CHAPTER

Clustering Algorithms IV 15

15.1 INTRODUCTION

The clustering algorithms presented in the previous two chapters evolved along two distinct major philosophies. The current chapter presents categories of algorithms that cannot be included in either of the previous two families, and they stem from various ideas. The first one includes clustering algorithms based on graph theory concepts, such as the minimum spanning tree, the directed tree and spectral clustering. The second category includes competitive learning algorithms. The third category includes branch and bound algorithms. These schemes guarantee to provide globally optimal clustering, in terms of a prespecified optimality criterion, at the cost of increased computational requirements. The fourth category contains algorithms that are based on morphological transformations. These have been inspired by the corresponding methods used in signal and image processing. The fifth category contains algorithms that are not based on cluster representatives but, instead, seek to place boundaries between clusters. Algorithms of the sixth category treat clusters as dense in data regions of the feature space separated by regions sparse in data. Alternatively, clusters may be viewed as peaks of the pdf, underlying the data in X, separated by valleys. The seventh category includes additional algorithms that are based on function optimization, such as simulated annealing and deterministic annealing. The difference from the algorithms presented in Chapter 14 is that the optimizing methods used in this chapter do not involve differential calculus concepts. In addition, this category also includes genetic algorithms modified suitably for clustering tasks. Finally, the eighth category includes algorithms that combine clusterings in order to produce a final (hopefully more accurate) one.

15.2 CLUSTERING ALGORITHMS BASED ON GRAPH THEORY

The algorithms of this family are capable of detecting clusters of various shapes, at least for the case in which they are well separated. Detection of clusters of various shapes is a feature that is shared by only a few other clustering algorithms.

15.2.1 Minimum Spanning Tree Algorithms

The first algorithm is based on the idea of the minimum spanning tree (MST) (Chapter 13) and is motivated by the way human perception works [Zahn 71]. More precisely, humans organize information with the most economical encoding [Hoch 64]. For example, the more likely way for a human to organize the points in Figure 15.1 is in four groups (clusters).

Let us consider the complete graph G, each node of which corresponds to a point of X. The weight of an edge $e = (\boldsymbol{x}_i, \boldsymbol{x}_j)$, w_e, connecting two nodes $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, is set equal to the distance $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ of the corresponding points in the feature space. Also, we say that two edges e_1 and e_2 are k *steps away* from each other if the minimum path that connects a vertex of e_1 and a vertex of e_2 has length equal to $k - 1$, that is, contains $k - 1$ edges.

The idea of the algorithm is the following: *determine the minimum spanning tree of G and then remove the edges that are "unusually" large compared with their neighboring edges*. These edges are called *inconsistent*, and it is expected that they connect points from different clusters. Next, we discuss a way to determine inconsistent edges. For each edge e, we consider all the edges, e_i, that lie k steps away from it, at the most, and we compute the mean, m_e, and the standard deviation, σ_e, of their weights. If w_e lies more than q (typically $q = 2$) standard deviations (σ_e) away from m_e, then we consider e as inconsistent. From this, it is clear that the characterization of an edge as inconsistent is somewhat subjective and depends on k and q, which are preselected.

Example 15.1

Consider Figure 15.2. Let $k = 2$ and $q = 3$. The edges lying two steps at the most from e_0 are e_i, $i = 1, \ldots, 10$. The mean m_{e_0} and the standard deviation σ_{e_0}, corresponding to e_0 are 2.3 and 0.95, respectively. Thus w_{e_0} lies 15.5 standard deviations (σ_{e_0}) away from m_{e_0}. Therefore, e_0 is an inconsistent edge.

Let us now consider the edge e_{11}. Working as before, we find that $m_{e_{11}}$ and $\sigma_{e_{11}}$ are 2.5 and 2.12, respectively. Thus $w_{e_{11}}$ is 0.24 standard deviations ($\sigma_{e_{11}}$) away from $m_{e_{11}}$. Therefore, e_{11} is not an inconsistent edge.

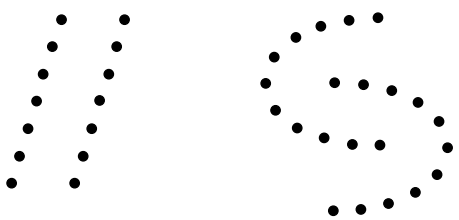

FIGURE 15.1

An arrangement of clusters.

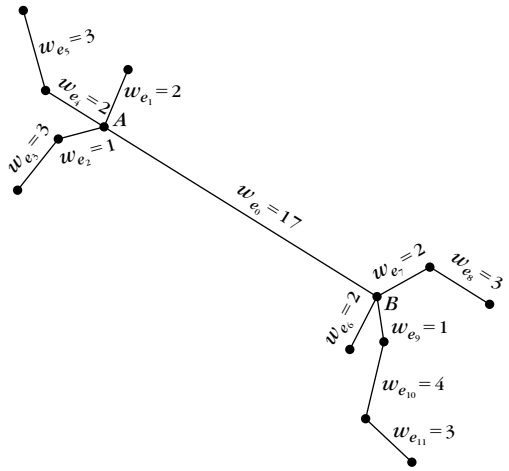

FIGURE 15.2

The minimum spanning tree of a graph. The edge e_0 is inconsistent, while e_{11} is consistent.

After these definitions, the MST clustering algorithm may be stated as follows.

The MST Clustering Algorithm

- Construct a complete graph G such that:
  - its vertices correspond to the vectors of X.
  - $w_{(\boldsymbol{x}_i,\boldsymbol{x}_j)} = d(\boldsymbol{x}_i, \boldsymbol{x}_j), i, j = 1, \ldots, N, i \neq j$.
- Determine the MST of G.
- Identify the inconsistent edges of the MST.
- The clusters are the connected components of the MST after the removal of the inconsistent edges.

This algorithm works satisfactorily for many cases where the clusters are well separated. However, this is not a panacea. Let us consider for example Figure 15.3. The edge AB has a very large neighboring edge (BC), which increases m_{AB} and σ_{AB}. Thus, AB may not be characterized as inconsistent and, as a consequence, the vectors from regions R_1 and R_2 are considered as members of the same cluster [Jarv 78].

Some suggestions for the use of the MST algorithm for the cases where we have touching clusters (Figure 15.4a), as well as for the case where the clusters have different densities (Figure 15.4b), are discussed in [Zahn 71]. However, they implicitly require knowledge of the shape of the clusters.

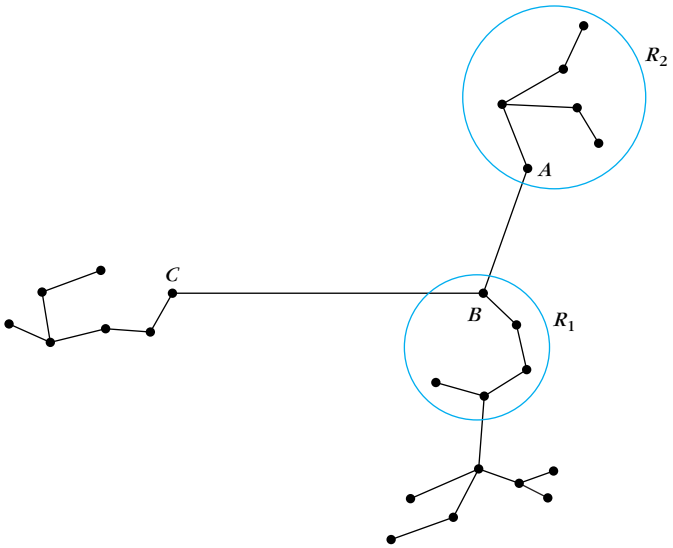

FIGURE 15.3

The MST clustering algorithm will assign the vectors of the regions R_1 and R_2 to the same cluster.

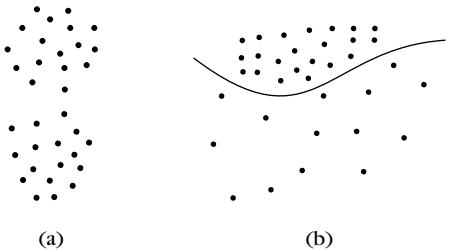

FIGURE 15.4

(a) Touching clusters. (b) Clusters with different densities.

Remark

- Note that this algorithm does not depend on the order in which data are considered by the algorithm and, also, no initial conditions are required, as is the case with the algorithms of Chapter 14.

15.2.2 Algorithms Based on Regions of Influence

An extension of the MST involves regions of influence for each pair of vectors of X. This idea was used by many researchers (e.g., [Tous 80, Gabr 69, Urqu 82]) in order to overcome the problems associated with the MST algorithms.

Let us consider two distinct vectors, $\boldsymbol{x}_i, \boldsymbol{x}_j \in X$. Their region of influence is defined as

$$R(\boldsymbol{x}_i, \boldsymbol{x}_j) = \{\boldsymbol{x}: cond(d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j), d(\boldsymbol{x}_i, \boldsymbol{x}_j)), \boldsymbol{x}_i \neq \boldsymbol{x}_j\} \tag{15.1}$$

where $cond(d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j), d(\boldsymbol{x}_i, \boldsymbol{x}_j))$ is a condition among the distances $d(\boldsymbol{x}, \boldsymbol{x}_i)$, $d(\boldsymbol{x}, \boldsymbol{x}_j)$, and $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$. Different choices of *cond* give rise to different shapes of regions of influence. Typical choices of *cond*, proposed in [Tous 80] and [Gabr 69], are

$$\max\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{15.2}$$

and

$$d^2(\boldsymbol{x}, \boldsymbol{x}_i) + d^2(\boldsymbol{x}, \boldsymbol{x}_j) < d^2(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{15.3}$$

respectively. Also, in [Urqu 82], the following two alternatives are proposed:

$$\begin{aligned} &(d^2(\boldsymbol{x}, \boldsymbol{x}_i) + d^2(\boldsymbol{x}, \boldsymbol{x}_j) < d^2(\boldsymbol{x}_i, \boldsymbol{x}_j)) \; OR \\ &(\sigma \min\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j)) \end{aligned} \tag{15.4}$$

and

$$\begin{aligned} &(\max\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j)) \; OR \\ &(\sigma \min\{d(\boldsymbol{x}, \boldsymbol{x}_i), d(\boldsymbol{x}, \boldsymbol{x}_j)\} < d(\boldsymbol{x}_i, \boldsymbol{x}_j)) \end{aligned} \tag{15.5}$$

where σ is a factor called *relative edge consistency*. This factor affects the size of the region of influence defined by $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. The shapes of these regions are shown in Figure 15.5. Other choices of *cond* are also possible. An algorithm based on the idea of the regions of influence is described next.

Algorithm Based on Regions of Influence

- For $i = 1$ to N
  - For $j = i + 1$ to N

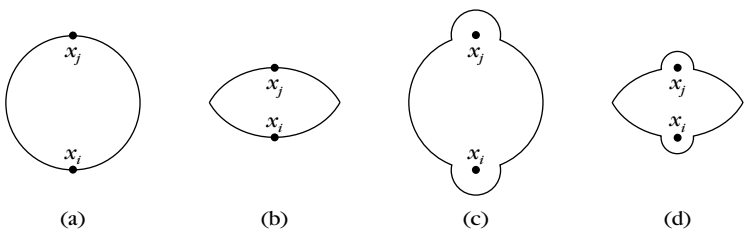

FIGURE 15.5

The shapes of the regions defined by (a) condition (15.2), (b) condition (15.3), (c) condition (15.4), and (d) condition (15.5).

- Determine the region of influence $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$.
- If $R(\boldsymbol{x}_i, \boldsymbol{x}_j) \cap (X - \{\boldsymbol{x}_i, \boldsymbol{x}_j\}) = \emptyset$ then
  - Add the edge connecting $\boldsymbol{x}_i, \boldsymbol{x}_j$
- End {If }

• End {For}

■ End {For}

■ Determine the connected components of the resulted graph and identify them as clusters.

In words, the edge between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is added if no other vector of X lies in $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$. This is because when $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are closely located, it is expected that no other points of X will be in $R(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The opposite is obviously true for points located further away.

The algorithm is insensitive to the order in which the pairs of vectors are considered. Also, for the last two choices of *cond*, the value of σ must be chosen *a priori*. The graphs produced by these algorithms when (15.2) and (15.3) are used are also called *relative neighborhood graphs (RNGs)* and *Gabriel graphs (GGs)*, respectively.

These techniques avoid situations such as the one shown in Figure 15.3. Moreover, several results are given in [Urqu 82] showing the superior performance for the last two choices of *cond* compared with the first two. Also, in [Urqu 82] it is shown how the idea of the regions of influence may be used to give rise to hierarchical algorithms. Finally, in [Ozbo 95], empirically defined regions of influence are used, that exhibit satisfactory behavior.

15.2.3 Algorithms Based on Directed Trees

An alternative clustering scheme, based on the idea of directed trees, is proposed in [Koon 76]. Before we proceed, let us give some definitions. We recall that a *directed graph* is a graph whose edges are directed (see Figure 15.6a). We say that a set of edges $e_{i_1}, \ldots, e_{i_q}$ constitute a *directed path* from a vertex A to a vertex B, if A is the initial vertex of e_{i_1}, B is the final vertex of e_{i_q}, and the destination vertex of the edge $e_{i_j}, j = 1, \ldots, q-1$, is the departure vertex of the edge $e_{i_{j+1}}$. For example, in Figure 15.6a, the sequence e_1, e_2, e_3 constitutes a directed path connecting the vertices A and B. Finally, a *directed tree* is a directed graph with a specific node A, known as *root*, such that (a) every node $B \neq A$ of the tree is the initial node of exactly one edge, (b) no edge departs from A, and (c) no circles are encountered, that is, there is no directed path from a node to itself (see Figure 15.6b).

The idea of the algorithm is the identification of directed trees in a graph, corresponding to the points of X, so that each of them corresponds to a cluster.

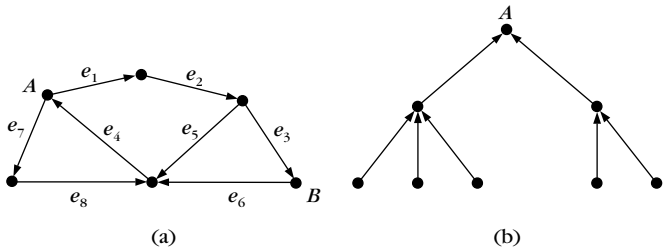

FIGURE 15.6

(a) A directed graph. (b) A directed tree.

The vectors of X are processed sequentially. For each point $\boldsymbol{x}_i$, we define its neighborhood as

$$\rho_i(\theta) = \{\boldsymbol{x}_j \in X : d(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq \theta, \boldsymbol{x}_j \neq \boldsymbol{x}_i\} \tag{15.6}$$

where θ determines the size of the neighborhood and $d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is the distance between the corresponding vectors of X. Let $n_i = |\rho_i(\theta)|$ be the number of points of X lying in $\rho_i(\theta)$. Finally, let $g_{ij} = (n_j - n_i)/d(\boldsymbol{x}_i, \boldsymbol{x}_j)$. This quantity will be used to determine the position of the point $\boldsymbol{x}_i$ in a directed tree. After these definitions, the clustering algorithm may be stated as follows.

Clustering Algorithm Based on Directed Trees

Set θ to a specific value.
Determine $n_i, i = 1, \ldots, N$.
Compute $g_{ij}, i, j = 1, \ldots, N, i \neq j$.
For $i = 1$ to N

- if $n_i = 0$ then
  - $\boldsymbol{x}_i$ is the root of a new directed tree.
- else
  - Determine $\boldsymbol{x}_r$ such that $g_{ir} = \max_{\boldsymbol{x}_j \in \rho_i(\theta)} g_{ij}$.
  - If $g_{ir} < 0$ then
    - $\boldsymbol{x}_i$ is the root of a new directed tree.
  - Else if $g_{ir} > 0$ then
    - $\boldsymbol{x}_r$ is the parent of $\boldsymbol{x}_i$.[1]
  - Else if $g_{ir} = 0$ then

[1] We say that $\boldsymbol{x}_r$ is the parent of $\boldsymbol{x}_i$ if there exists a directed edge from $\boldsymbol{x}_i$ to $\boldsymbol{x}_r$.

- Define $T_i = \{\boldsymbol{x}_j : \boldsymbol{x}_j \in \rho_i(\theta), g_{ij} = 0\}$.
- Eliminate all the elements $\boldsymbol{x}_j \in T_i$, for which there exists a directed path from $\boldsymbol{x}_j$ to $\boldsymbol{x}_i$.
- If the resulting T_i is empty then
  - $\boldsymbol{x}_i$ is the root of a new directed tree.
- Else
  - The parent of $\boldsymbol{x}_i$ is $\boldsymbol{x}_q$ such that $d(\boldsymbol{x}_i, \boldsymbol{x}_q) = \min_{x_s \in T_i} d(\boldsymbol{x}_i, \boldsymbol{x}_s)$.
- End {if}
- End {if}

End {for}

The directed trees formed by these steps identify the clusters.

It is clear from the preceding algorithm that the root, say $\boldsymbol{x}_i$, of a directed tree has the largest n_i among the points lying in $\rho_i(\theta)$. That is, among the points lying in $\rho_i(\theta)$, $\boldsymbol{x}_i$ is the point with the most dense neighborhood. It should be pointed out that the branch that handles the case in which $g_{ir} = 0$ ensures that no circles will occur. Also, this algorithm is sensitive to the order in which the vectors are processed. Finally, it can be shown that for proper values of θ and large N this scheme behaves as a mode-seeking algorithm [Koon 76].

Example 15.2

Consider Figure 15.7. The size of the edge of the grid is 1 and the diagonal of a small rectangle equals $\sqrt{2}$. Also, let $X = \{\boldsymbol{x}_i, i = 1, \ldots, 11\}$. It is clear that the vectors of X form two well-separated clusters. Let $\theta = 1.1$. Applying the preceding algorithm on X, we determine the two directed trees shown in Figure 15.7. However, if we present $\boldsymbol{x}_5$ before $\boldsymbol{x}_4$, the left-directed tree will have a different root. Nevertheless, the final results of the algorithm remain the same in this (rather easy) case.

15.2.4 Spectral Clustering

Spectral clustering is a class of graph-based techniques that unravel the structural properties of a graph using information conveyed by the spectral decomposition (eigendecomposition) of an associated matrix. The elements of this matrix code the underlying similarities among the nodes (data points) of the graph. Spectral clustering algorithms have attracted a lot of interest over the last years. Their high popularity springs from their improved performance in a number of applications, where several classical techniques fail and also from a number of interesting related theoretical issues. Among the earlier works on spectral clustering are [Scot 90] and [Hage 92].

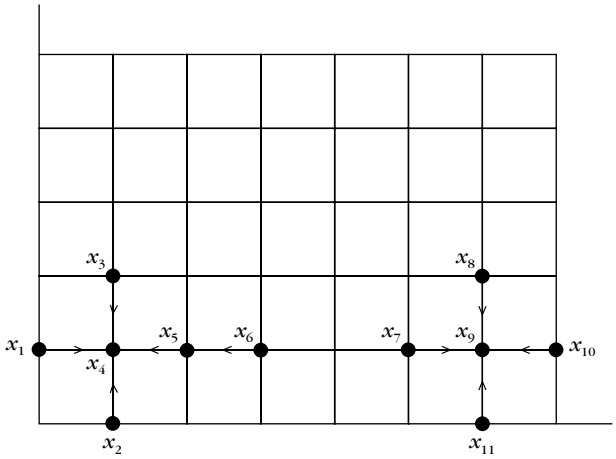

FIGURE 15.7

The setup of Example 15.2.

In this book, we will focus on the simplest task of bi-partitioning a given data set, X, into two clusters, A and B. Generalizations will be discussed later on. Let $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} \subset \mathcal{R}^l$. In graph-based clustering methods the following steps are in order:

- Construct a graph $G(V, E)$, where each point of the graph corresponds to a point $\boldsymbol{x}_i$, $i = 1, 2, \ldots, N$, of X. We will further assume that G is undirected and connected (Section 13.2.5).
- Weigh each one of the edges of the graph, e_{ij}, by a weight $W(i,j)$ that measures the similarity between the respective nodes, v_i, v_j in G[2]. The set of weights defines the proximity (sometimes called affinity) $N \times N$ matrix W with elements

$$W \equiv [W(i,j)], \quad i,j = 1, 2, \ldots, N$$

The proximity matrix is assumed to be symmetric, that is, $W(i,j) = W(j,i)$. The choice of the weights is up to the user and it is a problem dependent task. A common choice is

$$W(i,j) = \begin{cases} \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right), & \text{if } ||\boldsymbol{x}_i - \boldsymbol{x}_j|| < \epsilon \\ 0, & \text{otherwise} \end{cases}$$

where ϵ is a user-defined constant and $||\cdot||$ is the Euclidean norm.

[2] For notational convenience in some places we use i instead of v_i.

Choosing the proximity matrix is not always an "innocent" task. A right choice can have a significant improvement on the obtained results. For example, in the previous Gaussian kernel case, determining σ is a pivotal issue that significantly influences the resulting clustering. This is also a problem that we have faced with all kernel methods considered in previous chapters. A naive approach is to work with different values of σ and choose the one that is best according to a predetermined criterion [Ng 01]. The issue of how one can construct a good proximity matrix is treated in [Fisc 05, Weis 99]. There is also stated that a good proximity matrix must have a structure which is as close to a block diagonal as possible.

By the definition of clustering, $A \cup B = X$ and $A \cap B = \emptyset$. Once a weighted graph has been formed, the second phase in any graph-based clustering method consists of the following two steps:

- Choose an appropriate clustering criterion for the partitioning of the graph.
- Adopt an efficient algorithmic scheme to perform the partitioning, in accordance with the previously adopted clustering criterion.

A commonly used clustering criterion is the so called *cut* [Wu 93]. If A and B are the resulting clusters, the associated *cut* is defined as

$$cut(A, B) = \sum_{i \in A,\, j \in B} W(i, j) \tag{15.7}$$

Selecting A and B so that the respective $cut(A, B)$ is minimized means that the set of edges, connecting nodes in A with nodes in B, have the minimum sum of weights, that is, points in A and B have the least similarity compared to any other bi-partitioning. However, this simple criterion turns out to form clusters of small size of isolated points (least similar with the rest of the nodes). This is illustrated in Figure 15.8. The minimum *cut* criterion would result in the two clusters separated by the dotted line, although the partition by the full line seems to be a more natural partitioning.

To remedy the previous drawback, the *normalized cut* criterion has been suggested in [Shi 00]. This is one of the most popular criteria used in spectral clustering.

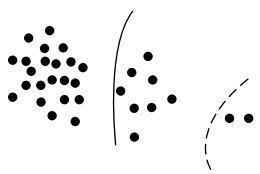

FIGURE 15.8

The cut criterion has the tendency to form small clusters of isolated points, as for example the two points separated by the dotted line. A more natural clustering for this case results by the full line.

The driving force behind this criterion is to minimize the *cut* and at the same time trying to keep the sizes of the clusters large. To this end, for each node, $v_i \in V$ in the graph G, define the index

$$D_{ii} = \sum_{j \in V} W(i,j) \tag{15.8}$$

This is an index measuring the "significance" of a node, v_i, $i = 1, 2, \ldots, N$. The higher the value of D_{ii} is, the higher the similarity of the respective ith node with respect to the rest of the nodes. A low D_{ii} value indicates an isolated (remote) point. Given a cluster A, a measure of the total significance of the points in A is given by the following index

$$V(A) = \sum_{i \in A} D_{ii} = \sum_{i \in A,\, j \in V} W(i,j) \tag{15.9}$$

where $V(A)$ is sometimes known as the *volume* or the *degree* of A. It is obvious that small and isolated clusters will have a small $V(\cdot)$. The *normalized cut* between two clusters A, B is defined as

$$Ncut(A,B) = \frac{cut(A,B)}{V(A)} + \frac{cut(A,B)}{V(B)} \tag{15.10}$$

It is easy to see that a small cluster, for example, A, will result in a large value (close to one) for the previous ratio, since in such a case $cut(A,B)$ will be a large percentage of $V(A)$.

Minimization of the $Ncut(A,B)$ turns out to be an NP-hard task. To alleviate this difficulty, we will reformulate the problem to bring it in a form that allows an efficient approximate solution. To this end define ([Belk 03])

$$y_i = \begin{cases} \frac{1}{V(A)}, & \text{if } i \in A \\ -\frac{1}{V(B)}, & \text{if } i \in B \end{cases} \tag{15.11}$$

$$\boldsymbol{y} = [y_1, y_2, \ldots, y_N]^T$$

In words, each y_i can be thought of as a cluster indicator of the corresponding point $\boldsymbol{x}_i$, $i = 1, 2, \ldots, N$. Taking into account the definitions in (15.11), it is straightforward to see that (see also Section 6.7)

$$\begin{aligned} \boldsymbol{y}^T L \boldsymbol{y} &= \frac{1}{2} \sum_{i \in V} \sum_{j \in V} (y_i - y_j)^2 W(i,j) \\ &= \sum_{i \in A} \sum_{j \in B} \left(\frac{1}{V(A)} + \frac{1}{V(B)} \right)^2 cut(A,B) \\ &\propto \left(\frac{1}{V(A)} + \frac{1}{V(B)} \right)^2 cut(A,B) \end{aligned} \tag{15.12}$$

since the contribution of $y_i - y_j$ is zero for points in the same cluster. The symbol $\propto$ denotes proportionality and

$$L = D - W, \; D \equiv \text{diag}\{D_{ii}\}$$

is the Laplacian matrix of the graph (Section 6.7). Matrix D is diagonal with the elements D_{ii} along the main diagonal. In addition we have that

$$\begin{aligned} \boldsymbol{y}^T D \boldsymbol{y} &= \sum_{i \in A} y_i^2 D_{ii} + \sum_{j \in B} y_j^2 D_{jj} = \frac{1}{V(A)^2} V(A) + \frac{1}{V(B)^2} V(B) \\ &= \frac{1}{V(A)} + \frac{1}{V(B)} \end{aligned} \tag{15.13}$$

Combining (15.12) and (15.13), it turns out that minimizing $Ncut(A, B)$ is equivalent with minimizing

$$J = \frac{\boldsymbol{y}^T L \boldsymbol{y}}{\boldsymbol{y}^T D \boldsymbol{y}} \tag{15.14}$$

subject to the constraint that $y_i \in \{\frac{1}{V(A)}, -\frac{1}{V(B)}\}$. Furthermore, direct substitution of the definitions of the involved quantities results in

$$\boldsymbol{y}^T D \mathbf{1} = 0 \tag{15.15}$$

where $\mathbf{1}$ is the N-dimensional vector with all its elements being equal to 1. In order to overcome the NP-hard nature of the original task we will solve, instead, the relaxed problem of minimizing (15.14) subject to the constraint in (15.15). The unknown variables $y_i, \; i = 1, 2, \ldots, N$, are now assumed to lie on the real axis. We are already very close to a well known optimization task. Define

$$\boldsymbol{z} \equiv D^{1/2} \boldsymbol{y}$$

Then (15.14) becomes

$$J = \frac{\boldsymbol{z}^T \tilde{L} \boldsymbol{z}}{\boldsymbol{z}^T \boldsymbol{z}} \tag{15.16}$$

and the constraint in (15.15)

$$\boldsymbol{z}^T D^{1/2} \mathbf{1} = 0 \tag{15.17}$$

where $\tilde{L} \equiv D^{-1/2} L D^{-1/2}$ and it is known as the *normalized graph Laplacian* matrix. It can easily be shown that $\tilde{L}$ has the following properties (see also, Section 6.7)

- It is symmetric and nonnegative definite. Thus, all its eigenvalues are nonnegative and the corresponding eigenvectors are orthogonal to each other (see Appendix B).

- It can easily be checked out that $D^{1/2}\mathbf{1}$ is an eigenvector corresponding to a zero eigenvalue, that is,

$$\tilde{L}D^{1/2}\mathbf{1} = 0$$

  Obviously, the zero eigenvalue is the smallest one of $\tilde{L}$, due to its nonnegative definite nature. As it was stated in Section 6.7, if the graph is connected then there is one eigenvector associated with the zero eigenvalue. This is an assumption which is adopted here.

We have by now all the ingredients to perform the final optimization. Observe that the ratio in (15.16) is the celebrated Rayleigh quotient. Recall from linear algebra, for example, [Golu 89], that

- The smallest value of the quotient, with respect to $\boldsymbol{z}$, is equal to the smallest eigenvalue of $\tilde{L}$ and it occurs for $\boldsymbol{z}$ equal to the eigenvector corresponding to this (smallest) eigenvalue.
- If we constraint the solution to be orthogonal to all eigenvectors associated with the j smallest eigenvalues, $\lambda_0 \leq \lambda_1 \leq \cdots \leq \lambda_{j-1}$, the Rayleigh quotient is minimized by the eigenvector corresponding to the next smallest eigenvalue, λ_j, and the minimum value is equal to λ_j.

Taking into account a) the orthogonality condition in the constraint (15.17) and b) the fact that $D^{1/2}\mathbf{1}$ is the eigenvector corresponding to the smallest eigenvalue $\lambda_0 = 0$, we end up that:

The optimal solution vector $\boldsymbol{z}$ minimizing the Rayleigh quotient in (15.16), subject to the constraint (15.17), is the eigenvector corresponding to the second smallest eigenvalue of $\tilde{L}$.

We are now ready to summarize the steps for our spectral clustering algorithm.

1. Given a set of points, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$, set up the weighted graph $G(V, E)$. Form the proximity matrix W by adopting a similarity rule.
2. Form the matrices $D, L = D - W$ and $\tilde{L}$. Perform the eigenanalysis

$$\tilde{L}\boldsymbol{z} = \lambda\boldsymbol{z}$$

   of the normalized Laplacian matrix $\tilde{L}$. Compute the eigenvector $\boldsymbol{z}_1$ corresponding to the second smallest eigenvalue λ_1. Compute the vector

$$\boldsymbol{y} = D^{-1/2}\boldsymbol{z}_1$$

3. Discretize the components of $\boldsymbol{y}$ according to a threshold value.

The final step is necessary since the components of the obtained solution are real-valued and our required solution is discrete. To this goal, different techniques can

be applied. For example, the threshold can be taken to be equal to zero. Another choice is to adopt the median value of the components of the optimum eigenvector. An alternative approach would be to select the threshold value that results in the minimum *cut* value.

The eigenanalysis of an $N \times N$ matrix, using a general purpose solver, amounts to $O(N^3)$ operations. Thus, for large number of data points, this may be prohibitive for some applications. However, for most of the practical applications, the resulting graph is only locally connected, and the proximity matrix is *sparse*. Moreover, only the smallest eigenvalues/eigenvectors are required and also the accuracy is not of major issue, since the solution is to be discretized. In such a setting, the efficient Lanczos algorithm (e.g., [Golu 89]) can be mobilized and the computational requirements drop down to approximately $O(N^{3/2})$.

So far, the partition of a data set into two clusters has been considered. If more clusters are expected, the scheme can be used in a hierarchical mode, where, at each step, each one of the resulting clusters is partitioned further into two clusters. This is continued until a prespecified criterion is satisfied. In [Shi 00] it is suggested that the third smallest eigenvalue can be used to sub-partition the first two clusters and so on. However, this procedure tends to become unreliable due to approximation errors.

In our discussion, so far, we focused on a specific clustering criterion, that is, the normalized cut, in order to present the basic philosophy behind the spectral clustering techniques. No doubt, a number of other criteria have been proposed in the related literature. For example, the ratio cut ([Chan 94]) is defined as

$$Rcut(A, B) = \frac{cut(A, B)}{|A|} + \frac{cut(A, B)}{|B|}$$

In [Kann 00] the Cheeger constant is used as the partition criterion, that is,

$$h_G = \frac{cut(A, B)}{\min(V(A), V(B))}$$

For each criterion, a different eigendecomposition problem results, each with its advantages and drawbacks. In [Verm 03], a review and a comparative study of a number of popular spectral clustering algorithms is presented. A comparative and insightful study of a number of spectral clustering algorithms is provided in [Weis 99].

The literature on spectral clustering is large and it is beyond the purpose of this section to cover it in detail. Besides the criteria mentioned before, other approaches to spectral clustering have also been proposed. For example, in [Meil 00] the pairwise similarities are interpreted as edge flows in a Markov random walk leading to a probabilistic interpretation of spectral clustering. In [Xian 08], the issues on how to determine the number of clusters and how to deal with noisy and sparse data are considered. They tackle this problem via a data-driven approach that selects the most relevant eigenvectors, which provide information about the natural grouping

of the data. Also, in [Jens 04] a spectral clustering algorithm based on an information theoretic framework is discussed. For more information, besides the references given before, the interested reader may consult, for example, [Chun 97, vonL 07].

Spectral clustering has been used in a number of applications such as image segmentation and motion tracking [Shi 00, Qiu 07], circuit layout [Chan 94], gene expression [Kann 00], machine learning [Ng 01], load balancing [Hend 93].

Remarks

- For those of the readers who have also covered Section 6.7, it is obvious to observe the close resemblance between spectral clustering and the dimensionality reduction methods that preserve locality (Laplacian eigenmaps and LLE). Attempting to preserve neighborhood information, while projecting in the low dimensional subspace, may be interpreted as imposing a "soft" clustering on the data, [Belk 03]. In spectral clustering, one can look at each y_i as the nonlinear mapping onto the real axis of the data point, $\boldsymbol{x}_i$, $i = 1, 2, \ldots, N$, and a hard clustering is obtained after discretization. Moreover, as Eq. (15.12) suggests, the cluster indicators y_i are "forced" to take similar values for closely located points. This is a consequence of the minimization task in Eq. (15.14) and of the fact that far away points result in small or zero values of the weights.
- A very interesting result, that ties the "old" and the "new", is the establishment of the mathematical equivalence between a weighted form of the kernelized k-means objective and a general weighted graph clustering objective. The normalized cut and the ratio cut objectives fall under this category. Thus, a kernelized version of the classical k-means algorithm can be employed to solve the task instead of a matrix eigendecomposition. This may have certain computational advantages, especially for large problems. However, spectral decomposition computes the global optimal, in contrast to the k-means algorithm that may be trapped in local minima. Such issues and a novel algorithm, exploiting this equivalence, are discussed in, for example, [Dill 07, Zass-05].

Example 15.3

In the example shown in Figure 15.9a, two concentric circularly shaped clusters are shown. The first cluster is spread around the circle of radius equal to 3 centered at $(0, 0)$, while the second cluster is spread around the circle of radius equal to 6 and also centered at $(0, 0)$. The spectral clustering algorithm with $\sigma^2 = 2$ and $\varepsilon = 2$, using the normalized cut criterion, is applied on the previous data set and the results are shown in Figure 15.9b. Clearly, the algorithm identifies successfully the two clusters. On the contrary, the k-means algorithm fails to identify the clusters successfully, as shown in Figure 15.9c. Recall that the k-means algorithm has the tendency of recovering compact clusters.

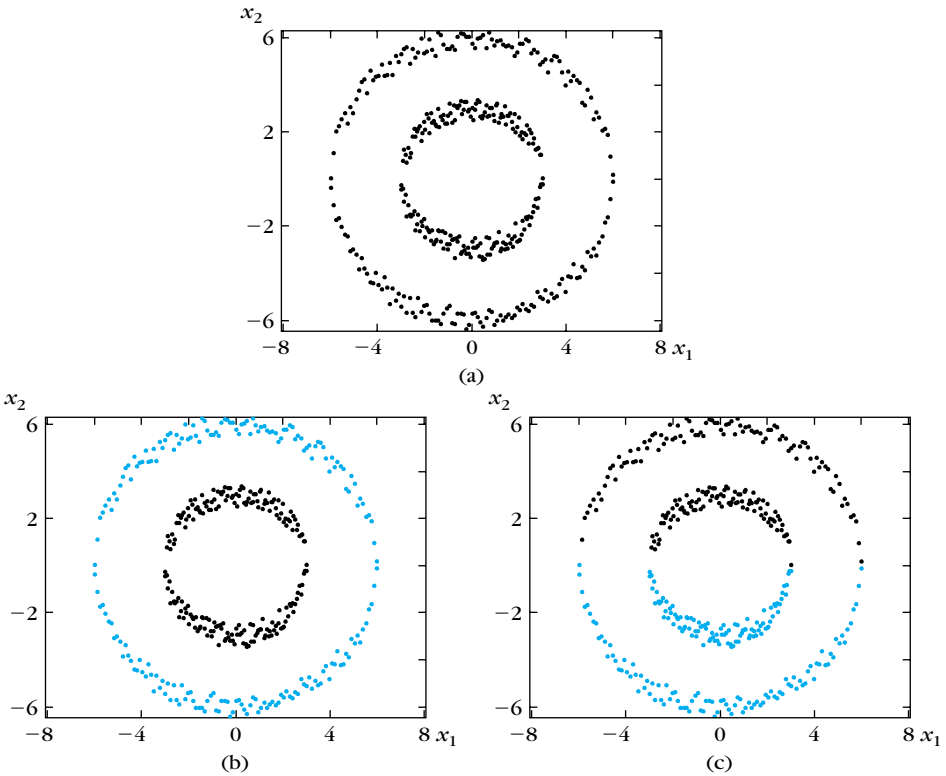

FIGURE 15.9

(a) The data set. (b) The two clusters (denoted by different colors) obtained by the spectral clustering algorithm. (c) The two clusters obtained by the k-means algorithm.

15.3 COMPETITIVE LEARNING ALGORITHMS

These algorithms employ a set of representatives $\boldsymbol{w}_j, j = 1, \ldots, m$.[3] *Their goal is to move each of them to regions of the vector space that are "dense" in vectors of* X. The representatives are updated each time a new vector $\boldsymbol{x} \in X$ is presented to the algorithm. Algorithms of this type are called *pattern mode algorithms*. This is a point of differentiation from the hard clustering algorithms discussed in Chapter 14. There the updating of the representatives takes place after the presentation of *all* the vectors of X to the algorithm. Algorithms of the latter kind are also called *batch mode algorithms*. It must be emphasized that competitive learning algorithms do not necessarily stem from the optimization of a cost function.

[3] We use $\boldsymbol{w}_j$ here instead of $\boldsymbol{\theta}_j$ to comply with the notation usually adopted for this type of schemes.

The general idea is very simple. When a vector $\boldsymbol{x}$ is presented to the algorithm, all representatives *compete* with each other. The winner of this competition is the representative that lies closer (according to some distance measure) to $\boldsymbol{x}$. Then, the winner is updated so as to move toward $\boldsymbol{x}$, while the losers either remain unchanged or are updated toward $\boldsymbol{x}$ but at a much slower rate.

Although, in most of the cases, $\boldsymbol{w}_j$'s are points in the l-dimensional space, other choices are also possible. For example, the representatives may be hyperplanes [Likh 97]. In the sequel, we consider only the case in which $\boldsymbol{w}_j$'s are points in the l-dimensional space.

Let t be the current iteration and $t_{\max}$ the maximum allowable number of iterations. Also, let m be the current number, m_{init} the initial number, and $m_{\max}$ the maximum allowable number of clusters (representatives). Then, a generalized competitive learning scheme (GCLS) may be stated as follows.

Generalized Competitive Learning Scheme (GCLS)

- $t = 0$
- $m = m_{\text{init}}$
- (A) Initialize any other necessary parameters (depending on the specific scheme).
- Repeat
  - $t = t + 1$
  - Present the next $\boldsymbol{x} \in X$ to the algorithm.
  - (B) Determine the winning representative $\boldsymbol{w}_j$.
  - (C) If (($\boldsymbol{x}$ is not "similar" to $\boldsymbol{w}_j$) OR (other condition)) AND ($m < m_{\max}$) then
    - $m = m + 1$
    - $\boldsymbol{w}_m = \boldsymbol{x}$
  - Else
    - (D) *Parameter updating*

$$\boldsymbol{w}_j(t) = \begin{cases} \boldsymbol{w}_j(t-1) + \eta h(\boldsymbol{x}, \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ is the winner} \\ \boldsymbol{w}_j(t-1) + \eta' h(\boldsymbol{x}, \boldsymbol{w}_j(t-1)), & \text{otherwise} \end{cases} \tag{15.18}$$

  - End
- (E) Until (convergence has occurred) OR ($t > t_{\max}$)
- Identify the clusters represented by $\boldsymbol{w}_j$'s, by assigning each vector, $\boldsymbol{x} \in X$, to the cluster that corresponds to the representative closest to $\boldsymbol{x}$.

The function $h(\boldsymbol{x}, \boldsymbol{w}_i)$ is an appropriately defined function. Also, η and η' are the *learning rates* controlling the update of the winner and the losers, respectively.

The parameter η' may be different for different losers. The similarity between a vector $\boldsymbol{x}$ and a representative $\boldsymbol{w}_j$ may be characterized by a threshold of similarity Θ; that is, if $d(\boldsymbol{x}, \boldsymbol{w}_j) > \Theta$, for some distance measure, $\boldsymbol{x}$ and $\boldsymbol{w}_j$ are considered as dissimilar and $\boldsymbol{w}_j$ cannot be used to represent $\boldsymbol{x}$ accurately. It is clear that improper choice of the value of Θ may lead to misleading results.

Termination of the algorithm is achieved via our familiar criterion $\|\boldsymbol{W}(t) - \boldsymbol{W}(t-1)\| < \varepsilon$, where $\boldsymbol{W} = [\boldsymbol{w}_1^T, \ldots, \boldsymbol{w}_m^T]^T$.

With appropriate choices of the parts (A), (B), (C), and (D), most of the competitive learning algorithms may be viewed as special cases of the GCLS. In the sequel, unless otherwise stated, we use the Euclidean distance, although other distances may also be used.

15.3.1 Basic Competitive Learning Algorithm

In this algorithm $m = m_{\text{init}} = m_{\max}$; that is, the number of representatives is constant. Thus, condition (C) is never satisfied. Also, no other parameters are necessary, so part (A) is omitted. The determination of the winning representative (part (B)) is carried out using the following rule.

- The representative $\boldsymbol{w}_j$ is the winner on $\boldsymbol{x}$ if

$$d(\boldsymbol{x}, \boldsymbol{w}_j) = \min_{k=1,\ldots,m} d(\boldsymbol{x}, \boldsymbol{w}_k)$$

Besides the Euclidean distance, other distances may also be used, depending on the application at hand. For example, in [Ahal 90], the Itakura–Saito distortion is proposed when dealing with speech coding applications in the clustering framework. Moreover, similarity measures may also be used (see, e.g., [Fu 93]). In this case, the min operator in the preceding relation is replaced by the max operator. Finally, the updating of the representatives (part (D)) is carried out as follows:

$$\boldsymbol{w}_j(t) = \begin{cases} \boldsymbol{w}_j(t-1) + \eta(\boldsymbol{x} - \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ is the winner} \\ \boldsymbol{w}_j(t-1), & \text{otherwise} \end{cases} \tag{15.19}$$

where η is the learning rate and takes values in $[0, 1]$. According to this algorithm, the losers remain unchanged. On the other hand, the winner $\boldsymbol{w}_j(t)$ moves toward $\boldsymbol{x}$. The size of the movement depends on η. In the extreme case where $\eta = 0$, no updating takes place. On the other hand, if $\eta = 1$, the winning representative is placed on $\boldsymbol{x}$. For all other choices of η, the new value of the winner lies in the line segment formed by $\boldsymbol{w}_j(t-1)$ and $\boldsymbol{x}$.

It is clear that this algorithm requires an accurate determination of the number of representatives; that is, knowledge of the number of clusters is required. Another related problem that may arise is associated with the initialization of $\boldsymbol{w}_j$'s. If a representative is initialized far away from its closest vector in X,[4] it will never win.

[4] More specifically, if it lies far away from the convex hull defined by the vectors of X.

Thus, the vectors of X will be represented by the remaining representatives. An easy way to avoid this situation is to initialize all representatives using vectors of X.

In the special case in which the vectors are always presented in the same order, that is, $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N, \ldots$, and under the assumption that after an iteration t_0 each representative wins on the same vectors, which is reasonable at least for the case where well-separated clusters are formed by the vectors of X, it can be shown that each representative converges to a weighted mean of the vectors it represents [Kout 95].

This algorithm has also been studied for a variable learning rate (e.g., [Likh 97]). Typical constraints for $\eta(t)$ in this case are:

- $\eta(t)$ is a positive decreasing sequence and $\eta(t) \to 0$.
- $\sum_{t=0}^{\infty} \eta(t) = \infty$.
- $\sum_{t=0}^{\infty} \eta^r(t) < +\infty$, for $r > 1$.

Note that these constraints are very similar to those required by the Robbins–Monro schemes, discussed in Section 3.4.2. This is not a coincidence. Let us consider for example the trivial case of a single representative ($m = 1$). If $\eta = \eta(t)$, the updating equation may be seen as the Robbins–Monro iteration for solving the problem

$$E[h(\boldsymbol{x}, \boldsymbol{w})] = 0$$

where $h(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x} - \boldsymbol{w}$.

Finally, competitive learning algorithms for binary-valued vectors are discussed in [Rume 86, Mals 73].

15.3.2 Leaky Learning Algorithm

This algorithm is the same as the basic competitive learning algorithm except for the updating equation of the representatives, which is

$$\boldsymbol{w}_j(t) = \begin{cases} \boldsymbol{w}_j(t-1) + \eta_w(\boldsymbol{x} - \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ is the winner} \\ \boldsymbol{w}_j(t-1) + \eta_l(\boldsymbol{x} - \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ is a loser} \end{cases} \tag{15.20}$$

where η_w and η_l are the learning rates in $[0, 1]$ and $\eta_w \gg \eta_l$. The basic competitive learning scheme may be viewed as a special case of the leaky learning scheme, for $\eta_l = 0$. In the general case where η_w and η_l are both positive, all representatives move toward $\boldsymbol{x}$. However, the losers move toward $\boldsymbol{x}$ at a much slower rate than the winner.

This algorithm does not suffer from the problem of poor initialization of the representatives. This is because the second branch of (15.20) ensures that even if some representatives are initialized away from their closest vectors of X, they will eventually come closer to the region where the vectors of X are located.

An algorithm in the same spirit is the *neural-gas* algorithm. However, in this case $n_w = \varepsilon$ and $\eta_l = \varepsilon g(k_j(\boldsymbol{x}, \boldsymbol{w}_j(t-1)))$, where $\varepsilon \in [0, 1]$ is the step size of the updating, $k_j(\boldsymbol{x}, \boldsymbol{w}_j(t-1))$ is the number of representatives that lie closer to $\boldsymbol{x}$ than

$\boldsymbol{w}_j(t-1)$ and $g(\cdot)$ is a function that takes the value 1 for $k_j(\boldsymbol{x}, \boldsymbol{w}_j(t-1)) = 0$ and decays to zero as $k_j(\boldsymbol{x}, \boldsymbol{w}_j(t-1))$ increases. It is worth noting that this algorithm results from the optimization of a cost function via a gradient descent technique ([Mart 93]).

15.3.3 Conscientious Competitive Learning Algorithms

Another way to utilize the representative power of $\boldsymbol{w}_j$'s is to discourage a representative from winning if it has won many times in the past. This is achieved by assigning a "conscience" to each representative. Several models of conscience have been proposed in the literature (e.g., [Gros 76a, Gros 76b, Gros 87, Hech 88, Chen 94, Uchi 94]).

Perhaps the simplest way to implement this idea is to equip each representative, $\boldsymbol{w}_j, j = 1, \ldots, m$, with a counter f_j, that counts the times that $\boldsymbol{w}_j$ wins. One way to proceed is the following [Ahal 90]. At the initialization stage (part (A)) we set $f_j = 1, j = 1, \ldots, m$. We define

$$d^*(\boldsymbol{x}, \boldsymbol{w}_j) = d(\boldsymbol{x}, \boldsymbol{w}_j) f_j$$

and part (B) becomes the following:

- The representative $\boldsymbol{w}_j$ is the winner on $\boldsymbol{x}$ if

$$d^*(\boldsymbol{x}, \boldsymbol{w}_j) = \min_{k=1,\ldots,m} d^*(\boldsymbol{x}, \boldsymbol{w}_k)$$

- $f_j(t) = f_j(t-1) + 1$.

This setup ensures that the distance is penalized to discourage representatives that have won many times. The parts (C) and (D) are the same as their corresponding parts of the basic competitive learning algorithm, and also $m = m_{\text{init}} = m_{\max}$.

An alternative way is to utilize f_j via the equation [Chou 97]

$$f_j = f_j + d(\boldsymbol{x}, \boldsymbol{w}_j)$$

Other schemes of this kind may be found in [Ueda 94, Zhu 94, Butl 96, Chou 97].

A different approach is followed in [Chen 94]. Here, in part (A), all f_j's are initialized to $1/m$. We define $d^*(\boldsymbol{x}, \boldsymbol{w}_j)$ as

$$d^*(\boldsymbol{x}, \boldsymbol{w}_j) = d(\boldsymbol{x}, \boldsymbol{w}_j) + \gamma(f_j - 1/m)$$

where γ is a conscience parameter. Letting $z_j(\boldsymbol{x})$ be 1 if $\boldsymbol{w}_j$ wins on $\boldsymbol{x}$ and 0 otherwise, part (B) of the algorithm becomes

- The representative $\boldsymbol{w}_j$ is the winner on $\boldsymbol{x}$ if

$$d^*(\boldsymbol{x}, \boldsymbol{w}_j) = \min_{k=1,\ldots,m} d^*(\boldsymbol{x}, \boldsymbol{w}_k)$$

- $f_j(t) = f_j(t-1) + \varepsilon(z_j(\boldsymbol{x}) - f_j(t-1))$

where $0 < \varepsilon \ll 1$. As we can easily observe, f_j increases for the winner and, in contrast to the previous case, decreases for the losers. Guidelines for the choice of the appropriate values of ε and γ, as well as a version of the algorithm where the value of γ is adaptively adjusted, are discussed in [Chen 94].

15.3.4 Competitive Learning–Like Algorithms Associated with Cost Functions

The basic philosophy behind the competitive learning schemes is to move representatives toward their closest points. If we want to express this in terms of a cost function, then a possible way is the following. Let us consider the cost function

$$J(\boldsymbol{W}) = \frac{1}{2m} \sum_{i=1}^{N} \sum_{j=1}^{m} z_j(\boldsymbol{x}_i) \|\boldsymbol{x}_i - \boldsymbol{w}_j\|^2 \tag{15.21}$$

where $\boldsymbol{W} = [\boldsymbol{w}_1^T, \ldots, \boldsymbol{w}_m^T]^T$ and $z_j(\boldsymbol{x}) = 1$, if $\boldsymbol{w}_j$ lies closer to $\boldsymbol{x}$, and 0 otherwise. This is basically the cost function associated with the isodata algorithm (Chapter 14), and it is not differentiable, due to the presence of $z_j(\boldsymbol{x})$. One way to overcome the problem of differentiability of $J(\boldsymbol{W})$ is to smooth $z_j(\boldsymbol{x})$. This implies that the concept of the competition is abandoned. Instead, each representative is updated in proportion to its distance from $\boldsymbol{x}$.

One way to smooth $z_j(\boldsymbol{x})$ is to redefine it as

$$z_j(\boldsymbol{x}) = \frac{\|\boldsymbol{x} - \boldsymbol{w}_j\|^{-2}}{\sum_{r=1}^{m} \|\boldsymbol{x} - \boldsymbol{w}_r\|^{-2}}, \quad j = 1, \ldots, m \tag{15.22}$$

where $\|\cdot\|$ is the Euclidean distance between two vectors. Clearly, $z_j(\boldsymbol{x})$ is no longer strictly equal to 0 or 1 but lies in $[0, 1]$. More specifically, the closer the $\boldsymbol{w}_j$ to $\boldsymbol{x}$, the larger the $z_j(\boldsymbol{x})$.

Using the preceding definition and after some rearrangements, $J(\boldsymbol{W})$ becomes

$$J(\boldsymbol{W}) = \frac{1}{2} \sum_{i=1}^{N} \left(\sum_{j=1}^{m} \|\boldsymbol{x}_i - \boldsymbol{w}_j\|^{-2} \right)^{-1} \tag{15.23}$$

The gradient of $J(\boldsymbol{W})$ with respect to $\boldsymbol{w}_k$, $\partial J / \partial \boldsymbol{w}_k$, after some algebra, becomes

$$\frac{\partial J}{\partial \boldsymbol{w}_k} = -\sum_{i=1}^{N} z_k^2(\boldsymbol{x}_i)(\boldsymbol{x}_i - \boldsymbol{w}_k) \tag{15.24}$$

In the context of the gradient descent algorithms and using the "instantaneous" value of the gradient, as it was the case with the backpropagation algorithm, the following updating algorithm results.

$$\boldsymbol{w}_k(t) = \boldsymbol{w}_k(t-1) + \eta(t) z_k^2(\boldsymbol{x})(\boldsymbol{x} - \boldsymbol{w}_k(t-1)), \quad k = 1, \ldots, m \tag{15.25}$$

where $\boldsymbol{x}$ is the vector currently presented to the algorithm.

Notice that in this scheme *all* representatives are updated in proportion to their distance from $\boldsymbol{x}$. Thus, by smoothing $z_k(\boldsymbol{x})$, we end up with algorithms that are competitive in a wider sense, for which general tools may apply for the establishment of their convergence properties.

Alternative choices of $z_j(\boldsymbol{x})$ and $J(\boldsymbol{W})$, leading to more general algorithmic schemes, are given in [Masu 93].

15.3.5 Self-Organizing Maps

So far, we have implicitly assumed that the representatives, $\boldsymbol{w} \in \mathcal{R}^l$, are not interrelated. We will now remove this assumption. Furthermore, the representatives will be "forced" to be *topologically ordered* in the one-dimensional or two-dimensional space. In other words, each $\boldsymbol{w}$ is parameterized in terms of an integer pair (for the two-dimensional case) (i, j), $i = 1, 2, \ldots, I$, $j = 1, 2, \ldots, J$, where IJ is the number of representatives. In this way, a *grid* of nodes is defined. The goal in this section is to place the representatives so that data points that lie close in the $\mathcal{R}^l$ space to be represented by representatives that lie close in the grid. Alternatively, the one-dimensional or two-dimensional grid can be seen as a *map* where we require that different "dense" in data regions, in the data space, are mapped to different regions in the map.

The concept of the topological ordering implies the adoption of a *topological distance* between two representatives. For example, if (i_1, j_1) and (i_2, j_2) denote the positions of two representatives in a (two-dimensional) grid of nodes, their topological distance may be defined as the l_1 distance (Section 11.2.2) between the two integer pairs. In this respect, in Figure 15.10a, $\boldsymbol{w}_r$ and $\boldsymbol{w}_q$ are topologically close to each other in the two-dimensional grid, while $\boldsymbol{w}_s$ is far from both of them. Figure 15.10b shows the case of an one-dimensional grid. In the sequel, we define a *topological neighborhood* Q_j for each representative $\boldsymbol{w}_j$, which consists of representatives that are close to $\boldsymbol{w}_j$ in terms of the topological distance. Typical topological neighborhood shapes are shown in Figures 15.11a and 15.11b for the two-dimensional and the one-dimensional cases, respectively. However, for the two-dimensional case, other shapes, such as hexagonal or rhombic, may also be employed.

As it was the case with the algorithms discussed in previous sections, at each iteration step, t, a single data vector, $\boldsymbol{x}$, is presented to the algorithm. However now, when a representative $\boldsymbol{w}_j$ wins on a given data vector $\boldsymbol{x}$, *all* the representatives *inside* its respective neighborhood $Q_j(t)$ are also updated (move toward $\boldsymbol{x}$). Note that the neighborhood is allowed to change with the iteration step. After a random initialization of the representatives, it is expected that, at the beginning of the training, topologically close representatives may win on data points that may not, necessarily, lie close in the data space. However, as the training evolves, this phenomenon decays and *after convergence, topologically neighboring representatives on the grid win on vectors lying in the same region in the data space. In contrast, topologically distant representatives win on data vectors that lie in*

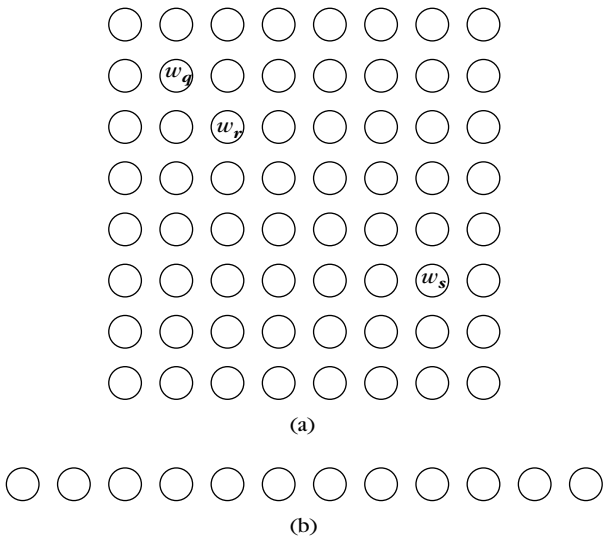

FIGURE 15.10

(a) An 8×8 two-dimensional grid of topologically ordered representatives. Adopting l_1 as the topological distance between two representatives, $\boldsymbol{w}_r$ and $\boldsymbol{w}_q$ are topologically close to each other, while $\boldsymbol{w}_s$ is distant from both of them. (b) A 1×12 one-dimensional grid.

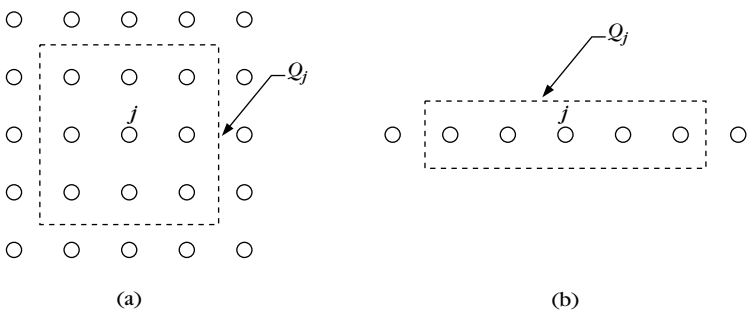

FIGURE 15.11

(a) A square-shaped 3×3 neighborhood. (b) A squared-shaped 5×1 neighborhood.

different regions in the data space. In other words, *each dense in data region in the data space is represented by a set of topologically neighboring representatives*.

The previous described approach leads to the celebrated *Kohonen self-organizing mapping (SOM) scheme* [Koho 89, Koho 95, Kask 98, Diam 07] and it has extensively been used for applications such as data visualization. In its simplest form, SOM may be viewed as a special case of the generalized competitive learning scheme (GCLS). Precisely, it is the same as the basic competitive learning algorithm

as far as parts (A), (B), and (C) are concerned. However, part (D) is different. If $\boldsymbol{w}_j$ wins on $\boldsymbol{x}$, this part becomes

$$\boldsymbol{w}_k(t) = \begin{cases} \boldsymbol{w}_k(t-1) + \eta(t)(\boldsymbol{x} - \boldsymbol{w}_k(t-1)), & \text{if } \boldsymbol{w}_k \in Q_j(t) \\ \boldsymbol{w}_k(t-1), & \text{otherwise} \end{cases} \tag{15.26}$$

where $\eta(t)$ is a variable learning rate, which is chosen to satisfy the conditions in Section 15.3.1. The choices of $\eta(t)$ and $Q_j(t)$ are crucial for the convergence of the algorithm. In practice, as t increases, $Q_j(t)$ shrinks and concentrates around $\boldsymbol{w}_j$, according to a preselected rule.

The SOM algorithm can also be seen as a constrained clustering scheme, where the representatives are encouraged to lie in a low (one or two)-dimensional manifold. This is the reason that, if the data do not lie close to such a low dimensional manifold, the performance of the method may degrade. The effect of the update in (15.26) is to move the winner as well as its neighboring representatives closer to the corresponding data point; this imposes a smooth spatial structure in the low-dimensional grid.

A suboptimal method for selecting the winner representative that may lead to computational savings is presented in [Vish 00]. Kernelized versions of SOM have appeared in [Inok 04, Macd 00].

Example 15.4

Let X be a data set consisting of 200 3-dimensional data points. The first 100 of them stem from a Gaussian distribution with mean $\boldsymbol{\mu}_1 = [0.3, 0.3, 0.3]^T$ while the rest stem from another Gaussian distribution with mean $\boldsymbol{\mu}_2 = [0.7, 0.7, 0.7]^T$. The covariance matrices of both distributions are equal to $0.01I$, where I is the 3×3 identity matrix. Clearly, the previous data vectors form two well separated clusters in the 3-dimensional space. Let C_1 denote the cluster corresponding to the first distribution and C_2 denote the cluster corresponding to the second distribution. Let us consider a SOM with a 10×10 grid of representatives. Figure 15.12a is a "snapsot" of the grid just after a random initialization of the representatives in $[0, 1]^3$. Each representative that wins on vectors from cluster C_1 is denoted by a blue circle, while each representative that wins on the vectors from cluster C_2 is denoted by a black circle. All the rest nodes of the grid are denoted by black dots. Note that representatives are spread throughout the grid, irrespective of the cluster they represent.

The results after the convergence of the SOM scheme are shown in Figure 15.12b; the representatives are concentrated in two distinctly different regions of the grid, depending on the cluster they represent.

15.3.6 Supervised Learning Vector Quantization

A supervised variant of the competitive schemes has been suggested and extensively used in the context of VQ [Koho 89, Kosk 92]. In this case, each cluster is treated as a class and the available vectors have known class labels. In this framework, let m be the number of classes. Supervised VQ *uses a set of m representatives, one*

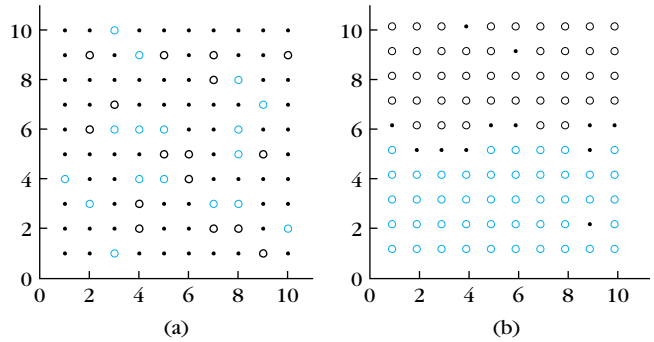

FIGURE 15.12

(a) The state of the map after the first iteration step for the case of the Example 15.4. The representatives of both clusters (black and red circles respectivley) are spread throughout the grid. (b) The state of the grid after the completion of the training. Now the clusters are represented by neighboring representatives which occupy two clearly distinct regions in the grid. Black dots are points where no representatives have been allocated by the algorithm.

for each class, and tries to place them in such a way that each class is "optimally" represented. The simplest version of the supervised VQ (also called LVQ1 [Tsyp 73]) may be derived from generalized competitive learning schemes by keeping parts (A), (B), and (C) the same as in the basic competitive learning scheme and modifying part (D) to

$$\boldsymbol{w}_j(t) = \begin{cases} \boldsymbol{w}_j(t-1) + \eta(t)(\boldsymbol{x} - \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ correctly wins on } \boldsymbol{x} \\ \boldsymbol{w}_j(t-1) - \eta(t)(\boldsymbol{x} - \boldsymbol{w}_j(t-1)), & \text{if } \boldsymbol{w}_j \text{ wrongly wins on } \boldsymbol{x} \\ \boldsymbol{w}_j(t-1), & \text{otherwise} \end{cases} \tag{15.27}$$

It is clear that the information related to the known class labels determines the direction in which $\boldsymbol{w}_j$ is moved. Specifically, we move $\boldsymbol{w}_j$ (a) toward $\boldsymbol{x}$ if $\boldsymbol{w}_j$ wins and $\boldsymbol{x}$ belongs to the jth class and (b) away from $\boldsymbol{x}$ if $\boldsymbol{w}_j$ wins and $\boldsymbol{x}$ does not belong to the jth class. In addition, all other representatives remain unaltered. Such algorithms have been used in speech recognition and OCR applications.

A variant of this scheme, where more than one representative is used to represent each class, is discussed in [Koho 89].

15.4 BINARY MORPHOLOGY CLUSTERING ALGORITHMS (BMCAs)

Algorithms of this type are suitable for cases in which clusters are not properly represented by a single representative ([Post 93, Mora 00]). The idea here is to map X to a discrete set S that facilitates the clustering procedure and then use the identified clusters in S as a guide for the identification of the clusters in X. In

the sequel, we describe such an algorithm, called the binary morphology clustering algorithm [Post 93]. The BMCA involves four main stages. During the first stage the data set is discretized and a new set is derived. This is the so-called *discrete binary (DB) set*. During the second stage, the basic morphological operators (opening and closing) are applied on the DB set, giving rise to a new discrete set. The third stage reveals the clusters formed in the last set. Finally, the last stage is responsible for the identification of the clusters formed by the original vectors of X, using as guide the clusters discovered during the third stage. Before we present the algorithm, let us first recall some basic tools and definitions.

15.4.1 Discretization

During the first step of the discretization stage, we normalize the vectors $\boldsymbol{x} \in X$ so that all their coordinates lie in the range $[0, r-1]$, where r is a user-defined parameter. This is achieved via the following transformation:

$$y_{ij} = \frac{x_{ij} - \min_{q=1,\ldots,N} x_{qj}}{\max_{q=1,\ldots,N} x_{qj} - \min_{q=1,\ldots,N} x_{qj}}(r-1), \quad i = 1,\ldots,N, j = 1,\ldots,l \tag{15.28}$$

where x_{ij} denotes coordinate j of vector i. Let us denote the resulting set by X', that is,

$$X' = \{\boldsymbol{y}_i \in [0, r-1]^l, i = 1,\ldots,N\}$$

Next, we discretize $[0, r-1]^l$ into r^l hypercubes. This is achieved by segmenting the $[0, r-1]$ interval, for each coordinate, into r subintervals (see Figure 15.13). Each hypercube is identified by the coordinates of its lower left corner. The parameter r defines the "resolution" of $[0, r-1]^l$.

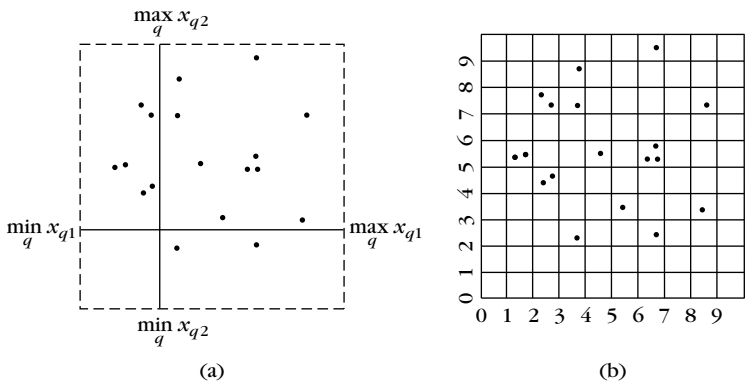

FIGURE 15.13

(a) The original data set X. (b) Normalization of the data set X in $[0, r-1]^l$, with $r = 10$, and discretization. The nonempty hypercubes define the discrete binary set.

In the sequel, we identify the hypercube where each vector $\boldsymbol{y}_i \in X'$ lies. This can be accomplished by simply taking the integer part of each coordinate of $\boldsymbol{y}_i$. The resulting vector, $\boldsymbol{z}_i$, will be the identity label of a hypercube in the defined grid. More specifically,

$$z_{ij} = [y_{ij}], \quad i = 1, \ldots, N, \quad j = 1, \ldots, l$$

where $[x]$ denotes the integer part of x. Let S be the set containing the new vectors $\boldsymbol{z}_i$, after removing all duplicates. Thus, each element of S corresponds to a nonempty hypercube, and S is the discrete binary set.

15.4.2 Morphological Operations

These operations are applied only to sets with discrete-valued vectors. The simplest operations of this kind are *dilation* and *erosion*. Based on these two operations, *opening* and *closing* operations are defined.

Let Y and T be subsets of Z^l, where Z is the set of integers and $\boldsymbol{s}$ a vector in Z^l. The *translation of Y by $\boldsymbol{s}$* is defined as

$$Y_{\boldsymbol{s}} = \{\boldsymbol{t} \in Z^l : \boldsymbol{t} = \boldsymbol{x} + \boldsymbol{s}, \ \boldsymbol{x} \in Y\} \tag{15.29}$$

Definition 1. *The dilation of Y by T, denoted by $Y \oplus T$, is defined as*

$$Y \oplus T = \{\boldsymbol{e} \in Z^l : \boldsymbol{e} = \boldsymbol{x} + \boldsymbol{s}, \boldsymbol{x} \in Y, \ \boldsymbol{s} \in T\} \tag{15.30}$$

Equivalently, the set $Y \oplus T$ is determined by translating Y by all elements of T and taking the union of the resulting sets [Gonz 93].

Definition 2. *The erosion of Y by T is denoted by $Y \ominus T$ and is defined as*

$$Y \ominus T = \{\boldsymbol{f} \in Z^l : \boldsymbol{x} = \boldsymbol{f} + \boldsymbol{s}, \boldsymbol{x} \in Y, \ \forall \ \boldsymbol{s} \in T\} \tag{15.31}$$

Equivalently, the set $Y \ominus T$ is determined by translating Y by all elements of T and taking the intersection of the resulting sets [Gonz 93].

In both of these cases, T is called the *structuring element*. Usually, it has a hypercubical shape (see Figure 15.14) but other choices, such as hyperspherical shape, are also possible.

Example 15.5

Let us consider a two-dimensional normal density function with mean $\boldsymbol{\mu} = [0, 0]^T$ and covariance matrix $\Sigma = 3I$, where I is the 2×2 identity matrix. Let X be a set containing 200 vectors stemming from this distribution (Figure 15.15a). We apply the discretization process on X, with $r = 20$, in order to obtain the corresponding discrete binary set, which is denoted by Y (Figure 15.15b). Let T be the 3×3 structuring element shown in Figure 15.14a, which consists of the points

$$\begin{aligned} T &= \{\boldsymbol{t}_1, \boldsymbol{t}_2, \ldots, \boldsymbol{t}_9\} \\ &= \{(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1)\} \end{aligned}$$

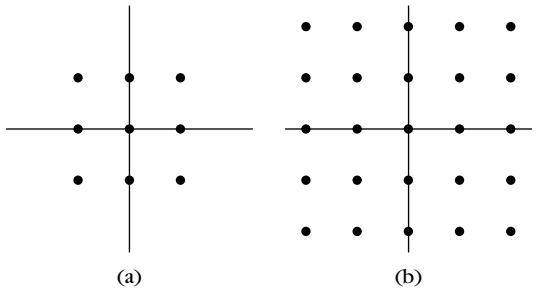

FIGURE 15.14

(a) A squared 3×3 structuring element. (b) A squared 5×5 structuring element.

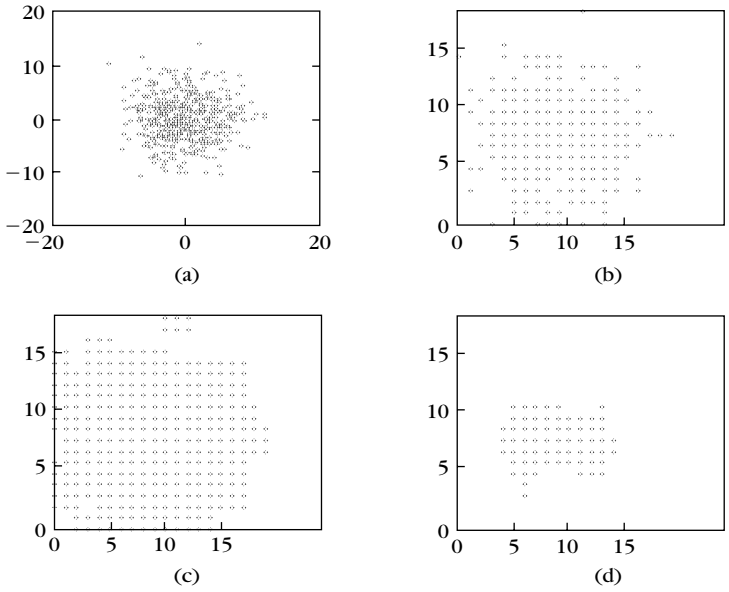

FIGURE 15.15

(a) The original data set X given in Example 15.5. (b) The discrete binary set Y derived from X. (c) The set Y dilated by T. (d) The set Y eroded by T.

In order to derive the dilation of Y by T, we compute the sets Y_i, $i = 1, \ldots, 9$, produced by the translation of Y by each element t_i of T, $i = 1, \ldots, 9$, and in the sequel, we take the union of all Y_i's. This is the dilation of Y by T (Figure 15.15c). The erosion of Y by T is computed by taking the intersection of all Y_i's defined above. The result of this operation is shown in Figure 15.15d.

Opening and *closing* are two additional basic operations that are defined in terms of the dilation and the erosion.

Definition 3. *The opening of Y by T is denoted by Y_T and is defined as*

$$Y_T = (Y \ominus T) \oplus T \tag{15.32}$$

that is, the opening of Y by T is the erosion of Y by T followed by the dilation of the resulting $Y \ominus T$ by T.

Definition 4. *The closing of Y by T is denoted by Y^T and is defined as*

$$Y^T = (Y \oplus T) \ominus T \tag{15.33}$$

that is, the closing of Y by T is the dilation of Y by T followed by the erosion of the resulting $Y \oplus T$ by T.

In general, Y is different from Y_T and Y^T. Note that the opening operation smooths out the boundary of Y by discarding irrelevant details of it. On the other hand, the closing operation fills the gaps in the set Y. These observations show that *opening and closing tend to produce new sets with simpler shapes than the original ones*. As pointed out in [Post 93], *"Opening and closing seem to be very effective to eliminate isolated groups of set points and holes, provided that these details do not exceed the size of the structuring element."* The following example shows how the opening and closing operations work.

Example 15.6

Let us consider the discrete binary set Y (see Figure 15.15b) and the structuring element T of Example 15.5. We derive first the opening of Y by T. The result is shown in Figure 15.16a. As one can observe, the resulting set retains the basic shape of Y, while irrelevant details

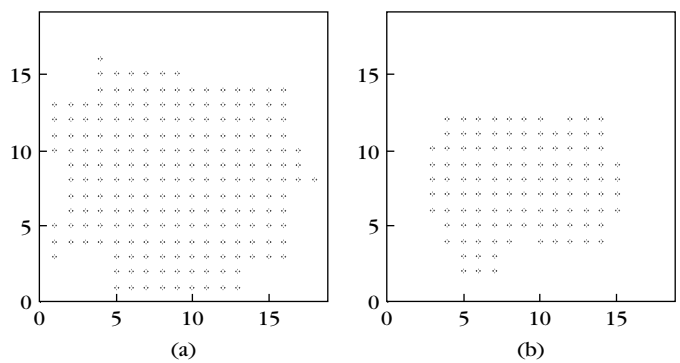

FIGURE 15.16

(a) The set Y given in Example 15.5, opened by T. (b) The set Y given in Example 15.5, closed by T.

of the boundary of Y have been discarded. The action of closing Y by T is shown in Figure 15.16b.

The above arguments indicate that the structuring element T plays an important role in the outcome of the above operations. Unfortunately, there are no general guidelines for choosing the appropriate T.

15.4.3 Determination of the Clusters in a Discrete Binary Set

We begin with a description of a rather simple algorithm suitable for clusters formed by the points of a discrete-valued data set $S \subset \{0, 1, \ldots, r-1\}^l$. Let us first define the neighborhood, $V(\boldsymbol{x})$, of a point $\boldsymbol{x} \in S$ as

$$V(\boldsymbol{x}) = \{\boldsymbol{y} \in S - \{\boldsymbol{x}\}: d(\boldsymbol{x}, \boldsymbol{y}) \leq d_q\}$$

where d may be any distance measure between two points (see Chapter 11) and d_q is a distance threshold. Also, let θ be a threshold of the density of the neighborhood $V(\boldsymbol{x})$ of a point $\boldsymbol{x}$. That is, if $V(\boldsymbol{x})$ contains at least θ points of S, it is considered "dense." These are user-defined parameters.

Let $U(\boldsymbol{x})$ be the immediate neighborhood of $\boldsymbol{x}$, that is, the set that contains all points lying at a (Euclidean) distance less than or equal to $\sqrt{l}$ from $\boldsymbol{x}$.

Cluster Detection Algorithm for Discrete-Valued Sets (CDADV)

- Initially no vector is considered as processed.
- Repeat
  - Choose a nonprocessed point $\boldsymbol{x}$ of S.
  - Determine the neighborhood $V(\boldsymbol{x})$.
  - If $V(\boldsymbol{x})$ contains at least θ points then
    - Create a new cluster that includes:
      - The point $\boldsymbol{x}$
      - All points $\boldsymbol{y} \in S$ for which there exists a sequence of points $\boldsymbol{y}_{j_s} \in S$, $s = 1, \ldots, q_y$, such that $\boldsymbol{y} \in U(\boldsymbol{y}_{j_1}), \boldsymbol{y}_{j_s} \in U(\boldsymbol{y}_{j_{s+1}}), s = 1, \ldots, q_y - 1$, and $\boldsymbol{y}_{j_{q_y}} \in U(\boldsymbol{x})$.
    - The defined points are considered as processed.
  - Else
    - Consider $\boldsymbol{x}$ as processed
  - End {if}
- Until all points of S have been processed.

Example 15.7

Consider the setup of Figure 15.17a. We choose $d_q = \sqrt{2}$ and $\theta = 1$ (in this case $V(\boldsymbol{x}) \equiv U(\boldsymbol{x})$). Also, the sides of the squares depicted in Figure 15.17a are of unit length. The CDADV

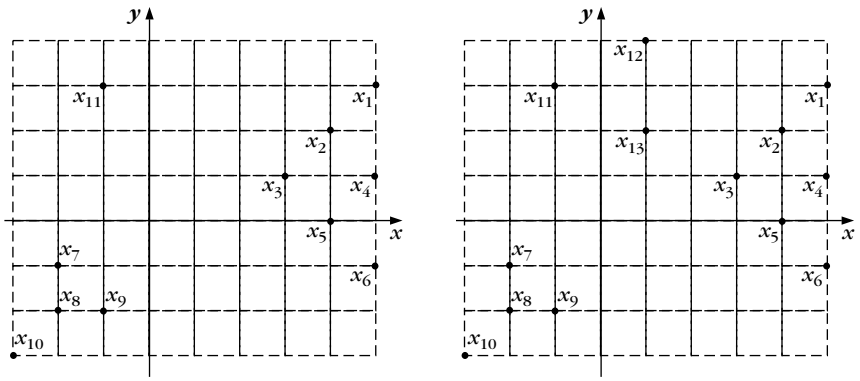

FIGURE 15.17

(a) The setup of Example 15.7. (b) A data set containing outliers.

considers first $\boldsymbol{x}_1$. Since it is "dense" (i.e., its neighborhood contains at least one point of S apart from $\boldsymbol{x}_1$), a new cluster is created. $\boldsymbol{x}_2$ also belongs to this cluster because $\boldsymbol{x}_2 \in U(\boldsymbol{x}_1)$. Moreover, $\boldsymbol{x}_3$, $\boldsymbol{x}_4$ belong to this cluster because $\boldsymbol{x}_3, \boldsymbol{x}_4 \in U(\boldsymbol{x}_2)$ and $\boldsymbol{x}_2 \in U(\boldsymbol{x}_1)$. In addition, $\boldsymbol{x}_5$ belongs to this cluster because $\boldsymbol{x}_5 \in U(\boldsymbol{x}_3)$, $\boldsymbol{x}_3 \in U(\boldsymbol{x}_2)$, and $\boldsymbol{x}_2 \in U(\boldsymbol{x}_1)$. Finally, since $\boldsymbol{x}_6 \in U(\boldsymbol{x}_5)$, $\boldsymbol{x}_5 \in U(\boldsymbol{x}_4)$, $\boldsymbol{x}_4 \in U(\boldsymbol{x}_2)$, and $\boldsymbol{x}_2 \in U(\boldsymbol{x}_1)$, $\boldsymbol{x}_6$ also belongs to this cluster. Working similarly, we find that $\boldsymbol{x}_7$, $\boldsymbol{x}_8$, $\boldsymbol{x}_9$, and $\boldsymbol{x}_{10}$ form a second cluster, while no action is taken for $\boldsymbol{x}_{11}$.

In the preceding scheme, all points of S are processed by the algorithm, regardless of their density. In fact, CDADV works well when the points of S form well-separated clusters. However, if this is not the case, for example, when we have a small number of outliers in S lying between its clusters (Figure 15.17b), we may proceed as follows. As it is expected, the neighborhood of an outlier is rather "sparse," and we first define a lower threshold θ_1 ($\leq\theta$) for the density of the neighborhood of a point and we consider the set S' of all points of S whose neighborhoods have density at least θ_1. That is, we exclude the outliers from S'. Then we run the CDADV on S' using θ, and, after its completion, we assign each vector of $S - S'$ to the cluster where its nearest point in S' belongs. Note that the distance function between a point and a cluster that we employ for this stage should not involve cluster representatives, since S is a discrete-valued set.

15.4.4 Assignment of Feature Vectors to Clusters

This subsection deals with the final stage of the algorithmic procedure. Let us recall that S is the discrete-valued set obtained from X, after applying the opening and closing transformations. Let $C'_1, \ldots, C'_m$ be the clusters formed in S, determined by the previously discussed CDADV algorithm. The aim of the current task is to

determine m clusters in X, denoted by $C_1, \ldots, C_m$, that correspond to the clusters $C_i', i = 1, \ldots, m$.

The algorithm assigns the vectors of X to clusters in two steps. During the first one, all vectors $\boldsymbol{x} \in X$ such that $[\boldsymbol{y}] \in S^5$ are considered. In the sequel, the algorithm assigns a vector $\boldsymbol{x}$ to the cluster C_i if $[\boldsymbol{y}]$ belongs to C_i'. Let X' be the set of the vectors of X that have been assigned to clusters during this step. At the second stage, the algorithm assigns each of the points in $X - X'$ to its closest $C_j, j = 1, \ldots, m$.

15.4.5 The Algorithmic Scheme

Having described the steps, we may proceed now to the description of the BMCA.

Binary Morphology–Based Clustering Algorithm (BMCA)

- *1st stage.* Discretize the data set X and let S be the resulting discrete binary set.
- *2nd stage.*
  - (a) Apply the opening transformation on S using a preselected structuring element T, to obtain S_T.
  - (b) Apply the closing transformation on S_T using T. Let $S_1 = (S_T)^T$ be the set obtained.
- *3rd stage.* Determine the underlying clusters of S_1 using the CDADV algorithm.
- *4th stage.* Based on the clusters formed in S_1, determine the underlying clusters of X.

It should be noted here that different choices of morphological operators can be used at the second stage of the algorithm. Thus, for example, one may use either the opening or the closing operator or both of them in the reverse order.

BMCA is sensitive to the parameter r and the structuring element T. These parameters may cause overestimation or underestimation of the true number of clusters underlying X. However, it is expected that when X contains clusters, their number remains unchanged for a significant range of values of the parameters involved (a similar situation has been met earlier in Chapter 12). Based on this assumption, we run the first three stages of the algorithm for various values of r and different T (for simplicity we may assume that T has a hypercubic scheme and, thus, its only parameter that is subject to change is the length of its side a). Then we plot the number of the resulting clusters versus r and a and we consider the widest area in the (r, a) plane, for which the number of clusters remains unchanged. The final values for r and a are chosen to be those corresponding to the middle point of the

[5] By $[\boldsymbol{y}]$ we denote the l-dimensional vector whose ith coordinate is the integer part of the ith coordinate of the l-dimensional vector $\boldsymbol{y}$.

above area. Using these values, we run the BMCA algorithmic scheme in order to determine the clusters of X.

A major drawback of the procedure is that it requires intensive computations, since many combinations of the values for r and a have to be considered. A way to reduce the required computations is to fix one of the two parameters to a reasonable value (if this is possible) and to apply the procedure only to the other parameter.

An important observation is that when the underlying clusters of X are compact and well separated, algorithms such as the isodata give better results than BMCA (see Problem 15.6). However, the situation is reversed when this is not the case. Let us consider the following example.

Example 15.8

Let X be a data set consisting of 1000 vectors. The first 500 of them, (x_{i_1}, x_{i_2}), are defined as

$$x_{i_1} = (i-1)\frac{2s}{500}$$

$$x_{i_2} = \sqrt{s^2 - x_{i_1}^2} + z_i$$

where $s = 10$ and z_i stems from a Gaussian distribution with zero mean and unit variance, $i = 1, \ldots, 500$. Similarly, the remaining vectors (x_{i_1}, x_{i_2}) are defined as

$$x_{i_1} = b_1 + (i-501)\frac{2s}{500}$$

$$x_{i_2} = b_2 + \sqrt{s^2 - (x_{i_1} - b_1)^2} + z_i'$$

where $b_1 = -10$, $b_2 = 3$, $s = 10$, and z_i' is normally distributed with zero mean and unit variance, $i = 501, \ldots, 1000$. It is not difficult to realize that the first 500 feature vectors spread around the upper half-circle with radius 10 centered at $(0, 0)$. Similarly, the rest of the 500 vectors spread around the lower half-circle with radius 10, centered at the point $(-10, 3)$ (see Figure 15.18a).

Clearly, two clusters are formed in X and each of them cannot be represented satisfactorily by a single point representative. In our simulations we use the 3×3 structuring element defined in Example 15.5 and $r = 25$. Also, the Euclidean distance between two vectors is adopted. The discrete binary set S, resulting from the discretization process, is shown in Figure 15.18b. Figure 15.18c shows the result of the opening of S by T, and Figure 15.18d shows the result of the closing of S_T by T. The two clusters involved in the last set are well separated. Application of the third stage of the BMCA algorithmic scheme reveals these two clusters. Finally, application of the fourth stage of the algorithm determines the clusters formed in X. The results obtained are excellent. Only two of the first 500 vectors were misclassified, and only 1 of the remaining 500 vectors was misclassified. Thus, 99.7% of the vectors of X were correctly classified. In contrast, the results obtained with the isodata algorithm were much inferior to these results.

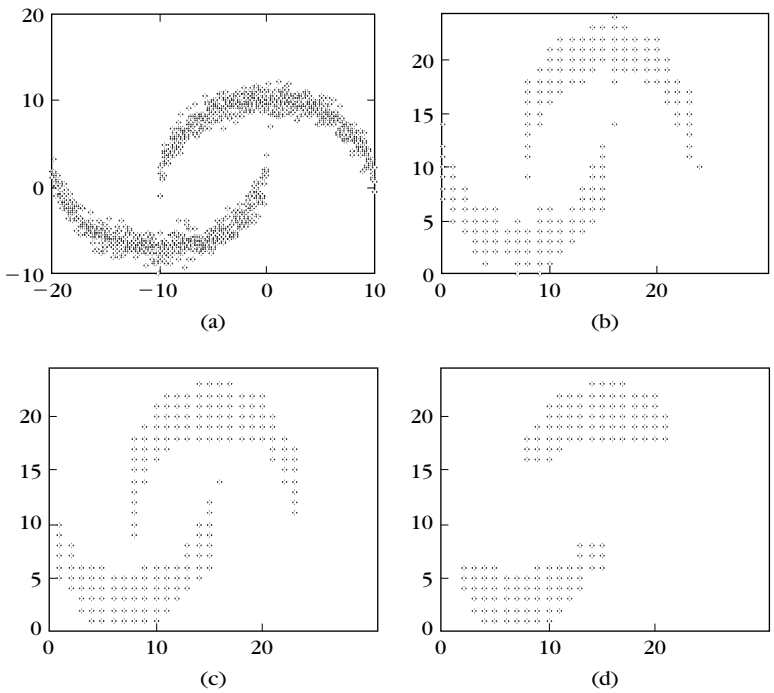

FIGURE 15.18

(a) The original data set X. (b) The set S resulting from the discretization of X. (c) The set resulting from opening of S by T. (d) The set resulting from closing of S_T by T.

15.5 BOUNDARY DETECTION ALGORITHMS

Most of the algorithms discussed so far determine clusters based on either the distance between vectors and clusters or the distance between clusters. In this section, a different rationale is discussed. Clusters are formed via the estimation of the boundary surfaces that separate them [Atiy 90]. This approach is well suited when the underlying clusters are compact. The idea is rather simple. The compact clusters are viewed as dense regions, in the l-dimensional space, separated by regions sparse in data vectors. Therefore, it suffices to begin with an initial estimate of the boundary and move it iteratively to regions that are sparse in vectors.

Let us consider first the case in which two clusters are present. Let $g(\boldsymbol{x}; \boldsymbol{\theta})$ be the function describing the decision boundary between the two clusters, where $\boldsymbol{\theta}$ is the unknown parameter vector describing the surface. If, for a specific $\boldsymbol{x}$, $g(\boldsymbol{x}; \boldsymbol{\theta}) > 0$, then $\boldsymbol{x}$ belongs to the first cluster, denoted by C^+. Otherwise, $\boldsymbol{x}$ belongs to the second cluster, denoted by C^-. The goal is to determine the unknown parameter vector $\boldsymbol{\theta}$. The situation looks similar to the supervised case where we identify the

decision boundary between classes, utilizing the labeling information of the feature vectors. However, no such information is available here. In the present case, *the adjustment of $\boldsymbol{\theta}$ relies exclusively on the distances of the vectors of X from the decision boundary.*

To this end, we define a cost function J, whose maximization will lead to locally optimal values for $\boldsymbol{\theta}$. Let J be defined as

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} f^2\left(g(\boldsymbol{x}_i;\boldsymbol{\theta})\right) - \left(\frac{1}{N}\sum_{i=1}^{N} f(g(\boldsymbol{x}_i;\boldsymbol{\theta}))\right)^{2q} \tag{15.34}$$

where q is a constant positive integer and $f(x)$ is a monotonically increasing symmetric squashing function with

$$\lim_{x\to+\infty} f(x) = 1, \qquad \lim_{x\to-\infty} f(x) = -1, \quad \text{and} \quad f(0) = 0 \tag{15.35}$$

A common choice for such a function is the hyperbolic tangent

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

Each of the two terms in (15.34) has a maximum value of 1. Also, $J(\boldsymbol{\theta})$ is nonnegative since

$$J(\boldsymbol{\theta}) \geq \frac{1}{N}\sum_{i=1}^{N}\left(f(g(\boldsymbol{x}_i;\boldsymbol{\theta})) - \frac{1}{N}\sum_{k=1}^{N} f(g(\boldsymbol{x}_k;\boldsymbol{\theta}))\right)^2 \geq 0 \tag{15.36}$$

One can easily observe that the first term in Eq. (15.34) is maximized when all $\boldsymbol{x} \in X$ lie away from the boundary. In this case $f^2(g(\boldsymbol{x}_i;\boldsymbol{\theta})) \to 1$ and the first term attains values close to 1. However, this argument holds also true if *all* vectors of X lie on the same side of the boundary and away from it. The role of the second term is to discourage such trivial solutions. Indeed, in such cases $(\frac{1}{N}\sum_{i=1}^{N} f(g(\boldsymbol{x}_i;\boldsymbol{\theta})))^{2q} \to 1$ and therefore $J(\boldsymbol{\theta})$ approaches zero, its minimum value. The role of q is to control the impact of the second term on the cost function J.

Let us now consider an intermediate case in which the boundary lies between two dense regions. In such cases, the contribution of the second term to J is small. Let us demonstrate it via a simplified example. Assume that the decision surface is a hyperplane H and that at the positive (negative) side of H we have k points lying at distance a $(-a)$ away from it. Then it is not difficult to show that the second term becomes zero while the first equals $f^2(a\|\boldsymbol{\theta}\|)$.

In the sequel we adopt a steepest ascent scheme in order to determine the optimal value for $\boldsymbol{\theta}$. Let θ_j be a coordinate of $\boldsymbol{\theta}$. Then

$$\theta_j(t+1) = \theta_j(t) + \mu\frac{\partial J(\boldsymbol{\theta})}{\partial\theta_j}\bigg|_{\theta_j=\theta_j(t)}$$

or

$$\theta_j(t+1) = \theta_j(t) + \mu \left(\frac{2}{N} \sum_{i=1}^{N} f(g(\boldsymbol{x}_i; \boldsymbol{\theta})) \frac{\partial f(g(\boldsymbol{x}_i; \boldsymbol{\theta}))}{\partial g(\boldsymbol{x}_i; \boldsymbol{\theta})} \frac{\partial g(\boldsymbol{x}_i; \boldsymbol{\theta})}{\partial \theta_j} - \frac{2q}{N} \left(\frac{1}{N} \sum_{i=1}^{N} f(g(\boldsymbol{x}_i; \boldsymbol{\theta})) \right)^{2q-1} \sum_{i=1}^{N} \frac{\partial f(g(\boldsymbol{x}_i; \boldsymbol{\theta}))}{\partial g(\boldsymbol{x}_i; \boldsymbol{\theta})} \frac{\partial g(\boldsymbol{x}_i; \boldsymbol{\theta})}{\partial \theta_j} \right) \Big|_{\theta_j = \theta_j(t)} \tag{15.37}$$

For the simple case where $g(\boldsymbol{x}; \boldsymbol{\theta})$ is a hyperplane we can write

$$g(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

where $\boldsymbol{\theta} \equiv [\boldsymbol{w}\ w_0]^T$. The updating equation for the parameters follows directly from Eq. (15.37) if we notice that

$$\frac{\partial g(\boldsymbol{x}; \boldsymbol{\theta})}{\partial w_j} = \begin{cases} x_j, & j = 1, \ldots, l \\ 1, & j = 0 \end{cases}$$

The resulting algorithm is rather simple in its formulation and may be stated as follows

Boundary Detection Algorithm (BDA)

- Choose an initial value $\boldsymbol{\theta}(0)$ for the parameter vector.
- Compute $J(\boldsymbol{\theta}(0))$ using Eq. (15.34).
- $t = 0$.
- Repeat
  - $t = t + 1$
  - Compute $\boldsymbol{\theta}(t)$ using Eq. (15.37).
  - Compute $J(\boldsymbol{\theta}(t))$ using Eq. (15.34).
- Until $\left| \dfrac{J(\boldsymbol{\theta}(t+1)) - J(\boldsymbol{\theta}(t))}{J(\boldsymbol{\theta}(t))} \right| < \varepsilon$.

We note here that the coordinates of $\boldsymbol{\theta}$ should not grow in an unbounded way. In the case that $g(\boldsymbol{x}; \boldsymbol{\theta})$ corresponds to a hyperplane, a bounded condition for the coordinates of $\boldsymbol{\theta}$ could be $\|\boldsymbol{\theta}\| \leq a$, where a is a user-defined parameter.

Let us now consider the case in which more than two clusters underlie X. In this case, we follow a hierarchical procedure. First we divide X into two clusters X^+ and X^- using the boundary detection algorithm. Then, using the algorithm again, we further divide X^+ (X^-) and obtain X^{+-} and X^{++} (X^{--} and X^{-+}). This procedure is then applied iteratively to the resulting clusters until a specific termination criterion is met. This procedure reminds us of the neural network design discussed in Chapter 4 [Atiy 90].

One can easily observe that if no sparse regions exist in a formed cluster C, the division of C will result in a low value of $J(\boldsymbol{\theta})$. Thus, an appropriate criterion that may be used to check whether C contains more clusters is

$$J(\boldsymbol{\theta}) \leq b$$

where b is a user-defined threshold. If this happens, the division of C is not acceptable. Otherwise, the division of C is accepted and we proceed with C^+ and C^-. It is clear that the smaller the value of b, the more clusters will be defined. On the other hand, higher values of b result in the acceptance of fewer clusters with well-defined borders among them. Thus, b should be chosen with care.

A different algorithm, called *OptiGrid*, that separates clusters by applying a suitably defined grid of hyperplanes on the feature space, is discussed in [Hinn 99].

15.6 VALLEY-SEEKING CLUSTERING ALGORITHMS

The method discussed here is in the same spirit as that of the previous section. Let $p(\boldsymbol{x})$ be the density function describing the distribution of the vectors in X. An alternative way to attack the clustering problem is to view the clusters as peaks of $p(\boldsymbol{x})$ separated by valleys. Inspired by this consideration, one can search to identify such valleys, and try to move and place the borders of the clusters in these valleys.

In the sequel, we discuss an iterative and computationally effective algorithm based on this idea [Fuku 90]. Once more, let $V(\boldsymbol{x})$ be the local region of $\boldsymbol{x}$, that is,

$$V(\boldsymbol{x}) = \{\boldsymbol{y} \in X - \{\boldsymbol{x}\}: d(\boldsymbol{x}, \boldsymbol{y}) \leq a\} \tag{15.38}$$

where a is a user-defined parameter. The distance $d(\boldsymbol{x}, \boldsymbol{y})$ can be taken to be

$$d(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{y} - \boldsymbol{x})^T A(\boldsymbol{y} - \boldsymbol{x}) \tag{15.39}$$

where A is a symmetric positive definite matrix. Also, let k_j^i denote the number of vectors of the j cluster that belong to $V(\boldsymbol{x}_i)$, excluding $\boldsymbol{x}_i$. Also, let $c_i \in \{1, \ldots, m\}$ denote the cluster to which $\boldsymbol{x}_i$ belongs, $i = 1, \ldots, N$. Then the algorithm is stated as follows.

Valley-Seeking Algorithm

- Fix a.
- Fix the number of clusters m.
- Define an initial clustering of X.
- Repeat
  - For $i = 1$ to N

- Find j: $k_j^i = \max_{q=1,\ldots,m} k_q^i$ [6]
- Set $c_i = j$

- End {For}
- For $i = 1$ to N
  - Assign $\boldsymbol{x}_i$ to cluster C_{c_i}.
- End {For}

- Until no reclustering of vectors occurs.

Remarks

- Observe that the preceding algorithm has close similarities to the Parzen windows method, for pdf estimation, discussed in Chapter 2. Indeed, all it does is to move a window $d(\boldsymbol{x}, \boldsymbol{y}) \leq a$ at $\boldsymbol{x}$ and to count points from different clusters. Then it assigns $\boldsymbol{x}$ to the cluster with the larger number of points in the window. That is, to the cluster with the highest local pdf density. This is equivalent with moving the boundary away from the "winning" cluster.
- The preceding is a mode seeking algorithm. That is, if more than enough clusters are initially appointed, some of them may be empty.

Example 15.9

(a) Consider Figure 15.19a. X consists of the following 10 vectors: $\boldsymbol{x}_1 = [0, 1]^T$, $\boldsymbol{x}_2 = [1, 0]^T$, $\boldsymbol{x}_3 = [1, 2]^T$, $\boldsymbol{x}_4 = [2, 1]^T$, $\boldsymbol{x}_5 = [1, 1]^T$, $\boldsymbol{x}_6 = [5, 1]^T$, $\boldsymbol{x}_7 = [6, 0]^T$, $\boldsymbol{x}_8 = [6, 2]^T$, $\boldsymbol{x}_9 = [7, 1]^T$, $\boldsymbol{x}_{10} = [6, 1]^T$. The squared Euclidean distance is employed.

The initial clustering consists of two clusters as shown in Figure 15.19a. Also, a decision line, $\boldsymbol{b}_1$, separating the two clusters is shown (Figure 15.19a). Let $a = 1.415$.[7] After the first iteration of the algorithm, $\boldsymbol{x}_4$ is assigned to the cluster denoted by "$\boldsymbol{x}$." This is equivalent to moving the decision curve separating the two clusters to the valley between the two high density areas.

(b) Now consider Figure 15.19b. The set X remains unchanged. Also, the initial clustering remains the same except that $\boldsymbol{x}_6$ is assigned to the cluster denoted by "$\boldsymbol{x}$." Three curves, $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, and $\boldsymbol{b}_3$, can now be used to separate the clusters. After the first iteration of the algorithm, $\boldsymbol{x}_4$ is assigned to the cluster denoted "$\boldsymbol{x}$" and $\boldsymbol{x}_6$ is assigned to the cluster denoted by "$\boldsymbol{o}$." Equivalently, $\boldsymbol{b}_1$, $\boldsymbol{b}_2$ can be thought to move to the valley between the two peaks in the place of $\boldsymbol{b}_3$.

---

[6] If ties occur, that is, more than one maximum are encountered, we choose the one with the smallest index.

[7] This number is slightly greater than $\sqrt{2}$.

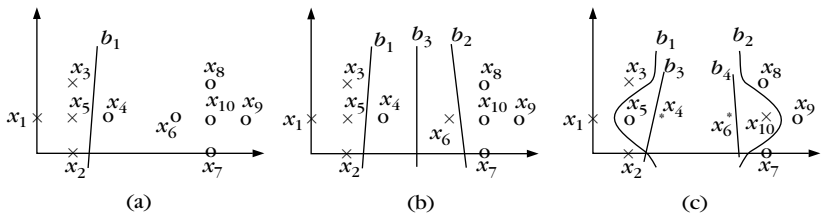

FIGURE 15.19

(a) The setup of Example 15.9(a). (b) The setup of Example 15.9(b). (c) The setup of Example 15.9(c).

(c) Finally, let us consider Figure 15.19c. Again X remains unchanged. However, the initial clustering consists of three clusters, denoted by "x," "o," and "$*$." Also, the initial clustering is the same as that of (b) except that $\boldsymbol{x}_4$ and $\boldsymbol{x}_6$ are assigned to the cluster denoted by "$*$," $\boldsymbol{x}_5$ is assigned to the cluster denoted by "o," and, finally, $\boldsymbol{x}_{10}$ is assigned to the cluster denoted by "x." The decision surfaces consist of the curves $\boldsymbol{b}_i$, $i = 1, \ldots, 4$. After the first iteration of the algorithm, $\boldsymbol{x}_4$ and $\boldsymbol{x}_5$ are assigned to the cluster denoted by "x." Likewise, $\boldsymbol{x}_6$ and $\boldsymbol{x}_{10}$ are assigned to the cluster denoted by "o." Finally, the rest vectors remain in the clusters where they were initially assigned. The important point here is that, although we initially considered three clusters, the algorithm ends up with two. This is because only two peaks are present. Moreover, in all cases, the decision surface is moved to the valley between the two peaks.

It should be emphasized here that the algorithm is sensitive to the value of the parameter a. Thus, one should run the algorithm several times for different values of a and interpret the results very carefully.

An alternative algorithm based on similar ideas is discussed in [Fuku 90]. It identifies the underlying clusters of X by moving the $\boldsymbol{x}_i \in X$ toward the direction of $\partial p(\boldsymbol{x})/\partial \boldsymbol{x}$, computed at $\boldsymbol{x}_i$, by $\eta \partial p(\boldsymbol{x})/\partial \boldsymbol{x}$, where η is a user-defined parameter. Iterating this procedure, points of the same cluster converge towards the same point in space (a method for the estimation of the gradient of $p(\boldsymbol{x})$ is given in [Fuku 90]). Finally, other related algorithms can be found in [Touz 88, Chow 97].

15.7 CLUSTERING VIA COST OPTIMIZATION (REVISITED)

In this section we present four optimization methods that have been used successfully in many fields of application.

15.7.1 Branch and Bound Clustering Algorithms

As already stated in Chapter 5, *branch and bound* methods compute the globally optimal solution to combinatorial problems, according to a prespecified criterion

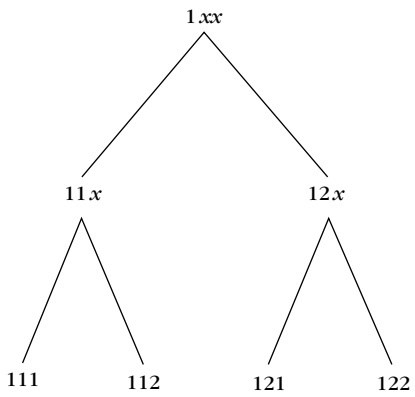

FIGURE 15.20

The classification tree corresponding to the grouping of three vectors in two clusters.

(cost) function J, overcoming the need for exhaustive search.[8] They are applicable to monotonic criteria. That is, if k vectors of X have been assigned to clusters, the assignment of an extra vector to a cluster does not decrease the value of J.

We will first attempt to gain some insight into these methods by considering an example. Let us assume that our goal is to find the best way (with respect to a criterion J) in which three vectors can be clustered in two clusters. To this end, we construct the *classification tree* of Figure 15.20. Each node is characterized by a string of three symbols, namely 1, 2, and x. For example, the string "122" means that the first vector is assigned to cluster 1 while the other two are assigned to cluster 2. Also, the string "1xx" means that the first vector is assigned to cluster 1 while the other two remain unassigned. The first vector is always assigned to the first cluster. Note that each leaf corresponds to an actual clustering and there are as many leaves as the possible clusterings of the three vectors in two clusters. All the other nodes correspond to the so-called *partial clusterings*, that is, to clusterings where not all the vectors of X have been assigned yet to a cluster.

We are now ready to see where the computational saving comes from. Let us assume that at an iteration step of the algorithm, the best computed value for the criterion J is B. Then, if at a node the corresponding value of J is greater than B, *no further search is performed for all subsequent descendants springing from this node.* This is because of the monotonicity of the criterion, which ensures that all descendants will result in values of J no less than B.

More formally, let $\mathcal{C}_r = [c_1, \ldots, c_r]$, $1 \leq r \leq N$, denote a partial clustering, where $c_i \in \{1, 2, \ldots, m\}$, $c_i = j$ if the vector $\boldsymbol{x}_i$ belongs to cluster C_j, and the vectors $\boldsymbol{x}_{r+1}, \ldots, \boldsymbol{x}_N$ are not yet assigned to any cluster.

[8] In the sequel we consider only the minimization problem.

In the sequel, we focus on compact clusters and we give a simple branch and bound algorithm [Koon 75]. We assume that the number of clusters, m, is fixed. The criterion function employed is defined as

$$J(\mathcal{C}_r) = \sum_{i=1}^{r} \|\boldsymbol{x}_i - \boldsymbol{m}_{c_i}(\mathcal{C}_r)\|^2 \tag{15.40}$$

where $\boldsymbol{m}_{c_i}$ is the mean vector of the cluster C_{c_i}, that is,

$$\boldsymbol{m}_j(\mathcal{C}_r) = \frac{1}{n_j(\mathcal{C}_r)} \sum_{\{q=1,\ldots,r:c_q=j\}} \boldsymbol{x}_q, \qquad j = 1, \ldots, m \tag{15.41}$$

with $n_j(\mathcal{C}_r)$ being the number of vectors $\boldsymbol{x} \in \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_r\}$ that belong to cluster C_j. Note that the computation of the mean vectors of the clusters takes into account only the first r vectors. We assume that $J(\mathcal{C}_1) = 0$. One can easily verify that

$$J(\mathcal{C}_{r+1}) = J(\mathcal{C}_r) + \Delta J(\mathcal{C}_r) \tag{15.42}$$

with $\Delta J(\mathcal{C}_r) \geq 0$. In words, $\Delta J(\mathcal{C}_r)$ denotes the increase in the value of J when the next vector is assigned to a cluster. More precisely, assuming that the $r+1$ vector is assigned to the cluster C_j, it can be shown that

$$\Delta J(\mathcal{C}_r) = \frac{n_j(\mathcal{C}_r)}{n_j(\mathcal{C}_r) + 1} \|\boldsymbol{x}_{r+1} - \boldsymbol{m}_j(\mathcal{C}_r)\|^2 \tag{15.43}$$

Let $\mathcal{C}_N^* = [c_1^*, \ldots, c_N^*]$ denote the optimal clustering. In the sequel, the index r denotes the vector that is currently considered for cluster assignment. Then the algorithm may be stated as follows.

Branch and Bound Clustering (BBC) Algorithm

$r = 1$
$B = +\infty$
While $r \neq 0$ do

- If $(J(\mathcal{C}_r) < B)$ AND $(r < N)$ then
  - $r = r + 1$
  - Among all possible assignments, c_r, of $\boldsymbol{x}_r$ that have not been tested yet, choose the one that minimizes the value of $\Delta J(\mathcal{C}_r)$.[9]
- End {If}
- If $(J(\mathcal{C}_r) < B)$ AND $(r = N)$ then
  - $\mathcal{C}_N^* = \mathcal{C}_N$
  - $B = J(\mathcal{C}_N)$
- End {If}

[9] If more than one c_r's minimize $\Delta J(\mathcal{C}_r)$, choose the smallest one.

- If (($J(\mathcal{C}_r) \leq B$) AND ($r = N$)) OR ($J(\mathcal{C}_r) > B$) then
  - (A) $r = r - 1$
  - If ($r = 0$) then
    - Stop
  - Else
    - If all possible clusterings that branch from this node have been exhausted for the rth vector then
      - Go to (A)
    - Else
      - $r = r + 1$
      - Among all possible c_r's that have not been tested yet, choose another path, the one that minimizes the value of $\Delta J(\mathcal{C}_r)$.
    - End {If}
  - End {If}
- End {If}

End {While}

The algorithm starts from the initial node of the tree and goes down until either (i) a leaf or (ii) a node q with cost function value greater than B is encountered. In case (i) if the cost for the clustering that corresponds to that leaf is less than B, then this cost becomes the new bound B, and the clustering is the best clustering found so far. In case (ii) all subsequent clusterings branching from q are not considered any further and we say that they are exhausted. The algorithm, then, backtracks to the parent node of q in order to span a different path. If all paths branching from the parent of q have already been considered, we move to the grandparent of q. The algorithm terminates when all possible paths have been considered explicitly or implicitly (via the aforementioned case (ii)). Clearly, in the beginning, the BBC algorithm spans first a whole path from the initial node of the tree down to a leaf. The cost function of the clustering corresponding to that leaf is the new value of B.

It is clear that the tighter the upper bound B, the more paths are rejected without explicit consideration. Variations of this algorithm that use better estimates of B are discussed in [Koon 75]. Moreover, the substitution of the values $J(\mathcal{C}_r)$ with tighter lower bounds of the optimal value of J is also suggested in [Koon 75]. This leads to the rejection of many more clusterings without considering them explicitly.

The major disadvantage of this algorithm is the excessive amount of computational time it requires even for moderate values of N.[10] In addition, this time is unpredictable. One way to face this problem is to run the algorithm for a prechosen time and use the best clustering found so far. It is clear that in this

[10] This is a common feature for the methods performing global optimization.

case, the algorithm can no longer guarantee the determination of the globally optimal clustering. Versions of the branch and bound algorithm that achieve some computational savings by reducing redundant J evaluations have appeared in [Yu 93, Chen 95].

15.7.2 Simulated Annealing

This is a global optimization algorithm. More specifically, under certain conditions, it guarantees, in probability, the computation of the globally optimal solution of the problem at hand via the minimization of a cost function J. This algorithm has been proposed by Kirkpatrick *et al.* [Kirk 83] (see also [Laar 87]) and it is inspired by the problem of condensed matter in physics.[11] In contrast to the algorithms that allow corrections of the unknown parameters only to directions that reduce the cost function J, simulated annealing allows moves that, temporarily, may increase the value of J. The rationale is that, by allowing such moves, we may escape from the region of attraction of a local minimum.

A very important parameter of this method is the so-called temperature T, which is the analog of the temperature in physical systems. The algorithm starts with a high temperature, which is reduced gradually. A *sweep* is the time that has to be spent at a given temperature so that the system can enter the "thermal equilibrium" state. Let T_{max} and $\mathcal{C}_{init}$ denote the initial value of the temperature, T, and the initial clustering, respectively. Also, $\mathcal{C}$ denotes the current clustering and t the current sweep. The general scheme of simulated annealing, in the clustering context, is the following.

Simulated Annealing for Clustering

- Set $T = T_{max}$ and $\mathcal{C} = \mathcal{C}_{init}$.
- $t = 0$.
- Repeat
  - $t = t + 1$
  - Repeat
    - Compute $J(\mathcal{C})$.
    - Produce a new clustering, $\mathcal{C}'$, by assigning a randomly chosen vector from X to a different cluster.
    - Compute $J(\mathcal{C}')$.
    - If $\Delta J = J(\mathcal{C}') - J(\mathcal{C}) < 0$ then
      - (A) $\mathcal{C} = \mathcal{C}'$

[11] Also, it shares many common features with the Metropolis algorithm [Metr 53].

- Else
  - (B) $\mathcal{C} = \mathcal{C}'$, with probability $P(\Delta J) = e^{-\Delta J/T}$.
- End if

- Until an equilibrium state is reached at this temperature.
- $T = f(T_{\max}, t)$

- Until a predetermined value T_{min} for T is reached.

It is clear that high values of the temperature imply that almost all movements of vectors between clusters are allowed, since, as $T \to +\infty$, $P(\Delta J) \simeq 1$. On the other hand, for low values of T, fewer moves of the (B) type are allowed and, finally, as $T \to 0$, the probability of such moves tends to zero. Thus, as T is lowered, it becomes more probable that clusterings that correspond to lower values of J will be reached. On the other hand, by keeping T positive, we ensure a nonzero probability for escaping from a local minimum.

A difficulty with this algorithm is the determination of the equilibrium state at a specific temperature. One heuristic rule for this case is to consider that the equilibrium state has been reached if for k successive random reassignments of patterns, $\mathcal{C}$ remains unchanged (typically k is of the order of a few thousand). Further discussion of this direction is provided in [Klei 89]. Also, another crucial point is the schedule for lowering T. It has been shown that if

$$T = T_{max}/\ln(1 + t) \tag{15.44}$$

this scheme converges to the global minimum with probability 1 [Gema 84]. However, this schedule is too slow. A faster schedule for lowering T is discussed in [Szu 86]. Despite this, the main disadvantage of this algorithm remains the vast amount of computations required.

Finally, in [Al-S 93] simulated annealing is used in terms of fuzzy clustering. Experiments with simulated annealing in clustering problems are presented in [Klei 89, Brow 92].

15.7.3 Deterministic Annealing

This is a hybrid parametric scheme combining the advantages of simulated annealing and the deterministic clustering methods. In contrast to simulated annealing, where successive clusterings are obtained by randomly disturbing the current one, no random disturbances occur here. On the other hand, the cost function is changed slightly, in order to accommodate the parameter $\beta = 1/T$, where T is defined as in the simulated annealing methods.[12] In contrast to simulated annealing, deterministic annealing is the counterpart of the phase transition phenomenon that is observed when the temperature of a material changes [Rose 91].

[12] We choose to work with β instead of T, because this notation is generally used for this algorithm.

In this framework, a set of representatives $\boldsymbol{w}_j, j = 1, \ldots, m$ (m is fixed), is adopted and our goal is to locate them in appropriate positions so that a distortion function is minimized. To this end, the following "effective" cost function J is constructed [Rose 91]:

$$J = -\frac{1}{\beta} \sum_{i=1}^{N} \ln \left(\sum_{j=1}^{m} e^{-\beta d(\boldsymbol{x}_i, \boldsymbol{w}_j)} \right) \tag{15.45}$$

where m is the number of clusters. Differentiating J with respect to the representative $\boldsymbol{w}_r$ and setting it equal to 0, we obtain

$$\frac{\partial J}{\partial \boldsymbol{w}_r} = \sum_{i=1}^{N} \left(\frac{e^{-\beta d(\boldsymbol{x}_i, \boldsymbol{w}_r)}}{\sum_{j=1}^{m} e^{-\beta d(\boldsymbol{x}_i, \boldsymbol{w}_j)}} \right) \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{w}_r)}{\partial \boldsymbol{w}_r} = \mathbf{0} \tag{15.46}$$

It is clear that the ratio in parentheses takes values in [0, 1] and, in addition, all these terms for a specific $\boldsymbol{x}_i$ sum up to 1. Thus, it may be interpreted as the probability, P_{ir}, that $\boldsymbol{x}_i$ belongs to $C_r, r = 1, \ldots, m$. Then Eq. (15.46) can be written as

$$\sum_{i=1}^{N} P_{ir} \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{w}_r)}{\partial \boldsymbol{w}_r} = \mathbf{0} \tag{15.47}$$

In the sequel, we assume that $d(\boldsymbol{x}, \boldsymbol{w})$ is a convex function of $\boldsymbol{w}$ for fixed $\boldsymbol{x}$. Note that for $\beta = 0$, all P_{ij}'s are equal to $1/m$, for all $\boldsymbol{x}_i$'s, $i = 1, \ldots, N$. Thus, in this case Eq. (15.47) becomes

$$\sum_{i=1}^{N} \frac{\partial d(\boldsymbol{x}_i, \boldsymbol{w}_r)}{\partial \boldsymbol{w}_r} = \mathbf{0} \tag{15.48}$$

Since $d(\boldsymbol{x}, \boldsymbol{w})$ is a convex function, $\sum_{i=1}^{N} d(\boldsymbol{x}_i, \boldsymbol{w}_r)$ is also a convex function and, thus, it has a unique minimum, which may be captured by any gradient descent scheme. Thus, in this case, *all the resulting representatives coincide with this unique global minimum.* That is, all data belong to a single cluster. As the value of β increases, it reaches a critical value where a phase transition occurs (alternatively, the probabilities P_{ir} "depart sufficiently" from the uniform model); that is, the clusters are no longer optimally represented by a single representative. Thus, the representatives split up in order to provide an optimal representation of the data set at the new phase. Further increase of β causes a new phase transition and the available representatives are further split up. By choosing m to be greater than the "actual" number of clusters, we ensure the ability of the algorithm to represent the data set properly. In the worst case, some of the representatives will coincide.

Note that as β increases, the probabilities P_{ij} depart from the uniform model and approach the hard clustering model; that is, for all $\boldsymbol{x}_i, P_{ir} \simeq 1$ for some r, and $P_{ij} \simeq 0$ for $j \neq r$.

Thus, the requirement for each vector to be assigned to a specific cluster with probability close to unity may serve as a termination criterion for the algorithm.

Schedules for the increase of β are discussed in [Rose 91]. Although simulation results show satisfactory performance of the algorithm, it is not guaranteed that it reaches the globally optimum clustering. Other applications of deterministic annealing to clustering are discussed in [Hofm 97, Beni 94].

15.7.4 Clustering Using Genetic Algorithms

Genetic algorithms have been inspired by the natural selection mechanism introduced by Darwin. They apply certain operators to a population of solutions of the problem at hand, in such a way that the new population is improved compared with the previous one according to a prespecified criterion function J. This procedure is applied for a preselected number of iterations and the output of the algorithm is the best solution found in the last population or, in some cases, the best solution found during the evolution of the algorithm.

In general, the solutions of the problem at hand are coded[13] and the operators are applied to the coded versions of the solutions. The way the solutions are coded plays an important role in the performance of a genetic algorithm. Inappropriate coding may lead to poor performance.

The operators used by genetic algorithms simulate the way natural selection is carried out. The most well-known operators used are the *reproduction*, *crossover*, and *mutation* operators applied in that order to the current population. The reproduction operator ensures that, in probability, the better (worse) a solution in the current population is, the more (less) replicates it has in the next population. The crossover operator, which is applied to the temporary population produced after the application of the reproduction operator, selects pairs of solutions randomly, splits them at a random position, and exchanges their second parts. Finally, the mutation operator, which is applied after the application of the reproduction and crossover operators, selects randomly an element of a solution and alters it with some probability. The last operator may be viewed as a way out of getting stuck in local minima. Apart from these three operators, many others have been proposed in the literature (e.g., [Mich 94]).

Besides the coding of the solutions, other parameters, such as the number of solutions in a population, p,[14] the probability with which we select two solutions for crossover, and the probability with which an element of a solution is mutated, play very important roles in the performance of the algorithm.

Several genetic algorithms with application to clustering have been proposed (e.g., [Bhan 91, Andr 94, Sche 97, Maul 00, Tsen 00]). In the sequel, we briefly discuss a simple parametric one that is suitable for hard clustering. We assume that the number of clusters, m, is fixed. As stated before, the first thing we have to decide

[13] Binary representations as well as more general ones are possible.
[14] This may be fixed or varied.

is how to code a solution. A simple (but not unique) way to achieve this is to use the representatives in order to form the following string:

$$[\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m] \tag{15.49}$$

or, in more detail,

$$[w_{11}, w_{12}, \ldots, w_{1l}, w_{21}, w_{22}, \ldots, w_{2l}, \ldots, w_{m1}, w_{m2}, \ldots, w_{ml}] \tag{15.50}$$

The cost function we use is

$$J = \sum_{i=1}^{N} u_{ij} d(\boldsymbol{x}_i, \boldsymbol{w}_j) \tag{15.51}$$

where

$$u_{ij} = \begin{cases} 1, & d(\boldsymbol{x}_i, \boldsymbol{w}_j) = \min_{k=1,\ldots,m} d(\boldsymbol{x}_i, \boldsymbol{w}_k) \\ 0, & \text{otherwise} \end{cases} \quad i = 1, \ldots, N \tag{15.52}$$

The allowable cut points for the crossover operator are between different representatives. Also, in this case the mutation operator selects randomly a coordinate of a vector of a solution and decides randomly to add a small random number to it.

An alternative to this algorithm is the following. Before we apply the reproduction operator to the current population, we run the hard clustering algorithm, described in Chapter 14, p times, each time using a different solution of the current population as the initial state. The p solutions produced after the convergence of the hard clustering algorithm constitute the population to which the reproduction operator will be applied. It is expected that this modification will give better results, as it is likely that the resulting p solutions are local minima of the cost function. This modified algorithm has been reported to give satisfactory results in a color image quantization application, [Sche 97].

15.8 KERNEL CLUSTERING METHODS

The minimal-enclosure (hyper)sphere (i.e., the sphere with the minimum volume enclosing all points in a vector space X) was briefly discussed in Chapter 5 (Problem 5.20). In [Tax 99], this problem was considered in a more relaxed framework, allowing some of the points of the data set to lie outside the volume of the sphere. This viewpoint makes the problem less sensitive to outliers. The optimization task now becomes similar to the soft-margin SVM and can be casted as

$$\text{minimize} \quad r^2 + C \sum_{i=1}^{N} \xi_i \tag{15.53}$$

$$\text{subject to} \quad ||\boldsymbol{x} - \boldsymbol{c}||^2 \leq r^2 + \xi_i, \; i = 1, 2, \ldots, N \tag{15.54}$$

$$\xi_i \geq 0, \quad i = 1, 2, \ldots, N \tag{15.55}$$

In other words, the radius r of the sphere, centered at $\boldsymbol{c}$, enclosing the data points of X, is minimized. However, some points in X are allowed to lie outside it ($\xi > 0$), but at the same time the number of these points must be as small as possible (due to the second term in Equation (15.53)). The Lagrangian of the above constrained problem (Chapter 3 and Appendix C) is given by

$$\mathcal{L}(r, \boldsymbol{c}, \xi, \mu, \lambda) = r^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\mu_i\xi_i - \sum_{i=1}^{N}\lambda_i\left(r^2 + \xi_i - ||\boldsymbol{x}_i - \boldsymbol{c}||^2\right) \tag{15.56}$$

Taking the derivatives of the above and equating to zero, the Wolfe dual form results in

$$\max_{\lambda}\left(\sum_{i=1}^{N}\lambda_i\boldsymbol{x}_i^T\boldsymbol{x}_i - \sum_{i=1}^{N}\sum_{j=1}^{N}\lambda_i\lambda_j\boldsymbol{x}_i^T\boldsymbol{x}_j\right) \tag{15.57}$$

$$\text{subject to } 0 \leq \lambda_i \leq C,\ i = 1, 2, \ldots, N \tag{15.58}$$

$$\sum_{i=1}^{N}\lambda_i = 1 \tag{15.59}$$

and the KKT conditions are

$$\mu_i\xi_i = 0 \tag{15.60}$$

$$\lambda_i\left[r^2 + \xi_i - ||\boldsymbol{x}_i - \boldsymbol{c}||^2\right] = 0 \tag{15.61}$$

$$\boldsymbol{c} = \sum_{i=1}^{N}\lambda_i\boldsymbol{x}_i \tag{15.62}$$

$$\lambda_i = C - \mu_i,\ i = 1, 2, \ldots, N \tag{15.63}$$

From these conditions the following remarks are easily deduced.

- Only points with $\lambda_i \neq 0$ contribute to the definition of the center of the optimal sphere (Eq. (15.62)). These points are known as *support vectors*.
- Points with $\xi_i > 0$ correspond to $\mu_i = 0$ (Eq. (15.60)), which leads to $\lambda_i = C$ (Eq. (15.63)) and, as (15.61) suggests, these points lie outside the sphere. We will refer to these points as *bounded support vectors*.
- Points with $0 < \lambda_i < C$ have corresponding $\mu_i > 0$ leading to $\xi_i = 0$ (Eq. (15.60)) and, as Equation (15.61) suggests, these points lie on the sphere.
- Points with $\lambda_i = 0$ correspond to $\xi_i = 0$. As (15.61) suggests, all points lying inside the sphere satisfy, necessarily, these two conditions.

As we already know from the SVM theory, exposed in Chapter 4, all we have said so far is still valid if the input space, X, is mapped into a high-dimensional Hilbert space, H, via the "kernel trick." The task now becomes

$$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i K(\boldsymbol{x}_i, \boldsymbol{x}_i) - \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \right) \tag{15.64}$$

$$\text{subject to } 0 \leq \lambda_i \leq C, \ i = 1, 2, \ldots, N \tag{15.65}$$

$$\sum_{i=1}^{N} \lambda_i = 1 \tag{15.66}$$

Maximization of the previous leads to the computation of the optimal Lagrange multipliers.

In [Ben 01] a methodology is presented that exploits the minimal enclosure sphere problem to unravel the clustering structure underlying the data set X. Let us denote the (implicit) mapping, induced by the adopted kernel function, from the original X to the high-dimensional space H as

$$\boldsymbol{x} \in X \longrightarrow \phi(\boldsymbol{x}) \in H \tag{15.67}$$

The distance of $\phi(\boldsymbol{x})$ from the center of the optimal sphere $\boldsymbol{c} = \sum_i \lambda_i \phi(\boldsymbol{x}_i)$ (as Equation (15.62) suggests) is equal to

$$r^2(\boldsymbol{x}) \equiv ||\phi(\boldsymbol{x}) - \boldsymbol{c}||^2 = \phi^T(\boldsymbol{x})\phi(\boldsymbol{x}) \tag{15.68}$$

$$+ \boldsymbol{c}^T \boldsymbol{c} - 2\phi^T(\boldsymbol{x})\boldsymbol{c} \tag{15.69}$$

or

$$r^2(\boldsymbol{x}) = K(\boldsymbol{x}, \boldsymbol{x}) + \sum_{i=1}^{N} \sum_{j=1}^{N} \lambda_i \lambda_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{15.70}$$

$$- 2 \sum_{i=1}^{N} \lambda_i K(\boldsymbol{x}, \boldsymbol{x}_i) \tag{15.71}$$

Obviously, for any of the support vectors, $\boldsymbol{x}_i$, this function is equal to the radius of the optimum sphere, that is,

$$r^2(\boldsymbol{x}_i) = r^2 \tag{15.72}$$

The contours formed in the original vector space, X, defined by the points

$$\{\boldsymbol{x} : r(\boldsymbol{x}) = r\} \tag{15.73}$$

are *interpreted* as forming cluster boundaries. It is apparent that the shapes of these contours are heavily dependent on the specific kernel function that has been adopted. In view of what we have already said, the vectors in X whose images in H

are support vectors ($0 < \lambda_i < C$) lie *on* these contours, points with images inside the sphere ($\lambda_i = 0$) lie *inside* these, contours, and points whose images lie outside the sphere ($\lambda_i = C$) lie outside these contours. A schematic representation is shown in Figure 15.21.

Contours, of course, do not suffice to define clusters. In [Ben 01] a geometric approach is suggested to differentiate points of the same and of different clusters. If the line segment, joining two points, does not cross a contour in X, these points reside in the same cluster. However, if a line segment crosses a contour it means that there are points on it (e.g., $\boldsymbol{y}$ in Figure 15.21), whose images in H lie outside the optimal sphere. Therefore, in order to check whether a line segment crosses a contour it suffices to detect some points on it with the property $r(\boldsymbol{y}) > r$. This leads to the definition of the *adjacency* matrix, A, with elements A_{ij}, referring to the pair of points $\boldsymbol{x}_i,\ \boldsymbol{x}_j \in X$, whose images lie in or on the sphere in H

$$A_{ij} = \begin{cases} 1 & \text{no points, } \boldsymbol{y}, \text{ on the respective segment exist such that } r(\boldsymbol{y}) > r \\ 0 & \text{otherwise} \end{cases} \tag{15.74}$$

Clusters are now defined as the *connected components* of the graph induced by A. An extension of the previous that combines the concept of fuzzy membership is discussed in [Chia 03]. In [Wang 07] a variant of the method is presented, which utilizes the minimal-enclosure hyperellipsoid instead of a hypersphere.

The other direction that has been followed, in order to exploit the power springing from the nonlinear nature of the "kernel trick," is of the same nature as the kernel-based principle component analysis (PCA), discussed in Chapter 6. To this end, any clustering approach that can be casted, throughout, in terms of inner product computations is a candidate to accommodate the "kernel trick." The classical k-Means algorithm, as well as some of its variants, are typical examples that have been proposed and used. See, for example, [Scho 98, Cama 05, Giro 02]. This is because the Euclidean distance, which is in one way or another at the heart of these

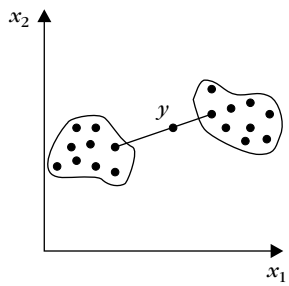

FIGURE 15.21

Point $\boldsymbol{y}$ on the line segment, crossing the contours, has an image outside the minimal enclosure sphere.

algorithms, is itself an inner product and all involved computations can be expressed in terms of inner products.

The advantage of the kernel-based methods lies in the fact that they can reveal clusters of arbitrary shapes, due to the nonlinear nature of the mapping. On the other hand, the results are very sensitive to the choice of the specific kernel function as well as of its defining parameters. Moreover, the required computational demands pose an obstacle for the use of these methods for large data sets

15.9 DENSITY-BASED ALGORITHMS FOR LARGE DATA SETS

In this framework, clusters are considered as regions in the l-dimensional space that are "dense" in points of X (Note the close agreement between this way of viewing clusters and the definition of clusters given in [Ever 01]). Most of the density-based algorithms do not impose any restrictions to the shape of the resulting clusters. Thus, these algorithms have the ability to recover arbitrarily shaped clusters. In addition, they are able to handle efficiently the outliers. Moreover, the time complexity of these algorithms is lower than $O(N^2)$, which makes them eligible for processing large data sets.

Typical density-based algorithms are the DBSCAN ([Este 96]), the DBCLASD ([Xu 98]), and the DENCLUE ([Hinn 98]). Although these algorithms share the same basic philosophy, they differ in the way the density is quantified.

15.9.1 The DBSCAN Algorithm

The "density" as it is considered in DBSCAN (Density-Based Spatial Clustering of Applications with Noise) around a point $\boldsymbol{x}$ is estimated as the number of points in X that fall inside a certain region in the l-dimensional space surrounding $\boldsymbol{x}$. In the sequel, we will consider this region to be a hypersphere $V_\varepsilon(\boldsymbol{x})$ centered at $\boldsymbol{x}$, whose radius ε is a user-defined parameter. In addition, let $N_\varepsilon(\boldsymbol{x})$ denote the number of points of X lying in $V_\varepsilon(\boldsymbol{x})$. An additional user-defined parameter is the minimum number of points, q, that must be contained in $V_\varepsilon(\boldsymbol{x})$, in order for $\boldsymbol{x}$ to be considered an "interior" point of a cluster. Before we proceed, the following definitions are in order.

Definition 5. *A point $\boldsymbol{y}$ is* directly density reachable *from a point $\boldsymbol{x}$ (see Figure 15.22a) if*

(i) $\boldsymbol{y} \in V_\varepsilon(\boldsymbol{x})$ *and*

(ii) $N_\varepsilon(\boldsymbol{x}) \geq q$.

Definition 6. *A point $\boldsymbol{y}$ is* density reachable *from a point $\boldsymbol{x}$ in X if there is a sequence of points $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p \in X$, with $\boldsymbol{x}_1 = \boldsymbol{x}$, $\boldsymbol{x}_p = \boldsymbol{y}$, such that $\boldsymbol{x}_{i+1}$ is directly density reachable from $\boldsymbol{x}_i$ (see Figure 15.22b).*

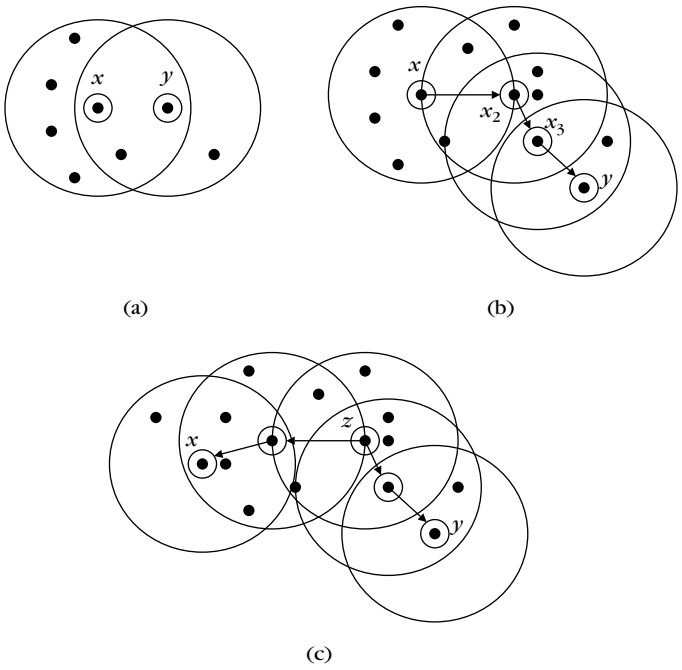

FIGURE 15.22

Assuming that $q = 5$, (a) $\boldsymbol{y}$ is directly density reachable from $\boldsymbol{x}$, but not vice versa, (b) $\boldsymbol{y}$ is density reachable from $\boldsymbol{x}$, but not vice versa, and (c) $\boldsymbol{x}$ and $\boldsymbol{y}$ are density connected (in addition, $\boldsymbol{y}$ is density reachable from $\boldsymbol{x}$, but not vice versa).

It is easy to note that if $\boldsymbol{x}$ and $\boldsymbol{y}$ in the previous definition are "dense enough" (that is, $N_\varepsilon(\boldsymbol{x}) \geq q$ and $N_\varepsilon(\boldsymbol{y}) \geq q$), the density reachability property is symmetric. On the other hand, symmetry is lost in the case where either of the points, $\boldsymbol{x}$ and $\boldsymbol{y}$, has less than q points in its neighborhood. However, in this case there must be a third point $\boldsymbol{z} \in X$ such that both $\boldsymbol{x}$ and $\boldsymbol{y}$ are density reachable from $\boldsymbol{z}$. This leads to the definition of the *density-connectivity*.

Definition 7. *A point $\boldsymbol{x}$ is* density connected *to a point $\boldsymbol{y} \in X$ if there exists $\boldsymbol{z} \in X$ such that both $\boldsymbol{x}$ and $\boldsymbol{y}$ are density reachable from $\boldsymbol{z}$ (see Figure 15.22c).*

After the previous definitions, a *cluster* C in the DBSCAN framework is defined as a nonempty subset of X satisfying the following conditions:

(i) If $\boldsymbol{x}$ belongs to C and $\boldsymbol{y} \in X$ is density reachable from $\boldsymbol{x}$, then $\boldsymbol{y} \in C$.

(ii) For each pair $(\boldsymbol{x}, \boldsymbol{y}) \in C$, $\boldsymbol{x}$ and $\boldsymbol{y}$ are density connected.

Let $C_1, \ldots, C_m$ be the clusters in X. Then, the set of points that are not contained in any of the clusters $C_1, \ldots, C_m$ is known as *noise*.

In the sequel, we define a point $\boldsymbol{x}$ as a *core point* if it has at least q points in its neighborhood. Otherwise, $\boldsymbol{x}$ is said to be a *noncore point*. A noncore point may be either a *border point* of a cluster (that is, density reachable from a core point) or a *noise point* (that is, not density reachable from other points in X). Having established the previous definitions, the following two propositions hold true ([Este 96]).

Proposition 1. If $\boldsymbol{x}$ is a core point and D is the set of points in X that are density reachable from $\boldsymbol{x}$, then D is a cluster.

Proposition 2. If C is a cluster and $\boldsymbol{x}$ a core point in C, then C equals to the set of the points $\boldsymbol{y} \in X$ that are density reachable from $\boldsymbol{x}$.

These two propositions imply that *a cluster is uniquely determined by any of its core points*. Keeping in mind the last conclusion, we proceed now with the description of the DBSCAN algorithm. Let X_{un} be the set of points in X that have not been considered yet, and let m denote the number of clusters.

DBSCAN Algorithm

- Set $X_{un} = X$
- Set $m = 0$
- While $X_{un} \neq \emptyset$ do
  - Arbitrarily select a $\boldsymbol{x} \in X_{un}$.
  - If $\boldsymbol{x}$ is a noncore point then
    - Mark $\boldsymbol{x}$ as noise point.
    - $X_{un} = X_{un} - \{\boldsymbol{x}\}$
  - If $\boldsymbol{x}$ is a core point then
    - $m = m + 1$
    - Determine all density-reachable points in X from $\boldsymbol{x}$.
    - Assign $\boldsymbol{x}$ and the previous points to the cluster C_m. The border points that may have been marked as noise are also assigned to C_m.
    - $X_{un} = X_{un} - C_m$
  - End { if }
- End { while }

A delicate point of the algorithm is the following. Suppose that a border point $\boldsymbol{y}$ of a cluster C has currently been selected by the algorithm. This point will be marked as a noise point, through the first branch of the *if* statement of the algorithm. Note, however, that when, later on, a core point $\boldsymbol{x}$ of C will be considered from which $\boldsymbol{y}$ is density reachable, then $\boldsymbol{y}$ will be identified as a density-reachable point from $\boldsymbol{x}$ and will be assigned to C. On the other hand, if $\boldsymbol{y}$ is a noise point it will be marked

as such and because it is not density reachable by any of the core points in X its "noise" label will remain unaltered.

Remarks

- The results of the algorithm are greatly influenced by the choice of ε and q. Different values of the parameters may lead to totally different results. One should select these parameters so that the algorithm is able to detect the least "dense" cluster. In practice, one has to experiment with several values for ε and q in order to identify their "best" combination for the data set at hand.
- Implementation of the algorithm by adopting the R^*-tree data structure can achieve time complexity of $O(N \log_2 N)$ for low-dimensional data sets ([Berk 02]).
- The DBSCAN is not appropriate for cases where the clusters in X exhibit significant differences in density, and it is not well suited for high-dimensional data ([Xu 05]).
- It is also worth noting the close resemblance of the previous algorithm with the CDADV algorithm described in Section 15.4.3. However, the latter is suitable for discrete-valued vectors.

An extension of DBSCAN that overcomes the necessity of choosing carefully the parameters ε and q is the OPTICS (Ordering Points To Identify the Clustering Structure) algorithm ([Anke 99]). This generates a density-based cluster ordering, representing the intrinsic hierarchical cluster structure of the data set in a comprehensible form ([Bohm 00]). Experiments indicate that the run time of OPTICS is roughly equal to 1.6 of the runtime required by DBSCAN ([Berk 02]). On the other hand, in practice one has to run DBSCAN more than one time for different values of ε and q. Another extension of DBSCAN is given in [Sand 98].

15.9.2 The DBCLASD Algorithm

An alternative to the DBSCAN approach is followed by the DBCLASD (Distribution-Based Clustering of LArge Spatial Databases) algorithm ([Xu 98]). In this case, the distance d of a point $\boldsymbol{x} \in X$ to its nearest neighbor is considered a *random variable* and the "density" is quantified in terms of probability distribution of d. In addition, it is assumed that the points in each cluster are uniformly distributed. However, not all points in X are assumed to be uniformly distributed. Based on this assumption, the distribution of d can be derived analytically ([Xu 98]). In the framework of DBCLASD, a *cluster* C is defined as a nonempty subset of X with the following properties.

(a) The distribution of the distances, d, between points in C and their nearest neighbors is in agreement with the distribution derived theoretically, within some confidence interval. (This is carried out by using the χ^2 test.)

(b) It is the *maximal set* with the previous property. That is, the insertion of additional points neighboring to points in C will cause (a) not to hold anymore.

(c) It is *connected*. Having applied a grid of cubes on the feature space, this property implies that for any pair of points $(\boldsymbol{x}, \boldsymbol{y})$ from C there exists a path of adjacent cubes that contains at least one point in C that connects $\boldsymbol{x}$ and $\boldsymbol{y}$.

In this algorithm, the points are considered in a sequential manner. A point that has been assigned to a cluster may be reassigned to another cluster at a later stage of the algorithm. In addition, some points are not assigned to any of the clusters determined so far, but they are tested again at a later stage.

Among the merits of the algorithm is that it is able to determine arbitrarily shaped clusters of various densities in X, and it requires no parameter definition. Experimental results given in [Xu 98] indicate that the runtime of DBCLASD is roughly twice the runtime of DBSCAN, whereas DBCLASD outperforms CLARANS by a factor of at least 60.

15.9.3 The DENCLUE Algorithm

In the previously described methods, "density" was defined based on the distance between neighboring points $\boldsymbol{x} \in X$, in either a deterministic (DBSCAN) or a probabilistic setting (DBCLASD). In the following, "density" is quantified via a different route. Specifically, for each point $\boldsymbol{y} \in X$ a so-called *influence function* $f^{\boldsymbol{y}}(\boldsymbol{x}) \geq 0$ is defined, which decreases to zero as $\boldsymbol{x}$ "moves away" from $\boldsymbol{y}$. Typical examples of $f^{\boldsymbol{y}}(\boldsymbol{x})$ include

$$f^{y}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } d(\boldsymbol{x}, \boldsymbol{y}) < \sigma \\ 0, & \text{otherwise} \end{cases} \tag{15.75}$$

and

$$f^{\boldsymbol{y}}(\boldsymbol{x}) = e^{-\frac{d(\boldsymbol{x},\boldsymbol{y})^2}{2\sigma^2}} \tag{15.76}$$

where $d(\boldsymbol{x}, \boldsymbol{y})$ denotes the distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ (this may be the Euclidean distance or any other dissimilarity measure) and σ is a user-defined parameter. Then, the *density function* based on X is defined as

$$f^{X}(\boldsymbol{x}) = \sum_{i=1}^{N} f^{\boldsymbol{x_i}}(\boldsymbol{x}) \tag{15.77}$$

Note the similarity of the previous with the Parzen windows approximation of the density function $p(\boldsymbol{x})$, discussed in Chapter 2. The goal here is to identify all "significant" local maxima, $\boldsymbol{x}_j^*, j = 1, \ldots, m$, of $f^X(\boldsymbol{x})$ and then to create a cluster C_j for each $\boldsymbol{x}_j^*$ and to assign to C_j all the points of X that lie in the *region of attraction* of $\boldsymbol{x}_j^*$. The region of attraction of $\boldsymbol{x}_j^*$ is defined as the set of points $\boldsymbol{x} \in \mathcal{R}^l$ such that if a "hill-climbing" method (a hill-climbing method aims at determining the local

maxima of a function; a typical example is the steepest ascent method, see Appendix C) is applied, initialized by $\boldsymbol{x}$, it will terminate arbitrarily close to $\boldsymbol{x}_j^*$.

As can be seen from Eqs. (15.75) and (15.76), σ quantifies the influence of a specific data point of X in the $\mathcal{R}^l$ space. In addition, a parameter ξ is required in order to quantify the significance of a local maximum. Thus, a local maximum is considered significant if $f^X(\boldsymbol{x}_j^*) \geq \xi$. Since by definition $f^{\boldsymbol{x}_i}(\boldsymbol{x})$ decreases as $\boldsymbol{x}$ moves away from $\boldsymbol{x}_i$, it is expected that $f^X(\boldsymbol{x})$ can be approximated satisfactorily by

$$\hat{f}^X(\boldsymbol{x}) = \sum_{\boldsymbol{x}_i \in Y(\boldsymbol{x})} f^{\boldsymbol{x}_i}(\boldsymbol{x}) \tag{15.78}$$

where $Y(\boldsymbol{x})$ is the set of points in X that lie "close" to $\boldsymbol{x}$.

A well-known representative that evolves around the previous methodology is the DENCLUE (DENsity-based CLUstEring) algorithm ([Hinn 98]). Since the "significant" local maxima are expected to be located in regions "dense" in data (and in order to reduce time complexity), DENCLUE applies first a *preclustering step* to determine regions that are dense in points of X. To this end, an l-dimensional grid of edge-length 2σ is applied on the feature space that contains X and the set D_p of the (hyper)cubes that contain at least one point of X is determined. Then, the subset D_{sp} of D_p that contains the highly populated cubes of D_p (a high populated cube contains at least $\xi_c > 1$ points of X, where ξ_c is a user-defined parameter) is determined. For each highly populated cube, c, a *connection* is defined with all neighboring cubes c_j in D_p for which $d(\boldsymbol{m}_c, \boldsymbol{m}_{c_j})$ is no greater than 4σ, where $\boldsymbol{m}_c$ and $\boldsymbol{m}_{c_j}$ are the mean values of the points in the respective cubes.

Once the previous step has been completed, DENCLUE proceeds as follows. First, it considers the set D_r of the highly populated cubes and the cubes that have at least one connection with a highly populated cube. The search for local maxima is constrained within the cubes in D_r. Then, for each point $\boldsymbol{x}$ in a cube $c \in D_r$ its neighboring points that influence it are considered. Specifically, $Y(\boldsymbol{x})$ contains all points that belong to cubes c_j in D_r such that the means of c_js lie at a distance less than $\lambda\sigma$ from $\boldsymbol{x}$ (typically $\lambda = 4$). Then, a hill climbing method ([Hinn 98]) starting from $\boldsymbol{x}$ is applied. The local maximum $\boldsymbol{x}^*$ to which the method converges is then tested to see if it is a significant local maximum (i.e., if $\hat{f}^X(\boldsymbol{x}^*) \geq \xi$). If it is not, no additional action is taken. Otherwise, if a cluster associated to $\boldsymbol{x}^*$ has not been created yet it is created now and $\boldsymbol{x}$ is assigned to it. In addition, all points of X for which the hill-climbing method leads to $\boldsymbol{x}^*$ will be assigned to the same cluster. Shortcuts that allow the assignment of points to clusters based on certain conditions to be satisfied, without having to apply the hill-climbing procedure, are also discussed in [Hinn 98].

The method, like all density-based methods, is able to detect arbitrarily shaped clusters ([Hinn 98]). In addition, DENCLUE deals with noise very satisfactorily. A procedure for selecting appropriate values for the parameters σ and ξ is discussed in ([Hinn 98]). The worst-case time complexity of DENCLUE is $O(N \log_2 N)$; that

is, of the same order as the time complexity of DBSCAN. However, experimental results reported in [Hinn 98] indicate that DENCLUE can be significantly faster than DBSCAN, and the reported experiments indicate that the *average* time complexity for DENCLUE is $O(\log_2 N)$. This is a consequence of the fact that only a small fraction of the total number of points in X are considered for determining the clusters. It has also been pointed out that the algorithm works efficiently with high-dimensional data, and it has been applied to a molecular biology experiment ([Hinn 98]).

A very recent density-based clustering algorithm, named ADACLUS, is discussed in [Noso 08]. It is able to discover clusters of various shapes and densities and to detect boundaries of the clusters. Also, it is robust to noise and it does not oblige the user to define the values of certain parameters. Finally, it has low computational requirements, which is very important in the processing of large data sets.

15.10 CLUSTERING ALGORITHMS FOR HIGH-DIMENSIONAL DATA SETS

As we have already discussed at several points in this book, concerning algorithms that have been designed to meet the needs of large data sets, special data structures for indexing data such as R-trees, $k-d$ trees, and so on have been employed in order to facilitate access to the data. As it is pointed out in [Berk 02], the performance of most of these data structures degrades to the level of sequential search for $l > 20$. Thus, one can consider the value of 20 as a "lower bound" that quantifies high dimensionality. In fact, there are applications such as bioinformatics or web mining in which the dimensionality of the feature space can be as high as a few thousands.

Most of the algorithms discussed so far consider all dimensions of the feature space simultaneously, trying to utilize as much of the available information as possible. However, this approach may turn to be problematic in high-dimensional spaces. One source of problems is the "curse of dimensionality" which besides the complexity issues discussed in Section 2.5.6 now shows another of its "faces." Having fixed the number of data points, N, as the dimensionality of the feature space increases, the points are spread out in the space. This can easily be verified by considering two points in the two-dimensional space. Then add a third dimension to each of them and compare the resulting Euclidean distances for both cases. As the points spread out in very high-dimensional spaces they become almost equidistant. Clearly, in such a case the terms *similarity* and *dissimilarity* between two points become increasingly meaningless ([Pars 04]). A second source of problems is that often in very high-dimensional spaces only a small fraction of the features contributes to the formation of each cluster. In other words, clusters can be identified in subspaces of the original feature space. In terms of Figure 15.23a, the clusters C_1 and C_2 are the result of the concentration of the values of x_2 within two small intervals, whereas the values along x_1 do not show such a preference. In a similar manner, the clusters in Figure 15.23b are due to data concentrations in the (x_1, x_2) subspace. Observe that these clusters could be identified by projecting the points in (x_1, x_2).

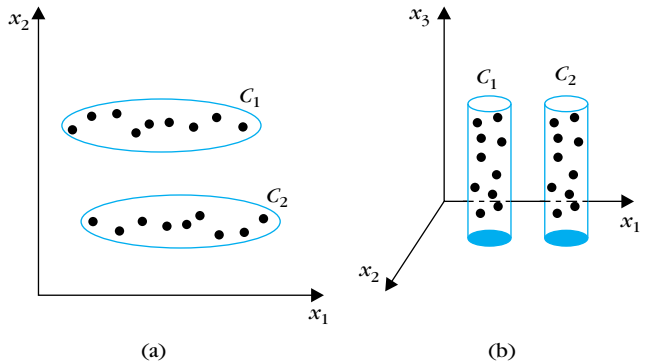

FIGURE 15.23

(a) The clusters C_1 and C_2 are the result of concentrations of the projections of the data points along x_2. (b) The clusters in this three-dimensional case are the result of concentrations of the projections of the data points in the (x_1, x_2) subspace.

Clearly, a way out of this situation is to work on subspaces of dimension lower than l. In the sequel we discuss two main approaches to this direction: the *dimensionality reduction clustering approach* and the *subspace clustering approach*.

15.10.1 Dimensionality Reduction Clustering Approach

The general idea of this approach is to identify an l'-dimensional space $H_{l'}$ ($l' < l$), project the data points in X onto it, and apply a clustering algorithm on the projections of the points of X into $H_{l'}$. For the identification of $H_{l'}$ one may use (a) *feature generation methods*, (b) *feature selection methods*, and (c) *random projections*. In the sequel, we "touch" briefly the first two methodologies, in that they build upon the techniques treated in more detail in Chapters 5 and 6, and continue to pursue the random projection method.

Feature generation methods, such as the principal component analysis (PCA) and the singular value decomposition (SVD), generally preserve the distances between the points in the high-dimensional space when these are mapped to the lower-dimensional space. These methods are very useful in producing compact representations of the original high-dimensional feature space. Algorithms that adopt feature generation methods are discussed in, for example, [Deer 90, Ding 99] and [Ding 02]. A notable characteristic of the latter work is that feature generation is integrated with the clustering algorithm (*k*-means or *EM*) and as a consequence applies iteratively as the clustering algorithm evolves. Feature generation methods can be proved useful in cases where a significant number of features contributes to the identification of the clusters. In addition to PCA or SVD, other techniques used for dimensionality reduction can also be employed, such as nonlinear PCA, ICA., and so on (see Chapter 6).

An alternative approach is to employ feature selection methods, in order to identify those features that are the main contributors to the formation of the clusters.

However, the feature selection methods proposed for the supervised case are no longer suitable in the clustering framework. In general, the criteria used to evaluate the "goodness" of a specific subset of features follows either the *wrapper model* or the *filter model* ([Koha 97]). According to the former, the clustering algorithm, $\mathcal{C}$, is first chosen and a subset of features, $\mathcal{F}_i$, is evaluated through the results obtained from the application of $\mathcal{C}$ on the data set X, where for each point only the features in $\mathcal{F}_i$ are taken into account. According to the latter, the evaluation of a subset of features is carried out using intrinsic properties of the data, prior to the application of the clustering algorithm. Feature selection methods are discussed in [Blum 97, Liu 98, Pena 01, Yu 03]. The feature selection approach is useful when all clusters lie in the same subspace of the feature space.

Clustering Using Random Projections

Unlike the previously cited dimensionality reduction methods, where $H_{l'}$ is identified deterministically, in the present case $H_{l'}$ will be identified in a random manner. Noting that a projection from an l-dimensional space, H_l, to an l'-dimensional space $H_{l'}$ ($l' < l$) is uniquely defined via an $l' \times l$ *projection matrix* A, the issues to be addressed here are: (a) the proper estimate of l' and (b) the definition of the projection matrix. In [Achl 01, Dasg 99] estimates of l' are given that guarantee (with a certain probability) that the distances between the points of the data set X, in the original feature space H_l, will be preserved up to a factor $1 \pm \varepsilon$ (where $\varepsilon > 0$ is an arbitrarily chosen constant) after the projection of X to a randomly chosen l'-dimensional space $H_{l'}$, whose projection matrix is constructed by following certain simple probabilistic rules. More specifically, in [Dasg 99] l' is shown to be bounded below by $4(\varepsilon^2/2 - \varepsilon^3/3)^{-1} \ln N$. (Note, however, that the choice for l' does not guarantee, even in probability that the separation of clusters is preserved in the general case of arbitrarily shaped clusters. This problem is studied in [Dasg 00] for the special case where the clusters stem from Gaussian distributions.)

As far as A is concerned, one way to construct it is to set each of its entries equal to a value stemming from an i.i.d. zero-mean unit variance Gaussian distribution and then to normalize each row to the unit length ([Fern 03]). Another way is to set each entry of A equal to -1 or $+1$, with probability 0.5. Still another way to generate A is to set each of its entries equal to $+\sqrt{3}$ with probability 1/6, $-\sqrt{3}$ with probability 1/6, or 0 with probability 2/3 ([Achl 01]).

After the definition of the projection matrix A, one proceeds by projecting the points of X into $H_{l'}$ and performing a clustering algorithm on the projections of the points of X into $H_{l'}$. However, a significant problem may arise here. Different random projections may lead to totally different clustering results. One way to cope with this problem is to perform several random projections, apply a clustering algorithm on the projections of X to each of them, and combine the clustering results in order to produce the final clustering.

A method in this spirit is discussed in the sequel ([Fern 03]). First, the dimension l' of the lower-dimensional space is selected and r different projection matrices, $A_1, \ldots, A_r$, are generated using the first of the three ways described previously. Then, the Generalized Mixture Decomposition Algorithmic Scheme (GMDAS) for

compact and hyperellipsoidal clusters (Section 14.2) is applied on each one of the r random projections of X. Let $P(C_j^s|\boldsymbol{x}_i)$ denote the probability that $\boldsymbol{x}_i$ belongs to the j-th cluster in the s-th projection (C_j^s) after the execution of GMDAS on each projection of X.

For each projection, a similarity matrix P^s is created, whose (i,j) element is defined as

$$P_{ij}^s = \sum_{q=1}^{m_s} P(C_q^s|\boldsymbol{x}_i)P(C_q^s|\boldsymbol{x}_j) \tag{15.79}$$

where m_s is the number of clusters in the s-th projection. Actually, P_{ij}^s is the probability that $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to the same cluster at the s-th projection. Then, the average proximity matrix P is defined, so that its (i,j) element is equal to the average of $P_{ij}^s, s = 1, \ldots, r$.

In the sequel, the Generalized Agglomerative Scheme (GAS) (Section 13.2) is employed in order to identify the final clusters. The similarity between two clusters C_i and C_j is defined as $\min_{\boldsymbol{x}_u \in C_i,\ \boldsymbol{x}_v \in C_j} P_{uv}$ (actually, this choice complies with the philosophy of the complete link algorithm).

For the estimation of the number of clusters m underlying X, it is proposed to let GAS run until all points of X are agglomerated to a single cluster. Then, the similarity between the closest pair of clusters, determined in each iteration of GAS, is plotted versus the number of iterations. The most abrupt decrease in the plot is determined and m is set equal to the value corresponding to this decrease. Results reported in [Fern 03] show that in principle this method produces better clusters and is more robust than the principal component analysis method followed by the EM algorithm. The complexity of this algorithm is controlled by the GAS algorithm and is larger than $O(N^2)$.

15.10.2 Subspace Clustering Approach

The general strategy followed by the dimensionality reduction methods was to project the data set into a lower-dimensional space and to apply a clustering algorithm on the projections of the data points in this space. This strategy copes well with the problems rising from the "curse of dimensionality," as stated previously. However, these methods cannot deal well with the cases where (a) a small number of features contributes to the identification of clusters and (b) different clusters reside in different subspaces of the original feature space. The latter situation is depicted in Figure 15.24.

A way to overcome these shortcomings is to develop special types of clustering algorithms that are able to search for clusters within the *various subspaces of the feature space*. In other words, these algorithms will reveal the clusters as well as the subspaces where they reside. Algorithms of this type are known as *subspace clustering algorithms* (SCAs). In contrast to all of the algorithms considered so far in the book, SCAs allow us to seek for clusters in different subspaces of the original feature space. SCAs algorithms can be divided into two categories: (a) grid-based

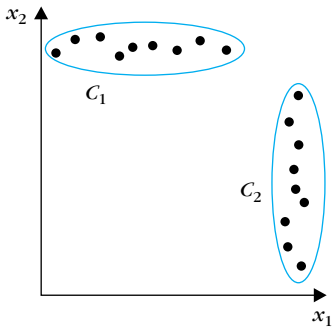

FIGURE 15.24

By projecting the data points in both x_1 and x_2 directions, it is readily seen that cluster C_1 lies in the x_2 subspace, whereas cluster C_2 lies in x_1.

SCAs and (b) point-based SCAs. In the sequel, we explore the basic philosophy behind the algorithms in each of the two categories and focus on some typical representatives.

Grid-based Subspace Clustering Algorithms (GBSCAs)

The main strategy adopted by these algorithms consists of the following steps: (a) identify the subspaces of the feature space that are *likely* to contain clusters, (b) determine the clusters lying in each of these subspaces, and (c) obtain descriptions of the resulting clusters.

The algorithms of this family apply an l-dimensional grid on the feature space and identify the subspaces that are likely to contain clusters, based on the k-dimensional units (boxes) ($k \leq l$) defined by the grid. However, the consideration of all possible subspaces becomes infeasible, especially when high-dimensional data sets are considered. To solve this problem, the algorithms establish certain criteria that are indicative of the presence of clusters in a subspace. These criteria must comply with the so-called *downward closure property*, which states that if a criterion is satisfied in a k-dimensional space, it is also satisfied in all of its $(k-1)$-dimensional subspaces. This allows the identification of the subspaces in an iterative bottom-up fashion, from lower to higher dimensional subspaces.

Having established the subspaces via the previously described procedure, the algorithms seek for clusters in each of them. Clusters are identified as maximally connected components of units in each subspace. In the sequel, we describe in more detail the CLIQUE and ENCLUS algorithms, which are among the most popular representatives of this family.

The CLIQUE (CLustering In QUEst) Algorithm

CLIQUE ([Agra 98]) partitions the feature space by applying an l-dimensional grid on it of edge size ξ (a user-defined parameter). Each unit u is written as $u_{t_1} \times \ldots \times u_{t_k}$

$(t_1 < \ldots < t_k, k \leq l)$, or for convenience as $(u_{t_1}, \ldots, u_{t_k})$, where $u_{t_i} = [a_{t_i}, b_{t_i})$ is a right-open interval in the partitioning of the t_i-th dimension of the feature space. For example, $t_1 = 2, t_2 = 5, t_3 = 7$ indicates a unit lying in the subspace spanned by x_2, x_5, and x_7 dimensions.

Before we proceed, some definitions are in order. We say that a point $\boldsymbol{x}$ is *contained* in a k-dimensional unit $u = (u_{t_1}, \ldots, u_{t_k})$ if $a_{t_i} \leq x_{t_i} < b_{t_i}$ for all t_i. The *selectivity* of a unit u is defined as the fraction of the total number of data points (N) contained in u. A unit u is called *dense* if its selectivity is greater than a user-defined threshold τ (see Figure 15.25). We say that two k-dimensional units, $u = (u_{t_1}, \ldots, u_{t_k})$ and $u' = (u'_{t_1}, \ldots, u'_{t_k})$, share a *face* if there are $(k-1)$ dimensions (e.g., $x_{t_1} \ldots, x_{t_{k-1}}$), such that $u_{t_j} = u'_{t_j}$, $j = 1, 2, \ldots, k-1$, and either $a_{t_k} = b'_{t_k}$ or $b_{t_k} = a'_{t_k}$. For example, units u_{12} and u_{22} in Figure 15.25 share a one-dimensional face. Two k-dimensional units u_1 and u_2 are said to be *directly connected* if they have in common a $(k-1)$-dimensional face. In addition, two k-dimensional units are said to be *connected* if there exists a sequence of k-dimensional units $v_1, \ldots, v_s$, with $v_1 = u_1$ and $v_s = u_2$, such that each pair (v_i, v_{i+1}) of units is directly connected. Finally, in the present framework, a *cluster* is defined as a maximal set of connected dense units in k dimensions.

We proceed now with the description of the algorithm. CLIQUE consists of three main stages. In the first stage identification of the subspaces that contain clusters takes place. This is carried out by first identifying all k-dimensional dense units $(1 \leq k \leq l)$ and then selecting those subspaces that contain dense units.

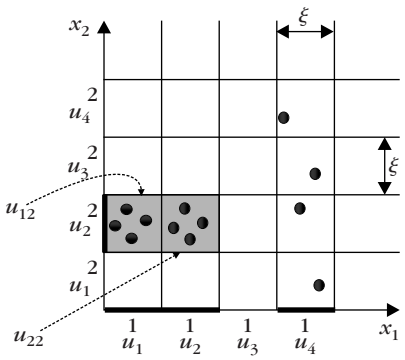

FIGURE 15.25

A two-dimensional grid of lines of edge size ξ is applied in the two-dimensional feature space, defining two-dimensional and one-dimensional boxes (units). In the figure, u_i^q denotes the i-th one-dimensional unit along x_q and u_{ij} denotes the two-dimensional unit resulting from the Cartesian product of the i-th and j-th intervals along x_1 and x_2, respectively. In addition, let $\tau = 3$. Then u_1^1, u_2^1, u_4^1, u_2^2 are one-dimensional dense units, each containing $4, 4, 4$, and 9 points, respectively, whereas u_{12} and u_{22} are two-dimensional dense units, each containing 4 points.

(In other words, the criterion a subspace has to satisfy to be selected is to have at least one dense unit.) We proceed by identifying dense units in a bottom-up fashion from lower to higher dimensionality. First, all one-dimensional dense units are identified, along each dimension of the feature space. At each step, the set D_k of the k-dimensional dense units is determined based on the set D_{k-1} of the $(k-1)$-dimensional dense units. To this end, a k-dimensional dense unit u in D_k results from two $(k-1)$-dimensional dense units in D_{k-1} that share a $(k-2)$-dimensional face *and at the same time*, all $(k-1)$-dimensional projections must belong to D_{k-1}. Clearly, this procedure can be terminated to a dimension less than l. Having identified the dense units, the subspaces that contain at least one such unit can in turn be determined.

The rationale behind this approach relies on the *downward closure property* of the density, which states that: "if there is a dense unit u in a k-dimensional space, there are also dense units in the projections of u in all $(k-1)$-dimensional subspaces of the k-dimensional space" (Note how the downward closure property appears in Figure 15.25) [Pars 04].

This procedure may lead to an increasing number of dense units in all subspaces (especially for large l), which increases significantly the required computational time. This problem can be overcome as follows. After the completion of the procedure, we consider all subspaces that contain dense units and sort *the subspaces* according to their *coverage*; that is, the fraction of the number of points of the original data set they contain. Then, subspaces with large coverage are selected and the rest are pruned. The threshold under which a coverage is considered "low" is determined by the optimization of a suitably defined Minimum Description Length criterion function ([Agra 98]).

During the second stage, CLIQUE identifies the clusters. The input to this stage are the subspaces with high coverage as determined by the previous step. *The clusters are formed in each subspace separately*. Specifically, for each selected subspace the dense units in it are considered. One such unit is randomly picked, and all of the dense units connected to it are identified and we assign all of them to a new cluster C. If there are other dense units left in the current subspace, the same procedure is applied to form a second cluster, and so on.

Finally, the goal of the third stage is to derive a *minimal cluster description* for each cluster; that is, to express each cluster as the minimum possible union of hyperrectangular regions (see Figure 15.26a). This stage consists of two phases. During the first phase, we pick randomly a (dense) unit of a cluster C formed by the previous stage, and we grow it in both directions along a dimension, trying to cover as many units in C as possible. Then the unit is grown along the second direction (as in the previous case) and we continue until all dimensions have been considered once. If there are uncovered units, we repeat the procedure starting from a new uncovered point. For example, the description in Figure 15.26a may result by selecting u_1 and growing along x_1 and then along x_2, producing A. In a similar way, B is produced starting from the (uncovered by A) unit u_2. In the second phase we consider all covers produced for each cluster and remove those whose

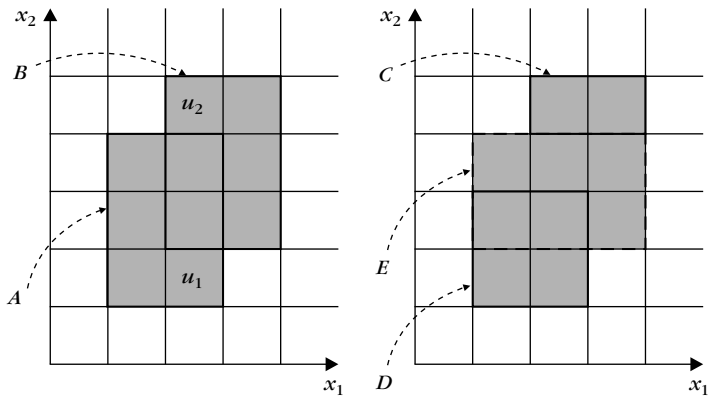

FIGURE 15.26

The minimal cluster description of the shaded region is $A \cup B$, shown in (a), whereas a nonminimal cluster description of the same region ($C \cup D \cup E$) is shown in (b).

units are covered by at least another cover. Example 15.10 will help us gain more insight on how CLIQUE works.

Example 15.10

Consider the two-dimensional data set shown in Figure 15.27, where the two-dimensional grid applied is also shown. By u_i^q, we denote the i-th one-dimensional unit along the q-th dimension, whereas by u_{ij} we denote the two-dimensional unit which results from the Cartesian product of the i-th unit along the first direction (x_1) times the j-th unit along the second direction (x_2). Assume that $\xi = 1$, which implies that $u_i^q = [i - 1, i)$, and that $\tau = 8\%$ (since we have 69 points in total, each unit with more than 5 points is considered to be dense). In addition, the points in units u_{48}, u_{58}, u_{75}, u_{76}, u_{83}, and u_{93} are collinear in the direction of one of the two axes. This implies that, for example, u_{48} contributes to u_8^2 a single point. Similar observations hold for the rest of the units. Applying the first stage of the CLIQUE algorithm, we identify the set D_1 of the one-dimensional dense units, which equals

$$D_1 = \{u_2^1, u_3^1, u_4^1, u_5^1, u_8^1, u_9^1, u_1^2, u_2^2, u_3^2, u_5^2, u_6^2\}$$

Based on D_1, D_2 is then determined and it equals

$$D_2 = \{u_{21}, u_{22}, u_{32}, u_{33}, u_{83}, u_{93}\}$$

Note that although each of the u_{48}, u_{58}, u_{75}, u_{76} contains more than 5 points they are not included in D_2, since each of them has one one-dimensional projection outside D_1 (for example, for u_{75}, u_5^2 belongs to D_1, wheras u_7^1 is not included in D_1). Furthermore, although it seems unnatural for u_{83} and u_{93} to be included in D_2 they are included, since u_3^2 is dense. However, u_3^2 is characterized as dense not because of the projections of the points in u_{83} and

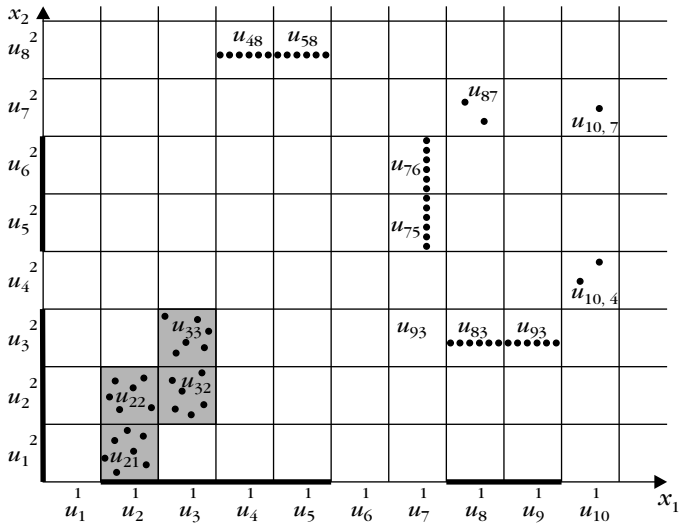

FIGURE 15.27

The set up of Example 15.10.

u_{93} but because of the projections of the points in u_{33}. Skipping the pruning step associated with the coverage criterion, the second stage of the algorithm ends up with the one-dimensional clusters $C_1 = \{u_2^1, u_3^1, u_4^1, u_5^1\}$, $C_2 = \{u_8^1, u_9^1\}$, $C_3 = \{u_1^2, u_2^2, u_3^2\}$, $C_4 = \{u_5^2, u_6^2\}$ and the two-dimensional clusters $C_5 = \{u_{21}, u_{22}, u_{32}, u_{33}\}$, $C_6 = \{u_{83}, u_{93}\}$.
Finally, the representation of the clusters after the completion of the third stage is as follows: $C_1 = \{(x_1) : 1 \leq x_1 < 5\}$, $C_2 = \{(x_1) : 7 \leq x_1 < 9\}$, $C_3 = \{(x_2) : 0 \leq x_2 < 3\}$, $C_4 = \{(x_2) : 4 \leq x_2 < 6\}$, $C_5 = \{(x_1, x_2) : 1 \leq x_1 < 2,\ 0 \leq x_2 < 2\} \cup \{(x_1, x_2) : 2 \leq x_1 < 3,\ 1 \leq x_2 < 3\}$, $C_6 = \{(x_1, x_2) : 7 \leq x_1 < 9,\ 2 \leq x_2 < 3\}$. Note that C_2 and C_6 are essentially the same cluster, which is reported twice by the algorithm.

Among the performance features of the CLIQUE algorithm is that it automatically determines the subspaces of the original feature space where high-density clusters exist. In addition, it is insensitive to the order the data are presented to the algorithm and does not impose any *data* distribution hypothesis on the data set. Also, it scales linearly with N but scales exponentially with l. In addition, the accuracy of the determined clusters may be degraded because the clusters are not given in terms of the points in X but as unions of dense units. Moreover, the performance of the algorithm is heavily dependent on the choices of ξ and τ, and it is not obvious how they must be selected in practice. Also, there is a large overlap among the reported clusters because for each dense unit (the structuring element of a cluster) all of its projections are also dense and contribute to clusters in lower-dimensional subspaces. Finally, there is a risk of losing small but meaningful clusters after the pruning of subspaces based on their coverage.

The ENCLUS Algorithm

An alternative grid-based SCA is the ENCLUS algorithm ([Chen 99]), which adopts the three-stage philosophy of CLIQUE. In particular, the last two stages of both algorithms are identical. However, during the first stage, ENCLUS seeks for subspaces with (a) *high coverage* (that is, high percentage of points covered by all dense units of the subspace), (b) *high density* of points in the dense units in the subspace, and (c) *high correlation* among the dimensions of the subspace. All of these requirements are indicative of a subspace with nonrandom structure, and these can be described utilizing the notion of entropy. The rationale behind the choice of entropy is that typically a subspace with a strong clustering structure has lower entropy than a subspace where data do not show a clustering tendency. Additional supportive evidence for the above choice is that under certain conditions the entropy decreases as the coverage increases. Moreover, entropy also decreases as the density increases.

In order to measure the entropy $H(X^k)$ of a k-dimensional subspace X^k ($k \leq l$) of X, a k-dimensional grid is applied on it and the percentage p_i of points that fall in each unit of the grid is calculated. (It is advisable to select the size of the edge of the grid such that each unit contains at least a minimum number of points. In [Devo 95] this minimum value is taken to be equal to 35.) Then, $H(X^k)$ is defined as

$$H(X^k) = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{15.80}$$

where n is the total number of units. Also, an additional entropy-based measure, called *interest*, which quantifies the degree of correlation among the dimensions of a subspace, is defined as

$$\text{interest}\ (X^k) = \sum_{j=1}^{k} H(x_{t_j}^k) - H(X^k) \tag{15.81}$$

where $t_1 < \ldots < t_k$ ($k \leq l$) with $x_{t_j}^k$ denoting the j-th dimension of the X^k subspace. The higher the interest the stronger the correlation among the dimensions of X^k. Note that the minimum value of interest is 0, which is achieved when $x_{t_j}^k$ are *independent* from each other. (By definition, $x_{t_j}^k$s are *independent* if and only if $H(X^k) = \sum_{j=1}^{k} H(x_{t_j}^k)$ [Chen 99].)

A subspace with "low" entropy (below a user-defined threshold ω) is considered to have *good clustering*. A *significant* subspace is a subspace with good clustering *and* interest above a user-defined threshold ε. Following these necessary definitions, we are now ready to outline the way ENCLUS identifies significant subspaces. (In other words, a subspace is selected provided it satisfies the "low entropy" and "high interest" criteria.) Let B_k denote the set of significant k-dimensional subspaces and D_k the set of subspaces having good clustering but low interest. As in CLIQUE, a bottom-up strategy in the consideration of the subspaces is adopted. First, the set A_1 of all one-dimensional subspaces is considered

and each is examined if it is significant or has a good clustering but is of low interest (below ε). In the former case, the subspace at hand is assigned to B_1, whereas the latter is assigned to D_1. In an iterative procedure, having defined B_k and D_k we determine the set A_{k+1} as follows: a $(k+1)$-dimensional subspace, $\mathcal{E}$, is assigned to A_{k+1} if (a) it is the result of the union of two k-dimensional subspaces in D_k, sharing $(k-1)$ dimensions, and (b) all k-dimensional projections of $\mathcal{E}$ belong to D_k. If A_{k+1} is nonempty, the procedure repeated. Otherwise, the procedure terminates and returns all significant subspaces found; that is, $B_1 \cup \ldots \cup B_k$.

The rationale behind this approach relies on the *downward closure property* of entropy, which states that if a k-dimensional space is of low entropy all of its $(k-1)$-dimensional subspaces are also of low entropy ([Chen 99]).

A variant of this scheme, based on a different criterion for measuring the correlation among the different dimensions of a subspace, is discussed in [Chen 99]. ENCLUS shares most of the features of CLIQUE; that is, insensitivity in the order of consideration of the data, the utility to unravel (in principle) arbitrarily shaped clusters, overlap among reported clusters, and linear dependence of the computational time on N. Also, the performance of ENCLUS is heavily dependent on the choice of the edge of the grid as well as on the parameters ω and ε.

Another algorithm of the grid-based SCA family is the so-called MAFIA algorithm (Merging of Adaptive Finite IntervAls) ([Goil 99]). In contrast to both algorithms discussed previously, where the grid applied on a subspace is static (that is, the edge size ξ is fixed for all dimensions), in MAFIA the grid is adaptively adjusted to match the distribution of the data. More specifically, the algorithm divides each dimension x_i of the feature space into (one-dimensional) windows of small size d. Then, it projects the points of the data set on each dimension, determines the projections that lie in each window, and sets the value of the corresponding window equal to the maximum among the projections lying in the window. In the sequel, it considers each dimension separately, scans its windows from left to right, and merges two adjacent windows if their values happen to be within a user-defined threshold β (a typical value for β is 20%). In the case where all windows in x_i are merged to a single one, which implies that the data are uniformly distributed along x_i, MAFIA applies on this a grid of fixed edge size. The unions of the windows resulting from this procedure are the one-dimensional units. Once these have been established, all other k-dimensional units can be defined $(1 \leq k \leq l)$, and the algorithm proceeds exactly as CLIQUE.

Parallel versions of the MAFIA algorithm are discussed in [Goil 99]. As it was the case with the CLIQUE and ENCLUS, MAFIA is sensitive on the choice of the related parameters. The required computational time increases linearly with N and performs better than the other two algorithms of this family, as far as the dimensionality is concerned. However, the computational time still grows exponentially as l increases ([Pars 04]). Finally, considering its parallel implementation as well as some other improvements, it is reported that MAFIA performs much faster than CLIQUE. Other algorithms that exploit the adaptive partition of each dimension

are the *cell-based clustering method (CBF)* ([Chan 02]) and the *CLTree algorithm* ([Liu 00]).

Point-based Subspace Clustering Algorithms (PBSCA)

According to the philosophy of grid-based algorithms, clusters are defined as unions of dense units. In addition, a data point may "contribute" to more than one cluster in different subspaces through its projections. Moreover, the identification of clusters takes place after the establishment of the appropriate subspaces. In the present framework, a different philosophy is adopted. The clusters are defined in terms of data points and each data point contributes to a single cluster. Furthermore, the clusters as well as the subspaces in which they live are simultaneously determined in an iterative fashion. Two typical representatives of this category, the PROCLUS and ORCLUS, are described next.

The PROCLUS Algorithm

This algorithm ([Agga 99]) borrows concepts from the *k*-medoids algorithmic family, described in Chapter 14. Its main idea is to generate an initial set of medoids and to iterate until an "optimum" set of medoids results. However, at each iteration a special treatment is required in order to determine the subspace where each cluster resides. The number of clusters, m, as well as the *average dimensionality*, s, of the subspaces where the clusters lie, are given as inputs to the algorithm. PROCLUS consists of three main stages, namely, the *initialization stage*, the *iterative stage*, and the *refinement stage*.

In the initialization stage a sample $X^{'}$ of size am is generated via a random selection from the entire data set (a is a constant positive integer). Then, a subset $X^{''}$ of $X^{'}$ consisting of bm points ($b < a$) is selected such that each of its points lies as far as possible from the remaining points in it. The latter set is likely to contain points from all "physical" clusters underlying X. Then, a set Θ with a corresponding index set I_Θ (for the notation see related section in Chapter 14) containing m randomly selected elements of $X^{''}$ is formed. Its elements are taken as the initial estimates of the medoids of the m clusters. The iterative stage of the algorithm is outlined in the following in algorithmic form.

- Set $cost = \infty$
- $iter = 0$
- Repeat
  - $iter = iter + 1$
  - (A) For each $i \in I_\Theta$ determine the set of dimensions D_i of the subspace where cluster C_i lives.
  - (B) For each $i \in I_\Theta$ determine the corresponding cluster C_i.
  - (C) Compute the cost $J(\Theta)$ associated with Θ.

- if $J(\Theta) < cost$ then
  - $\Theta_{best} = \Theta$
  - $cost = J(\Theta_{best})$
- End { if }
- (D) Determine the "bad" medoids of Θ_{best}.
- Set Θ equal to Θ_{best} and replace its bad medoids with randomly selected points from X''.

■ Until a termination condition is satisfied.

In the sequel we describe in more detail the steps (A) through (D) of the algorithm.

■ *(A) Determination of the cluster subspaces:* For each $\boldsymbol{x}_i, i \in I_\Theta$, the minimum among its distances from all other medoids in Θ is computed; that is, $\delta_i = \min_{r \in I_\Theta - \{i\}} d(\boldsymbol{x}_i, \boldsymbol{x}_r)$. Then, for each $\boldsymbol{x}_i$, $i \in I_\Theta$, the set L_i containing the points of X that lie in the sphere of radius δ_i, centered at $\boldsymbol{x}_i$, is determined. Along each dimension, j, of the feature space, the average distance between each $\boldsymbol{x} \in L_i$ and $\boldsymbol{x}_i$, $i \in I_\Theta$ is calculated (i.e., $d_{ij} = \sum_{\boldsymbol{x} \in L_i} |x_j - x_{ij}|/|L_i|$), where $|L_i|$ is the cardinality of L_i. In the sequel, for each $\boldsymbol{x}_i$, $i \in I_\Theta$, the average distance and standard deviation along all dimensions is computed; that is, $e_i = (\sum_{j=1}^{l} d_{ij})/l$ and $\sigma_i = \sqrt{\sum_{j=1}^{l} (d_{ij} - e_i)^2/(l-1)}$.

Then, for each medoid $\boldsymbol{x}_i$, $i \in I_\Theta$ and for each dimension of the feature space, the value $z_{ij} = (d_{ij} - e_i)/\sigma_i$ is computed. Clearly, the smaller the value of z_{ij} the more concentrated the points in L_i around $\boldsymbol{x}_i$ along the j-th dimension are. Then the D_is are determined based exclusively on the z_{ij}s values, under the following conditions: (a) each D_i has at least two dimensions and (b) the total number of dimensions contained in all D_is is equal to sm, where s is the (user-defined) average dimensionality of the cluster subspaces. Specifically, for each medoid $\boldsymbol{x}_i, i \in I_\Theta$ the two dimensions with the smallest z_{ij} values among the z_{iq}s, $q = 1, \ldots, l$, are assigned to D_i. Then, all the rest $m(l-2)$ z_{ij} values, for all medoids, are simultaneously considered, the $m(s-2)$ lowest of them are identified, and the corresponding dimensions are assigned to the appropriate D_i. If, for example, z_{34} is among the $m(s-2)$ lowest values, the fourth dimension will be assigned to D_3.

■ *(B) Determination of the clusters:* For each data point, its *Manhattan segmental distance* from each medoid $\boldsymbol{x}_i, i \in I_\Theta$ is computed; that is,

$$d_{D_i}(\boldsymbol{x}, \boldsymbol{x}_i) = \frac{\sum_{j \in D_i} |x_j - x_{ij}|}{|D_i|} \tag{15.82}$$

Note that in $d_{D_i}(\boldsymbol{x}, \boldsymbol{x}_i)$ only the coordinates in the dimensions that belong to D_i are taken into account. In addition, this distance is defined as an average

and not as a summation. Then $\boldsymbol{x}$ is assigned to the cluster whose medoid lies closer to it.

- *(C) Computation of $J(\Theta)$:* $J(\Theta)$ is defined as the average Manhattan segmental distance between the points of X and the *means* of the clusters to which they belong; that is,

$$J(\Theta) = \frac{1}{N} \sum_{i=1}^{m} \sum_{\boldsymbol{x} \in C_i} d_{D_i}(\boldsymbol{x}, \boldsymbol{m}_i) \tag{15.83}$$

  where $\boldsymbol{m}_i$ is the mean of the vectors in C_i.

- *(D) Determination of "bad" medoids:* A medoid is considered to be a "bad" one if (a) its corresponding cluster has the least number of points and (b) its corresponding cluster has less than $(N/m)q$ points, where q is a user-defined constant (typically $q = 0.1$). The rationale behind this rule is that a medoid whose corresponding cluster is of small size is likely to be an outlier or to belong to a "physical" cluster that contains at least one of the other medoids in Θ.

Finally, in the refinement stage for the set Θ_{best} that has been determined during the iteration stage the sets D_i are recomputed by applying the (A) step of the iteration stage on the C_is resulting from the iteration stage, rather than the L_is. Once the new D_is have been established, the C_is are recomputed based on the new D_is.

Like many other algorithms of this algorithmic class, PROCLUS is biased toward hyperspherically shaped clusters. In addition, the cluster subspaces must be of similar size since the average subspace dimensionality is required as input to the algorithm. Also, special care is required during the initialization stage in order to get representative points (in the set X'') from all the ("physical") clusters underlying the data set X. Otherwise, some clusters will not be recovered. In general, PROCLUS is sensitive to the input parameters it requires, which are not always easy to determine. On the other hand, PROCLUS is somewhat faster than CLIQUE on large data sets ([Pars 04]) and the required computational effort increases linearly with the dimension of the feature space, l ([Agga 99]).

The ORCLUS Algorithm

This is a point-based SCA algorithm of an agglomerative hierarchical nature ([Agga 00]). However, unlike the classical agglomerative hierarchical algorithms apart from reducing the number of clusters at each iteration (down to a user-defined value m) ORCLUS also successively reduces the dimensionality of the subspaces, where the clusters lie, down to a (user-defined) dimensionality l. (Only for this section, we denote the dimensionality of the feature space with l_0 instead of l. With l we denote the user-defined value to which the dimensionality of the cluster subspaces will gradually converge.) At each iteration of the algorithm the number of clusters as well as the dimensionality of the subspace of each cluster are reduced by

user-defined factors $a < 1$ (a typical value for a is 0.5) and $b < 1$, respectively. However, a and b must be chosen so that the reduction of the initial number of clusters m_0, down to m, and the reduction of the initial dimensionality l_0 (of the original feature space), down to l, to be achieved in the same number of iterations. Assuming that t is the total number of iterations, we have that $m = a^t m_0$ and $l = b^t l_0$, which implies that a and b are related via the condition

$$\frac{\ln(m/m_0)}{\ln(l/l_0)} = \frac{\ln a}{\ln b} \tag{15.84}$$

In addition, the subspace where each cluster lies is represented by a set of vectors, which are not necessarily parallel to the axes of the original feature space. Specifically, the set of vectors $\mathcal{E}_i$, defining the "best" q-dimensional subspace for the cluster C_i, is chosen as the subspace where the points of C_i exhibit the least spread (highest concentration). Hence, $\mathcal{E}_i$ consists of the eigenvectors of the $l_0 \times l_0$ covariance matrix Σ_i of the points in C_i that correspond to the q smallest eigenvalues of Σ_i (see also Section 6.3). The sum of these eigenvalues is the (projected) *energy* of the cluster C_i in $\mathcal{E}_i$, denoted by $E(C_i, \mathcal{E}_i)$. The ORCLUS algorithm is summarized as follows.

- Generate a set S_0 consisting of $m_0 (> m)$ points selected randomly from X.
- Set $m_c = m_0, l_c = l_0$ and $S_c = S_0$.
- Set $\mathcal{E}_i$ equal to the set of vectors defining the original feature space, for $i = 1, \ldots, m_c$.
- Set $a = 0.5$ and compute b by solving Equation (15.84).
- While $m_c > m$ do
  - For each $i, i = 1, \ldots, m_c$, define the C_i cluster as the one containing all points in X that lie closer to the i-th element of S_c. (In this case, the distance between two points is computed in the $\mathcal{E}_i$ subspace.)
  - For each $i, i = 1, \ldots, m_c$, define $\mathcal{E}_i$ as the set of eigenvectors corresponding to the l_c smallest eigenvalues of the $l_0 \times l_0$ covariance matrix of C_i.
  - Set $m_{\text{new}} = \max\{m, am_c\}$ and $l_{\text{new}} = \max\{l, bl_c\}$.
  - For each pair $(C_i, C_j), i, j = 1, \ldots, m_c, i < j$ determine $\mathcal{E}_{ij}$ for the $C_i \cup C_j$ as well as $E(C_i \cup C_j, \mathcal{E}_{ij})$.
  - While $m_c > m_{\text{new}}$ do
    - Determine $E(C_u \cup C_v, \mathcal{E}_{uv}) = \min_{i,j=1,\ldots,m_c,\ i \neq j} E(C_i \cup C_j, \mathcal{E}_{ij})$ and merge C_u and C_v to $C_r = C_u \cup C_v$.
    - Recompute the necessary $E(C_i \cup C_r, \mathcal{E}_{ir})$s in light of the previous merging.
    - $m_c = m_c - 1$

- End { While }. (Note that during this *While* loop the subspace dimension remains unchanged.)

- $m_c = m_{\text{new}}$
- $l_c = l_{\text{new}}$
- Set S_c equal to the means of the m_{new} clusters formed by the previous *While* loop.
- End { While }

As PROCLUS, ORCLUS is biased toward hyperspherical clusters, due to the fact that the mean of a cluster is used as its representative. In addition, the required computational time is $O(m_0^3 + m_0 N l_0 + m_0^2 l_0^3)$. That is, it increases linearly with N and cubically with l_0. It is worth noting that although increasing the value of m_0 increases the computational time this may improve the quality of the final clustering. Furthermore, note that the subspaces of all clusters are restricted to the same dimensionality. Criteria for choosing a proper value for l as well as extensions of the algorithm that are able to handle outliers are discussed in [Agga 00]. Finally, random sampling techniques may be used to reduce the computational time. Other point-based subspace clustering algorithms are discussed in [Frie 02, Woo 02], and [Yang 02].

Remarks

- In GBSCAs, the clusters are represented as unions of dense units (which is a rather "rough" description), whereas in PBSCAs the clusters are represented in terms of data points (exact description). Moreover, in GBSCAs each point may contribute to more than one cluster in different subspaces through its projections, whereas in the PBSCAs each point contributes to a single cluster.
- In GBSCAs the identification of clusters is carried out only after the appropriate subspaces have been determined. In contrast, in PBSCAs the clusters as well as the subspaces where they lie are simultaneously determined in an iterative fashion.
- The GBSCAs are able, in principle, to unravel arbitrarily shaped clusters, whereas several PBSCAs are biased toward hyperellepsoidal clusters.
- The computational time required by most of the GBSCAs and PBSCAs scales linearly with N, the number of points. (For PROCLUS this has been established experimentally [Agga 99].)
- The computational time required by the described GBSCAs increases exponentially with the dimensionality of the feature space l, whereas in PBSCAs it exhibits a polynomial dependence.
- In the GBSCAs there are no restrictions concerning the dimensionality of the subspaces, wheras the PBSCAs pose constraints on it.

- In GBSCAs there exists a large overlap in the resulting clusters. On the contrary, most of the PBSCAs produce disjoint clusters.
- Both GBSCAs and PBSCAs are sensitive to the choice of the involved user-defined parameters.

15.11 OTHER CLUSTERING ALGORITHMS

A clustering algorithm that is based on the so-called *tabu search method* is presented in [Al-S 95]. Its initial state is an arbitrarily chosen clustering. The algorithm proceeds as follows. Based on the current state of the algorithm, a set of candidate clusterings, A, is created. The next state is chosen to be the "best" element of A, according to some criterion function. Certain criteria are used to prevent the algorithm from returning to recently visited states. The procedure is repeated for a prespecified number of iterations. Preliminary results reported in [Al-S 95] show that this algorithm compares favorably with the hard clustering and the simulated annealing algorithms. A recent tabu search based heuristic scheme for clustering is presented in [Sung 00].

A method that directly relates clusters to peak values of the pdf has been suggested in [Tou 74], where the estimation of the pdf is achieved via Parzen windows. A related method is the *mountain method* (see, e.g., [Dave 97]). The idea is to assign to each vector, $\boldsymbol{x}$, an energy source. The generated potential has a peak at $\boldsymbol{x}$ and rapidly decays as we move away from it. The total potential function at a specific point in the vector space is the summation of the potentials produced by all the vectors of X. We compute the value of this function at each data point and form the array $\boldsymbol{v}$ of the N resulting values. The maximum one, which corresponds to the highest peak, is identified and the corresponding vector is considered as the representative of the first cluster. Then we remove the largest value of $\boldsymbol{v}$ and update the rest of the components appropriately. This procedure is repeated until a specific termination criterion is met.

An algorithm that combines ideas from both the fuzzy clustering schemes and the agglomerative algorithms (Chapter 13) is discussed in [Frig 97]. This scheme produces clusterings that minimize the following cost function

$$J(\boldsymbol{\theta}, U) = \sum_{i=1}^{N}\sum_{j=1}^{m} u_{ij}^2 d(\boldsymbol{x}_i, C_j) - a\sum_{j=1}^{m}\left[\sum_{i=1}^{N} u_{ij}^2\right] \tag{15.85}$$

where now m varies. Clearly, the first term in the above equation is minimized when $m = N$, and the second term is maximized when $m = 1$. It is worth mentioning that the hierarchies of clusterings produced do not necessarily possess the nested property.

Besides the preceding algorithms, many other clustering algorithms based on very different ideas have been proposed. For example, in [Matt 91] a scheme that does not use the concept of the distance between vectors is proposed. Also, in

[Kodr 88] a clustering technique based on a conceptual distance is presented. Clustering approaches that borrow concepts from gravity theory are discussed in [Oyan 01, Chen 05]. Clustering algorithms, suitable for discrete-valued feature vectors, that construct classification trees are discussed in [Fish 87, Bisw 98]. A graph theory-based algorithm that uses a probabilistic framework is discussed in [Ben 99].

Another technique combining supervised and unsupervised methods has been proposed in [Pedr 97]. The latter may be useful in applications in which only a fraction of the data have a class label for the training.

In [Robe 00], a different clustering method is discussed. The clustering problem is stated in information theoretic terms and it is shown that minimization of the entropy can be used in order to estimate the clustering structure underlying the data set X. This is equivalent to obtaining the structure associated with maximum certainty. Another algorithm based on information theoretic criteria has been suggested in [Goks 02]. It is a valley-seeking algorithm and builds upon Renyi's entropy estimator.

Another interesting clustering algorithm that utilizes the wavelet transform (see Section 6.13) is the so-called *WaveCluster* algorithm ([Shei 98]). The method applies, first, an l-dimensional grid on the feature space by dividing each dimension in r intervals and it determines the data points contained in each unit (box), M_i, of the grid. In the sequel, an l-dimensional signal is generated by representing each unit by the number of points it contains. Then, it applies the l-dimensional wavelet transform on the units (points) M_i of the grid (the multidimensional wavelet transform is actually a generalization of the two-dimensional wavelet transform discussed in Section 6.14) and produces a new set of units T_j in the transformed space at various resolutions. Then, at each resolution it determines the clusters in the transformed space as connected components of T_j units. In the sequel, based on the correspondence between M_is and T_js the algorithm assigns the points contained in each M_i to the appropriate cluster. Among the advantages of WaveCluster is the efficient handling of outliers, its insensitivity to the order of the presentation of the points to the algorithm, its ability to determine arbitrarily shaped clusters, and the fact that it does not require as input the exact number of the clusters. Its computational complexity is $O(N + r^l)$. Thus, the algorithm is best suited for large data sets of relatively low dimensionality.

In recent years there has been an increasing interest in clustering sequential data. DNA sequencing and web data mining are examples of two typical applications of this type. Techniques for clustering this type of data are based on tools discussed in Chapters 8 and 9, with the Edit distance and the HMM being among the most popular. A major problem associated with these applications is the very long length of the sequences to be matched, which renders most of the classical algorithms computationally infeasible. To overcome such difficulties, a number of algorithmic schemes have been suggested adopting various heuristics. Some well-known schemes include BLAST [Alts 97] and FASTA [Pear 88].

Reviews of such clustering methods and applications are discussed in, for example, [Durb 98, Gusf 97, Beng 99, Mill 01, Liew 05].

Constraint clustering

In many cases, the use of labeled data is critical for the success of the clustering process. A subset of labeled data also makes the evaluation of the resulting clustering more accurate. Consequently, several clustering approches have been introduced that use both labeled and unlabeled data, as it was the case with the semi-supervised learning treated in chapter 10.

Most of these approaches rely on getting input from the user in terms of constraints on feasible clusterings. The simplest constraints are placed on pairs of patterns and either force both patterns to be in the same cluster (*Must-Link constraint*), or force them to be in different clusters (*Cannot-Link constraint*). Initial efforts concentrated on modifying existing algorithms, such as the *k*-means or hierarchical algorithms, to operate with constraints ([Wags 01, Davi 05]). More recent approaches use the constraints imposed by the user to learn distance measures that conform to the user expectation, as this is expressed by the given constraints ([Basu 04, Xing 02, Kuli 05, Halk 08]).

In [Bene 00, Bane 02] two modifications of the *k*-means algorithm are proposed that deal with the case where the number of points in each cluster is bounded below. Also, an approach that builds balanced clusters is discussed in [Stre 00]. In [Tung 01] the problem of clustering two-dimensional data in the presence of *obstacles* in the feature space is considered. In this case, the distance between two points is defined as the shortest path from one point to the other, taking into account the obstacles in the feature space.

15.12 COMBINATION OF CLUSTERINGS

Throughout the second part of the book, we discussed algorithms that produce a single clustering (or a hierarchy of clusterings) for a given data set $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, $\boldsymbol{x}_i \in \mathcal{R}^l$, $i = 1, 2, \ldots, N$. In this section, the situation is different. A (final) clustering is obtained based on a set of n different clusterings of X. Specifically, the aim here is two-fold: (a) the production of an appropriate set of clusterings, $\mathcal{E}$, for the data set, X, called *ensemble* of clusterings, and (b) the combination of the clusterings of $\mathcal{E}$ to produce a final clustering, called *consensus clustering*. The main motivation for considering such techniques is that it is expected that the resulting consensus clustering, based on $\mathcal{E}$, will model the data better that any single clustering.

Various techniques have been proposed in the above spirit but none of them seems to clearly outperform the rest ([Topc 05]). In the sequel, after giving some necessary definitions, we consider separately the two previously stated goals and we focus on the most representative methods for each case.

Let $\mathcal{E} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n\}$ be a set of clusterings of the data set X, where $\mathcal{R}_i = \{C_i^1, C_i^2, \ldots, C_i^{m_i}\}$, with m_i being the number of clusters in the $\mathcal{R}_i$ clustering. The superscript j in C_i^j denotes the *label* of the corresponding cluster in the $\mathcal{R}_i$ clustering. Alternatively, each clustering, $\mathcal{R}_i$, can be represented by an N dimensional row vector, $\boldsymbol{y}_i$, called *label vector*, whose k-th element $y_i(k)$ contains the cluster label of the k-th data point. If for example $\mathcal{R}_i = \{C_i^1, C_i^2, C_i^3\} = \{\{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_6, \boldsymbol{x}_{10}\}, \{\boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_7\}, \{\boldsymbol{x}_5, \boldsymbol{x}_8, \boldsymbol{x}_9\}\}$, then $\boldsymbol{y}_i = [1, 1, 2, 2, 3, 1, 2, 3, 3, 1]$. Clearly, both $\mathcal{R}_i$ and $\boldsymbol{y}_i$ are equivalent representations of the same clustering.

A. Generation of an ensemble of clusterings

Generation of each clustering $\mathcal{R}_i$ of $\mathcal{E}$ involves the following two steps: (a) the choice of a subspace to project the data points in X and (b) the application of a clustering algorithm on this subspace. Note that, in general, the $\mathcal{R}_i$s are not constrained to have the same number of clusters. Unless otherwise stated, this assumption has been adopted here.

In analogy with the schemes for combining classifiers (Section 4.21), it is desirable for $\mathcal{E}$ to contain clusterings that are as "independent" as possible. To this end, the following general directions are followed:

- *All data, all features.* In this case, all l features and all N data points are used. The n different clusterings, $\mathcal{R}_i$, $i = 1, 2, \ldots, n$, result by employing either different clustering algorithms or the same clustering algorithm with different parameters (e.g., in the case of the k-means algorithm, different initial conditions or different distance measures can be used) (see, e.g., [Fred 05]). This method of generating the $\mathcal{R}_i$s is also called *robust centralized clustering* ([Stre 02]).
- *All data, some features.* In this case, all the data points of X are considered. A number of n sets, $X_i, i = 1, \ldots, n$, are formed from X, where in each one of them the data points are represented (a) either by selecting a subset of features or (b) by projecting onto a randomly chosen lower dimensional space, see for example, [Fern 03, Topc 05] (see Section 15.10.1). Clearly, the cardinality of each X_i is N. In the sequel, eihter the same algorithm (also called *base algorithm*) with the same parameters (e.g., [Fern 04]) or different algorithms (e.g., [Stre 02]) are applied on the X_is in order to produce the respective clusterings $\mathcal{R}_i$s. This method of generating the $\mathcal{R}_i$s is also called *feature-distributed clustering* ([Stre 02]).
- *Some data, all features.* In this case, techniques like bootstrapping or sampling are applied on X, in order to produce data sets $X_i, i = 1, \ldots, n$, on which (usually) the same clustering algorithm is applied to produce the respective n clusterings of $\mathcal{E}$. The points of X, which have not been selected to participate in X_i, may be assigned to their nearest cluster in $\mathcal{R}_i$. When X is a high dimensional data set, its points may first be projected to a lower dimensional

space (using, e.g., PCA), forming a lower dimensional data set X'. Then the method that generates the X_is is applied on the reduced dimensionality data set X' (e.g., [Fern 04]).

B. Combination of clusterings

Having generated the clustering ensemble, $\mathcal{E} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n\}$, the next step is to combine the $\mathcal{R}_i$s in order to produce the final (consensus) clustering $\mathcal{F} = \{F_1, \ldots, F_m\}$.

Among the various combination techniques that have been proposed, several of them make use of the so-called *co-association matrix* $\mathcal{C}$. This is an $N \times N$ dimensional matrix, whose (i,j) element $c(i,j)$ equals to n_{ij}/n, where n_{ij} is the number of times the ith and the jth points of X have been assigned to the same cluster, among the n clusterings of $\mathcal{E}$. Note that $c(i,j) \in [0,\ 1]$. Clearly, large values of $c(i,j)$ imply that the ith and jth points of X are likely to be similar to each other.

In the sequel, a brief description is given for the most popular methods for combining clusterings.

- *Methods based on the co-association matrix.* The co-association matrix $\mathcal{C}$ is computed and then, using $\mathcal{C}$ as a similarity matrix, the single link algorithm is applied. From the resulting dendrogram, the clustering with the larger lifetime (see Section 13.6) is selected as the final clustering ([Fred 05]). Other hierarchical clustering algorithms, such as the complete link and the average link, can also be applied on $\mathcal{C}$ (e.g., [Topc 05]).

  In general, these methods require a large number of clusterings n, in order to estimate more reliably the elements of $\mathcal{C}$.

- *Graph-based methods.* In the sequel, we discuss three different formulations in this framework.
  - *Instance-based graph formulation (IBGF).* A fully connected (complete) graph $G(V,E)$ is constructed, where *each vertex of V corresponds to a data point* and each edge e_{ij} is weighted by $c(i,j)$ (the (i,j) element of the co-association matrix). Then the graph is partitioned into m disjoint subsets of vertices $V_1, \ldots, V_m$ such that: (i) the sum of the weights of the edges that connect vertices from different subsets is minimized and (ii) the V_is are approximately of the same size (note, however, that (ii) is not always required). To this end, different optimization criteria are used, such as the *normalized cut criterion* ([Shi 00]) and the *ratio-cut criterion* ([Hage 98]) (see also Section 15.2.4 on spectral clustering).
  - *Cluster-based graph formulation (CBGF).* Let

$$\mathcal{E}' = \{C_1^1, \ldots, C_1^{m_1}, C_2^1, \ldots, C_2^{m_2}, \ldots, C_n^1, \ldots, C_n^{m_n}\}$$

    be the set of all the clusters contained in the n clusterings of $\mathcal{E}$ and let $t = \sum_{i=1}^{n} m_i$ be its cardinality. In this case, a graph $G = (V,E)$ is constructed where *each vertex of V corresponds to a cluster in $\mathcal{E}'$*. Also, the weight w_{ij}

of the edge connecting the vertices associated with the clusters $C_i, C_j \in \mathcal{E}'$ is defined as the *Jaccard measure*, which is the ratio of the number of the data points in $C_i \cap C_j$ to the number of the data points in $C_i \cup C_j$, that is, $w_{ij} = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$. Then, an m-partition $\mathcal{P} = \{P_1, \ldots, P_m\}$ of the graph is obtained, under the constraints (i) and (ii) used in the previous case (Note that each P_i corresponds to a set of clusters). Ultimately, the final clustering $\mathcal{F} = \{F_1, \ldots, F_m\}$ is obtained as follows: For each data point $\boldsymbol{x} \in X$ we count the number of its occurrences in the clusters contained in $P_i, i = 1, \ldots, m$. Then $\boldsymbol{x}$ is assigned to F_i, if it is more frequently met in P_i ([Fern 04]).

- *Hybrid bipartite graph formulation (HBGF).* Let $\mathcal{E}'$ and t be defined as in CBFG. A graph $G = (V, E)$ is constructed, however, in this case, *data points as well as clusters in $\mathcal{E}'$ are represented by vertices.* Thus, V contains a total of $N + t$ vertices. The weight w_{ij} between the v_i and v_j vertices of the graph equals to 0 if either both vertices correspond to data points or both correspond to clusters. Otherwise, if v_j corresponds to a cluster and v_i to a point *and in addition* the point belongs to this cluster, then $w_{ij} = w_{ji} = 1$. In all other cases we set $w_{ij} = 0$. A graph clustering approach (e.g., spectral clustering) is then applied to the previous graph, leading to the consensus (final) clustering of the data in X.

  Consider, for example, the following case: $\mathcal{E} = \{\mathcal{R}_1, \mathcal{R}_2\}$, where $\mathcal{R}_1 = \{C_1^1, C_1^2\} = \{\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_6\}, \{\boldsymbol{x}_7, \boldsymbol{x}_8, \boldsymbol{x}_9\}\}$ and $\mathcal{R}_2 = \{C_2^1, C_2^2\} = \{\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_5\}, \{\boldsymbol{x}_6, \ldots, \boldsymbol{x}_9\}\}$. The corresponding graph, for this case, is shown in Figure 15.28. The line shows a bi-partition of the graph which implies that the final clustering is $\{\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_5\}, \{\boldsymbol{x}_6, \ldots, \boldsymbol{x}_9\}\}$.

In general, IBGF exhibits higher computational complexity compared to CBGF and HBGF. This happens because in IBGF a fully connected graph is generated and the graph partitioning problem is of size $O(N^2)$. On the other hand, in CBGF the size of the problem is $O(t^2)$, where (usually) $t \ll N$, while in HBGF the size of the problem is $O(nN)$.

Experimental results ([Fern 04]) suggest that HBGF achieves comparable or better performance than IBGF, CBGF. Also, HBGF consistently improves over the average performance of the members of the clustering ensemble $\mathcal{E}$ (this is measured in terms of data sets, where the true clusters are known). In addition, all methods perform, in general, better as n increases.

Another graph-based method is discussed in [Ayad 03]. A graph $G = (V, E)$ is constructed where each vertex of V corresponds to a data point of X and the co-association matrix is used to define the *nearest neighbor vertices* of a given vertex. In the sequel (a) each edge between two vertices v_i and v_j is weighted accordingly, depending on the number of their common nearest neighbors, n_{ij}, provided that n_{ij} exceeds a certain threshold (otherwise, the edge weight is set to 0) and (b) each vertex is weighted according to the number of its nearest neighbor vertices that shares with other vertices. Having

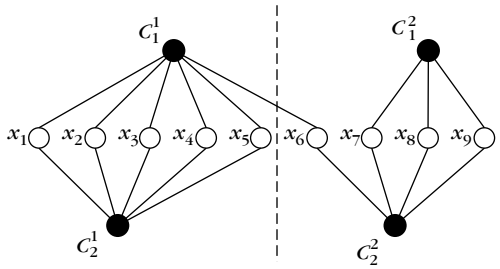

FIGURE 15.28

The graph constructed by HBGF for an ensemble of two clusterings, each one containing two clusters. Only edges associated with nonzero weights are shown. The dashed line cuts the graph and it defines the ensemble clustering (see text for more explanation).

defined G, a graph partition algorithm is employed to determine the final clustering of X, subject to some constraints imposed on the weights of the vertices.

- *Cost function optimization methods.* In this framework, the ensemble clustering $\mathcal{F} = \{F_1, \ldots, F_m\}$ is obtained via cost optimization techniques. In the sequel, three such methods are described.
  - *Utility function optimization.* According to this method, $\mathcal{F}$, also called *median clustering*, is defined as the clustering that "summarizes" best the clusterings of $\mathcal{E}$, according to the *utility function criterion* ([Fish 87, Topc 05]). The latter measures the quality of a candidate median clustering $\mathcal{F}$ against some other clustering $\mathcal{R}_i$ and it is defined as

$$U(\mathcal{F}, \mathcal{R}_i) = \sum_{r=1}^{m} P(F_r) \sum_{j=1}^{m_i} P(C_i^j | F_r)^2 - \sum_{j=1}^{m_i} P(C_i^j)^2 \tag{15.86}$$

    where $P(F_r) = |F_r|/N, P(C_i^j) = |C_i^j|/N$ and $P(C_i^j | F_r) = |C_i^j \cap F_r|/|F_r|$, with $|A|$ denoting the cardinality of the set A. $U(\mathcal{F}, \mathcal{R}_i)$ measures the agreement between the two clusterings $\mathcal{R}_i$ and $\mathcal{F}$. It turns out that $U(\mathcal{F}, \mathcal{R}_i)$ achieves its maximum value when $\mathcal{F}$ and $\mathcal{R}_i$ are identical (in this case, the probabilities $P(C_i^j | F_r)$ are either one or zero), while as $\mathcal{F}$ and $\mathcal{R}_i$ deviate from each other $U(\mathcal{F}, \mathcal{R}_i)$ decreases.

    The *overall utility* of $\mathcal{F}$ on $\mathcal{E}$ is defined as

$$U(\mathcal{F}, \mathcal{E}) = \sum_{i=1}^{n} U(\mathcal{F}, \mathcal{R}_i) \tag{15.87}$$

    The best summary of the clusterings of $\mathcal{E}$ (the median clustering) follows from the maximization of the cost given in (15.87). In [Mirk 01], it is shown

that maximizing the overall utility is equivalent to minimizing the square error clustering criterion (Eq. (14.85) associated with the *k*-means algorithm) over the data points (assuming that the number of clusters m in $\mathcal{F}$ is fixed). However, now, each data point is represented by a vector whose coordinates are the respective cluster labels in the n clusterings of $\mathcal{E}$ ([Mirk 01, Topc 05]). Hence by applying, for example, the *k*-means algorithm on the transformed data set a solution for the maximization problem of $U(\mathcal{F}, \mathcal{E})$ is obtained.

- *Normalized mutual information (NMI) criterion.* This criterion has been inspired from the information theory, where the statistical information shared between two distributions is assessed by the *mutual information measure*. In the context of clustering, NMI is a measure analogous to the mutual information between two distributions and it is defined as

$$NMI(\mathcal{R}_1, \mathcal{R}_2) = \frac{2}{N} \sum_{q=1}^{m_1} \sum_{r=1}^{m_2} n_q^r \log_{m_1 m_2} \left(\frac{n_q^r N}{n_q n_r} \right) \tag{15.88}$$

where m_1, m_2 are the number of clusters in $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively, $n_q^r = |C_1^q \cap C_2^r|, n_q = |C_1^q|, n_r = |C_2^r|$, where $|A|$ denotes the cardinality of a set A. Let us define the *average normalized mutual information criterion (ANMI)* between a clustering $\mathcal{F}$ and an ensemble of clusterings $\mathcal{E}$ as follows

$$ANMI(\mathcal{F}, \mathcal{E}) = \frac{1}{n} \sum_{i=1}^{n} NMI(\mathcal{F}, \mathcal{R}_i) \tag{15.89}$$

Then the ensemble clustering $\mathcal{F}$ is obtained as the one that maximizes the previous criterion ([Stre 02]).

It is worth noting that NMI may also be used to assess the performance of a clustering algorithm when it is performed on a data set where the labels of the data points are known (e.g. [Fred 05]).

- *Mixture model formulation.* Such a formulation is discussed in [Topc 05]. In this framework, each data point $\boldsymbol{x}_i \in X$ is represented by a new n-dimensional vector, $\boldsymbol{x}_i'$, whose jth component equals to the cluster label of $\boldsymbol{x}_i$ in the jth clustering. Let $X' = \{\boldsymbol{x}_1', \ldots, \boldsymbol{x}_N'\}$. Recall that $\boldsymbol{y}_i$ is the label vector for clustering $\mathcal{R}_i$; the relation between the $\boldsymbol{x}_i$s, $\boldsymbol{y}_i$s and $\boldsymbol{x}_i'$s is schematically given below:

$$\begin{array}{ccccccccc}
 & & & \boldsymbol{y}_1 & \cdots & \boldsymbol{y}_n & & \\
\hline
\boldsymbol{x}_1 & \rightarrow & [& y_1(1) & \ldots & y_n(1) &] & \equiv \boldsymbol{x}_1' \\
\boldsymbol{x}_2 & \rightarrow & [& y_1(2) & \ldots & y_n(2) &] & \equiv \boldsymbol{x}_2' \\
\vdots & \rightarrow & & & \vdots & & & \\
\boldsymbol{x}_N & \rightarrow & [& y_1(N) & \ldots & y_n(N) &] & \equiv \boldsymbol{x}_N'
\end{array}$$

The goal, now, is to determine the (unknown) cluster labels of the data in the final clustering $\mathcal{F} = \{F_1, \ldots, F_m\}$. To this end, a finite mixture model of probability functions is defined

$$P(\boldsymbol{x}^{'}; \boldsymbol{\Theta}) = \sum_{q=1}^{m} P_q P(\boldsymbol{x}^{'}|F_q; \boldsymbol{\theta}_q) \tag{15.90}$$

where m is the number of clusters in the consensus clustering $\mathcal{F}$, P_q is the a priori probability of the qth cluster in $\mathcal{F}$ and $P(\boldsymbol{x}^{'}|F_q; \boldsymbol{\theta}_q)$ is the probability function describing the cluster F_q, which is parametrized by the vector $\boldsymbol{\theta}_q$. Finally, $\boldsymbol{\Theta} = \{P_1, \ldots, P_m, \boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_m\}$ is the set of all the parameters of the model.

Assuming statistical independence among the components of $\boldsymbol{x}^{'}$, it follows that

$$P(\boldsymbol{x}^{'}|F_q; \boldsymbol{\theta}_q) = \Pi_{j=1}^{n} P^j(x_j^{'}|F_q; \boldsymbol{\theta}_q^j) \tag{15.91}$$

Since the values of $x_j^{'}$ are nominal (integers) it seems natural to model the respective probabilities via a multinomial distribution, that is,

$$P^j(x_j^{'}|F_q; \boldsymbol{\theta}_q^j) = \Pi_{r=1}^{m_j} \theta_{jq}(r)^{\delta(x_j^{'}, r)} \tag{15.92}$$

where $\delta(a, b) = 1$, if $a = b$ and 0 otherwise. Keeping in mind the definition of $\boldsymbol{x}^{'}$, $\theta_{jq}(r)$ models the probability that the data point $\boldsymbol{x}$ belongs to the rth cluster in the jth clustering $\mathcal{R}_j$ of $\mathcal{E}$, given that $\boldsymbol{x}$ belongs to cluster F_q of $\mathcal{F}$.

In the sequel, the EM algorithm can be employed to derive the estimates for the parameters of $\boldsymbol{\Theta}$ (see [Topc 05]). Experimental results ([Topc 05]) show that the EM algorithm converges fast and its performance is slightly better than that of graph-based methods for small n. As n increases, the performance of EM is expected to degrade, due to the increase of the parameters that have to be estimated. An alternative EM formulation of the problem is discussed in [Lang 05].

An interesting variation of the clustering combination problem is discussed in [Topc 04]. In this case, the sets X_i, $i = 1, \ldots, n$, used for the generation of the ensemble $\mathcal{E}$, are generated *sequentially* via sampling of X, where now *the sampling probabilities for the data points differ from each other*. Specifically, for each X_i (a) the clustering $\mathcal{R}_i$ is produced (b) the sampling probabilities for the data points in X are re-estimated according to a rule and (c) the next set X_{i+1} performs sampling on X based on the updated probabilities. The essence of this method is to assign higher sampling probabilities to points of X, which lie in regions where overlap between clusters is encountered. In this way, the focus of the X_is to be formed at the later stages will be on those overlapping regions. Experimental results reported in [Topc 04] suggest that this technique improves (even slightly) the results

of other methods, where the X_ts are drawn independently from each other. Other techniques are discussed in, for example, [Law 04, Qian 00, Fred 05] where the case of combining different objective functions is also considered.

Remarks

- In comparing two clusterings usually one faces the problem of determining the optimal correspondence between the two clusterings. For example the clusterings with cluster vectors $\boldsymbol{y}_1 = [1,\ 1,\ 1,\ 1,\ 2,\ 2,\ 2]$ and $\boldsymbol{y}_2 = [2,\ 2,\ 2,\ 2,\ 1,\ 1,\ 1]$ define exactly the same clustering on the data set, however, the same cluster in the above clusterings is denoted with different labels. This problem is usually faced using the so called *Hungarian method* (see, e.g., [Papa 82]).
- The problem of combining various clusterings is in close affinity with the so-called *distributed clustering*, which has attracted significant attention due to the increasing size of the current databases ([Kots 04]). In distributed clustering, the objects to be clustered and/or their features reside in different (local) sites in one of the following ways: (a) each site has access to a specific subset of features from *all* objects (*feature distributed clustering*), (b) each site stores *all* the available features for *some* objects (*object distributed clustering*), (c) each site stores *some* features for *some* objects (*feature/object distributed clustering*). Instead of transmitting all of them to a single site and perform a standard clustering algorithm, the data are clustered independently on the different local sites and then the clustering results are moved to a single site, where they contribute to the generation of a final clustering.

15.13 PROBLEMS

15.1 Consider the set $X = \{\boldsymbol{x}_i, i = 1, \ldots, 7\}$, where $\boldsymbol{x}_1 = [1, 1]^T$, $\boldsymbol{x}_2 = [1,\ 2]^T$, $\boldsymbol{x}_3 = [2, 1]^T$, $\boldsymbol{x}_4 = [3,\ 1]^T$, $\boldsymbol{x}_5 = [6, 1]^T$, $\boldsymbol{x}_6 = [7,\ 1]^T$, $\boldsymbol{x}_7 = [6, 2]^T$.

a. Determine the value of q (Section 15.2.1) for which the MST clustering algorithm gives two clusters.

b. Apply the algorithms that are based on the idea of regions of influence when these regions are defined by Eqs. (15.2)–(15.5).

c. Run the directed tree-based clustering algorithm and determine the values of θ for which it gives two clusters.

15.2 Consider the basic competitive learning algorithm with $\eta = 0.2$. Let $X = \{x_i, i = 1, \ldots, 4\}$, where $x_1 = -3, x_2 = -2, x_3 = 2, x_4 = 3$. Also, let $m = 2$ and $w_1 = -1$ and $w_2 = 1$. Assume that the vectors of X are presented to the algorithm in the same order, $x_1, x_2, x_3 x_4, x_1, x_2, x_3, x_4, \ldots$. Let us call the time required for the consideration of the feature vectors of X once the *updating circle*.

a. Show that w_1 (w_2) always wins on the first (last) two feature vectors when the squared Euclidean distance is in use.

b. Will w_1 and w_2 converge to the values -2.5 and 2.5, respectively, after an infinite number of updating circles? Give intuitive arguments.

15.3 What would the behavior of the leaky learning algorithm be if $\eta_w = \eta_l$?

15.4 *von der Malsburg learning rule.* Assume that the data set X consists of N binary-valued feature vectors, and for each of the m available representatives, $\boldsymbol{w}_j$, we have $\sum_{k=1}^{l} w_{jk} = 1$, where w_{jk} is the kth coordinate of $\boldsymbol{w}_j$. The rule may be stated as follows:

- Present an input vector $\boldsymbol{x} \in X$.
- Determine the winner, $\boldsymbol{w}_j$, of the competition for $\boldsymbol{x}$.
- Update the representatives

$$w_{jk}^{\text{new}} = \begin{cases} w_{jk} + \eta\left(\frac{x_k}{n_x} - w_{jk}\right), & \text{if } \boldsymbol{w}_j \text{ wins on } \boldsymbol{x} \\ w_{jk}, & \text{if } \boldsymbol{w}_j \text{ loses on } \boldsymbol{x} \end{cases} \tag{15.93}$$

where x_k is the kth coordinate of $\boldsymbol{x}$. In the last equation $n_x = \sum_{k=1}^{l} x_k$, that is, it is equal to the number of 1's contained in $\boldsymbol{x}$, and η, the learning rate, takes values in $[0, 1]$. The updating rule may be stated in words as follows: "If a representative wins, each of its coordinates gives up some proportion η that in the sequel is equally distributed among the coordinates w_{ji} that correspond to $x_k = 1$. All the remaining representatives do not change."

a. Verify that this statement is equivalent to the updating rule given by Eq. (15.93).

b. Prove that $\sum_{k=1}^{l} w_{jk}^{\text{new}} = 1, j = 1, \ldots, m$.

15.5 Prove Eq. (15.43).

15.6 Consider three 2-dimensional Gaussian probability density functions with means $\boldsymbol{\mu}_1 = [-1, -1]^T$, $\boldsymbol{\mu}_2 = [6, 3]^T$, $\boldsymbol{\mu}_3 = [-0.7, 7]^T$ and covariance matrices $\Sigma_1 = \Sigma_2 = \Sigma_3 = 2I$, respectively, where I is the 2×2 identity matrix. Draw 200 points from each distribution and form a data set X with the resulting 600 points.

a. Run the Binary Morphology Clustering Algorithm (BMCA) when T is the 3×3 square structuring element given in Example 15.5. For each case use different values of r and proceed to the third stage of the algorithm with the best one of them.

b. Run the Generalized Hard Clustering Scheme (GHAS) algorithm from Chapter 14, using the squared Euclidean distance as the dissimilarity function between two vectors, for the optimum number of clusters derived by the previous procedure. Compare the results of the two algorithms.

15.7 For the data set X given in Example 15.8, run the isodata algorithm assuming that the number of clusters is 2. Compare the results obtained in Example 15.8 and those obtained with isodata. Give a qualitative explanation of the differences that may be observed.

15.8 Verify Eqs. (15.36) and (15.37).

15.9 Derive the updating equation for the coordinates of the parameter vector $\boldsymbol{\theta}$ when $g(\boldsymbol{x}; \boldsymbol{\theta})$, defined in Section 15.6, is a quadratic function of $\boldsymbol{\theta}$.
Hint: In this case $g(\boldsymbol{x}; \boldsymbol{\theta}) = w_0 + \sum_{i=1}^{l} w_i x_i + \sum_{s=1}^{l} \sum_{r=1}^{l} w_{sr} x_s x_r$.

15.10 Consider two 2-dimensional Gaussian distributions with means $\boldsymbol{\mu}_1 = [0, 0]^T$ and $\boldsymbol{\mu}_2 = [3, 3]^T$ and covariance matrices $\Sigma_1 = I$ and $\Sigma_2 = 1.5I$, respectively, where I is the 2×2 identity matrix. Create a data set X such that 100 feature vectors stem from the first and another 100 feature vectors stem from the second Gaussian distribution.

a. Run the Boundary Detection Algorithm (BDA) algorithm on X assuming that the decision boundary is a hyperplane. Use the hyperbolic tangent as f.

b. Run the BDA algorithm on the X^+ and X^- stemming from the previous running. Comment on the results.

15.11 **a.** What is the shape of $V(\boldsymbol{x})$, defined by Eq. (15.38) when $d(\boldsymbol{x}, \boldsymbol{y})$ is given as in Eq. (15.39)?

b. How should $d(\boldsymbol{x}, \boldsymbol{y})$ be defined in order to have a hypercubical shape for $V(\boldsymbol{x})$?

c. Does the shape of $V(\boldsymbol{x})$ affects the behavior of the valley-seeking clustering algorithm? Give an example.

15.12 Consider the set $X = \{\boldsymbol{x}_i, i = 1, \ldots, 10\}$, where $\boldsymbol{x}_1 = [0, 1]^T$, $\boldsymbol{x}_2 = [0, 2]^T$, $\boldsymbol{x}_3 = [0, 3]^T$, $\boldsymbol{x}_4 = [0, 4]^T$, $\boldsymbol{x}_5 = [1, 1]^T$, $\boldsymbol{x}_6 = [1, 2]^T$, $\boldsymbol{x}_7 = [1, 3]^T$, $\boldsymbol{x}_8 = [2, 1]^T$, $\boldsymbol{x}_9 = [2, 2]^T$, $\boldsymbol{x}_{10} = [2, 3]^T$. Initially, the first six of them belong to cluster C_1 and the next four belong to cluster C_2. Apply the valley-seeking algorithm to X and comment on the results.

15.13 If $T_{\max} = 5$ and $T_{\min} = 0.5$, estimate the number of sweeps required with the simulated annealing algorithm in order to determine (in probability) the clustering with the globally minimum value of J.
Hint: Use Eq. (15.44).

15.14 Modify the deterministic annealing algorithm so that the number of representatives is not fixed *a priori* but increases as β increases.

15.15 Consider the function

$$J = \sum_{i=1}^{N} d(\boldsymbol{x}_i, C_{\boldsymbol{x}_i}) \tag{15.94}$$

where $C_{\boldsymbol{x}_i}$ is the cluster to which $\boldsymbol{x}_i$ belongs and $d(\boldsymbol{x}_i, C_{\boldsymbol{x}_i})$ is a distance between a point and a set using no representative for the set (e.g., Chapter 11). Propose a coding of the solutions for a genetic algorithm that uses this function. Discuss the merits and the disadvantages of the proposed coding.

MATLAB PROGRAMS AND EXERCISES

Computer Programs

15.1 *Competitive learning.* Write a MATLAB function named *leaky_learn* that implements the leaky learning algorithm. This function takes as inputs: (a) an $l \times N$ dimensional matrix X each column of which is a data vector, (b) an $l \times m$ dimensional matrix w whose columns contain initial estimates of the m representatives, (c) the learning rate for the winner unit, gw, (d) the learning rate for the rest of the units, gl, (e) the maximum number of iterations, (f) the parameter e, which is used in the termination condition of the algorithm ($||\boldsymbol{w}(t) - \boldsymbol{w}(t-1)|| < e$). The output consists of: (a) the vector w whose columns are the final estimates of the representatives and (b) an N dimensional row vector *bel*, the i-th element of which contains the cluster to which the i-th vector belongs. The Euclidean distance is used to measure the distance between two vectors and the vectors are presented always in the same order to the algorithm, until convergence.

Solution

```
function [w,bel]=leaky_learn(X,w,gw,gl,maxiter,e)
  [l,N]=size(X);
  [l,m]=size(w);
  diff=e+1;
  iter=0;
  while(diff>e)&(iter<=maxiter)
    iter=iter+1;
    wold=w;
    for i=1:N
      %Computation of the distances
      dist=sum((X(:,i)*ones(1,m)-w).^ 2);
      [mval,mind]=min(dist);
      %Updating the representatives
      w=w+gl*(X(:,i)*ones(1,m)-w);
```

```
    w(:,mind)=w(:,mind)+(gw-gl)*(X(:,i)-w(:,mind));
  end
  diff=sum(sum(abs(w-wold)))
end
%Assigning vectors to clusters
bel=zeros(1,N);
for i=1:N
  dist=sum((X(:,i)*ones(1,m)-w).^2);
  [mval,mind]=min(dist);
  bel(i)=mind;
end
```

15.2 *Self-Organizing Map.* Write a MATLAB function named *som_experi* that implements the Self-Organizing Map. More specifically, this function takes as input (a) an $l \times N$ dimensional matrix X each column of which is a data vector, (b) the cluster label (a positive integer) where each vector belongs (this is used only in the plot of the results), (c) the number of iterations, *iter*, the algorithm will perform, (d) the size *side* of the side of the two-dimensional grid (only squared grids are considered). The function returns: (a) a matrix w each column of which contains the final estimates of the representatives and (b) a plot of the grid after the convergence of SOM, where the representatives of the same cluster are denoted by the same color.

Solution

The following function can plot a map for up to 4 different clusters.

```
function w=som_experi(X,y,iter,side)
  [l,N]=size(X);
  p=[side side];
  q=side^2; %Number of representatives
  minmax=[];
  for i=1:l
    minmax=[minmax; min(X(i,:)) max(X(i,:))];
  end
  % Defining and training the SOM
  net=newsom(minmax,p,'gridtop', 'mandist');
  net.trainParam.epochs=iter;
  net.trainParam.show=50;
  net = train(net,X);
  % Check if representatives represent data points
  % of a cluster.
  w=net.iw1';
  repr=zeros(1,q);
  map=zeros(side,side);
```

```
for i=1:N
  [s1,s2]=min(sum((X(:,i)*ones(1,q)-w).^2));
  repr(s2)=y(i);
end
% Creation of the map
for i=1:q
  i1=fix(i/side)+(mod(i,side)>0);
  i2=mod(i,side)+side*(mod(i,side)==0);
  map(i1,i2)=repr(i);
end
% Plot of the map. Up to four clusters can be plotted
if(max(y)<=4)
  figure(1), hold on
  palet=['k.';'ro'; 'ko'; 'go'; 'bo'];
  for i=1:side
    for j=1:side
      figure(1), plot(j,i,palet(map(i,j)+1,:))
    end
  end
end
```

Computer Experiments

15.1 a. Generate three data sets each one consisting of $q = 100$ two dimensional vectors stemming from normal distributions with means $[1, 1]^T$, $[5, 5]^T$, $[9, 1]^T$ and covariance matrices all equal to the 2×2 identity matrix I. Form the matrix X using as columns the data vectors from these data sets.

b. Initialize randomly $m = 3$ representatives using the *rand_vec* function (see in the "computer programs" section of chapter 14) and apply the leaky learning algorithm with $gw = 0.1, gl = 0.001$, *maxiter* $= 200$ and $e = 0.001$.

c. Repeat (b) with $gl = 0$ (basic competitive learning algorithm), initializing the representatives from the same values as in (b).

d. Comment on the results.

15.2 a. Repeat 1(b) and 1(c) using as initial estimates for the representatives the following: $[-100, -100]^T$, $[3.5, 4.5]^T$, $[2.5, 3]^T$. Comment on the results.

b. Taking into account the comments of 1 and 2(a) can you propose a combination of the leaky learning algorithm and the basic competitive learning algorithm that takes advantage of the merits of both methods?

15.3 a. Consider three 3-dimensional Gaussian distributions with means $[0.3,\ 0.3,\ 0.3]^T$, $[0.7,\ 0.7,\ 0.7]^T$ and $[0.3,\ 0.7,\ 0.3]^T$ and covariance matrices equal to $0.01I$, where I is the 3×3 identity matrix. Generate 100 data vectors from each distribution and let X be the resulting data set containing the above vectors (300 in total).

b. Apply the Self-Organizing Map (SOM) on the previous data set using a squared two-dimensional grid of size 10×10. Let the algorithm run for 300 iterations.

c. Repeat (b) for grid sizes 6×6 and 15×15.

d. Comment on the results.

REFERENCES

[Achl 01] Achlioptas D. "Database-friendly random projections," *Symposium on Principles of Database Systems (PODS)*, pp. 274–281, 2001.

[Agga 99] Aggarwal C.C., Wolf J.L., Yu P.S., Procopiuc C., Park J.S. "Fast algorithms for projected clustering," *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, pp. 61–72, 1999.

[Agga 00] Aggarwal C.C., Yu P.S. "Finding generalized projected clusters in high dimensional spaces," *Proceedings of the 2000 ACM SIGMOD International Conference on Management and Data*, pp. 70–81, 2000.

[Agra 98] Agrawal R., Gehrke J., Gunopoulos D., Raghavan P. "Automatic subspace clustering of high dimensional data for data mining applications," *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pp. 94–105, 1998.

[Ahal 90] Ahalt S.C., Krishnamurthy A.K., Chen P., Melton D.E. "Competitive learning algorithms for vector quantization," *Neural Networks*, Vol. 3, pp. 277–290, 1990.

[Al-S 95] Al-Sultan K.S. "A tabu search to the clustering problem," *Pattern Recognition*, Vol. 28(9), pp. 1443–1451, 1995.

[Al-S 93] Al-Sultan K.S., Selim S.Z. "A global algorithm for the fuzzy clustering problem," *Pattern Recognition*, Vol. 26(9), pp. 1357–1361, 1993.

[Alts 97] Altschul S.F., Madden T.L., Schaffer A.A., Zhang J., Zhang Z., Miller W., Lipman D.J. "Gapped-BLAST and PSI-BLAST: A new generation of protein database search programs," *Nucleic Acids Research*, Vol. 25, pp. 3389–3402, 1997.

[Andr 94] Andrey P., Tarroux P. "Unsupervised image segmentation using a distributed genetic algorithm," *Pattern Recognition*, Vol. 27(5), pp. 659–673, 1994.

[Anke 99] Ankerst M., Breunig M., Kriegel H.-P., Sander J. "OPTICS: Ordering points to identify clustering structure," *Proceedings of the ACM SIGMOD Conference*, pp. 49–60, Philadelphia, PA, 1999.

[Atiy 90] Atiya A.F. "An unsupervised learning technique for artificial neural networks," *Neural Networks*, Vol. 3, pp. 707–711, 1990.

[Ayad 03] Ayad H., Kamel M. "Finding natural clusters using multi-clusterer combiner based one shared nearest neighbors," *Multiple Classifier Systems: 4th International Workshop*, 2003.

[Bane 02] Banerjee A., Ghosh J. "On scaling up balanced clustering algorithms," *Proceedings of the 2nd SIAM International Conference on Data Mining*, pp. 333–349, Arlington, VA, 2002.

[Banz 90] Banzhaf W., Haken H. "Learning in a competitive network," *Neural Networks*, Vol. 3, pp. 423–435, 1990.

[Basu 04] Basu S., Bilenko M., Mooney R.J. "A probabilistic framework for semi-supervised clustering," *International Conference on Knowledge Discovery and Data Mining*, pp. 59–68, 2004.

[Belk 03] Belkin M., Niyogi P. "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, Vol. 15(6), pp. 1373–1396, 2003.

[Ben 99] Ben-Dor A., Shamir R., Yakhimi Z. "Clustering gene expression patterns," *Journal of Computational Biology*, Vol. 6, pp. 281–297, 1999.

[Ben 01] Ben-Hur A., Horn D., Siegelmann H.T., Vapnik V. "Support vector clustering," *Journal of Machine Learning Research*, Vol. 2, pp. 125–137, 2001.

[Bene 00] Bennett K.P., Bradley P.S., Demiriz A. "Constraint *k*-means clustering," *Technical Report MSR-TR-2000-65*, Microsoft Research, Redmond, CA, 2000.

[Beng 99] Bengio Y. "Markovian models for sequential data," *Neural Computation Survey*, Vol. 2, pp. 129–162, 1999.

[Beni 94] Beni G., Liu X. "A least biased fuzzy clustering method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, pp. 954–960, September 1994.

[Benv 87] Benveniste A., Metivier M., Priouret P. *Adaptive Algorithms and Stochastic Approximation*, Springer-Verlag, 1987.

[Berk 02] Berkhin P. "Survey of clustering data mining techniques," Technical report, Accrue Software, San Jose, CA, 2002.

[Bhan 91] Bhanu B., Lee S., Ming J. "Self-optimizing image segmentation system using a genetic algorithm," *Proceedings, Fourth International Conference on Genetic Algorithms*, pp. 362–369, 1991.

[Bisw 98] Biswas G., Weinberg J.B., Fisher D.H. "ITERATE: A conceptual clustering algorithm for data mining," *IEEE Transactions on Systems, Man and Cybernetics, Part C*, Vol. 28(2), pp. 100–111, 1998.

[Blum 97] Blum A., Langley P. "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, Vol. 97, pp. 245–271, 1997.

[Bohm 00] Bohm C., Braunmuller B., Breunig M., Kriegel H.P. "High performance clustering based on the similarity join," *Proceedings of the 9th International Conference on Information and Knowledge Management, CIKN*, pp. 298–313, Washington, DC, 2000.

[Brow 92] Brown D.E., Huntley C.L. "A practical application of simulated annealing to clustering," *Pattern Recognition*, Vol. 25(4), pp. 401–412, 1992.

[Burr 91] Burrascano P. "Learning vector quantization for the probabilistic neural network," *IEEE Transactions on Neural Networks*, Vol. 2(4), pp. 458–461, 1991.

[Butl 96] Butler D., Jiang J. "Distortion equalized fuzzy competitive learning for image data vector quantization," *Proceedings of ICAPPS'96*, pp. 3390–3393, 1996.

[Cama 05] Camastra F. "A novel kernel method for clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 27(5), pp. 801–805, 2005.

[Chan 02] Chang J.-W., Jin D.-S. "A new cell-based clustering method for large high-dimensional data in data mining applications," *Proceedings of 2002 ACM Symposium on Applied Computing*, pp. 503–507, 2002.

[Chan 94] Chan P., Schlag M., Zien J. "Spectral *k*-way ratio cut partitioning," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 13, pp.1088–1096, 1994.

[Chen 94] Chen L., Chang S. "An adaptive conscientious competitive learning algorithm and its applications," *Pattern Recognition*, Vol. 27(12), pp. 1787–1813, 1994.

[Chen 99] Cheng C.-H., Fu A.W., Zhang Y. "Entropy-based subspace clustering for mining numerical data," *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 84–93, 1999.

[Chen 03] Chen X. "An improved branch and bound algorithm for feature selection," *Pattern Recognition Letters*, Vol. 24, pp. 1925–1933, 2003.

[Chen 05] Chen C.-Y., Hwang S.-C., Oyang Y.-J., "A statistics-based approach to control the quality of subclusters in incremental gravitational clustering", *Pattern Recognition*, Vol. 38, pp. 2256–2269, 2005.

[Chia 03] Chiang J., Hao P. "A new kernel-based fuzzy clustering approach: Support vector clustering with cell growing," *IEEE Transactions on Fuzzy Systems*, Vol. 11(4), pp. 518–527, 2003.

[Chou 97] Chou C.S., Siu W. "Distortion sensitive competitive learning for vector quantizer design," *IEEE Proc.*, pp. 3405–3408, 1997.

[Chow 97] Chowdhury N., Murthy C.A. "Minimal spanning tree based clustering technique: Relationship with Bayes classifier," *Pattern Recognition*, Vol. 30(11), pp. 1919–1929, 1997.

[Chun 97] Chung F.R.K. *Spectral Graph theory*, American Mathematical Society, 1997.

[Davi 05] Davidson I., Ravi I.I. "Agglomerative hierarchical clustering with constraints: theoretical and empirical results," *9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, pp. 59–70, 2005.

[Dasg 99] Dasgupta S., Gupta A. "An elementary proof of the Johnson–Lindenstrauss lemma," Technical Report TR-99-006, International Computer Science Institute, Berkeley, California, 1999.

[Dasg 99a] Dasgupta S. "Learning mixtures of Gaussians," *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1999.

[Dasg 00] Dasgupta S. "Experiments with random projections," *Proceedings of the 16th Conference of Uncertainty in Artificial Intelligence (UAI)*, 2000.

[Dave 97] Dave R.N., Krishnapuram R. "Robust clustering methods: A unified view," *IEEE Transactions on Fuzzy Systems*, Vol. 5(2), pp. 270–293, May 1997.

[Deer 90] Deerwester S., Dumais S.T., Landauer T.K., Furnas G.W., Harshman R.A. "Indexing by latent semantic analysis," *Jouranl of American Society of Information Sciences*, Vol. 41, pp. 391–407, 1990.

[Dela 80] Delattre M., Hansen P. "Bicriterion cluster analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 2(4), pp. 277–291, July 1980.

[Devo 95] Devorer J.L. *Probability and Statistics for Engineering and the Sciences*, (4th ed.), Duxbury Press, 1995.

[Diam 07] Diamantaras K. *Artificial Neural Networks*, Klidarithmos, 2007, (in Greek).

[Dill 07] Dillon I., Guan Y., Kullis B. "Weighted graph cuts without eigenvectors: A multilevel approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 29(11), pp. 1945–1957, 2007.

[Ding 99] Ding C.H.Q. "A similarity-based probability model for latent semantic indexing," *Proceedings of the 22th ACM SIGIR Conference*, pp. 59–65, 1999.

[Ding 02] Ding C.H.Q., He X., Zha H., Simon H.D. "Adaptive dimension reduction for clustering high dimensional data," *Proceedings of the 2nd IEEE International Conference on Data Mining*, pp. 147–154, 2002.

[Durb 98] Durbin R., Eddy S., Krogh A., Mitchison G. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press, UK, 1998.

[Este 96] Ester M., Kriegel H.-P., Sander J., Xu X. "A density-based algorithm for discovering clusters in large spatial databases with noise," *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pp. 226–231, Portland, OR, 1996.

[Ever 01] Everitt B., Landau S., Leese M. *Cluster Analysis*, Arnold, 2001.

[Fern 04] Fern X.Z., Brodley C.E. "Solving ensemble problems by bipartite graph partitioning," *Proceedings of the 21th International Conference on Machine Learning*, Banff, Canada, 2004.

[Fern 03] Fern X.Z., Brodley C.E. "Random projection for high dimensional data clustering: A cluster ensemble approach," *Proceedings of the 20th International Conference on Machine Learning*, 2003.

[Fisc 05] Fischer I., Poland J. "Amplifying the block matrix structure for spectral clustering," *Technical Report No. IDSIA-03-05*, Instituto Dalle Molle di Studi sull' Intelligenza Artificialle (IDSIA), 2005.

[Fish 87] Fisher D. "Knowledge acquisition via incremental conceptual clustering," *Machine Learning*, Vol. 2, pp. 139–172, 1987.

[Fred 05] Fred A., Jain A.K., "Combining multiple clustering using evidence accumulation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 27(6), pp. 835–850, 2005.

[Frie 02] Friedman J.H., Meulman "Clustering objects on subsets of attributes," *http://citeseer.nj.nec.com/friedman02clustering.html*, 2002.

[Frig 97] Frigui H., Krishnapuram R. "Clustering by competitive agglomeration," *Pattern Recognition*, Vol. 30(7), pp. 1109–1119, 1997.

[Fu 93] Fu L., Yang M., Braylan R., Benson N. "Real-time adaptive clustering of flow cytometric data," *Pattern Recognition*, Vol. 26(2), pp. 365–373, 1993.

[Fuku 90] Fukunaga K. *Introduction to Statistical Pattern Recognition*, 2nd ed., Academic Press, 1990.

[Gabr 69] Gabriel K.R., Sokal R.R. "A new statistical approach to geographic variation analysis," *Syst. Zool*. Vol. 18, pp. 259–278, 1969.

[Gan 04] Gan G., Wu J. "Subspace clustering for high dimensional categorical data," *ACM SIGKDD Explorations Newsletter*, Vol. 6(2), pp. 87–94, 2004.

[Gema 84] Geman S., Geman D. "Stochastic relaxation, Gibbs distribution and Bayesian restoration of images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 6, pp. 721–741, 1984.

[Gers 79] Gersho A. "Asumptotically optimal block quantization," *IEEE Transactions on Information Theory*, Vol. 25(4), pp. 373–380, 1979.

[Gers 92] Gersho A., Gray R.M. *Vector Quantization and Signal Compression*, Kluwer Academic, 1992.

[Giro 02] Girolami M. "Mercer kernel-based clustering in feature space," *IEEE Transactions on Neural Networks*, Vol. 13(2), pp. 780–784, 2002.

[Goil 99] Goil S., Nagesh H., Choudhary A. "Mafia: Efficient and scalable subspace clustering for very large data sets," Technical Report CPDC-TR-9906-010, Northwestern University, June 1999.

[Goks 02] Goksay E., Principe J.C. "Information theoretic clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 24(2), pp. 158–171, 2002.

[Golu 89] Golub G.H., Van Loan C.F. *Matrix Computations*, John Hopkins Press, 1989.

[Gonz 93] Gonzalez R.C., Woods R.E. *Digital Image Processing*, Addison Wesley, 1993.

[Gray 84] Gray R.M. "Vector quantization," *IEEE ASSP Magazine*, pp. 4–29, April 1984.

[Gros 76a] Grossberg S. "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors," *Biological Cybernetics*, Vol. 23, pp. 121–134, 1976.

[Gros 76b] Grossberg S. "Adaptive pattern classification and universal recoding: II. Feedback, expectation, olfaction, illusions," *Biological Cybernetics*, Vol. 23, pp. 187–202, 1976.

[Gros 87] Grossberg S. "Competitive learning: From interactive activation to adaptive resonance," *Cognitive Science*, Vol. 11, pp. 23–63, 1987.

[Gusf 97] Gusfield D. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*, Cambridge University Press, UK, 1997.

[Halk 08] Halkidi M., Gunopulos D., Vazirgiannis M., Kumar N., Domeniconi C. "A clustering framework based on subjective and objective validity criteria," *ACM Transactions on Knowledge Discovery from Data*, Vol. 1(4), 2008.

[Hage 98] Hagen L., Kahng A. "New spectral methods for ratio cut partitioning and clustering," *IEEE Transactions on Computer-Aided-Design of Integrated Circuits and Systems*, Vol. 11, pp. 1074–1085, 1998.

[Hage 92] Hagen L.W., Kahng A.B. "New spectral methods for ratio cut partitioning and clustering," *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, Vol. 11(9), pp. 1074–1085, 1992.

[Hech 88] Hecht-Nielsen R. "Applications of counter-propagation networks," *Neural Networks*, Vol. 1(2), pp. 131–141, 1988.

[Hend 93] Hendrickson B., Leland R. "Multidimensional spectral load balancing," *Proceedings 4th SIAM Conference on Parallel Processing*, pp. 953–961, 1993.

[Hinn 98] Hinneburg A., Keim D. "An efficient approach to clustering large multimedia databases with noise," *Proceedings of the 4th ACM SIGKDD*, pp. 58–65, New York, NY, 1998.

[Hinn 99] Hinneburg A., Keim D.A. "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering," *Proceedings of the 25th Conference on Very Large Databases*, Edinburgh, Scotland, 1999.

[Hoch 64] Hochberg J.E. *Perception*, Prentice-Hall, 1964.

[Hofm 97] Hofmann T., Buchmann J.M. "Pairwise data clustering by deterministic annealing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19(1), pp. 1–14, 1997.

[Inok 04] Inokuchi R., Miyamoto S. "LVQ clustering and SOM using a kernel function," *Proceedings of the IEEE International Conference on Fuzzy Systems*, Vol. 3, pp. 1497–1500, 2004.

[Jarv 78] Jarvis R.A. "Shared nearest neighbor maximal spanning trees for cluster analysis," *Proceedings, 4th Joint Conference on Pattern Recognition*, Kyoto, Japan, pp. 308–313, 1978.

[Jayn 82] Jaynes E.T. "On the rationale of maximum-entropy methods," *Proc. IEEE*, Vol. 70(9), pp. 939–952, September 1982.

[Jens 04] Jenssen R., Eltoft T., Principe J.C. "Information theoretic spectral clustering," *Proceedings of the International Joint Conference on Neural Networks*, pp. 111–116, 2004.

[Kann 00] Kannan R., Vempala S., Vetta A. "On clusterings- good, bad and spectral," *41st Annual Symposium on Foundations of Computer Science, FOCS*, pp. 367–377, Redondo Beach, California, USA, 2000.

[Kara 96] Karayiannis N.B., Pai P. "Fuzzy algorithms for learning vector quantization," *IEEE Transactions on Neural Networks*, Vol. 7(5), pp. 1196–1211, 1996.

[Kask 98] Kaski S., Kangas J., Kohonen T. "Bibliography of SOM papers: 1981–1997," *Neural Computing Reviews*, Vol. 1, pp. 102–350, 1998.

[Kirk 83] Kirkpatrick S., Gelatt C.D. Jr., Vecchi M.P. "Optimization by simulated annealing," *Science*, Vol. 220, pp. 671–680, 1983.

[Klei 89] Klein R.W., Dubes R.C. "Experiments in projection and clustering by simulated annealing," *Pattern Recognition*, Vol. 22(2), pp. 213–220, 1989.

[Kodr 88] Kodratoff Y., Tecuci G. "Learning based on conceptual distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10(6), pp. 897–909, 1988.

[Koha 97] Kohavi R., John G. "Wrappers for feature subset selection," *Artificial Intelligence*, Vol. 97(1–2), pp. 273–324, 1997.

[Koho 89] Kohonen T. *Self-Organization and Associative Memory*, 2nd ed., Springer-Verlag, 1989.

[Koho 95] Kohonen T. *Self-Organizing Maps*, Springer-Verlag, 1995.

[Koon 75] Koontz W.L.G., Narendra P.M., Fukunaga K. "A branch and bound clustering algorithm," *IEEE Transactions on Computers*, Vol. 24(9), pp. 908–914, September 1975.

[Koon 76] Koontz W.L.G., Narendra P.M., Fukunaga K. "A graph-theoretic approach to nonparametric cluster analysis," *IEEE Transactions on Computers*, Vol. 25(9), pp. 936–944, September 1976.

[Kosk 91] Kosko B. "Stochastic competitive learning," *IEEE Transactions on Neural Networks*, Vol. 2(5), pp. 522–529, 1991.

[Kosk 92] Kosko B. *Neural Networks for Signal Processing*, Prentice Hall, 1992.

[Kotr 92] Kotropoulos C., Auge E., Pitas I. "Two-layer learning vector quantizer for color image quantization," *Signal Processing VI*, pp. 1177–1180, 1992.

[Kots 04] Kotsiantis S.B., Pintelas P.E. "Recent advances in clustering: a brief survey," *WSEAS Transactions on Information Science and Applications*, Vol. 1(1), pp. 73–81, 2004.

[Kout 95] Koutroumbas K. "Hamming neural networks, architecture design and applications," Ph.D. dissertation, Department of Informatics, University of Athens, 1995 (in Greek).

[Kuli 05] Kulis B., Basu S., Dhilon I.S., Mooney R.J., "Semi-supervised graph clustering: a kernel approach," *International Conference on Machine Learning*, pp. 457–464, 2005.

[Laar 87] van Laarhoven P.J.M., Aarts E.H.L. *Simulated Annealing: Theory and Applications*, Reidel, Hingham, MA, 1987.

[Lang 05] Lange T., Buhmann J.M. "Combining partitions by probabilistic label aggregation," *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge discovery in data mining*, pp. 147–155, 2005.

[Law 04] Law M., Topchy A., Jain A.K., "Multiobjective data clustering," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 2, pp. 424–430, 2004.

[Liew 05] Liew A.W.C., Yan H., Yang M. "Pattern recognition techniques for the emerging field of bioinformatics: A review," *Pattern Recognition*, Vol. 38, pp. 2055–2073, 2005.

[Likh 97] Likhovidov V. "Variational approach to unsupervised learning algorithms of neural networks," *Neural Networks*, Vol. 10(2), pp. 273–289, 1997.

[Lind 80] Linde Y., Buzo A., Gray R.M. "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, Vol. 28(1), pp. 84–95, 1980.

[Liu 98] Liu H., Motoda H. *Feature Selection for Knowledge Discovery and Data Mining*, Kluwer Academic Publishers, 1998.

[Liu 00] Liu B., Xia Y., Yu P.S. "Clustering through decision tree construction," *Proceedings of the ninth International Conference on Information and Knowledge Management*, pp. 20–29, 2000.

[Luen 84] Luenberger D.G. *Linear and Nonlinear Programming*, Addison Wesley, 1984.

[Macd 00] Macdonald D., Fyfe C. "The kernel self- organizing map," *Proceedings of the fourth International Conference on Knowledge-based Intelligent Engineering Systems and Allied Technologies*, Vol. 1, pp. 317–320, 2000.

[Mals 73] von der Maslburg. "Self-organization sensitive cells in the striate cortex," *Kybernetic*, Vol. 14, pp. 85–100, 1973.

[Mara 80] Maragos P., Schafer R.W. "Morphological systems for multidimensional signal processing," *Proc. IEEE*, Vol. 78(4), pp. 690–710, April 1980.

[Mart 93] Martinetz T.M., Berkovich S.G., Schulten K.J. "Neural-gas network for vector quantization and its application to time-series prediction," *IEEE Transactions on Neural Networks*, Vol. 4(4), pp. 558–569, July 1993.

[Masu 93] Masuda T. "Model of competitive learning based upon a generalized energy function," *Neural Networks*, Vol. 6, pp. 1095–1103, 1993.

[Matt 91] Matthews G., Hearne J. "Clustering without a metric," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13(2), pp. 175–184, 1991.

[Maul 00] Maulik U., Bandyopadhyay S. "Genetic algorithm-based clustering technique," *Pattern Recognition*, Vol. 33, pp. 1455–1465, 2000.

[Meil 00] Meilă M., Shi J. "A random walk view of spectral segmentation," *Proceedings Neural Information Processing Conference*, pp. 873–879, 2000.

[Metr 53] Metropolis N., Rosenbluth A.W., Rosenbluth M.N., Teller A.H., Teller E. "Equations of state calculations by fast computing machines," *Journal of Chemical Physics*, Vol. 21, pp. 1087–1092, 1953.

[Mich 94] Michalevitz Z. *Genetic Algorithms + Data Structures = Evolutionary Programming*, 2nd ed., Springer-Verlag, 1994.

[Mill 01] Miller W. "Comparison of genomic DNA sequences: Solved and unsolved problems," *Bioinformatics*, Vol. 17, pp. 391–397, 2001.

[Mirk 01] Mirkin B. "Reinterpreting the category utility function," *Machine Learning*, Vol. 45(2), pp. 219–228, 2001.

[Mora 00] Morales E., Shih F.Y. "Wavelet coefficients clustering using morphological operations and pruned quadtrees," *Pattern Recognition*, Vol. 33, pp. 1611–1620, 2000.

[Ng 01] Ng A.Y., Jordan M., Weiss Y. "On spectral clustering analysis and an algorithm," *Proceedings 14th Conference on Advances in Neural Information Processing Systems*, 2001.

[Noso 08] Nosovskiy G.V., Liu D., Sourina O. "Automatic clustering and boundary detection algorithm based on adaptive influence function," *Pattern Recognition*, Vol. 41, pp. 2757–2776, 2008.

[Nyec 92] Nyeck A., Mokhtari H., Tosser-Roussey A. "An improved fast adaptive search algorithm for vector quantization by progressive codebook arrangement," *Pattern Recognition*, Vol. 25(8), pp. 799–802, 1992.

[Oyan 01] Oyang Y.-J., Chen C.-Y., Yang T.-W. "A study on the hierarchical data clustering algorithm based on gravity theory," *Lecture Notes in Artificial Intelligence: Principles of Data Mining and Knowledge Discovery*, Vol. 2168, pp. 350–361, Springer, 2001.

[Ozbo 95] Osbourn G.C., Martinez R.F. "Empirically defined regions of influence for cluster analysis," *Pattern Recognition*, Vol. 28(11), pp. 1793–1806, 1995.

[Papa 82] Papadimitriou C.H., Steiglitz K. *Combinatorial Optimization: algorithms and complexity*, Prentice-Hall, 1982.

[Pars 04] Parsons L., Haque E., Liu H. "Subspace clustering for high dimensional data: A review," *ACM SIGKDD Explorations Newsletter*, Vol. 6(1), pp. 90–105, 2004.

[Pear 88] Pearson W. "Improved tools for biological sequence comparison," *Proceedings National Academy of Sciences*, Vol. 85, pp. 2444–2448, 1988.

[Pedr 97] Pedrycz W., Waletzky J. "Neural–network front ends in unsupervised learning," *IEEE Transactions on Neural Networks*, Vol. 8(2), pp. 390–401, March 1997.

[Pena 01] Pena J.M., Lozano J.A., Larranaga P., Inza I. "Dimensionality reduction in unsupervised learning of conditional gaussian networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 23(6), pp. 590–603, 2001.

[Post 93] Postaire J.G., Zhang R.D., Lecocq-Botte C. "Cluster analysis by binary morphology," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(2), pp. 170–180, 1993.

[Proc 02] Procopiuc C.M., Jones M., Agarwal P.K., Murali T.M. "A Monte-Carlo algorithm for fast projective clustering," *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pp. 418–427, 2002.

[Qian 00] Qian Y., Suen C. "Clustering combination method," *15th International Conference on Pattern Recognition (ICPR00)*, Vol. 2, pp. 732–735, 2000.

[Qiu 07] Qiu H., Hancock E.R. "Clustering and embedding using commute times," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 29(11), pp. 1873–1890, 2007.

[Robe 00] Roberts S.J., Everson R., Rezek I. "Maximum certainty data partitioning," *Pattern Recognition*, pp. 833–839, 2000.

[Rose 91] Rose K. "Deterministic annealing, clustering and optimization," Ph.D. dissertation, California Institute of Technology, 1991.

[Rose 93] Rose K., Gurewitz E., Fox G.C. "Constrained clustering as an optimization method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(8), pp. 785–794, 1993.

[Rume 86] Rumelhart D.E., Zipser D. "Feature discovery by competitive learning," *Cognitive Science*, Vol. 9, pp. 75–112, 1986.

[Rume 86] Rumelhart D.E., McLelland J.L. *Parallel Distributed Processing*, Cambridge, MA: MIT Press, 1986.

[Sand 98] Sander J., Ester M., Kriegel H.-P., Xu X. "Density based clustering in spatial databases: The algorithm GDBSCAN and its applications," *Data Mining and Knowledge Discovery*, Vol. 2(2), pp. 169–194, 1998.

[Sche 97] Scheunders P. "A genetic c-means clustering algorithm applied to color image quantization," *Pattern Recognition*, Vol. 30(6), pp. 859–866, 1997.

[Scho 98] Schölkopf B., Smola A., Müller K.R. "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, Vol. 10(5), pp. 1299–1319, 1998.

[Scot 90] Scott G., Longuet-Higgins H. "Feature grouping by relocalization of eigenvectors of the proximity matrix," *Proceedings British Machine Vision Conference*, pp. 103–108, 1990.

[Shei 98] Sheikholeslami G., Chatterjee S., Zhang A. "WaveCluster: A multi-resolution clustering approach for very large spatial databases," *Proceedings of the 24th Conference on Very Large Databases*, New York, 1998.

[Shi 00] Shi J., Malik J. "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22(8), pp.888–905, 2000.

[Stre 02] Strehl A., Ghosh J. "Cluster ensembles - a knowledge reuse framework for combining multiple partitions," *Journal of Machine Learning Research*, Vol. 3, pp. 583–617, 2002.

[Stre 00] Strehl A., Ghosh J. "A scalable approach to balanced, high-dimensional clustering of market baskets," *Proceedings of the 17th International conference on High Performance Computing*, pp. 525–536, Springer LNCS, Bangalore, India, 2000.

[Sung 00] Sung C.S., Jin H.W., "A tabu-search-based heuristic for clustering," *Pattern Recognition*, Vol. 33, pp. 849–858, 2000.

[Szu 86] Szu H. "Fast simulated annealing," in *Neural Networks for Computing* (Denker J.S., ed.), American Institute of Physics, 1986.

[Tax 99] Tax D.M.J., Duin R.P.W. "Support vector domain description," *Pattern Recognition Letters*, Vol. 20, pp. 1191–1199, 1999.

[Topc 05] Topchy A., Jain A.K., Punch W. "Clustering ensembles: Models of consensus and weak partitions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 27(12), pp. 1866–1881, 2005.

[Topc 04] Topchy A., Minaei B., Jain A.K., Punch W. "Adaptive clustering ensembles," *Proceedings of the International Conference on Pattern Recognition (ICPR)*, U.K., August 23–26, 2004.

[Tou 74] Tou J.T., Gonzales R.C. *Pattern Recognition Principles*, Addison-Wesley, 1974.

[Tous 80] Toussaint G.T. "The relative neighborhood graph of a finite planar set," *Pattern Recognition*, Vol. 12, pp. 261–268, 1980.

[Touz 88] Touzani A., Postaire J.G. "Mode detection by relaxation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10(6), pp. 970–977, 1988.

[Tsen 00] Tseng L.Y., Yang S.B., "A genetic clustering algorithm for data with non-spherical-shape clusters," *Pattern Recognition*, Vol. 33, pp. 1251–1259, 2000.

[Tsyp 73] Tsypkin Y.Z. *Foundations of the Theory of Learning Systems*, Academic Press, 1973.

[Tung 01] Tung A.K.H., Ng R.T., Lakshmanan L.V.S., Han J. "Constraint-based clustering in large databases," *Proceedings of the 8th International Conference on Database Theory*, London, 2001.

[Uchi 94] Uchiyama T., Arbib M.A. "An algorithm for competitive learning in clustering problems," *Pattern Recognition*, Vol. 27(10), pp. 1415–1421, 1994.

[Ueda 94] Ueda N., Nakano R. "A new competitive learning approach based on an equidistortion principle for designing optimal vector quantizers," *Neural Networks*, Vol. 7(8), pp. 1211–1227, 1994.

[Urqu 82] Urquhart R. "Graph theoretical clustering based on limited neighborhood sets," *Pattern Recognition*, Vol. 15(3), pp. 173–187, 1982.

[Verm 03] Verma D., Meilă M. "A comparison of spectral clustering algorithms," *Technical Report, UW-CSE-03-05-01*, University of Washington, Seattle, CSE Department, 2003.

[Vish 00] Vishwanathan S.V.N., Murty M.N., "Kohonen's SOM with cashe," *Pattern Recognition*, Vol. 33, pp. 1927–1929, 2000.

[vonL 07] von Luxburg U. "A Tutorial on Spectral Clustering," *Statistics and Computing*, Vol. 17(4), 2007.

[Wags 01] Wagstaff K., Cardie C., Rogers S., Schrodl S. "Constraint k-means clustering with background knowledge," *International Conference on Machine Learning*, pp. 577–584, 2001.

[Weis 99] Weiss Y. "Segmentation using eigenvectors: A unifying view," *Proceedings 7th IEEE International Conference on Computer Vision*, pp. 975–982, 1999.

[Wang 07] Wang D., Shi L., Yeung D.S., Tsang E.C.C., Heng P.A. "Ellipsoidal support vector clustering for functional MRI analysis," *Pattern Recognition*, Vol. 40(10), pp. 2685–2695, 2007.

[Woo 02] Woo K.-G., Lee J.-H. "FINDIT: A fast and intelligent subspace clustering algorithm using dimension voting," Ph.D. Thesis, Korea Advanced Institute of Science and Technology, Taejon, Korea, 2002.

[Wu 93] Wu Z., Leahy R. "An optimal graph theoretic approach to data clustering: Theory and its applications to image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(11), pp. 1101–1113, 1993.

[Xian 08] Xiang T., Gong S. "Spectral clustering with eigenvalue selection," *Pattern Recognition*, Vol. 41(3), pp. 1012–1029, 2008.

[Xie 93] Xie Q., Laszlo A., Ward R.K. "Vector quantization technique for nonparametric classifier design," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(12), pp. 1326–1330, 1993.

[Xing 02] Xing E.P., Ng A.Y., Jordan M.I., Russell S.J. "Distance metric learning with application to clustering with side-information," *International Conference on Neural Information Processing Systems*, pp. 505–512, 2002.

[Xu 98] Xu X., Ester M., Kriegel H.P., Sander J. "A distribution-based clustering algorithm for mining in large spatial databases," *Proceedings of the 14th ICDE*, pp. 324–331, Orlando, FL, 1998.

[Xu 05] Xu R., Wunsch D. II "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, Vol. 16(3), pp. 645–677, 2005.

[Yama 80] Yamada Y., Tazaki S., Gray R.M. "Asymptotic performance of block quantizers with difference distortion measures," *IEEE Transactions on Information Theory*, Vol. 26(1), pp. 6–14, 1980.

[Yang 02] Yang J., Wang W., Wang H., Yu P. "δ-clusters: Capturing subspace correlation in a large data set," *Proceedings of the 18th International Conference on Data Engineering*, pp. 517–528, 2002.

[Yu 93] Yu B., Yuan B. "A more efficient branch and bound algorithm for feature selection," *Pattern Recognition*, Vol. 26, pp. 883–889, 1993.

[Yu 03] Yu L., Liu H. "Feature selection for high-dimensional data: A fast correlation-based filter solution," *Proceedings of the 20th International Conference on Machine Learning*, pp. 856–863, 2003.

[Zahn 71] Zahn C.T. "Graph-theoretical methods for detecting and describing gestalt clusters," *IEEE Transactions on Computers*, Vol. 20(1), pp. 68–86, January 1971.

[Zass-05] Zass R., Shashua A. "A unifying approach to hard and probabilistic clustering," *Proceedings of the 10th IEEE International Conference on Computer Vision, ICCV*, pp. 294–301, 2005.

[Zhu 94] Zhu C., Li L., He Z., Wang J. "A new competitive learning learning algorithm for vector quantization," *Proceedings of ICASSP'94*, pp. 557–560, 1994.