Chapter 7

Cluster Analysis: Advanced Concepts

"*The crowd is just as important as the group. It takes everything to make it work.*"—Levon Helm

7.1 Introduction

In the previous chapter, the basic data clustering methods were introduced. In this chapter, several advanced clustering scenarios will be studied, such as the impact of the size, dimensionality, or type of the underlying data. In addition, it is possible to obtain significant insights with the use of advanced supervision methods, or with the use of ensemble-based algorithms. In particular, two important aspects of clustering algorithms will be addressed:

1. *Difficult clustering scenarios:* Many data clustering scenarios are more challenging. These include the clustering of categorical data, high-dimensional data, and massive data. Discrete data are difficult to cluster because of the challenges in distance computation, and in appropriately defining a "central" cluster representative from a set of categorical data points. In the high-dimensional case, many irrelevant dimensions may cause challenges for the clustering process. Finally, massive data sets are more difficult for clustering due to scalability issues.

2. *Advanced insights:* Because the clustering problem is an unsupervised one, it is often difficult to evaluate the quality of the underlying clusters in a meaningful way. This weakness of cluster validity methods was discussed in the previous chapter. Many alternative clusterings may exist, and it may be difficult to evaluate their relative quality. There are many ways of improving application-specific relevance and robustness by using external supervision, human supervision, or meta-algorithms such as ensemble clustering that combine multiple clusterings of the data.

The difficult clustering scenarios are typically caused by particular aspects of the data that make the analysis more challenging. These aspects are as follows:

C. C. Aggarwal, *Data Mining: The Textbook*, DOI 10.1007/978-3-319-14142-8_7

1. *Categorical data clustering:* Categorical data sets are more challenging for clustering because the notion of similarity is harder to define in such scenarios. Furthermore, many intermediate steps in clustering algorithms, such as the determination of the mean of a cluster, are not quite as naturally defined for categorical data as for numeric data.

2. *Scalable clustering:* Many clustering algorithms require multiple passes over the data. This can create a challenge when the data are very large and resides on disk.

3. *High-dimensional clustering:* As discussed in Sect. 3.2.1.2 of Chap. 3, the computation of similarity between high-dimensional data points often does not reflect the intrinsic distance because of many irrelevant attributes and concentration effects. Therefore, many methods have been designed that use projections to determine the clusters in relevant subsets of dimensions.

Because clustering is an unsupervised problem, the quality of the clusters may be difficult to evaluate in many real scenarios. Furthermore, when the data are noisy, the quality may also be poor. Therefore, a variety of methods are used to either supervise the clustering, or gain advanced insights from the clustering process. These methods are as follows:

1. *Semisupervised clustering:* In some cases, partial information may be available about the underlying clusters. This information may be available in the form of labels or other external feedback. Such information can be used to greatly improve the clustering quality.

2. *Interactive and visual clustering:* In these cases, feedback from the user may be utilized to improve the quality of the clustering. In the case of clustering, this feedback is typically achieved with the help of visual interaction. For example, an interactive approach may explore the data in different subspace projections and isolate the most relevant clusters.

3. *Ensemble clustering:* As discussed in the previous chapter, the different models for clustering may produce clusters that are very different from one another. Which of these clusterings is the best solution? Often, there is no single answer to this question. Rather the knowledge from multiple models may be combined to gain a more unified insight from the clustering process. Ensemble clustering can be viewed as a meta-algorithm, which is used to gain more significant insights from multiple models.

This chapter is organized as follows: Section 7.2 discusses algorithms for clustering categorical data. Scalable clustering algorithms are discussed in Sect. 7.3. High-dimensional algorithms are addressed in Sect. 7.4. Semisupervised clustering algorithms are discussed in Sect. 7.5. Interactive and visual clustering algorithms are discussed in Sect. 7.6. Ensemble clustering methods are presented in Sect. 7.7. Section 7.8 discusses the different applications of data clustering. Section 7.9 provides a summary.

7.2 Clustering Categorical Data

The problem of categorical (or discrete) data clustering is challenging because most of the primitive operations in data clustering, such as distance computation, representative determination, and density estimation, are naturally designed for numeric data. A salient observation is that categorical data can always be converted to binary data with the use of

Table 7.1: Example of a 2-dimensional categorical data cluster

Data	(Color, Shape)
1	(Blue, Square)
2	(Red, Circle)
3	(Green, Cube)
4	(Blue, Cube)
5	(Green, Square)
6	(Red, Circle)
7	(Blue, Square)
8	(Green, Cube)
9	(Blue, Circle)
10	(Green, Cube)

Table 7.2: Mean histogram and modes for categorical data cluster

Attribute	Histogram	Mode
Color	Blue= 0.4 Green = 0.4 Red = 0.2	Blue *or* Green
Shape	Cube = 0.4 Square = 0.3 Circle = 0.3	Cube

the binarization process discussed in Chap. 2. It is often easier to work with binary data because it is also a special case of numeric data. However, in such cases, the algorithms need to be tailored to binary data.

This chapter will discuss a wide variety of algorithms for clustering categorical data. The specific challenges associated with applying the various classical methods to categorical data will be addressed in detail along with the required modifications.

7.2.1 Representative-Based Algorithms

The centroid-based representative algorithms, such as k-means, require the repeated determination of centroids of clusters, and the determination of similarity between the centroids and the original data points. As discussed in Sect. 6.3 of the previous chapter, these algorithms iteratively determine the centroids of clusters, and then assign data points to their closest centroid. At a higher level, these steps remain the same for categorical data. However, the specifics of both steps are affected by the categorical data representation as follows:

1. *Centroid of a categorical data set:* All representative-based algorithms require the determination of a central representative of a set of objects. In the case of numerical data, this is achieved very naturally by averaging. However, for categorical data, the equivalent centroid is a probability histogram of values on *each attribute*. For each attribute i, and possible value v_j, the histogram value p_{ij} represents the fraction of the number of objects in the cluster for which attribute i takes on value v_j. Therefore, for a d-dimensional data set, the centroid of a cluster of points is a set of d different histograms, representing the probability distribution of categorical values of each attribute in the cluster. If n_i is the number of distinct values of attribute i, then such an approach will require $O(n_i)$ space to represent the centroid of the ith attribute. A cluster of 2-dimensional data points with attributes *Color* and *Shape* is illustrated in Table 7.1. The corresponding histograms for the *Color* and *Shape* attributes are illustrated in Table 7.2. Note that the probability values over a particular attribute always sum to one unit.

2. *Calculating similarity to centroids:* A variety of similarity functions between a pair of categorical records are introduced in Sect. 3.2.2 of Chap. 3. The simplest of these is match-based similarity. However, in this case, the goal is to determine the similarity

between a probability histogram (corresponding to a representative) and a categorical attribute value. If the attribute i takes on the value v_j for a particular data record, then the analogous match-based similarity is its histogram-based probability p_{ij}. These probabilities are summed up over the different attributes to determine the total similarity. Each data record is assigned to the centroid with the greatest similarity.

The other steps of the k-means algorithm remain the same as for the case of numeric data. The effectiveness of a k-means algorithm is highly dependent on the distribution of the attribute values in the underlying data. For example, if the attribute values are highly skewed, as in the case of market basket data, the histogram-based variation of the match-based measure may perform poorly. This is because this measure treats all attribute values evenly, however, rare attribute values should be treated with greater importance in such cases. This can be achieved by a prekshiprocessing phase that assigns a weight to each categorical attribute *value*, which is the inverse of its global frequency. Therefore, the categorical data records now have weights associated with each attribute. The presence of these weights will affect both probability histogram generation and match-based similarity computation.

7.2.1.1 k-Modes Clustering

In k-modes clustering, each attribute value for a representative is chosen as the mode of the categorical values for that attribute in the cluster. The *mode* of a set of categorical values is the value with the maximum frequency in the set. The modes of each attribute for the cluster of ten points in Table 7.1 are illustrated in Table 7.2. Intuitively, this corresponds to the categorical value v_j for each attribute i for which the frequency histogram has the largest value of p_{ij}. The mode of an attribute may not be unique if two categorical values have the same frequency. In the case of Table 7.2, two possible values of the mode are $(Blue, Cube)$, and $(Green, Cube)$. Any of these could be used as the representative, if a random tie-breaking criterion is used. The mode-based representative may not be drawn from the original data set because the mode of each attribute is determined independently. Therefore, the particular combination of d-dimensional modes obtained for the representative may not belong to the original data. One advantage of the mode-based approach is that the representative is also a categorical data record, rather than a histogram. Therefore, it is easier to use a richer set of similarity functions for computing the distances between data points and their modes. For example, the inverse occurrence frequency-based similarity function, described in Chap. 3, may be used to normalize for the skew in the attribute values. On the other hand, when the attribute values in a categorical data set are naturally skewed, as in market basket data, the use of modes may not be informative. For example, for a market basket data set, all item attributes for the representative point may be set to the value of 0 because of the natural sparsity of the data set. Nevertheless, for cases where the attribute values are more evenly distributed, the k-modes approach can be used effectively. One way of making the k-modes algorithm work well in cases where the attribute values are distributed unevenly, is by dividing the cluster-specific frequency of an attribute by its (global) occurrence frequency to determine a *normalized* frequency. This essentially corrects for the differential global distribution of different attribute values. The modes of this normalized frequency are used. The most commonly used similarity function is the match-based similarity metric, discussed in Sect. 3.2.2 of Chap. 3. However, for biased categorical data distributions, the inverse occurrence frequency should be used for normalizing the similarity function, as discussed in Chap. 3. This can be achieved indirectly by weighting each attribute of

each data point with the inverse occurrence frequency of the corresponding attribute *value*. With normalized modes and weights associated with each attribute of each data point, the straightforward match-based similarity computation will provide effective results.

7.2.1.2 k-Medoids Clustering

The medoid-based clustering algorithms are easier to generalize to categorical data sets because the representative data point is chosen from the input database. The broad description of the medoids approach remains the same as that described in Sect. 6.3.4 of the previous chapter. The only difference is in terms of how the similarity is computed between a pair of categorical data points, as compared to numeric data. Any of the similarity functions discussed in Sect. 3.2.2 of Chap. 3 can be used for this purpose. As in the case of k-modes clustering, because the representative is also a categorical data point (as opposed to a histogram), it is easier to directly use the categorical similarity functions of Chap. 3. These include the use of inverse occurrence frequency-based similarity functions that normalize for the skew across different attribute values.

7.2.2 Hierarchical Algorithms

Hierarchical algorithms are discussed in Sect. 6.4 of Chap. 6. Agglomerative bottom-up algorithms have been used successfully for categorical data. The approach in Sect. 6.4 has been described in a general way with a distance matrix of values. As long as a distance (or similarity) matrix can be defined for the case of categorical attributes, most of the algorithms discussed in the previous chapter can be easily applied to this case. An interesting hierarchical algorithm that works well for categorical data is *ROCK*.

7.2.2.1 ROCK

The *ROCK* (RObust Clustering using linKs) algorithm is based on an agglomerative bottom-up approach in which the clusters are merged on the basis of a similarity criterion. The *ROCK* algorithm uses a criterion that is based on the shared nearest-neighbor metric. Because agglomerative methods are somewhat expensive, the *ROCK* method applies the approach to only a sample of data points to discover prototype clusters. The remaining data points are assigned to one of these prototype clusters in a final pass.

The first step of the *ROCK* algorithm is to convert the categorical data to a binary representation using the binarization approach introduced in Chap. 2. For each value v_j of categorical attribute i, a new pseudo-item is created, which has a value of 1, only if attribute i takes on the value v_j. Therefore, if the ith attribute in a d-dimensional categorical data set has n_i different values, such an approach will create a binary data set with $\sum_{i=1}^{d} n_i$ binary attributes. When the value of each n_i is high, this binary data set will be sparse, and it will resemble a market basket data set. Thus, each data record can be treated as a binary transaction, or a set of items. The similarity between the two transactions is computed with the use of the Jaccard coefficient between the corresponding sets:

$$Sim(T_i, T_j) = \frac{|T_i \cap T_j|}{|T_i \cup T_j|}. \tag{7.1}$$

Subsequently, two data points T_i and T_j are defined to be *neighbors*, if the similarity $Sim(T_i, T_j)$ between them is greater than a threshold θ. Thus, the concept of neighbors implicitly defines a graph structure on the data items, where the nodes correspond to

the data items, and the links correspond to the neighborhood relations. The notation $Link(T_i, T_j)$ denotes a shared nearest-neighbor similarity function, which is equal to the number of shared nearest neighbors between T_i and T_j.

The similarity function $Link(T_i, T_j)$ provides a merging criterion for agglomerative algorithms. The algorithm starts with each data point (from the initially chosen sample) in its own cluster and then hierarchically merges clusters based on a similarity criterion between clusters. Intuitively, two clusters $\mathcal{C}_1$ and $\mathcal{C}_2$ should be merged, if the cumulative number of shared nearest neighbors between objects in $\mathcal{C}_1$ and $\mathcal{C}_2$ is large. Therefore, it is possible to generalize the notion of link-based similarity using clusters as arguments, as opposed to individual data points:

$$GroupLink(\mathcal{C}_i, \mathcal{C}_j) = \sum_{T_u \in \mathcal{C}_i, T_v \in \mathcal{C}_j} Link(T_u, T_v). \tag{7.2}$$

Note that this criterion has a slight resemblance to the group-average linkage criterion discussed in the previous chapter. However, this measure is not yet normalized because the *expected* number of cross-links between larger clusters is greater. Therefore, one must normalize by the expected number of cross-links between a pair of clusters to ensure that the merging of larger clusters is not unreasonably favored. Therefore, the normalized linkage criterion $V(\mathcal{C}_i, \mathcal{C}_j)$ is as follows:

$$V(\mathcal{C}_i, \mathcal{C}_j) = \frac{GroupLink(\mathcal{C}_i, \mathcal{C}_j)}{E[CrossLinks(\mathcal{C}_i, \mathcal{C}_j)]}. \tag{7.3}$$

The expected number of cross-links between $\mathcal{C}_i$ and $\mathcal{C}_j$ can be computed as function of the expected number of intracluster links $Intra(\cdot)$ in individual clusters as follows:

$$E[CrossLinks(\mathcal{C}_i, \mathcal{C}_j)] = E[Intra(\mathcal{C}_i \cup \mathcal{C}_j)] - E[Intra(\mathcal{C}_i)] - E[Intra(\mathcal{C}_j)]. \tag{7.4}$$

The expected number of intracluster links is specific to a single cluster and is more easily estimated as a function of cluster size q_i and θ. The number of intracluster links in a cluster containing q_i data points is heuristically estimated by the *ROCK* algorithm as $q_i^{1+2 \cdot f(\theta)}$. Here, the function $f(\theta)$ is a property of both the data set, and the kind of clusters that one is interested in. The value of $f(\theta)$ is heuristically defined as follows:

$$f(\theta) = \frac{1-\theta}{1+\theta}. \tag{7.5}$$

Therefore, by substituting the expected number of cross-links in Eq. 7.3, one obtains the following merging criterion $V(\mathcal{C}_i, \mathcal{C}_j)$:

$$V(\mathcal{C}_i, \mathcal{C}_j) = \frac{GroupLink(\mathcal{C}_i, \mathcal{C}_j)}{(q_i + q_j)^{1+2 \cdot f(\theta)} - q_i^{1+2 \cdot f(\theta)} - q_j^{1+2 \cdot f(\theta)}}. \tag{7.6}$$

The denominator explicitly normalizes for the sizes of the clusters being merged by penalizing larger clusters. The goal of this kind of normalization is to prevent the imbalanced preference toward successively merging only large clusters.

The merges are successively performed until a total of k clusters remain in the data. Because the agglomerative approach is applied only to a sample of the data, it remains to assign the remaining data points to one of the clusters. This can be achieved by assigning each disk-resident data point to the cluster with which it has the greatest similarity. This similarity is computed using the same quality criterion in Eq. 7.6 as was used for cluster–cluster merges. In this case, similarity is computed between clusters and individual data points by treating each data point as a singleton cluster.

7.2.3 Probabilistic Algorithms

The probabilistic approach to data clustering is introduced in Sect. 6.5 of Chap. 6. Generative models can be generalized to virtually any data type as long as an appropriate generating probability distribution can be defined for each mixture component. This provides unprecedented flexibility in adapting probabilistic clustering algorithms to various data types. After the mixture distribution model has been defined, the E- and M-steps need to be defined for the corresponding expectation–maximization (EM) approach. The main difference from numeric clustering is that the soft assignment process in the E-step, and the parameter estimation process in the M-step will depend on the relevant probability distribution model for the corresponding data type.

Let the k components of the mixture be denoted by $\mathcal{G}_1 \ldots \mathcal{G}_k$. Then, the generative process for each point in the data set $\mathcal{D}$ uses the following two steps:

1. Select a mixture component with prior probability α_i, where $i \in \{1 \ldots k\}$.
2. If the mth component of the mixture was selected in the first step, then generate a data point from $\mathcal{G}_m$.

The values of α_i denote the prior probabilities $P(\mathcal{G}_i)$, which need to be estimated along with other model parameters in a data-driven manner. The main difference from the numerical case is in the mathematical form of the generative model for the mth cluster (or mixture component) $\mathcal{G}_m$, which is now a discrete probability distribution rather than the probability density function used in the numeric case. This difference reflects the corresponding difference in data type. One reasonable choice for the discrete probability distribution of $\mathcal{G}_m$ is to assume that the jth categorical value of ith attribute is independently generated by mixture component (cluster) m with probability p_{ijm}. Consider a data point $\overline{X}$ containing the attribute value indices $j_1 \ldots j_d$ for its d dimensions. In other words, the rth attribute takes on the j_rth possible categorical value. For convenience, the entire set of model parameters is denoted by the generic notation Θ. Then, the discrete probability distribution $g^{m,\Theta}(\overline{X})$ from cluster m is given by the following expression:

$$g^{m,\Theta}(\overline{X}) = \prod_{r=1}^{d} p_{rj_r m}. \tag{7.7}$$

The discrete probability distribution is $g^{m,\Theta}(\cdot)$, which is analogous to the continuous density function $f^{m,\Theta}(\cdot)$ of the EM model in the previous chapter. Correspondingly, the *posterior* probability $P(\mathcal{G}_m|\overline{X}, \Theta)$ of the component $\mathcal{G}_m$ having generated *observed* data point $\overline{X}$ may be estimated as follows:

$$P(\mathcal{G}_m|\overline{X_j}, \Theta) = \frac{\alpha_m \cdot g^{m,\Theta}(\overline{X})}{\sum_{r=1}^{k} \alpha_r \cdot g^{r,\Theta}(\overline{X})}. \tag{7.8}$$

This defines the E-step for categorical data, and it provides a soft assignment probability of the data point to a cluster.

After the soft assignment probability has been determined, the M-step applies maximum likelihood estimation to the *individual components* of the mixture to estimate the probability p_{ijm}. While estimating the parameters for cluster m, the *weight* of a record is assumed to be equal to its assignment probability $P(\mathcal{G}_m|\overline{X}, \Theta)$ to cluster m. For each cluster m, the *weighted* number w_{ijm} of data points for which attribute i takes on its jth possible categorical value is estimated. This is equal to the sum of the assignment probabilities (to

cluster m) of data points that *do take on the jth value*. By dividing this value with the aggregate assignment probability of *all* data points to cluster m, the probability p_{ijm} may be estimated as follows:

$$p_{ijm} = \frac{w_{ijm}}{\sum_{\overline{X} \in \mathcal{D}} P(\mathcal{G}_m | \overline{X}, \Theta)}. \tag{7.9}$$

The parameter α_m is estimated as the average assignment probabilities of data points to cluster m. The aforementioned formulas for estimation may be derived from maximum likelihood estimation methods. Refer to the bibliographic notes for detailed derivations.

Sometimes, the estimation of Eq. 7.9 can be inaccurate because the available data may be limited, or particular values of categorical attributes may be rare. In such cases, some of the attribute values may not appear in a cluster (or $w_{ijm} \approx 0$). This can lead to poor parameter estimation, or *overfitting*. The *Laplacian smoothing* method is commonly used to address such ill-conditioned probabilities. This is achieved by adding a small positive value β to the estimated values of w_{ijm}, where β is the smoothing parameter. This will generally lead to more robust estimation. This type of smoothing is also applied in the estimation of the prior probabilities α_m when the data sets are very small. This completes the description of the M-step. As in the case of numerical data, the E-step and M-step are iterated to convergence.

7.2.4 Graph-Based Algorithms

Because graph-based methods are *meta-algorithms*, the broad description of these algorithms remains virtually the same for categorical data as for numeric data. Therefore, the approach described in Sect. 6.7 of the previous chapter applies to this case as well. The only difference is in terms of how the edges and values on the similarity graph are constructed. The first step is the determination of the k-nearest neighbors of each data record, and subsequent assignment of similarity values to edges. Any of the similarity functions described in Sect. 3.2.2 of Chap. 3 can be used to compute similarity values along the edges of the graph. These similarity measures could include the inverse occurrence frequency measure discussed in Chap. 3, which corrects for the natural skew in the different attribute values. As discussed in the previous chapter, one of the major advantages of graph-based algorithms is that they can be leveraged for virtually any kind of data type as long as a similarity function can be defined on that data type.

7.3 Scalable Data Clustering

In many applications, the size of the data is very large. Typically, the data cannot be stored in main memory, but it need to reside on disk. This is a significant challenge, because it imposes a constraint on the algorithmic design of clustering algorithms. This section will discuss the *CLARANS*, *BIRCH*, and *CURE* algorithms. These algorithms are all scalable implementations of one of the basic types of clustering algorithms discussed in the previous chapter. For example, the *CLARANS* approach is a scalable implementation of the k-medoids algorithm for clustering. The *BIRCH* algorithm is a top-down hierarchical generalization of the k-means algorithm. The *CURE* algorithm is a bottom-up agglomerative approach to clustering. These different algorithms inherit the advantages and disadvantages of the base classes of algorithms that they are generalized from. For example, while the *CLARANS* algorithm has the advantage of being more easily generalizable to different data types (beyond numeric data), it inherits the relatively high computational complexity

of k-medoids methods. The *BIRCH* algorithm is much faster because it is based on the k-means methodology, and its hierarchical clustering structure can be tightly controlled because of its top-down partitioning approach. This can be useful for indexing applications. On the other hand, *BIRCH* is not designed for arbitrary data types or clusters of arbitrary shape. The *CURE* algorithm can determine clusters of arbitrary shape because of its bottom-up hierarchical approach. The choice of the most suitable algorithm depends on the application at hand. This section will provide an overview of these different methods.

7.3.1 CLARANS

The *CLARA* and *CLARANS* methods are two generalizations of the k-medoids approach to clustering. Readers are referred to Sect. 6.3.4 of the previous chapter for a description of the generic k-medoids approach. Recall that the k-medoids approach works with a set of representatives, and iteratively exchanges one of the medoids with a non-medoid in each iteration to improve the clustering quality. The generic k-medoids algorithm allows considerable flexibility in deciding how this exchange might be executed.

The *Clustering LARge Applications (CLARA)* method is based on a particular instantiation of the k-medoids method known as *Partitioning Around Medoids (PAM)*. In this method, to exchange a medoid with another non-medoid representative, all possible $k \cdot (n-k)$ pairs are tried for a possible exchange to improve the clustering objective function. The best improvement of these pairs is selected for an exchange. This exchange is performed until the algorithm converges to a locally optimal value. The exchange process requires $O(k \cdot n^2)$ distance computations. Therefore, each iteration requires $O(k \cdot n^2 \cdot d)$ time for a d-dimensional data set, which can be rather expensive. Because the complexity is largely dependent on the number of data points, it can be reduced by applying the algorithm to a smaller sample. Therefore, the *CLARA* approach applies *PAM* to a smaller sampled data set of size $f \cdot n$ to discover the medoids. The value of f is a sampling fraction, which is much smaller than 1. The remaining nonsampled data points are assigned to the optimal medoids discovered by applying *PAM* to the smaller sample. This overall approach is applied repeatedly over independently chosen samples of data points of the same size $f \cdot n$. The best clustering over these independently chosen samples is selected as the optimal solution. Because the complexity of each iteration is $O(k \cdot f^2 \cdot n^2 \cdot d + k \cdot (n-k))$, the approach may be orders of magnitude faster for small values of the sampling fraction f. The main problem with *CLARA* occurs when each of the preselected samples does not include good choices of medoids.

The *Clustering Large Applications based on RANdomized Search (CLARANS)* approach works with the full data set for the clustering in order to avoid the problem with preselected samples. The approach iteratively attempts exchanges between random medoids with random non-medoids. After a randomly chosen non-medoid is tried for an exchange with a randomly chosen medoid, it is checked if the quality improves. If the quality does improve, then this exchange is made final. Otherwise, the number of unsuccessful exchange attempts is counted. A local optimal solution is said to have been found when a user-specified number of unsuccessful attempts *MaxAttempt* have been reached. This entire process of finding the local optimum is repeated for a user-specified number of iterations, denoted by *MaxLocal*. The clustering objective function of each of these *MaxLocal* locally optimal solutions is evaluated. The best among these local optima is selected as the optimal solution. The advantage of *CLARANS* over *CLARA* is that a greater diversity of the search space is explored.

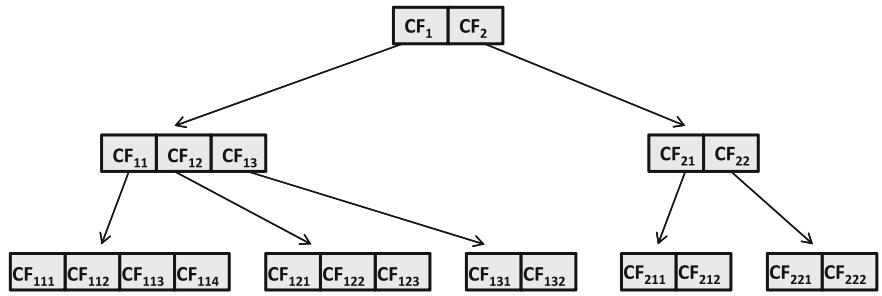


Figure 7.1: The CF-Tree

7.3.2 BIRCH

The *Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)* approach can be viewed as a combination of top-down hierarchical and k-means clustering. To achieve this goal, the approach introduces a hierarchical data structure, known as the CF-Tree. This is a height-balanced data structure organizing the clusters hierarchically. Each node has a branching factor of at most B, which corresponds to its (at most) B children subclusters. This structure shares a resemblance to the B-Tree data structure commonly used in database indexing. This is by design because the CF-Tree is inherently designed to support dynamic insertions into the hierarchical clustering structure. An example of the CF-Tree is illustrated in Fig. 7.1.

Each node contains a concise summary of each of the at most B subclusters that it points to. This concise summary of a cluster is referred to as its *cluster feature (CF)*, or *cluster feature vector*. The summary contains the triple $(\overline{SS}, \overline{LS}, m)$, where $\overline{SS}$ is a vector[1] containing the sum of the square of the points in the cluster (second-order moment), $\overline{LS}$ is a vector containing the linear sum of the points in the cluster (first-order moment), and m is the number of points in the cluster (zeroth-order moment). Thus, the size of the summary is $(2 \cdot d + 1)$ for a d-dimensional data set and is also referred to as a CF-vector. The cluster feature vector thus contains all moments of order at most 2. This summary has two very important properties:

1. Each cluster feature can be represented as a linear sum of the cluster features of the individual data points. Furthermore, the cluster feature of a parent node in the CF-Tree is the sum of the cluster features of its children. The cluster feature of a merged cluster can also be computed as the sum of the cluster features of the constituent clusters. Therefore, incremental updates of the cluster feature vector can be *efficiently* achieved by adding the cluster feature vector of a data point to that of the cluster.

2. The cluster features can be used to compute useful properties of a cluster, such as its radius and centroid. Note that these are the only two computations required by a centroid-based algorithm, such as k-means or *BIRCH*. These computations are discussed below.

To understand how the cluster feature can be used to measure the radius of a cluster, consider a set of data points denoted by $\overline{X_1} \ldots \overline{X_m}$, where $\overline{X_i} = (x_i^1 \ldots x_i^d)$. The mean and

[1]It is possible to store the sum of the values in $\overline{SS}$ across the d dimensions in lieu of $\overline{SS}$, without affecting the usability of the cluster feature. This would result in a cluster feature of size $(d+2)$ instead of $(2 \cdot d+1)$.

variance of any set of points can be expressed in terms of the their first and second moments. It is easy to see that the centroid (vector) of the cluster is simply $\overline{LS}/m$. The variance of a random variable Z is defined to be $E[Z^2] - E[Z]^2$, where $E[\cdot]$ denotes expected values. Therefore, the variances along the ith dimension can be expressed as $SS_i/m - (LS_i/m)^2$. Here SS_i and LS_i represent the component of the corresponding moment vector along the ith dimension. The sum of these dimension-specific variances yields the variance of the entire cluster. Furthermore, the distance of any point to the centroid can be computed using the cluster feature by using the computed centroid $\overline{LS}/m$. Therefore, the cluster feature vector contains all the information needed to insert a data point into the CF-Tree.

Each leaf node in the CF-Tree has a diameter threshold T. The diameter[2] can be any spread measure of the cluster such as its radius or variance, as long as it can be computed directly from the cluster feature vector. The value of T regulates the granularity of the clustering, the height of the tree, and the aggregate number of clusters at the leaf nodes. Lower values of T will result in a larger number of fine-grained clusters. Because the CF-Tree is always assumed to be main-memory resident, the size of the data set will typically have a critical impact on the value of T. Smaller data sets will allow the use of a small threshold T, whereas a larger data set will require a larger value of the threshold T. Therefore, an incremental approach such as *BIRCH* gradually increases the value of T to balance the greater need for memory with increasing data size. In other words, the value of T may need to be increased whenever the tree can no longer be kept within main-memory availability.

The incremental insertion of a data point into the tree is performed with a top-down approach. Specifically, the closest centroid is selected at each level for insertion into the tree structure. This approach is similar to the insertion process in a traditional database index such as a B-Tree. The cluster feature vectors are updated along the corresponding path of the tree by simple addition. At the leaf node, the data point is inserted into its closest cluster only if the insertion does not increase the cluster diameter beyond the threshold T. Otherwise, a new cluster must be created containing only that data point. This new cluster is added to the leaf node, if it is not already full. If the leaf node is already full, then it needs to be split into two nodes. Therefore, the cluster feature entries in the old leaf node need to be assigned to one of the two new nodes. The two cluster features in the leaf node, whose centroids are the furthest apart, can serve as the seeds for the split. The remaining entries are assigned to the seed node to which they are closest. As a result, the branching factor of the parent node of the leaf increases by 1. Therefore, the split might result in the branching factor of the parent increasing beyond B. If this is the case, then the parent would need to be split as well in a similar way. Thus, the split may be propagated upward until the branching factors of all nodes are below B. If the split propagates all the way to the root node, then the height of the CF-Tree increases by 1.

These repeated splits may sometimes result in the tree running out of main memory. In such cases, the CF-Tree needs to be rebuilt by increasing the threshold T, and reinserting the old leaf nodes into a new tree with a higher threshold T. Typically, this reinsertion will result in the merging of some older clusters into larger clusters that meet the new modified threshold T. Therefore, the memory requirement of the new tree is lower. Because the old leaf nodes are reinserted with the use of cluster feature vectors, this step can be accomplished without reading the original database from disk. Note that the cluster feature

[2]The original *BIRCH* algorithm proposes to use the *pairwise* root mean square (RMS) distance between cluster data points as the diameter. This is one possible measure of the intracluster distance. This value can also be shown to be computable from the CF vector as $\sqrt{\frac{\sum_{i=1}^{d}(2 \cdot m \cdot SS_i - 2 \cdot LS_i^2)}{m \cdot (m-1)}}$.

vectors allow the computation of the diameters resulting from the merge of two clusters without using the original data points.

An optional cluster refinement phase can be used to group related clusters within the leaf nodes and remove small outlier clusters. This can be achieved with the use of an agglomerative hierarchical clustering algorithm. Many agglomerative merging criteria, such as the variance-based merging criterion (see Sect. 6.4.1 of Chap. 6), can be easily computed from the CF-vectors. Finally, an optional refinement step reassigns all data points to their closest center, as produced by the global clustering step. This requires an additional scan of the data. If desired, outliers can be removed during this phase.

The *BIRCH* algorithm is very fast because the basic approach (without refinement) requires only one scan over the data, and each insertion is an efficient operation, which resembles the insertion operation in a traditional index structure. It is also highly adaptive to the underlying main-memory requirements. However, it implicitly assumes a spherical shape of the underlying clusters.

7.3.3 CURE

The *Clustering Using REpresentatives (CURE)* algorithm is an agglomerative hierarchical algorithm. Recall from the discussion in Sect. 6.4.1 of Chap. 6 that single-linkage implementation of bottom-up hierarchical algorithms can discover clusters of arbitrary shape. As in all agglomerative methods, a current set of clusters is maintained, which are successively merged with one another, based on single-linkage distance between clusters. However, instead of directly computing distances between all pairs of points in the two clusters for agglomerative merging, the algorithm uses a set of representatives to achieve better efficiency. These representatives are carefully chosen to capture the shape of each of the current clusters, so that the ability of agglomerative methods to capture clusters of arbitrary shape is retained even with the use of a smaller number of representatives. The first representative is chosen to be a data point that is farthest from the center of the cluster, the second representative is farthest to the first, the third is chosen to be the one that has the largest distance to the closest of two representatives, and so on. In particular, the rth representative is a data point that has the largest distance to the closest of the current set of $(r-1)$ representatives. As a result, the representatives tend to be arranged along the contours of the cluster. Typically, a small number of representatives (such as ten) are chosen from each cluster. This farthest distance approach does have the unfortunate effect of favoring selection of outliers. After the representatives have been selected, they are shrunk toward the cluster center to reduce the impact of outliers. This shrinking is performed by replacing a representative with a new synthetic data point on the line segment L joining the representative to the cluster center. The distance between the synthetic representative and the original representative is a fraction $\alpha \in (0, 1)$ of the length of line segment L. Shrinking is particularly useful in single-linkage implementations of agglomerative clustering because of the sensitivity of such methods to noisy representatives at the fringes of a cluster. Such noisy representatives may chain together unrelated clusters. Note that if the representatives are shrunk too far ($\alpha \approx 1$), the approach will reduce to centroid-based merging, which is also known to work poorly (see Sect. 6.4.1 of Chap. 6).

The clusters are merged using an agglomerative bottom-up approach. To perform the merging, the minimum distance between any pair of representative data points is used. This is the *single-linkage approach* of Sect. 6.4.1 in Chap. 6, which is most well suited to discovering clusters of arbitrary shape. By using a smaller number of representative data points, the *CURE* algorithm is able to significantly reduce the complexity of the merging

criterion in agglomerative hierarchical algorithms. The merging can be performed until the number of remaining clusters is equal to k. The value of k is an input parameter specified by the user. *CURE* can handle outliers by periodically eliminating small clusters during the merging process. The idea here is that the clusters remain small because they contain mostly outliers.

To further improve the complexity, the *CURE* algorithm draws a random sample from the underlying data, and performs the clustering on this random sample. In a final phase of the algorithm, all the data points are assigned to one of the remaining clusters by choosing the cluster with the closest representative data point.

Larger sample sizes can be efficiently used with a partitioning trick. In this case, the sample is further divided into a set of p partitions. Each partition is hierarchically clustered until a desired number of clusters is reached, or some merging quality criterion is met. These intermediate clusters (across all partitions) are then reclustered together hierarchically to create the final set of k clusters from the sampled data. The final assignment phase is applied to the representatives of the resulting clusters. Therefore, the overall process may be described by the following steps:

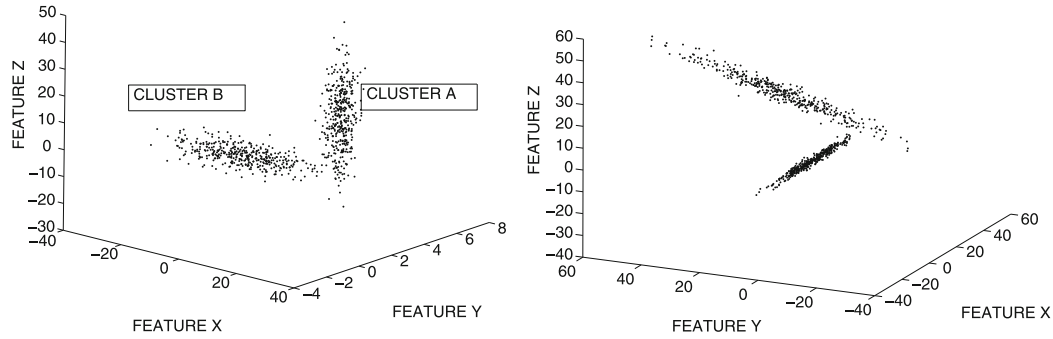
1. Sample s points from the database $\mathcal{D}$ of size n.
2. Divide the sample s into p partitions of size s/p each.
3. Cluster each partition independently using the hierarchical merging to k' clusters in each partition. The overall number $k' \cdot p$ of clusters across all partitions is still larger than the user-desired target k.
4. Perform hierarchical clustering over the $k' \cdot p$ clusters derived across all partitions to the user-desired target k.
5. Assign each of the $(n - s)$ nonsample data points to the cluster containing the closest representative.

The *CURE* algorithm is able to discover clusters of arbitrary shape unlike other scalable methods such as *BIRCH* and *CLARANS*. Experimental results have shown that *CURE* is also faster than these methods.

7.4 High-Dimensional Clustering

High-dimensional data contain many irrelevant features that cause noise in the clustering process. The feature selection section of the previous chapter discussed how the irrelevant features may be removed to improve the quality of clustering. When a large number of features are irrelevant, the data cannot be separated into meaningful and cohesive clusters. This scenario is especially likely to occur when features are uncorrelated with one another. In such cases, the distances between all pairs of data points become very similar. The corresponding phenomenon is referred to as the *concentration* of distances.

The feature selection methods discussed in the previous chapter can reduce the detrimental impact of the irrelevant features. However, it may often not be possible to remove any particular set of features *a priori* when the optimum choice of features depends on the underlying data locality. Consider the case illustrated in Fig. 7.2a. In this case, cluster A exists in the XY-plane, whereas cluster B exists in the YZ-plane. Therefore, the feature relevance is *local*, and it is no longer possible to remove any feature *globally* without losing



(a) Axis-parallel (b) Arbitrary subspaces

Figure 7.2: Illustration of axis-parallel and arbitrarily oriented (correlated) projected clusters

relevant features for some of the data localities. The concept of projected clustering was introduced to address this issue.

In conventional clustering methods, each cluster is a set of points. In *projected clustering*, each cluster is defined as a set of points *together with* a set of dimensions (or *subspace*). For example, the *projected* cluster A in Fig. 7.2a would be defined as its relevant set of points, *together* with the subspace corresponding to the X and Y dimensions. Similarly, the projected cluster B in Fig. 7.2a is defined as its relevant set of points, together with the subspace corresponding to the Y and Z dimensions. Therefore, a projected cluster is defined as the pair $(\mathcal{C}_i, \mathcal{E}_i)$, where $\mathcal{C}_i$ is a set of points, and the subspace $\mathcal{E}_i$ is the subspace defined by a set of dimensions.

An even more challenging situation is illustrated in Fig. 7.2b in which the clusters do not exist in axis-parallel subspaces, but they exist in arbitrarily oriented subspaces of the data. This problem is also a generalization of the principal component analysis (PCA) method discussed in Chap. 2, where a single global projection with the *largest* variance is found to retain the greatest information about the data. In this case, it is desired to retain the best local projections with the *least* variance to determine the subspaces in which each set of data points is tightly clustered. These types of clusters are referred to as *arbitrarily oriented projected clusters*, *generalized projected clusters*, or *correlation clusters*. Thus, the subspace $\mathcal{E}_i$ for each cluster $\mathcal{C}_i$ cannot be described in terms of the original set of dimensions. Furthermore, the orthogonal subspace to $\mathcal{E}_i$ provides the subspace for performing *local dimensionality reduction*. This is an interesting problem in its own right. Local dimensionality reduction provides enhanced reduction of data dimensionality because of the local selection of the subspaces for dimensionality reduction.

This problem has two different variations, which are referred to as *subspace clustering* and *projected clustering*, respectively.

1. *Subspace clustering:* In this case, overlaps are allowed among the points drawn from the different clusters. This problem is much closer to pattern mining, wherein the association patterns are mined from the numeric data after discretization. Each pattern therefore corresponds to a hypercube within a subspace of the numeric data, and the data points within this cube represent the subspace cluster. Typically, the number of subspace clusters mined can be very large, depending upon a user-defined parameter, known as the density threshold.

Algorithm *CLIQUE*(Data: $\mathcal{D}$, Ranges: p, Density: τ)
begin
Discretize each dimension of data set $\mathcal{D}$ into p ranges;
Determine dense combinations of grid cells at minimum support τ
using any frequent pattern mining algorithm;
Create graph in which dense grid combinations are
connected if they are adjacent;
Determine connected components of graph;
return (point set, subspace) pair for each connected component;
end

Figure 7.3: The *CLIQUE* algorithm

2. *Projected clustering:* In this case, no overlaps are allowed among the points drawn from the different clusters. This definition provides a concise summary of the data. Therefore, this model is much closer, in principle, to the original goals of the clustering framework of data summarization.

In this section, three different clustering algorithms will be described. The first of these is *CLIQUE*, which is a subspace clustering method. The other two are *PROCLUS* and *ORCLUS*, which are the first projected clustering methods proposed for the axis-parallel and the correlated versions of the problem, respectively.

7.4.1 CLIQUE

The *CLustering In QUEst (CLIQUE)* technique is a generalization of grid-based methods discussed in the previous chapter. The input to the method is the number of grid ranges p for each dimension, and the density τ. This density τ represents the minimum number of data points in a dense grid cell and can also be viewed as a minimum *support* requirement of the grid cell. As in all grid-based methods, the first phase of discretization is used to create a grid structure. In *full-dimensional* grid-based methods, the relevant dense regions are based on the intersection of the discretization ranges across *all* dimensions. The main difference of *CLIQUE* from these methods is that it is desired to determine the ranges only over a relevant *subset of* dimensions with density greater than τ. This is the same as the frequent pattern mining problem, where each discretized range is treated as an "item," and the support is set to τ. In the original *CLIQUE* algorithm, the *Apriori* method was used, though any other frequent pattern mining method could be used in principle. As in generic grid-based methods, the adjacent grid cells (defined on the same subspace) are put together. This process is also identical to the generic grid-based methods, except that two grids have to be defined on the same subspace for them to even be considered for adjacency. All the found patterns are returned together with the data points in them. The *CLIQUE* algorithm is illustrated in Fig. 7.3. An easily understandable description can also be generated for each set of k-dimensional connected grid regions by decomposing it into a *minimal* set of k-dimensional hypercubes. This problem is NP-hard. Refer to the bibliographic notes for efficient heuristics.

Strictly speaking, *CLIQUE* is a quantitative frequent pattern mining method rather than a clustering method. The output of *CLIQUE* can be very large and can sometimes be greater than the size of the data set, as is common in frequent pattern mining. Clustering

and frequent pattern mining are related but different problems with different objectives. The primary goal of frequent pattern mining is that of finding dimension correlation, whereas the primary goal of clustering is summarization. From this semantic point of view, the approach does not seem to achieve the primary application-specific goal of data summarization. The worst-case complexity of the approach and the number of discovered patterns can be exponentially related to the number of dimensions. The approach may not terminate at low values of the density (support) threshold τ.

7.4.2 PROCLUS

The *PROjected CLUStering (PROCLUS)* algorithm uses a medoid-based approach to clustering. The algorithm proceeds in three phases: an initialization phase, an iterative phase, and a cluster refinement phase. The initialization phase selects a small candidate set M of medoids, which restricts the search space for hill climbing. In other words, the final medoid set will be a subset of the candidate set M. The iterative phase uses a medoid-based technique for hill climbing to better solutions until convergence. The final refinement phase assigns data points to the optimal medoids and removes outliers.

A small candidate set M of medoids is selected during initialization as follows:

1. A random sample M of data points of size proportional to the number of clusters k is picked. Let the size of this subset be denoted by $A \cdot k$, where A is a constant greater than 1.

2. A greedy method is used to further reduce the size of the set M to $B \cdot k$, where $A > B > 1$. Specifically, a farthest distance approach is applied, where points are iteratively selected by selecting the data point with the farthest distance to the closest of the previously selected points.

Although the selection of a small candidate medoid set M greatly reduces the complexity of the search space, it also tends to include many outliers because of its farthest distance approach. Nevertheless, the farthest distance approach ensures well-separated seeds, which also tend to separate out the clusters well.

The algorithm starts by choosing a random subset S of k medoids from M, and it progressively improves the quality of medoids by iteratively replacing the "bad" medoids in the current set with new points from M. The best set of medoids found so far is always stored in S_{best}. Each medoid in S is associated with a set of dimensions based on the statistical distribution of data points in its locality. This set of dimensions represents the subspace specific to the corresponding cluster. The algorithm determines a set of "bad" medoids in S_{best}, using an approach described later. These bad medoids are replaced with randomly selected replacement points from M and the impact on the objective function is measured. If the objective function improves, then the current best set of medoids S_{best} is updated to S. Otherwise, another randomly selected replacement set is tried for exchanging with the bad medoids in S_{best} in the next iteration. If the medoids in S_{best} do not improve for a predefined number of successive replacement attempts, then the algorithm terminates. All computations, such as the assignment and objective function computation, are executed in the subspace associated with each medoid. The overall algorithm is illustrated in Fig. 7.4. Next, we provide a detailed description of each of the aforementioned steps.

Determining projected dimensions for a medoid: The aforementioned approach requires the determination of the quality of a particular set of medoids. This requires the assignment of

Algorithm *PROCLUS*(Database: $\mathcal{D}$, Clusters: k, Dimensions: l)
begin
Select candidate medoids $M \subseteq \mathcal{D}$ with a farthest distance approach;
S = Random subset of M of size k;
$BestObjective = \infty$;
repeat
Compute dimensions (subspace) associated with each medoid in S;
Assign points in $\mathcal{D}$ to closest medoids in S using projected distance;
$CurrentObjective$ = Mean projected distance of points to cluster centroids;
if ($CurrentObjective < BestObjective$) **then begin**
$S_{best} = S$;
$BestObjective = CurrentObjective$;
end;
Recompute S by replacing bad medoids in S_{best} with random points from M;
until termination criterion;
Assign data points to medoids in S_{best} using refined subspace computations;
return all cluster-subspace pairs;
end

Figure 7.4: The *PROCLUS* algorithm

data points to medoids by computing the distance of the data point to each medoid i in the subspace $\mathcal{E}_i$ relevant to the ith medoid. First, the *locality* of each medoid in S is defined. The locality of the medoid is defined as the set of data points that lies in a sphere of radius equal to the distance to the closest medoid. The (statistically normalized) average distance along each dimension from the medoid to the points in its locality is computed. Let r_{ij} be the average distance of the data points in the locality of medoid i to medoid i along dimension j. The mean $\mu_i = \sum_{j=1}^{d} r_{ij}/d$ and standard deviation $\sigma_i = \sqrt{\frac{\sum_{j=1}^{d}(r_{ij}-\mu_i)^2}{d-1}}$ of these distance values r_{ij} are computed, specific to each locality. This can then be converted into a statistically normalized value z_{ij}:

$$z_{ij} = \frac{r_{ij} - \mu_i}{\sigma_i}. \tag{7.10}$$

The reason for this locality-specific normalization is that different data localities have different natural sizes, and it is difficult to compare dimensions from different localities without normalization. Negative values of z_{ij} are particularly desirable because they suggest smaller average distances than expectation for a medoid-dimension pair. The basic idea is to select the smallest (most negative) $k \cdot l$ values of z_{ij} to determine the relevant cluster-specific dimensions. Note that this may result in the assignment of a different number of dimensions to the different clusters. The sum of the total number of dimensions associated with the different medoids must be equal to $k \cdot l$. An additional constraint is that the number of dimensions associated with a medoid must be at least 2. To achieve this, all the z_{ij} values are sorted in increasing order, and the two smallest ones are selected for each medoid i. Then, the remaining $k \cdot (l-2)$ medoid-dimension pairs are greedily selected as the smallest ones from the remaining values of z_{ij}.

Assignment of data points to clusters and cluster evaluation: Given the medoids and their associated sets of dimensions, the data points are assigned to the medoids using a single pass over the database. The distance of the data points to the medoids is computed using the Manhattan segmental distance. The *Manhattan segmental distance* is the same as the Manhattan distance, except that it is normalized for the varying number of dimensions associated with each medoid. To compute this distance, the Manhattan distance is computed using only the relevant set of dimensions, and then divided by the number of relevant dimensions. A data point is assigned to the medoid with which it has the least Manhattan segmental distance. After determining the clusters, the objective function of the clustering is evaluated as the average Manhattan segmental distance of data points to the *centroids* of their respective clusters. If the clustering objective improves, then S_{best} is updated.

Determination of bad medoids: The determination of "bad" medoids from S_{best} is performed as follows: The medoid of the cluster with the least number of points is bad. In addition, the medoid of any cluster with less than $(n/k) \cdot minDeviation$ points is bad, where $minDeviation$ is a constant smaller than 1. The typical value was set to 0.1. The assumption here is that bad medoids have small clusters either because they are outliers or because they share points with another cluster. The bad medoids are replaced with random points from the candidate medoid set M.

Refinement phase: After the best set of medoids is found, a final pass is performed over the data to improve the quality of the clustering. The dimensions associated with each medoid are computed differently than in the iterative phase. The main difference is that to analyze the dimensions associated with each medoid, the distribution of the points in the clusters at the end of the iterative phase is used, as opposed to the localities of the medoids. After the new dimensions have been computed, the points are reassigned to medoids based on the Manhattan segmental distance with respect to the new set of dimensions. Outliers are also handled during this final pass over the data. For each medoid i, its closest other medoid is computed using the Manhattan segmental distance in the relevant subspace of medoid i. The corresponding distance is referred to as its sphere of influence. If the Manhattan segmental distance of a data point to each medoid is greater than the latter's sphere of influence, then the data point is discarded as an outlier.

7.4.3 ORCLUS

The *arbitrarily ORiented projected CLUStering (ORCLUS)* algorithm finds clusters in arbitrarily oriented subspaces, as illustrated in Fig. 7.2b. Clearly, such clusters cannot be found by axis-parallel projected clustering. Such clusters are also referred to as *correlation clusters*. The algorithm uses the number of clusters k, and the dimensionality l of each subspace $\mathcal{E}_i$ as an input parameter. Therefore, the algorithm returns k different pairs $(\mathcal{C}_i, \mathcal{E}_i)$, where the cluster$\mathcal{C}_i$ is defined in the arbitrarily oriented subspace $\mathcal{E}_i$. In addition, the algorithm reports a set of outliers $\mathcal{O}$. This method is also referred to as *correlation clustering*. Another difference between the *PROCLUS* and *ORCLUS* models is the simplifying assumption in the latter that the dimensionality of each subspace is fixed to the same value l. In the former case, the value of l is simply the *average* dimensionality of the cluster-specific subspaces.

The *ORCLUS* algorithm uses a combination of hierarchical and k-means clustering in conjunction with subspace refinement. While hierarchical merging algorithms are generally more effective, they are expensive. Therefore, the algorithm uses hierarchical *representatives* that are successively merged. The algorithm starts with $k_c = k_0$ initial seeds, denoted by S.

Algorithm *ORCLUS*(Data: $\mathcal{D}$, Clusters: k, Dimensions: l)
begin
  Sample set S of $k_0 > k$ points from $\mathcal{D}$;
  $k_c = k_0$; $l_c = d$;
  Set each $\mathcal{E}_i$ to the full data dimensionality;
  $\alpha = 0.5$; $\beta = e^{-\log(d/l) \cdot \log(1/\alpha)/\log(k_0/k)}$;
  while $(k_c > k)$ **do**
  begin
    Assign each data point in $\mathcal{D}$ to closest seed in S using
      projected distance in $\mathcal{E}_i$ to create $\mathcal{C}_i$;
    Re-center each seed in S to centroid of cluster $\mathcal{C}_i$;
    Use PCA to determine subspace $\mathcal{E}_i$ associated with $\mathcal{C}_i$ by selecting
      smallest l_c eigenvectors of covariance matrix of $\mathcal{C}_i$;
    $k_c = \max\{k, k_c \cdot \alpha\}$; $l_c = \max\{l, l_c \cdot \beta\}$;
    Repeatedly merge clusters to reduce number of clusters to
      the new reduced value of k_c;
  end;
  Perform final assignment pass of points to clusters;
  return cluster-subspace pairs $(\mathcal{C}_i, \mathcal{E}_i)$ for each $i \in \{1 \ldots k\}$;
end

Figure 7.5: The *ORCLUS* algorithm

The current number of seeds, k_c, are reduced over successive merging iterations. Methods from representative-based clustering are used to assign data points to these seeds, except that the distance of a data point to its seed is measured in its associated subspace $\mathcal{E}_i$. Initially, the current dimensionality, l_c, of each cluster is equal to the full data dimensionality. The value l_c is reduced gradually to the user-specified dimensionality l by successive reduction over different iterations. The idea behind this gradual reduction is that in the first few iterations, the clusters may not necessarily correspond very well to the natural lower dimensional subspace clusters in the data; so a larger subspace is retained to avoid loss of information. In later iterations, the clusters are more refined, and therefore subspaces of lower rank may be extracted.

The overall algorithm consists of a number of iterations, in each of which a sequence of merging operations is alternated with a k-means style assignment with projected distances. The number of current clusters is reduced by the factor $\alpha < 1$, and the dimensionality of current cluster $\mathcal{C}_i$ is reduced by $\beta < 1$ in a given iteration. The first few iterations correspond to a higher dimensionality, and each successive iteration continues to peel off more and more noisy subspaces for the different clusters. The values of α and β are related in such a way that the reduction from k_0 to k clusters occurs in the same number of iterations as the reduction from $l_0 = d$ to l dimensions. The value of α is 0.5, and the derived value of β is indicated in Fig. 7.5. The overall description of the algorithm is also illustrated in this figure.

The overall procedure uses the three alternating steps of assignment, subspace recomputation, and merging in each iteration. Therefore, the algorithm uses concepts from both hierarchical and k-means methods in conjunction with subspace refinement. The assignment step assigns each data point to its closest seed, by comparing the projected distance

of a data point to the ith seed in S, using the subspace $\mathcal{E}_i$. After the assignment, all the seeds in S are re-centered to the centroids of the corresponding clusters. At this point, the subspace $\mathcal{E}_i$ of dimensionality l_c associated with each cluster $\mathcal{C}_i$ is computed. This is done by using PCA on cluster $\mathcal{C}_i$. The subspace $\mathcal{E}_i$ is defined by the l_c orthonormal eigenvectors of the covariance matrix of cluster $\mathcal{C}_i$ with the least eigenvalues. To perform the merging, the algorithm computes the projected energy (variance) of the union of the two clusters in the corresponding least spread subspace. The pair with the least energy is selected to perform the merging. Note that this is a subspace generalization of the variance criterion for hierarchical merging algorithms (see Sect. 6.4.1 of Chap. 6).

The algorithm terminates when the merging process over all the iterations has reduced the number of clusters to k. At this point, the dimensionality l_c of the subspace $\mathcal{E}_i$ associated with each cluster $\mathcal{C}_i$ is also equal to l. The algorithm performs one final pass over the database to assign data points to their closest seed based on the projected distance. Outliers are handled during the final phase. A data point is considered an outlier when its projected distance to the closest seed i is greater than the projected distance of other seeds to seed i in subspace $\mathcal{E}_i$.

A major computational challenge is that the merging technique requires the computation of the eigenvectors of the union of clusters, which can be expensive. To efficiently perform the merging, the *ORCLUS* approach extends the concept of cluster feature vectors from *BIRCH* to covariance matrices. The idea is to store not only the moments in the cluster feature vector but also the sum of the products of attribute values for each pair of dimensions. The covariance matrix can be computed from this extended cluster feature vector. This approach can be viewed as a covariance- and subspace-based generalization of the variance-based merging implementation of Sect. 6.4.1 in Chap. 6. For details on this optimization, the reader is referred to the bibliographic section.

Depending on the value of k_0 chosen, the time complexity is dominated by either the merges or the assignments. The merges require eigenvector computation, which can be expensive. With an efficient implementation based on cluster feature vectors, the merges can be implemented in $O(k_0^2 \cdot d \cdot (k_0 + d^2))$ time, whereas the assignment step always requires $O(k^0 \cdot n \cdot d)$ time. This can be made faster with the use of optimized eigenvector computations. For smaller values of k_0, the computational complexity of the method is closer to k-means, whereas for larger values of k_0, the complexity is closer to hierarchical methods. The *ORCLUS* algorithm does not assume the existence of an incrementally updatable similarity matrix, as is common with bottom-up hierarchical methods. At the expense of additional space, the maintenance of such a similarity matrix can reduce the $O(k_0^3 \cdot d)$ term to $O(k_0^2 \cdot \log(k_0) \cdot d)$.

7.5 Semisupervised Clustering

One of the challenges with clustering is that a wide variety of alternative solutions may be found by various algorithms. The quality of these alternative clusterings may be ranked differently by different internal validation criteria depending on the alignment between the clustering criterion and validation criterion. This is a major problem with any unsupervised algorithm. Semisupervision, therefore, relies on external *application-specific* criteria to guide the clustering process.

It is important to understand that different clusterings may not be equally useful from an application-specific perspective. The utility of a clustering result is, after all, based on the ability to use it effectively for a given application. One way of guiding the clustering results

toward an application-specific goal is with the use of *supervision*. For example, consider the case where an analyst wishes to segment a set of documents approximately along the lines of the *Open Directory Project (ODP)*,[3] where users have already manually labeled documents into a set of predefined categories. One may want to use this directory only as soft guiding principle because the number of clusters and their topics in the analyst's collection may not always be exactly the same as in the *ODP* clusters. One way of incorporating supervision is to download example documents from each category of *ODP* and mix them with the documents that need to be clustered. This newly downloaded set of documents are labeled with their category and provide information about how the features are related to the different clusters (categories). The added set of labeled documents, therefore, provides *supervision* to the clustering process in the same way that a teacher guides his or her students toward a specific goal.

A different scenario is one in which it is known from background knowledge that certain documents should belong to the same class, and others should not. Correspondingly, two types of semisupervision are commonly used in clustering:

1. *Pointwise supervision:* Labels are associated with individual data points and provide information about the category (or cluster) of the object. This version of the problem is closely related to that of data classification.

2. *Pairwise supervision:* "Must-link" and "cannot-link" constraints are provided for the individual data points. This provides information about cases where pairs of objects are allowed to be in the same cluster or are forbidden to be in the same cluster, respectively. This form of supervision is also sometimes referred to as *constrained clustering*.

For each of these variations, a number of simple semisupervised clustering methods are described in the following sections.

7.5.1 Pointwise Supervision

Pointwise supervision is significantly easier to address than pairwise supervision because the labels associated with the data points can be used more naturally in conjunction with existing clustering algorithms. In *soft supervision*, the labels are used as guidance, but data points with different labels are allowed to mix. In *hard* supervision, data points with different labels are not allowed to mix. Some examples of different ways of modifying existing clustering algorithms are as follows:

1. *Semisupervised clustering by seeding:* In this case, the initial seeds for a k-means algorithm are chosen as data points of different labels. These are used to execute a standard k-means algorithm. The biased initialization has a significant impact on the final results, even when labeled data points are allowed to be assigned to a cluster whose initial seed had a different label (soft supervision). In hard supervision, clusters are explicitly associated with labels corresponding to their initial seeds. The assignment of *labeled* data points is constrained so that such points can be assigned to a cluster with the same label. In some cases, the weights of the unlabeled points are discounted while computing cluster centers to increase the impact of supervision. The second form of semisupervision is closely related to semisupervised classification, which is

[3] `http://www.dmoz.org/`.

discussed in Chap. 11. An EM algorithm, which performs semisupervised classification with labeled and unlabeled data, uses a similar approach. Refer to Sect. 11.6 of Chap. 11 for a discussion of this algorithm. For more robust initialization, an unsupervised clustering can be separately applied to each labeled data segment to create the seeds.

2. *EM algorithms:* Because the EM algorithm is a soft version of the k-means method, the changes required to EM methods are exactly identical to those in the case of k-means. The initialization of the EM algorithm is performed with mixtures centered at the labeled data points. In addition, for hard supervision, the posterior probabilities of labeled data points are always set to 0 for mixture components that do not belong to the same label. Furthermore, the unlabeled data points are discounted during computation of model parameters. This approach is discussed in detail in Sect. 11.6 of Chap. 11.

3. *Agglomerative algorithms:* Agglomerative algorithms can be generalized easily to the semisupervised case. In cases where the merging allows the mixing of different labels (soft supervision), the distance function between clusters during the clustering can incorporate the similarity in their class label distributions across the two components being merged by providing an extra credit to clusters with the same label. The amount of this credit regulates the level of supervision. Many different choices are also available to incorporate the supervision more strongly in the merging criterion. For example, the merging criterion may require that only clusters containing the same label are merged together (hard supervision).

4. *Graph-based algorithms:* Graph-based algorithms can be modified to work in the semisupervised scenario by changing the similarity graph to incorporate supervision. The edges joining data points with the same label have an extra weight of α. The value of α regulates the level of supervision. Increased values of α will be closer to hard supervision, whereas smaller values of α will be closer to soft supervision. All other steps in the clustering algorithm remain identical. A different form of graph-based supervision, known as collective classification, is used for the semisupervised classification problem (cf. Sect. 19.4 of Chap. 19).

Thus, pointwise supervision is easily incorporated in most clustering algorithms.

7.5.2 Pairwise Supervision

In pairwise supervision, "must-link" and "cannot-link" constraints are specified between pairs of objects. An immediate observation is that it is not necessary for a feasible and consistent solution to exist for an arbitrary set of constraints. Consider the case where three data points A, B, and C are such that (A, B), and (A, C) are both "must-link" pairs, whereas (B, C) is a "cannot-link" pair. It is evident that no feasible clustering can be found that satisfies all three constraints. The problem of finding clusters with pairwise constraints is generally more difficult than one in which pointwise constraints are specified. In cases where only "must-link" constraints are specified, the problem can be approximately reduced to the case of pointwise supervision.

The k-means algorithm can be modified to handle pairwise supervision quite easily. The basic idea is to start with an initial set of randomly chosen centroids. The data points are processed in a random order for assignment to the seeds. Each data point is assigned to

its closest seed that does not violate any of the constraints implied by the assignments that have already been executed. In the event that the data point cannot be assigned to any cluster in a consistent way, the algorithm terminates. In this case, the clusters in the last iteration where a feasible solution was found are reported. In some cases, no feasible solution may be found even in the first iteration, depending on the choice of seeds. Therefore, the constrained k-means approach may be executed multiple times, and the best solution over these executions is reported. Numerous other methods are available in the literature, both in terms of the kinds of constraints that are specified, and in terms of the solution methodology. The bibliographic notes contain pointers to many of these methods.

7.6 Human and Visually Supervised Clustering

The previous section discussed ways of incorporating supervision in the *input data* in the form of constraints or labels. A different way of incorporating supervision is to use direct feedback from the user *during the clustering process*, based on an understandable summary of the clusters.

The core idea is that semantically meaningful clusters are often difficult to isolate using fully automated methods in which rigid mathematical formalizations are used as the only criteria. The utility of clusters is based on their application-specific usability, which is often semantically interpretable. In such cases, human involvement is necessary to incorporate the intuitive and semantically meaningful aspects during the cluster discovery process. Clustering is a problem that requires both the computational power of a computer and the intuitive understanding of a human. Therefore, a natural solution is to divide the clustering task in such a way that each entity performs the task that it is most well suited to. In the interactive approach, the computer performs the computationally intensive analysis, which is leveraged to provide the user with an intuitively understandable summary of the clustering structure. The user utilizes this summary to provide feedback about the key choices that should be made by a clustering algorithm. The result of this cooperative technique is a system that can perform the task of clustering better than either a human or a computer.

There are two natural ways of providing feedback during the clustering process:

1. *Semantic feedback as an intermediate process in standard clustering algorithms:* Such an approach is relevant in domains where the objects are semantically interpretable (e.g., documents or images), and the user provides feedback at specific stages in a clustering algorithm when critical choices are made. For example, in a k-means algorithm, a user may choose to drop some clusters during each iteration and manually specify new seeds reflecting uncovered segments of the data.

2. *Visual feedback in algorithms specifically designed for human–computer interaction:* In many high-dimensional data sets, the number of attributes is very large, and it is difficult to associate direct semantic interpretability with the objects. In such cases, the user must be provided visual representations of the clustering structure of the data in different subsets of attributes. The user may leverage these representatives to provide feedback to the clustering process. This approach can be viewed as an interactive version of projected clustering methods.

In the following section, each of these different types of algorithms will be addressed in detail.

7.6.1 Modifications of Existing Clustering Algorithms

Most clustering algorithms use a number of key decision steps in which choices need to be made, such as the choice of merges in a hierarchical clustering algorithm, or the resolution of close ties in assignment of data points to clusters. When these choices are made on the basis of stringent and predefined clustering criteria, the resulting clusters may not reflect the natural structure of clusters in the data. Therefore, the goal in this kind of approach is to present the user with a small number of *alternatives* corresponding to critical choices in the clustering process. Some examples of simple modifications of the existing clustering algorithms are as follows:

1. *Modifications to k-means and related methods:* A number of critical decision points in the k-means algorithm can be utilized to improve the clustering process. For example, after each iteration, representative data points from each cluster can be presented to the user. The user may choose to manually discard either clusters with very few data points or clusters that are closely related to others. The corresponding seeds may be dropped and replaced with randomly chosen seeds in each iteration. Such an approach works well when the representative data points presented to the user have clear semantic interpretability. This is true in many domains such as image data or document data.

2. *Modifications to hierarchical methods:* In the bottom-up hierarchical algorithms, the clusters are successively merged by selecting the closest pair for merging. The key here is that if a bottom-up algorithm makes an error in the merging process, the merging decision is final, resulting in a lower quality clustering. Therefore, one way of reducing such mistakes is to present the users with the top-ranking choices for the merge corresponding to a small number of different pairs of clusters. These choices can be made by the user on the basis of semantic interpretability.

It is important to point out that the key steps at which a user may provide the feedback depend on the level of semantic interpretability of the objects in the underlying clusters. In some cases, such semantic interpretability may not be available.

7.6.2 Visual Clustering

Visual clustering is particularly helpful in scenarios, such as high-dimensional data, where the semantic interpretability of the individual objects is low. In such cases, it is useful to visualize lower dimensional projections of the data to determine subspaces in which the data are clustered. The ability to discover such lower dimensional projections is based on a combination of the computational ability of a computer and the intuitive feedback of the user. *IPCLUS* is an approach that combines interactive projected clustering methods with visualization methods derived from density-based methods.

One challenge with high-dimensional clustering is that the density, distribution, and shapes of the clusters may be quite different in different data localities and subspaces. Furthermore, it may not be easy to decide the optimum density threshold at which to separate out the clusters in any particular subspace. This is a problem even for full-dimensional clustering algorithms where any particular density threshold[4] may either merge clusters or completely miss clusters. While subspace clustering methods such as *CLIQUE* address these issues by reporting a huge number of overlapping clusters, projected clustering methods such

[4]See discussion in Chap. 6 about Fig. 6.14.

Algorithm *IPCLUS*(Data Set: $\mathcal{D}$, Polarization Points: k)
begin
  while not(termination_criterion) **do**
  begin
    Randomly sample k points $\overline{Y_1} \ldots \overline{Y_k}$ from $\mathcal{D}$;
    Compute 2-dimensional subspace $\mathcal{E}$ polarized around $\overline{Y_1} \ldots \overline{Y_k}$;
    Generate density profile in $\mathcal{E}$ and present to user;
    Record membership statistics of clusters based on
        user-specified density-based feedback;
  end;
  return consensus clusters from membership statistics;
end

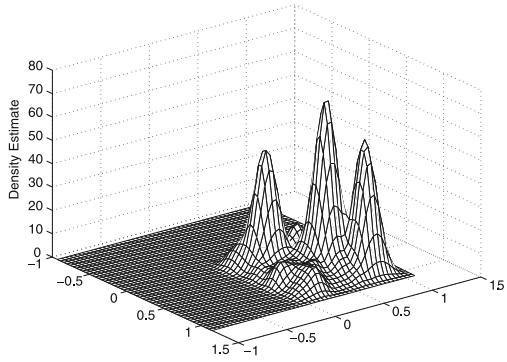
Figure 7.6: The *IPCLUS* algorithm

as *PROCLUS* address these issues by making hard decisions about how the data should be most appropriately summarized. Clearly, such decisions can be made more effectively by interactive user exploration of these alternative *views* and creating a final *consensus* from these different views. The advantage of involving the user is the greater intuition available in terms of the quality of feedback provided to the clustering process. The result of this cooperative technique is a system that can perform the clustering task better than either a human or a computer.

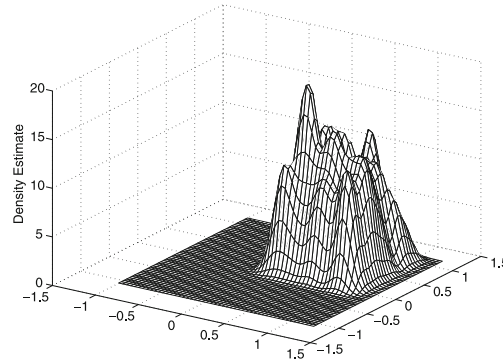
The idea behind the *Interactive Projected CLUStering algorithm (IPCLUS)* is to provide the user with a set of meaningful visualizations in lower dimensional projections together with the ability to decide how to separate the clusters. The overall algorithm is illustrated in Fig. 7.6. The interactive projected clustering algorithm works in a series of iterations; in each, a projection is determined in which there are distinct sets of points that can be clearly distinguished from one another. Such projections are referred to as *well polarized*. In a well-polarized projection, it is easier for the user to clearly distinguish a set of clusters from the rest of the data. Examples of the data density distribution of a well-polarized projection and a poorly polarized projection are illustrated in Fig. 7.7a and b, respectively.

These polarized projections are determined by randomly selecting a set of k records from the database that are referred to as the *polarization anchors*. The number of polarization anchors k is one of the inputs to the algorithm. A 2-dimensional subspace of the data is determined such that the data are clustered around each of these polarization anchors. Specifically, a 2-dimensional subspace is selected so that the mean square radius of assignments of data points to the polarization points as anchors is minimized. Different projections are repeatedly determined with different sampled anchors in which the user can provide feedback. A consensus clustering is then determined from the different clusterings generated by the user over multiple subspace views of the data.

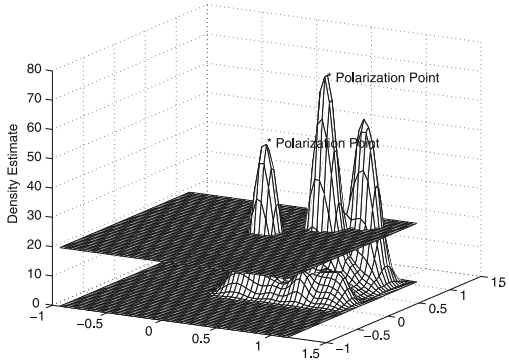
The polarization subspaces can be determined either in axis-parallel subspaces or arbitrary subspaces, although the former provides greater interpretability. The overall approach for determining polarization subspaces starts with the full dimensionality and iteratively reduces the dimensionality of the current subspace until a 2-dimensional subspace is obtained. This is achieved by iteratively assigning data points to their closest subspace-specific anchor points in each iteration, while discarding the most noisy (high variance) dimensions in each iteration about the polarization points. The dimensionality is reduced



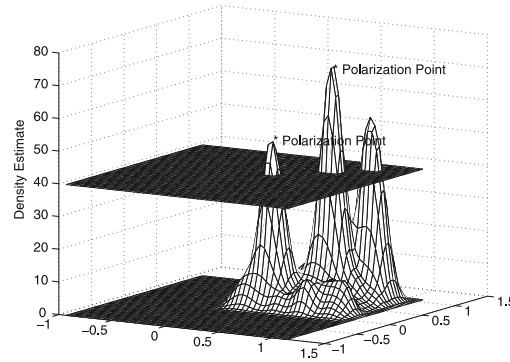
(a) Well-polarized projection



(b) Poorly polarized projection



(c) User separation (lower τ)



(d) User separation (higher τ)

Figure 7.7: Polarizations of different quality and flexibility of user interaction

by a constant factor of 2 in each iteration. Thus, this is a k-medoids type approach, which reduces the dimensionality of the subspaces for distance computation, but not the seeds in each iteration. This typically results in the discovery of a 2-dimensional subspace that is highly clustered around the polarization anchors. Of course, if the polarization anchors are poorly sampled, then this will result in poorly separated clusters. Nevertheless, repeated sampling of polarization points ensures that good subspaces will be selected in at least a few iterations.

After the projection subspace has been found, kernel density estimation techniques can be used to determine the data density at each point in a 2-dimensional grid of values in the relevant subspace. The density values at these grid points are used to create a surface plot. Examples of such plots are illustrated in Fig. 7.7. Because clusters correspond to dense regions in the data, they are represented by peaks in the density profile. To actually separate out the clusters, the user can visually specify density value thresholds that correspond to noise levels at which clusters can be separated from one another. Specifically, a cluster may be defined to be a connected region in the space with density above a certain noise threshold τ that is specified by the user. This cluster may be of arbitrary shape, and the points inside it can be determined. Note that when the density distribution varies significantly with locality, different numbers, shapes, and sizes of clusters will be discovered by different density thresholds. Examples of density thresholding are illustrated in Fig. 7.7c and 7.7d,

where clusters of different numbers and shapes are discovered at different thresholds. It is in this step that the user intuition is very helpful, both in terms of deciding which polarized projections are most relevant, and in terms of deciding what density thresholds to specify. If desired, the user may discard a projection altogether or specify multiple thresholds in the same projection to discover clusters of different density in different localities. The specification of the density threshold τ need not be done directly by value. The density separator hyperplane can be visually superposed on the density profile with the help of a graphical interface.

Each feedback of the user results in the generation of connected sets of points within the density contours. These sets of points can be viewed as one or more binary "transactions" drawn on the "item" space of data points. The key is to determine the consensus clusters from these newly created transactions that encode user feedback. While the problem of finding consensus clusters from multiple clusterings will be discussed in detail in the next section, a very simple way of doing this is to use either frequent pattern mining (to find overlapping clusters) or a second level of clustering on the transactions to generate nonoverlapping clusters. Because this new set of transactions encodes the user preferences, the quality of the clusters found with such an approach will typically be quite high.

7.7 Cluster Ensembles

The previous section illustrated how different views of the data can lead to different solutions to the clustering problem. This notion is closely related to the concept of *multiview clustering* or *ensemble clustering*, which studies this issue from a broader perspective. It is evident from the discussion in this chapter and the previous one that clustering is an unsupervised problem with many alternative solutions. In spite of the availability of a large number of validation criteria, the ability to truly test the quality of a clustering algorithm remains elusive. The goal in ensemble clustering is to combine the results of many clustering models to create a more robust clustering. The idea is that no single model or criterion truly captures the optimal clustering, but an ensemble of models will provide a more robust solution.

Most ensemble models use the following two steps to generate the clustering solution:

1. Generate k different clusterings with the use of different models or data selection mechanisms. These represent the different ensemble *components*.

2. Combine the different results into a single and more robust clustering.

The following section provides a brief overview of the different ways in which the alternative clusterings can be constructed.

7.7.1 Selecting Different Ensemble Components

The different ensemble components can be selected in a wide variety of ways. They can be either *modelbased* or *data-selection* based. In model-based ensembles, the different components of the ensemble reflect different models, such as the use of different clustering models, different settings of the same model, or different clusterings provided by different runs of the same randomized algorithm. Some examples follow:

1. The different components can be a variety of models such as partitioning methods, hierarchical methods, and density-based methods. The qualitative differences between the models will be data set-specific.

2. The different components can correspond to different settings of the same algorithm. An example is the use of different initializations for algorithms such as k-means or EM, the use of different mixture models for EM, or the use of different parameter settings of the same algorithm, such as the choice of the density threshold in *DBSCAN*. An ensemble approach is useful because the optimum choice of parameter settings is also hard to determine in an unsupervised problem such as clustering.

3. The different components could be obtained from a single algorithm. For example, a 2-means clustering applied to the 1-dimensional embedding obtained from spectral clustering will yield a different clustering solution for *each* eigenvector. Therefore, the smallest k nontrivial eigenvectors will provide k different solutions that are often quite different as a result of the orthogonality of the eigenvectors.

A second way of selecting the different components of the ensemble is with the use of *data selection*. Data selection can be performed in two different ways:

1. *Point selection:* Different subsets of the data are selected, either via random sampling, or by systematic selection for the clustering process.

2. *Dimension selection:* Different subsets of dimensions are selected to perform the clustering. An example is the *IPCLUS* method discussed in the previous section.

After the individual ensemble components have been constructed, it is often a challenge to combine the results from these different components to create a *consensus* clustering.

7.7.2 Combining Different Ensemble Components

After the different clustering solutions have been obtained, it is desired to create a robust consensus from the different solutions. In the following section, some simple methods are described that use the base clusterings as input for the generation of the final set of clusters.

7.7.2.1 Hypergraph Partitioning Algorithm

Each object in the data is represented by a vertex. A cluster in any of the ensemble components is represented as a *hyperedge*. A hyperedge is a generalization of the notion of edge, because it connects more than two nodes in the form of a complete clique. Any off-the-shelf hypergraph clustering algorithm such as *HMETIS* [302] can be used to determine the optimal partitioning. Constraints are added to ensure a balanced partitioning. One major challenge with hypergraph partitioning is that a hyperedge can be "broken" by a partitioning in many different ways, not all of which are qualitatively equivalent. Most hypergraph partitioning algorithms use a constant penalty for breaking a hyperedge. This can sometimes be undesirable from a qualitative perspective.

7.7.2.2 Meta-clustering Algorithm

This is also a graph-based approach, except that vertices are associated with each cluster in the ensemble components. For example, if there are $k_1 \ldots k_r$ different clusters in each of the r ensemble components, then a total of $\sum_{i=1}^{r} k_i$ vertices will be created. Each vertex therefore represents a set of data objects. An edge is added between a pair of vertices if the Jaccard coefficient between the corresponding object sets is nonzero. The weight of the edge is equal to the Jaccard coefficient. This is therefore an r-partite graph because there are no edges

between two vertices from the same ensemble component. A graph partitioning algorithm is applied to this graph to create the desired number of clusters. Each data point has r different instances corresponding to the different ensemble components. The distribution of the membership of different instances of the data point to the meta-partitions can be used to determine its meta-cluster membership, or soft assignment probability. Balancing constraints may be added to the meta-clustering phase to ensure that the resulting clusters are balanced.

7.8 Putting Clustering to Work: Applications

Clustering can be considered a specific type of data summarization where the summaries of the data points are constructed on the basis of similarity. Because summarization is a first step to many data mining applications, such summaries can be widely useful. This section will discuss the many applications of data clustering.

7.8.1 Applications to Other Data Mining Problems

Clustering is intimately related to other data mining problems and is used as a first summarization step in these cases. In particular, it is used quite often for the data mining problems of outlier analysis and classification. These specific applications are discussed below.

7.8.1.1 Data Summarization

Although many forms of data summarization, such as sampling, histograms, and wavelets are available for different kinds of data, clustering is the only natural form of summarization based on the notion of similarity. Because the notion of similarity is fundamental to many data mining applications, such summaries are very useful for similarity-based applications. Specific applications include recommendation analysis methods, such as collaborative filtering. This application is discussed later in this chapter, and in Chap. 18 on Web mining.

7.8.1.2 Outlier Analysis

Outliers are defined as data points that are generated by a different mechanism than the normal data points. This can be viewed as a complementary problem to clustering where the goal is to determine groups of closely related data points generated by the same mechanism. Therefore, outliers may be defined as data points that do not lie in any particular cluster. This is of course a simplistic abstraction but is nevertheless a powerful principle as a starting point. Sections 8.3 and 8.4 of Chap. 8 discuss how many algorithms for outlier analysis can be viewed as variations of clustering algorithms.

7.8.1.3 Classification

Many forms of clustering are used to improve the accuracy of classification methods. For example, nearest-neighbor classifiers report the class label of the closest set of training data points to a given test instance. Clustering can help speed up this process by replacing the data points with centroids of fine-grained clusters belonging to a particular class. In addition, semisupervised methods can also be used to perform categorization in many domains such as text. The bibliographic notes contain pointers to these methods.

7.8.1.4 Dimensionality Reduction

Clustering methods, such as nonnegative matrix factorization, are related to the problem of dimensionality reduction. In fact, the *dual* output of this algorithm is a set of concepts, together with a set of clusters. Another related approach is probabilistic latent semantic indexing, which is discussed in Chap. 13 on mining text data. These methods show the intimate relationship between clustering and dimensionality reduction and that common solutions can be exploited by both problems.

7.8.1.5 Similarity Search and Indexing

A hierarchical clustering such as CF-Tree can sometimes be used as an index, at least from a heuristic perspective. For any given target record, only the branches of the tree that are closest to the relevant clusters are searched, and the most relevant data points are returned. This can be useful in many scenarios where it is not practical to build exact indexes with guaranteed accuracy.

7.8.2 Customer Segmentation and Collaborative Filtering

In customer segmentation applications, similar customers are grouped together on the basis of the similarity of their profiles or other actions at a given site. Such segmentation methods are very useful in cases where the data analysis is naturally focused on similar segments of the data. A specific example is the case of *collaborative filtering* applications in which ratings are provided by different customers based on their items of interest. Similar customers are grouped together, and recommendations are made to the customers in a cluster on the basis of the distribution of ratings in a particular group.

7.8.3 Text Applications

Many Web portals need to organize the material at their Web sites on the basis of similarity in content. Text clustering methods can be useful for organization and browsing of text documents. Hierarchical clustering methods can be used to organize the documents in an exploration-friendly tree structure. Many hierarchical directories in Web sites are constructed with a combination of user labeling and semisupervised clustering methods. The semantic insights provided by hierarchical cluster organizations are very useful in many applications.

7.8.4 Multimedia Applications

With the increasing proliferation of electronic forms of multimedia data, such as images, photos, and music, numerous methods have been designed in the literature for finding clusters in such scenarios. Clusters of such multimedia data also provide the user the ability to search for relevant objects in social media Web sites containing this kind of data. This is because heuristic indexes can be constructed with the use of clustering methods. Such indexes are useful for effective retrieval.

7.8.5 Temporal and Sequence Applications

Many forms of temporal data, such as time-series data, and Web logs can be clustered for effective analysis. For example, clusters of sequences in a Web log provide insights

about the normal patterns of users. This can be used to reorganize the site, or optimize its structure. In some cases, such information about normal patterns can be used to discover anomalies that do not conform to the normal patterns of interaction. A related domain is that of biological sequence data where clusters of sequences are related to their underlying biological properties.

7.8.6 Social Network Analysis

Clustering methods are useful for finding related communities of users in social-networking Web sites. This problem is known as *community detection*. Community detection has a wide variety of other applications in network science, such as anomaly detection, classification, influence analysis, and link prediction. These applications are discussed in detail in Chap. 19 on social network analysis.

7.9 Summary

This chapter discusses a number of advanced scenarios for cluster analysis. These scenarios include the clustering of advanced data types such as categorical data, large-scale data, and high-dimensional data. Many traditional clustering algorithms can be modified to work with categorical data by making changes to specific criteria, such as the similarity function or mixture model. Scalable algorithms require a change in algorithm design to reduce the number of passes over the data. High-dimensional data is the most difficult case because of the presence of many irrelevant features in the underlying data.

Because clustering algorithms yield many alternative solutions, supervision can help guide the cluster discovery process. This supervision can either be in the form of background knowledge or user interaction. In some cases, the alternative clusterings can be combined to create a *consensus* clustering that is more robust than the solution obtained from a single model.

7.10 Bibliographic Notes

The problem of clustering categorical data is closely related to that of finding suitable similarity measures [104, 182], because many clustering algorithms use similarity measures as a subroutine. The k-modes and a fuzzy version of the algorithm may be found in [135, 278]. Popular clustering algorithms include *ROCK* [238], *CACTUS* [220], *LIMBO* [75], and *STIRR* [229]. The three scalable clustering algorithms discussed in this book are *CLARANS* [407], *BIRCH* [549], and *CURE* [239]. The high-dimensional clustering algorithms discussed in this chapter include *CLIQUE* [58], *PROCLUS* [19], and *ORCLUS* [22]. Detailed surveys on many different types of categorical, scalable, and high-dimensional clustering algorithms may be found in [32].

Methods for semisupervised clustering with the use of seeding, constraints, metric learning, probabilistic learning, and graph-based learning are discussed in [80, 81, 94, 329]. The *IPCLUS* method presented in this chapter was first presented in [43]. Two other tools that are able to discover clusters by visualizing lower dimensional subspaces include *HD-Eye* [268] and *RNavGraph* [502]. The cluster ensemble framework was first proposed in [479]. The hypergraph partitioning algorithm *HMETIS*, which is used in ensemble clustering, was proposed in [302]. Subsequently, the utility of the method has also been demonstrated for high-dimensional data [205].

7.11 Exercises

1. Implement the k-modes algorithm. Download the *KDD CUP 1999 Network Intrusion Data Set* [213] from the *UCI Machine Learning Repository*, and apply the algorithm to the categorical attributes of the data set. Compute the cluster purity with respect to class labels.

2. Repeat the previous exercise with an implementation of the *ROCK* algorithm.

3. What changes would be required to the *BIRCH* algorithm to implement it with the use of the Mahalanobis distance, to compute distances between data points and centroids? The diameter of a cluster is computed as its RMS Mahalanobis radius.

4. Discuss the connection between high-dimensional clustering algorithms, such as *PROCLUS* and *ORCLUS*, and wrapper models for feature selection.

5. Show how to create an implementation of the cluster feature vector that allows the incremental computation of the covariance matrix of the cluster. Use this to create an incremental and scalable version of the Mahalanobis k-means algorithm.

6. Implement the k-means algorithm, with an option of selecting any of the points from the original data as seeds. Apply the approach to the quantitative attributes of the data set in Exercise 1, and select one data point from each class as a seed. Compute the cluster purity with respect to an implementation that uses random seeds.

7. Describe an automated way to determine whether a set of "must-link" and "cannot-link" constraints are consistent.