data. To enhance the reliability of the results, another option is to cluster the prototypes developed for each sample. This overall scheme is illustrated in Figure 9.14.

We draw a random sample, complete clustering and return a collection of the prototypes. Denote these by $\mathbf{v}_i[ii]$ with the index ii denoting the ii^{th} random sample. Then all the prototypes are again clustered in this way, reconciling the structures developed at the lower level. Since the number of elements to cluster at this level is far lower than in the sample itself, the clustering at the higher level of the structure does not require any substantial computing.

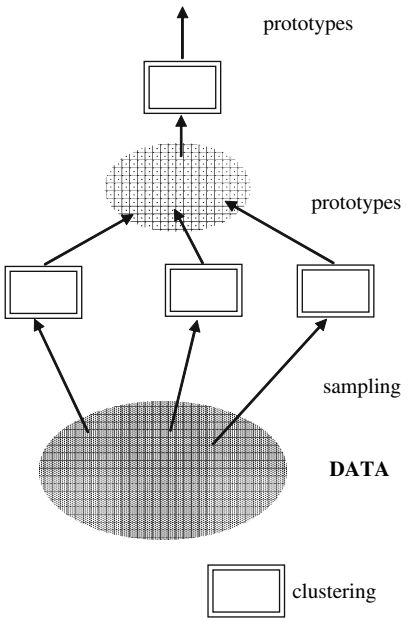

Figure 9.14. A two-level (double) clustering: random sampling followed by clustering of the prototypes.

11. Summary and Biographical Notes

In this Chapter, we have covered clustering, which occupies a predominant position in unsupervised learning and data mining. There are several compelling reasons. For this first, clustering techniques play an important role in revealing structure in data, basically without supervision. This feature of clustering is valuable given the fact that we do not know the structure in data and thus any mechanism that could help develop some insights into it is highly desirable. We presented a spectrum of clustering methods and elaborated on their conceptual properties, computational aspects and scalability. We addressed the issue of validity of clustering, stressing that although several cluster validity indexes are available, their outcomes should be treated with caution. The treatment of huge databases through mechanisms of sampling and distributed clustering was discussed as well. The latter two approaches are essential for dealing with huge datasets.

Clustering has been an area of very intensive research for several decades; the reader may consult classic texts in this area such as [1, 9, 11, 14, 17, 18, 20, 35]. The literature on fuzzy clustering is also abundant starting with the book by Bezdek [3]; interesting and useful references are [4, 7, 8, 12, 13, 16, 21, 22]. Cluster validity issues are presented in [10, 35, 36, 37]. The concept of self-organizing feature maps is well presented by Kohonen [23, 24, 25, 26]. Hierarchical clustering is covered in [31].

References

1. Anderberg, M.R. 1973. *Cluster Analysis for Applications*, Academic Press
2. Babu, G.P., and Murthy, M.N. 1994. Clustering with evolutionary strategies. *Pattern Recognition*, 27, 321–329
3. Bezdek, J.C. 1981. *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press
4. Bezdek, J.C, Coray, C.R., Guderson, R., and Watson, J. 1981. Detection and characterization of cluster substructure, *SIAM Journal of Applied Mathematics*, 40: 339–372
5. Bezdek, J.C., Keller, J., Krishnampuram, R., and Pal, N.R. 1999. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*, Kluwer Academic Publishers
6. Dave, R.N. 1990. Fuzzy shell clustering and application to circle detection in digital images, I*nternational Journal of General Systems*, 16: 343–355
7. Dave, R.N. 1991.Characterization and detection of noise in clustering. *Pattern Recognition Letters*, 12, 657–664
8. Dave, R.N., and Bhaswan, K. 1992. Adaptive c-shells clustering and detection of ellipses, *IEEE Transactions on Neural Networks*, 3: 643–662
9. Devijver, P.A., and Kittler, J. (Eds.). 1987. *Pattern Recognition Theory and Applications*, Springer-Verlag
10. Dubes, R. 1987. How many clusters are the best? – an experiment. *Pattern Recognition*, 20(6): 645–663
11. Duda, R.O., Hart, P.E., and Stork, D.G. 2001. *Pattern Classification*, 2nd edition, John Wiley
12. Dunn, J.C. 1974. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters, *Journal of Cybernetics*, 3(3): 32–57
13. Frigui, H., and Krishnapuram, R. 1996. A comparison of fuzzy shell clustering methods for the detection of ellipses, *IEEE Transactions on Fuzzy Systems*, 4: 193–199
14. Fukunaga, K. 1990. *Introduction to Statistical Pattern Recognition*, 2nd edition, Academic Press
15. Girolami, M. 2002. Mercer kernel-based clustering in feature space. *IEEE Transactions on Neural Networks*, 13(3): 780–784
16. Hoppner, F., Klawonn, F., Kruse, R., and Runkler, T. 1999. *Fuzzy Cluster Analysis*, John Wiley
17. Jain, A.K., Murthy, M.N., and Flynn, P.J. 1999. Data clustering: A review, *ACM Computing Survey*, 31(3): 264–323
18. Jain, A.K., Duin, R.P.W., and Mao, J. 2000. Statistical Pattern recognition: a review, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1): 4–37

19. Jarvis, R.A., and Patrick, E.A. 1973. Clustering using a similarity measure based on shared near neighbors, *IEEE Transactions on Computers*, C22(11): 1025–1034
20. Kaufmann, L., and Rousseeuw, P.J. 1990. *Finding Groups in Data: An Introduction to Cluster Analysis*, John Wiley
21. Kersten, P.R. 1999. Fuzzy order statistics and their applications to fuzzy clustering, *IEEE Transactions on Fuzzy Systems*, 7(7): 708–712
22. Klawonn, F., and Keller, A. 1998. Fuzzy clustering with evolutionary algorithms, *International Journal of Intelligent Systems*, 13: 975–991
23. Kohonen, T. 1982. Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, 43: 59–69
24. Kohonen, T. 1989. *Self-organization and Associative Memory*, Springer Verlag
25. Kohonen, T. 1995. *Self-organizing Maps*, Springer Verlag
26. Kohonen, T., Kaski, S., Lagus, K., and Honkela, T. 1996. Very large two-level SOM for the browsing of newsgroups, In: *Proceedings of ICANN96*, *Lecture Notes in Computer Science*, 1112, Springer, 269–274.
27. Krishnapuram, R., and Keller, J. 1993. A possibilistic approach to clustering, *IEEE Transactions on Fuzzy Systems*, 1(1993): 98–110
28. Krishnapuram, R., and Keller, J. 1996. The possibilistic C-Means algorithm: insights and recommendations, *IEEE Transactions on Fuzzy Systems*, 4: 385–393
29. Mali, K., and Mitra, S. 2002. Clustering of symbolic data and its validation, In: Pal, N.R., and Sugeno, M. (Eds.), *Advances in Soft Computing – AFSS 2002*, Springer Verlag, 339–344
30. Michalewicz, Z. 1992. *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag
31. Pedrycz, A., and Reformat, M. 2006. Hierarchical FCM in a stepwise discovery of structure in data, *Soft Computing*, 10: 244–256
32. Roth, V., and Steinhage, V. 1999. Nonlinear discriminant analysis using kernel functions, In: Solla, S., Leen, T.K., and Muller, K.R. (Eds.), *Advances in Neural Information Processing Systems*, MIT Press, 568–574.
33. Sammon, J.W. Jr. 1969. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, 5: 401–409
34. Xie, X.L., and Beni, G. 1991. A validity measure for fuzzy clustering, I*EEE Transactions on Pattern Analysis and Machine Intelligence*, 13: 841–847
35. Webb, A. 2002. *Statistical Pattern Recognition*, 2nd edition, John Wiley
36. Windham, M.P. 1980. Cluster validity for fuzzy clustering algorithms, *Fuzzy Sets & Systems*, 3: 1–9
37. Windham, M.P. 1982. Cluster validity for the fuzzy C-Means clustering algorithms, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11: 357–363
38. Vapnik, V.N. 1998. *Statistical Learning Theory*, John Wiley
39. Vesanto, J., and Alhoniemi, A. 2000. Clustering of the self-organizing map, *IEEE Transactions on Neural Networks*, 11: 586–600

12. Exercises

1. In grid-based clustering, form a grid along each variable of d intervals. If we are concerned with n dimensional data, how may hyperboxes are needed in total? If the data set consists of N data points, elaborate on the use of grid-based clustering vis-à-vis the ratio of the number of hyperboxes and the size of the data set. What conclusions could be derived? Based on your findings, offer some design guidelines.
2. Think of possible advantages of using the Tchebyschev distance in clustering versus some other distance functions (hint: think about the interpretability of the clusters by referring to the geometry of this distance).
3. In hierarchical clustering, we use different ways of expressing distance between clusters, which lead to various dendrograms. Elaborate on the impact of this approach on the shape of the resulting clusters.

4. Calculate the similarity between the two binary strings [1 1 0 0 1 1 0 0] and [00 11 11 1 0]. Compare differences between the results.
5. Run a few iterations of the K-Means for the toy dataset (1.0, 0.2) (0.9, 0.5) (2.0, 5.0) (2.1, 4.5) (3.1, 3.2) (0.9, 1.3). Select the number of clusters and justify your choice. Interpret the results.
6. Elaborate on the main differences between clustering and vector quantization.
7. Consider the same data set clustered by two different clustering algorithms and thus yielding two different partition matrices. How could you describe the clusters obtained by one method by using the clusters formed by another one?
8. The weights of a small 2×2 SOM developed for some three-dimensional data are as follows:

   (1,1): (0.3 -1.5 2.0) (1,2): (0.0 0.5 0.9)
   (2,1): (4.0 2.1 -1.5) (2,2): (0.8 -4.0 -1.0).

   Where would you locate the inputs $\mathbf{a} = [0.20.61.1]$ and $\mathbf{b} = [2.2\ -1.5\ -1.2]$?
   If you were to assess confidence of this mapping, what could you say about **a** and **b**?
9. In the hierarchical buildup of clusters governed by some objective function, we split the data into a very limited number of clusters and then proceed with a series of successive refinements. Consider that you allow for p phases of usage of the algorithm while using "c" clusters at each level. Compare computing costs of this clustering method with the clustering realized at a single level when using cp clusters.